statics: a C++ implementation of functional iteration synthesis

STEPHEN F. LILLY

Art Institute of Washington, Arlington, VA 22209, USA
E-mail: sflilly@yahoo.com

***statics* is a non-standard sound synthesis program that uses functional iteration to both generate sound events and organise them temporally. This paper gives a brief history on the use of functional iteration in music composition and then describes how it is implemented in *statics*. Also discussed is the conception of program design as compositional process. Since *statics* was designed by a composer with musical goals in mind, the program is itself is a collection of compositional decisions. Furthermore, the unique timbres and structures created by *statics* are a direct result of these decisions. Accordingly, I discuss my compositional process in two stages: those choices that are embedded into the program (general decisions) and those choices that occurred post-design (specific decisions). Using as examples the three individual compositions created with *statics* (*congruent*, *convergence* and *cyclian*), I delineate the effects of both types of decision on the compositional process and then describe how these three compositions embody different characteristics of my implementation of functional iteration.**

1. INTRODUCTION

> No longer a language sign or symbol, the acoustical order itself becomes the subject and content of communication, and thus liberated from inaudible purposes, it becomes music, and, for that matter, important, significant music – or, as I like to call it, 'new music'. (Brün 2004)

In computer music, the digital sample is the means by which musical sounds are transmitted and stored. Similar to Herbert Brün's example of transforming a meaning-laden acoustic phenomenon into an abstract musical expression, the program *statics* creates musical events by working abstractly with the digital sample. Rather than using a known acoustic model, as is the basis for spectral and physical modelling, *statics* generates and organises individual samples algorithmically through the iteration of a nonlinear map, i.e. functional iteration. This means that the traditional musical notions of pitch, rhythm and structure do not function as the building blocks of musical composition; instead, they are emergent characteristics (Di Scipio 1995). In other words, composition is not the manipulation of pitches in time within a predetermined formal scheme. Musical sound and structure are instead the direct results of the concatenation of samples as determined by functional iteration, and designing how functional iteration is implemented and then later selecting musically viable renderings from all the possible outputs are what constitute composition. Furthermore, since functional iteration is responsible not only for the creation of sonic events, which range from pitched material to percussive impulses, but also for where these events are placed temporally, the composition of timbre is linked to the composition of form. In the words of Walter Branchi, 'the sound material ... is therefore both structured and structuring' (Branchi 1983).

2. A BRIEF HISTORY OF FUNCTIONAL ITERATION

Functional iteration has been useful in modelling the behaviour of dynamic systems (Gleick 1987 provides an informative and non-technical description of the history of nonlinear dynamics and the science of 'chaos'). In short, the future state of the system is shown to be dependent upon its current state: $N_{t+1}=f(N_t)$. This formula is known as a finite-difference equation (for more information, see Kaplan and Glass 1995). Functional iteration is the process by which one recursively applies the function to its output. If the function is nonlinear, this process results in complex behaviour dependent on the initial conditions and any functional constant. Consider, for example, the equation used to generate the Mandelbrot set: $z_{t+1}=z^2+c$ (z denotes that this map is applied to the complex number plane).[1] The results of iterating this equation change depending on the initial value of z (z_0) and the constant (c) (Mandelbrot 1983; Kaplan and Glass 1995). The familiar cardioid graph associated with the Mandelbrot set is generated by studying the asymptotic dynamics of the aforementioned map when $z_0=0$, i.e. the behaviour

[1]Imaginary numbers stem from the premise of $i^2=-1$, where $\sqrt{-1}$, or i, exists outside of the real numbers. Complex numbers consist of both real and imaginary numbers, and every complex number is notated as $x+yi$, where x is the real part and y, as a coefficient of i, is the complex part (in fact, a real number is simply a complex number in which $y=0$). The complex number plane is a graphic representation of the complex numbers where x and y are plotted as Cartesian coordinates.

Organised Sound **13**(1): 61–76 © 2008 Cambridge University Press. Printed in the United Kingdom. doi: 10.1017/S1355771808000083

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

of the output (z_{t+1}) as the number of iterations approaches infinity. Those values of c inside the cardioid are bounded, which means the results of successive iteration never escape to infinity. In fact, successive outputs either tend toward a fixed point, where eventually $z_{t+1}=z_t$; a limit cycle, where the output is a periodic sequence of values; or chaos, where the output is an aperiodic sequence. Many graphic representations of the Mandelbrot set also include a swirling of colours that borders the cardioid (see Peitgen and Richter 1986). This is linked to the behaviour of the unbounded values of c; the constants are coloured according to the number of iterations before the output of the map escapes some predetermined boundary (Kaplan and Glass 1995).

The artistic potential of functional iteration was first realised in the visual representations of fractals (Mandelbrot 1983; Peitgen and Richter 1986). Initial attempts to use functional iteration in music applied nonlinear maps to traditional parameters such as pitch, rhythm, dynamics and textural density (see Pressing 1988). For example, composer Roger Reynolds used the bifurcation diagram commonly associated with nonlinear dynamics to determine sectional durations for his chamber work *Dyonisus* and similarly used the Hénon map in the composition of the electro-acoustic work *Odyssey* (Reynolds 2002). Later, functional iteration was used to control synthesis parameters, e.g. grain size, transposition, density, etc., for granular synthesis (Di Scipio 1990; Truax 1990). Agostino Di Scipio was the first to describe a direct synthesis implementation, which he termed functional iteration synthesis (FIS) (Di Scipio and Prignano 1996; Di Scipio 2001). Di Scipio's approach explores a function over a range of constants and initial values at a given iteration: 'While not waveforms, these graphs do represent audio signals, although in a very particular way: indeed, the operation of FIS is the same as scanning such uneven surfaces, i.e. drawing a path or orbit across them' (Di Scipio 2002). In scanning these graphs, the value of the function at a given point (constant and initial value) represents the amplitude of a single digital sample.

Using the data derived from a nonlinear map to generate pitches, rhythms, or sample amplitudes is not problematic in and of itself, but controlling two or more parameters simultaneously in a manner that produces musically useful results poses a dilemma that many researchers have explored. Employing a different function for each parameter is the simplest solution, but then the evolution of each parameter is completely independent. Another method is to use a network of maps so that each parameter can be linked to the function which produces the best results for that parameter, but by allowing the maps to exchange data or in some way influence the behaviour of one another, this method allows for some interaction between parameters (Battey 2004). Alternatively, multi-dimensional maps can be used to control several parameters simultaneously, but as one researcher noted, 'although musical control variables are better linked … these links are not necessarily musically well-founded' (Pressing 1988). In functional iteration synthesis, this problem translates into how one should 'scan' the surface of a given map. This is a twofold problem: the manner in which one scans the map will determine the sequence of data, and the amount of time allotted for this task will determine the precision of the map. For instance, one could traverse the map by means of a control function (linear, quadratic, etc.) or by utilising some real-time interface. Then, given a sampling rate of 44.1 kHz, the precision of a given segment would be 44,100 samples if duration is set at 1 second or 441,000 samples if the duration is set at 10 seconds. Agostino Di Scipio recognised the importance of this:

> Due to the nonlinear dynamics of the synthesis mechanism, a carefully designed synthesis control interface is needed, such that the user may handle the critical interconnection among the synthesis parameters in rather intuitive ways during real-time operations. (Di Scipio 2002)

3. THE *STATICS* IMPLEMENTATION

While Di Scipio's implementation is a practical solution for real time synthesis, it means that temporal organisation is controlled externally and independent of the generation of the samples themselves. The *statics* implementation of functional iteration synthesis uses the same two-dimensional nonlinear map to link sample generation with temporal organisation. In *statics*, the two dimensions are not arbitrarily assigned: one to amplitude and the other to time. Instead, *statics* borrows its logic from Fourier analysis. When analysing a complex wave, the Fourier transform produces a real value (x) and an imaginary value (y) in Cartesian coordinates for each component frequency, and these complex values translate to an amplitude (magnitude) and a phase (angle) in polar coordinates. I decided to use the complex number plane as my data set, so that all output values could be formulated in a similar manner to the Fourier method. Thus, the real and imaginary outputs of functional iteration translate to amplitude and phase, where phase is interpreted as a position in time. This seemed to me a logical solution to the above dilemma that does produce very interesting musical results, but unlike Di Scipio's implementation, it limits *statics* to being a rendered synthesis engine. Like the amplitudes of the individual samples, time is also determined nonlinearly, and thus, audio samples are produced in non-sequential order.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

3.1. The map

The nonlinear function around which *statics* is constructed is $z_{t+1} = (z_t + c)^2$. Although similar to the Mandelbrot map, it has some unique properties for certain values of z_0. In particular, when $z_0 = -1+i$, the values of c for which the map is bounded constitute a very small, concentrated region in the complex number plane. The size of this region is dependent on the maximum number of iterations one tests (see Table 1 and Figures 1 & 2) and the precision with which one steps through the possible values of c (see Figures 2 & 3).

For all of the bounded values of c with $z_0 = -1+i$, the attractor was a limit cycle. Most of the constants exhibited a limit cycle of period 20 when iterated, and only one of the constants I tested ($c = 0.426 - 0.3414i$) had a limit cycle of period 15.[2] Even more interesting was the fact that for all of the constants with a limit cycle of period 20, the cycle consisted of almost identical values (see Table 2), and the constants were aligned in five lines radiating out from the point $-0.274-0.156i$ (Figure 4).

