Durham University

School of Engineering and Computing Sciences

Microtonal Melody Generator

L4 MEng Research and Development Project

Tim Pearce

Supervisor: Professor A Purvis

28th April 2010

ACKNOWLEDGEMENTS

The author would like to acknowledge Bill Sethares. and Ken Schutte for their MATLAB code contributions. These saved the author many stressful weeks. Thanks goes to John Starrett for his helpful e-mail correspondence during the definition period of the project and also to those that participated in the evaluation survey. Most importantly, the author is grateful to Alan Purvis for his continual input into and interest of the project throughout.

SUMMARY

A program capable of composing short melody phrases has been developed. It distinguishes itself from other work through its microtonal capabilities.

The melody generation process employed initially creates a rhythm structure using a Markov chain, to which pitches are subsequently assigned – information for this chain is collected from a library of melodies. Pitches are selected using a genetic algorithm with fitness function criteria that compare the output melody with compiled distributions from either the aforementioned melody library or else that manually input.

Aside from the microtonal component, novel aspects in this project include: the application of quantifiable dissonance values for any note frequency; a statistical analysis of a group of thirty melodies for significant variables that could be coded as formalisms; and a proposed evaluation scheme where the algorithmic composer's melodies are compared with randomly produced melodies in terms of 'listening enjoyment'.

The statistical analysis revealed the majority of trends/patterns tested to not vary on global trends, with the exception of note occurrence and pitch jump occurrence.

The evaluation was encouraging, revealing the developed system to consistently write more enjoyable melodies than for music randomly produced. However relative to a human standard, the system was still deemed poor and additional functions were discussed. The main highlighted reason for this difference was the pitch contour pattern.

The musical basis for the melody composition used was considered sound and the system was technically successful at carrying out the desired functions. A solid base has been formed with the potential to produce an original system, significant in its field.

PROJECT PLAN

The desire to mechanize an artistic process has always been a controversial pursuit, proving a source of worry for some and a momentous challenge for others. This project will shy away from philosophical or psychological debates and focus on the technological and theoretical aspects of melody generation as an automated process.

The project aims to develop a computer programme that may be used in the process of melody creation. Exactly what will be input & output is yet to be decided, however it is aimed to differentiate itself from similar work by not being restricted to one single musical tuning system.

It is envisaged the project will take on the standard research/implementation/evaluation structure. Initially melody will be decomposed in terms of its sense of order, repetition, tension/release etc. A decisive element of the project will be in formalizing this information into a set of rules and constraints that can be written as computer code. A literature review will help to understand what techniques have been used in the past to generate music and how successfully. These methods will be fully explored and may be taken forward to be implemented in a programme.

Development in a computing language to be chosen will take the form as determined by the choices made (for method, inputs/outputs, scales) and an evaluation will then be carried out to determine the value of the resultant melodies. Finally further required work and possible directions of expansion will be considered.

Email extract with the project supervisor at the project definition stage:

Hi Alan,

…Something I think could be really interesting is writing a microtonal melody generator. I've only come across those using the western system of 12-TET so far, i wonder what kind of music would be produced without those initial constraints? surely a human composer is limited in the realm of an infinite frequency range where a computer isn't?!!...

Thanks, Tim

(see appendix 12.1 for Gantt chart)

CONTENTS

ILLUSTRATIONS

NOMENCLATURE

Symbol	Description	Units
d	Dissonance	-
f_1	Base frequency (of first sine wave)	Hz
f_2	Frequency of second sine wave ($f_1 < f_2$)	Hz
s	Constant	-
d^*	Point of maximum dissonance	-
X^2	Chi-squared statistic	-
O	Observed occurrences	count
E	Expected occurrences	count
r	Number of rows	count
c	Number of columns	count
DOF	Degrees of Freedom	-
n	Number of categories	-
p	Number of parameters used to fit distribution plus one	-
G	G-test statistic	-
T	T-test statistic	-
$\overline{X_1}$	Mean of sample 1	-
s_1^2	Variance of sample one	-
s_2^2	Variance of sample two	-
N_1	Sample size one	count
N_2	Sample size two	count
$score1$	Fitness function (closeness to note distribution)	-
$score2$	Fitness function (closeness to pitch jump distribution)	-
$score3$	Fitness function (closeness to desired tension envelope)	-
$score$	Fitness function (total)	-
$npow$	GA user coefficient (closeness to note distribution)	-
$jpow$	GA user coefficient (closeness to pitch jump distribution)	-
$tpow$	GA user coefficient (closeness to desired tension envelope)	-

ABBREVIATIONS

EMI ..... Experiments in Musical Intelligence

MMG ..... Microtonal Melody Generator

GA ..... Genetic Algorithm

MIDI ..... Musical Instrument Digital Interface

PBR ..... Pitch Bend Range

Synth ..... Synthesiser

$1/16^{th}$ ..... Sixteenth of a bar

TET ..... Tone Equal Temperament

DOF ..... Degrees of Freedom

GUI ..... Graphical User Interface

BPM ..... Beats per Minute

1 INTRODUCTION

1.1 OUTSET IDEALS

Close to the entirety of music heard in Western culture uses a set of note frequencies (pitches) consisting of twelve options per octave (a doubling of frequency). This could be likened to painting with the seven colours traditionally associated with the rainbow, and no shades in between. The thought of being limited to only those 'ROYGBIV' divisions seems ridiculous and yet Western music does just this with music. Microtonal music is that produced using non standard pitches.

This project originally had the broad title of **'Microtonal Music Synthesis'**. After exploring various directions that the project could take, the idea for an **algorithmic composition system** (program to write music) was selected. It was considered that the concept of microtonal music has been restricted in the past because a human composer with a physical instrument is unable to consider an infinite range of frequencies when writing music – scale constraints make it a manageable task. However, the immense power offered by a computer could make this a possibility.

The system developed has been named the **Microtonal Melody Generator (MMG)** and this report presents the research, implementation and evaluation completed for it. The MMG aims to compose short standalone melodies – that is, isolated from musical contexts such as chord sequences.

As will be seen in the literature review, the field of algorithmic composition consists of numerous projects of increasing success levels. As of yet none have demonstrated superiority. These are exciting times for algorithmic composers, as the quality of output music soars higher, applications are starting to be realised, they include: aids to human composers; music creators of infinite length pieces used in places where music tends to grate after repeated listens; and as composers in their own right – redefining how we listen to music.

1.2 REALITY

Reality proved the microtonal ideals to be rather an ambitious goal to take in one step, and it was realised that a number of tuning systems would have to first be understood in compositional terms before theories governing all scales could be developed. Presented here is the development of an algorithmic melody generator that operates principally with the standard western tuning system, and to some degree with microtones. Its design choices reflect the future adoption of microtonal scales.

1.3 REPORT STRUCTURE

The report begins by describing past algorithmic composition systems in the **'literature review'**. **'A musical setting'** then provides a brief musical background for the reader with some topics discussed in rather more detail, followed by justifications for musical decisions made by the MMG. **'Microtonal applicability'** communicates some detail about microtones and how the MMG has been designed to compose with these. **'Theory'** outlines the main engineering techniques used in this report and comes before the chapter **'melody statistics'** – an investigation to discover formal rules governing melody. The programs workings are then described alongside code extracts in **'the program'**. An '**evaluation'** of the MMG is conducted through both a survey and an assessment of the programs functions. Finally **'Discussion'** hypothesises the MMG's achievements and failings.

1.4 THE MICROTONAL MELODY GENERATOR

Before beginning the report, it is beneficial for the reader to have an overview of the MMG's operation. It may be useful to refer back to during some sections.

The program's function is to generate a novel & pleasing two bar melody and save it as a MIDI file.

The programme is written in MATLAB and consists of several '.m' files (found in the appendices). Figure 1 gives a summary of the operation as described in the following paragraphs:

To begin, the user specifies values for a number of parameters for the melody. These include the tuning system and scale to compose in, along with the maximum and minimum allowed pitches. Envelopes for density of notes and tension (how these variables change across the output two bars) are also requested.

The program then creates a structure of notes without assigning pitches to them via a Markov chain – stating when notes should turn on, but not at what pitch. Information for this Markov process is extracted from a collection of melodies (library) of a dance/pop genre. This structure is then adjusted according to the input density envelope.

Assignment of pitches is performed by a genetic algorithm. At full functionality, the pitches are allocated according to a comparison with 'ideal' note occurrence and note pitch jump occurrence distributions (extracted from the library) and also a comparison with the user specified tension envelope. However the number of these criteria actually applied is dependent on the type of tuning system selected.

The programme then saves the melody as a MIDI file.

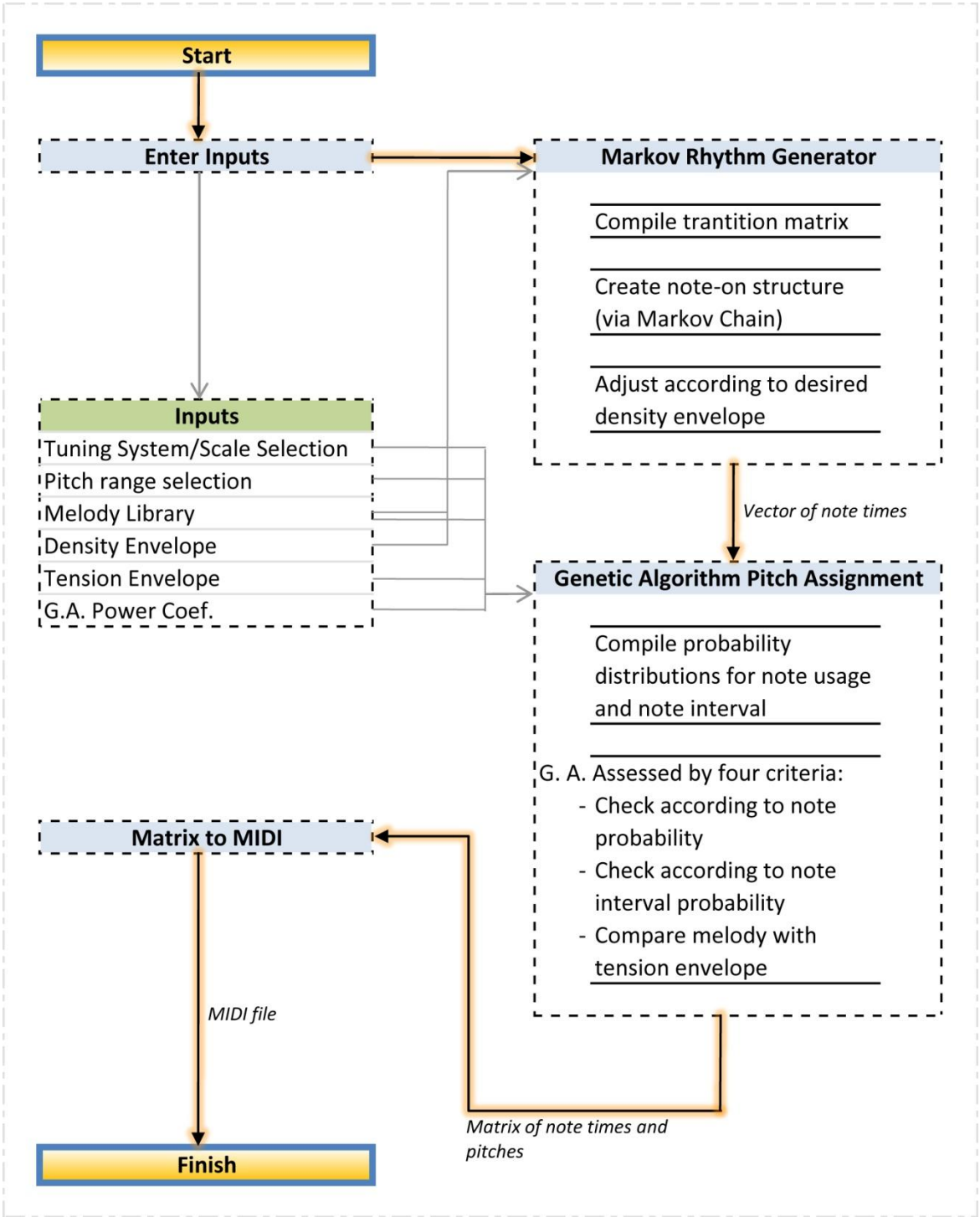


Figure 1 - Program overview

2 LITERATURE REVIEW

2.1 INTRODUCTION

The algorithmic composition of music has proved an attractive field to a large number of composers, academics and hobbyists. Early exploration into the subject includes Mozart's "Musikalisches Würfelspiel" – a technique the great composer devised where pre-written measures of music were arranged in combinations determined by the throw of a dice (Levitt & Schwanauer, 1992). It is only the last fifty years however, that we have seen significant progress in the field. Unsurprisingly, this coincides with the introduction of the computer – its immense processing power and programmable capability making it our best suited tool to date for such a challenge.

Literature linking music and the computer is vast, and much has relevance to algorithmic melody composition, even if not directly (e.g. the sub-field of genre recognition). However, given the space limit of this report, only the content directly relating to the topic of algorithmic composition shall be reviewed. These works shall roughly be categorised by technique and chronology.

Directions and development [sec. 2.2] outlines the different approaches that the field has experimented with since the first computerised attempt at algorithmic composition - Hiller and Isaacson's "Illiac Suite" in 1957. Each technique mentions the major projects that have used them. Some milestone works fall outside these groupings and are discussed under the heading "**additional papers**" [sec. 2.3].

Current state [sec. 2.4] illustrates the field as it is today, including the commercial applications currently being realized.

Literature on **microtonal scales** and music is not addressed in this review. The reader is referred to sec. 4.1 for this thesis' information on microtonal music. The only literature concerning microtonal pitches in relation to algorithmic music that this author found was a mention in passing that a genetic algorithm system could in theory be expanded to compose in microtones (Jacob, 1995). All projects discussed use 12-TET tuning.

A recurring theme in the literature of algorithmic composition is that of **novelty vs. structure**. That is, the balance between making a composing system too rigid and too vague. The rigid system will return musical results of an acceptable nature, but with little variation (novelty). The vague system will be open to so many options that whilst originality comes easily, the results will

tend toward sounding alien and unmusical. Automated composition has a tough task in getting this balance right (Werner, 1998).

Evaluation of the results of past projects is very difficult. The produced music is not always included in the relevant academic paper, and where it is, the question begs as to how randomly the chosen example was selected. It is understandably difficult to get a quantitative analysis of *any* piece of music, and we are forced to settle with authors' concluding comments. These are typically guarded with an indication of hope. For example Gary M. Rader had this to say about his results:

> *"... the program 'composes' at a mediocre level, though at a generally acceptable level for the man on the street..."* (Rader, 1992)

And so, frustratingly there is no absolute way to measure a projects success.

2.2 DIRECTIONS AND DEVELOPMENT

Following are the main approaches that exist to compose music algorithmically.

GENETIC ALGORITHMS

The use of genetic algorithms (see sec. 5.2) has been one of the most explored methods used to create an automated composition system. A set of randomly generated notes undergo "musically meaningful mutation's"(Biles J. A., 1994), such as transposition, inversion or reversal operations. They are then assessed by a fitness function – this being the crucial part of the algorithm – for which two approaches have been used.

One is the use of an **objective fitness function**. David Baker (Baker, 1988) used five criteria to test generated offspring; The level of novelty compared to a melody database (separately assessing tone and rhythm), the balance between stepwise and greater than stepwise note intervals, the existence of stuttering (short or repeated notes), and a comparison with the program's initial input melody (due its improvisation aspect). For another example of this fitness function type, see (Goldberg & Horner, 1991).

Alternatively, a **human** may be used to grade the offspring. This carries the clear advantage of being a completely accurate fitness function – in that a human is also the judge of the final result quality. Good results might be expected but for the practical disadvantage of the time required to listen to all potential candidates. Hence, the population size of each generation must be limited, and the result quality is lowered. Nevertheless, a human has been implemented in many projects. The reader is referred to (Horowitz, 1994) and (Biles J. A., 1994). One suggested way to overcome these problems has been to use a wider audience to evaluate the populations.

The use of a learning based fitness function that becomes able to reproduce human preferences produced disappointing results (Biles, Anderson, & Loggi, 1996). An interesting thought is to create a larger loop with the genetic algorithm – using an entire composing system as the species, rather than just the musical results. (Werner, 1998).

GenJam(Biles J. A., 1994) is a long term project working on the development of (principally) a real time jazz solo improviser, implementing an audio to MIDI converter combined with a genetic algorithm (in its original version), allowing a relationship with a human performer. The fitness function is performed by a human, and each generation size is limited to forty-eight. Biles reported that it takes about ten generations before phrases begin to sound reasonable. He performs gigs alongside GenJam and the music he shares on his website (Biles J. A.) seems very agreeable. He concludes his paper:

> *"...GenJam shows that GAs can be a useful tool for searching a constrained melodic space..."*(Biles J. A., 1994)

Simulated Annealing

Simulated annealing is inspired by the heating and cooling process used for metals to increase the size of their crystals, and reduce their defects. The key to it being successfully applied to the task of music composition is the quality of its goal function test. This test decides whether an altered solution should be accepted or rejected to use for subsequent operations. This makes simulated annealing fundamentally limited by the same issues as for the genetic algorithm.

Iazzetta used an interesting element in their simulated annealing composer (Kon & Iazzetta, 1995), where the user **draws envelopes** for certain parameters of a melody. These include tension which is used in the goal function test. Pitches are assigned different tension levels and the melody notes are then compared to the desired curve.

Markov Models

Markov processes (see sec. 5.1) carry out composition on a note-by-note basis, where the probability of selecting a specific note as the next note is dependent on preceding events. Generally, these events are in terms of the pitch jump magnitude between adjacent notes.

To create the required **transition matrix**, two methods have been used. One is to analyse a library of music, extracting note information and counting say, the number of times a certain note is followed by another note (F.P. Brooks, 1992). The type(s) of music input will clearly affect the produced results. Secondly, the transition matrix may be manually formed, using for example harmonic rules. 'Illiac Suite' famously used this approach (Hiller & Isaacson, 1957).

Once the transition matrix has been generated, this method is very fast at producing music, and hence useful in real-time applications. Originality is introduced by the random element, though the model will not often produce say, an unheard pitch step, and so its novelty value is limited (McCormack, 1996). Should a Markov model be used to create pieces longer than a phrase, there is the disadvantage of its lack of inherent structure (Werner, 1998).

Brooks et al. (F.P. Brooks, 1992) conducted musical experiments to determine the effect order had on the music composed via a Markov model. He found first and second order to produce uncomfortably random music, whilst orders of seven and above reproduce material in the library almost exactly. Orders between these values were therefore recommended.

"Modelling and generating music using multiple viewpoints"(Conklin & McCleary, 1988) used several Markov models with each looking at a different aspect of the music and combining these to generate notes. Some of the viewpoints used include the pitch interval between notes, the absolute pitch of the notes and duration of a note.

SONIFICATION OF RANDOM DATA

The perception that music is simply patterns of sound has invited attempts to transform data sets believed to have an underlying order into music. Often the data set used is inventive, such as John Cage's experiment (Alpern, 1995) where chess moves triggered sound events, or Ransbeek et al. who sonified stock-market data (Guedes & Van Ransbeeck, 2009).

2.3 ADDITIONAL WORKS

The following do not fit directly in the previous classes, but are worth attention nonetheless.

XENAKIS

Iannis Xenakis is a hugely renowned (and controversial) figure in the field of computer music. His book: Formalized music, thoughts and mathematics in music (Xenakis, 1971), has been cited by much subsequent work whilst also collecting critics. His writings relate many of his philosophical ideas about what music should be. They largely concern a "higher order of aesthetics...strongly influenced by the aesthetics of mathematics" (Alpern, 1995). Most relevant to this thesis are the stochastic and probabilistic methods he employed in his compositions, though these were used as *aids* to composition and still required much human input.

EXPERIMENTS IN MUSICAL INTELLIGENCE - COPE

Cope began experiments in musical intelligence (EMI) in 1981, and is still developing it. The project is unique in that it aims to compose new music not only in a set genre, but actually of a single musician's style within that genre.

> *"...My idea was that every work of music contains a set of instructions for creating different but highly related replications of itself. These instructions, interpreted correctly, can lead to interesting discoveries about musical structure as well as, hopefully, create new instances of stylistically-faithful music..."* (Cope, Experiments in musical intelligence website)

"A computer model of music composition" (Cope, 1992) explains the techniques used to achieve this. First, a library of works by a particular composer is required. These are analysed for common trends and this information guides a recombination process to create a new piece of music with traits similar to those found in the library. Analysis takes the form of pattern matching where rhythm and pitch are scanned separately and then together in a search for patterns recurrent in individual pieces. Those patterns found more commonly will more likely be replicated in the new composition.

Sound excerpts found on Copes website demonstrate an impressive system, that to this author's ear seem passable as the target composers own. Critics suggest that Copes system is not truly creative (Conklin, 2003) labelling it "style imitation".

