Rising, Falling

For solo violin

Symbolic Composer Score Files annotated by Phil Legard

Nigel Morgan

Rising, Falling

For solo violin

Nigel Morgan

This short study provides an opportunity to show how *Symbolic Composer* may be used to build a kind of improvising machine. Such a machine brings together a number of functions that act successively on a small pitch motif (a sequence from the Slonimsky Thesaurus) to generate musical phrases containing pitch, rhythm, dynamics and articulation material.

There are four score-files that generate the music for *Rising, Falling*. Each file presents a function 'machine' that is gradually added to and/or customized to produce different but evolving outcomes. The first two files generate rising phrases, the last two falling phrases.

The code in the four score-files that make up this composition has been annotated and presented alongside transcriptions of the original midi-file output with extracts from the final score. This demonstrates how sometimes the compiled output is just the first step towards a musical composition, whilst at other times may contain all the elements needed to make an exact representation in a musical score.

Despite the sophistication of score-writing software the notated extracts that accompanying the score-files are a reminder of how far such applications have to go before being able to accurately interpret midi-file data into scores fit for human performers to play from.

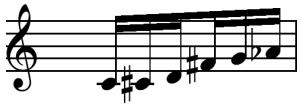
In an appendix an example is given of how to reconfigure a score-file so that musical phrases are organised into discrete zones (like extended bars). This can make the interpretation from a midi-file into a score-display more efficient and the representation more accurate.

Rising, Falling is a one of series of compositions that explore the idea proposed by Sam Richards that the essence of melody is 'rising, falling, hovering'.

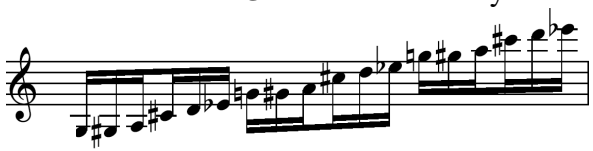
```
;;; Rising, Falling (:1) . . . for solo violin

;; functions

(defun symbol-trim-r (trim-value symbol-list)
   (reverse (symbol-trim trim-value (reverse symbol-list))))

; (symbol-trim-r 8 '(-m -l -k -g -f -e a b c g h i m n o s t u))
; > (h i m n o s t u)


;; material

(setq material '(a b c g h i)) ; Slonimsky Pattern 05

(setq value 8)

(setq range (gen-process '(symbol-transpose x y) '(-12 0 12) material))
; (-m -l -k -g -f -e a b c g h i m n o s t u)


(setq variants
              (gen-collect 0.491 value :list
                     '(symbol-trim
                            (pick1 nil
                                   (mapcar
                                    (function (lambda (x) (- 18 x)))
                                    '(0 0 0 1 1 2 2 6 7 8)))
                            (symbol-transpose
                             (pick1 nil '( 0 0 1 2 2 6 7 8))
                                (filter-delete
                        (pick-rnd nil :content
                                  (get-random 2 5) range) range))))
```

An extension of <symbol-trim>. This function trims all of the symbols from a list, with the exception of a specified number of notes from the end of the list.

Slonimsky Pattern 5:



This pattern is transposed by an octave in either direction, creating an ascending figure, upon which the composition will be based, shown here in the G chromatic tonality:



<gen-collect> is executed 8 times (*value*). Each time the material generated above is transposed between 0 and 8 semitones. 2-5 values are also chosen and filtered out, replaced with rests (=). Finally the sequence is trimmed to between 10 and 18 symbols in length. For example: (-k -j -i = -d -c c = e i j k o p q u =)
This is the original material trimmed to 17 characters, transposed by 2 and with –e and v filtered out.

```
(setq variants-1
      (flatten (mapcar
                (function (lambda (x) (e-insert '= 0 x)))
                (do-section (gen-template 0.51 1 1 value)
                            '(symbol-trim-r (get-random 5 10) x)
                            variants))))
```

<gen-template> creates a template which will further trim certain lists within the collection *variants* to the last 5-10 symbols. For each list a rest (=) is added to the start. The sequence in the last example may become: `(= j k o p q u =)` Therefore a collection of material is arrived at consisting of phrases of varying degrees of fragmentation, divided by rests, as illustrated in the score output at the end of this section.

```
(def-neuron len-gen
 (all-in 1 '(b c)  -5 12) '1/32
 (all-in 1 '(= a) -12 12) '1/16
 (all-in 1 '(c g) -12 12) '1/16
 (all-in 1 '(a =) -12 12) '1/4
 (otherwise  '1/8)
)

(setq lens (run-neuron 'len-gen variants-1))

;; score

(def-tonality
 violin (activate-tonality (chromatic g 5))
)

(def-symbol
 violin variants-1
)

(def-length
 violin lens
)

(def-velocity
 violin (vector-round 64 96 (gen-noise-white (length variants-1)))
)
```

A neural processor is set up to derive a scheme of note-lengths by matching patterns within the symbolic data. For example if the pattern (b c) – and by extension (c d) (-d –c) (a b) etc. – is found then a value of 1/32 is used for the duration of that note (the b in the case of (b c)). If a pair of notes do not match the patterns defined then a value of 1/8 is used.