These characteristics are reflected in the program's design. The *statics* synthesis engine consists of two hierarchically related processes: a sound event generator and a control structure generator. The relationship between the number of bounded values and the precision and number of iterations is the basis for sample creation and organisation (sound event generator), whereas the control structure is organised around the patterns of iteration, specifically the limit cycle and the transient region that precedes it. With regard to the map, the sound event generator explores variations in the bounded and unbounded values of c given $z_0 = -1+i$, and the control structure generator uses the bounded values of c when $z_0 = -1+i$ to explore the other values of z_0 for which these constants are still bounded.

Table 1. Number of constants for a given number of iterations at a precision of 0.0001.[3]

Iterations of iteration range	Number of constants
100	733
200	28
300	14
369 – 422	8
423 – 817	7
818 – 3,305	6
≥ 3,305	5

[2]In fact, iteration produced a limit cycle for these constants given any value of z_0 that did not cause the map to escape to infinity.

[3]The values in Table 1 were determined by using a magnitude $\left(\sqrt{(real)^2 + (imaginary)^2}\right)$ of 2 as the boundary between bounded and unbounded. Investigation revealed that the number of iterations increases only marginally if the boundary is set any higher, e.g. with a magnitude of 100 as the boundary, the same five constants are reached after 3,308 iterations.

3.2. Control structure generator

In *statics* the control structure is responsible for directing sound synthesis by passing parameter settings to the sound event generator. It also determines where in the final WAV files each sound event will be placed. The sound event generator constructs the initial sound event from seed values, one for each parameter setting, that are supplied by the user. The control structure generator then subjects these seed values to functional iteration so that the sound event generator is supplied with new parameter settings for every sound event.

The final output of *statics* is a monophonic WAV file for each channel of sound. To execute this, the control structure generator requires only a single constant. I decided that this constant should be bounded given $z_0 = -1+i$ because I wanted iteration to eventually settle into a limit cycle, at which point iteration of the control structure and likewise the generation of sound events would stop. I compiled a list of bounded constants, at various levels of precision, using a very simple program that tested numbers over a given range and discarded those whose magnitude exceeded 2 after a large number of iterations. While the equation is stable for some values of z_0 and c whose magnitude is greater than 2 (e.g. when $c = -2$ and $z_0 = 4$), this is not true when $z_0 = -1+i$. Thus, 2 was adopted as the upper range for checking if a value was bounded. Accordingly, it was also used to map amplitude values to 16-bit digital audio; 2 was set as the maximum positive displacement, and 0 was set as the maximum negative displacement (a majority of the magnitudes are between 0 and 1 and so have a negative displacement, but the chosen range of [0, 2] utilised the bit-depth resolution better than setting the mean at 0 or restricting all values to a positive displacement; see Figure 7). For each constant I used, I also generated a list of possible z_0's for which the map would be bounded. Each seed value for the sound event generator's parameter settings was selected from this list to ensure that successive iterations of the seed values would remain bounded, i.e. within a useable, scalable range.

Investigation revealed that the magnitude of the largest bounded z_0 is just under 1.64 for all constants. This value was used to scale the iteration results to values appropriate for each parameter setting. The concentration of values, however, tends to gravitate toward numbers in the limit cycle. This means that the median value generated by the control structure generator will be near the median value of the limit cycle; for a limit cycle of period 20, this is 0.469, which is only 28.6 per cent of the maximum magnitude (the median value for the limit cycle of period 15 is slightly lower at 0.453). This was taken into consideration when determining how the distribution of magnitude values would map onto each parameter setting, i.e. by using either a linear or nonlinear mapping (see Section 4).

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

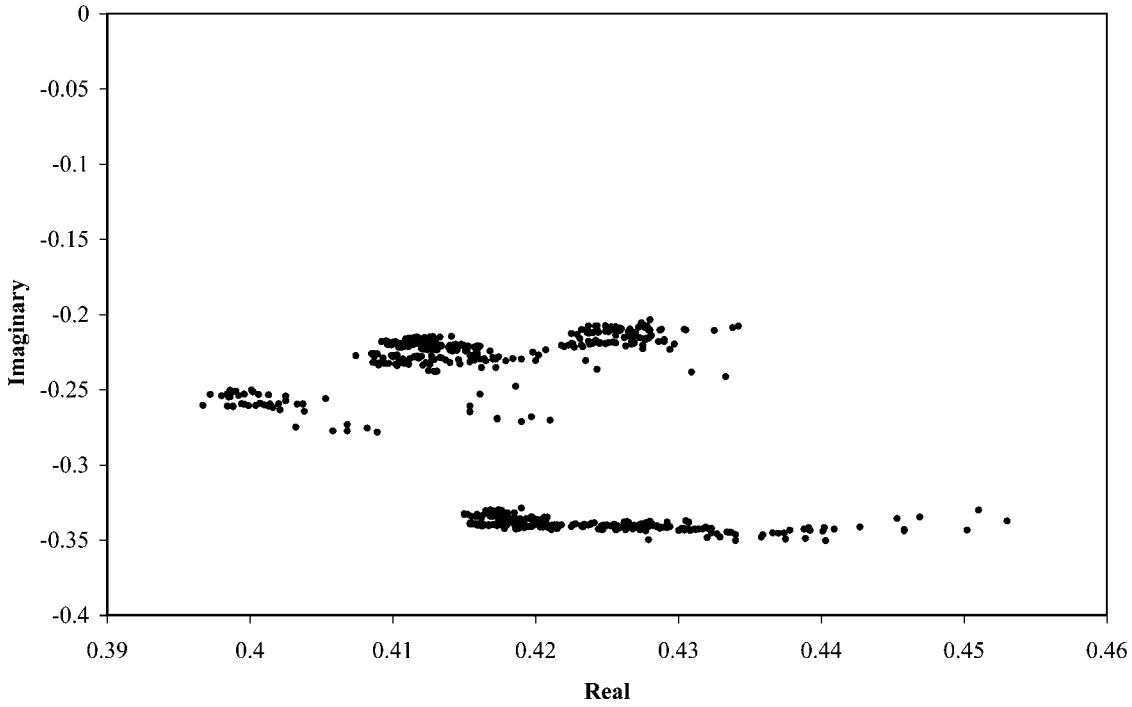


Figure 1. Bounded constants after 100 iterations with 0.0001 precision.

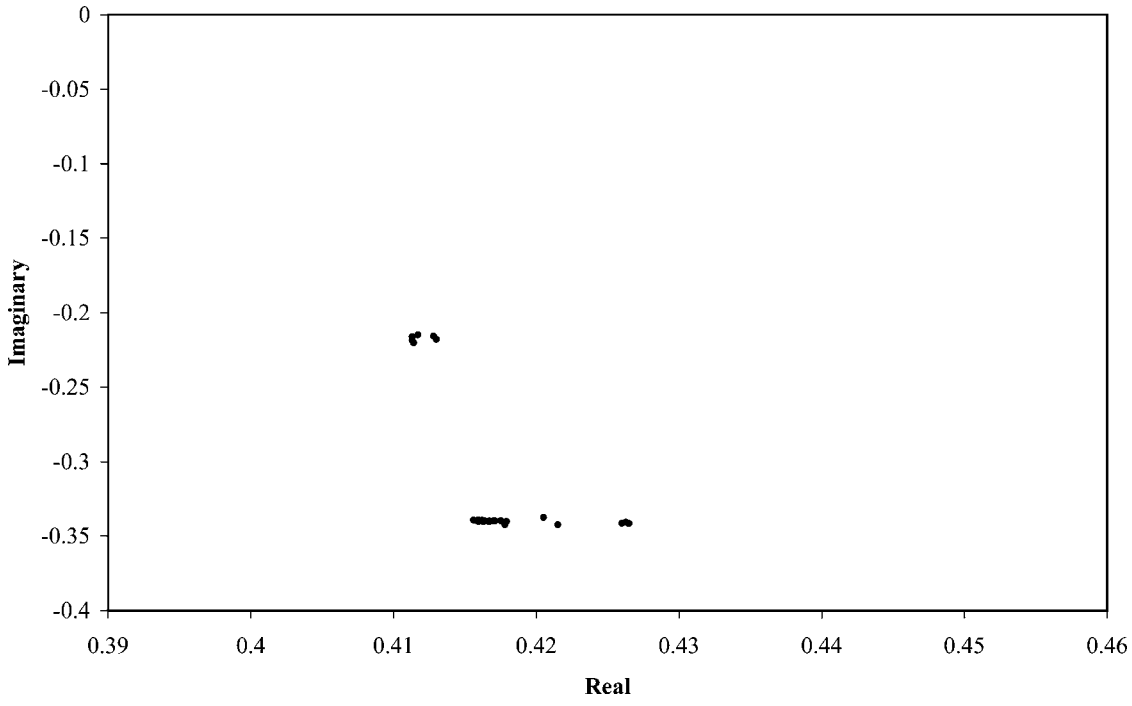


Figure 2. Bounded constants after 200 iterations with 0.0001 precision.

Additionally, this means that values greater than the largest magnitude in the cycle (0.928) only occur during the initial iterations (the transient phase) before the map settles into the cycle. This fact greatly influenced the choice of seed values for each parameter (see Section 4.2.1).

