CHAPTER 8

MUSICAL DESIGN I: EVOLUTIONS

<u>Evolutions</u> are gradual changes. In normal usage, the word also connotes physical or normative growth -- as contrasted to <u>regression</u> -- but for the purposes of this chapter it is more constructive to embrace all gradual changes, regardless of direction. Commonplace musical examples of evolutions in this generalized sense include crescendi and diminuendi, accelerandi and ritardandi, harmonic sequences, gradually ascending or descending melodies, and so on. Often two or more evolutions will be concerted; for example, a musical passage might rise in register while increasing in tempo and loudness until it reaches a "plateau" during which register, tempo, and loudness remain constant for several seconds prior to a dramatic shift to low register, slow tempo, and soft dynamics.

8.1 DETERMINATE EVOLUTIONS

The most versatile approach to implementing evolutions using a computer is to divide them into segments, with pairs of consecutive segments joining at single common points called _nodes_. Each segment will then be describable by three quantities:

1. an initial value,
2. a final value, and
3. the time required to evolve from one value to the next.

A composer can use this approach either to describe intuitively conceived evolutions or to approximate any mathematical relationship with arbitrary accuracy. It is left to the computer to effect gradual transitions between the initial and final values of a segment by determining intermediate values along a continuous line or curve for each item in the segment.

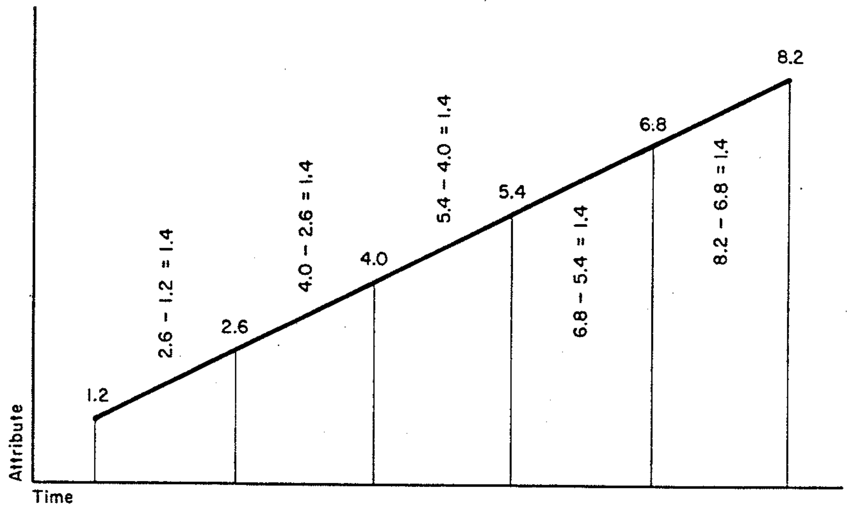
8.1.1 Evolutionary Segments

The two simplest types of evolution requiring only endpoints and a duration are the _linear_ and _exponential_ evolutions illustrated in Figure 8-1 and Figure 8-2. Figure 8-3 illustrates _piecewise linear_ and _piecewise exponential_ evolutions. An evolution is piecewise linear when each of its segments is linear. It is piecewise exponential when each of its segments is exponential. With a sufficient number of segments, one may describe any evolution imaginable as a piecewise linear or piecewise exponential curve. Notice the evolutions in Figure 8-3 each include two nodes for which the final point of the old segment does not coincide with the initial point of the new one; such abrupt changes are called _discontinuities_.

Figure 8-1: Linear evolutions - a) Ascending (increment positive); b) Descending (increment negative).

Figure 8-2: Exponential evolutions - a) Ascending (proportion exceeds unity); b) Descending (proportion smaller than unity).

Figure 8-3: Pieced evolutions - a) Linear; (b) Exponential. Closed dots joining segments indicate connected nodes; open dots indicate discontinuities.


1.2
2.6 − 1.2 = 1.4
2.6
4.0 − 2.6 = 1.4
4.0
5.4 − 4.0 = 1.4
5.4
6.8 − 5.4 = 1.4
6.8
8.2 − 6.8 = 1.4
8.2
Attribute
Time

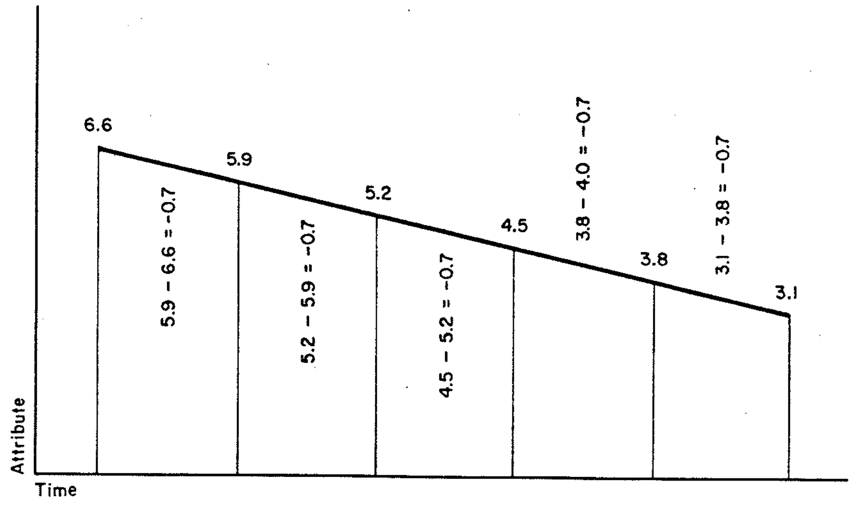
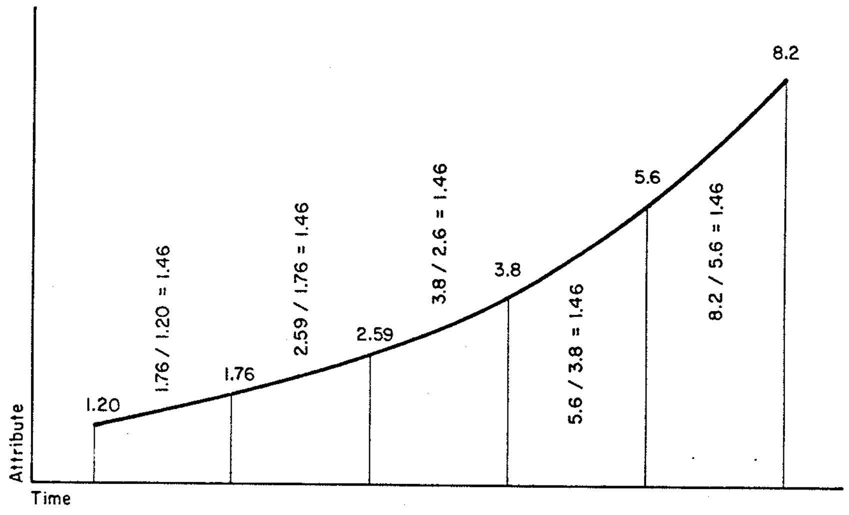

6.6
5.9 − 6.6 = −0.7
5.9
5.2 − 5.9 = −0.7
5.2
4.5 − 5.2 = −0.7
4.5
3.8 − 4.0 = −0.7
3.8
3.1 − 3.8 = −0.7
3.1
Attribute
Time

Fig 8-1


1.20
1.76 / 1.20 = 1.46
1.76
2.59 / 1.76 = 1.46
2.59
3.8 / 2.6 = 1.46
3.8
5.6 / 3.8 = 1.46
5.6
8.2 / 5.6 = 1.46
8.2
Attribute
Time

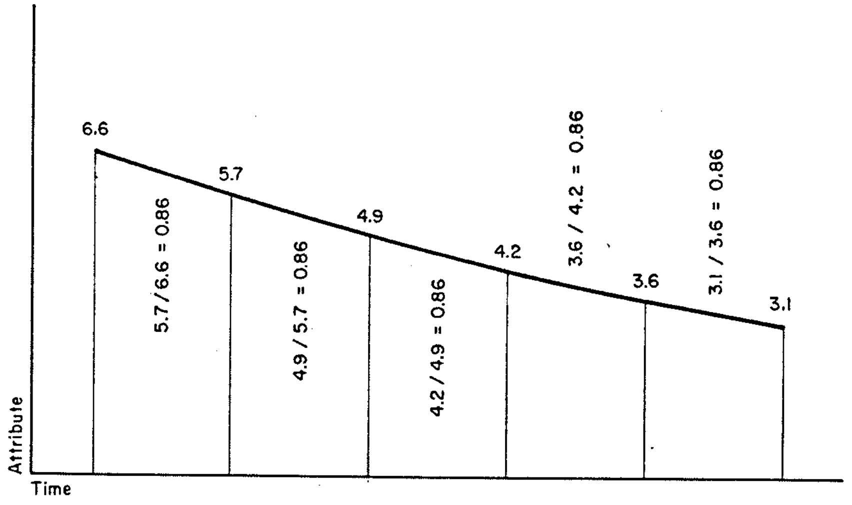

6.6
5.7/6.6 = 0.86
5.7
4.9 / 5.7 = 0.86
4.9
4.2 / 4.9 = 0.86
4.2
3.6 / 4.2 = 0.86
3.6
3.1 / 3.6 = 0.86
3.1
Attribute
Time

