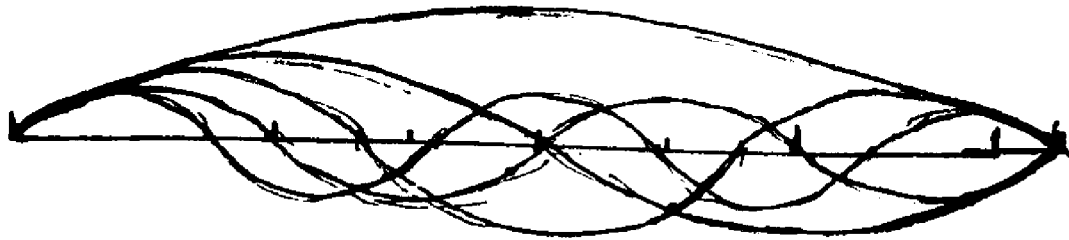


Data Science In The Creative Process

Composing With Functions

2018



Final Report



Oscar South

20/9/2018

info@oscarsouth.com

Dublin Business School -- 10382982
Higher Diploma: Science In Data Analytics

Supervised by Terri Hoare

Introductory

Abstract

With relevance to creativity in the design and implementation of data analysis systems, a range of literature including past work by the author is reviewed in order to provide a context and methodology for exploration as well as a variety of relevant mathematical, computational and conceptual models. Drawing on concepts established in the theoretical sections, the subject matter is implemented and explored in depth through a *Practice Based Research* methodology, inside which which operationally significant observations are discussed in depth. Useful analogy is developed between implementation of data analysis systems and models of conceptualised creativity.

Acknowledgments

I'd like to give my thanks to the following individuals, for the inspiration and/or support which they have provided to me with regard to the present work:

- Alex McLean, for his inspiring work on the TidalCycles music DSL for Haskell and related academic submissions.
- Michael Manring, for inspiring the content of my prior MA Thesis (which underpins the present work) through his lifelong exploration of instrumental technique.
- Terri Hoare, for her tutorage, wisdom, academic support, and for allowing me the creative freedom to pursue esoteric uses for Data Science methodology.
- Jason Roche, for facilitating the cross-practice collaboration I have engaged in with faculty of DBS and thus for the inspiring discussion throughout that work.
- Colin South, my brother, who I've spent many hundreds of hours discussing parallels and building analogy between music and computation.
- Saydyy Kuo South, my wife, through whose support and inspiration I have been able to pursue my present research goals.

Table Of Contents

<i>Title Page</i>	(i)
Introductory	1
Acknowledgements	1
Table Of Contents	2
Introduction	3
Theoretical	4
Practice Led Research	4
The Harmonic Algorithm	4
Markov Chain Models	4
Monadic Computation	5
Modelling Creative & Novel Behavior	5
Discussion	7
Motivation	7
Overview Of Functionality	8
'Main' Module	12
'MusicData' Module	15
'Markov' Module	18
'Overtone' Module	19
Closing Sections	20
Conclusion	20
References/Bibliography	21

Introduction

Data Science is commonly described as the crossover of the field of computation with that of numerical analysis. These technical areas are integrated with domain specific knowledge to facilitate the extraction of useful numerical insights from large amounts of domain relevant data.

The mainstream popularisation of any field of business interest is often brought on by the convergence of mass awareness in potential business application with mass accessibility to the required technology, tools or resources that allow a large number of individuals to become involved with the field at relatively low expense. The field of Data Science has now been popularised and elevated in general awareness for a number of years whilst access to powerful computation has dramatically dropped in price. The resulting mass engagement in a newly popularised field commonly results in a 'gold rush' in which individuals and organisations initially race to leverage the 'low hanging fruit' available within the field to maximise early competitive advantage.

In the field of Data Science this is primarily represented by the exertion of the maximum amount of computational ability available to be applied to the largest amount of data. Open source projects such as Hadoop/Spark have been highly successful in pursuit of this goal. Considering the statistical elements of Data Science, the utilisation of the most sophisticated and most highly advanced modelling techniques and applied statistical methodology has been a focus of interest. One example of this is the recent rapid growth of interest in Deep Learning techniques.

This process serves a very useful and valid role in the development of any field — essentially lifting the currently available methodology to a level of standardisation and accessibility where competitors are operating on a relatively level playing field. It is the opinion of the author that given the assumptions of a relatively mature degree of technical development and standardisation of methodology, that the true potential for benefit in implementation of data analysis systems can only be realised through human creativity.

The present work proposes to explore the practical implications of data analysis system design with the aim of discovering new insights of operationally significant value to creative system design and implementation. To facilitate maximal potential benefit, the practical component research will be carried out on a highly specific 'creative' domain area (discussed below) in which the author is already an expert — the purpose of this is to allow for the greatest possible insight into programmatic creative system design (abstractly), from the point of view of a domain expert who is using the system operationally. To achieve an optimum balance of creative discovery and research scientific validity, the research will be conducted under the methodology of 'Practice Based Research' (also discussed below).

Theoretical

Practice Led Research

Given that the desired outcome of the present study is the discovery of operationally significant insight into the application of creativity on system design, it seems logical that a research methodology designed to optimise creative discovery through practice should be utilised throughout the process of the work. For this purpose, the process of '*Practice Led Research*' will be implemented. Practice Led Research is a subset of Practice Based Research that is commonly utilised in creative academic practices in order to facilitate transferability of understanding gained through a practice-centric research process. In her 2006 guide to Practise Based Research, Candy, L. states that "*Practise Led Research is concerned with the nature of practice*" and continues to specify that "*The primary focus of the research is to advance knowledge about practice, or to advance knowledge within practice*".