The dynamic scheme for each movement of *Rising*, *Falling* is created from samples of white noise, rounded to velocity values between 64 and 96.

```
(def-zone
 violin (make-zone (get-lengths-of 'violin))
)

(def-channel
 violin 1
)

(def-program gm-sound-set
 violin violin
)

(def-tempo 72)

(compile-instrument-p "ccl;output:" "rising1"
  violin
)
```

Rising, Falling I

Original Output

4

7

Final Score

♩ = 72 **rhapsodic**

9

mf *sul tasto* *tasto* *mp*

12

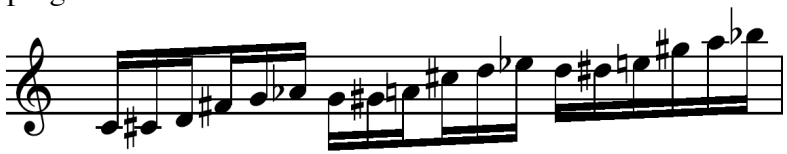
mf *mp* *mercurial (tasto)* (uneven vibrato) *mf* *f*

17

meno *mf* *mf* *mp* *echo*

```
;;; Rising, Falling (:2) . . . for solo violin

(setq material '(a b c g h i)) ; Slonimsky Pattern 05

(setq range (gen-process '(symbol-transpose x y) '(-7 0 7) material))
; (-h -g -f -b a b a b c g h i h i j n o p)

(setq value 12)

(setq variants
              (gen-collect 0.491 value :list
                      '(symbol-trim
                             (pick1 nil
                                    (mapcar
                                     (function (lambda (x) (- 18 x)))
                                     '(0 0 0 1 1 2 2 6 7 8)))
                             (symbol-transpose
                              (pick1 nil '( 0 0 1 2 2 6 7 8))
                                 (filter-delete
                          (pick-rnd nil :content
                                   (get-random 2 5) range) range)))))

(setq variants-1
      (flatten (mapcar
                (function (lambda (x) (e-insert '= 0 x)))
                (do-section (gen-template 0.51 1 1 value)
                            '(symbol-trim-r (get-random 5 10) x)
                            variants))))

(def-neuron sym-rpt
  (all-in 1 '(a =) -12 12)
  (symbol-repeat (get-random 2 7) (list (in 1 0)))
  (all-in 1 '(c g) -5 0)
          (symbol-repeat (get-random 3 5) (list (in 1 0)))
  (otherwise (in 1 0))
)

(setq variants-2 (run-neuron 'sym-rpt variants-1))
```

The techniques used in the subsequent sections of *Rising, Falling* elaborate upon the initial code. In this section a rising phrase is created in a similar way to movement 1, but with a smaller range (transposed a 5th each way instead of an octave.) This produces a raga-like progression:



One of the characteristics of this section is the generation of repeating notes. In a similar way to the rhythmic structure in the previous section, a neural processor is used to create random numbers of repeats for symbols matching a given sequence. So that a sequence such as `(c g = a)` may become `(c c c c g g g = a)`.

```
(def-neuron len-gen
 (all-in 1 '(b c)  -5 5) '1/8
 (all-in 1 '(c g) -12 0) '1/4
 (otherwise  '1/16)
)

(setq lens (run-neuron 'len-gen variants-2))

;; score
(def-tonality
 violin (activate-tonality (chromatic g 5))
)

(def-symbol
 violin variants-2
)

(def-length
 violin lens
)

(def-velocity
 violin (vector-round 48 74 (gen-noise-white (length variants-2)))
)

(def-zone
 violin (make-zone (get-lengths-of 'violin))
)

(def-channel
 violin 1
)

(def-program gm-sound-set
 violin violin
)

(def-tempo 110)

(compile-instrument-p "ccl;output:" "rising2"
  violin)
```

The dance-like rhythm of this section is created by using fewer rhythmic possibilities than in the rhaphsodic first section. The shape of the material being used, with its intervals of the minor second occurring at the points at which the original Slonimsky series was transposed increases the probability of repeated notes occurring.