Fig 8-2

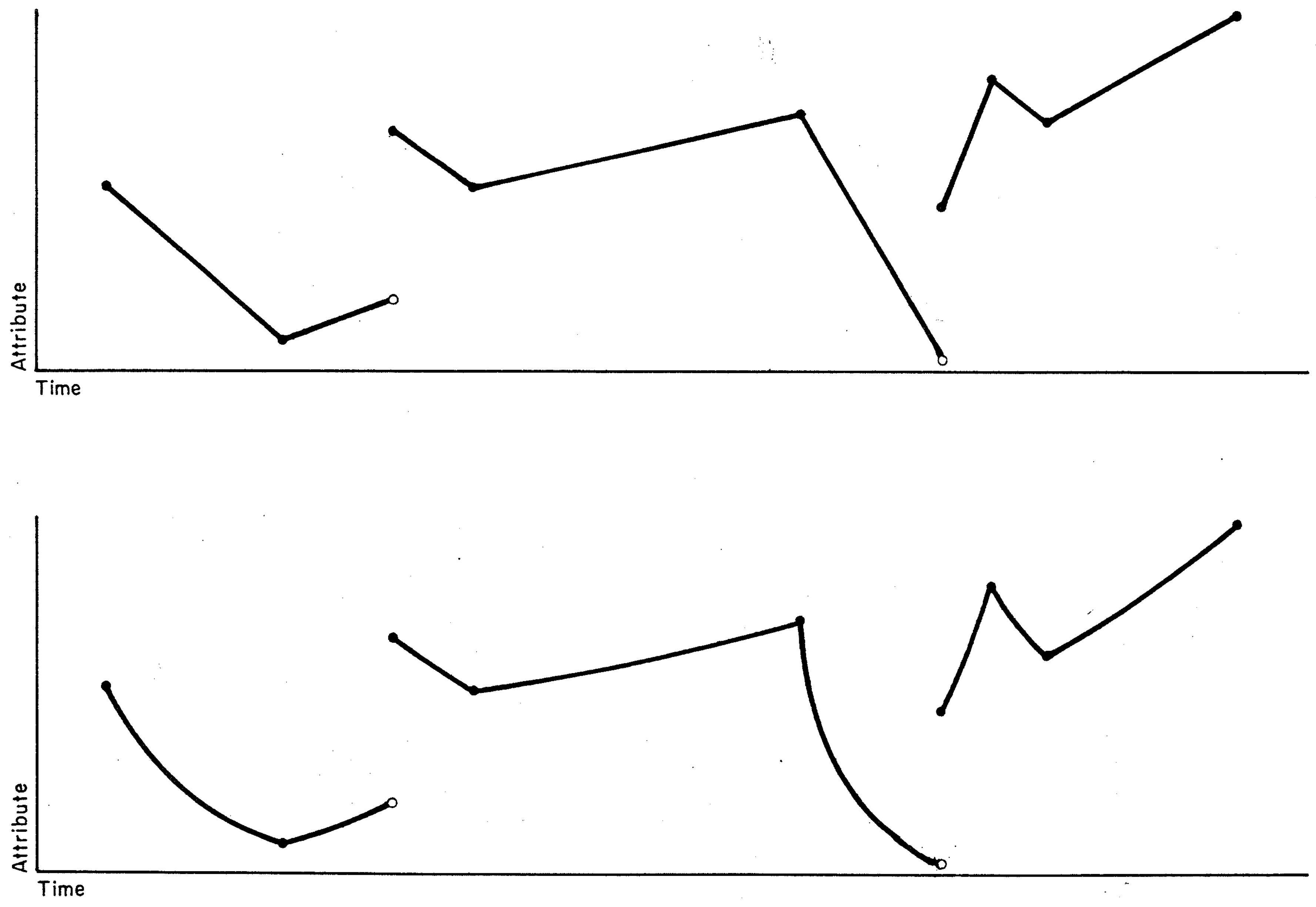


Fig 8-3

8.1.1.1 Linear Segments - An attribute of a musical passage evolves linearly if it changes by equal *increments* over equal units of time. Linear evolutions *ascend* when increments are positive and *descend* when increments are negative. A typical attribute which might follow a linear evolution is register.

8.1.1.2 Exponential Segments - An attribute evolves exponentially if it changes by equal *proportions* over equal units of time. Exponential evolutions *ascend* when proportions exceed unity and *descend* when proportions are smaller than unity. A gradual change in tempo (that is, an accelerando or ritardando) is felt to be most uniform when it follows an exponential evolution. An important property of exponential evolutions is that whenever an attribute evolves exponentially in one direction, its inverse evolves exponentially in the opposite direction. For example the inverse of duration is tempo. Therefore, if the tempo of a musical passage exponentially increases then the durations in the musical passage will exponentially decrease, as illustrated in Figure 8-4. Another property of exponential evolutions is that they are only capable of assuming positive values: it makes little sense to talk of a negative duration, for instance.

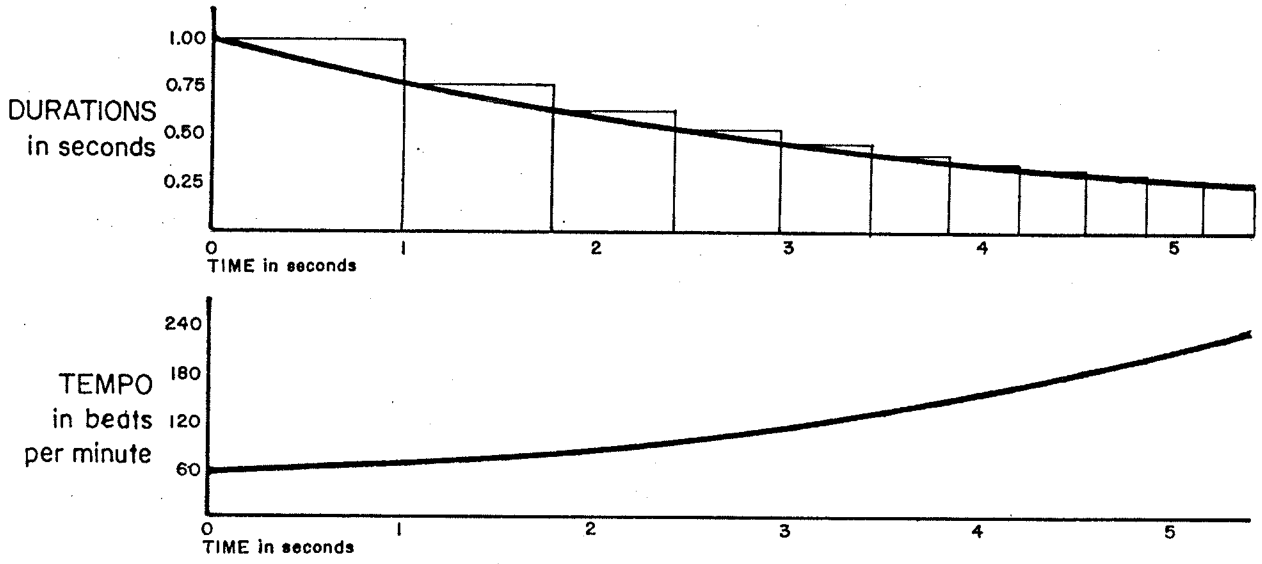

DURATIONS
in seconds
1.00
0.75
0.50
0.25
0
1
2
3
4
5
TIME in seconds
TEMPO
in beats
per minute
240
180
120
60
0
1
2
3
4
5
TIME in seconds

Fig 8-4

Figure 8-4: Relationship between duration and tempo.

8.1.1.3 Relationship between Linear and Exponential Evolutions - The relationship between linear and exponential evolutions is direct and significant: one obtains a linear evolution from an exponential evolution by taking a logarithm to a suitable base; conversely, one obtains an exponential evolution from a linear evolution by taking an exponent to a suitable base. For example, the distance traveled by a glissando may be expressed either in semitones or in frequency ratios. The most uniform glissandi proceed linearly when expressed in semitones, exponentially when expressed in frequencies. To express a frequency ratio as a distance in semitones, one takes the logarithm to the base twelveth-root-of-two, as computed in Equation 8-1:

$$\begin{aligned}\text{(semitones)} &= \log_{2**(1/12)} \text{(frequency ratio)} \\ &= 17.31 \log_{e} \text{(frequency ratio)} \qquad \text{(Equation 8-1)}\end{aligned}$$

To express a distance in semitones as a frequency ratio, one takes the exponent to the base twelveth-root-of-two, as computed in Equation 8-2:

$$\text{(frequency ratio)} = (2^{**}(1/12))^{\text{(semitones)}}$$

$$= 1.0595^{\text{(semitones)}} \qquad \text{(Equation 8-2)}$$

8.1.1.4 Implementation - The process of computing an intermediate value of an evolution given its initial and final values is called _interpolation_. Both linear and exponential interpolations depend on the ratio of the elapsed time for the current segment to the total duration of the segment. This ratio is called the _interpolating factor_. In cases where an item within a segment adheres to several concerted evolutions, it becomes redundant to recompute this factor in each case. The real-valued library function FACTOR isolates the task of computing an interpolating factor. FACTOR requires three real arguments:

1. T - Current time.
2. TA - Beginning time for current segment. TA must not exceed T.
3. TB - Ending time for current segment. T must not exceed TB.

Intermediate values along both types of segment may be computed directly from the initial value, final value, and interpolating factor. The real-valued library functions EVLIN and EVEXP

Ex 8-1

```
function FACTOR(T,TA,TB)
if (TA.gt.T .or. T.gt.TB) stop 'Bad argument to FACTOR.'
FACTOR = (T-TA) / (TB-TA)
return
end
```

[illegible]

```
function EVLIN(A,B,F)
EVLIN = A + (B-A)*F
return
end
```

```
function EVEXP(A,B,F)
if (A.le.0.0 .or. B.le.0.0) stop 'Bad argument to EVEXP.'
EVEXP = A * (B/A)**F
return
end
```

compute values along linear and exponential evolutions, respectively, given three real arguments:

1. A - value of evolution at beginning of current segment.
2. B - value of evolution at end of current segment.
3. F - interpolating factor (provided by function FACTOR).