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

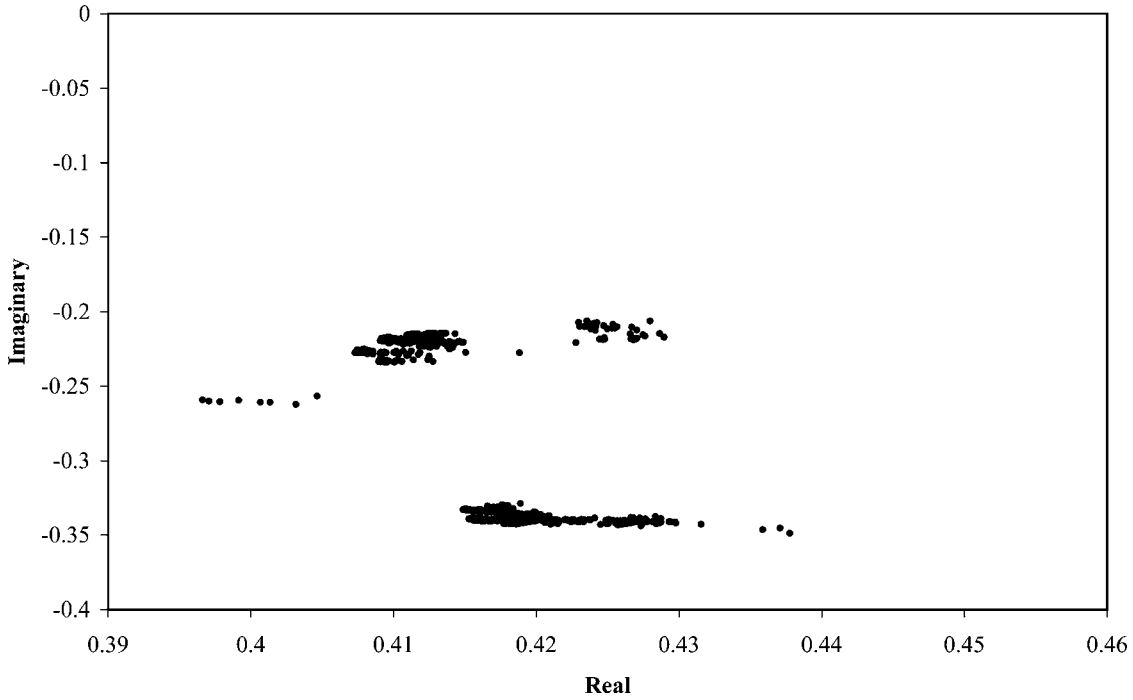


Figure 3. Bounded constants after 200 iterations with 0.00001 precision.

Table 2. 20-cycle comparison across six constants.

Index in cycle	Average real value	Standard deviation	Average imaginary value	Standard deviation
1	−0.00510	0.000465	−0.269	0.00102
2	−0.202	0.00195	−0.500	0.000915
3	−0.660	0.00246	−0.360	0.00273
4	−0.430	0.00450	0.341	0.00243
5	0.000215	0.000138	−0.0000213	0.0000522
6	0.0553	0.00244	−0.283	0.000456
7	−0.163	0.000919	−0.590	0.000426
8	−0.801	0.00158	−0.470	0.00123
9	−0.507	0.00263	0.623	0.00185
10	−0.0719	0.00129	−0.0517	0.00129
11	−0.0349	0.000606	−0.269	0.00161
12	−0.226	0.00157	−0.464	0.00106
13	−0.611	0.00177	−0.306	0.00241
14	−0.379	0.00421	0.251	0.00190
15	−0.00640	0.000581	−0.00659	0.000622
16	0.0477	0.000274	−0.284	0.000586
17	−0.174	0.000217	−0.578	0.000387
18	−0.785	0.000307	−0.445	0.000522
19	−0.480	0.00158	0.578	0.000600
20	−0.0528	0.000373	−0.0303	0.000683

Another factor in seed choice was the length and shape of the transient phase, which depends on the initial seed value. Figure 5 illustrates this with three different values of z_0.[4] For $z_0 = 0$, there is no transient phase, and immediately after the initial seed at iteration index 16, successive values are drawn from the limit cycle. $z_0 = -0.366 + 0.576i$ has a fairly long transient phase (the seed is at iteration index 9, and successive iterations do not fully line up with the limit cycle until

[4]The three iteration series in Figure 5 are plotted so that once the transient phase is complete, the limit cycles line up. Because the length of the transient phase depends on the value of z_0, these three series do not all start at iteration index 1. $z_0 = -0.735 + 1.449i$ starts at iteration index 1; $z_0 = -0.366 + 0.576i$ starts at iteration index 9; and $z_0 = 0$ starts at iteration index 16.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

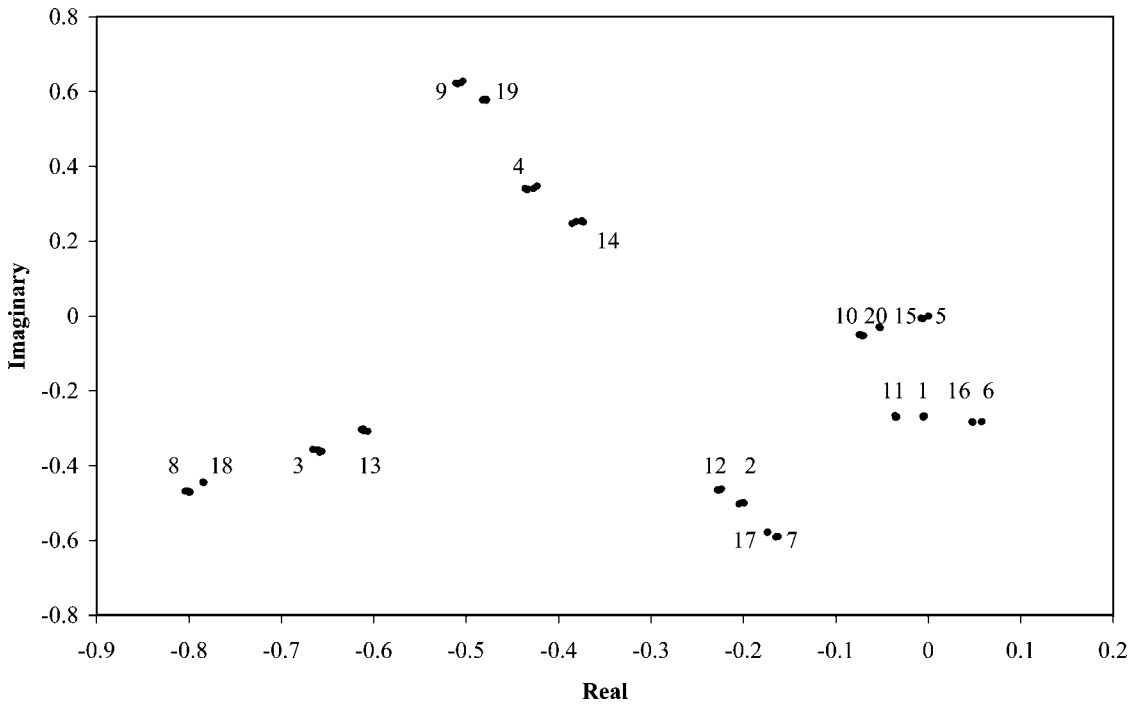


Figure 4. Graph of Table 2.

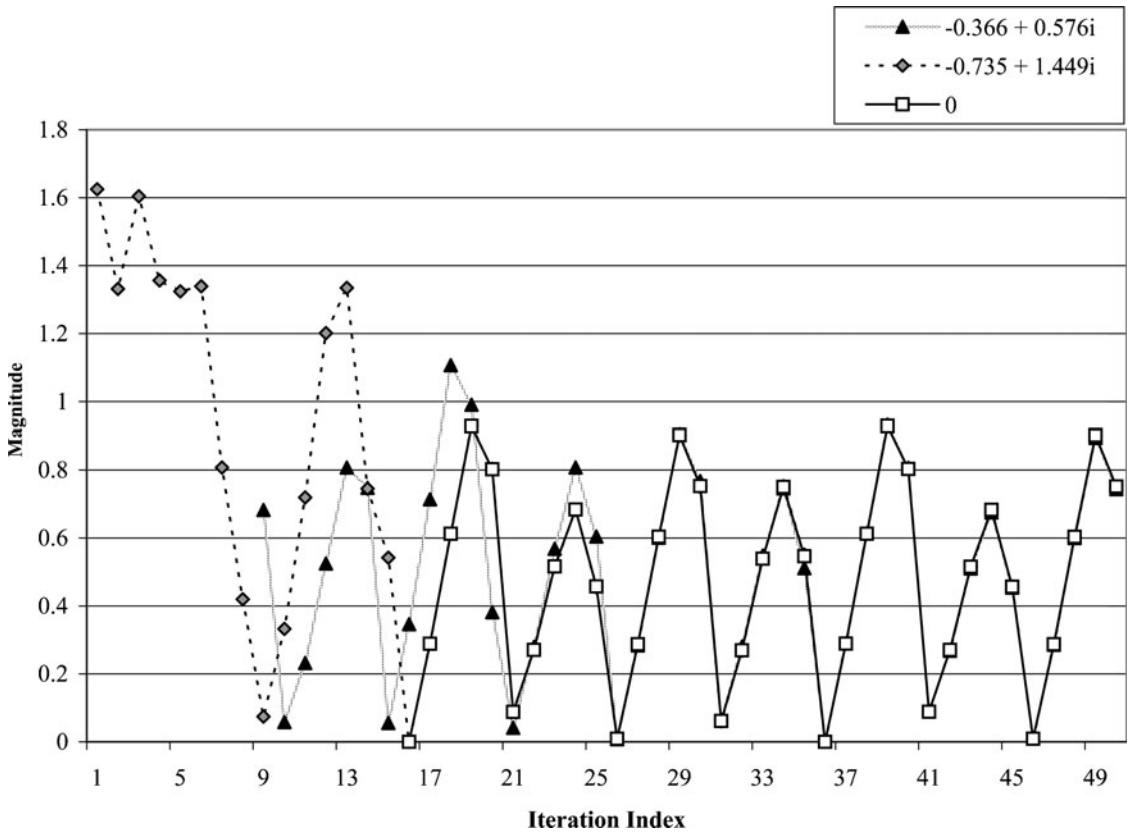


Figure 5. Overlapping $z(0)$'s with constant $= 0.416 - 0.3401i$.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

iteration index 36), but even when values do not exactly correspond with the limit cycle, they do not vary by much. Finally, $z_0 = -0.735 + 1.449i$ has the most dramatic transient phase that includes values near the maximum magnitude. This means that even if all of the parameter settings are based on the same map that eventually cycles through the same twenty or fifteen values, the limit cycles for all of the parameters will not necessarily be in sync because of the transient phases, which vary in length based on the seed value used for each parameter.