Rising, Falling II

Original Output

3

6

9

11

Final Score

13

♩= 110 **gently, dance like**

mf *mp* *p*

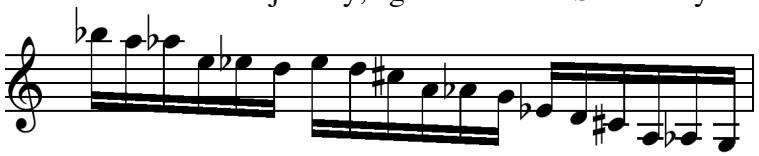
19

(pizz.)

mf *mf*

```
;;; Rising, Falling (:3) . . for solo violin

;; functions

(defun symbol-trim-r (trim-value symbol-list)
   (reverse (symbol-trim trim-value (reverse symbol-list))))
; (symbol-trim-r 8 '(-m -l -k -g -f -e a b c g h i m n o s t u))
; > (h i m n o s t u)

;; material

(setq material '(a b c g h i)) ; Slonimsky Pattern 05
(setq material-r '(m i h g c b))

(setq value 8)

(setq range (gen-process '(symbol-transpose x y) '(7 0 -12) material-r))

; (t p o n j i m i h g c b a -e -f -g -k -l)

(setq variants
              (gen-collect 0.4915 value :list
                      '(symbol-trim
                             (pick1 nil
                                    (mapcar
                                     (function (lambda (x) (- 18 x)))
                                     '(0 0 0 1 1 2 2 6 7 8)))
                             (symbol-transpose
                              (pick1 nil '(0 0 1 2 2 6 7 8))
                                 (filter-delete
                          (pick-rnd nil :content
                                   (get-random 2 5) range) range)))))

(setq variants-1
      (flatten (mapcar
                (function (lambda (x) (e-insert '= 0 x)))
                (do-section (gen-template 0.51 1 1 value)
                            '(symbol-trim-r (get-random 5 10) x)
                            variants))))
```

This section is a variation on the first, but using a phrase with a downward trajectory, again based on Slonimsky 5:



```
(def-neuron len-gen
 (all-in 1 '(c b)  -5 12) '1/16
 (all-in 1 '(= m) -12 12) '1/8
 (all-in 1 '(g c) -12 12) '1/8
 (all-in 1 '(m =) -12 12) '1/4
 (otherwise  '1/32)
)

(setq lens (run-neuron 'len-gen variants-1))

;; score
(def-tonality
 violin (activate-tonality (chromatic g 5)))

(def-symbol
 violin variants-1)

(def-length
 violin lens)

(def-velocity
 violin (vector-round 32 72 (gen-noise-white (length variants-1))))

(def-zone
 violin (make-zone (get-lengths-of 'violin)))

(def-channel
 violin 1)

(def-program gm-sound-set
 violin violin)

(def-tempo 64)

(compile-instrument-p "ccl;output:" "rising3"
  violin)
```

Rising, Falling III

Original Output

4

7

Final Score

♩= 64 **rhapsodic, but more deliberation**

9

naturale

mp

p

mp

pp

p

15

sul ponticello

poco a poco normale

21

mp

sul ponticello

poco a poco naturale

poco rit. . .