Values of A and B supplied to EVEXP must always be positive.

-- Programming example 8-1: functions FACTOR, EVLIN, EVEXP --

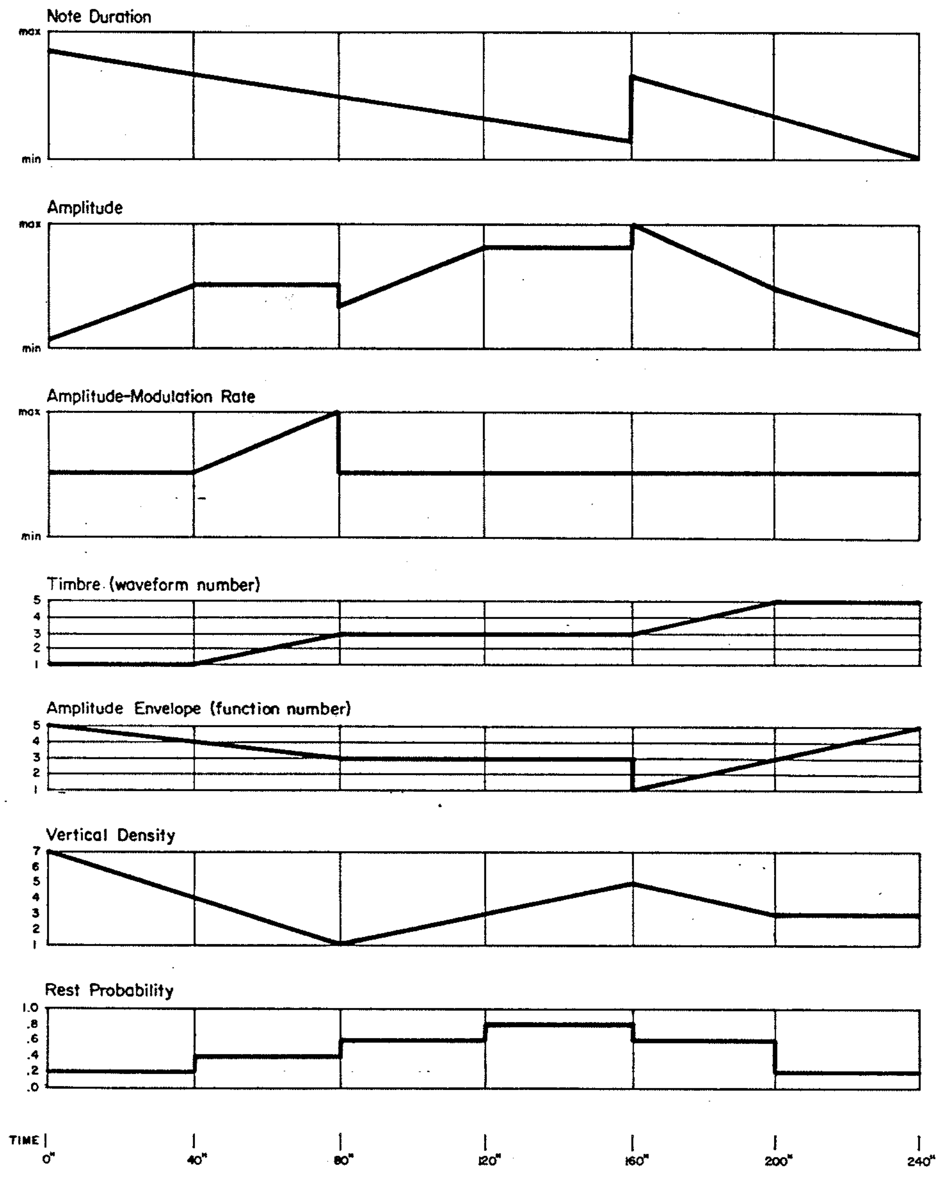
8.1.1.5 Transformations - Other types of evolutions can be derived from linear and exponential ones through the use of appropriate statistical transformations. Xenakis (1971, pages 47-49) suggests using the Fletcher-Munson equal-loudness contours in order to map the human ear's very non-linear responses to loudness and frequency onto a rectangular grid. For his composition <u>Gradient</u> (1982), the author used the proportional transformation described under heading 4.2.3.2 in order to 'warp' essentially linear evolutions so that intervals in the lowest registers would be about 1.5 times more widely spread than intervals in the highest registers.

8.1.2 Musical Evolutions

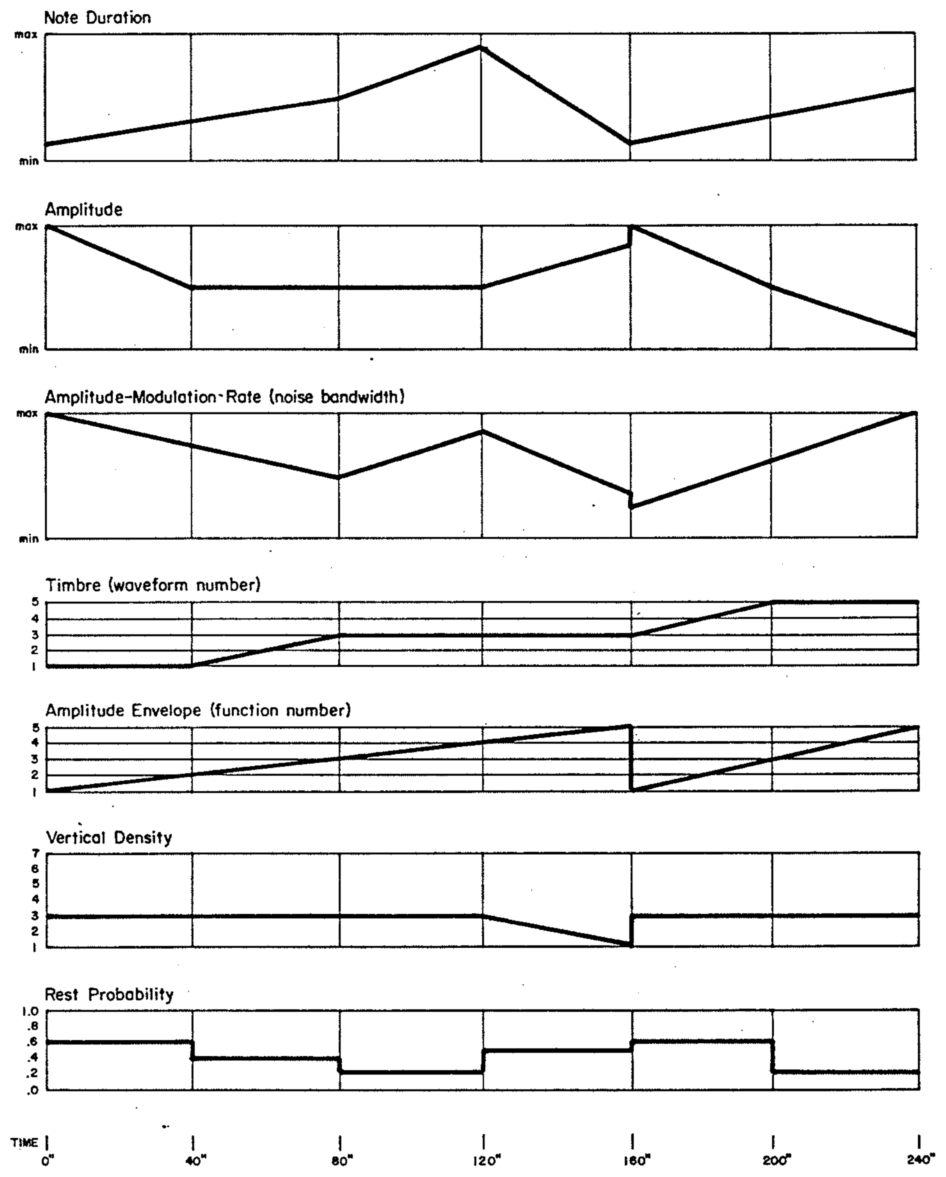
8.1.2.1 Johannes Kepler: Harmonices Mundi - The first musical composition exploiting evolutions as a fundamental precept is the "Music of the Spheres" by the celebrated astronomer Johannes Kepler. Kepler gives very explicit instructions for producing this work in his 1619 treatise Harmonices Mundi. The composition consists of several continuous tones, one for each planet, where the frequency of each tone is proportional to the speed at which the planet moves through space. Since planets accelerate as they approach the sun and decelerate as they leave it (as Kepler himself discovered), the tones of Kepler's musical composition are heard to gliss upwards and downwards in periodic cycles. The technology required to realize this curious work did not exist until the advent of computers; however, it has been realized digitally by Laurie Spiegel (1977) and independently by digital synthesists at Yale University.

8.1.2.2 Joseph Schillinger's Melody Graphs - Joseph Schillinger (1941) advocates a method using the above approach to compose melodies on graph paper. This method involves three steps: 1) drawing out a melodic contour as a sequence of slanted lines, with time represented horizontally in beats and pitch represented vertically in semitones, 2) marking off a rhythm along the horizontal axis, and 3) selecting pitches for each note by determining the scale degree in closest proximity to the line at each point of attack. Schillinger describes elaborate procedures for designing contours and also includes graphs of non-linear contours such as sinusoidal curves. His confidence in his method brings him to contrast melodies composed in this manner to "melodies composed by a Verdi or a Bellini [, where] the mechanical efficiency is so low that it makes us smile, if not laugh" (page 283).