2.4 CURRENT STATE

The last decade has seen major development in automated music composition. A larger community than ever before are currently working on the design of new systems and the first commercial applications are just being realised. The internet has helped this movement both through freedom of information on websites such as Algorithmic.net(Ariza) – an online database of automated composition projects – and also to aid software releases (Eno & Chilvers).

The new wave of programmers tends to be of the younger generation and they focus efforts on their genres of interest – namely electronica. A disadvantage (if there can be such a thing) of the recent influx of interest is a decline in the proportion of projects being written up formally. Often an individual will create a programme, possibly with some sort of internet release, but the only explanation to its workings will be a short write up on the owner's website for example;(AutoGam.com).

As the technological level of algorithmic composition rises, new terms have been given to music made in this way. **Generative music** describes ever-changing music (generally multi voiced) created by a system. **Interactive systems** are those utilising high level feedback parameters influencing what kind of music is generated as it is being heard (e.g. a computer game changing the tempo or 'mood' of music with the players level advancement).

Applications and Commercial Projects

Currently, every possible application for generative music is being explored. Suggested purposes include; video game music, film soundtracks and shopping mall background music. These tend to be situations where music can become dull and grating after numerous listens, and generic enough to be implemented automatically through control of the correct parameters. Perhaps most significantly, generative music as an alternative attitude toward listening is being considered.

> *"...I really think it is possible that our grandchildren will look at us in wonder and say: "You mean you used to listen to exactly the same thing over and over again?"..."* (Eno & Chilvers)

Indeed, there have been albums released along with the required software providing infinite length listening experiences, such as the 2009 album Laconicism by Batuhan Bozkurt. He describes his music as "organised sound... not finalized onto a static medium" (Bozkurt).

MadWaves released the first generative music hardware – the MadPlayer – in 2002(Dr.M, 2007). It is essentially an MP3 player with generative capabilities allowing the user some control over the tracks it produces. The user selects a genre and can perform basic manipulations on MIDI lanes of the song. This product has been discontinued for reasons unknown.

Bloom (Eno & Chilvers) is a downloadable application for the iPhone and iPod touch which produces generative ambient music. It interlinks the audio output with visual displays.

Noatikl is a MIDI based generative music engine that composes in real time. The user selects what 'rules' to use for each voice in a piece (e.g. selecting the scale, or the degree of harmonisation with other voices) which guides the process as poetically explained:

> *"...Each time a Noatikl piece plays it will have certain boundaries set by the author, outside which the music will not go. The result is that the music created by Noatikl can be different every time you hear it. That keeps the music fresh and interesting. This is the difference between looking at a photograph of a river, and watching a real live river flowing in nature..."* (Noatikil-Website)

A further interesting aspect of Noatikl is its use as a plug-in to popular electronic music production software such as Ableton or Cakewalk – in this mode Noatikl triggers MIDI events in the parent software.

Specific Recent Software

The melody generation process of three of the most recent projects is now briefly discussed.

Infno (Collins, 2008) implements a 'minimal cost' strategy. The melody is added after a chord sequence has been determined and is composed on a note by note basis. Every pitch in a two octave range has an associated cost calculated via a number of criteria such as; number of note repeats, distance from previous note (smaller the better) and variance from ideal contour for the part. The optimal note sequence (that minimises total 'cost') is selected after forward and backward tracking or alternatively a rapid setting simply chooses the lowest note cost at each stage.

LEMu (Rene Wooller, 2007) concerns itself with the morphing of two melodies – that is, beginning with one input melody, and gradually altering it over a transition period until it matches the second input melody. Three morphing methods are tested; probabilistic (converts the source and target into probability matrices and plays notes according to the last run through), evolutionary (using a genetic algorithm) and parametric morphing (converting both melodies to envelopes and slowly altering the output envelope).

Deviate (Keith, 2009) takes a statistical approach to melody composition. Manually compiled data maps of (principally) pitch and timing are used to select notes on a probability basis. Interestingly, the pitch map used was based on the key profiles obtained by Krumhansl and Kessler (C. L. Krumhansl, 1982) who conducted experiments, using twenty-five students to determine the note changes most favoured for a given harmonic context.

3 A MUSICAL SETTING

This chapter's first purpose is to provide basic musical knowledge necessary to understand the MMG. The MIDI system and dissonance are two topics discussed in a (necessarily) higher level of detail. Leading on are a series of justifications of certain musical decisions made for the MMG.

3.1 BASICS

This section describes musical terminology/techniques used in this project.

Single Melodic Line – As generated by the MMG. This is a collection of notes, perceived as a continuous entity, which may not overlap each other in time – i.e. the music which could be produced by a single instrument capable of playing only one note at a time (e.g. trumpet).

Tuning System – In choosing the frequency for a musical note, a continuous range within the audible limits of human hearing is available. A tuning system narrows this continuous range into a selection of discrete frequency values. There are several main classes of tuning system (sec. 4.1).

Scale – A musical scale is a collection of notes belonging to the same tuning system (presented in ascending or descending order). In mainstream western music, a melody is often made up only of those notes in a chosen scale (most commonly the **major** or **minor** scale).

Key – Generally, when a musical phrase is said to belong to a certain key (e.g. in the key of C), it means that the music is harmonically centered on this note (known as the **tonic**) – i.e. this tonic note produces minimal dissonance when played alongside the other notes in the phrase. Further to this single letter, a scale may be indicated when describing the key (e.g. **C major**). This suggests that the majority of notes in the phrase may be found in the scale described.

It is possible to alter the tonic of a piece by shifting all notes by a set number of pitch intervals (in equal tempered tuning). For example raising every note in a C major piece by two semitones would produce the piece in D major, the character of the piece would be only slightly affected. Altering the scale (e.g. C major to C minor) changes the feel significantly and has not been done in this project.

Transposition – The aforementioned process of multiplying all notes frequencies in a piece by a set ratio is known as transposition.

Quantisation (time) – This could be seen as a discretisation of the time domain. For example, during musical recordings, small deviations from ideal timing may occur. The process of rounding the start and end of these imperfect notes to a nearest specified time unit (e.g. sixteenth of a bar) is known as quantisation.

3.2 DISSONANCE AS A QUANTIFIABLE PROPERTY

Dissonance is a property describing how multiple sounds interact. Two notes are said to be **dissonant** (high tension) if they produce a disagreeable, inharmonious sound. **Consonance** (low tension) is the opposite of this and results in a smooth sound – when the waveforms of separate sources seem to 'match' and be in harmony such as with a major chord. A balance between dissonant (tension building) and consonant (release) sounds is important in creating a piece with interest.

Rather than using both terms – 'consonance' and 'dissonance' – a consonant sound shall be described as having zero dissonance. It is not essential for notes to occur simultaneously to produce dissonance – if a single melodic line is in a particular key, notes appear dissonant if they clash with the tonic pitch of the phrase.

Sethares (Sethares, 1993) demonstrates how the magnitude of dissonance can be found given two frequencies and their timbres as follows.

The dissonance produced by two pure sine waves is described in Figure 2. Perhaps surprisingly, intervals widely regarded as consonant such as fifth (seven semitones) appear to possess no special dissonance value.

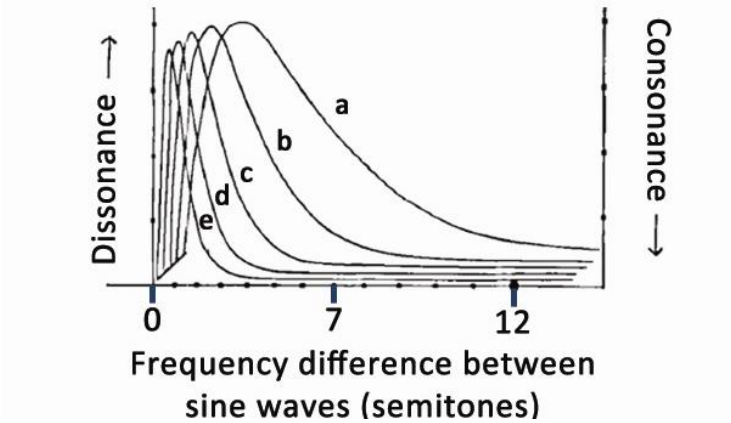


Figure 2 - Dissonance curves for pure sine waves, dissonance scale is arbitrary. Base freq. varies: a) 125 b) 250 c) 500 d) 1000 e) 2000 Hz. Adapted from (Sethares, 1993)

The curves in Figure 2 may be modeled by the following equations (assuming equal amplitudes for both sine waves):

$$\boldsymbol{d(f_1, f_2) = e^{-3.5s(f_2-f_1)} - e^{-5.75s(f_2-f_1)}} \qquad \text{Eqn. 1}$$

With, $$\boldsymbol{s = d^*/(0.021f_1 + 19)} \qquad \text{Eqn. 2}$$

Where; f_1*= base frequency,* f_2*= frequency (*$f_1 < f_2$*),* d^**= point of maximum dissonance (~ 0.24)*

Since musical tones generally consist of a fundamental (base) frequency and a series of harmonics (at integer ratios) above this, Plomp and Levelt suggested that the total dissonance between two complex tones could be seen as a sum of the dissonances of all its sinusoidal components. Figure 3 uses this assumption for two tones with typical timbres.

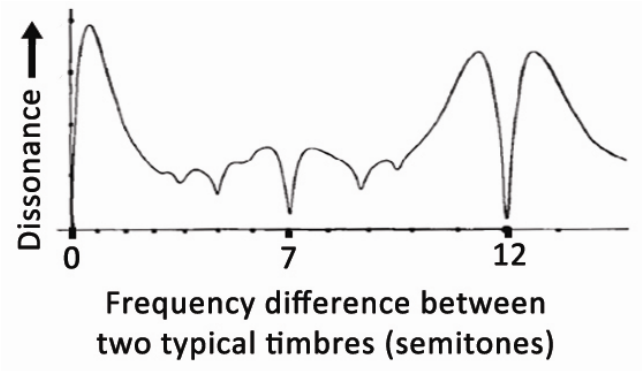


Figure 3 - Dissonance produced by two typical musical timbres. Adapted from(Sethares, 1993)

Dissonant minima are now found in expected locations (such as for the perfect fourth and fifth intervals). The computer code implementing Eqn. 1 and Eqn. 2 for multi-harmonic sounds as modified from Sethares may be found in appendix 12.2.

3.3 MIDI

MIDI (Musical Instrument Digital Interface) is an industry-standard protocol. It is a digital communications language and compatible hardware specification that allows multiple electronic devices to communicate with each other. Rather than transmitting audio signals, event messages are used to trigger operations in devices – for example the pitch and intensity of a note may be described, rather than its actual waveform.

Relevant to this project is the way in which MIDI stores note information and selects pitch.

Basic MIDI File

For each note to be played, a MIDI file (*.mid) must contain the following information (with the range of possible values specified in brackets):

Pitch (0-127)	Velocity (0-127)
Note-on (time)	Channel number (0-15)
Note-off (time)	Track number

Figure 4 shows a typical MIDI editor screenshot of a single track and channel with the previous parameters displayed graphically, followed by an explanation of each.

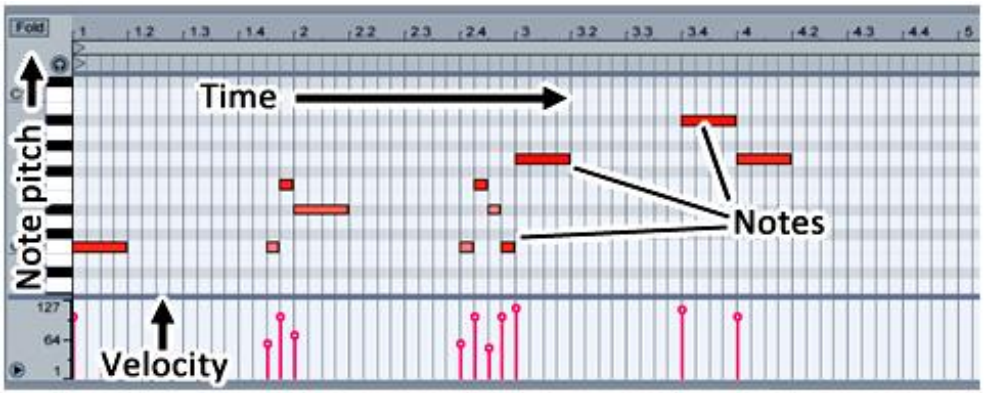


Figure 4 - Visual representation of MIDI note information (screen shot - Ableton Live 7)

PITCH **–** This is assigned as a value from 0-127 where each integer corresponds to a note on the 12-TET scale. So for example 0 = C1, 60 = C5, 61 = C#5, 127 = G9. With the conversion equation:

$$MIDI\ pitch\ =\ 69\ +\ 12\ \times\ log_2\ (desired\ frequency\ Hz\ /\ 440\ Hz) \qquad \text{Eqn. 3}$$

Note the musical reference value of 440Hz (A4) which is assigned a MIDI pitch of 69.

For frequencies in a 12-TET tuning system this formula delivers an integer MIDI note number. Other pitches fill the space between the whole numbers logarithmically.

VELOCITY **–** Determines volume level of the note to be played. A value of 0 corresponds to silence and 127 to maximum. The actual level of sound produced is dependent on the individual device settings. A note with velocity zero may be used to turn off a note of the same pitch previously started.

NOTE ON/OFF **–** MIDI systems communicate in real time by sending messages to the appropriate devices. A MIDI file must therefore store the time that each message is to be sent. MIDI ticks are a length of time per-quarter-note as defined by the user.

Each MIDI event (e.g. note-on) is accompanied by a quantity of these ticks since the last event occurred. When reached, the message is triggered. The actual time (seconds) this becomes depends on settings for both the tempo and ticks per quarter note.

$$Time\ of\ event = Time\ of\ preceeding\ event + Delta\ time \qquad \text{Eqn. 4}$$

$$Delta\ time = event\ ticks \times \frac{60{,}000{,}000}{ticks\ per\ 1/4\ note \times tempo\ (Beats\ Per\ Minute)} \qquad \text{Eqn. 5}$$

CHANNEL/TRACK **–** A MIDI file can store events for separate devices – each requiring the assignment of a separate track. Within a track, up to sixteen (0-15) channels are available. Separate channels allow isolated events to be sent to the same device.

Microtonal Suitability

The MIDI system is inherently suited to 12-TET tuning with integer MIDI notes correlating to frequencies of this system. The MMG however, requires alternative tunings and a number of workarounds to achieve this were considered:

Synth Tuning Tables – The internal tuning tables of a synth may be altered so that given an input MIDI value, a specified frequency is output. As default, these are set with integers corresponding to notes of the 12-TET scale, but this can be changed producing a system where integer notes are sent to the synth, with microtonal scales being output. Unfortunately, both the tuning capabilities and the way in which these tables are altered vary from synth to synth, with no simple universal way of setting them.

Pitch Bend (single-channel) – Pitch bend is a function supported by all synths. It alters the pitch of a note according to two parameters – pitch bend range (PBR) and magnitude of pitch bend. When using one channel to send notes, the amount of pitch bend must be altered for every note to achieve the exact pitch required. The disadvantage of pitch bending is that the amount of MIDI data communicated is increased, which can be a problem for slower hardware. It also denies the use of multiple notes where several notes all pitch bent to different degrees may be required.

Pitch Bend (multi-channel) – Using multiple channels allows each individual channels pitch to be fine tuned by a specified amount (without interfering with the other channels). Once set, notes may be sent in the usual way and by selecting the appropriate channel, microtonal scales are achieved. This method works well for equal tempered tuning systems (see sec.4.1) such as 24-TET where one channel could be raised by half a semitone (a quartertone) enabling all frequencies to be accessed through two channels. It does however, introduce a limit of sixteen notes per octave should the desired tuning system not be equally tempered or else not fall in sync with the 12-TET system.

The MMG implements the **single-channel pitch bending** method. This allows compatibility with any tuning system and with any synthesiser at the expense of added MIDI data flow – which proved manageable with the modern software synthesisers used. The MMG produces single melody lines only, negating added benefit of multi-channel pitch bending.

Pitch Bend Detail

As mentioned, two parameters must be set for a pitch bend. The range of pitch bend determines the maximum and minimum amount the pitch can be altered by and is measured in semitones

from the natural MIDI value. The magnitude of bend is specified by the value of fourteen bits and as such allows values of; 0-16383. Table 1 demonstrates how this affects the pitch produced.

Pitch bend magnitude	Resulting pitch	Example – MIDI pitch for: *Natural note = 60 [C4]* *PBR = 2 semitones*
0	Normal pitch – bending range	58 [A#3]
4096	Normal pitch – ½ bending range	59 [B3]
8192	Unbent (normal pitch)	60 [C4]
12288	Normal pitch + ½ bending range	61 [C#4]
16383	Normal pitch + bending range	62 [D4]

Table 1 - Pitch bending illustrated

Setting the PBR requires either a MIDI message to be sent to the synth, else a manual adjustment by the operator. After extensive attempts to set the PBR via MIDI message, problems were still being encountered and so the decision was made to set the range manually (a simple task on most synths). Typically, the default PBR is (+/-) two semitones and, without any disadvantages to this value, it was selected as the MMG's default.

To convert any frequency (Hz) to its MIDI note number (*m*) and accompanying pitch bend value (*decvalue*), the following code is used (full code in appendix 12.2):

```
m = 69 + 12*log2(f/440);                          (as in Eqn. 3)
difference = round(m) - m;
decvalue = round(8192 - (difference * 4096));
```

The pitch bend value is read in MIDI format as two seven bit numbers.

3.4 JUSTIFICATION OF MUSICAL DECISIONS

Outlined here are the major musical decisions and assumptions made for the MMG.

GENRE CHOICE

Every genre of music (by definition) has a set of common characteristics that it obeys. By selecting a single genre for the MMG to operate in, musical choices are narrowed. The genre chosen was pop/dance because songs are based around short, repetitive, clear musical phrases (minimal composition effort), material (MIDI files) was more readily available for analysis, and both the author and general population are familiar with this (allowing an effective evaluation).

QUANTISATION INTO 1/16TH'S

Quantisation again narrows musical decisions during composition. It is also necessary in the analysis of MIDI files. Sixteenths were chosen following the observation that only four instances in the melody library (out of 709) required finer intervals.

The Independence of Rhythm and Pitch

In the MMG, a melodies rhythm is created and pitches assigned subsequently. This method may seem surprising, but trials performed manually with such a process by the author seemed satisfactory. Much past work also uses this assumption – e.g. (Baker, 1988) and no reasons could be formulated showing it as detrimental.

Ignoring Note Length

A simplification the MMG makes when composing is to assume all notes are short and of equal length, it is envisaged that the user would alter the note lengths manually post-generation. This removes the need for any calculations to be made regarding the ending of notes. When a piece of music has all its notes shortened to the same length, it was found that the piece was still recognisable, seemingly making it a reasonable simplification.

Ignoring Note Velocity (amplitude)

As with length, the velocity of a note is constant for all notes in the MMG's output. It was found the listening experience of existing melodies is largely unaffected by this assumption which simplifies generation.

Ignoring The Octave

There are instances where the MMG treats notes as being equal, regardless of their octave (a doubling of frequency). Figure 5 shows how the dissonance varies for a frequency difference ranging two octaves. Whilst it is not an exact match, the two octaves can be seen as an approximation of each other. The fact that the octave of the tonic note is often ambiguous further justifies the assumption.

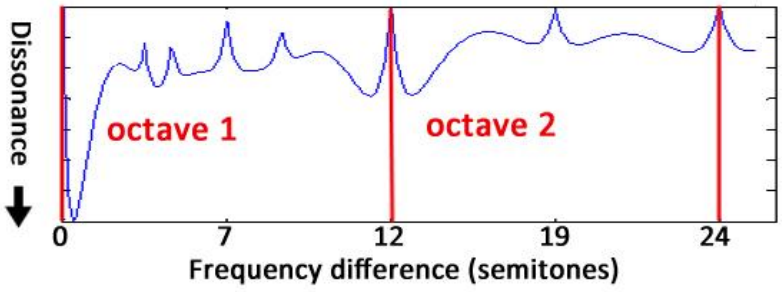


Figure 5 - Dissonance over two octaves

Tension Envelope -as used by (Kon & Iazzetta, 1995).

Tension balance is a recurring theme in melody theory. During development, it was considered that the MMG had sufficient functions to calculate an effective pitch envelope (through the pitch and pitch jump occurrence distributions –sec. 7.2), however tension was not accounted for. After being observed to vary on a case-by-case basis (sec. 6.3) a tension envelope was decided to be offered as a user parameter, included to provide control over the melodies produced

4 MICROTONAL APPLICABILITY

Microtonal music refers to that made when using a tuning system other than the twelve-tone equal tempered system (12-TET) as heard in standard western compositions. Motivation for this project was centered on creating a compositional system able to output music containing microtonal frequencies. This chapter looks at some of the more popular alternative tuning systems and considers the MMG's capability of composing with these.

4.1 MICROTONAL SYSTEMS

12-TET AND CRITICISMS

12-TET splits the octave into twelve pitches, where every consecutive pitch pair has the same frequency ratio (see **Eqn. 6** for note frequency calculations). Criticisms often revolve around the fact that these twelve pitches do not match exactly with pure ratios of frequencies. For example an interval of a major third (C to E) in the 12-TET system has the error as seen in Table 2 when compared to its nearest ideal ratio of 5/4.