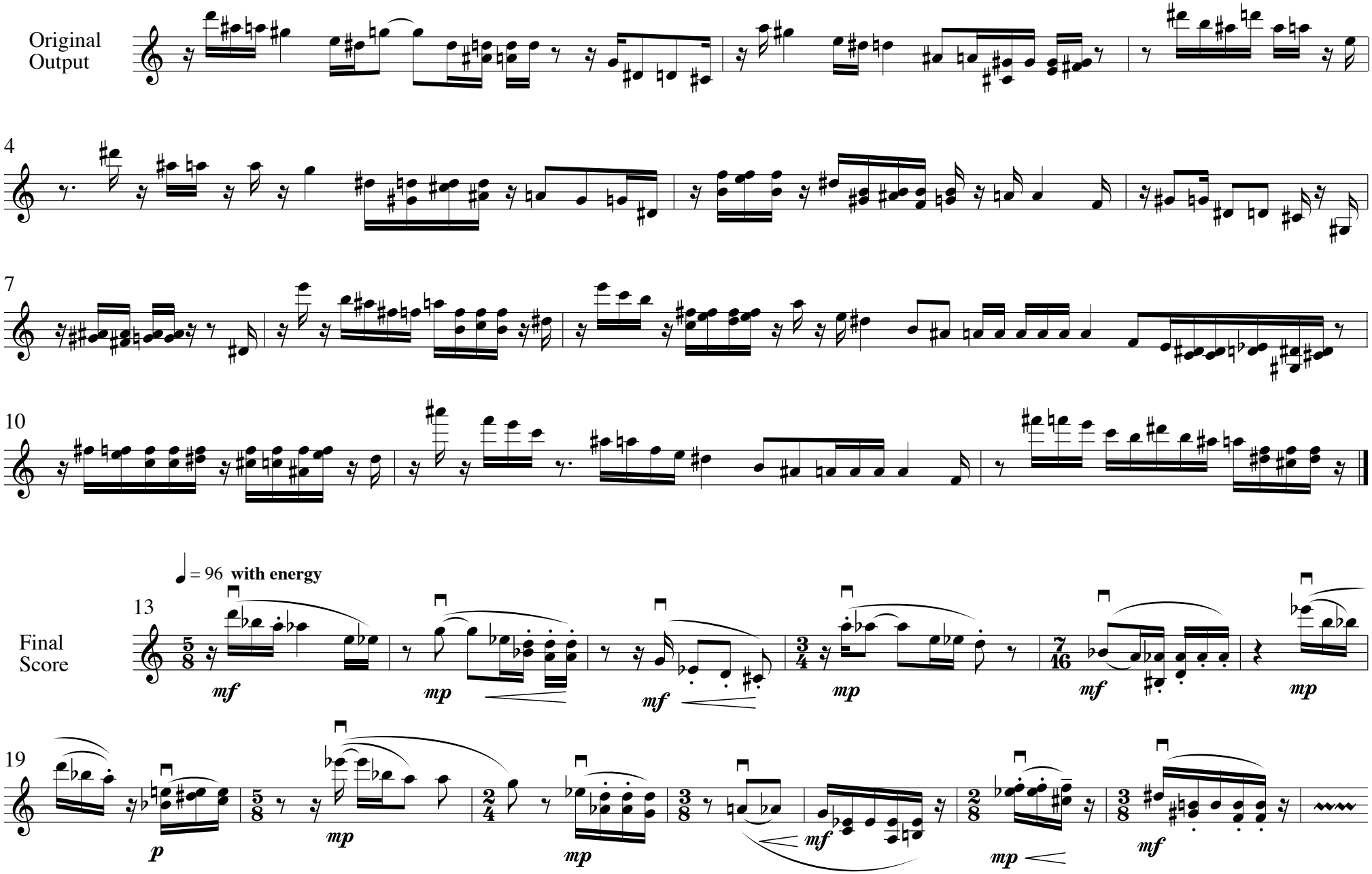
```
;;; Rising, Falling (:4) . . . for solo violin

(setq material '(a b c g h i)) ; Slonimsky Pattern 05
(setq material-r '(m i h g c b))

(setq range (gen-process '(symbol-transpose x y) '(7 0 -12) material-r))
; (t p o n j i m i h g c b a -e -f -g -k -l)

(setq value 12)

(setq variants
              (gen-collect 0.4915 value :list
                      '(symbol-trim
                             (pick1 nil
                                    (mapcar
                                     (function (lambda (x) (- 18 x)))
                                     '(0 0 0 1 1 2 2 6 7 8)))
                             (symbol-transpose
                              (pick1 nil '( 0 0 1 2 2 6 7 8))
                                 (filter-delete
                          (pick-rnd nil :content
                                   (get-random 2 7) range) range)))))

(setq variants-1
      (flatten (mapcar
                (function (lambda (x) (e-insert '= 0 x)))
                (do-section (gen-template 0.51 1 1 value)
                            '(symbol-trim-r (get-random 5 10) x)
                            variants))))


(def-neuron sym-rpt
  (all-in 1 '(a =) -5 12)
  (randomize-harmony nil 'mix -7 0
  (symbol-repeat (get-random 3 5) (list (in 1 0))))
  (all-in 1 '(g c) -5 0)
          (symbol-repeat (get-random 2 7) (list (in 1 0)))
  (otherwise (in 1 0))
)
```

This section is an elaboration on section 2, but uses the same downward phrase as the preceding section.