8.1.2.3 James Tenney: Dialogue - More recent activity in this area sheds Schillinger's pretentions while retaining his basic approach. In his Dialogue for computer-generated sounds (1963; described 1969) and in the earlier described Stochastic String Quartet (heading 4.3.3), James Tenney used graphs pieced together from linear segments to describe gradually evolving



Tonal Stratum



Noise Stratum

Fig 8-5

8-9a

values affecting such musical attributes as average tempo, dynamics, timbre, and articulation. These values then served as parameters for random automata which in turn determined specific attributes of notes. Figure 8-5 depicts the parametric graphs used by Tenney to create Dialogue.

Figure 8-5: Parametric means for James Tenney's Dialogue - Copyright James Tenney 1969. Spectral increases with waveform numbers; rise times increase with amplitude envelope numbers (after Tenney, 1969).

8.1.2.4 Other Applications - Tenney's work is reflected in the article "Graphical language for the scores of computer-generated sounds" (1968) and book The Technology of Computer Music (1969), where Max Mathews, et al, describe rudimentary procedures for "quantizing" line graphs in order to produce sequences of durations, "duty factors", pitches, and amplitudes. Each graph is pieced together out of linear segments.

Among the procedures described by Emmanuel Ghent (1978) for manipulating stored melodies in real time is a method of "interpolating" between melodic pitches. Figure 8-6 illustrates how this method works. As with Ghent's method of "translocation" (heading 3.3.1.), it is necessary that the original melody be stored as a sequence of indicies relative to a separate array holding the repertory of pitches. Given two consecutive notes in a melody, interpolations are effected by sliding a real variable X linearly from the first index to its successor. Intervening pitches are obtained by taking the integer part of X and using this truncated value as an index in its own right.

Figure 8-6: Emmanuel Ghent's Method of Melodic Interpolation - after Ghent (1978). Copyright 1978 Emmanuel Ghent.

Applications of gradual evolutions include some fanciful effects in which one evolution works perceptually to neutralize another. Such effects trace back to the infinitely rising glissandi conceived by Shepherd and realized by Jean-Claude Risset (1970): at least two frequencies related by an octave are always constantly sounding, but as each frequency approaches the upper limit of register, it fades away to be replaced by a gradually intensifying frequency one or more octaves lower. Scott Kim has incorporated this principle in a realization of a perpetually modulating canon by Bach so that whenever the canon completes a full cycle of keys, it returns to its original register. Laurie Spiegel (1974) has realized a "perpetual acceleration" in which every other beat fades away with each doubling of tempo.

An alternative to encoding an evolution as alphanumeric data is to 'perform' the evolution by controlling a knob or similar device which the computer monitors in real time. Mathews and Moore's GROOVE (1970) was designed for creating and editing evolutions in this very manner; Laurie Spiegel has employed GROOVE in just such a capacity for works such as Pentachrome and The Expanding Universe (both 1974). In each case, Spiegel supplied her own programs which accept information supplied in real time from GROOVE and "derive from it much more complex music than [Spiegel] actually played" (Spiegel, 1980).

8.1.3 Oscillations

Figure 8-7: A sinusoidal oscillation.

In contrast to linear and exponential evolutions are oscillations. Where linear and exponential evolutions proceed directly from a source to a destination, oscillations swing back and forth around a point of equilibrium. The "smoothest" oscillations are sinusoidal oscillations such as the curve

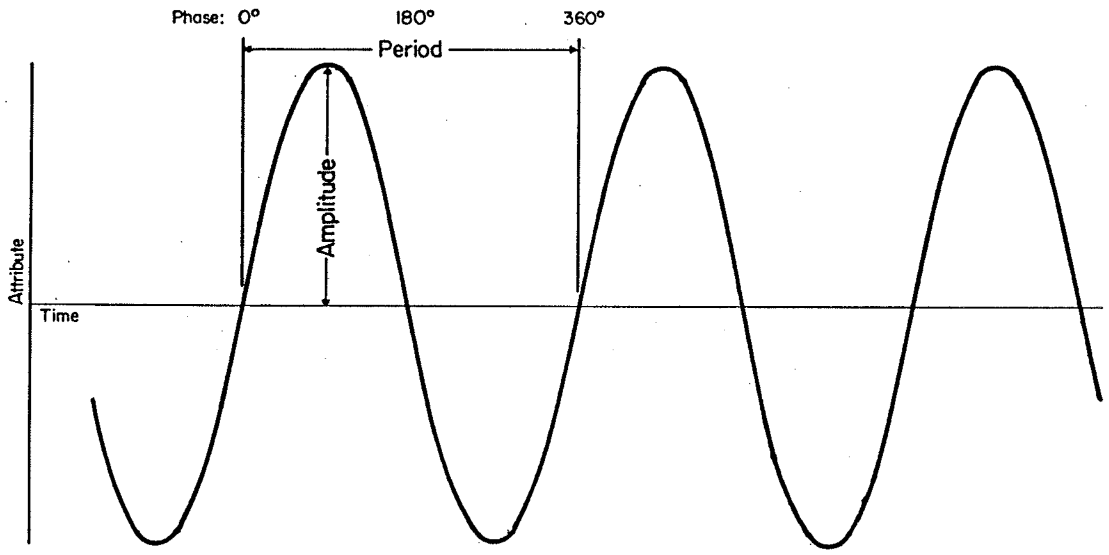


Fig 8-7

depicted in Figure 8-7. Each full swing of a sinusoid brings the oscillation from the point of equilibrium up to a "peak", back through the point of equilibrium to a "trough", to return once again to the point of equilibrium. The length of time occupied by one full swing is called the <u>period</u>, while the number of periods contained within a given unit of time (e.g., a second, a whole note, etc.) is called the <u>frequency</u>. The vertical distance vertically from the point of equilibrium to either the top of a peak or to the bottom of a trough is called the <u>amplitude</u>. The <u>phase</u> expresses the relative position within a swing at the beginning of an oscillation. Because a sinusoid "wraps around" upon itself with each new swing, it is most appropriate to express phase using a circular measurement such as degrees or radians.

The smoothness of sinusoidal oscillations has made them especially attractive to composers interested in cyclic forms. James Tenney's <u>Phases</u> (1963; described 1969), John Myhill's <u>Scherzo a tre voce</u> (1964; partially described in Hiller, 1970) and Gary Kendall's <u>Five Leaf Rose</u> (1980; described 1981) each exploit sinusoids, though in widely different ways.

8.1.3.1 John Myhill: _Scherzo a tre voce_ - John Myhill's _Scherzo a tre voce_ derives its melodic contours from gradually evolving sinusoids. The _Scherzo_ is a work both composed and realized using computers, and it is directly inspired by graphic contours depicted by Schillinger (1941). Each of the three voices remains within a unique one-and-one-half octave range and proceeds independently of the others in a chain of phrases. Myhill's program selected melodic contours from the eight basic forms illustrated in Figure 8-8 by specifying two characteristics:

1. _Deviation from central pitch_, with three options: a) deviation _constant_ at a tritone (trough-to-peak range: one octave), b) deviation _increasing_ linearly from a unison to a tritone, c) deviation _decreasing_ linearly from a tritone to a unison.

2. _Period of oscillation_, with three options: a) constant period, b) decreasing period (frequency of melodic oscillation increases linearly from zero) more often), c) increasing period (frequency decreases linearly to zero).