Tuning system	Interval	Decimal frequency ratio	Error from perfect ratio	Error relative to base freq. of 261.63 Hz
Just Intonation	5/4	1.25	0	0
12-TET	Major third (4 semitones)	1.25991	0.00991	+ 2.59 Hz

Table 2 - 12-TET major third compared to a perfect frequency ratio

Whilst it appears small, the significance of this error is noticeable to the ear. Dr Robert Smith, writing in 1759, described 12-TET as;

> *"...that inharmonious system of 12 semitones, which produces a harmony extremely coarse and disagreeable…"* (Oosterhoff, 2005)

The system does have positives – for those instruments capable of discrete notes only it offers fair approximations to the main consonant intervals (see Figure 6) without confusing the musician with a staggering number of options. It also offers the advantages shared by all equal tempered systems (to be discussed).

POPULAR MICROTONAL SYSTEMS

JUST INTONATION – This refers to tuning systems that use small whole number ratios of a base frequency as the criteria for frequency selection. An example tuning system using a tonic frequency of (440Hz) is given in Table 3 .

Just frequency ratio	1/1	9/8	5/4	4/3	3/2	5/3	15/8	2/1
Tuning system (Hz)	440	495	550	586.7	660	733	825	880

Table 3 – Just diatonic scale - as found in Indian music

Just intonation is often said to produce a smooth and aesthetically perfect sound, though it has disadvantages; Transposing to a different key requires retuning of the instrument, some just-systems create 'wolf intervals' (severely dissonant intervals) – these are mostly avoided.

Equal Temperament – Here, all note frequencies are defined as multiples of the same basic interval – i.e. any pair of notes separated by the same number of steps will always have the same frequency ratio. Equal temperament offers the advantage that music can be transposed up and down at will without affecting the pieces feel. This is particularly useful for instruments capable of discrete frequencies only.

Generally, equal temperament splits an octave into a set number of steps (given by the prefix e.g. '12'-TET) and for this the following equation is used to calculate frequencies:

$$\boldsymbol{frequency\ (Hz)\ =\ P_a \sqrt[s]{2}^{n-a}} \qquad \text{Eqn. 6}$$

Where; s = no. divisions per octave / n = note number / a = reference note number/ P_a = frequency (Hz) of reference note

Well Temperament **–** This system implements twelve notes per octave with individually assigned frequencies allowing a piece to be played in any key (major or minor, transposed as desired). No 'wolfs' exist, and each key produces a different feel affording the composer more variation than with equal temperament. It is essentially a compromise between equal temperament and just intonation. Not every interval is represented exactly, but neither are the steps between notes equal in value as in equal temperament.

Microtonal Overview

Figure 6 visually displays the priorly discussed tuning systems. It is convinient to plot them against the dissonance curve as described in sec. 3.2.

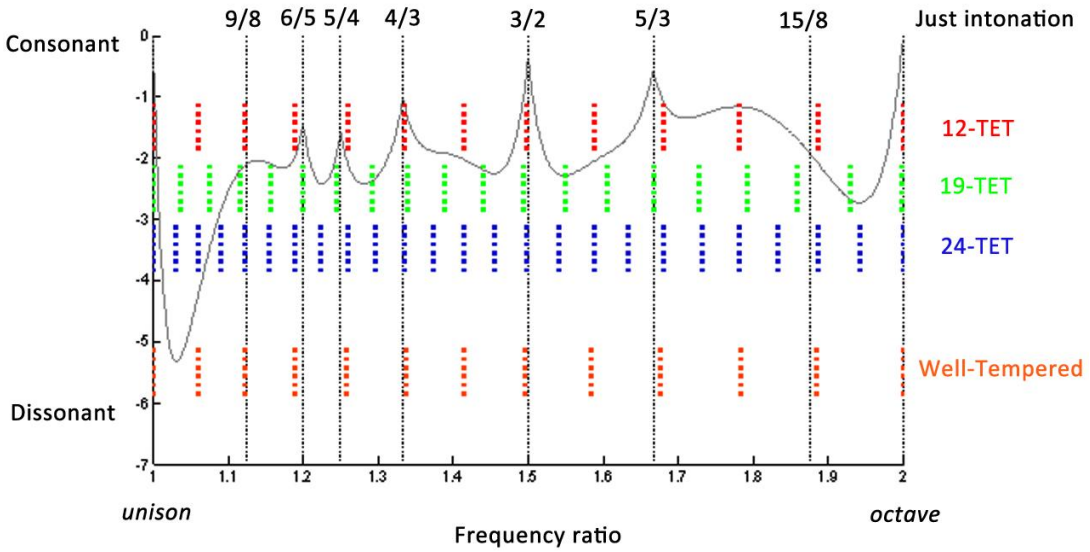


Figure 6 - Comparison of various tuning systems plotted on a dissonance graph for one octave

The diagram reveals how a just intonation system seeks the consonant peaks present in the octave. The magnitude of errors relative to these perfect ratios are visualised for the 12-TET and 19-TET systems, and those extra options provided by 24-TET can be seen. The well-tempered (specifically *'temperament ordinaire'*) system's similarity with 12-TET may be observed along with its slight but significant deviations.

This chapter has not been a detailed analysis of tuning systems, but has aimed to make the reader aware of alternatives (and their advantages) to that which the western ear hears and accepts as fixed. It is important to note that different culture's across the world use systems other than 12-TET as standard. There is a strong feeling that the dominance of a single twelve tone scale unnecessarily restricts the variety of music possible:

> *"Equal temperament was not adopted because it sounded better… …By substituting twelve equally spaced tones for a universe of subtle intervallic relationships, the composers and theorists of the eighteenth and nineteenth centuries effectively painted Western music into a corner from which it has not yet succeeded in extricating itself..."* (The-Just-Intonation-Network, 2008)

4.2 THE MMG AND MICROTONES

Following, the MMG's attributes and drawbacks concerning microtones are discussed. Refer to sec.1.4 for an overview of the program.

DESIGN OF THE MMG

The literature review conducted showed no evidence of an algorithmic composition system that used microtonal scales. This meant that there were no precedents to learn from. The MMG composes with any input tuning system, though the type of system dictates the level of functionality applied during generation.

The major attribute of the MMG with reference to microtonal scales is its **pitch-independent rhythm generator**. The structure of note timings is created using rhythm data from the 12-TET melody library. A significant part of the melody is therefore created before pitches have even been considered.

All '*.m*' files (see sec. 7.3) understand both whole and fractional MIDI note numbers and the program fluently converts between absolute frequencies, fractional MIDI notes and MIDI notes with pitch bend values. Microtones are output with accuracy in the saved MIDI files (using the pitch bend function). The dissonance of any frequency may be calculated and used during pitch assignment when comparing with the desired dissonance envelope.

A weakness of the MMG is its reliance on distributions for note usage and pitch jump occurrence compiled from the 12-TET MIDI library (these distributions are specific to 12-TET). Efforts were made to collect libraries of melodies in other tuning systems, but files of this type proved available. A basic program was written to convert microtonal audio to MIDI files (those available commercially tend to go directly from audio to 12-TET MIDI) though this was not used. Currently, the MMG requires manual input of the distributions when working with microtonal scales. Further to this, the utilisation of the pitch jump distribution when assigning pitches is invalid for unequal-tempered tuning systems where intervals between notes vary for the same number of steps. For this tuning type, the genetic algorithm bypasses the pitch jump part of the fitness function.

The compilation of the occurrence distributions for scales other than 12-TET is pivotal in developing the MMG further. Links between the dissonance and note/pitch jump occurrence plots have been observed (e.g. unison and fifth have noticeable peaks in all three). This is perhaps a path toward the creation of a compositional system free of scale constraints and even a unifying theory of tuning systems.

USING A MICROTONAL SCALE

The following summarises the use of the MMG with a microtonal scale.

First the user must input the desired tuning system using MIDI note numbers (not necessarily integers). The Markov rhythm generator then creates the structure for the melody independent of tuning system. When assigning pitches, the functions applied by the MMG depend on the type of microtonal scale input. This is indicated by setting the MMG variable 'MicroType' as 0, 1 or 2. This selection determines what inputs are required and what functions are used in assignment of pitches. A tiered functionality system is created as follows:

	MICROTYPE = 0	MICROTYPE = 1	MICROTYPE = 2
Tuning system	12-TET	n-TET (equal-temp.)	Unequal temperament
Rhythm generation	Markov chain	Markov chain	Markov chain
Note probability distribution	Automatically compiled	Manually input	Manually input
Pitch jump probability distribution	Automatically compiled	Manually input	Not required
Pitch assignment method	Utilises all functions	Utilises all functions	Utilises all except pitch jump

Table 4 - Summary of tuning system input types

5 THEORY

Following are introductions to the major engineering 'tools' that appear in the next chapters.

5.1 MARKOV CHAIN

A Markov chain is a mathematical tool used for statistical modelling (predicting future behaviour) of a system. It operates in discrete steps where the system modelled may be in one of a finite number of states at each step. The systems future state depends on its current state and is determined according to a set of probabilities compiled in what's known as a transition matrix. These states have a number of criteria they must fulfil. The order of a Markov chain indicates how many past/current states the model considers when choosing future states (e.g. 2nd order takes account of the current and previous state).

5.2 GENETIC ALGORITHM

A genetic algorithm (GA) is a method used to search for solutions to an optimisation or search problem. The technique draws parallels with the evolution process in order to achieve its task – Populations of potential solutions (formed of abstracted 'genes') are assessed by a fitness function, those that fare best are more likely to survive into the next generation after simulated 'mating' and 'mutation' occur. This progression aims eventually to create an optimal solution to the fitness function.

5.3 (PEARSON'S) CHI-SQUARED TEST

This is a statistical procedure where results are evaluated against the chi-square distribution (Engineering_statistics_handbook_website). It is used both to test observed distributions compared to theoretical distributions (goodness of fit) and to test the level of independence of two variables. The goodness of fit calculation is based around Eqn. 7, And the test of independence calculations are based around Eqn. 8 and Eqn. 9:

$$\mathrm{X}^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}, \qquad \text{Eqn. 7}$$

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}. \qquad \text{Eqn. 8}$$

$$E_{i,j} = \frac{\sum_{k=1}^{c} O_{i,k} \sum_{k=1}^{r} O_{k,j}}{N}, \qquad \text{Eqn. 9}$$

Where: O = observed value E = expected (theoretical) value X^2 = chi-squared value (test statistic)

r = total rows (in table of observations) c = total columns N = total observation number

The tests assume errors have a normal distribution. Knowledge of the degrees of freedom (DOF) present and the level of significance (p-value) desired allow a comparison of the calculated test statistic to be made with the chi-square distribution. The degrees of freedom are found by Eqn. 10.

$$DOF's = n - p \qquad \text{Eqn. 10}$$

Where; n *= no. categories* p *= no. parameters used to fit distribution +1*
And: For independence (two variable) test $p = r + c - 1$

The p-value (not related to p above) is the probability of obtaining a test statistic at least as extreme as the one that was observed, assuming that the null hypothesis is true. In other words, the p-value is the chance that the null hypothesis explains the result.

Pearson's chi-squared test requires a sufficiently large size for both the total sample and the individual frequencies for each category. If expected frequencies are too low, the approximation to the chi-square distribution can break down. A rule of thumb states the test is reliable provided no more than 10% of cells have expected frequencies of smaller than five.

5.4 G-TEST

The G-test is an alternative to the chi-squared test in cases where expected frequencies (E) are low (Handbook_of_biological_statistics_website, 2009), and particularly if for any cell; |Observed − Expected | > Expected.

If sample size is sufficient, the G-test and the chi-squared test will lead to the same conclusions. But for low expected frequencies, the approximation to the chi-square distribution is better for the G-test. DOF's and p-values are as for chi-squared. The G-test is based on Eqn. 11.

$$G = 2\sum_i O_i \cdot \ln(O_i/E_i) \qquad \text{Eqn. 11}$$

Where symbol meanings are as for the chi-squared equations

In cases where the expected frequency adequacy is questionable, the G-test has can be used to confirm the chi-squared statistic.

5.5 TWO SAMPLE T-TEST

The two sample *t*-test is used to determine if two population means are equal in situations where n < 30 (Engineering_statistics_handbook_website). Data sets should be independent from each other and distributions are assumed to be normal. In this report, it is used for unpaired data with variances assumed unequal. The test statistic uses Eqn. 12 and the critical value requires Eqn. 13. Knowledge of the DOF retrieves a critical t-value to be compared with the test statistic.

$$T = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{s_1^2/N_1 + s_2^2/N_2}} \qquad \text{Eqn. 12}$$

$$DOF's = \frac{\left(s_1^2/N_1 + s_2^2/N_2\right)^2}{\left(s_1^2/N_1\right)/(N_1 - 1) + \left(s_2^2/N_2\right)/(N_2 - 1)} \qquad \text{Eqn. 13}$$

Where; N_i *= sample size,* $\overline{X_i}$ *= sample mean, and* s_1^2 *= sample variance.*

6 MELODY STATISTICS

6.1 THE NEED FOR AN ANALYSIS

A program creating melodies algorithmically requires explicit knowledge of the rules and principles underlying melody. **Musical theory** consulted was frustratingly vague and, whilst providing clues as to elements that may be important, proved little help in formalising rules governing melody – typical wisdom included; "create an effective shape for the melody", "establish a clear sense of keynote" or "find the proper balance in the melody between steps and leaps".

The literature review revealed some individual approaches that had been taken in the formalisation of these rules – namely through Baker, Iazetta, Collin and Keith – who largely formulated them through **personal intuition**. Others had removed the need for these formalisations in various ways – Wooller, Biles, Cope and Werner – though with other disadvantages introduced. Markov models also remove this need, though at the expense of novelty.

An example of this personal intuition included Collins' work where penalties were given to a musical phrase if notes were repeated or the melody contained large note intervals. The basis for assigning these penalties is sound, but the arbitrary scheme seems unscientific. Keith was the only one to base formalisations around something of more substance – data from experimental research concerning psychologically favoured notes was used to compile pitch and time probabilities.

Hence, there seemed an opportunity for improvement by replacing the usual subjective nature of these rule formalisations with something solid. A solution appeared to be through an analysis of a number of melodies, the aim being to seek out patterns and relationships between variables, as well as the significance of these relationships. Any consistent discoveries could then be coded and used in the generation of new melodies.

6.2 THE ANALYSIS

Analysis Details

The aim of the analysis was to discover relationships between variables within a library of music that appear unable to have occurred randomly.

The following results and tests compare **observed data distributions** extracted from a library of melodies with **'idealised random music'** – that in which no order would be present.

The **melody library** contained thirty melodies of a dance/pop genre that have appeared in the UK charts in the last five years. Examples included; ATB/9AM, Crystal Waters/Destination unknown, Calvin Harris/Vegas. The majority of the library consisted of four bar melodies, with a minority lasting eight bars. All were in 4/4 time. Melodies were transposed to the key of' 'C', and categorised as 'C major' or 'C minor' melodies. Seven out of the thirty were classed as C major, with the other twenty-three belonging to C minor.

The number of potential relationships was vast, and investigated were those deemed by the author as most likely to be found significant.

Potential variables list:

- *Pitch*
- *Note length*
- *Location in melody*
- *Pitch as a ratio of the max/min pitch in the melody*
- *Pitch interval between adjacent notes*
- *Tension level of note (dissonance)*
- *Frequency of occurrence*

Variables to do specifically with the rhythmic nature of the notes (e.g. note density), were ignored. This was due to the implementation of rhythm structure via a Markov chain – this method does not require formal rules.

The statistical tests used to decide if a relationship was significant were Pearson's chi–squared test and the G-test (sec. 5.3).

Melody Analysis

The results presented contain data extracted from all melodies except for those where the variable of 1/16th bars is included – in these cases, data is taken from all melodies but only over the first four bars.

Note occurrence

Perhaps the simplest potential pattern was the number of times each note of the 12-TET system was used. The observed data (blue) in Figure 7 is compared against a theoretical 'random' distribution (red) that would be expected for no preference regarding note usage.

The **random distribution** was formed assuming that notes in a given melody must be chosen from the scale (C minor or C major) assigned to that melody. The total number of notes (709) was therefore distributed as; 709x23/30 = 543.6 minor scale notes and; 709x7/30 = 165.4 major scale notes. With these tallies assumed evenly distributed over the seven notes in their scale. The note occurrences ignore changes of octave.

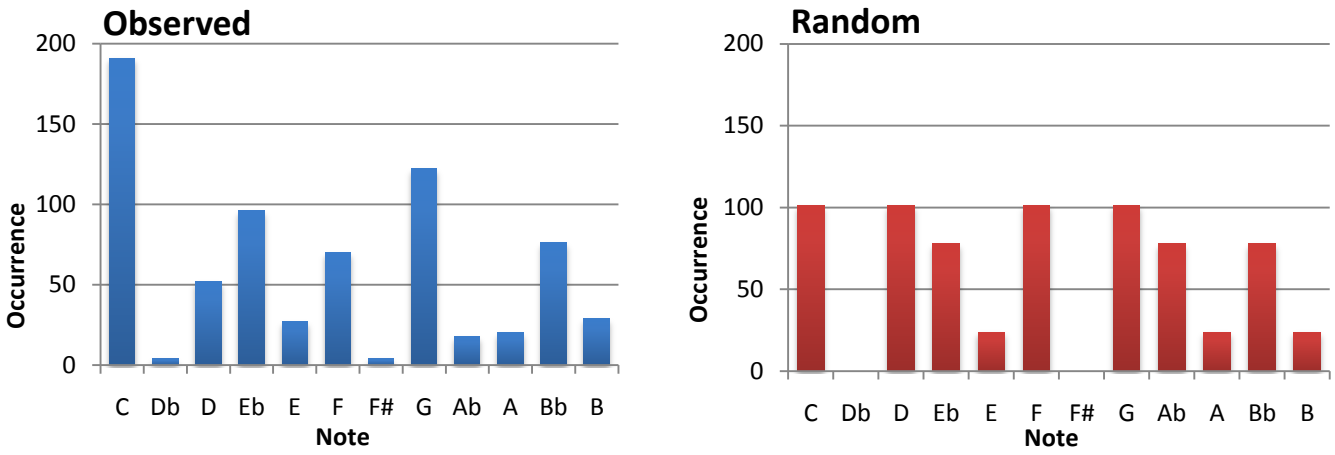


Figure 7 - Note occurrence in 12-TET comparison

In conducting statistical tests below, the approximation that Db and F# contained zero notes (as for expected distribution) was made.

H_0 *(null hypothesis):* All notes are used the same amount (in a given scale)

H_A *(alternative hypothesis):* All notes are not used equally.

DOF's = 9, p-value = 0.01, $X^2 critical = 21.7$

Chi-Squared test statistic: $X^2 = 169.8$

G-test statistic*: $G = 163.9$

$X^2 \gg X^2 critical$, $G \gg X^2 critical$; Therefore reject H_0, and state **all notes are not used equally** – the profile observed could not belong to a random music.

* *G-test was not actually required in this instance, but was calculated to show the similarity with chi-squared*

PITCH INTERVAL OCCURRENCE

*Interval refers to the pitch difference **not** the time period between two notes. It is used interchangeably with 'jump'.*

In the same vein as for note occurrence, it seemed likely that the occurrence of different sized pitch jumps between notes would be significantly different to that obtained by chance.

To form the expected distribution for the **random music** equivalent, first it was assumed that the maximum and minimum pitch jumps allowed were +/-12 semitones (as observed). Further to this, it was assumed that melodies were constrained to remain within a single octave (12

semitone band). This meant that some pitch jumps were more likely than others – e.g. a jump of 12 semitones can only occur when a note is on the edge of the allowed pitch band, whereas a jump of zero (a repeat) could occur every time. When scale restraints were imposed (as for note occurrence), Figure 8 was produced.