This section makes use of chords within the repeated notes by employing the <randomize-harmony> function. This creates a second note 0 to 7 semitones below the note in question. For each repetition of a note the harmony will change. A harmonising value of 0 is considered as a unison and discarded by the midi output. Therefore (c =) may become: ((c –5 c) (c –6 c) (c 0 c)).

```
(setq variants-2 (run-neuron 'sym-rpt variants-1))

(def-neuron len-gen
 (all-in 1 '(c b)  -7 5) '1/8
 (all-in 1 '(g c) -5 7) '1/4
 (otherwise  '1/16)
)

(setq lens (run-neuron 'len-gen variants-2))

;; score
(def-tonality
 violin (activate-tonality (chromatic g 5)))

(def-symbol
 violin variants-2)

(def-length
 violin lens)

(def-velocity
 violin (vector-round 96 52 (gen-noise-white (length variants-2))))

(def-zone
 violin (make-zone (get-lengths-of 'violin)))

(def-channel
 violin 1)

(def-program gm-sound-set
 violin violin)

(def-tempo 110)

(compile-instrument-p "ccl;output:" "rising2-ri"
  violin)
```

Rising, Falling IV



Appendix – Updated version of Rising, Falling I

This is an updated version of the score for *Rising, Falling I*. The data streams output by the original code are all presented as a single list of symbols, velocities or note-lengths. This makes interpreting the output in a sequencer or score-writer a difficult procedure. It is, therefore, better to divide this output into its constituent phrases. The method of deriving 'zone-lengths' (analogous to time-signatures) for each phrase is detailed here.

```
;;; Rising, Falling (:1) . . . for solo violin
;; functions

(defun symbol-trim-r (trim-value symbol-list)
   (reverse (symbol-trim trim-value (reverse symbol-list))))

; (symbol-trim-r 8 '(-m -l -k -g -f -e a b c g h i m n o s t u))
; > (h i m n o s t u)

;; material

(setq material '(a b c g h i)) ; Slonimsky Pattern 05

(setq value 8)

(setq range (gen-process '(symbol-transpose x y) '(-12 0 12) material))

; (-m -l -k -g -f -e a b c g h i m n o s t u)

(setq variants
              (gen-collect 0.491 value :list
                      '(symbol-trim
                             (pick1 nil
                                    (mapcar
                                     (function (lambda (x) (- 18 x)))
                                     '(0 0 0 1 1 2 2 6 7 8)))
                             (symbol-transpose
                              (pick1 nil '( 0 0 1 2 2 6 7 8))
                                 (filter-delete
                          (pick-rnd nil :content
                                   (get-random 2 5) range) range)))))
```

```
(setq variants-1
      (do-section :all '(e-insert '= 0 x)
                (do-section (gen-template 0.51 1 1 value)
                            '(symbol-trim-r (get-random 5 10) x)
                            variants)))
```

In the original code, v*ariants-1* was flattened to a single list. In this version the lists are preserved, resulting in:

```
((= -k -j -i = -d -c c = e i j k o p q u =) (= =
i = k o p q u) (= s t u y z [) (= -g –f …
```

as opposed to:

```
(= -k -j -i = -d -c c = e i j k o p q u = = = i =
k o p q u = s t u y z [ = -g –f …
```

```
(def-neuron len-gen
 (all-in 1 '(b c)  -5 12) '1/32
 (all-in 1 '(= a) -12 12) '1/16
 (all-in 1 '(c g) -12 12) '1/16
 (all-in 1 '(a =) -12 12) '1/4
 (otherwise  '1/8)
)

(setq lens (do-section :all '(run-neuron 'len-gen x) variants-1))

;; score

(def-tonality
 violin (activate-tonality (chromatic g 5))
)

(def-symbol
 violin variants-1
)


(def-length
 violin lens
)

(def-velocity
 violin (do-section '(= x = x) '(change-length :sub 30 x)  (symbol-divide (mapcar 'length variants-1) nil nil
   (vector-to-list (vector-round 64 96 (gen-noise-white (length (flatten variants-1)) nil 0.1)))))
)
```

In order to process each list in the collection of *variants-l* the <do-section> is used. For each list in the collection <do-section> will execute a function, in this case <run-neuron>. The output from this routine will have the same structure as *variants-1*.

```
(def-zone
 violin (make-zone (get-lengths-of 'violin) :ratio)
)

(def-channel
 violin 1
)

(def-program gm-sound-set
 violin violin
)

(def-tempo 72)

(compile-instrument-p "ccl;output:" "rising"
  violin
)
```

When used with the :ratio flag <make-zone> will return bar-lengths from a set of note-length data. In this case the output lengths from each phrase are:

```
59/32  17/16  7/8  59/32  15/16  3/4  7/8  23/16
```

Although impractical from the player's point of view this allows the music to be divided logically within the sequencer or score-writer for further work. The scores used to illustrate this document have been divided in such a way.