Myhill eliminates those contours characterized by both constant

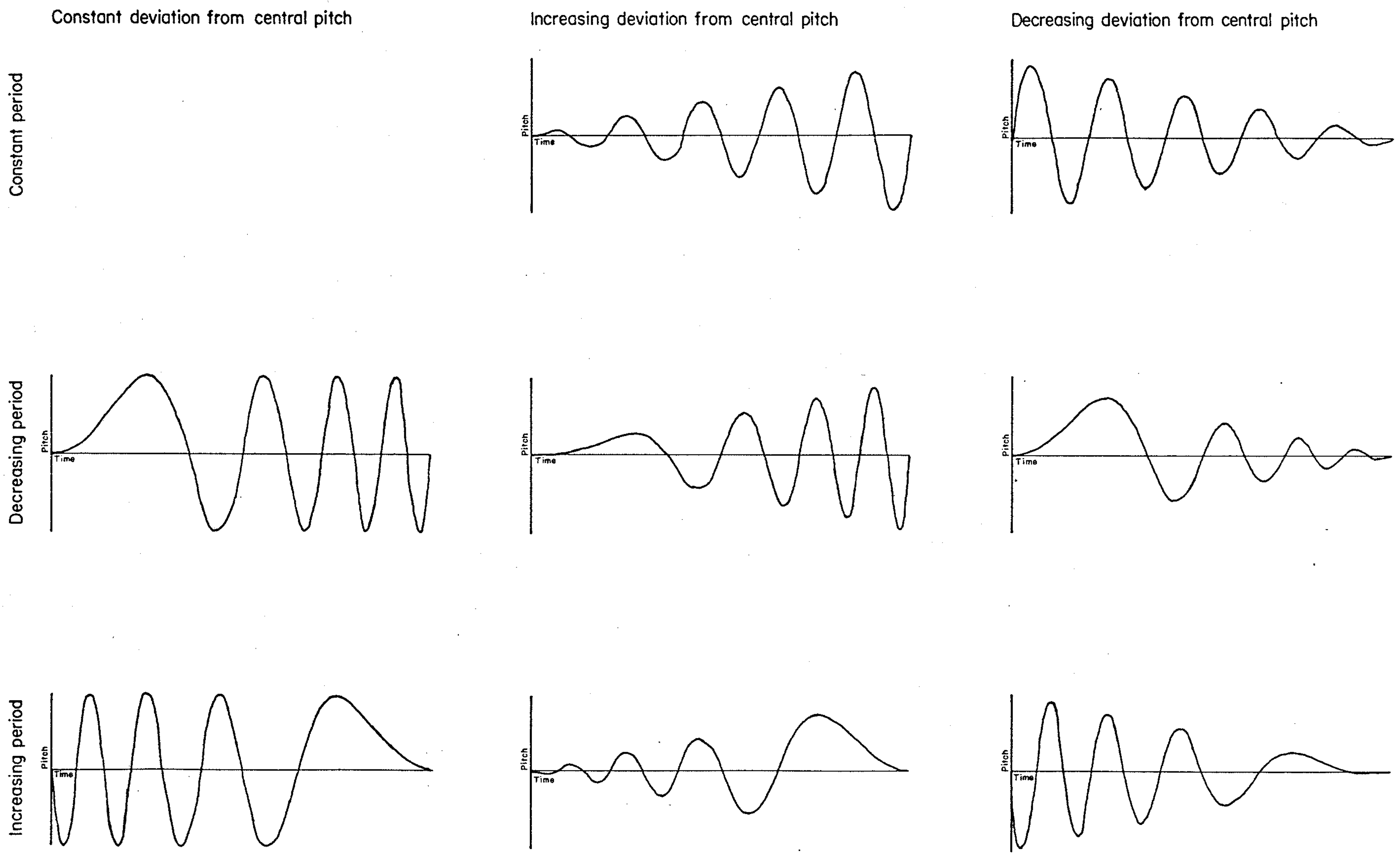

Constant deviation from central pitch
Increasing deviation from central pitch
Decreasing deviation from central pitch
Constant period
Decreasing period
Increasing period
Pitch
Time

Fig 8-8

gamut and constant rate of oscillation, reducing the number of possible shapes from nine to eight. In addition, each phrase may trace its assigned contour in one of three ways: a) the melody can consist of a continous sinusoidal glissando, b) the melody can consist of discrete pitches proceeding according to a predefined scale and rhythm, with each pitch determined by the scale degree in closest proximity to the contour (Schillinger's approach) or c) the melody can consist of discrete pitches proceeding according to a predefined scale, so that each pitch begins when the contour crosses the corresponding scale degree. Durations in the last case depend on the steepness of the curve as is passes from one scale step to the next.

Figure 8-8: Melodic contours in John Myhill's Scherzo a tre voce.

8.1.3.2 Implementation - Sinusoidal oscillations may be computed using the real-valued function SIN provided by the standard FORTRAN library of mathematical functions. Suppose we wish to compute the value of a sinusoidal evolution after T units, given amplitude AMP, frequency FREQ, and phase PHASE. Our call to function SIN (note 1) would then take the form:

Ex 8-2

```
      V = AMP * SIN(6.2831853*T/FREQ+PHASE) + CENTER
```

Ex 8-3

```
C     Determine interpolating factor
      F = FACTOR(T,T1,T2)
C     Determine lower and upper bounds for current range
      A = EVLIN(A1,A2,F)
      B = EVLIN(B1,B2,F)
C     Select random number uniformly from this region
      X = A + (B-A)*RANF()
```

Ex 8-~~4~~5

```
      F = FACTOR(T,T1,T2)
      DUR = RANX(EVEXP(2.0,7.0,F),EVEXP(1.0,8.0,F))
```

-- Programming Example 8-2 --

8.2 EVOLVING RANDOMNESS

8.2.1 Evolving Uniform Randomness

It is a simple matter to describe evolving ranges of uniform randomness. Suppose at time T1 we want some musical attribute X to be distributed uniformly between a lower bound A1 and an upper bound B1. Suppose further that we wish this range to evolve gradually until at time T2, X is distributed uniformly between a new lower bound A2 and a new upper bound B2. Then for each time T between T1 and T2 it will be necessary 1) to determine momentary bounds A and B and 2) to select X uniformly between A and B. The following excerpt of code illustrates this process:

-- Programming example 8-3 --

The last calculation occurs frequently enough to merit a special

Ex 8-4

```
function UNIFRM(A,B)
if (A.gt.B) stop 'Bad argument to UNIFRM.'
UNIFRM = A + (B-A)*RANF()
return
end
```

real-valued library function UNIFRM which returns random values uniformly distributed between a lower bound A and an upper bound B. Both A and B must be real:

-- Programming example 8-4: function UNIFRM --

8.2.2 Evolving Means and Variances of Random Automata

Using random functions like RANX, and GAUSS, it is sufficient to obtain means and variances from EVLIN or EVEXP. For example, suppose we wish to generate a sequence of durations which is fast and periodic at time T1 but which evolves gradually until at time T2 it is slower and moderately periodic. Table 8-1 gives illustrative values for average durations and for ratios between the maximum and minimum durations:

Table 8-1: Illustrative values for an evolving sequence of rhythmic periods.

Then for each individual duration DUR beginning at time T between T1 and T2, our composing program would invoke RANX (heading

Time	Average Period	Maximum Proportion	Description
T1	2.0	1.0	fast, periodic
T2	7.0	8.0	slower, moderately aperiodic

Table 8-1

4.2.1.1) as follows:

-- Programming example 8-5 --

8.2.3 Evolving Discrete Randomness

Evolutions affecting discrete repertories of options can be implemented by supplying gradually changing parameters to random automata such as IBINOM or IPOISS. However, for many applications it may be desirable to employ some specialized density function or to take steps to insure that the individual decisions adhere rigorously to the intended statistical characteristics.

8.2.3.1 Direct Selection

Direct Selection - A direct approach to evolving discrete randomness is to compute a stored distribution of weights using some specialized density function. If the distribution evolves rapidly, such computations should be reinitiated prior to each new decision. The program might then pass this stored distribution to library subroutines such as

SELECT (heading 4.2.2.5), CURVE (heading 4.2.3.4), HEUR (heading 7.1.1), or DECIDE (heading 7.2.1).

An elegant instance of direct selection may be found in the TENDENCY feature of Gottfried Michael Koenig's PROJECT2 program (1970b). Like PROJECT2's other selection features, TENDENCY selects options from a "supply" of NUM options provided by the user. In addition, the user provides an evolving "mask" which determines upper and lower limits for region of uniform probability between 0 and NUM. In order to select an attribute for a note, TENDENCY selects a random value within the region effective at the note's starting time and then rounds this value upward to determine which option in the repertory is to be chosen. Koenig's method seems to make most sense when the options are equally spaced along a continuum. For examples, TENDENCY might be asked to select from the following repertory of dynamics: pp, p, mp, mf, ff; alternately, TENDENCY might be used to select from the following ways of producing string colors: playing on the fingerboard, normal playing, playing on the bridge.

Because Koenig's procedure requires information describing not only the repertory of options, but also the starting time of the note (alternatively, chord, phrase, and so on) and the characteristics of the mask, it is usually most convenient to implement Koenig's procedures explicitly in a composing program.

Ex 8-6

```
        subroutine TEND(RESULT,VALUE,T,T1,T2,A1,A2,B1,B2,NUM)
        dimension VALUE(1)
C       Determine interpolating factor
        F = FACTOR(T,T1,T2)
C       Determine lower and higher limits for current region
        A = EVLIN(A1,A2,F)
        B = EVLIN(B1,B2,F)
        if (A.lt.0.0.or.B.gt.1.0) stop 'Bad argument to TEND.'
C       Select random number uniformly from this region
        R = UNIFRM(A,B)
C       Scale this number to size of table
        R = R * float(NUM)
C       Look up corresponding entry in array VALUE
        RESULT = VALUE(ifix(R)+1)
        return
        end
```

Strictly for the purpose of illustrating what considerations are involved in implementing this procedure, we supply a subroutine called TEND which details the basic calculations in FORTRAN '77. In addition to a 'return' argument RESULT, a repertory of options VALUE, and a number of options NUM, TEND requires a current time T and six characteristics describing the current segment of the mask: a starting time T1, an ending time T2, initial lower and upper limits A1 and B1, and final lower and upper limits A2 and B2. The task of determining these characteristics is left to the main program. TEND allows the repertory of options and the mask to be described independently by assuming that a lower limit (A value) of zero and an upper limit (B value) of unity describes the full range of options. (Such independence is not always desirable). Notice that in order to keep the quantity R computed in line 12 within the range from 0 to NUM, A1 and A2 may never fall below zero while B1 and B2 may never exceed 1.

-- Programming example 8-6: subroutine TEND --

A useful generalization of Koenig's approach would embrace options with varying weights. For example, one might use the following repertory of durations, weighted so that the longer durations occur proportionately less often: duration 2 with weight .405, duration 3 with weight .270, duration 5 with weight

.101, duration 8 with weight .086, and duration 13 with weight .062. One would then describe evolving limits for regions of uniform probability between zero and the sum of these weights, 1.000, and use the methods of the library subroutine SELECT (heading 4.2.2.5) to transform values chosen from these regions into discrete durations.

8.2.3.2 Evolving Statistical Frames - Another approach to evolving discrete randomness involves dividing an evolutionary segment into statistical frames. At the beginning of each frame, the program determines weights for each option and then proceeds to generate a determinate pool of samples (chapter 5). It may then shuffle this pool randomly or organize it in various other ways (such as sorting, described in chapter 9 and comparative search, described in chapter 12).

8.2.4 Musical Applications

The excerpt of Koenig's "Uebung fur Klavier" examined in chapter 5 (heading 5.4) is exceptional in that Koenig eschews using TENDENCY, practically every other "structure" in the work employs TENDENCY in one role or another. A more recent computer-composed work for piano, Thomas DeLio's _Serenade_ (1974) provides a further illustration of evolving discrete randomness.