A further key point of note is that Practice Led Research methodology differentiates between an individual seeking out new knowledge relating to their personal goals and an individual contributing new knowledge to a more general collective or shared store of knowledge. In other words, Practice Led Research aims to discover new knowledge and novel ideas through calculated application of practice.

The Harmonic Algorithm

The Harmonic Algorithm is the title of an MA thesis written by the author (South, O) in 2016. The paper presents a deep exploration of extended electric bass playing technique before proposing and exploring a methodology for composition through systematic application of musical analysis techniques.

The unconventional nature of musical data, creative inclination of the field and intimacy of the author with the subject matter represents an ideal domain area to model computationally in pursuit of new discoveries relating to creativity in system design. The specific concepts related to *The Harmonic Algorithm* (2016) will be discussed more deeply in relation to their computational implementation.

Markov Chain Models

While statistical methodology is not the primary concern of this paper from a research perspective, correct utilisation and implementation of appropriate numerical machinery is still very important to the methodology. The primary statistical technique used in the implementation of the present system is Markov Chain Probability Modelling.

A Markov Chain model has two assumptions:

1. *That the probability of the next state depends only the previous state.*
2. *That the probability of the next state is not time dependent.*

Further, a limited degree of 'memory' can be accommodated for in the model without invalidating the assumptions, by implementing a 'second order' chain (IE. the next state depends on the sequential combination of the previous two states).

The nature of musical cadence meet can be considered to meet both of these assumptions (in fact, tonal harmony depends on this fact) thus this model is valid for the purposes of modelling the present domain.

Monadic Computation

Monadic computation is a programmatic design pattern which has been adopted into computer science from the mathematical field of Category Theory. Monadic computation facilitates sequential programmatic instructions that 'embed' execution of additional logic in the passage from one computation in the sequence to the next. The additional steps of logic can be defined by the system designer, thus Monadic programming is an extremely powerful and flexible tool in the architecture of of computational systems.

Monadic computation is a specifically important concept in lazily executed functional programming languages such as Haskell, because it allows for sequential tasks to be performed in an environment which has no concept of order of computation. One specific Monad utilised in the present work is the '*R Monad*' (defined in the `inline-r`` library for Haskell). This Monad facilitates real time operations over data structures shared interactively by both the R interpreter & a compiled Haskell program.

The concept of moving computationally from state to state under a defined set of logical rules is also highly synchronous with the nature of a Markov Chain model moving from probabilistic state to probabilistic state. This relationship will be explored further in the process of the work.

Modelling Creative & Novel Behavior

A logical source from which to seek insight into creative system design is in modelling of creative and novel patterns in human behavior. In creative academia, there have been many interpretations towards modelling such an esoteric and intangible concept as creativity itself. Mclean, A. describes two such existing conceptual models for creative behavior in regard to creative computation in his 2006 paper relating to computational creativity:

1. *Creative Systems Framework (Wiggins, 2001) — Influenced by Boden (1990)*
2. *Improvisation Model (Pressing, 1987)*

Both of these models are uniquely useful for their description of creative behavior in terms of arrays of rules, objects, events and/or processes. Beyond describing the array itself, each model describes the environment within which the array exists in terms of how the array's action motivates change or interaction with the environment, as well as exploring how the array itself is transformed inside its environment over time.

In the scope of the present work, the Creative Systems Framework is of specific interest. The model describes an array $\langle \mathbf{R}, \mathbf{T}, \mathbf{E} \rangle$ inside a conceptual universe \mathbf{U} of potentially possible concepts and language \mathbf{L} by which creative rules can be expressed. The elements of the array represent the following:

R Rules by which the validity of a concept is assessed. **

T 'Traversal strategy' for locating concepts inside \mathbf{U} .

E Rules by which the quality of a concept is assessed.

***not to be confused with the R scripting language for statistical computation*

Thus, when applied to \mathbf{U} , \mathbf{R} defines the limits of potential inside the 'creative space' while \mathbf{E} judges whether these possibilities are subjectively desirable. \mathbf{T} describes the methodology by which concepts are located inside \mathbf{U} for assessment by \mathbf{R} & \mathbf{E} . Wiggins provides an excellent example for \mathbf{T} in his 2003 paper 'Categorising Creative Systems' in the difference in work of a student composer and J.S.Bach. While both the student and Bach may potentially be working with the same rules defining \mathbf{R} , the student may be locating concepts (applying \mathbf{T}) by random trial and error while Bach relies on an intuition developed over years of experience. Additionally, each composer would have a different set of rules defining \mathbf{E} , representing their judgement of the quality of a located concept.

The strength of this model is the ability it grants to facilitate discussion and reasoning about elements of creative systems abstractly of whether that system is a representation of a human process or algorithmic in nature. Further, Wiggins (2003) specifies various cases for addressing failure in creative systems in terms of $\langle \mathbf{R}, \mathbf{T}, \mathbf{E} \rangle$ — for example the inability of \mathbf{E} to select valued concepts within the conceptual space defined by \mathbf{R} is designated 'Conceptual Uninspiration' and can be remedied by transforming \mathbf{R} or modifying \mathbf{E} .

Discussion

Motivation

The purpose of this work is to motivate thinking about creative systems. With regard to this, a creative system will be developed, presented and discussed.

The Harmonic Algorithm (South, 2006) presented an algorithmic methodology for composing with the overtones of any stringed instrument. While the research was successful in the scope of instrumental study, the methodology proved to be an unwieldy tool in practice.