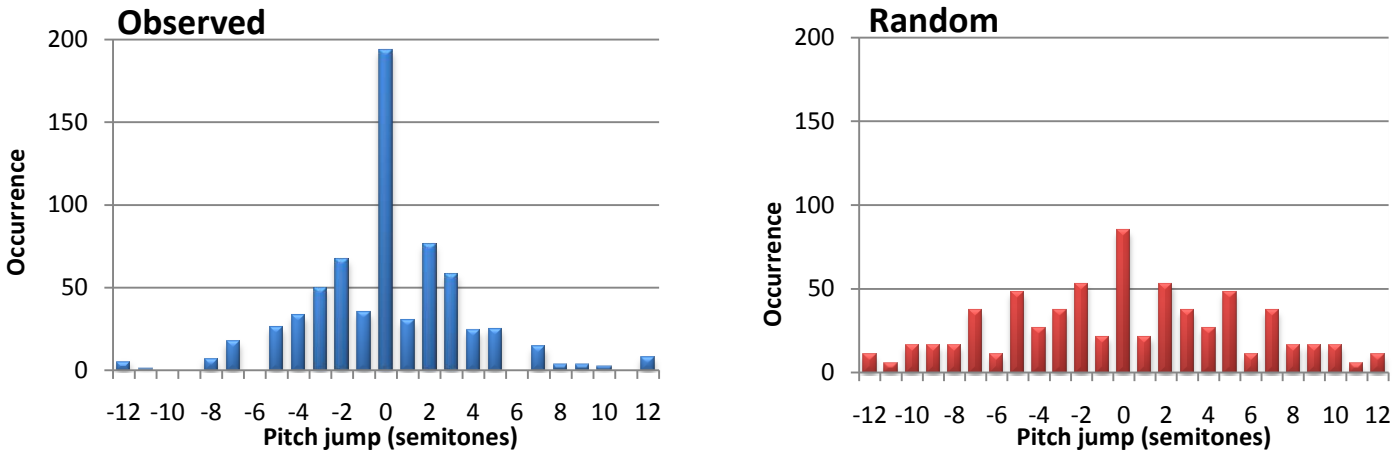


Figure 8 - Occurrence of pitch interval comparison

For the conducted chi-squared test, pitch jump categories were merged in such a way so as to avoid frequency counts of less than five. The G-test was necessary to check the result due to the low counts in the distributions periphery, and because |Observed − Expected | > Expected for the central spike.

H_0: All pitch jump sizes are equally likely to occur (given fixed pitch boundaries)

H_A: All pitch jump sizes are not equally likely to occur

DOF's = 24, p-value = 0.01, $X^2 critical = 43.0$

Chi-Squared test statistic: $X^2 = 387.7$

G-test statistic: $G = 328.1$

$X^2 \gg X^2 critical$, $G \gg X^2 critical$; Therefore reject H_0, and state **all pitch jump sizes are not equally likely to occur** – the set of observations could not belong to a random music.

LOCATION OF EXTREME PITCHES IN MELODY

The location of extreme notes is given in Figure 9 (visualised as a breakdown of max and min notes). An 'extreme' note refers to the maximum or minimum pitch contained in that melody (there may be more than two extreme pitches per melody). The expected distribution was calculated as the total number of extreme pitches spread uniformly over the two bars investigated.

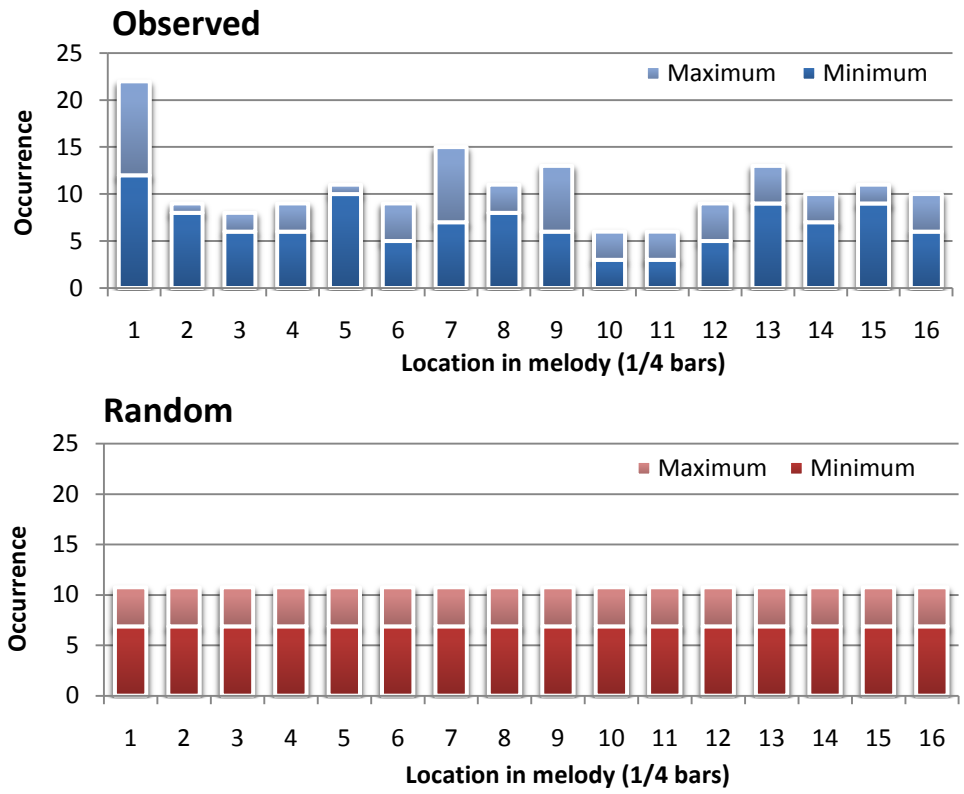


Figure 9 - Extreme pitch location comparison

The combined tally of maximum and minimum pitches was used to conduct a chi-squared test.

H_0 ; Max/min pitches equally likely to occur anywhere in melody

H_A ; Max/min pitches more likely to occur in some parts of melody than others

DOF's = 15, p-value = 0.01, $X^2 critical = 30.6$

Chi-Squared test statistic: $X^2 = 20.6$

$X^2 < X^2 critical$; Therefore the relationship cannot be deemed significantly different from a uniform distribution – H_0 is withheld and state **extreme pitches are equally likely to occur anywhere in the melody**. This supports knowledge that pitch contours vary more on a case-by-case basis than on global trends.

HOW TENSION VARIES WITH LOCATION IN MELODY

Figure 10 displays how tension (dissonance) was observed to change with position in the melody, compared to how it should for a random melody. In order to conduct a test of independence, the continuous dissonance data was put into bins of three levels corresponding to highly dissonant ¼ bars, medium dissonance ¼ bars and low dissonance ¼ bars. The random distribution was created with Eqn. 9, where rows correspond to the three levels of dissonance and columns to the sixteen ¼ bars. Data is displayed in a stacked bar chart to mimic the categories used in the statistical tests.

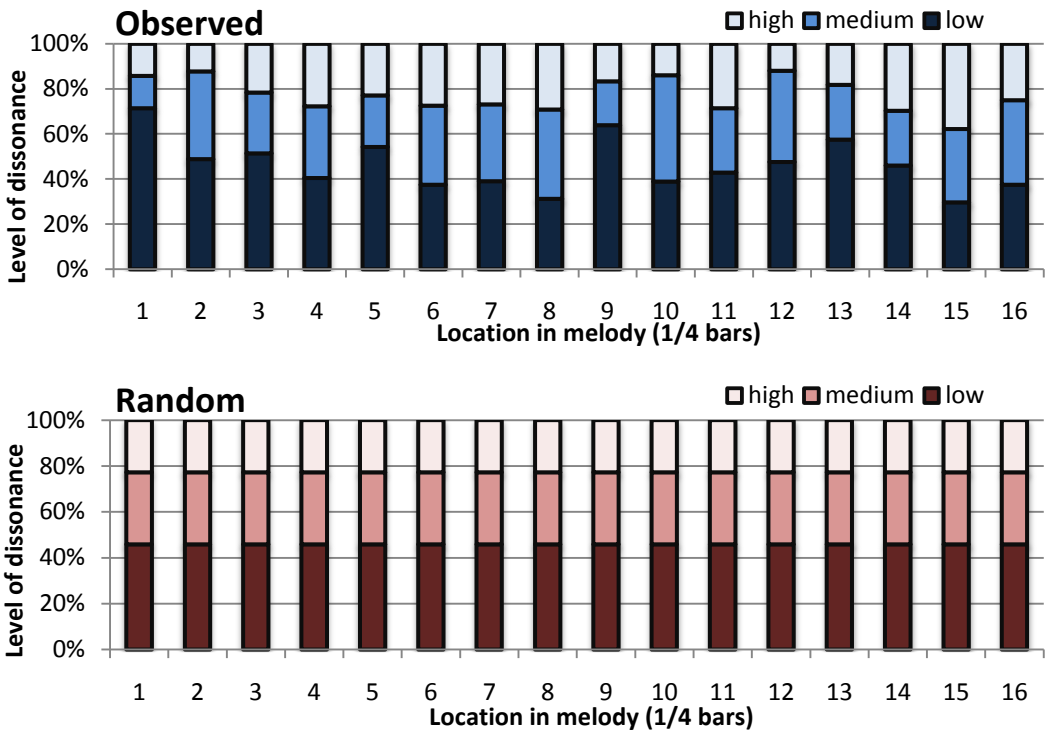


Figure 10 - Tension's relationship with location in melody (high = dissonant, low = consonant)

H_0 ; Tension level is unrelated to the position over two bars

H_A ; Tension level is related to the position over two bars

DOF's = 30, p-value = 0.01, $X^2 critical = 50.9$

Chi-Squared test statistic: $X^2 = 46.27$

$X^2 < X^2 critical$; Therefore H_0 cannot be rejected and maintain **tension level is unrelated to the position over two bars**. As with melody contour, this supports knowledge that tension contours vary more on a case-by-case basis than on global trends.

NOTE LENGTH WITH PITCH AND TENSION

It was hypothesised that longer notes might tend to be associated with certain pitches, perhaps those with a lower dissonance. Figure 11 shows the observed distribution alongside the expected data if this hypothesis were not to be the case. The methods of calculation are as for the previous test. Insufficient data was available for Db and F# and these notes have been ignored in graphs and calculations.

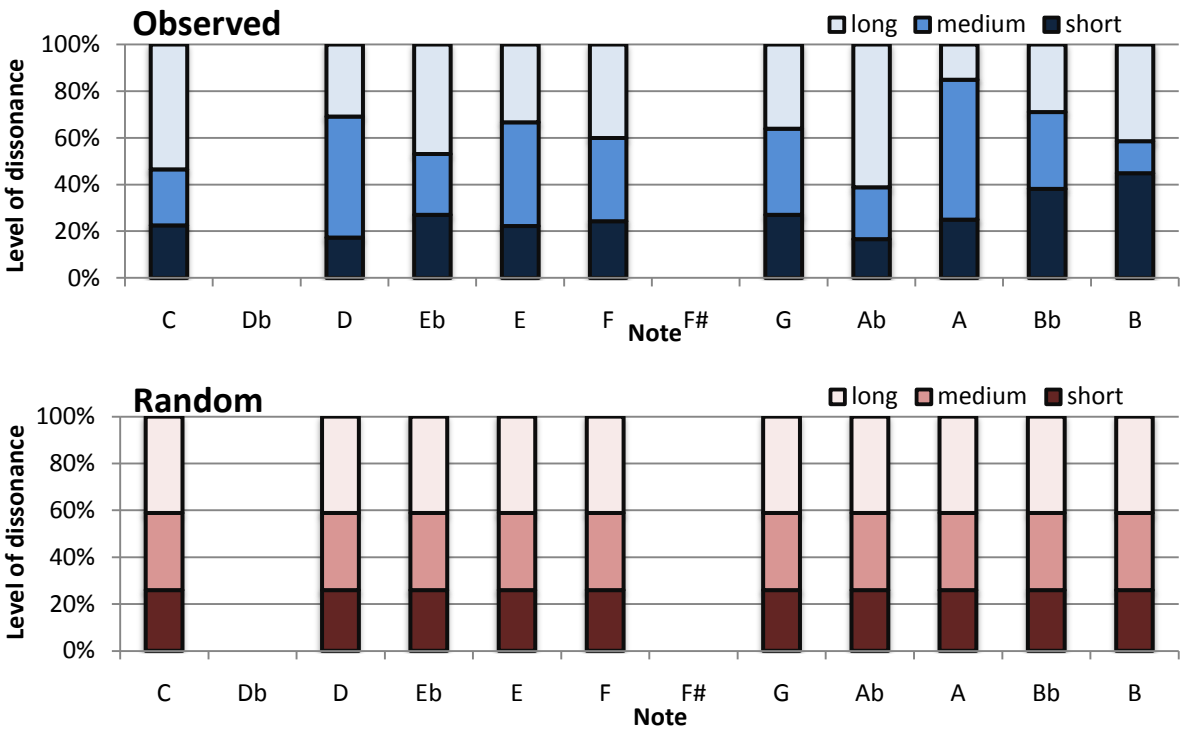


Figure 11 - Note length varying with pitch comparison

H_0 ; Note pitch has no influence on the length of note

H_A ; Note pitch does have an influence on the length of note

DOF's = 18, p-value = 0.01, $X^2 critical = 34.8$

Chi-Squared test statistic: $X^2 = 51.0$

$X^2 > X^2 critical$; This shows H_0 to be weak, though the small margin indicates a borderline case. It was suggested that a plot of tension Vs dissonance might visually display some trend but Figure 12 showed this not to be the case. Along with the random appearances of both graphs, the relationship was not deemed significant enough to justify modelling during melody generation. H_0 was not rejected so state **note pitch has no influence on the length of note**. Further searches regarding note length were ended in parallel with the decision to ignore note length (see sec.3.4).

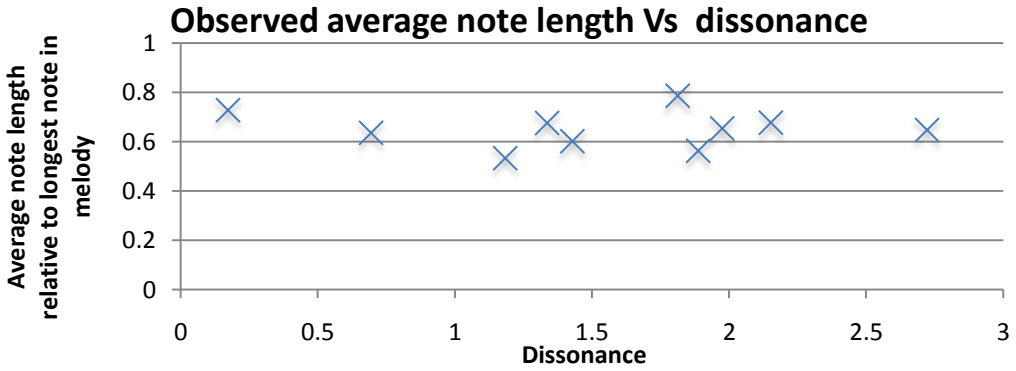


Figure 12 – Average note length Vs Dissonance

Figure 13 shows categorised data for the pitch interval magnitude (ignoring the direction of the jump) against position in melody. The data comprises 597 instances and uses methods as previously described.

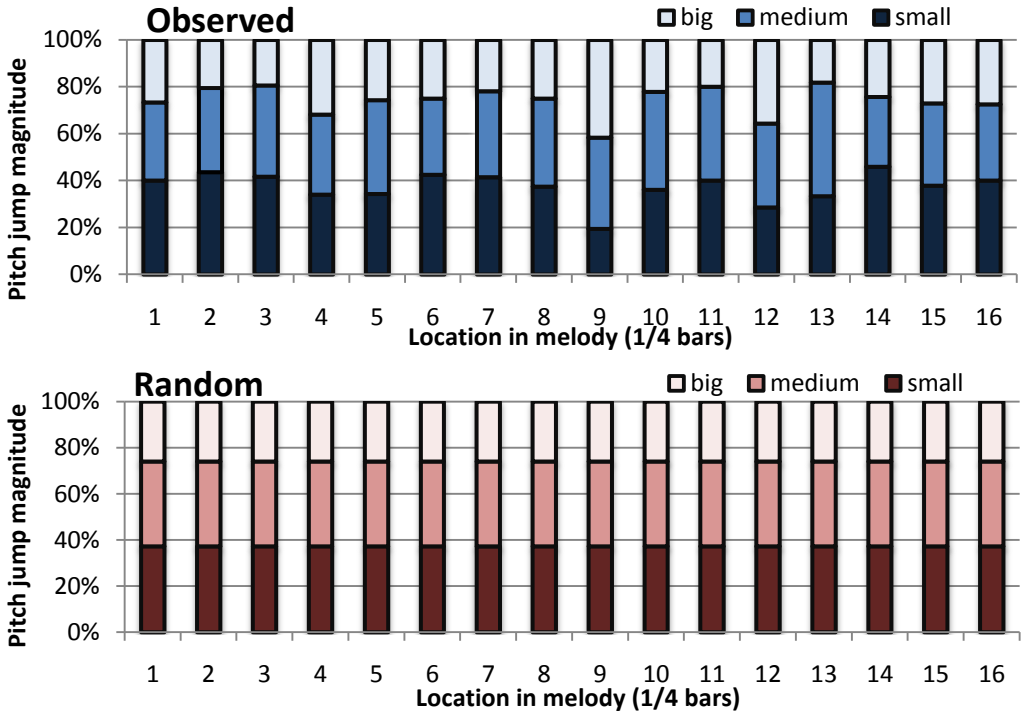


Figure 13 - Potential relationship between pitch interval magnitude and position in melody

---

H_0 ; Pitch interval magnitude has no relationship with position in melody

H_A ; Pitch interval magnitude does have a relationship with position in melody

DOF's = 30, p-value = 0.01, $X^2 critical = 50.9$

Chi-Squared test statistic: $X^2 = 17.62$

$X^2 < X^2 critical$; The result dictated that H_0 should not be rejected, so state **pitch interval magnitude has no relationship with position in melody**. This again confirmed thinking that melody contours are described on a case by case basis rather than globally.

6.3 ANALYSIS CONCLUSIONS

Two key distributions were found to be significant. They were the occurrence of notes and occurrence of pitch jump. Failure to find additional significant relationships led to further investigations being ceased. During analysis it was observed how important the melody contour (i.e. the shape in which the pitches rise and fall over time) was, with every melody appearing to have its own form. Repeated themes were noticed (e.g. a series of rising tones followed by a series of descending tones) and creation of a system with the purpose of identifying and reproducing these contours is a possible area of development.

7 THE PROGRAM

Refer back to sec. 1.4 for a general introduction.

7.1 PROGRAM PHILOSOPHY

The program's function is to generate a novel & pleasing two bar melody and save it as a MIDI file.

The program's generation essentially relies on three elements:

1. User parameters – Tuning system and scale, pitch range, density envelope, tension envelope and G.A. power coefficients.
2. Data compiled from the melody library or input manually – for note occurrence, pitch jump occurrence and rhythm information.
3. Random component – present in the Markov chain, density adjustment process and the genetic algorithm.

The MMG is not a 'hit spacebar and relax' algorithmic composer. It aims to give the user a level of **effective control** over the melody produced.

Two bar melodies are typically found in pop/dance songs where they are repeated with slight alterations to make up a four or eight bar loop (e.g. using transposition of some notes). It is envisaged that the user would **copy/paste/alter** the output melodies in this way.

The program takes on average a couple of seconds to produce a melody. This makes **real-time** applications possible. It also means that the user can generate a vast quantity of melodies quickly, allowing a wide range of melodies to select from.

The output MIDI files, whilst being the programs end product, still benefit from **user attention**. For example alteration of the note lengths is suggested. Experience has showed that sometimes a minor modification to the tune (such as small pitch adjustment) can have a large positive effect.

Currently, the user is required to alter the source file 'microspine.m' to input their selections. Whilst this process has been made as simple to understand as possible, users must really understand the MATLAB language. It is recommended in sec. 0 that development of the MMG should include a **GUI**. Specifically a MATLAB toolbox could be created, or else a plug-in application for Ableton Live (music arrangement/composition software) or similar.

By using a **Markov chain** to generate the **rhythm structure** and assigning pitches after, it was hoped that the benefits of a Markov process (simple method, good quality mimicries) could be achieved without the melodies produced losing their novelty. A first-order scheme used was first

– it was reasoned that the chain's purpose is essentially to create a random set of notes with some sense of order. The adjustments for the density envelope would largely negate a higher than first order process, making the added complexity redundant.

A **Genetic Algorithm** was used for **pitch assignment** because it seemed the best way of balancing the number of criteria demanded, whilst allowing sufficient space for novelty.

7.2 USER INPUT PARAMETERS

The user inputs and their effects are summarised below:

TUNINGS SYSTEM/SCALE – Any tuning system may be input by specifying the appropriate MIDI note numbers. The notes to be used (the scale) within this are indicated by assigning each value – '0' (don't use) or '1' (use).

RANGE SELECTION – This details the maximum and minimum (and hence the range of) pitches allowed during generation.

NOTE OCCURRENCE – This can be thought of as the probabilities of notes within the tuning system being used in generation. Should 12-TET tuning be chosen, the distribution of note probabilities is automatically compiled via the library, otherwise it must be input manually.

PITCH JUMP OCCURRENCE – Similar to note occurrence, this defines the likelihood of each pitch jump being used in generation. For 12-TET, the distribution is compiled automatically and for equal tempered scales it needs a manual input. Un-equal tempered scales do not require this.

DENSITY ENVELOPE – Density relates to the quantity of notes in a given unit of time. The MMG density envelope divides the output melody into eight ¼ bars, and each is assigned a density value from 0.0 (few notes) – 10.0 (many notes).

TENSION ENVELOPE – As for density, the desired tension is specified for each ¼ bar. Tension refers to the average dissonance of the notes. The range is from 0.0 (very consonant) – 10.0 (very dissonant).