8.2.4.1 Thomas DeLio: _Serenade_ - DeLio's _Serenade_ is an elaborate work for which the form was composed manually while the details were selected automatically. It divides into three parts, each dividing in turn into ten sections; marked contrasts and occasional long silences between many of these sections give the work an episodic quality congenial to its title. The basic ideas consist of various rhythmic patterns and various chromatic "cells". Part I introduces these ideas is "embryonic states"; they emerge "fully formed" in part II, which also serves a developmental purpose; part III extrapolates several of the ideas from earlier parts into "broad sweeping gestures".

Figure 8-9: Part 1, section 10 of Thomas DeLio's _Serenade_ for solo piano. Copyright 1982 Dorn Publications.

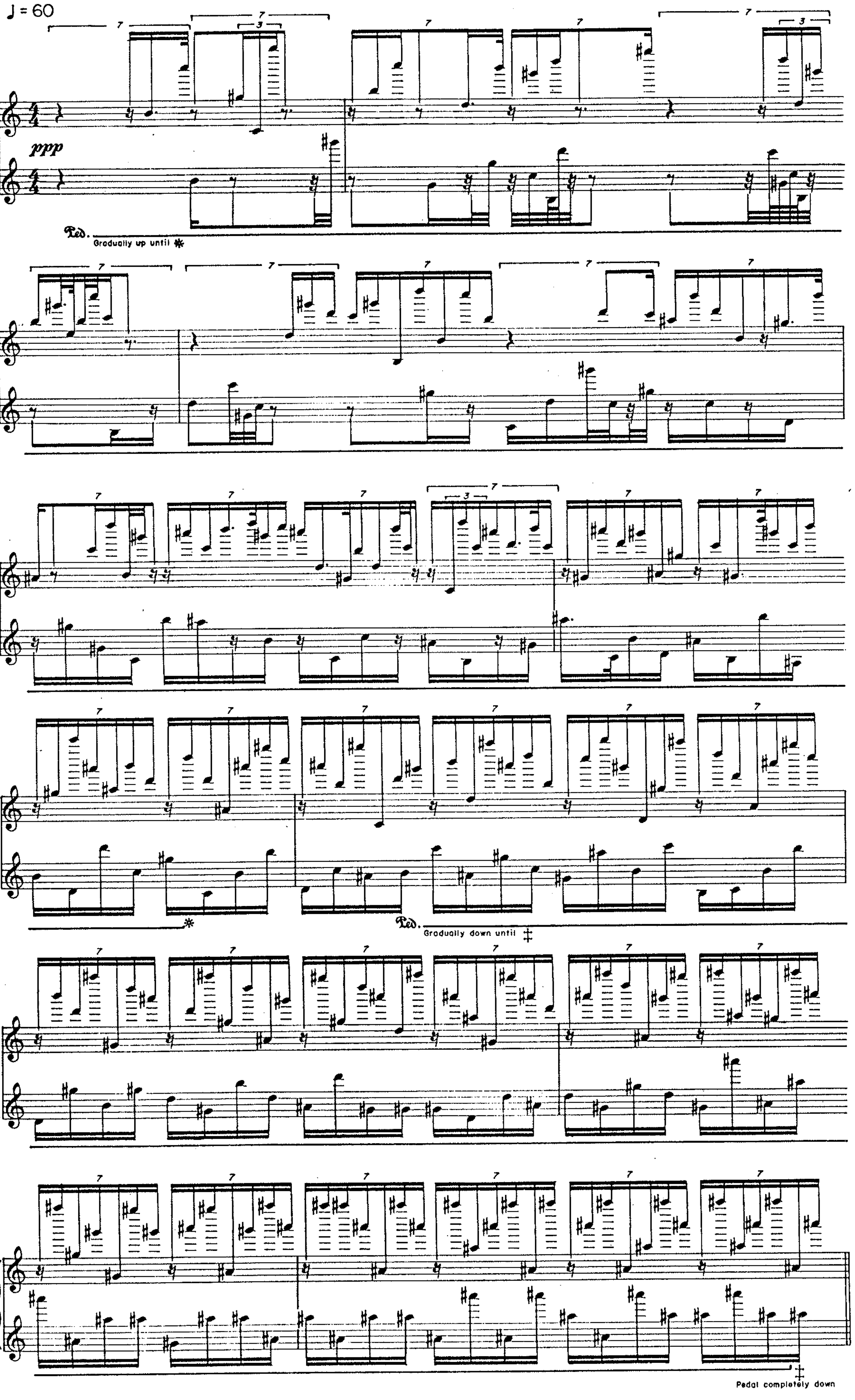

♩ = 60
ppp
Ped.
Gradually up until ✱
✱
Ped.
Gradually down until ‡
‡
Pedal completely down

Fig 8-9

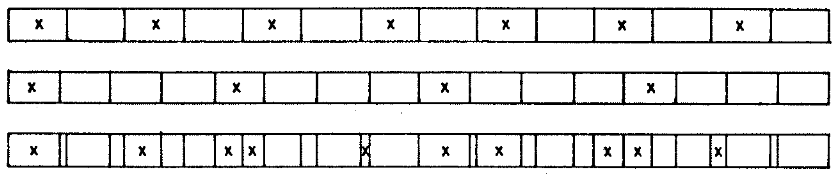


Fig 8-10

Figure 8-10: Interferences resulting from a 7:4 polyrhythm - X's indicate 'primary' beats.

The following description of the compositional procedures used in part I of section 10 has been drawn from commentary supplied to the author by DeLio. This excerpt is reproduced in Figure 8-9. The basic "ideas" employed in this section are a 7:4 polyrhythm and two chromatic cells: the collection G# B C D and its reflection, G# A# B D. Registers are distributed uniformly over the range from middle C upward. The section consists of 34 beats and breaks into two gradual evolutions lasting an equal number of beats.

Rhythm - Over the first 17 beats, the rhythm coalesces from a sparse, irregular texture to a consistent 7:4 pattern. DeLio accomplishes this evolution by treating each beat as a statistical frame whose elements are the 29 unequal rhythmic units obtained by interfering septuplet thirty-seconds in seven-brackets against duple sixty-fourths. Figure 8-10 depicts the resulting pattern. In this section, only one note may attack at a time; the number of attacks per beat results from a Markov process in which the number of attacks may either stay the same, increase by one, or (with relatively small probability) decrease by one from one beat to the next. This process resulted in the

following chain:

5 6 6 7 7 7 8 7 8 8 9 9 10 10 10 10 10 10 ...

Once the number of attacks had been determined, DeLio's program selected attacks without replacement (chapter 5) from the units depicted in Figure 8-10. Initially, each unit received equal weight; as the rhythm evolved the thirty seconds and sixty fourths received less and less weight until only the 7:4 pattern remained with one attack per unit (note 2). This pattern holds consistently through the remaining 17 beats.

Pitches - At the beginning of the section, DeLio's program selected uniformly with replacement (heading 4.2.2.5) from G#, B, C, and D. Through the first 17 beats, the likelihood of selecting C decreases gradually while the likelihood of selecting A# gradually increases, so that G#, A#, B, and D receive uniform weight at the mid-point of the section. Over the remaining 17 beats, the weights for G#, B, and D decrease gradually until only A# remains at the end of the section.

8.3 DEMONSTRATION 6: EVOLUTIONS

Demonstration 6 illustrates how evolutions might be incorporated into a composing program. The piece divides into eight segments, within which the following four <u>global</u> attributes gradually change: average period, proportion between minimum and maximum periods (syncopation), articulation, and register. The eight segments in turn divide into individual notes, each characterized by three <u>local</u> attributes: period, duration, and pitch.

8.3.1 Compositional Directives

Figure 8-11: Profile of Demonstration 6.

Figure 8-11 depicts graphically how the global attributes evolve. Both the average period and proportion between minimum and maximum periods are described by piecewise exponential curves. Articulations are selected using the methods of Gottfried Michael Koenig's TENDENCY feature from a repertory of four options:

8-25 a

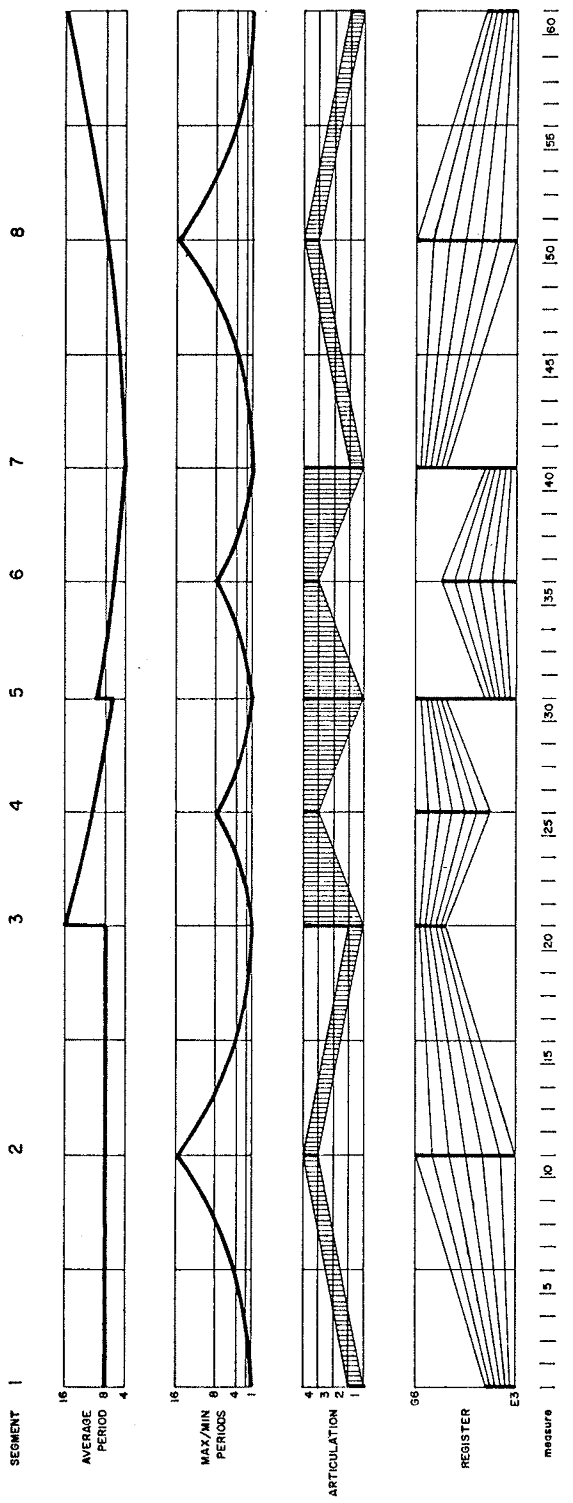

SEGMENT
1
2
3
4
5
6
7
8
AVERAGE PERIOD
16
8
4
MAX/MIN PERIODS
16
8
4
1
ARTICULATION
4
3
2
1
REGISTER
G6
E3
measure
5
10
15
20
25
30
35
40
45
50
55
60