The 'output' of the analysis takes the form of numerous hand-prepared and labelled spreadsheets containing many rows, generally looking like this:

E Ae Gb	C	G	D	A	E	B	F#	C#	G#	D#	A#	F	Overtone Location
E G G#	C ^{b11}	G ^b	D ^b	A ^b	E ^b	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	A2/e1 G1+3 E4/e4
E G A	C6	G6	D ^b	A ^b	Em	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/e1 G1+3 A3
E G B	C ^b	G ^b	D ^b	A ^b	Em	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 G1+3 e2/G4/b1+3
E G C#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/e1 G1+3 A4
E G F#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 G1+3 b2
E G D#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 G1+3 b4
E G# A	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/e1 E4/e4 A3
E G# B	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 E4/e4 e2/G4/b1+3
E G# C#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/e1 E4/e4 A4
E G# D	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 E4/e4 G2
E G# F#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 E4/e4 b2
E G# D#	C ^b	G ^b	D ^b	A ^b	Em ⁰	B ^b	F# ^b	C# ^b	G# ^b	D# ^b	A# ^b	F ^b	E3/A2/e1 E4/e4 b4

Excerpt of analysis from *The Harmonic Algorithm* (2016)

In terms of $\langle R, T, E \rangle$:

- R** Adherence to the laws of tonal harmony & physical existence on the present instrument or tuning (exclusion from **R** notated as 'Chr.Cluster').
- T** Manual creation of and reference to a combination matrix such as the above.
- E** Methodological judgement of quality based on formal musical acceptance and subjective opinion of the composer (colour coded).

The primary reasoning for limited usefulness in this work's outcome in practice can be identified as:

1. Large amount of time and human expertise required to prepare a combination matrix — it would be unfeasible for any one author to prepare a matrix for every possible context, compounded by the fact that multiple contexts can be passed through inside even a short passage of music.
2. Slow process of reference, even where a matrix is prepared in advance — while 'absolute' desirability is colour coded, much trial, error & refinement is necessary in order to locate desirable possibilities in an actual musical context where

desirability is relative. There are simply too many options to make a fully informed decision.

Ineffectiveness of the creative agent in finding valued concepts within the space defined by **R** is termed by Wiggins as '*Generative Uninspiration*'. Wiggins states that this form of 'uninspiration' usually occurs as a result of an ill-formed traversal strategy **T** acting on a theoretically sound creative system and suggests transformation of **T** as an optimal solution. It is clear that the traversal strategy **T** is inadequate for practical usage purposes in the nature of the original Harmonic Algorithm system.

With the purpose of discovering new operationally significant insights into creative system design and implementation, the Haskell programming language (representing **L**) will be used to build The Harmonic Algorithm as an interactive software program. To remedy Issue 1. (**T**), an automated traversal strategy (**T**) will be implemented in order to render locating of concepts within the conceptual space (defined by **R**) as instantaneous as possible. Finally, Issue 2. (**E**) will be addressed through implementation of a Markov Chain model. This model will be trained according to machine learning methodology and can then be used to deterministically rank the quality of 'next' states with regard to the current 'relative' state. As the user interactively moves from state to state, the model (**E**) will adjust its ranking according to context.

Overview Of Functionality

The program at time of assessment submission is made up of 1458 lines of code, written in a combination of Haskell and R language. A brief run through of interaction with the software by command line interface (for use by a non-technical domain expert) is demonstrated below.

On load, the R language interpreter is booted, required libraries loaded and dataset read in/preprocessed:

```

Terminal - oscar south@OS-ARC:~
File Edit View Terminal Tabs Help
=====
Initialising R Interpreter..

--- Attaching packages --- tidyverse 1.2.1 ---
✓ ggplot2 3.0.0      ✓ purrr   0.2.5
✓ tibble  1.4.2      ✓ dplyr   0.7.6
✓ tidyr   0.8.1      ✓ stringr 1.3.1
✓ readr   1.1.1      ✓ forcats 0.3.0

--- Conflicts --- tidyverse_conflicts() ---
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()

session will be logged:
/home/oscar south/haskellProjects/theHarmonicAlgorithm/output/sessionlog.txt

Loading Bach Chorale Dataset from UCI Machine Learning Repository...

+-----+
| Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository |
| [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California,   |
| School of Information and Computer Science.                               |
+-----+