The control structure consists of eleven parameter settings. Two of these are non-continuous variables:

(1) *Organisation*. For the sound event generator, the value of z_0 is fixed (see Section 3.1). Therefore, all information needed to construct unique audio samples is derived from the constant c and successive values of z as the map is iterated. Constants, over an empirically derived range (see Parameter 7, precision), are tested by the sound event generator. When a constant passes the test, which is dependent on the audible iterations parameter, one or more z's (see Parameter 2, audible iterations) are written to a dynamic buffer used by the sound event generator to organise individual samples before they are written to the final WAV files. The organisation parameter is controlled by a Boolean variable that determines which value, either the angle of the constant (input) or the angle of the last iteration (output), signifies location within the dynamic buffer. In other words, if the organisation variable is false, samples written to the dynamic buffer are input-organised, i.e. the samples are placed into the dynamic buffer according to the initial value passed to the sound event generator's iteration process, the constant. If, on the other hand, the organisation variable is true, then the samples written to the dynamic buffer are output-organised, i.e. the samples are placed into the dynamic buffer according to the value returned from the iteration process, the angle of z after the last iteration.

(2) *Audible iterations*. This parameter also has a true/false distinction. True means the magnitude of all z's from the sound event generator's iteration process are written to the dynamic buffer, whereas false means only the magnitude of the final iteration is written to the dynamic buffer. The true and false distinctions also determine whether the sound event generator tests for bounded or unbounded constants. When the parameter is true, a constant must cause the output of the equation to escape, i.e. exceed a magnitude of 2, within the predetermined iteration test range for anything to be written to the dynamic buffer.[5] When this happens, first the magnitude of z_0 is written to the location designated by the organisation parameter, and then successive values of z are written in consecutive locations in the dynamic buffer up to the z immediately before the magnitude exceeds 2 (see Section 3.3 for more details on this process). On the other hand, when the audible iterations parameter is false, the constant must be bounded up to the number of iterations designated by the iterations parameter (see Parameter 8, iterations), and then only the magnitude of z after the last iteration is written to the dynamic buffer at the location determined by the organisation parameter.

The next eight parameter settings are continuous variables:

(3) *Start time*. The magnitude of this value is scaled over the final length of the output WAV files, which was arbitrarily set at 6 minutes and 17 seconds. The resulting value is the point at which the generated sound event will start. Unlike the sound event generator, start time is determined by magnitude rather than angle; this decision was made due to the larger number of magnitude values in the limit cycle (20 versus only 5 different angles).[6] In addition to final location of each sound event, start time is also responsible for stopping the program. As soon as a start time parameter setting is repeated, denoting the onset of the limit cycle, control structure iteration and sound event generation cease, and the output WAV files are written.

(4) *Stretch factor*. The magnitude of this value is scaled over the range [0.5s, 376.99s]. The resulting value is the maximum length of the sound event. Accordingly, the sound event generator creates an appropriately sized dynamic buffer. If this buffer is large (close to 376.99s), the sound event is 'stretched' over a larger number of samples than if the buffer is small (near 0.5s).

(5) *Location*. Rather than having every sound event start with the first index of the dynamic buffer, this parameter setting designates another index in the buffer as the first sample. When reading through the buffer, a wraparound protocol is used to ensure that all indices are used and that the index of the final sample always immediately precedes the

[5]As will be explained in Section 4.1, the audible iterations parameter is not strictly Boolean. In addition to determining whether there will be audible iterations, the magnitude of this parameter is also used, in conjunction with the iterations parameter, to construct a test range for iterations.

[6]Additionally, the cycle of five angles is not oriented around the origin, but around the point $-0.274-0.156i$. Thus, if the angles are measured from the origin, the limit cycle is obscured. Since the number of unique magnitude values in the limit cycle is identical whether measured from the point $-0.274-0.156i$ or the origin, magnitude was chosen as the more universal way to represent the limit cycle.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

index of the first sample. Additionally, the first sample has to be a value other than zero, which means that the magnitude of the location value is scaled over the number of non-zero values in the buffer. This guarantees that the start time will always be articulated.

(6) *Duration*. The magnitude of this value is scaled to be a percentage of the stretch factor, i.e. less than or equal to the size of the dynamic buffer. This value determines how much of the sound event will actually be written to the output WAV files.

(7) *Precision*. The magnitude of this value is scaled over the range [0.00001, 0.001]. The resulting number is the increment with which the sound event generator steps through the range of possible constants (this range was empirically determined: real values over [0.3961, 0.42968] and imaginary values over [−0.35584, −0.20202]). This parameter setting, along with the stretch factor, controls the overall precision of the sound event. A high stretch factor means a finer resolution of the possible angles, which, as previously mentioned, the sound event generator uses to organise samples in the dynamic buffer (this also means less overwriting – see Section 3.3). Finer precision parameter settings correspond to better resolution of the complex number plane, providing for a greater number of bounded constants or unbounded constants (depending on the setting of the organisation parameter).

(8) *Iterations*. The magnitude of this value is scaled over the range [2, 1000]. The interpretation of this parameter depends on the audible iterations parameter. When the setting of the audible iterations parameter is true, the iterations value is the lower boundary of the aforementioned iteration test range, and the upper boundary of this range is derived from both the iterations value and the audible iterations value (see Section 4.1). For anything to be written to the dynamic buffer, a constant must be bounded up to the lower boundary and then escape before the upper boundary. When the setting of the audible iterations parameter is false, the iterations value determines the number of iterations before the magnitude of the output is written to the dynamic buffer. If the output escapes before completing the required number of iterations, nothing is written to the dynamic buffer.

(9) *Dynamics*. The magnitude of this value is scaled to be a percentage by which every sample in the dynamic buffer is multiplied. In essence, this value sets the overall dynamic level for each sound event.

(10) *Spatialisation*. The magnitude of this value is scaled to be a percentage of the duration. The resulting value determines how often the channel changes in course of reading through the dynamic buffer, and it works in conjunction with the spatialisation seed parameter (see below). For instance, if the percentage is 25%, the dynamic buffer is divided into quarters, and at the beginning of each quarter the program consults the spatialisation seed to determine which channel will receive the next set of samples.

(11) *Spatialisation seed*. This parameter setting is not continuous, nor is it updated (i.e. iterated) only once per sound event like all of the other parameter settings. The magnitude of this value is arbitrarily scaled over the total number of samples in each WAV file. Then the modulo operation determines the channel number,[7] e.g. if the program is set up for quadraphonic output, the operation would be (*scaled spatialisation seed*) % 4. This produces a number in the range [0, 3], and each of these numbers corresponds to an output channel. This parameter setting is iterated every time the spatialisation value requires a new value, so in the example discussed above regarding spatialisation, the spatialisation seed would be iterated four times.

3.3. Sound event generator

As previously stated, the sound event generator has one set value for z_0 $(-1 + i)$, and the output is determined by varying the value of c. The sound event generator creates a dynamic buffer whose size is determined by the stretch factor parameter. The audio samples are written to a location in this buffer according to the angle of either the constant or the final output value, as determined by the organisation parameter (Figures 6–8). Using radians, this angle is in the range $[-\pi, \pi)$. This range is mapped onto the dynamic buffer so that reading through the buffer from beginning to end is like circularly sweeping through the magnitudes plotted on the complex number plane (like the updating of a radar screen) (Figures 6 & 7).

The amplitude of each audio sample is derived from the magnitude of the final output value. This is important because the range of available constants is fixed. Thus, if the organisation parameter is set to input, the available positions in the dynamic buffer are limited, but since magnitude is derived from the final output value that changes depending on the iterations parameter setting, the sample amplitudes will always be different and thus so will the resultant timbre (compare Figures 9 & 10). The rhythm of input-organised sound events, on the other hand, is more or less fixed, but the stretch factor does create diminutions and augmentations of this rhythm (Figure 9 is an augmentation of Figure 8), and location creates permutations.