Fig 8-11

1. Short: duration fills 50% of period,

2. Detached: duration fills 75% of period,

3. Sustained: duration fills entire period, but next note is tongued, and

4. Slurred: duration fills entire period and next note is slurred.

Registers are determined by locating a gamut of twelve adjacent semitones uniformly within the lower and upper boundaries depicted in the registral graph.

Figure 8-12: Stylistic matrix for Demonstration 6.

Of the local attributes, periods are selected accordance with the average period and maximum proportion using John Myhill's generalization of the exponential distribution (heading 4.2.3.1); durations are calculated from the period in accordance with the articulation. Chromatic degrees are selected heuristically with cumulative feedback so that the twelve degrees are equally balanced (with sensitivity to duration). However, choices of degrees are subject to the stylistic matrix depicted

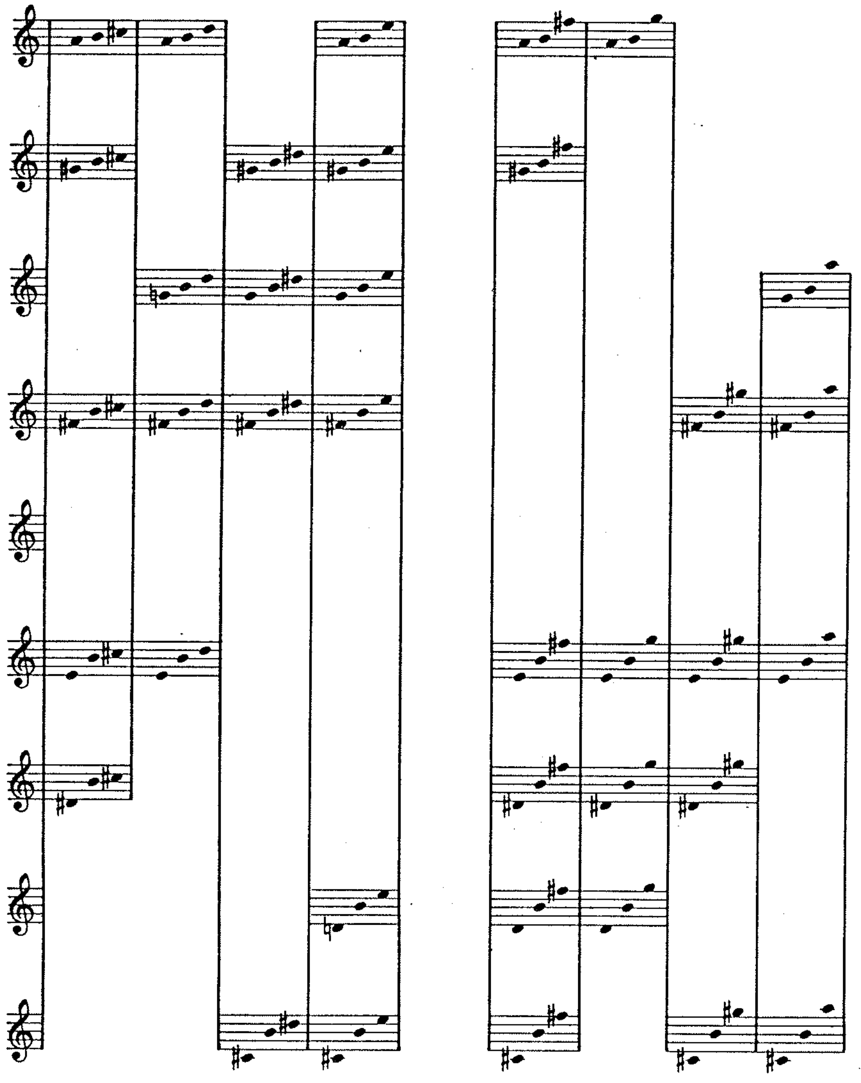


Fig 8-12

8-26b

Demonstration 6

Clarinet
STRICTLY ♩ = 80

Charles AMES

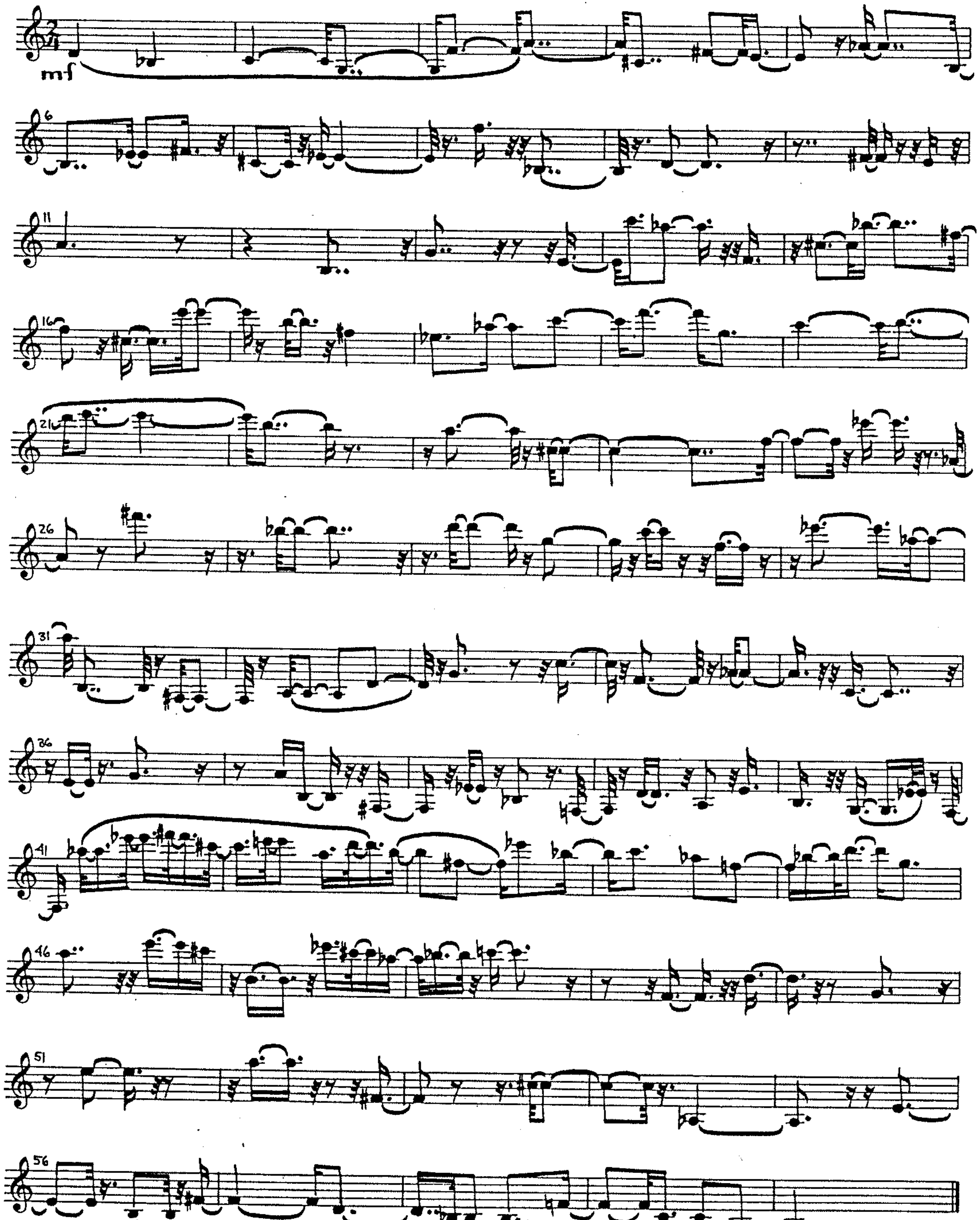


Fig 8-13

in Figure 8-12. This matrix constrains the progression of degrees so that no two out of any three consecutive degrees may form a unison, tritone, minor second, major seventh, or expansion of the any of the proceeding intervals by one or more octaves. The complete product appears in Figure 8-13.

Figure 8-13: Transcription of Demonstration 6.