G.A. POWER COEFFICIENTS – These alter the level of emphasis placed on each criterion during pitch assignment. May range from 0.0 – infinity where the relative NOT absolute magnitude is important.

7.3 ARCHITECTURE, '.m' FILES

The MMG program was written in MATLAB and consists of a number of '.m' files, those written purposefully for the MMG (given a * in the following list) can be found in appendix 12.2, those dealing with the MIDI interface are not included. The parent file is 'microspine.m' and this should be run to start the program. User inputs are altered by editing this file. As the name suggests, this file acts as the spine for the program – calling the other files as necessary and collating all variables together. Figure 14 displays the architecture of the MMG's .m files. Leading on is a brief description of each file.

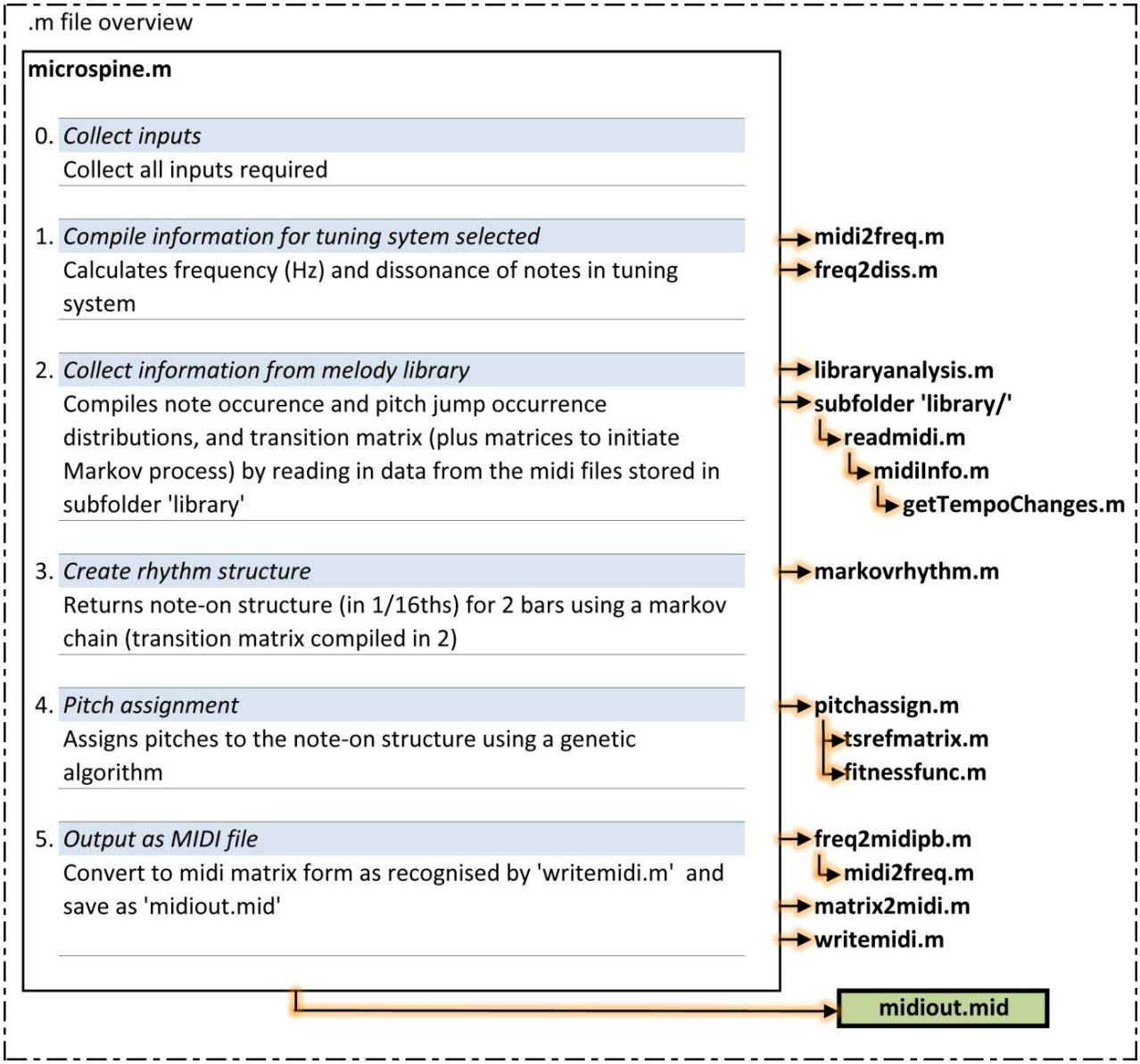


Figure 14 - .m file overview

microspine.m* – Collates all information and calls other functions.

midi2freq.m* – A short function that converts MIDI note numbers to frequencies (Hz). The MIDI numbers may be non-integers (microtonal pitches).

freq2diss.m* – Assigns values of dissonance for a vector of frequencies (Hz), relative to the first frequency in the vector. The timbre must be specified (what harmonics to consider and their decay rate).

libraryanalysis.m* – Gathers data from the melodies in 'library/' for note timings and pitches. For a 12-TET tuning system, the distributions of note usage and pitch jump occurrence are then compiled. Data required by the Markov chain is also collected.

readmidi.m – Reads MIDI file format.

midiInfo.m – Displays (amongst other things) the 'readmidi.m' output as a matrix of MIDI notes and timings.

getTempochanges.m – This is called by 'midiInfo.m'. It searches the 'readmidi.m' output for changes to tempo (which affect note timings).

markovrhtyhm.m* – Returns a vector of note timings (in $1/16^{th}$'s) corresponding to when notes should be turned on for a two bar melody. This is done via a Markov process and subsequent adjustment according to the density envelope.

pitchassign.m* – Takes the note timings and returns a vector of pitches (MIDI notes) for them. It sets various parameters for the Genetic Algorithm, which it then calls.

tsrefmatrix.m* – Creates a two column matrix, where all notes dictated by the tuning system are found over the pitch range specified. It's required for finding the nearest note for a given random number in the allowed pitch range.

fitnessfunc.m* – Used in the G.A. It returns a value for the variable 'score' according to three criteria. The weighting of these three criteria are affected by G.A. coefficients.

freq2midipb.m* – Takes a frequency (Hz) and returns the whole MIDI note number that a synthesiser should play, as well as values that it should bend the pitch by to achieve the desired frequency.

matrix2midi.m – Converts a matrix holding note pitch and timing information into a MIDI MATLAB format recognised by 'writemidi.m'

writemidi.m – Saves the information in MIDI MATLAB format as the MIDI file 'midiout.mid'

library/ – The library of melodies used in this project was identical to that described in the statistical analysis – 30 melodies of pop/dance genre (see sec. 6.2).

7.4 TECHNICAL WORKINGS

Following, the major components of the MMG program are described in detail.

MARKOV RHYTHM GENERATION

Files used: libraryanalysis.m, markovrhtyhm.m

The purpose of the rhythm generator is to create a vector where each value represents a point in time that a note in the melody should begin on. It could be viewed as producing a melody with notes in fixed timing positions, but with no pitches assigned.

A first-order Markov chain (see sec. 5.1) uses a transition matrix compiled from the supplied melody library to generate an initial version of this vector. This is then adjusted by a probabilistic algorithm according to the desired density envelope already provided by the user.

STATES

As a variable, continuous time has been divided into discrete units of 1/16th bars. The states used by the Markov chain are defined as the time interval between consecutive notes. In order for a Markov model to be valid, the following criteria must be met by the state space:

- States must be mutually exclusive
- A finite number of states must exist
- States must be exhaustive

The state type selected clearly fulfils the first criteria, and the second two are achieved by placing a limit on the maximum interval recognised. No intervals in the melody library were found to be larger than one bar (sixteen 1/16th's), and so this was selected as the maximum state. Any interval that is found to be larger than this is simply ignored.

INITIATION

To initiate the Markov chain, the start time of the first note in the melody must be selected along with the first interval (state) to be used. These are chosen in the MMG with a probability that mimics their occurrence in the melody library – data for first notes and intervals is collected by the file 'libraryanalysis.m' – see later. Table 5 shows the actual compiled matrix for the probabilities of the time of first note-on alongside the accompanying selection code. The third column has been manually added to show the probability of the row being selected (found as the difference of adjacent rows in column two). The same method is used for selecting values for other matrices througout the programme including the transition matrix during the Markov process and for first note interval (initiation).

Table 5 – First note-on time probability matrix ('firstnoteon')

1/16th's	Form in program	Probability being selected
0	0.7586	0.7586
1	0.7586	0
2	0.8966	0.138
3	0.8966	0
4	0.9655	0.0689
5	0.9655	0
6	0.9655	0
7	0.9655	0
8	1	0.0345

Accompanying code:

```
x = rand;          % random number
SELECTION = 1;     % default value
for i = 1:size(MATRIX,1)
% using random number, find where falls in matrix
  if x < MATRIX(i,2)
    SELECTION = i; break;
  end
end
```

The code selects the time of the first note according to the probability of being selected which is calculated from the melody library. The default value of one is a safety measure against incorrect matrices.

MARKOV CHAIN

Having determined the first state, the Markov chain then runs using the code opposite Figure 15 which shows a visualisation of the compiled transition matrix used. The size of intervals considered is shown to be adequate, with the majority of non-zero cells in short intervals (top left corner). Note the preference for notes to occur on 1/8th's of a bar and the matrices diagonally dominant tendency.

```
noteons = [];
%generate until (over) two bars are
%completed (31 sixteenths = 2 bars)
while sum(noteons) < 31
  x = rand;
  next = 1;
  for i = 1:size(transition,1)
    if x < transition(current,i)
      next = i; break;
    end
  end
  noteons = [noteons; next];
  current = next;
end
```

where 'transition' is the transition matrix:

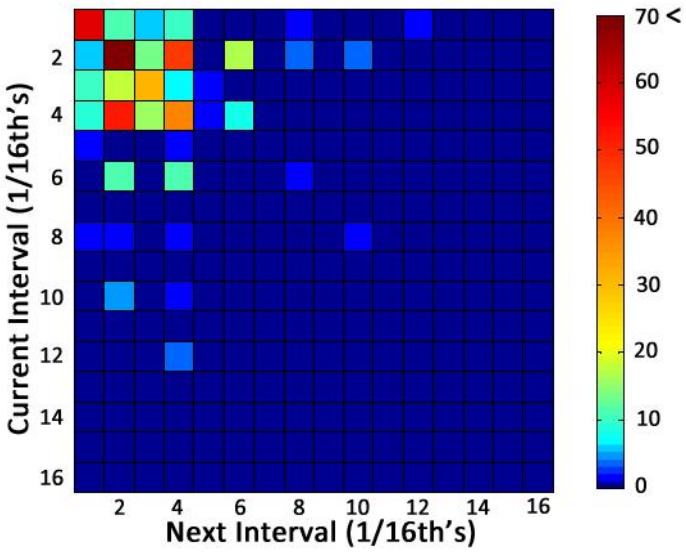


Figure 15 - Visualisation of the cumulative transition matrix

The process stops once the produced rhythm has exceeded the two bar limit. The last note is then deleted as it will be outside the two bar limit. The intervals (time relative to the previous note) are then accumulated to produce the actual times that each note should be turned on.

Adjustment Via Density Envelope

The generated note-on structure is altered according to the user specified density envelope in which each ¼ bar has been assigned an associated density value (0.0 – 10.0). The algorithm used considers every 1/8th of a bar individually (this value was chosen due to the preference observed in the transition matrix) and adds or removes a note at that 1/8th (if possible) with a probability dependent on the density value for that ¼ bar. Figure 16 shows this algorithm.

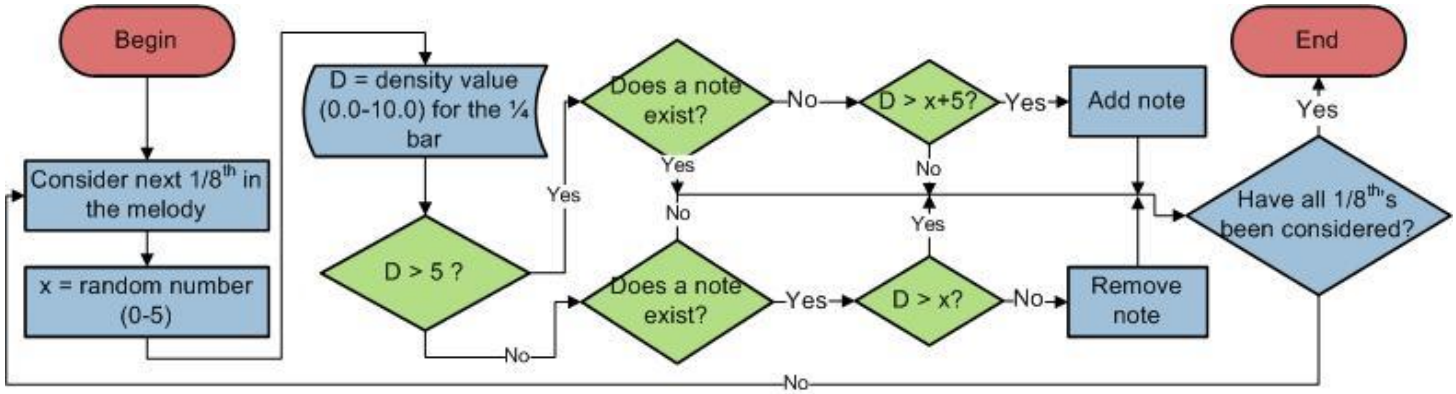


Figure 16 - Density envelope adjustment algorithm

Statistical Library Analyser (implementation)

Files used: libraryanalysis.m

The library analyser opens each MIDI melody in the subfolder in turn, compiling relevant information into a matrix (variable name 'grand') as in Table 6. Each note in the melody library corresponds to a row in 'grand'. The third column is only worked out if 12-TET tuning is selected. It contains the note number regardless of octave – e.g. 'C#' which can have a MIDI number of 61, 73 or 85 will always be assigned '2'.

MIDI note	1/16th on (note-on time)	MIDI note (1-12) – ignore octave
72	18	1
75	20	4
77	22	6

Table 6 - Extract from the statistical matrix 'grand'

This master matrix is then used to form a collection of five sub-matrices that are returned in order to be used in other functions. Each will now be described headed by their variable name.

notefreqcount **–** This matrix contains the occurrence (count) of each of the twelve notes (C, Db, D...Bb, B) in the 12-TET scale. It is not formed when a microtonal tuning system is selected. The matrix 'grand' is searched for each of the twelve notes (column three) and 'notefreqcount' stores the number of times each is used. Plotted, it produces Figure 17.

PJUMPFREQCOUNT – The cumulative frequency of every sized pitch jump between consecutive notes is stored in 'pjumpfreqcount'. The jumps are measured in semitones, both positive (up) and negative (down). To tally, the first column of 'grand' is used where the difference between two consecutive rows constitutes a pitch jump. If the note is the first of a melody, its previous jump is ignored. This criterion is implemented by comparing the note-on time of the row under inspection with its predecessor. If the time has decreased since the last instance, the note is assumed to be the first of the melody. This works because notes have been sorted chronologically in 'readmidi.m'.

The range of values for pitch jumps were found to be from +12 semitones to -12 semitones. Figure 17 – Used note occurrence distribution and used displays the distributions for the library used.

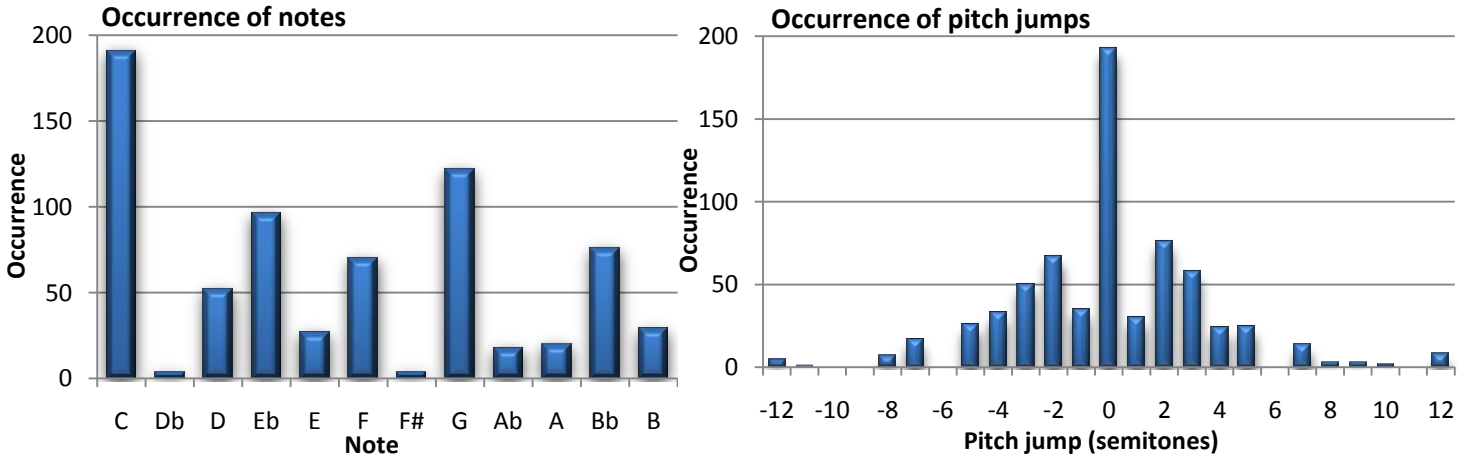


Figure 17 – Used note occurrence distribution and used pitch jump distribution

FIRSTNOTEON – This matrix was seen in (first two columns only). It contains probabilities of the first note of a melody beginning on a given 1/16th step. This is equal to the number of first notes in the library that have been observed to begin on that step, divided by the total number of melodies.

To form this matrix, the second column of 'grand' is needed. For the first note in each melody (found using method as for 'pjumpfreqcount'), the time in 1/16th's that it begins on is saved. When all first notes have been found, the matrix is ordered into the form presented.

FIRSTINTERVAL – Taking the same form as 'firstnoteon', the first column indicates the size (in 1/16th's) of the interval between the first and second notes of a melody and the second column relates to the number of times that this interval was used in the library. Once the frequency count is completed, the data is transformed into the form shown for 'firstnoteon'.

TRANSITION – This matrix contains the transition probabilities for the subsequent Markov chain. Columns relate to the current state (interval) and rows to the next state (both in 1/16th's).

Initially, a blank matrix of 99x99 is created, and a count ensues where the program scrolls through column two of grand and at each row adds a count to the appropriate cell. The method as previously described seeks out the beginning of a song (and does not add these instances). Once the run through is complete, the matrix is reduced to a 16x16 size with any surplus instances being ignored.

All that is then left is to convert the matrix from count form into that which can be selected via the process described earlier. This creates the change as demonstrated by Table 7 and Table 8.

		Next Interval			
	1/16th's	1	2	3	4
Current Interval	1	59	11	6	.
	2	6	203	14	.
	3	10	18	31	.
	4	.	.	.	.

Table 7 – (extract of) Cumulative transition matrix

		Next Interval			
	1/16th's	1	2	3	4
Current Interval	1	0.65	0.77	0.84	.
	2	0.02	0.70	0.74	.
	3	0.15	0.41	0.87	.
	4	.	.	.	.

Table 8 - (extract of) Transformed transition matrix

For those rows containing no values, the Markov chain sets a default value of one 1/16th as the next interval.

GENETIC ALGORITHM PITCH ASSIGNER

Files used: pitchassign.m, fitnessfunc.m, tsrefmatrix.m

The purpose of the GA is to assign pitches to the note structure. To begin, a number of options must be set. These include the pitch boundaries (user input), mutation coefficients (set largely through trial and error) and plot options which is useful to have if melodies are being produced singly, but slows the process down unnecessarily for multiple generations.

The initial population is set by an algorithm that assigns all notes random pitches within the central half of the allowed melody band – trial and error found this to produce the best results.

In order to convert random numbers into the MIDI notes used by the tuning system, 'tsrefmatrix.m' must be called. This returns a matrix 'subtunsyst' of all legitimate notes (for microtonal systems and 12-TET) within the allowed pitch band. The following code then calculates and selects the nearest available note to the random number.

```
[trash, array_position] = min(abs(subtunsyst(:,2) - x));
initialpopulation(i,1) = subtunsyst(array_position,2);
```

Where x = a random number in allowed pitch range

The GA is then called using the initial population generated and options as set. The number of variables equals the number of notes (as determined by the Markov rhythm) and the file 'fitnessfunc.m' is used to test the choice of assigned pitches according to **three criteria** that total to make up 'score' – the fitness value for an individual. The GA aims to minimise variable 'score'.

Before entering the genetic algorithm, the note occurrence distribution is first altered according to the notes chosen as the selected scale (in 'tuningsystem'). For these notes, the distribution is left un-tampered, but for those not desired, the distribution changes the appropriate row in 'notefreqcount' to zero –indicating not to use them.

Criteria one and two

Having decided that the discovered distributions for note occurrence and pitch jump occurrence were significant enough to model in generation (sec. 6.3), the fitness function's task was to ensure output melodies matched these distributions.