[7]The modulo operation (%) returns the remainder after division, e.g. 8 % 2 = 0 but 9 % 2 = 1.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

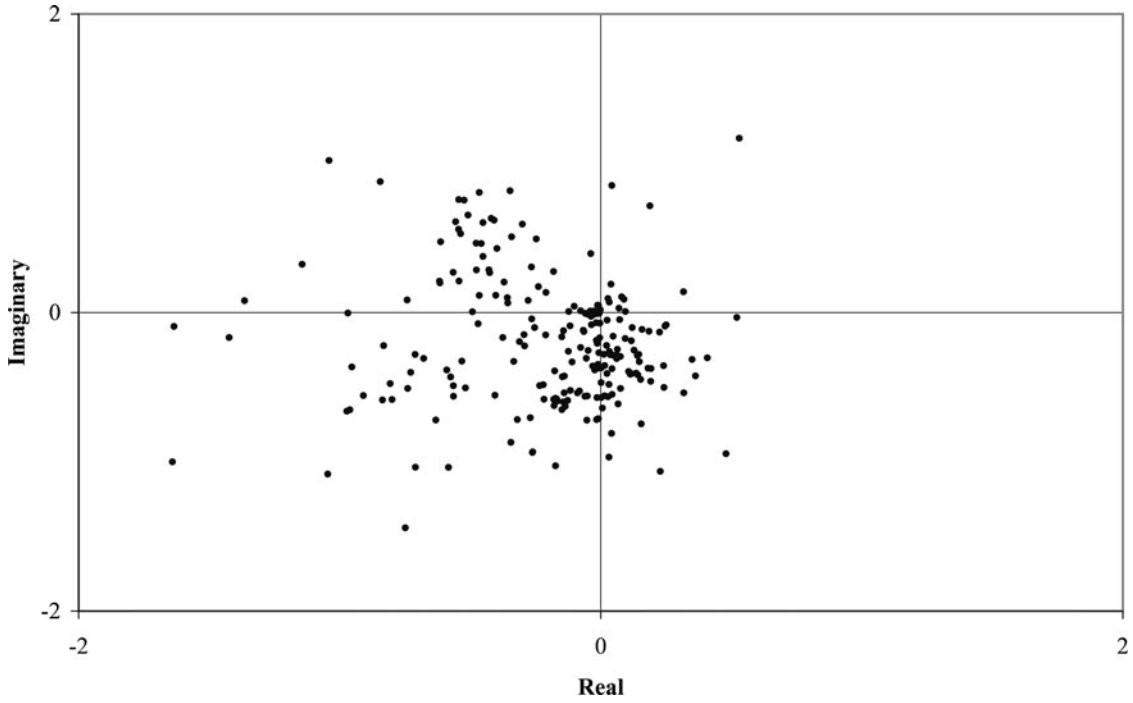


Figure 6. Output after 125 iterations at 0.0001 precision.

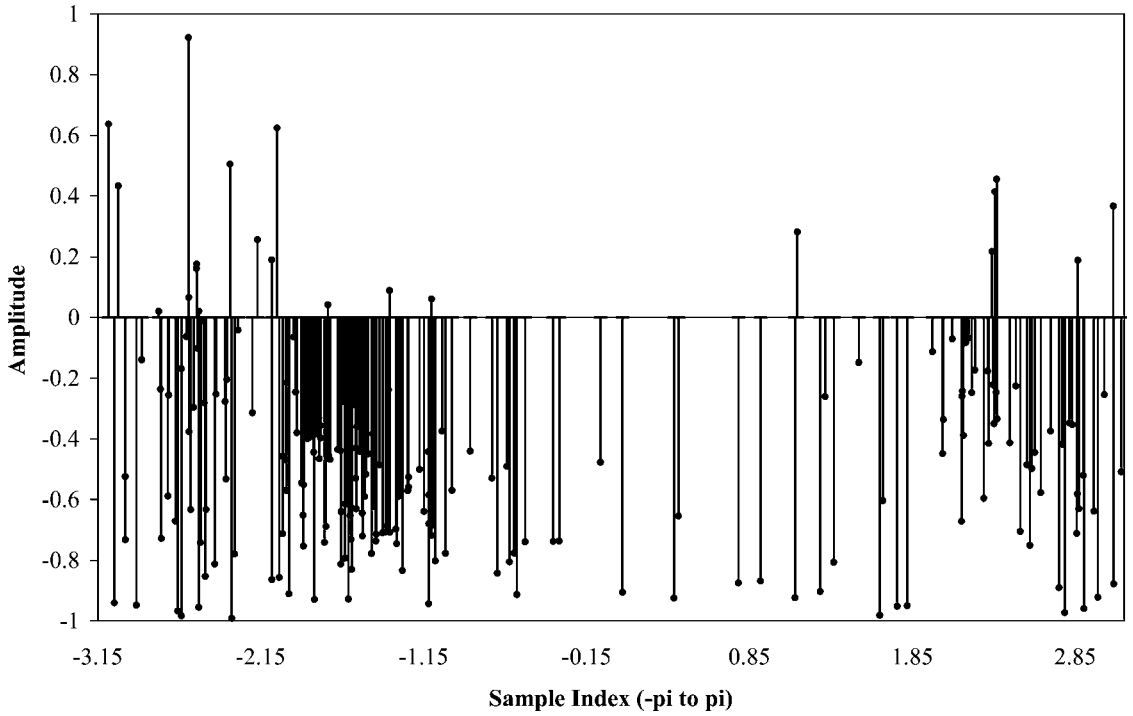


Figure 7. Soundfile for 0.0001 precision at 125 iterations.

Output-organisation is much more varied, being dependent on the precision (compare Figures 6 & 7 with Figures 11 & 12) and iterations parameter settings (compare Figures 6 & 7 with Figures 13 & 14). Comparisons of sound events like those presented in Figures 6 & 7 and 11–14 illustrate the fact that the

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

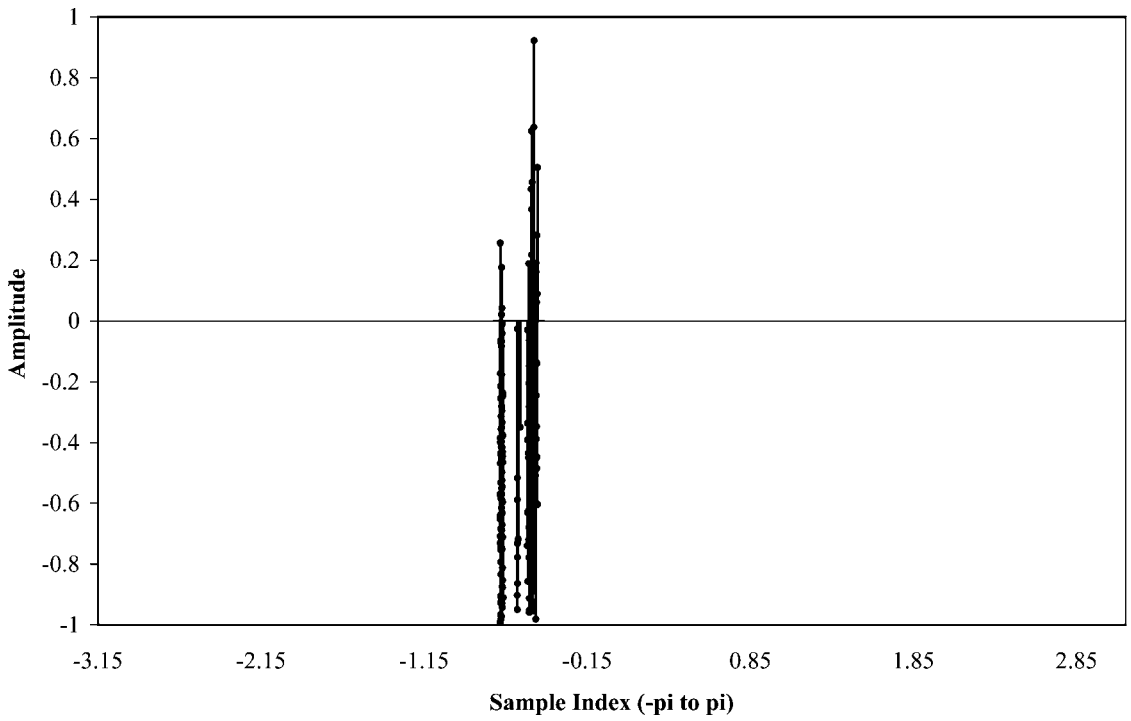


Figure 8. Soundfile of input-organisation for 0.0001 precision at 125 iterations.