8.3.2 Implementation

-- Programming example 8-7: program DEMO6 (2 pages) --

Because the global structure of Demonstration 6 is fully specified, program DEMO6 reduces to a single loop for composing notes (lines 35-86). The parameter MSEG gives the total number of segments, while the index ISEG indicates which segment the program is currently composing. Lines 37-46 serve to update ISEG whenever the current time ITIME crosses a boundary between segments. Included here is the test for completion (lines 40-43). Array KTIME holds ending times for each segment; these times are initally expressed in seconds, so must be converted to

```
      program DEMO6
C
C     Demonstration of evolutions
C
      parameter (MSEG=8)
      integer KTIME(0:MSEG)
      real BEGAVG(MSEG),ENDAVG(MSEG),BEGPRO(MSEG),ENDPRO(MSEG)
      real BEGALW(MSEG),ENDALW(MSEG),BEGAHG(MSEG),ENDAHG(MSEG)
      real BEGRLW(MSEG),ENDRLW(MSEG),BEGRHG(MSEG),ENDRHG(MSEG)
      data KTIME(0) /0,   10,   20,   25,   30,   35,   40,   50,   60/
      data BEGAVG   /   8.0, 8.0,16.0,10.1,10.1, 6.3, 4.0, 8.0/,
     :     ENDAVG   /   8.0, 8.0,10.1, 6.3, 6.3, 4.0, 8.0,16.0/
      data BEGPRO   /   1.0,16.0, 1.0, 8.0, 1.0, 8.0, 1.0,16.0/,
     :     ENDPRO   /  16.0, 1.0, 8.0, 1.0, 8.0, 1.0,16.0, 8.0/
      data BEGALW   /   0.0, 3.0, 0.0, 3.0, 0.0, 3.0, 0.0, 3.0/,
     :     ENDALW   /   3.0, 0.0, 3.0, 0.0, 3.0, 0.0, 3.0, 0.0/
      data BEGAHG   /   1.0, 4.0, 4.0, 4.0, 4.0, 4.0, 1.0, 4.0/,
     :     ENDAHG   /   4.0, 1.0, 4.0, 4.0, 4.0, 4.0, 4.0, 1.0/
      data BEGRLW   /  40.0,40.0,68.9,49.0,40.0,40.0,68.9,68.9/,
     :     ENDRLW   /  40.0,68.9,49.0,68.9,40.0,40.0,40.0,40.0/
      data BEGRHG   /  40.0,68.9,68.9,68.9,40.0,58.0,68.9,40.0/,
     :     ENDRHG   /  68.9,68.9,68.9,68.9,58.0,40.0,68.9,40.0/
      data REST/.true./,ISEG/0/,ITIME/0/,ENDTIM/0./,IDEG/1/,ITVL/3/
C
C     Initialize
C
      open (2,file='DEMO6.DAT',status='NEW')
C     Convert times into thirty-seconds
      do (I=0,MSEG)
        KTIME(I) = KTIME(I) * 16
      repeat
C
C     Main composing loop
C
      do
C       Test for end of segment
        if (ITIME.ge.KTIME(ISEG)) then
          ISEG = ISEG + 1
C         Test for end of compostion
          if (ISEG.gt.MSEG) then
            close (2)
            stop
          end if
          BEGTIM = ENDTIM
          ENDTIM = float(KTIME(ISEG))
        end if
C       Compute interpolating factor
        F = FACTOR(float(ITIME),BEGTIM,ENDTIM)
C       Compute average period between attacks
        AVGPER = EVEXP(BEGAVG(ISEG),ENDAVG(ISEG),F)
C       Compute proportion between minimum and maximum periods
        PROPOR = EVEXP(BEGPRO(ISEG),ENDPRO(ISEG),F)
C       Select period between attacks
        PER = RANX(AVGPER,PROPOR) + REMAIN
C       Select articulation
        ALOW = EVLIN(BEGALW(ISEG),ENDALW(ISEG),F)
        AHGH = EVLIN(BEGAHG(ISEG),ENDAHG(ISEG),F)
        IART = ifix(UNIFRM(ALOW,AHGH)) + 1
C       Select duration of note and rest (if any)
        if (IART.le.2) then
C         Note is sustained or slurred to successor
          IDUR = max0(2,ifix(PER+0.5))
          REMAIN = PER - float(IDUR)
          IGAP = 0
        else if (IART.eq.3) then
C         Notes filling 75% of period
          IDUR = max0(2,ifix(PER*0.75+0.5))
          IGAP = max0(0,ifix(PER-float(IDUR)))
          REMAIN = PER - float(IDUR+IGAP)
        else
C         Notes filling 50% of period
          IDUR = max0(1,ifix(PER*0.5+0.5))
          IGAP = max0(0,ifix(PER-float(IDUR)))
          REMAIN = PER - float(IDUR+IGAP)
        end if
```

Ex 8-7

8-27b

```
C       Select register
        ALOW = EVLIN(BEGALW(ISEG),ENDALW(ISEG),F)
        AHGH = EVLIN(BEGAHG(ISEG),ENDAHG(ISEG),F)
        IREG = ifix(UNIFRM(ALOW,AHGH)+0.5)
C       Select degree
        call DEGREE(IDEG,ITVL,float(IDUR))
C       Write note
        call WNOTE(ITIME,IDUR,IDEG,IREG)
C       Write rest or break (if any)
        if (IART.gt.1) call WNOTE(ITIME,IGAP,0,0)
      repeat
      end

      subroutine DEGREE(IDEG,ITVL,DUR)
      parameter (MDEG=12,MTVL=11)
      integer IGLTVL(MTVL,MTVL),ISCHED(MDEG)
      real CUMDEG(MDEG)
      logical LGLTVL(MTVL,MTVL)
      equivalence (LGLTVL,IGLTVL)
      data CUMDEG/0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./
      data IGLTVL/ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     :             0,-1,-1, 0,-1, 0,-1,-1, 0, 0, 0,
     :             0,-1, 0,-1,-1, 0,-1, 0, 0, 0, 0,
     :             0, 0,-1,-1,-1, 0, 0, 0, 0,-1, 0,
     :             0,-1,-1,-1,-1, 0, 0, 0,-1,-1, 0,
     :             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     :             0,-1,-1, 0, 0, 0,-1,-1,-1,-1, 0,
     :             0,-1, 0, 0, 0, 0,-1,-1,-1, 0, 0,
     :             0, 0, 0, 0,-1, 0,-1,-1, 0,-1, 0,
     :             0, 0, 0,-1,-1, 0,-1, 0,-1,-1, 0,
     :             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0/
      data ISCHED/1,2,3,4,5,6,7,8,9,10,11,12/
      data HUGE/10000000.0/
      call SHUFLE(ISCHED,MDEG)
      CMIN = HUGE
      do (J=1,MDEG)
        JDEG = ISCHED(J)
        JTVL = JDEG - IDEG
        if (JTVL.lt.0) JTVL = JTVL + 12
        if (JTVL.gt.0 .and. LGLTVL(JTVL,ITVL)) then
          C = CUMDEG(JDEG)
          if (C.lt.CMIN) then
            CMIN = C
            LDEG = JDEG
            LTVL = JTVL
          end if
        end if
      repeat
      IDEG = LDEG
      ITVL = LTVL
      CUMDEG(IDEG) = CMIN + DUR
      return
      end
```

thirty-second notes (lines 29-33).

The remaining symbols of DEMO6 adhere to the following mnemonic 'roots' corresponding to attributes of notes:

1. PER - average period between attacks (equivalently, average tempo). Each period divides into two components: the duration of a note is indicated by the mnemonic DUR, while the remaining silence is indicated by the mnemonic GAP.

2. PRO - proportion between maximum and minimum periods.

3. ART - articulation; the subsidiary mnemonic roots ALW and AHG correspond to lower and upper limits, respectively, on the range of possible articulations.

4. REG - register, expressed as the lowest pitch in a 12-semitone gamut. The subsidiary mnemonic roots RLW and RHG correspond to lower and upper limits, respectively, on the range of possible lowest pitches.

5. DEG - degree of the chromatic scale.

6. TVL - interval in semitones.

The graphic information depicted in Figure 8-11 corresponds to the numeric information supplied in the DATA statements (lines 11-23) at the head of program DEMO6. Each array whose name begins with BEG holds the value associated with an attribute at the beginning of the segment; its counterpart whose name begins with END holds the evolutionary 'target'. Before computing any specific evolutionary values, DEMO6 calculates a single interpolating factor F (line 48) which holds through an iteration.

8.3.2.1 Rhythm - DEMO6 composes periods between consecutive attacks using the library function RANX (line 54). The library function EVEXP provides exponentially evolving values for both the average period (line 50) and the ratio between minimum and maximum periods (line 52).

In order to select a duration, the library function EVLIN first computes linearly evolving limits for a region of uniform probability ranging at most from 0.0 to 4.0 (lines 56-57). These limits are then employed (line 58) to choose one of the four types of articulation listed above.

8.3.2.2 Pitches - Selection of register for a note uses the strategy of the library function EVRAN. Function EVLIN first computes linearly evolving limits for a region of uniform probability ranging at most from E3 to Ab5 (lines 77-78). These limits are then employed (by line 79) to select the lowest note in a one-octave gamut. Subroutine DEGREE then provides a chromatic degree which (line 81) subroutine WNOTE places within this gamut (line 83). DEGREE employs the methods of the library subroutine HEUR (heading 7.1) to select chromatic degrees subject to the stylistic matrix illustrated in Figure 8-12.

8.4 NOTES

1. The SIN function requires that phases be expressed in radians. To convert from degrees to radians, multiply by 0.0174533.

2. The triplets nested under the septuplet brackets in the first and third systems of Figure 8-8 result from manual interventions by the composer.

8.5 RECOMMENDED READING

Mathews, Max, with F. Richard Moore. "GROOVE: A program to compose, store, and edit functions of time", _Communications of the Association for Computing Machinery_, volume 13, number 12 (December 1970), page 715.

Tenney, James. "Computer music experiences, 1961-1964", _Electronic Music Reports_, volume 1, number 1 (1969) page 23.