This was done on a statistical basis. For note occurrence, the proportion of the generated melody made up of each note is first calculated, and compared to that which should be expected. The residuals are used in the following way to increase 'score':

$$\boldsymbol{score1} = \sum_{i=1}^{n} \frac{(\boldsymbol{Obsereved\ proportion_i - Expected\ proportion_i})^4}{\boldsymbol{Expected\ proportion_i}} \qquad \text{Eqn. 14}$$

Where; n *= total number of notes*

For those notes with an expected proportion of zero, a sufficiently small number is used in the denominator. Raising the residual to the power of four was found to optimise the output distributions similarity to those input.

Pitch jump occurrence uses the same method, but is summed over the range for which 'pjumpfreqcount' has information. Outliers from this assume an expected proportion of near zero as in Eqn. 15.

$$\boldsymbol{score2} = \sum_{i=x1}^{x2} \frac{(\boldsymbol{Obsereved\ proportion_i - Expected\ proportion_i})^4}{\boldsymbol{Expected\ proportion_i}} \qquad \text{Eqn. 15}$$

Criterion three

This criteria aims to match the melodies tension levels to the user's input tension envelope. For every quarter bar section, the average dissonance is worked out (having already calculated the dissonance of each note in the scale) and compared to the user's desired value. The larger the error magnitude is, the larger the score is increased by as in Eqn. 16.

$$\boldsymbol{score3} = \boldsymbol{0.007} \times \sum_{i=1}^{8} |\boldsymbol{average\ dissonance_i - envelope\ dissonance_i}| \qquad \text{Eqn. 16}$$

'Score'

The user is provided with the option to set the strength of each criterion via the 'GA power coefficients' – *'npow' (note occurrence), 'jpow' (jump occurrence) and 'tpow' (tension envelope).* These have an effect as follows:

$$score = npow \times score1 + jpow \times score2 + tpow \times score3 \qquad \text{Eqn. 17}$$

Where score (1-3) represent the criteria as discussed. It should be noted that it is the GA coefficients *relative* value that is important, and increasing all by the same amount will have no effect on the resultant melodies. The scores have been pre-adjusted to be of the same order as each other, and so using equal values for all coefficients gives equal weighting to all criteria.

Ending the GA

The GA's early generations have a rapidly increasing fitness, however improvements plateau with time to become negligible and eventually halt. The process must therefore have stopping criteria. These may include; maximum number of generations is reached, average change of fitness becomes too small (known as a stall) or time limit is reached. Whichever of these is first reached determines the stopping point of the GA.

Matrix to MIDI file

Files used: midi2freq.m, freq2midipb.m, midi2matrix.m, writemidi.m

For the melody to be written as a MIDI file requires three steps. First, the melody data is collated in a matrix as in Table 9 (each column corresponds to a note).

Track	All notes in track 1
Channel	All notes in channel 1
MIDI pitch	Requires integer
Velocity	All notes set at 100 (out of 127 maximum)
Time ON (secs)	= value in 1/16th's divided by 8 (for seconds at 120bpm)
Time OFF (secs)	= Time ON + 1/8th of a bar
Pitch bend (MSB)	Calculated using freq2midipb.m
Pitch bend (LSB)	Calculated using freq2midipb.m

Table 9 - Transposed MIDI matrix

The matrix is read and converted to a saveable form via 'matrix2midi.m' and then saved with 'writemidi.m'. The output filename is determined by the variable 'savefilename'.

8 MMG EVALUATION

Having written the MMG, there was a need to measure the quality of the output melodies. **Two tests** were used: a survey of the output to discover if the MMG succeeded in creating **enjoyable melodies**; and an analysis of the main program functions that determined the MMG's output. It was reasoned that the transparency of the rhythm creation code was a sufficient evaluation in itself; however the success of the genetic algorithms **fitness function** needed verification.

8.1 DISTRIBUTIONS

The MMG pitch assignment is based on the three criteria used in the GA's fitness function. These criteria had objectives as follows:

- Use notes with the same probability as their occurrence in the melody library dictates
- Use pitch jump sizes with the same probability as their occurrence in the melody library dictates
- Create a melody with tension levels emulating the input tension envelope

There was a desire to see how well these objectives were being achieved. This was done by comparing the input distributions and envelope with those of the output melodies.

It was discovered that when only a single criteria was implemented by the fitness function, the output distribution for that criteria matched the input very closely, but as additional criteria were considered, the distributions tended to deviate further from those desired. Figure 18 shows this effect, where each criterion is displayed on a separate graph with the **input distribution** (input) in green, the output distribution for when only that variables criterion was considered (**exclusive output**) in blue and the resulting distribution for the case of all criteria being considered (**merged output**) in red. Scale selection was ignored for these melodies (all notes considered). Each output distribution contains data from eighty melodies.

The deviation is particularly noticeable for the tension envelope, where the average tension for each ¼ bar becomes noticeably closer to the middle value of '5' for merged output. The pitch jump occurrence appears to hold the input distribution closely with the exception of the central peak, however the profile in fact morphs into the random distribution predicted in sec. 6.2 (Figure 8). The note occurrence distribution deviates the least from the input.

These figures show the worth of including GA coefficients as user parameters – the user can decide which aspect of the pitch assignment is important for the melody, and adjust the coefficients appropriately. The result is a better match for the distribution with a higher associated coefficient (at the expense of the rest).

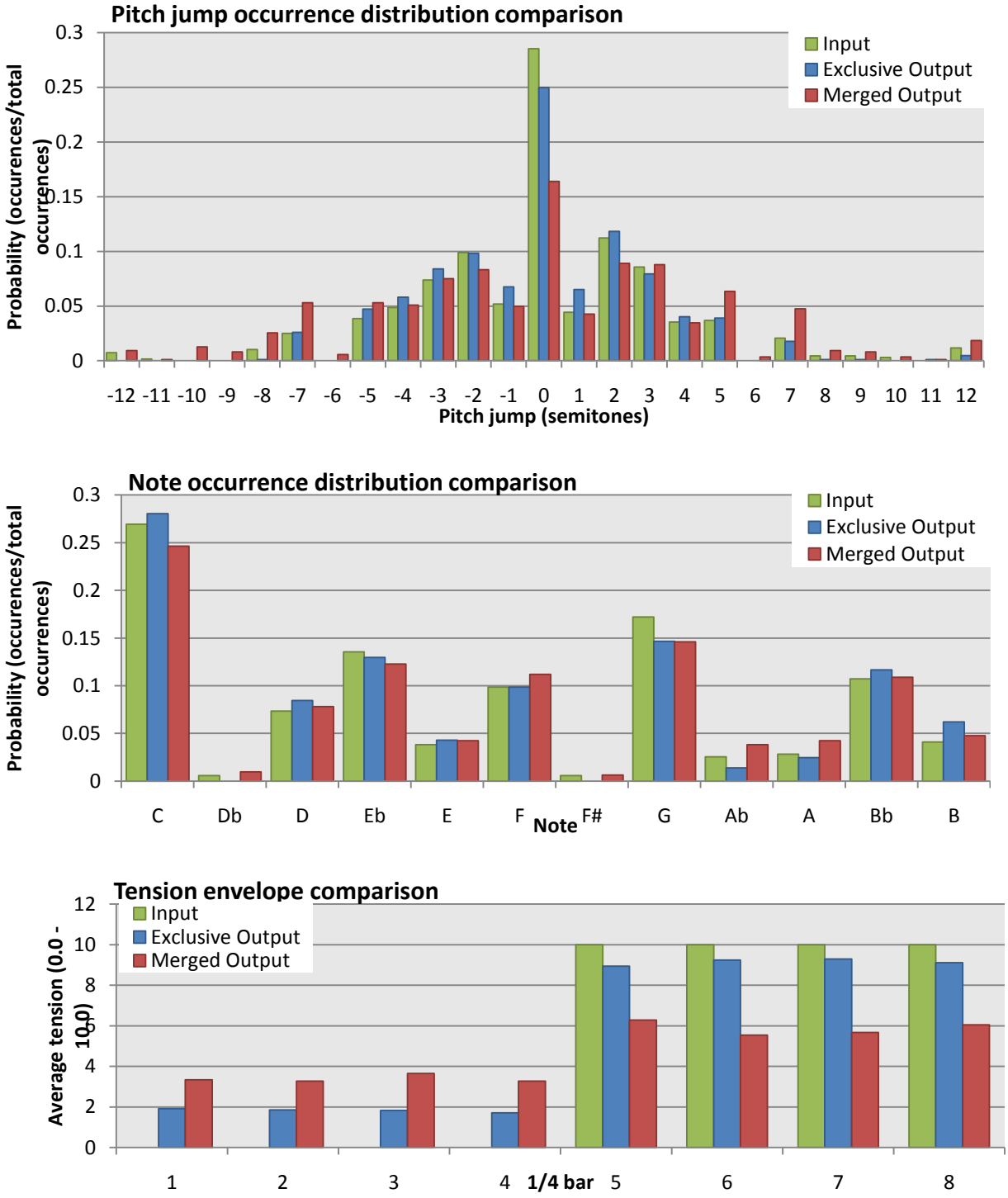


Figure 18 - Comparison of input/output distributions with varying criteria implemented in fitness function

8.2 SURVEY

The literature review earlier highlighted the lack of a formal test for the music produced by algorithmic composition systems. The solution used here jumbled up a selection of melodies produced by the MMG with melodies produced by a random composition process. This mixture of melodies was presented to a number of subjects, who assigned each melody a value **(1 *awful* – 5 *great*)** according to their enjoyment of that melody. Data was then analysed to discover how

the MMG's output compares with random music in terms of listening enjoyment. It is envisaged that variants on this scheme could be used for other algorithmic systems.

SURVEY DETAILS

Survey aim: To determine the level of enjoyment of the MMG's output, relative to random melodies.

The same melody set was presented to **ten subjects.** The set consisted of four 'preparation' melodies and **thirty 'tested' melodies** – where the first four were included to give participants something to base subsequent ratings on. Every melody was two bars in length, and was played twice (sequentially) to allow time for a judgement to form. The result was a series of **four bar melodies** with a rest of one bar in between, during which the number of the next melody was stated. Playback was at **117bpm** and assisted by a 4/4 metronome to provide an audible time reference. Survey's lasted six minutes. The **instrument voice** used was a piano, chosen for its neutral melodic timbre.

The microtonal aspect of the MMG has not been tested – it was presumed that a layperson ears are not accustomed to tunings other than 12-TET would not be able to assess microtonal melodies accurately. Therefore test melodies were all in **12-TET**.

As mentioned, the melody selection under survey was a mixture of random and MMG produced melodies. Further to this, the independence of rhythm and pitch in the MMG allowed four types of melodies to be produced:

Random pitches and random rhythm	*7 instances*
Random pitches and MMG produced rhythm	*8 instances*
MMG assigned pitches and random rhythm	*8 instances*
MMG assigned pitches and MMG produced rhythm	*7 instances*

These melodies were sorted into an order determined by a random number generator. It resulted in the number of instances for each type as stated.

RANDOMISED MELODY

Following are explanations as to how the randomised melody components were generated for the survey. The settings used for the MMG are then communicated.

RANDOMISED PITCH **–** Pitches were constrained to the allowed pitch range (corresponding to the MMG settings), and also to the scale of 'C minor'. These legitimate pitches were then assigned randomly to a note-on structure.

RANDOMISED RHYTHM – The average number of notes to be contained in an MMG produced melody was calculated as twelve. Each 1/16th of the two bar random rhythm therefore had a note beginning on it with a probability of 32/12 = 0.375.

MMG SETTINGS – During generation by the MMG, the tuning system was set as 12-TET, the scale within was chosen as 'C minor' and the allowed pitch range were MIDI notes 58-73 (A#3 – C#5). Density and tension envelopes were as in Table 10 (set as the authors' choice). The GA power coefficients were all set as one.

1/4 bar	Density	Tension
1	6	0
2	5	3
3	4	4
4	4	4
5	6	0
6	4	2
7	2	1
8	9	4

Table 10 - MMG set envelopes

RESULTS

Figure 19 shows the survey results where the mean for each of the thirty melodies (ten readings per melody) was calculated. This data was then used in calculating standard deviations for each melody type. *See appendix 12.3 for the survey data.*

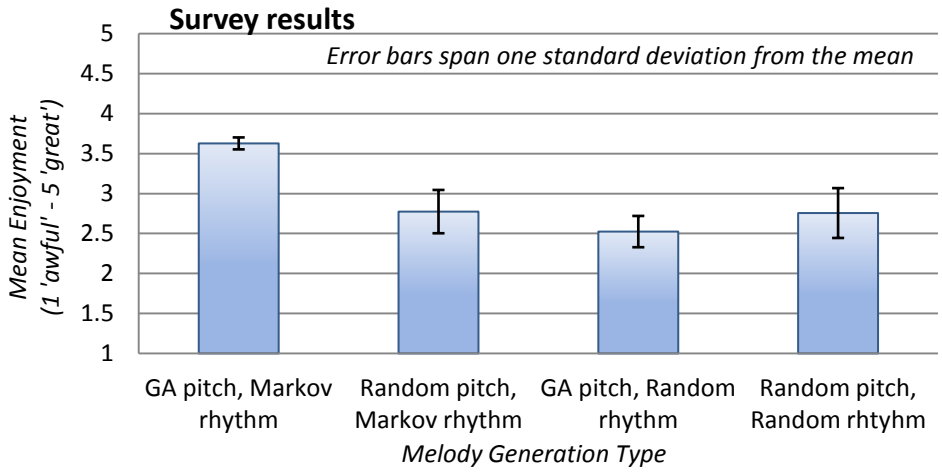


Figure 19 - Average survey rating for each melody generation type

The graph shows the MMG at full functionality to produce the highest rated melodies. The other types show little variation from eachother. The standard deviations show how ratings appear more consistent for full functionality compared with the completely random melodies.

What the graph does not show is that the melody with the highest mean was in fact produced by the random pitch/random rhythm method (rating of '4'). The 2nd,3rd,4th and 5th top rated melodies were all produced using full functionality generation.

TWO-SAMPLE T-TEST

To determine whether the difference in means showed by Figure 19 were statistically significant, a two sample *t*-test at the 1% level (sec. 5.5) was conducted as follows:

H_0 ; Survey means for full functionality and completely randomly produced melodies are equal

H_A ; Survey mean for full functionality is higher than for completely randomly produced melodies

DOF's = 6.69, 1% level, $t\ critical(one\ tail) = 3.04$

t test statistic: $t = 3.96$

$t > t\ critical$; This implies that H_A is likely. Of course, this is subject to fair survey conditions and assumptions as described in sec. 5.5. However there is sufficient reason to reject H_0 and state: **survey mean for full functionality is higher than for completely random produced melodies.**

9 DISCUSSION

In review, it can be seen that this projects main challenge has been in applying engineering methods to a subject that traditionally shirks rigid formalisms. The **statistical analysis** seemed to show that few rigid patterns exist, with the majority of explored relationships being deemed insignificant. This could indicate either a lack of importance for those particular relationships tested, or else that variations occur on a case by case basis rather than globally for all melodies. If the latter, then perhaps small groups of melodies appearing to share some similar trait should be analysed rather than using a global approach as in this report. Given the range of tests performed, it seems likely that more patterns would have been discovered if global trends *were* of a larger importance.

Microtonality has taken a sideline in this project. The motivation for creating the MMG was to create a system capable of composing with any note frequency; however it quickly became clear that a number of tuning systems must be mastered before this can occur –their compositional rules must be understood and then combined to be extrapolated to any note frequency. The program is currently capable of operating with microtonal systems, but the requirement of a manual input for the distributions seems to negate much use it has in this role. Some way of compiling distributions automatically is needed before it is useful to a majority.

Nothing has come to light throughout the project to make the **musical assumptions** made in sec. 3.4 be deemed inappropriate, and so the basis for making them is maintained as sound.

The **survey** conducted was encouraging to a degree, demonstrating that the MMG usually produces melodies of a more enjoyable nature than those created by a random generation. However, the top rated melody was produced in the latter way. This is worrying because there are many applications where only the top rated melody from a selection would be of use. This largely negates the consistency provided by the MMG which is perhaps expected given its heavy

reliance on global statistics. The consistency of MMG at full functionality meant it was a surprise that the mixed functions did not fare better than they did in the survey, and no conclusions could be drawn about the MMG's rhythm Vs pitch assignment importance. It can only be suggested that not enough melodies were surveyed. Confidence in the survey is considered reasonable though would benefit from a larger participant number and also a larger quantity of melodies tested.

Melody Contour

Whilst the produced melodies seemed to perform reasonably against random music, sights should really be set on competing against human composed tunes. At this stage, it would not have been helpful – on listening to output melodies, though pleasing to a degree they were undeniably lacking compared to those composed by a person. The reason for this deficit is hard to pinpoint, however on reflection, it was hypothesised that the MMG melodies had a lower **sense of order** compared to those human produced – i.e. the recognition of a pattern was much easier in the human melodies than for the MMG's output.

The clearest way to see this is to visually compare the pitch contours in the following figures which take three examples of melodies performing worst in the survey, followed by those three performing best, followed by three melodies input existing in the library subfolder.

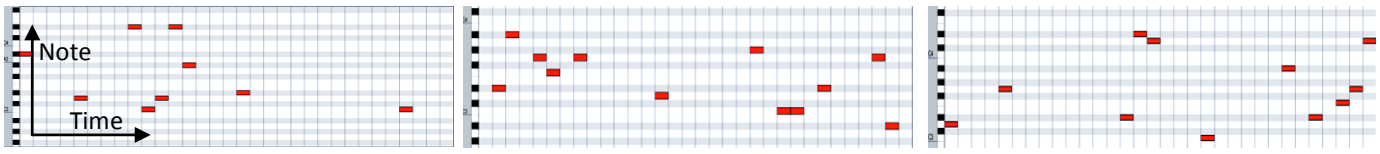


Figure 20 - Three melodies placing bottom in the survey

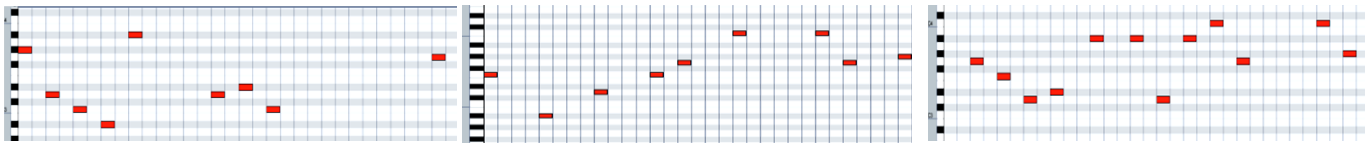


Figure 21 - Three most popular melodies produced by the MMG in the survey

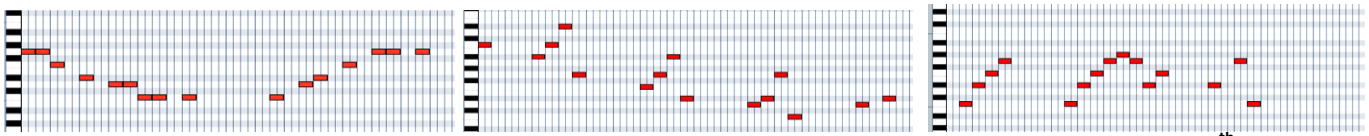


Figure 22 - Three human composed melodies from the used library (notes stunted to two 1/16th's)

The reader is invited to make their own observations, however the increasing sense of order as melodies become more popular seems undeniable –the unordered noting of melodies in Figure 20 becomes more well-ordered in Figure 21 and even more so in Figure 22. For example the ascending triplet in the middle melody of Figure 22 is repeated three times in the same formation, transposed to various pitches. The defining difference between the MMG melodies and those from recent pop hits is therefore hypothesised to be this sense of order.

The focus is then on how to reproduce this property found in human composed melodies. Several possible ways of accomplishing this are now proposed. Once this has been done to a sufficient level, re-evaluation would be required to test the hypothesis.

One way is to relinquish responsibility for the pitch contour by requesting the user for an envelope. Combined with the existing envelopes, this is a step toward turning the MMG into a composition assistant rather than automatic composer, with the output melody quality relying to some degree on the users input.

Alternatively, some form of pattern recognition could be included in the scan of the melody library. Patterns would have to be grouped by type – e.g. small clusters of notes repeated – and once discovered, coded into generation. Parallels should be drawn with; (Cope, 1992).

A substitute or add-on to this analysis approach could be a structured synthesis system – where a generated melody is composed of small units and built up using these. For example doubling a small phrase and transposing some of those notes.

It is important to note that the above suggestions are in addition to the MMG's current functions, which have not been proved unsuccessful but merely weak when used as the sole criteria for generation in comparison to what a human can produce. It takes the MMG about four seconds to produce a short melody whilst such a phrase may take a human minutes or even hours to perfect. All things considered, the MMG is currently useful as a tool to spark melody ideas, with its chief advantage being that it can output approximately twenty melodies in a minute – so the average quality doesn't have to be high provided something of interest is made occasionally. It could be argued that its lack of pattern makes it suited to this application.

