CSYS5030 Information Theory and Self-organisation
Assessments 4 – Information Theory Project Report
Information-theoretic analysis of JS Bach's St Matthew Passion

Table of Contents

1 Brief introduction

The St Matthew Passion, written by Johann Sebastian Bach in 1727, is one of the greatest masterpieces of classical sacred music. It is an oratorio setting chapters 26 and 27 of the Book of Matthew in bible, interspersing chorales and arias, and lasts for less than 3 hours. (Wikipedia, 2020a)

Talking about its structure(Wikipedia, 2020b), St Matthew Passion's movements consist of two parts and 16 scenes based on the dramatic action in different locations which are described in the Gospel. Each part (Part I and Part II) begins with chorus and aria and the scenes in it usually starts with Gospel text sung by the Evangelist, and after the narration of the action, chorales and arias follow as a mediation. There are different ways for the scene to end, whether it ends on an aria or a chorus, or in a midst of Gospel text section. For further details in the scene division based on the Gospel text, see Figure A-1.

2 Aim

The main problem is basically the prediction of next note in a voice type given the information of different sources. It is further broken into two questions that are wished to answer using information-theoretic measures:

1) Knowing the past notes of a voice type itself, how much information does it store to predict the next note? and
2) Building on the answer from 1), how much does the other voice type able to help to predict the note transition in current process?

3 Dataset

The dataset used in this project is retrieved from the corpus library of Music21. It consists of 13 movements of St Matthew Passion: movement numbers 3,10, 15, 17, 25, 29-a, 32, 37,40, 44, 46, 54 and 62. These movements are all the chorales of the masterpiece by Bach. Which also means, the dataset used is not the full piece of St Matthew Passion, but rather the choral movements of choir (with four voice types Soprano, Alto, Tenor and Bass) only.

4 Method

4.1 Pre-processing/ Preparation of data

The pre-processing of data is done in the Python language environment with the aid of Music21, a powerful toolkit for musicology. Other than loading the data at first and lastly writing the data to csv file, there are four main steps involved in between, namely centring, extracting, concatenating, and symbol-assigning.

4.1.1 Centring

Centring data is a common pre-processing technique, where the values of data are being centred or normalised so that they are on a similar scale. In music context, this is done by transposing all the streams to similar key.

Every piece of music has their own key, which forms its basis. Taking St Matthew Passion chorales as an example, every voice type in each movement has their own key, even of the same voice type i.e. Soprano in Movement 3 is in key “b minor” but “A major” in Movement 25 and so on. Transposing them into a same key can avoid in getting a uniform distribution of probabilities of each note when the streams of music are pooled together (see Figure 4-1). It is also important because information-theoretic analysis is mainly based on probability and uncertainty, where a uniform probability distribution will bring to a lower statistical significance. In this study, all voice types in all 13 parts are transposed to key “D”.

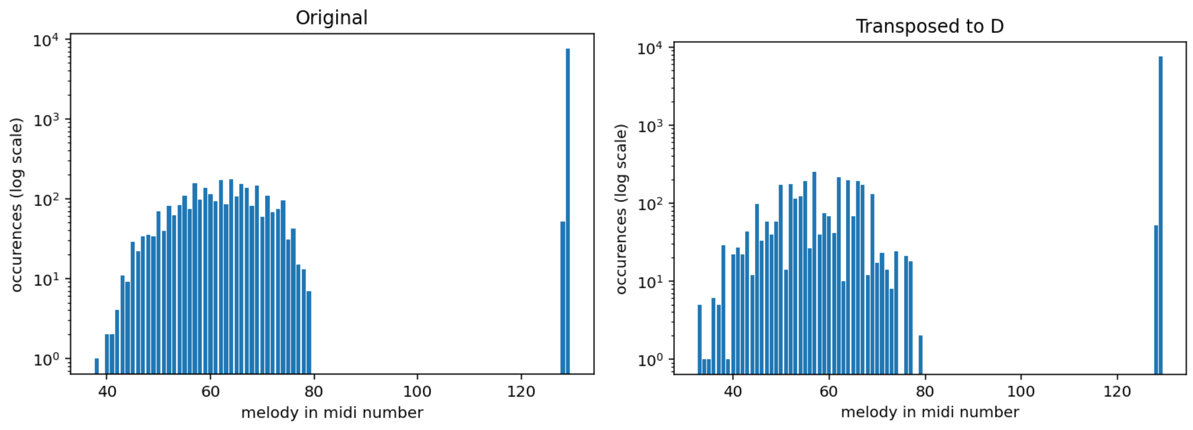


Figure 4-1 Distribution of pitches in whole dataset before (left) and after (right) transposing to key “D”.