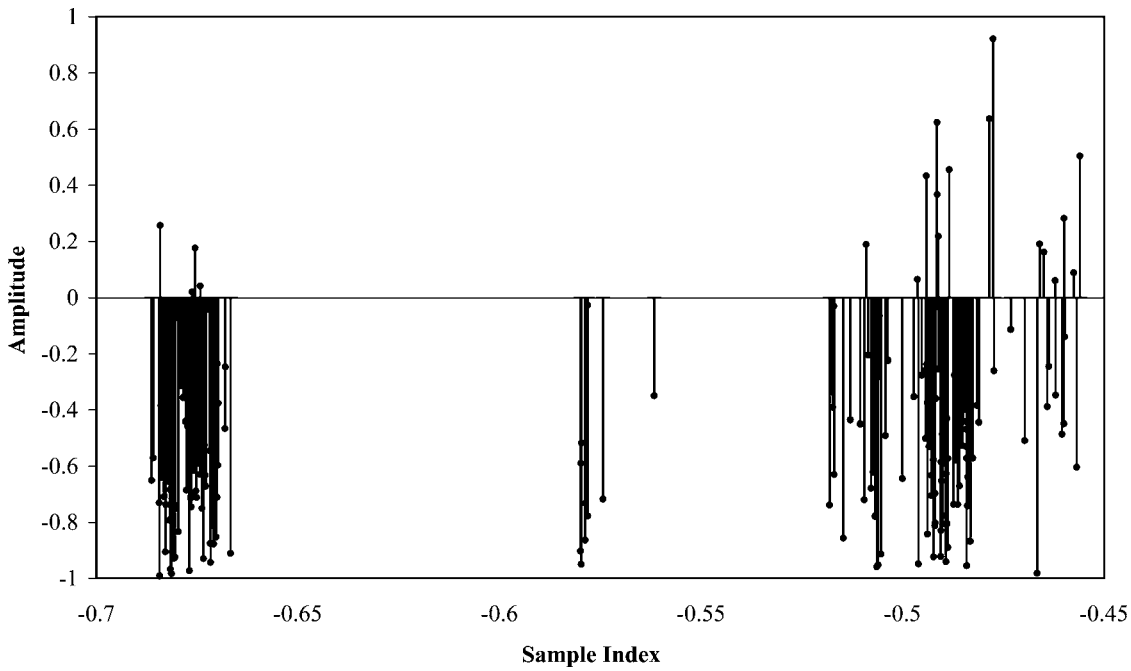


Figure 9. Soundfile of input-organisation for 0.0001 precision at 125 iterations: zoom-in.

number of bounded constants decreases as the precision parameter approaches 0.001 and as iterations increase, e.g. at a precision of 0.0001, increasing the number of iterations will eventually leave only five values (see Table 1). Conversely, if the precision and iterations parameters are both set to maximise the number of

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

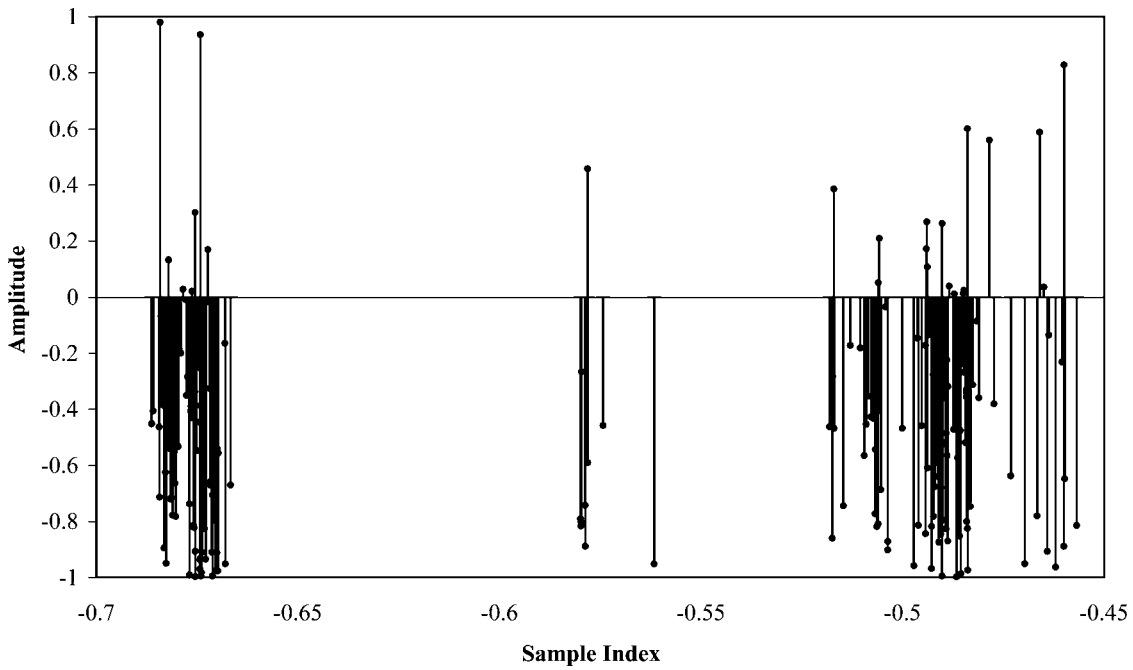


Figure 10. Soundfile of input-organisation for 0.0001 precision at 126 iterations: zoom-in.

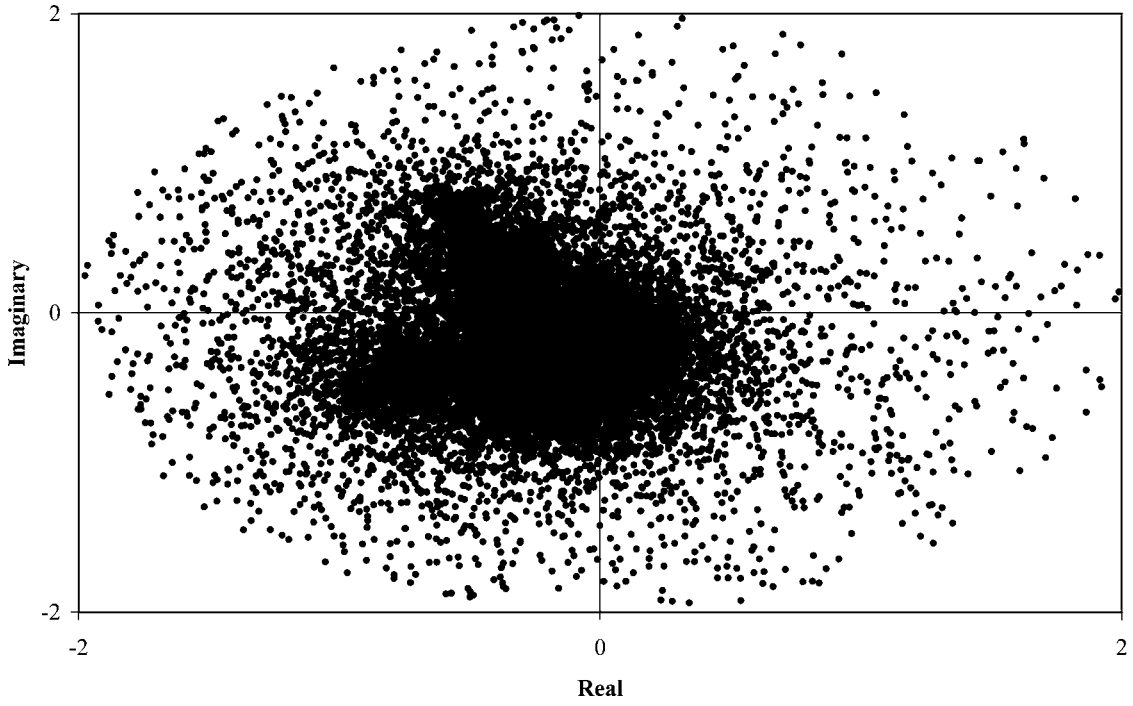


Figure 11. Output after 125 iterations at 0.00001 precision.

bounded constants, there is a higher risk of two output values being written to the same location in the dynamic buffer. *statics* is constructed in such a way that if a new audio sample is to be written at the same location in the dynamic buffer (this does not apply to the output WAV files, where the overlapping of sound events is solved by

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

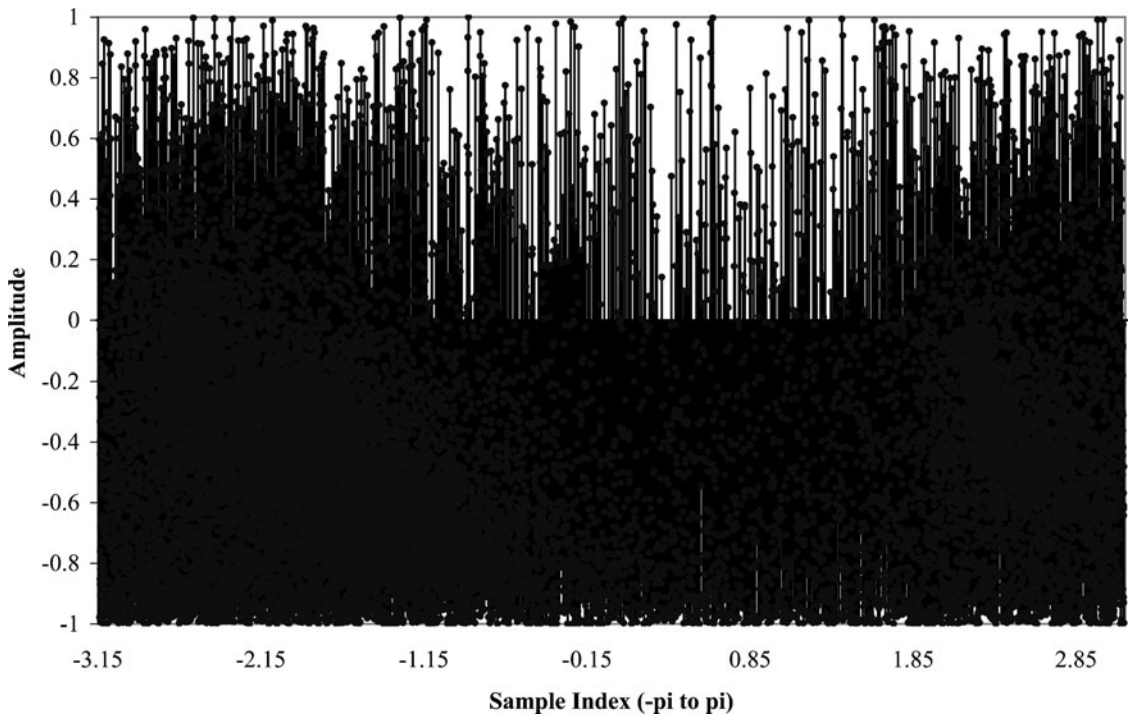


Figure 12. Soundfile for precision 0.00001 at 125 iterations.