```

After printing the main header, a number of initialisation questions are asked:

```

Terminal - oscar south@OS-ARC:~
File Edit View Terminal Tabs Help

|-----|
| / \ . The |
| \ / | . Harmonic |
| / \ | Algorithm |
| / \ | |
|-----|
by |
  | - () -
  | - () -scar South, 2018

Welcome to The Harmonic Algorithm!

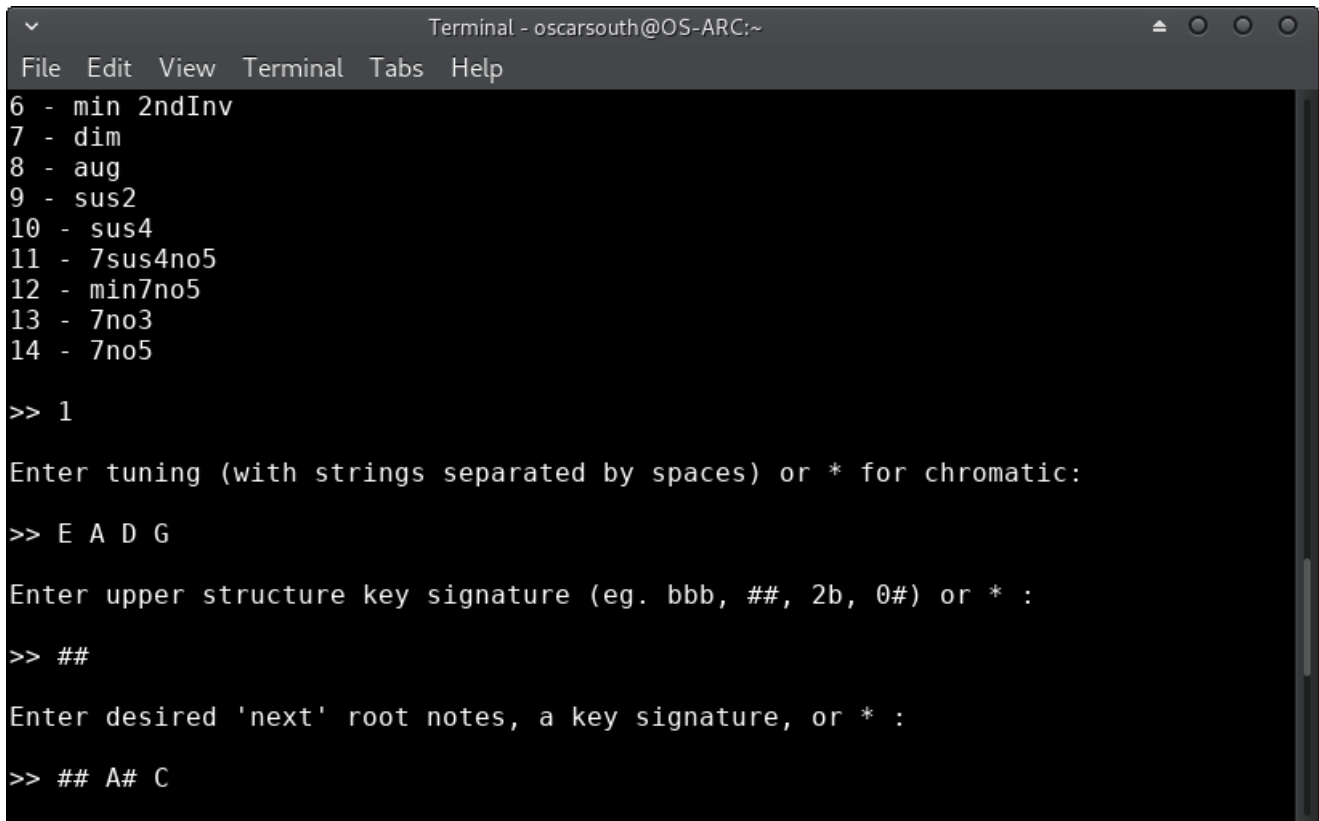
Do you want to begin with b (flat) or # (sharp) notation?

1 - Flat b
2 - Sharp #

>> █

```

Next, filters are defined to limit potential possibilities to a desirable scope (representing **R**). These can also be left open and can be modified at any point during interaction with the program:



```

Terminal - oscarssouth@OS-ARC:~
File Edit View Terminal Tabs Help
6 - min 2ndInv
7 - dim
8 - aug
9 - sus2
10 - sus4
11 - 7sus4no5
12 - min7no5
13 - 7no3
14 - 7no5

>> 1

Enter tuning (with strings separated by spaces) or * for chromatic:
>> E A D G

Enter upper structure key signature (eg. bbb, ##, 2b, 0#) or * :
>> ##

Enter desired 'next' root notes, a key signature, or * :
>> ## A# C

```

On entering into the interactive analysis environment, ranked choices for next state (representing **E**) are presented. These are delivered by interaction between the trained Markov model and currently defined filters:

```

Terminal - oscar south@OS-ARC:~
File Edit View Terminal Tabs Help
The current chord is D maj -- Select next chord or choose another option:

1 - G maj
2 - D maj
3 - A maj/C#
4 - E min
5 - A maj
6 - E min7no5
7 - F# min/C#
8 - D sus2
9 - B min/D
10 - G maj/B
11 - C 6sus2no5
12 - B min
13 - C maj
14 - E min6no5
15 - [      Modify filter      ]
16 - [      Random sequence    ]
17 - [      Show more          ]
18 - [      Switch to flat notation ]
19 - [      Select new starting chord ]
20 - [      Quit                ]

>>

```

Randomised deterministic sequences can be generated from the trained Markov model for faster traversal through the conceptual space. The level of 'entropy' in the randomisation process can be defined by the user and sequences generated until a new state is accepted or sequence rejected:

```

Terminal - oscar south@OS-ARC:~
File Edit View Terminal Tabs Help
20 - [      Quit                ]

>> 16

Enter desired length of sequence (default 4, max 16):

>> 16

Choose entropy level as a number between 1 and 10 (default 2):

>> 3

Press enter to accept, specify a bar number, or choose another option:

1  ||   D maj           |   B min/D           |   D maj           |   G maj           |
5  |   E min/G         |   A maj             |   D maj           |   F# min/C#      |
9  |   B sus2          |   F# sus4          |   D sus4          |   A sus4          |
13 |   A sus2          |   G maj/B          |   G maj/B         |   G maj/B         ||

17 - [      Reject sequence      ]
18 - [      Regenerate sequence  ]
19 - [      Show with flat notation ]

>>

```

This comparatively simple interactive loops offers an infinite degree of potential for exploration inside the methodology of *The Harmonic Algorithm* (2016). The system performs analytic processes which may have taken hours or days of manual labour instantaneously, requiring only the cognitive will of the user and a couple of keystrokes.