4.1.2 Extracting note sequence

After transposing the notes to one scale, the notes are extracted so that the time series of notes being played (or sung in this context) can be obtained. To make sure that all voice types in each movement will have the same sequence length, aka their notes are lined up in time, the music is represented as a sequence of notes and rests. They are denoted in integers between 0-127 corresponding to their MIDI pitch number (see Figure A-2), while integer 128 is assigned to indicate a stop (usually appears at the end of a line or a movement), and 129 represents no event (empty notes). In order to capture each note and assure the details, each integer lasts for one sixteenth note (one semiquaver) of duration (see Figure 4-2).

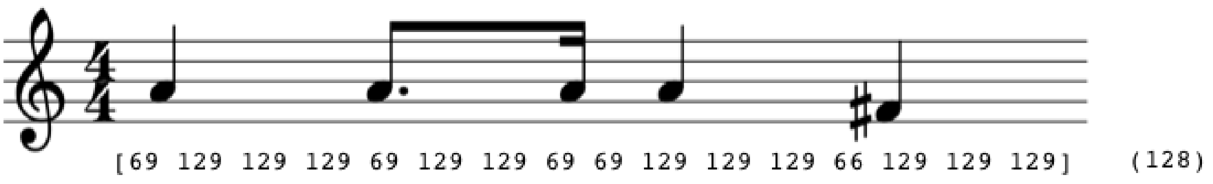


Figure 4-2 Example of conversion of melody into a note sequence.

4.1.3 Concatenating sequences/ Pooling

The note sequences from 13 movements are then pooled by four voice types by concatenating them together into four arrays, with each length of 2605. It is noted that not all 128 pitches occurred in the dataset, hence, re-assigning numerical symbols to the sequences comes into play.

4.1.4 Assigning numerical symbols

Note arrays of four voice types are pooled to find the pitches that occurred in the dataset. It is found that there are 47 integers used (including 128 and 129). The pitches are grouped to their corresponding notes so that only 14 symbols are used to represent the music as shown in Table A-1.

4.2 Information-theoretic measures

Information-theoretic analysis of discrete data with base 14 in this study is performed with the use of Java Information Dynamic Toolkit (JIDT) by Lizier (2014) in Matlab environment.

4.2.1 Active Information Storage (AIS)

To approach the first question in Chapter 2, AIS measure is applied. It captures the mutual information of past variables with k history length and the next state, which tells us the information about the future.

$$A_{note}(k) = I(note_n^{(k)}; note_{n+1}) \tag{1}$$

As the trend of notes is not of a regular pattern, longer k will yield higher AIS. At the same time, however, it increases also the bias. It is found that even k =1, the bias of 0.0325 bits exists for all voice types (variables). Graphs of AIS vs k were plotted for each variable (see Figure A-3), and it is noted that for all variables, the bias goes extremely high (over 1300 bits) when k > 4. In the end, k = 4 was selected for the remaining study, since our notes are represented in one sixteenth duration (see Chapter 4.1.2), k = 4 indicates that we consider quarter of the duration as the history length. Other than that, local AIS over time is plotted to capture the dynamics of information storage at each point.

$$a_{note}(n,k) = \log_2 \frac{p(note_{n+1}|note_n^{(k)})}{note_{n+1}} \tag{2}$$

4.2.2 Transfer Entropy (TE)

TE can outline the second question in Chapter 2. It tells how much information from the past of a source variable (or a voice type) helps us predict the next note of the target (another voice type) given that the past of the target has been considered. Note that in this study, only one of the past notes (and delay = default, 1) from the source is considered.

$$T_{source \rightarrow target}(k) = I(source_n; target_{n+1}|target_n^{(k)}) \tag{3}$$

Again, local TE over time is plotted to observe the changes in information from a specific source value to predict specific next value in context of past.

$$t_{source \rightarrow target}(k) = \log_2 \frac{p(target_{n+1}|\, target_n^{(k)}, source_n)}{p(target_{n+1}|\, target_n^{(k)})} \tag{4}$$

5 Results

As the dataset is huge (4 variables and 13 movements), the scope of this study is defined to one variable, Soprano. It is hence also the target or destination for TE measures.

5.1 LAIS

5.1.1 Full time series

First of all, LAIS of all time was computed for Soprano and the result was plotted as shown in Figure 5-1. Red dash lines were plotted to indicate the sections of movements.