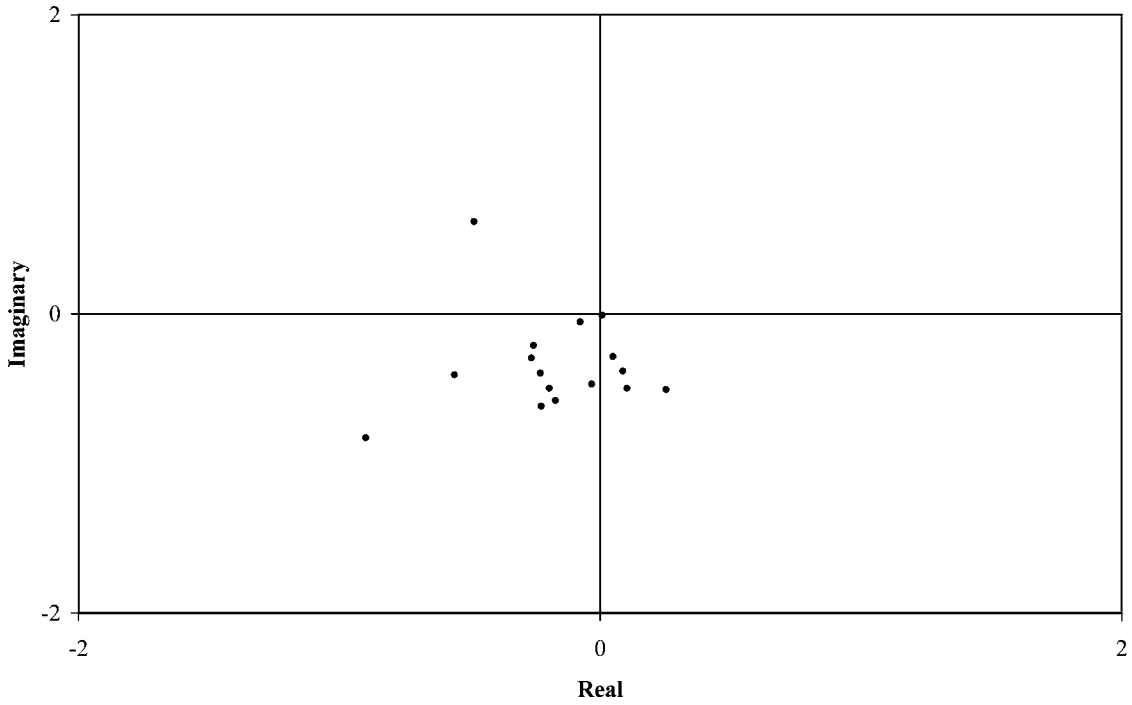


Figure 13. Output after 200 iterations at 0.0001 precision.

addition), it will overwrite whatever value was there previously. This destructive method of creating sound events is more pronounced when the stretch factor parameter is low, since $[-\pi, \pi)$ is mapped onto fewer samples.

The situation is substantially different when the iterations are audible. Even though this method explores the unbounded constants, for the majority of the iterations the map is locally stable and approximates the limit cycles of the bounded constants before finally

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

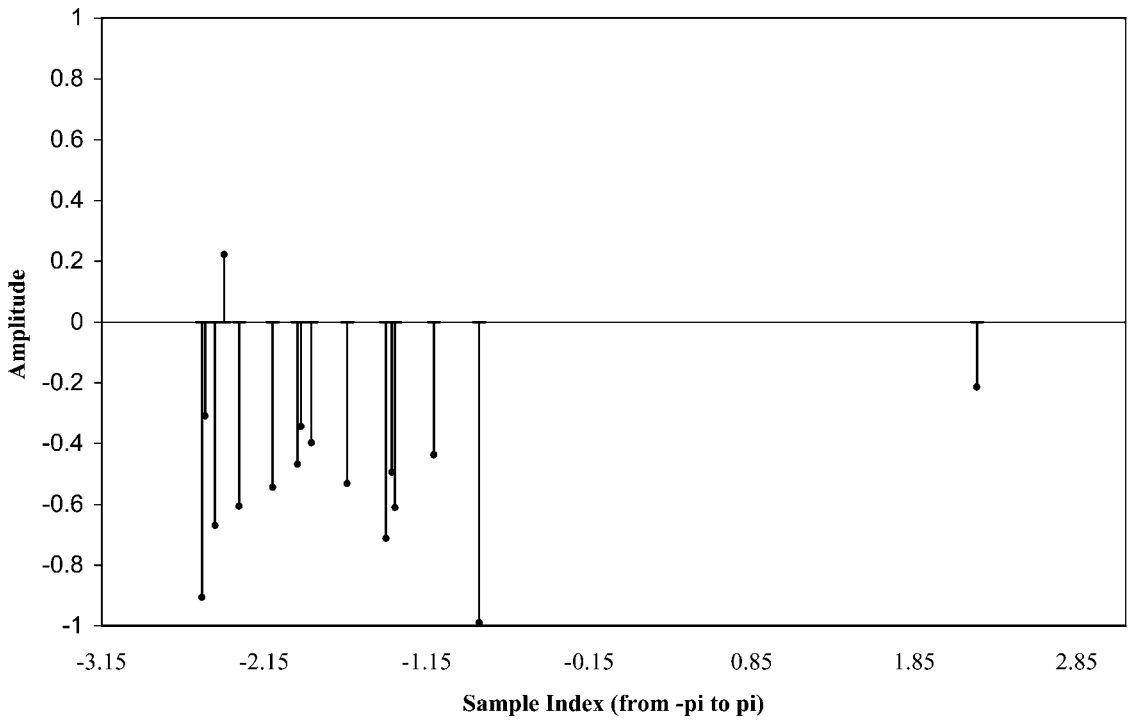


Figure 14. Soundfile for 0.0001 precision at 200 iterations.

escaping to infinity. Thus, one hears a periodic waveform that is shaped by one of the two limit cycles. Assuming a sampling rate of 44,100 Hz, one hears the frequency 2,940 Hz (a slightly flat G-flat7) for constants with a local cycle of period 15 and the frequency 2,205 Hz (a slightly flat D-flat7) for constants with a local cycle of period 20. In the construction of audible iteration sound events, successive iterations are written into the dynamic buffer as consecutive samples. The length of a single tone is dependent upon when the output magnitude exceeds 2. A single sound event consist of many such tones, since the entire constant range is inspected for values that are unbounded between the lower iteration limit set by the iterations parameter and the upper limit derived from the magnitude of the audible iterations parameter (see Section 4). Like the inaudible iterations setting, the starting location of each tone is either determined by the angle of the constant or the angle of the last output value before the magnitude exceeds 2.

4. COMPOSITION

In designing *statics* and then subsequently crafting three independent compositions from the resulting WAV files, the creative process was applied through either general or specific decisions. General decisions altered the overall output quality of *statics*. By evaluating the relationship between parameter settings and the resulting sound events, I was able to modify the mapping of parameter magnitudes in order to focus on and explore sounds I found intriguing either sonically or formally. Specific decisions were made once these mappings were fixed. At that point, I experimented with various seed values for each parameter to find settings that resulted in, by means of the iterative process, interesting juxtapositions and superimpositions of sound events; these specific decisions are what eventually led to three very different compositions.

4.1. General decisions

I have already discussed general decisions such as the empirically determined range limits (both those used as boundaries for testing and those used as scaling factors for various parameters – see Section 3.2). In other general decision cases, however, the alterations were more severe.

Each of the Boolean variables uses a different criterion to distinguish true from false. The organisation parameter uses the median for limit cycle of period 15, which intentionally alters the distribution to favour organisation by output; since the majority of the constants have a limit cycle of period 20, about 67.8% of all sound events are organised by the angle of the final z. The process used to determine the audible iterations parameter is more complicated. The magnitude of this parameter is scaled over the range [0, 1) and then multiplied by 3. This is then the power to which the iterations parameter, after scaling, is raised. If the

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

resulting number is larger than the initial number of iterations, which is true 46.7% of the time, then iterations between the two limits are audible if and only if the equation is unbounded for the given constant.[8]

Many of the continuous variables also use a weighting scheme that favours one type of output over another. Since duration is dependent on the size of the dynamic buffer, two different mappings were used. For buffer lengths up to and including 5s, a linear mapping is employed, but for longer buffers an exponential mapping (the duration value, which is a percentage of the dynamic buffer, is squared, e.g. 15% would become $0.15^2 = 2.3$ %) ensures that short durations will predominate. This decision was made to limit the number of long sound events, which tended to make the texture of the final output WAV files too monotonous. Likewise, an exponential mapping is used for spatialisation (the spatialisation value, which is a percentage of the duration value, is raised to a power of 4, e.g. 15% would become $0.15^4 = 0.051$%). Once again, this mapping favours smaller values so that more sound events would be distributed amongst more than one channel, and only a few sound events would be restricted to a single channel. Dynamics were also scaled exponentially to ensure that loud sounds were rarer and thus more musically effective.