At all stages of interaction there are a number of other related actions which can be taken by the user. These are all based on domain specific requirements for interactivity or display preferences.

'Main' module

The 'Main' module is where all interaction between the program and the user (or other sources of outside 'state') takes place. On initialising the program the `main` function is executed, inside which the entire program is nested:

```
23 main = withEmbeddedR defaultConfig $ do
24   ·initR -- load R libraries & settings, initialise R log, print info to stout
25   ·model ← choraleData -- bind trained model
26   ·header -- print main title
27   ·putStrLn "Welcome to The Harmonic Algorithm!\n"
28   ·runReaderT loadLoop model -- enter ReaderT (Model) monad with trained model
29   ·return ()
```

'main' function

The main function (above) initialises the R language interpreter then executes an 'Extract, Transform & Load' script in a mixture of Haskell and R, returning the trained model inside a monadic context:

```

31 -- |script directing process of loading & transforming data then training model
32 choraleData :: IO MarkovMap
33 choraleData = do
34   ·uciRef -- print dataset source reference
35   ·bachData -- execute R script to preprocess data
36   ·chFunds ← bachFundamental -- retrieve and bind R column of fundamental notes
37   ·x1 ← fromRMatrix 1 -- retrieve and bind columns from R matrix
38   ·x2 ← fromRMatrix 2 -- |
39   ·x3 ← fromRMatrix 3 -- |
40   ·x4 ← fromRMatrix 4 -- |
41   ·x5 ← fromRMatrix 5 -- V
42   ·let model = markovMap $ -- call Markov module to train model
43     ····· fmap toCadence <$> -- map bigram sets into Cadence data types
44     ····· bigrams $ -- combine chords into sequential bigrams
45     ····· flatTriad <$> -- convert to 'Chord' data type
46     ····· mostConsonant . possibleTriads'' <$> -- derive most suitable triad
47     ····· filter (\(_, ys) → length ys >= 3) ( -- remove sets of less than 3
48     ····· zip chFunds $ -- zip with fundamentals R column
49     ····· (fmap round) . unique <$> -- remove duplicate elems . convert to Integer
50     ····· [[a,b,c,d,e] | (a,b,c,d,e) ← List.zip5 x1 x2 x3 x4 x5]
51     ····· ) -- ^ convert R matrix columns to a list of lists
52   ·return model

```

Haskell 'ETL' and modelling script

The raw data (retrieved from the UCI Machine Learning Repository) is a 5665 row CSV file of records describing what pitchclasses are present on any given beat of 100 J.S.Bach Chorale Harmonisations:

```

1 000106b_,1,YES, NO, NO, NO, NO,YES, NO, NO, NO,YES, NO, NO,F,3, F_M
2 000106b_,2,YES, NO, NO, NO,YES, NO, NO,YES, NO, NO, NO, NO,E,5, C_M
3 000106b_,3,YES, NO, NO, NO,YES, NO, NO,YES, NO, NO, NO, NO,E,2, C_M
4 000106b_,4,YES, NO, NO, NO, NO,YES, NO, NO, NO,YES, NO, NO,F,3, F_M
5 000106b_,5,YES, NO, NO, NO, NO,YES, NO, NO, NO,YES, NO, NO,F,2, F_M
6 000106b_,6, NO, NO,YES, NO, NO,YES, NO, NO, NO,YES, NO, NO,D,4, D_m
7 000106b_,7, NO, NO,YES, NO, NO,YES, NO, NO, NO,YES, NO, NO,D,2, D_m

```

Excerpt from raw dataset

The CSV file is read and rapidly transformed by the R language interpreter, according to R scripts embedded into Haskell source code. An excerpt is presented below:

```

425 bach ←
426   · bach %>%
427   ·· select(seq, event, fund, acc, label) %>%
428   ·· add_column(pitch = bach %>%
429   ······ select(`0`:`11`) %>%
430   ······ t() %>%
431   ······ as.data.frame() %>%
432   ······ unname() %>%
433   ······ map(function(x) str_which(x, "YES")-1)
434   ······ )
435
436 bachMatrix <-
437   · reduce(bach$pitch,
438   ······ rbind,
439   ······ matrix(,0,bach$pitch %>%
440   ······ map(length) %>%
441   ······ rapply(c) %>%
442   ······ max()
443   ······ )
444   ······ ) %>%
445   · unname()
446
447 bachFund <- bach$fund

```

Excerpt from embedded R language 'ETL' script

A number of helper functions are defined to access the data from the R language interpreter and bring it into Haskell:

```

452 -- |helper function to extract R matrix column from R and deliver to Haskell
453 fromRMatrix :: Double → IO [Double]
454 fromRMatrix x =
455   ·· let rData x = R.fromSomeSEXP <$> [r| bachMatrix[,x_hs] |]
456   ·· in runRegion $ rData x
457
458 -- |helper function to extract vector of fundamental notes from R into Haskell
459 bachFundamental :: IO [String]
460 bachFundamental =
461   ·· let rData () = R.fromSomeSEXP <$> [r| bachFund |]
462   ·· in runRegion $ rData ()

```

Haskell mappings for accessing R language data-structures

After binding the trained model inside a Monadic context, a 'Reader' monad is instantiated and entered (inside which, the model is available to all functions). The rest of the program takes place inside the Reader Monad:

```

55 type Model a = ReaderT MarkovMap IO a -- representation of trained model