Further Work

This project has seen plenty of groundwork put into the MMG and a solid foundation has been laid from which to proceed. Future efforts require focus on a few areas. One is in improving the 'sense of order' of 12-TET songs through methods suggested. Most likely coming after that, the issue of microtonal scales needs to be revisited, with the aim of coming up with a unified compositional theory governing all frequencies.

After these (or before should the envelope system be adopted) the creation of a user friendly format is required – specifically development of a GUI – to enable those other than the program author to use the MMG. A MATLAB toolbox or plug-in for Ableton live (computer composition software) are suggested.

10 CONCLUSIONS

This project has seen the development of an algorithmic melody composer 'MMG'. This system creates a rhythm for melody notes using a Markov chain which are then assigned pitches with a genetic algorithm. The criteria for the GA include comparison with a desired tension envelope and similarity to some statistical distributions. Information required for all functions is compiled from a melody library. The program was written in MATLAB and has been explained in full.

Technically, the project has been a success – the programme does all it has been coded to do with accuracy and efficiency. Desired musical distributions were shown to be met, the melody library was read successfully, MIDI files were stored without hitches, the Markov process was effectively implemented and a practical method for microtonal output was created. The melody generation process is faster than needed for real-time applications.

The MMG has been successfully designed and developed to operate with any microtonal scale, where the functionality level used is dependent on the tuning system. It was acknowledged that jumping straight to an infinite range machine as in the outset aim was perhaps too ambitious, though the option for development in this direction has been made possible with the rhythm of melodies being independently produced, along with an integrated dissonance system returning the tension for any frequency note.

After identifying in past projects the lack of formal melody rules which were based on more than personal intuition, a statistical investigation of melody was conducted, with the aim of determining underlying trends and relationships guiding melody. The two discoveries deemed significant enough for coding in generation were note occurrence and pitch jump size occurrence. The lack of discovered global trends should be heeded in future work.

Another weakness of past literature on algorithmic composers was seen to be the lack of a formal testing system. A survey comparing the systems output against random music has been proposed and used, showing the MMG to consistently produce more enjoyable melodies than that randomly produced, though not without exception.

In essence, the bases for the MMG's functions have been shown as sound, and the implementation of these was successful, however the resultant melodies still seemed poor relative to those composed by a human. A key difference between the MMG and human composed melodies was highlighted in response, and suggestions regarding this were made.

11 BIBLIOGRAPHY

Alpern, A. (1995). Techniques for Algorithmic Composition of Music.

Ariza, C. (n.d.). Retrieved January 2010, from algorithmic composition resources website: http://www.flexatone.net/algoNet/

AutoGam.com. (n.d.). *Automatic Music MIDI Generator*. Retrieved January 2010, from http://autogam.free.fr/a_index.htm

Baker, D. (1988). jazz improvisation.

Biles, J. A. (n.d.). *GenJam Website*. Retrieved January 2010, from http://www.ist.rit.edu/~jab/GenJam.html

Biles, J. A. (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos.

Biles, J., Anderson, P., & Loggi, L. (1996). Neural network fitness functions for a musical IGA.

Bozkurt, B. (n.d.). *Laconicism website* . Retrieved January 2010, from http://www.batuhanbozkurt.com/

C. L. Krumhansl, E. J. (1982). Tracing the Dynamic Changes in Perceived Tonal Organization in a Spatial Representation of Keys. *Psychological Review, vol. 89, no. 4* , 334-368.

Collins, N. (2008). Infno: Generating Synth Pop and Electronic Dance Music On Demand. *in Proc. of The International Computer Music Conference (ICMC)* .

Conklin. (2003). Music generation from statistical models.

Conklin, D., & McCleary. (1988). Modelling and generating music using multiple viewpoints.

Cope, D. (1992). A computer model of music composition. In S. S. Levitt, *Machine Models of Music* (p. 403). Cambridge: MIT Press.

Cope, D. (n.d.). *Experiments in musical intelligence website*. Retrieved December 2009, from David Copes Website: http://artsites.ucsc.edu/faculty/cope/experiments.htm

Dr.M. (2007). *Madplayer website*. Retrieved April 2010, from Dr Mad: Funk House Electro: http://www.doctor-mad.com/DrMadMusic/DrMadMusic.htm

Engineering_statistics_handbook_website. (n.d.). *Chi Square test*. Retrieved April 2010, from Engineering statistics handbook_NIST/SEMATECH: http://www.itl.nist.gov/div898/handbook/index.htm

Eno, B., & Chilvers, P. (n.d.). Retrieved January 2010, from Bloom|GenerativeMusic.com: http://generativemusic.com

F.P. Brooks, A. H. (1992). An experiment in musical composition. In S. S. Levitt, *Machine Models of Music* (p. 23). Cambridge,: MIT Press.

Goldberg, D. E., & Horner, A. (1991). Genetic algorithms and computer-assisted music composition. *Proceedings of the 1991 International Computer Music Conference*, (pp. 479—482).

Guedes, C., & Van Ransbeeck, S. (2009). Stockwatch, a Tool for Composition with Complex Data. *Parsons Institute for information Mapping* .

Handbook_of_biological_statistics_website. (2009). *G-test of independence*. Retrieved April 2010, from Handbook of biological statistics website: http://udel.edu/~mcdonald/statgtestind.html#chivsg

Hiller, L., & Isaacson, L. (1957). Musical composition with a high speed digital computer, 1958. In S. S. Levitt, *Machine Models of Music* (p. 7). Cambridge: MIT Press.

Horowitz, D. (1994). Generating Rhythms with Genetic Algorithms.

Jacob, B. L. (1995). Composing with genetic algorithms. *Proceedings of the International Computer Music Conference.* Banff Alberta.

Keith, S. (2009). Controlling Live Generative Electronic Music with Deviate. *NIME* .

Kon, F., & Iazzetta, F. (1995). Max Annelaing a tool for Algorithmic composition based on simulated annesaling. *Proceedings of the Second Brazilian Symposium on Computer Music.*

Levitt, D., & Schwanauer, S. (1992). *Machine Models of Music.* Cambridge: MIT Press.

McCormack, J. (1996). Grammar Based Music Composition.

Noatikil-Website. (n.d.). *Noatikil users guide*. Retrieved January 2010, from http://www.intermorphic.com/tools/noatikl/doc/intermorphic_noatikl_UserGuide.pdf

Oosterhoff, J. (2005). Retrieved April 2010, from Joanne Oosterhoff Music: http://joanne.oosterhoff.info/PDF/Overtones.pdf

Rader, G. (1992). composing simple traditional music by computer. In S. S. Levitt, *Machine Models of Music* (p. 259). Cambridge: MIT Press.

Rene Wooller, ,. (2007). Techniques for automated and interactive note sequence morphing of mainstream electronic music.

Sethares, W. A. (1993). Local consonance and the relationship between timbre and scale. *Journal of the Acoustical Society of America* .

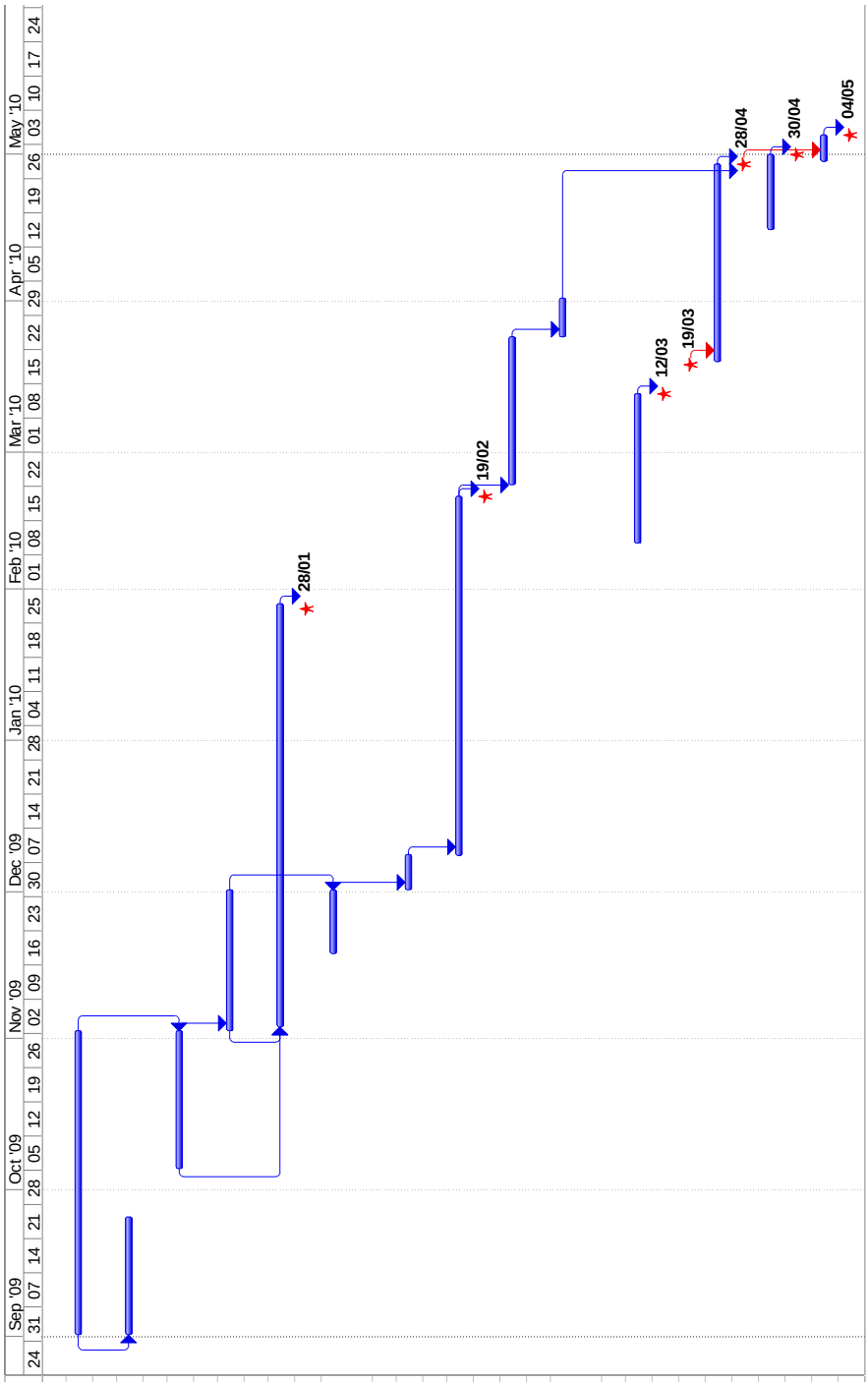
The-Just-Intonation-Network. (2008). Retrieved April 2010, from The Just Intonation Network: http://www.justintonation.net/

Werner, P. M. (1998). Frankensteinian Methods for Evolutionary Music Composition. MIT Press/Bradford Books.

Xenakis, I. (1971). *Formalized Music: Thought and Mathematics in Composition.*

12 APPENDICES

12.1 GANTT CHART



ID	Task Name	Duration	Start
1			
2	Broad research	39 days	Tue 01/09/09
3			
4	Programming familiarisation	16 days	Tue 01/09/09
5			
6	Project definition	18 days	Mon 05/10/09
7			
8	Focused research	18 days	Mon 02/11/09
9			
10	Literature review	55 days	Tue 03/11/09
11	Literature review deadline	0 days	Thu 28/01/10
12	Method of implementation selection	8 days	Wed 18/11/09
13			
14	Programme definition	5 days	Tue 01/12/09
15			
16	Programme Development	47 days	Tue 08/12/09
17	Computational task deadline	0 days	Fri 19/02/10
18	Programme evaluation	20 days	Mon 22/02/10
19			
20	Project analysis & future expansion	5 days	Wed 24/03/10
21			
22	Draft report	20 days	Wed 10/02/10
23	Draft report deadline	0 days	Fri 12/03/10
24	Draft report returned	0 days	Fri 19/03/10
25	Final report compilation	25 days	Fri 19/03/10
26	Project deadline	0 days	Wed 28/04/10
27	Poster design	10 days	Thu 15/04/10
28	Poster deadline	0 days	Fri 30/04/10
29	Oral preparation	3 days	Thu 29/04/10
30	Oral exam	0 days	Tue 04/05/10

12.2 PROGRAMME '.M' FILES

microspine .m

```
% Tim Pearce - Durham University - 2010

%
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% creates MIDI file of an algorithmically composed (2 bar) melody
% a.
% user specifies parameters for the melody - tuning system, density
% envelope, tension envelope, range selection, G.A. coefficients. Also
% dependent on scale needs note occurrence and pitch jump occurrence
% distributions
% b.
% generates a series of 1/16 bars for note on's. via a Markov process by
% analysing sub folder (/library) containing midi melodies. Then alters
% this according to density envelope
% c.
% assigns pitches to this rhthym structure using a genetic algorithm - uses
% fitness fcn utilising: tension envelope and note/pitch interval
% occurence distributions
% d.
% converts the midi matrix to a MIDI file and saves
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
clear all;
% ========================================================================
% USER INPUT REQUIRED
% ========================================================================
%
% specify tuning system/scale ---------------------------------------------
% -------------------------------------------------------------------------
% first column = order of tones
% second column = midi note number (doesnt have to be integers)
% third column = specify which notes to use in scale
% where: 0 = not in scale, 1 = used in scale (e.g. major/minor)
% N.B. first row value is taken as the tonic (root) note
% micro assigns what type of scale is used so programme runs correct
% functions.
% 0 = 12-TET            includes all functions
% 1 = N-TET             requires manual input of distrubutions
% 2 = unequal tempered  req. manual note prob dist, and doesnt use pitch j
global tuningsystem MicroType;


MicroType = 0;
% midi no. of each note in scale
tuningsystem =   [...           %12-tet
   1     2     3     4     5     6     7     8     9    10    11    12;...
  60    61    62    63    64    65    66    67    68    69    70    71;...
   1     0     1     1     0     1     0     1     1     0     1     0]';
%   c    c#    d     eb    e     f     f#    g     ab    a     Bb    B

%{
MicroType = 1;
tuningsystem =   [...           % 19-tet
  1  2  3  4  5  6  7  8  9 10 11 12 13  14  15  16  17  18  19;...
 60.0  60.63  61.26  61.89  62.53  63.16  63.78  64.42  65.05 65.68 ...
 66.31  66.94 67.57 68.21   68.84   69.47  70.10   70.73   71.36;...
  1  0  1  1  0  1  1  0  0  1  0  0  1   0   1   0   0   0   1]';
%}
```

```
% note occurrence distribution ------------------------------------------
% -----------------------------------------------------------------------
% user can input this manually if a microtonal scale selected, otherwise
% compile in libraryanalysis.m
global notefreqcount;
if MicroType ~= 0   % only manually input if microtonal scale selected
  notefreqcount = [...
     1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19;...
   100 10 20 50 70 50  70 5 60  20  10  70  30   5  20   5  10  30  10]';
end


% interval occurrence distribution----------------------------------------
% -----------------------------------------------------------------------
% user can input this manually if a microtonal scale selected, otherwise
% compile in libraryanalysis.m
global pjumpfreqcount;
if MicroType ~= 0   % if microtonal scale selected
  pjumpfreqcount =[...
   0  1  2  3  4  5  6  7  8  9 10 11 12 13  14  15  16  17  18  19;...
 100 20 30 40 30 28 26 20 18 50 30 20 40 15  10   8   8   5   2  10]';

  % mirror pitch jumps & quantities for -ve pitch jumps
  pjumpfreqcount = [flipud(pjumpfreqcount(2:size(pjumpfreqcount,1),:));...
                    pjumpfreqcount];
  pjumpfreqcount(1:floor(size(pjumpfreqcount,1)/2),1) = ...
                    -pjumpfreqcount(1:floor(size(pjumpfreqcount,1)/2),1);
end


% density envelope -------------------------------------------------------
% -----------------------------------------------------------------------
% sets how the number (density) of notes changes over the (2 bar) melody
% each row represents a 1/4 bar as indicated by the first column
% values must be between 10.0 (lots of notes) and 0.0 (few notes)
global densityenv;
densityenv =  [...
     1     2     3     4     5     6     7     8;...
     6     5     4     4     6     4     2     9]';

% tension envelope -------------------------------------------------------
% -----------------------------------------------------------------------
% sets how the tension level of the melody varies over the 2 bars
% each row represents a 1/4 bar as indicated by the first column
% values must be between 10.0 (very dissonant) and 0.0 (very consonant)
% 10.0 corresponds to most dissonant out of notes allowed
global dissenv;
dissenv =  [...
     1     2     3     4     5     6     7     8;...
     0     5     4     4     0     4     2     9]';

% range selection --------------------------------------------------------
% -----------------------------------------------------------------------
% sets the maximum and minimum pitches that are allowed to be considered
% in generation
global lowestpitch highestpitch;
lowestpitch = 58;
highestpitch = 73;
```

```
% Genetic Algorithm power coeff. ---------------------------------------
% ----------------------------------------------------------------------
% vary the (relative) effect of each criteria in fitness function
% values may be as large as desired, set as zero to negate the criteria
global tpow npow jpow;
npow = 3; % larger value = a closer match to note prob distribution
jpow = 2; % larger value = a closer match to note pitch jump distribution
tpow = 1; % larger value = a closer match to desired tension envelope
% N.B. by increasing the value of one criteria, the influence of all others
% are effectively reduced

% ======================================================================
% USER INPUT COMPLETE
% could add in some checks here to verify data is input correctly
% ======================================================================

% 1. compile information for tuning sytem selected ----------------------
% ----------------------------------------------------------------------
% calc. frequency(Hz) of MIDI notes
tuningsystem(:,4) = midi2freq(tuningsystem(:,2));
% calc. dissonance(arbitary scale) of frequencies (compared to root freq.)
tuningsystem(:,5) = freq2diss(tuningsystem(:,4));

% 2. collect information from melody library ----------------------------
% ----------------------------------------------------------------------
% call the library analyser to compile; note freq. count, pitch jump
% freq. count, and rhythm transition matrix
global transition firstnoteon firstinterval;
[notefreqcount, pjumpfreqcount...
        transition, firstnoteon, firstinterval] = libraryanalysis;

% 3. create rhythm structure --------------------------------------------
% ----------------------------------------------------------------------
% returns note-on structure (1/16ths) for 2 bars using a markov chain
global noteons;
noteons = markovrhythm;


% 4. pitch assignment ---------------------------------------------------
% ----------------------------------------------------------------------
% assigns pitches to the note-on structure using a genetic algorithm
notepitches = pitchassign;


% 5. output midi file ---------------------------------------------------
% ----------------------------------------------------------------------
% convert to midi matrix form as recognised by 'writemidi.m'
M = zeros(size(noteons,1),8);       % create matrix
M(:,1) = 1;                         % all in track 1
M(:,2) = 1;                         % all in channel 1
M(:,3) = notepitches(:,1);          % note midi pitch (may be non-integer)
M(:,4) = 100;                       % velocity   (all = 100)
M(:,5) = noteons(:,1)/8;            % time note on (seconds @120bpm)
M(:,6) = (noteons(:,1)+1)/8;        % time note off (duration = 1/16th)
% round midi pitch and add pitch bend:
[M(:,3), M(:,7), M(:,8)] = freq2midipb(midi2freq(M(:,3)));
M   % print info in command window for the MIDI file

savefilename = 'midiout.mid';
midi_new = matrix2midi(M);          % convert to saveable form
writemidi(midi_new, savefilename);  % save midi file
```

freq2diss .m

```
% William Sethares - http://eceserv0.ece.wisc.edu/~sethares/comprog.html
%
% adapted (with permission) by Tim Pearce - Durham University - 2010

function diss = freq2diss(freq)

% given a frequency(Hz), returns dissonances relative to first freq in the
% vector

% first specify harmonic content of a standard timbre
basefreq = freq(1,1)*[1 2 3 4 5 6];     % all dissonances relative to first
                                        % note in vector
amp = ones(size(basefreq));
for i = 1:size(basefreq,2)
  amp(1,i) = amp(1,i)*(0.88^(i-1));     % give amplitude of harmonics a
end                                     % decay rate of 0.88

for i = 1:size(freq,1)
  % base frequency, compared to new freq.
  f = [basefreq, freq(i,1)*[1 2 3 4 5 6]];
  a = [amp, amp];

  % returns dissonance value for freq compared to ratio*freq
  diss(i,1) = dissmeasure(f, a);
end


function d=dissmeasure(fvec,amp)

% given a set of partials in fvec,
% with amplitudes in amp,
% this routine calculates the dissonance

Dstar=0.24; S1=0.0207; S2=18.96; C1=5; C2=-5;
A1=-3.51; A2=-5.75; firstpass=1;
N=length(fvec); % no. harmonics
[fvec,ind]=sort(fvec);
ams=amp(ind);
D=0;
for i=2:N  % no. of harmonics add
  Fmin=fvec(1:N-i+1);
  S=Dstar./(S1*Fmin+S2);
  Fdif=fvec(i:N)-fvec(1:N-i+1);
  a=min(ams(i:N),ams(1:N-i+1));
  Dnew=a.*(C1*exp(A1*S.*Fdif)+C2*exp(A2*S.*Fdif));
  D=D+Dnew*ones(size(Dnew))';
end
d=D;
```