As with the duration parameter, the iterations and precision parameters use a two-part distribution formula. For iterations, a process was employed to favour lower numbers of iterations, which tended to create sound events containing denser, more complex timbres, since there were more constants in the test range whose outputs were still within the magnitude boundary. The first step mapped the magnitude of the parameter over the range [2, 30]. If the resulting value was 11 (a magnitude of 0.527) or higher, it was remapped onto the range [−457, 1000]. This meant that sound events were split evenly between the ranges [2, 10] and [11, 1000]. For precision, the decision was made to make this parameter dependent on iterations; a higher iteration setting created richer sound events with a finer precision setting, i.e. closer to 0.00001. Thus, if the number of iterations exceeded 50 (only 39% of the sound events), then the precision magnitude was mapped onto the range [0.00001, 0.0001]. The majority of sound events, with 49 or fewer iterations, used the full precision range [0.00001, 0.001].

4.2. Specific decisions

4.2.1. Seed values

As previously mentioned, specific composition decisions were based on seed selection for individual parameters. Based on a table that listed bounded values of z_0 and their magnitudes for a given c, I could carefully design a single sound event, upon which all the other sound events would be shaped by successive iterations of the parameter seeds. In most cases, this 'sound event seed' was placed at or near the end of the WAV files to ensure that there would be a longer, more angular transient phase for start times. In other words, more sounds events were placed outside of the limit cycle (since the largest magnitude in either cycle was 0.928, all limit cycle start times were in the first 56.6% of the piece), and more sound events were produced since it took longer for the start time to conform to the limit cycle and thus terminate the program.

4.2.2. Post-statics composition

After listening to several WAV file outputs produced by *statics*, I decided that a single execution of the program yielded too thin a sonic texture. Accordingly, I decided that for each composition, I would layer multiple outputs,[9] not only to increase the density of sound events, but also to highlight certain emergent formal characteristics.

For the composition *congruence*, all layers used the same constant: $c=0.426 - 0.3414i$ (the only constant I found to have a limit cycle of period 15). In this way, the limit cycle itself became the structural focus. The start time parameter ensured that the densest points in the piece occurred at the fifteen time points corresponding to the limit cycle. Due to differences in parameter seeds, however, each layer articulated the fifteen points with a different type of sound event, and the transient phase of each layer varied in length and quality, although generally this resulted in sparse, isolated sound events outside of the limit cycle, specifically in the last 43.4% of the piece.

In the composition of *convergence*, on the other hand, each of the five layers used a different constant, but the parameter seeds were as close as possible, if not identical from layer to layer. The different constants caused the length of the transient phase to vary amongst the layers even with the same parameter settings; however, the initial sound events in the transient phase for all the layers began almost identical to one another, in timing and quality, and then slowly diverged. Because the 'seed sound event' was placed at the end of the piece, the start time parameter began as far from the limit cycle as possible, maximising the number of sound events in the final 43.4% of the piece. Also, since those events in the transient phase are more similar to one another than those in the limit cycle, the impression while listening to

[8]To ensure that the program does not try to execute an excessively high number of iterations, the program will not allow the maximum number of iterations to exceed 1,000,000.

[9]All three of the pieces I composed with *statics* consist of five layers. I empirically determined that superimposing five layers created enough textural variety without making the piece too busy or dense.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

the piece is that the quality of timbre slowly converges toward the 'seed sound event'.

statics only exploits the magnitudes of the limit cycles in the control structure generator. As mentioned earlier (see Section 3), angles also gravitate toward a limit cycle, but of only five values (these five angles are cycled through three times in the magnitude limit cycle of period 15 and four times in the magnitude limit cycle of period 20). *cyclian* was composed to exploit this characteristic. Like the start time parameter for each sound event, I decided to use these five angles as the starting points for each layer. I set one of the points as time zero, where the first layer begins and continues until the end of the piece, and then the second point is where the second layer starts and continues until the end of the piece. Eventually all five layers are active. As in *convergence*, one hears similar seeds but different constants for each layer. In this case, however, all of the constants have limit cycle of period 20. Because of this, the form for each layer is similar, but the form is progressively compressed as each successive layer has less time in which to articulate it.

5. CONCLUSIONS

Otto Laske describes the computer as an 'alter ego' or a 'dialoguing tool', whereby the composer strengthens his 'capacity of experimentation and unlearning' (Laske 1992). I used the computer to generate unfamiliar material by means of functional iteration. As I learned how variations in the program's input affected the audible results, I modified the implementation in order to focus the output timbre and clarify what I perceived as the formal implications of the material. This degree of flexibility necessitated that I design my own program rather than use an existing software package. It was not my goal to design a utilitarian program for use by any composer; *statics* is an embodiment of my compositional decisions. Furthermore, *statics* is specific to one series of compositions. While I view each of these compositions as an independent work, there is a limit to the output variety of *statics*, and therefore the number of compositions that can be produced by it is finite. Nevertheless, in designing the program, issues were raised that could inform further exploration of functional iteration and lead to other compositional pursuits.

In functional iteration synthesis, formal and sonic diversity are linked to implementation. The nonlinear map one chooses to use is largely irrelevant:

> Precisely because the same operation is reapplied, it is conceivable that only a select few self-consistent patterns might emerge where the consistency is determined by the key notion of iteration and *not* by the particular function performing the iterates. (Feigenbaum 1980)

These patterns include fixed points, limit cycles, and chaotic behaviour, and all nonlinear maps exhibit these behaviours, dependent on the initial settings. For example, the map used in *statics* [$z_{t+1} = (z_t+c)^2$] has a fixed point of 4 when $c = -2$ and $z_0 = 4$, 2 or 0; and the map exhibits chaotic behaviour when $z_0 = 1.5$.

The *statics* implementation is very narrowly focused on exploring periodicities in the iteration of a nonlinear map. I decided against exploring different z_0's that would yield regions exhibiting more chaotic behaviour, since this would generate a less structured form and fewer pitched/semi-pitched materials. While I certainly was not concerned with creating a pitch structure, I was attracted to the obscured pitched elements and periodic rhythms that were generated by this implementation. Obviously, *statics* could be altered to explore a wider range of possibilities, and it could then generate material for contrasting compositions.

Functional iteration synthesis is still a relatively unexplored approach to what Julius Smith III referred to as 'synthesis by abstract algorithm'. As he pointed out, 'it is difficult to find a wide variety of musically pleasing sounds by exploring the parameters of a mathematical expression' (Smith 1991). *statics* is an attempt to show that the mathematical expression by itself does not generate music; it is how the composer chooses to implement and interpret the dynamics of a mathematical system that creates musically viable material.

REFERENCES

Battey, B. 2004. Musical pattern generation with variable-coupled iterated map networks. *Organised Sound* **9**(2): 137–50.

Branchi, W. 1983. The state of anxiety. *Computer Music Journal* **7**(1): 8–10.

Brün, H. 2004. Music and information and communication and chaos… In A. Chandra (ed.) *When Music Resists Meaning: The Major Writings of Herbert Brün*, p. 44. Middletown, CT: Wesleyan University Press.

Di Scipio, A. 1990. Composition by exploration of nonlinear dynamical systems. *Proc. of the 1990 Int. Computer Music Conf.*, pp. 324–7. Glasgow, Scotland: ICMA.

Di Scipio, A. 1995. Inseparable models of materials and of musical design in electroacoustic and computer music. *Journal of New Music Research* **24**(1): 34–50.

Di Scipio, A. 2001. Iterated nonlinear functions as a sound-generating engine. *Leonardo* **34**(3): 249–54.

Di Scipio, A. 2002. The synthesis of environmental sound textures by iterated nonlinear functions, and its ecological relevance to perceptual modelling. *Journal of New Music Research* **31**(2): 109–17.

Di Scipio, A., and Prignano, I. 1996. Synthesis by functional iterations: a revitalization of nonstandard synthesis. *Journal of New Music Research* **25**(1): 31–46.

Feigenbaum, M. 1980. Universal behavior in nonlinear systems. *Los Alamos Science* **1**(1): 4–27.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press

Gleick, J. 1987. *Chaos: Making a New Science*. New York: Viking.

Kaplan, D., and Glass, L. 1995. *Understanding Nonlinear Dynamics*. New York: Springer-Verlag.

Laske, O. 1992. *Furies and Voices*: composition-theoretical observations. In D. Baggi (ed.) *Readings in Computer-Generated Music*, pp. 181–97. Los Alamitos, CA: IEEE Computer Society Press.

Mandelbrot, B. 1983. *The Fractal Geometry of Nature: Updated and Augmented*. New York: W. H. Freeman.

Peitgen, H.-O., and Richter, P. H. 1986. *The Beauty of Fractals*. Berlin; New York: Springer Verlag.

Pressing, J. 1988. Nonlinear maps as generators of musical design. *Computer Music Journal* **12**(2): 35–46.

Reynolds, R. 2002. *Form and Method: Composing Music: The Rothschild Essays*, pp. 27–9. Edited by Stephen McAdams, New York; London: Routledge.

Smith, J. 1991. Viewpoints on the history of digital synthesis. *Proc. of the 1991 Int. Computer Music Conf.*, pp. 1–10. Montreal: ICMA.

Truax, B. 1990. Chaotic nonlinear systems and digital synthesis: an exploratory study. *Proc. of the 1990 Int. Computer Music Conf.*, pp. 100–3. Glasgow, Scotland: ICMA.

https://doi.org/10.1017/S1355771808000083 Published online by Cambridge University Press