```

Type synonym for Reader Monad

Inside the Reader monad, a variety of lengthy functions are defined, facilitating the 'flow' of interactive engagement with the program. The most significant

```

146 -- |function to retrieve and filter down possibilities based on current state
147 recommendations :: Enharmonic → Root → Cadence → Filters → Int → Model [Cadence]
148 recommendations fs root prev filters n = do
149   model ← ask
150   let enharm = enharmMap fs -- extract enharmonic 'key' into function
151       hAlgo = (\xs → [ toCadence (transposeCadence enharm root prev, nxt)
152                       | nxt ← xs ]) $ -- ^ Convert list of Chords into Cadences from last state
153       List.sortBy (compare `on` (\(Chord (_,x)) → -- sort by dissonance level
154                   fst . dissonanceLevel $ x)) $ filters enharm -- get values from Filters
155       bach = filter (\(x,_) → x `elem` hAlgo) $ -- remove elements not in Filters
156       List.sortBy (compare `on` (\(_,x) → 1-x)) $ -- sort by markov probability
157       fromMaybe [(prev, 1.0)] $ -- extract Cadence list from maybe
158       Map.lookup prev model -- extract current markov state from model
159       nexts = take n $ (fst <$> bach) ++ -- append to markov list and take n
160       (filter (\x → x `notElem` fmap fst bach) hAlgo)
161       -- ^ keep elements of Filters list not in markov list
162   return nexts

```

Function returning ranked list of recommendations based on current state

The `recommendations` function is the ‘heart and soul’ of the creative system and encompasses the entirety of the Creative Systems matrix $\langle R, T, E \rangle$. The `hAlgo` statement calls programmatic rules for assessing validity of a concept in the creative space (R) while the `bach` statement utilises the trained machine learning model in assessing quality of concepts (E). The entire operation of the interactive program and its numerical backend as well as the intuition and experience of the user can be considered to represent aspects of the traversal strategy (T).

‘MusicData’ module

The purpose of the ‘MusicData’ module is to provide a domain specific ‘grammar’ inside which data representing musical structures and movements can be represented and analysis can be performed.

This could facilitate a ‘tiered’ organisational structure of separation/collaboration between the following:

- The expert in computation, who could implement a domain specific grammar in a low level language (notated by Wiggins as L^* and L , respectively) and additionally implement usability features such as user interface or API functionality.
- The ‘intermediate’ domain expert or statistician who is reasonably comfortable in programming/scripting and can apply contextually valid rules in L^* relevant to the specific business problem or use case, representing partial automation of R and T .
- The ‘end user’ — usually a non-technical domain expert with practical experience in decision making (relating to E) who can augment the value of his experience through interaction with an interface or service tailored to the requirements of domain expectation.

The Harmonic Algorithm (software) has been developed inside this mental model. The MusicData module bridges the middle ground between library level programming and domain specific interaction — in this case via a command line interface.

Inside this module, the 'MusicData' class is defined, through which the fundamental laws of arithmetic are redefined for musical information. As a result, the 'standard' arithmetic operators can be used with machine readable representations of musical data. A number of 'MusicData' instances are declared, representing both machine and human readable representations of music as well as mappings between them:

```

16 -- |newtype defining PitchClass
17 newtype PitchClass = P Int deriving (Ord, Eq, Show, Read)
18
19 -- |Bounded instance definition for PitchClass
20 instance Bounded PitchClass where
21   · minBound = P 0
22   · maxBound = P 11
23
24 -- |Num instance definition for PitchClass, defining PitchClass arithmetic
25 instance Num PitchClass where
26   · (+) (P n1) (P n2) = P (n1 + n2) `mod` P 12
27   · (-) (P n1) (P n2) = P (n1 - n2) `mod` P 12
28   · (*) (P n1) (P n2) = P (n1 * n2) `mod` P 12
29   · negate ..... = id
30   · fromInteger n ..... = P $ fromInteger n `mod` 12
31   · abs ..... = id
32   · signum a ..... = 1
33

```

Machine readable definition of musical pitchclass

```

65 -- | MusicData class definition
66 class Ord a => MusicData a where
67   · pitchClass :: a -> PitchClass -- mappings into pitch classes
68   · sharp :: a -> NoteName -- mappings into sharp note names
69   · flat :: a -> NoteName -- mappings into flat note names
70   · (<+>) :: a -> Integer -> PitchClass -- addition between MusicData & Integer
71   · (<->) :: a -> Integer -> PitchClass -- subtraction between MusicData & Integer
72   · i :: Num b => a -> b -- polymorphic mapping from MusicData into numeric types
73   · (+|) :: a -> Integer -> NoteName -- addition by Integer to flat representations
74   · (-|) :: a -> Integer -> NoteName -- subtraction by Integer to flat representations
75   · (+#) :: a -> Integer -> NoteName -- addition by Integer to sharp representations
76   · (-#) :: a -> Integer -> NoteName -- subtraction by Integer to sharp representations
77   · i ..... = fromIntegral . toInteger . pitchClass -- automatic derivation
78   · (+|) a = flat . (<+>) a .....|
79   · (-|) a = flat . (<->) a .....|
80   · (+#) a = sharp . (<+>) a .....|
81   · (-#) a = sharp . (<->) a .....v

```

'MusicData' class definition

An important aspect of this module's design is that the overall goal is not to predict the next 'object' or the probability of an eventual result but to predict the next 'movement' from one state into another (including itself). With regard to this, it is a notable consideration with musical data that everything is subjectively 'relative' inside its deterministic ordered context. A practical example would be that if the established key centre at a given time was C major, then the cadence of Dmin -> Gmaj (a ii -> V cadence with root movement ascending 5 semitones) could be considered an identical

cadence relative to the key-centre as a movement of F#min to Bmaj in the key of E major (the same movement transposed up four semitones).