libraryanalysis .m

```
% Tim Pearce - Durham University - 2010


function [notefreqcount, pjumpfreqcount...
          transition, firstnoteon, firstinterval] = libraryanalysis

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% inputs:
% a hoard of midi files in a sub-folder
%
% outputs:
% returns matrix of note and pitch jump occurrence distributions and
% transition matrix (for note rhythms). Also matrices of info regarding
% timing of first note (to initiate later Markov process)
%
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

% name of subfolder in directory w/ .mid files
subfol = 'library\';
list = dir(fullfile(subfol,'*.mid'));       % lists all the .mid file names

global grand;
grand = [];                                 % main matrix


for i = 1:size(list)                        % open each midi file in turn
  % get note information for each midi file
  [midmat,endtime] = midiInfo(readmidi([cd, '\', subfol,list(i).name]),0);

  midmat(:,5:6) = round(midmat(:,5:6));     % quantizes note on/off
  endtime = ceil(endtime);                  % rounds up endtime

  grand = [grand;...
           midmat(:,3)...                   % midi note no.
           midmat(:,5)...                   % 1/16th note on
           zeros(size(midmat,1),1)];    % midi note no. 1-12 (workoutlater)
end




% work out the unfinished columns of grand
global MicroType;
if MicroType == 0       % only bother working out if 12-TET
  for i = 1:size(grand,1)
    % calc. midi note no. 1-12
    m = 0; L = 12;      % convert from midi no. to 1-12 (c is 1, b is 12)
    while L>=12
      L = grand(i,1) - m;
      m = m + 12;
    end
    grand(i,3) = L+1;
  end
end
```

```
% ONLY DO IF MICROTONAL SCALE NOT SELECTED
% cumulative freq. of each note - - - - - - - - - - - - - - - - - - - - - - - -
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
global notefreqcount pjumpfreqcount;
if MicroType == 0       % only if 12-TET
  notefreqcount = [];
  for i = 1:12
   row = find(grand(:,3) == i);
   notefreqcount = [notefreqcount; i, size(row,1)];
  end
end
% could adjust here for those notes only found in maj or min scale (if
% library contains more of one type)



% ONLY DO IF MICROTONAL SCALE NOT SELECTED
% cumulative freq. of each pitch jump - - - - - - - - - - - - - - - - - - - - -
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% first find pitch jump for each interval
if MicroType == 0       % only if 12-TET
  pj = [];
  for i = 2:size(grand,1) % for all potential intervals
    % check for if note is first of song
    if grand(i,2)>grand(i-1,2)   % if later than last note
      pj = [pj; grand(i,1)-grand(i-1,1)];% save as pitchjump from last note
    end
  end

  % now count how many times each pitch jump occurs
  pjumpfreqcount = [];
  for i = min(pj):max(pj)
    [row,col] = find(pj(:,1) == i);
    pjumpfreqcount = [pjumpfreqcount; i, size(row,1)];
  end
end



% transition matrix - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% compile transition matrix for use in Markov process to create note rhythm
% structure. Also compile information for the first note of melodies
%
% for each note interval, must look at what each note separation is, and
% then the proceeding division size. add each instance to cumulative
% counter (effectively transition probability matrix)
%
% transition matrix has current states in rows, and next state in cols
% states go up in 1/16 graduations

maxint = 16;                    % max gap recognised by transition matrix
transition = zeros(99, 99);     % make excessively big at first
% matrices to do with first note
fno = []; fi = [];                      % preliminary matrices
firstinterval = []; firstnoteon = [];   % actual matrices
```

```
currentint = grand(2,2) - grand(1,2);       % for first interval
for i = 3:size(grand,1) % for all potential intervals
  % check for if note is first of song (time on is earlier than prev note)
  if grand(i,2)>=grand(i-1,2)   % if not first note in song
      nextint = grand(i,2) - grand(i-1,2);
      if nextint ~= 0 &&  currentint ~= 0  % as long as 2 notes not at once
     transition(currentint, nextint) = transition(currentint, nextint) + 1;
      end
      currentint = nextint; % old becomes new for next iteration
  else                            % if first note in song
    fno = [fno; grand(i,2)];                % store time of first note on
    currentint = grand(i+1,2) - grand(i,2);
    fi = [fi; currentint];                  % store first time interval
    i = i+1;                    % fast-forward a step
  end
end


% order the 'first note timing' - - - - - - - - - - - - - - - - - - - - - -
for i = 0:max(fno)
  amount = find(fno(:,1)==i);
  firstnoteon = [firstnoteon; i, size(amount,1)];
end
% now must make so the column totals to one (as for transition matrix)
total = sum(firstnoteon(:,2));
firstnoteon(:,2) = firstnoteon(:,2)/total;
for i = 2: size(firstnoteon,1) % this orders row values so -> 1.0
  firstnoteon(i,2) = firstnoteon(i-1,2) + firstnoteon(i,2);
end


% order the 'first note interval' - - - - - - - - - - - - - - - - - - - - -
for i = 1:max(fi)
  amount = find(fi(:,1)==i);
  firstinterval = [firstinterval; i, size(amount,1)];
end
% ignore intervals greater than maxint
firstinterval = firstinterval(1:maxint,:);
% now must make so the column totals to one
total = sum(firstinterval(:,2));
firstinterval(:,2) = firstinterval(:,2)/total;
for i = 2: size(firstinterval,1) % this orders row values so -> 1.0
  firstinterval(i,2) = firstinterval(i-1,2) + firstinterval(i,2);
end


% order the 'transition matrix' - - - - - - - - - - - - - - - - - - - - - -
% ignore intervals greater than maxint
transition = transition(1:maxint,1:maxint);
% now must make so each row in transition totals to 1
for i = 1:size(transition,1)
  total = sum(transition(i,:));
  if total ~= 0         % this avoids NaN's
    transition(i,:) = transition(i,:)/total;
  end
  for j = 2: size(transition,2) % this orders row values so -> 1.0
    transition(i,j) = transition(i,j-1) + transition(i,j);
  end
end
```

```
% Tim Pearce - Durham University - 2010


function [noteons] = markovrhythm;


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% generates timings of notes-on (pitches not assigned) for a 2 bar melody
% via Markov process and adjust according to a density envelope
%
% inputs:
% firstnoteon           (time and probabilities of first note in melody)
% firstinterval         (size and prob of first interval in melody)
% transition matrix     (rows = current state, cols = next state)
% density envelope      (1/4 bar bins with values relating to how many
%                       notes are desired in each)
%
% outputs:
% a vector of times (1/16ths) corresponding to notes being triggered 'on'
%
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


% markov chain ------------------------------------------------------------
% -------------------------------------------------------------------------
% create a note on rhythm structure using transition matrix prev. compiled
% in a markov process
global transition firstnoteon firstinterval;




% beginning ---------------------------------------------------------------
% to initiate: choose when first note should be & first interval
x = rand;
firstn = 1;             % default value - incase next unassigned below
for i = 1:size(firstnoteon,1)
  % using random number, find where falls in transition matrix
  if x < firstnoteon(i,2)
    firstn = i-1; break;
  end
end




x = rand;
firsti = 1;
for i = 1:size(firstinterval,1)
  % using random number, find where falls in transition matrix
  if x < firstinterval(i,2)
    firsti = i; break;
  end
end
```

```
% rhythm generation ------------------------------------------------------
noteons = firstn;       % initiate with calculated first note time
current = firsti;       % initiate with calculated interval
while sum(noteons) < 31 % generate until (over) two bars are completed
  x = rand;
  next = 1;
  for i = 1:size(transition,1)
    % using random number, find where falls in transition matrix
    if x < transition(current,i)
      next = i; break;
    end
  end
  noteons = [noteons; next];
  current = next;
end
noteons = noteons(1:size(noteons,1)-1,:);   % delete last note

% now tally the intervals
for i = 2:size(noteons,1)
  noteons(i) = noteons(i) + noteons(i-1);
end


% adjust notes according to the density envelope --------------------------
% -------------------------------------------------------------------------
global densityenv;

for i = 1:size(densityenv,1)                    % for each 1/4 bar segment
  for j = ((i-1)*4):2:((i-1)*4) + 3             % step through in 1/16ths
        % if need to reduce the amount of notes
    if densityenv(i,2) < 5 & isempty(find(noteons == j)) == 0
      if densityenv(i,2) < 5* rand          % probability of removing note
        noteons(find(noteons == j),:) = []; % remove note
      end
    end
        % if need to increase the amount of notes
    if densityenv(i,2) > 5 & isempty(find(noteons == j)) ~= 0
      if densityenv(i,2) > 5* rand +5       % prob to add note
        noteons = [noteons; j];             % add note
      end
    end
  end
end

noteons = sort(noteons);                    % put in chronological order
```

pitchassign .m

```
% Tim Pearce - Durham University - 2010

function notepitches = pitchassign
% -------------------------------------------------------------------------
% sets up options for genetic algorithm and then calls it

global noteons lowestpitch highestpitch;

gmax = 1000;                    % max number of generations
scale = 4; shrink = 0.8;        % mutation parameters
nvars = size(noteons,1);        % no. of variables = no. of notes
LB(1:nvars,1) = lowestpitch;    % lower and upper bounds for variables
UB(1:nvars,1) = highestpitch;

global subtunsyst;
subtunsyst = tsrefmatrix;       % need to compile this for fitness fncn

% generate initial population - random values within center of pitch band
initialpop(1:size(noteons,1),1) = zeros;
for i = 1:size(initialpop,1)
  x = rand*(highestpitch - lowestpitch)/2 + ...
                        lowestpitch + (highestpitch - lowestpitch)/4;
  % find closest note in tuning system to random number
  [trash, array_position] = min(abs(subtunsyst(:,2) - x));
  initialpop(i,1) = subtunsyst(array_position,2);
end
initialpop = [initialpop]';     % change from row -> col vector

% put above selections into options
options = gaoptimset('Generations',gmax,'MutationFcn', ...  % std options
    {@mutationgaussian, scale, shrink},...
    'InitialPopulation',initialpop  ); %,...
 %
'PlotFcns',{@gaplotbestf,@gaplotbestindiv,@gaplotexpectation,@gaplotstopping});

% before entering G.A. adjust the note occurrence distribution according to
% which notes have been selected for use in the tuning system
global tuningsystem notefreqcount;
[row,col] = find(tuningsystem(:,3) == 0);
%notefreqcount(row,2) = 0;

global scorecounter;                                % #### useful for testing
scorecounter = [];                                  % ####

% call the g.a. - fitness assessment performed by pitchassign.m
[notepitches, fval] = ga(@fitnessfunc,nvars,[],[],[],[],LB,UB,[],options);
% where fval = fitness function score
notepitches = [notepitches]';   % change from row -> col vector

% must round notes to nearest in scale again
for i=1:size(notepitches,1)
  [trash, array_position] = min(abs(subtunsyst(:,2) - notepitches(i,1)));
  notepitches(i,1) = subtunsyst(array_position,2);
end
```

tsrefmatrix .m

```
% Tim Pearce - Durham University - 2010

function subtunsyst = tsrefmatrix
% -------------------------------------------------------------------------
% necessary before using gapitch.m
% returns matrix of all notes (in specified tuning system) that exist in
% allowable range
global tuningsystem; global lowestpitch; global highestpitch;

k = tuningsystem(1,2) - tuningsystem(size(tuningsystem,1),2);
numoctaves = 1;
while k > 12
  numoctaves = numoctaves + 1;  % finds the number of octaves for a scale
  k = k-12;                     % to repeat
end

% for above
subtunsyst = tuningsystem(:,1:2);
k=1;
while subtunsyst(size(subtunsyst,1),2) < highestpitch
  subtunsyst = [subtunsyst;...
                tuningsystem(:,1), tuningsystem(:,2)+(numoctaves*12*k)];
  k=k+1;
end

% for below
k=1;
while subtunsyst(size(subtunsyst,1)-size(tuningsystem,1)+1,2) > lowestpitch
  subtunsyst = [subtunsyst;...
                tuningsystem(:,1), tuningsystem(:,2)-(numoctaves*12*k)];
  k=k+1;
end
```

fitnessfunc .m

```
% Tim Pearce - Durham University - 2010

function score = fitnessfunc(pitch)
% -------------------------------------------------------------------------
% for use in genetic algorithm (this is fitness function) -----------------
% pitch = offspring
% increase variable 'score' according to how badly match criteria ---------
% no. variables = no. of notes

global tuningsystem dissenv noteons MicroType ...
                    notefreqcount pjumpfreqcount subtunsyst;
% begin by rounding pitch to a note in tuning system, then work out
% associated dissonance and join with time position in melody
pitch = [pitch]';       % change from row -> col vector
pitch(:,2:4) = zeros;   % fill up matrix

% closest pitch in scale ----
for i=1:size(pitch,1)
  [trash, array_position] = min(abs(subtunsyst(:,2) - pitch(i,1)));
  pitch(i,1) = subtunsyst(array_position,2);
  pitch(i,2) = subtunsyst(array_position,1);
end

% dissonance value ----
for i = 1:size(pitch,1)
  [row,col] = find(tuningsystem(:,1) == pitch(i,2));
  pitch(i,3) = tuningsystem(row,5);
end

% time of note-on ----
pitch(:,4) = noteons(:,1);

% first criteria - check notes according to note probability --------------
% -------------------------------------------------------------------------
% find expected hurt from note picking for the number of notes used
residual1 = [];
% chi-squared/residual method
for i = 1:size(tuningsystem,1)
  [row,col] = find(pitch(:,2) == tuningsystem(i,1));
  obsv = size(row,1)/size(pitch,1);
  expec = notefreqcount(i,2)/sum(notefreqcount(:,2));
  if expec ~= 0
    residual1(i,1) = abs(obsv - expec)^4/expec;
  else
    residual1(i,1) = abs(obsv - expec)^4/0.0001;
  end
end
score1 = 8*sum(residual1(:,1));

% second criteria - check notes according to note jump probability --------
% -------------------------------------------------------------------------
% chi-squared/residual method
score2 = 0;
if MicroType ~= 2   % only enter if scale selected is equal temperament
  residual = [];
  % work out pitch jump size for scale
  jumpsize = tuningsystem(2,2) - tuningsystem(1,2);
  % find actual pitch jumps present
  for i = 2:size(pitch,1)
    jump(i-1,1) = round((pitch(i,1) - pitch(i-1,1))/jumpsize);
  end
```

```
  for i = 1:size(pjumpfreqcount,1)
    [row,col] = find(jump(:,1) == pjumpfreqcount(i,1));
    obsv = size(row,1)/size(jump,1);
    expec = pjumpfreqcount(i,2)/sum(pjumpfreqcount(:,2));
    if expec ~= 0
      residual(i,1) = abs(obsv - expec)^4/expec;
    else
      residual(i,1) = abs(obsv - expec)^4/0.005;
    end
  end

  % incase of rogue pitch jumps (beyond compiled range)
  [row,col] = find(jump(:,1) > max(pjumpfreqcount(:,1))); % if too large
  for i = min(jump(row,1)):max(jump(row,1))
    [row2,col2] = find(jump(:,1) == i);
    obsv = size(row2,1)/size(jump,1);
    residual = [residual; obsv^4/0.0003];       % expec = 0 less likely than in
range desired - cld put in factor so further from 0, gets less likely
  end

  % same for if too small
  [row,col] = find(jump(:,1) < min(pjumpfreqcount(:,1)));
  for i = min(jump(row,1)):max(jump(row,1))
    [row2,col2] = find(jump(:,1) == i);
    obsv = size(row2,1)/size(jump,1);
    residual = [residual; obsv^4/0.0003];       % expec = 0
  end
  score2 = 3*sum(residual(:,1));    % was 0.1
end



% third criteria - check tension level of notes against tension envelope --
% ------------------------------------------------------------------------
score3 = 0;

global test;test = [];
biggest = max(tuningsystem(find(tuningsystem(:,3) == 1),5));

for i = 1:size(dissenv,1)  % cycle through in 1/4 bars
  % find notes in that 1/4 bar
  row = find( (noteons(:,1) >= (i-1)*4) & (noteons(:,1) < (((i-1)*4)+4)) );
  if isempty(row) == 0  % if have found notes
    % calculate mean dissonance
    avgdis = mean(pitch(row,3));
    % scale avgdis to envelope dissonance
    avgdis = avgdis* 10/biggest;

    test = [test; i avgdis];

    % check if mean dissonance is different to desired envelope
    if avgdis ~= dissenv(i,2)
      % score is increased by amount proportional to the difference
      score3 = score3 + 0.007 * norm(avgdis - dissenv(i,2));
    end
  end
end
```

```
% total score = sum of all components ------------------------------------
% ------------------------------------------------------------------------
% weight of each adjusted by user specified power coefficients
global npow jpow tpow;
score =  (npow*score1) + (jpow*score2) + (tpow*score3);


global scorecounter;                              % #### useful for testing
scorecounter = [scorecounter;score1, score2, score3, score];
```

freq2midipb.m

```
% Tim Pearce - Durham University - 2010

function [m, msb, lsb] = freq2midipb(f)

% Convert frequency, f, in Hz to MIDI note number (m=0-127) and values to
% bend by (if required) to achieve freq (f can also be a vector or matrix)
% Make sure pitch bend range on synth is set to +/- 2 semitones
% if not, then can alter line 15 (4096 = 1683/(PBR*2))
%
% returns [integer MIDI note, most sig. bend byte, least sig. bend byte]

m = 69 + 12*log2(f/440);              % finds closest midi value
difference = round(m) - m;            % find dif. between closest 12-TET
                                      % freq and actual desired  freq (Hz)
decvalue = round(8192 - (difference * 4096));   % = 14bit dec must bend by

% convert to two 7 bit decimal numbers
msb = bitshift(decvalue, -7);           % Most Significant Bit
lsb = decvalue - (msb*(2^7));           % Least Significant Bit

m=round(m);                             % nearest whole midi note
```

midi2freq.m

```
% Copyright (c) Ken Schutte 2009 - http://www.kenschutte.com/midi
%
% adapted (with permission) by Tim Pearce - Durham University - 2010

function f = midi2freq(m)
% Convert MIDI note number (m=0-127 - but may be non integer)
% to frequency, f, in Hz
% (m can also be a vector or matrix)

f = (440/32)*2.^((m-9)/12);
```

12.3 EVALUATION DATA

Survey - melody enjoyment rating data (1 awful - 5 great)

		Participant number										
melody	**type**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**mean**
1	**1**	4	3	4	3	2	3	3	4	3	3	3.20
2	**3**	3	3	1	3	1	4	4	2	3	2	2.60
3	**2**	2	3	3	4	1	2	2	3	2	3	2.50
4	**0**	2	4	2	4	3	5	4	3	4	4	3.50
5	**2**	2	2	2	2	2	4	4	3	3	2	2.60
6	**1**	4	3	2	3	3	3	3	2	4	2	2.90
7	**3**	2	2	3	4	2	4	2	2	2	3	2.60
8	**2**	3	2	3	2	3	5	2	2	2	4	2.80
9	**1**	4	3	1	1	1	1	4	1	1	3	2.00
10	**0**	5	4	3	4	4	1	4	4	3	4	3.60
11	**3**	4	3	3	1	4	3	3	4	3	3	3.10
12	**1**	3	2	4	2	4	4	2	2	2	2	2.70
13	**0**	4	2	4	4	5	4	4	3	3	5	3.80
14	**2**	3	1	3	3	3	1	2	3	1	1	2.10
15	**0**	2	3	4	4	3	2	5	3	3	5	3.40
16	**1**	3	4	1	2	2	3	2	2	2	3	2.40
17	**0**	2	3	4	3	5	4	4	4	3	5	3.70
18	**1**	4	4	3	4	3	4	4	2	3	3	3.40
19	**1**	2	2	3	2	1	3	2	3	2	2	2.20
20	**0**	5	4	4	5	4	2	4	4	2	4	3.80
21	**3**	4	4	5	4	4	5	4	3	3	4	4.00
22	**2**	3	2	4	4	2	3	3	4	2	3	3.00
23	**3**	3	1	4	2	2	3	2	2	3	3	2.50
24	**3**	2	1	4	3	1	2	3	1	2	2	2.10
25	**2**	2	1	2	2	2	3	1	2	2	1	1.80
26	**3**	4	1	2	1	3	2	4	3	2	2	2.40
27	**2**	2	1	3	1	4	4	2	4	3	3	2.70
28	**1**	3	2	4	4	2	4	3	5	3	4	3.40
29	**0**	4	3	5	4	3	3	4	2	4	4	3.60
30	**2**	3	3	2	1	4	2	2	4	4	2	2.70
										std dev		0.61
										mean		2.90

Where:

type	Generation type	mean	std dev
0	GA pitch, Markov rhythm	3.63	0.15
1	Random pitch, Markov rhythm	2.78	0.54
2	GA pitch, Random rhythm	2.53	0.39
3	Random pitch, Random rhtyhm	2.76	0.62