It makes little practical sense then to process each deterministic state as unique to its own transposition. This would cause various practical and theoretical issues such as:

- Cadences into harmonic structures common to keys preferred in the input data would be ranked artificially highly.
- Different predictions returned for different or differently tuned instruments — this is of little practical purpose given the commonplace use of equal temperament in modern music, rendering all keys equally viable.
- Intermediate data structures such as the transition matrix would be massively inflated — with each theoretical state expanded into twelve unique instances.
- Much potential information contained in the input dataset would be lost due to the many theoretically identical but literally different movements present being considered as unrelated events.

It can be observed that the symptoms described above present a general description for overfitting a dataset. It is certainly noteworthy that it is possible to conceptually overfit (or avoid overfitting) a model in this manner at such an early stage of system design — even before selection/implementation or tuning of an algorithm. This emphasises the need to carefully consider and model a suitable representation for domain data at an early design stage.

In the case of the current work, this phenomenon was solved by representing deterministic data-points as the difference between two ‘concrete’ points. Within operation of the software, all analysis is performed on ‘relative’ Cadence representations which are only converted to ‘concrete’ Chord representations for display to the user:

```

472 -- |representation of a Cadence as a transition to a structure by an interval
473 data Cadence = Cadence (Functionality, (Movement, [PitchClass]))
474   deriving (Eq, Ord)
475
476 -- |customised Show instance for readability
477 instance Show Cadence where
478   show (Cadence (functionality, (dist, ps))) =
479     show $"( " ++ show dist ++ " → " ++ functionality ++ " )"
480
481 -- |mapping from two Chord data structures to a Cadence
482 toCadence :: (Chord, Chord) → Cadence
483 toCadence ((Chord (_, _), from@(x:_)), (Chord (_, new), to@(y:_))) =
484   Cadence (new, (toMovement x y, zeroForm to))
485
486 -- |mapping from possible Cadence and Pitchclass into next Chord with transposition
487 fromCadence :: (PitchClass → NoteName) → PitchClass → Cadence → Chord
488 fromCadence f root c@(Cadence (_,(_,tones))) =
489   (toTriad f) $ i . (+ movementFromCadence c) . (+ root) <$> tones
490
491 -- |mapping from prev Cadence and Pitchclass into current Chord with transposition
492 transposeCadence :: (PitchClass → NoteName) → PitchClass → Cadence → Chord
493 transposeCadence f root (Cadence (_,(_,tones))) =
494   (toTriad f) $ i . (+ root) <$> tones

```

Representation of musical cadence

‘Markov’ Module

The purpose of the ‘Markov’ module is to process the deterministic dataset (which has been transformed according to domain specific rules defined in the ‘MusicData’ module) into a data structure suitable for interactive traversal in running program. Programmatically the resulting structure is represented as a mapping of key-value pairs where the key represents the current state and the lookup value is a list of all theoretically possible ‘next’ states with assigned probabilities. In statistical terminology with reference to Markov Chain models, this is the Transition Matrix:

```
28 -- |representation of markov transition matrix as key-value pairs
29 type MarkovMap = Map Cadence [(Cadence, Double)]
```

Type signature of trained model

A challenge faced both in concept and implementation was that of what to do when there were no movements from one state to any other state present in the input data. While uncommon, this conundrum could potentially result in poor recommendation ranking (**E**) and hard to diagnose anomalies in statistical calculations due to probability rows in the transition matrix not having a sum of 1.

The logical solution was to consider this event to be a harmonic movement from the current state into an identical state. This trivial movement represents a change in deterministic state and gives rise to additional possibilities for movement from the new state. In the case that the new state also has no cases of cadence to a different state then the underlying mathematics will at least be working without error and the user can draw on practical domain knowledge to ‘break’ the cycle, arbitrarily specifying a movement (resulting in a new state with new possibilities).

This phenomenon is relatable to Wiggins (2003) definition of ‘*Hopeless Uninspiration*’ — the inability of **R** to select valid concepts, presenting an empty conceptual space that is incapable of generating valid concepts. While Wiggins states that this may represent an ill-formed creative system, in this case it can be deemed to be acceptable for two reasons:

1. The empty conceptual space is limited to only specific undesirable ‘corner’ states which are extremely uncommon and very difficult to locate — even intentionally.
2. The user is able to apply their own expertise and intuition to guide traversal at any time, thus would be able to easily break out of any such conceptual space.

It can also be noted that in extensive testing of the Harmonic Algorithm software that this phenomenon was never encountered. It is possible however to ‘self-define’ an empty conceptual space by applying the interactive filters defined in the ‘Overtone’ module in a careless manner.

'Overtone' Module

The Overtone module primarily contains functions for parsing user inputted domain specific musical syntax which is used to generate machine readable sets of theoretically possible overtone combinations. The operation of this module serves as a programmatic bridge that facilitates interaction between the non-technical user and the domain specific backend defined in 'MusicData'.

The sets of combinations generated by this module represent the conceptual space defined by **R**. This conceptual space is then ranked in terms of desirability (considering current state) by the machine learning model defined in the 'Markov' module.

Below is an example of a parsing function defined in the 'Overtone' module:

```

49 -- |mapping from String to Integral list representing tones present in a key
50 parseKey' :: (Num a, Integral a) => Int -> String -> [a]
51 parseKey' n str = unique $ pitchList n str []
52   where
53     pitchList n str =
54       let
55         xs = words $ Char.toLower <$> str
56         k n = (`mod`12) . (+n*5 `mod` 12) <$> [0,2,4,5,7,9,11]
57         ys ?? z = if any (`elem` ys) $ xs
58                 then (mappend z)
59                 else (mappend mempty)
60         chain =
61           [ "*" ,"a11","chr"]?? [0..11]
62           , ["0b","0#","0","c","b#","am"]?? k 0
63           , ["1b","b","11#","#####","f","e#","dm"]?? k 1
64           , ["2b","bb","10#","#####","a#","gm"]?? k 2
65           , ["3b","bbb","9#","#####","eb","d#","cm","b#m"]?? k 3
66           , ["4b","bbbb","8#","#####","ab","g#","fm","e#m"]?? k 4
67           , ["5b","bbbbb","7#","#####","db","c#","bbm","a#m"]?? k 5
68           , ["6b","bbbbb","6#","#####","gb","f#","ebm","d#m"]?? k 6
69           , ["7b","bbbbb","5#","#####","cb","b","abm","g#m"]?? k 7
70           , ["8b","bbbbb","4#","#####","fb","e","dbm","c#m"]?? k 8
71           , ["9b","bbbbb","3#","#####","a","gbm","f#m"]?? k 9
72           , ["10b","bbbbb","2#","#####","d","cbm","bm"]?? k 10
73           , ["11b","bbbbb","1#","#####","g","fbm","em"]?? k 11
74         ]
75     in foldr (.) id chain

```

Function for parsing key signature

Closing Sections

Conclusion

A conceptually complete realisation of a theoretical interactive data analysis system in a highly specific domain area has been implemented on a programmatic level. Throughout this process, a number of design decisions as well as conceptual shortcomings and solutions have been discussed in practical terms as well as under the lens of existing models of creative behavior.

An important decision which was relevant throughout the entire design/implementation process is how much domain knowledge should be enforced by the system designer. For example, in music theory there is much established practice dictating how viable different possibilities are in any given context. It is extremely tempting as a domain expert to use these rules to enforce constraints on the system that would not only but add what seems like a layer of insightful expertise, but also potentially make system implementation easier. However, these are creative decisions by the designer. If the goal of the system is to augment the individual creativity of the user then enforcing creative decisions at a system level which the user will not even engage with in their own process is actually limiting creativity. Likewise, if a system is designed for use by non-technical users then a limited degree of enforced domain knowledge may be necessary in order to facilitate practical transference of logic from the analysis system into the 'hands' of the user in a way that allows them to apply their practical experience most effectively.

By exploring the design and implementation of data analysis systems on a programmatic level while superimposing a model for conceptualising creative systems, a mental model for conceptualisation of system design under a framework that allows for domain expertise to be applied in the most effective manner has been established. Wiggins' development of Bodens 'Creative Systems Framework' (as defined by Mclean, 2007) presents an excellent conceptual mechanism in the matrix $\langle R, T, E \rangle$ for describing creative behavior. It is the opinion of the author that there is much scope for application and development of this concept in creative system design and implementation.

References/Bibliography

- Lipovača, M. (2012). *Learn You a Haskell For Great Good!* San Francisco, Calif.: No Starch Press.
- O'Sullivan, B., Stewart, D., Goerzen, J. (2012). *Real World Haskell*. Sebastopol: O'Reilly.
- Wickham, H., Grolemund, G. (2017). *R For Data Science*. Sebastopol: O'Reilly.
- Wickham, H. (2014). *Advanced R*. Chapman and Hall/CRC
- Colah.github.io. (2018). *Neural Networks, Types, and Functional Programming*. [online] Available at: <http://colah.github.io/posts/2015-09-NN-Types-FP/>
- Hughes, J. (1989). *Why Functional Programming Matters*. The Computer Journal, 32(2), pp.98-107.
- Wadler, P. (1989). *Theorems for free!* Functional Programming Languages and Computer Architecture, pp.347-359
- Boespflug, M., Dominguez, F., Vershilov, A., (2014) *Project H: Programming R in Haskell* [online] Available at: https://ifl2014.github.io/submissions/ifl2014_submission_16.pdf
- McLean, A. (2007) *Improvising with Synthesised Vocables, with Analysis Towards Computational Creativity*, Goldsmiths College, University of London [online] Available at: <https://pdfs.semanticscholar.org/99ac/092d014aac16728912563975282e20039e19.pdf>
- Candy, L. (2006) *Practice Based Research: A Guide, CCS Report: 2006 - V1.0 November*, University of Technology Sydney [online] Available at: <https://www.creativityandcognition.com/resources/PBR%20Guide-1.1-2006.pdf>
- South, O. (2016) *The Harmonic Algorithm*, UK, University Of Chester [online] Available at: <https://www.dropbox.com/s/68gsasxqep8h7g3/The%20Harmonic%20Algorithm.pdf?dl=0>
- Grinstead, C., M., Snell, J., L. 1997). *Introduction to Probability*, American Mathematical Society
- Wiggins, G., A. (2001) *Towards a More Precise Characterisation of Creativity in AI*, Department of Computing, City University, London [online] Available at: <https://pdfs.semanticscholar.org/f266/a962520f89b77e9e4023a76a3a76eede4687.pdf>
- Wiggins, G., A. (2003) *Categorising Creative Systems*, Centre for Computational Creativity, City University, London [online] Available at: <http://www.doc.gold.ac.uk/~mas02gw/papers/ijcai03.pdf>
- Milewski, B. (2018). *Category Theory for Programmers* [online] Available at: <https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/>