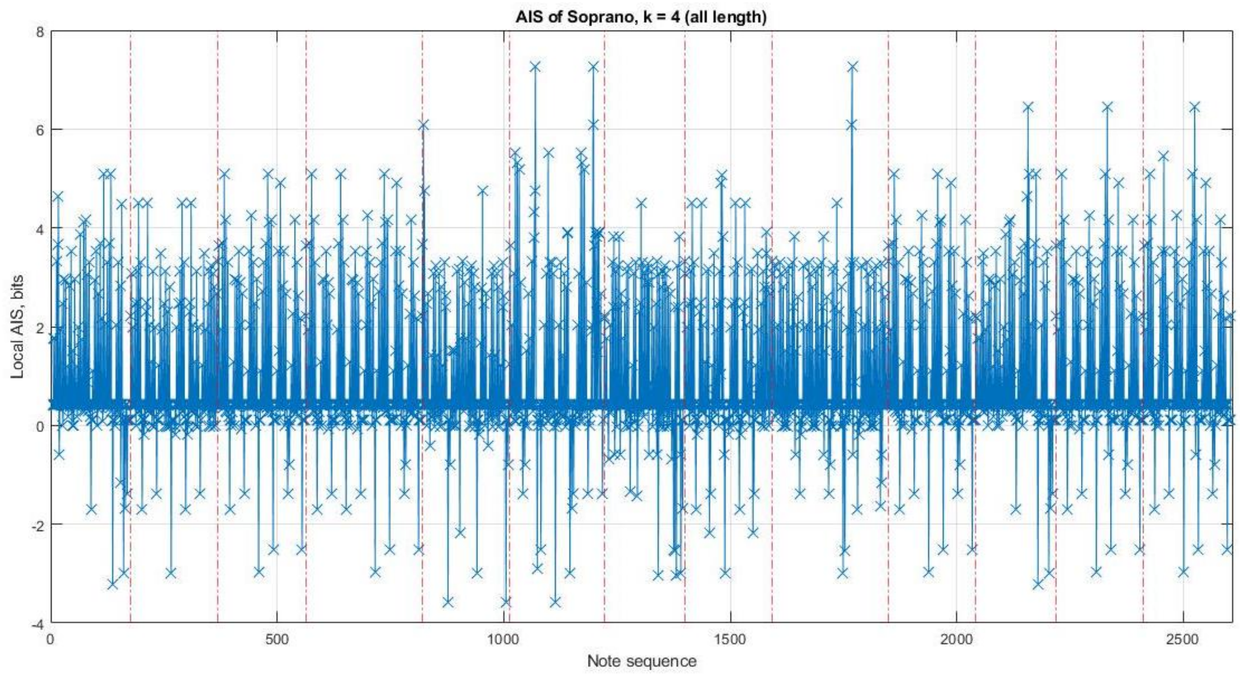


Figure 5-1 Local Activation Information Storage against time for Soprano, with k = 4.

From the graph above, it is observed that with the knowledge of past four notes, the amount of information in each note that is predictable from this knowledge and thereby has been stored in this past is different through time. There is no obvious and repeated pattern seen from the figure and it is hard to explain the dynamics since the scale is too small with all 2506 states display in one graph. To better visualise what is happening in the time series of notes, it is suggested to observe it by movements. For this study, we are going to look at only one movement, which is the sixth movement in the time series plotted, where there are a few peaks as well as troughs of LAIS observed.

5.1.2 BWV244.29-a

The name (or code number) of the selected movement is BWV244.29-a. It is actually a chorale sung in the scene "The Arrest of Jesus" (see Figure A-1). In order to understand the dynamics of LAIS, the corresponding notes distribution of Soprano is plotted on the same graph as seen in Figure 5-2. From the figure, firstly, it is observed that there is a small part of notes has been repeated in this movement, as highlighted in the yellow boxes. The trend of LAIS for these two parts are exactly the same. Secondly, the LAIS shoots each time when there is a note other than note number 13 (which denotes "no events"), and it seems like they happen at the same points. This is not a coincidence, in fact, this is because the note sequences were extracted such a way that each symbol represents one semiquaver of time (see Chapter 4.1.2), hence sequence like

“A number, 13,13,13, other number, 13, 13, 13…” is commonly observed. Moreover, we consider a history length of 4 states, hence, the easiest way to interpret the LAIS trend is that, whenever it shoots, it reflects the last note sung by Soprano. The amount of LAIS depends on the last note sung, for example, if the current note is the same as the last note sung, the LAIS is higher, indicating that the past states were informative to predict the current. Continuing from what being discussed in second point, thirdly, a contrast in LAIS values is observed when the sequence does not follow the “number, 13, 13, 13” pattern. When 13 occurs more than 3 consecutive times, LAIS is in negative value. Negative LAIS indicates that “the observed past state made the following observation less likely to occur than we would have guessed without the knowledge of the past state” (Wibral, Lizier, Vögler, Priesemann, & Galuske, 2014). The past notes are said to be misinformative about the current note, where it is predicted to have a note sung but it ends up by no events. On the other hand, peaks of LAIS are observed when notes are sung consecutively 3 times. This is because the combination of a sung note followed by another is less probable and more surprising, and therefore higher information content and higher pointwise mutual information to predict it.

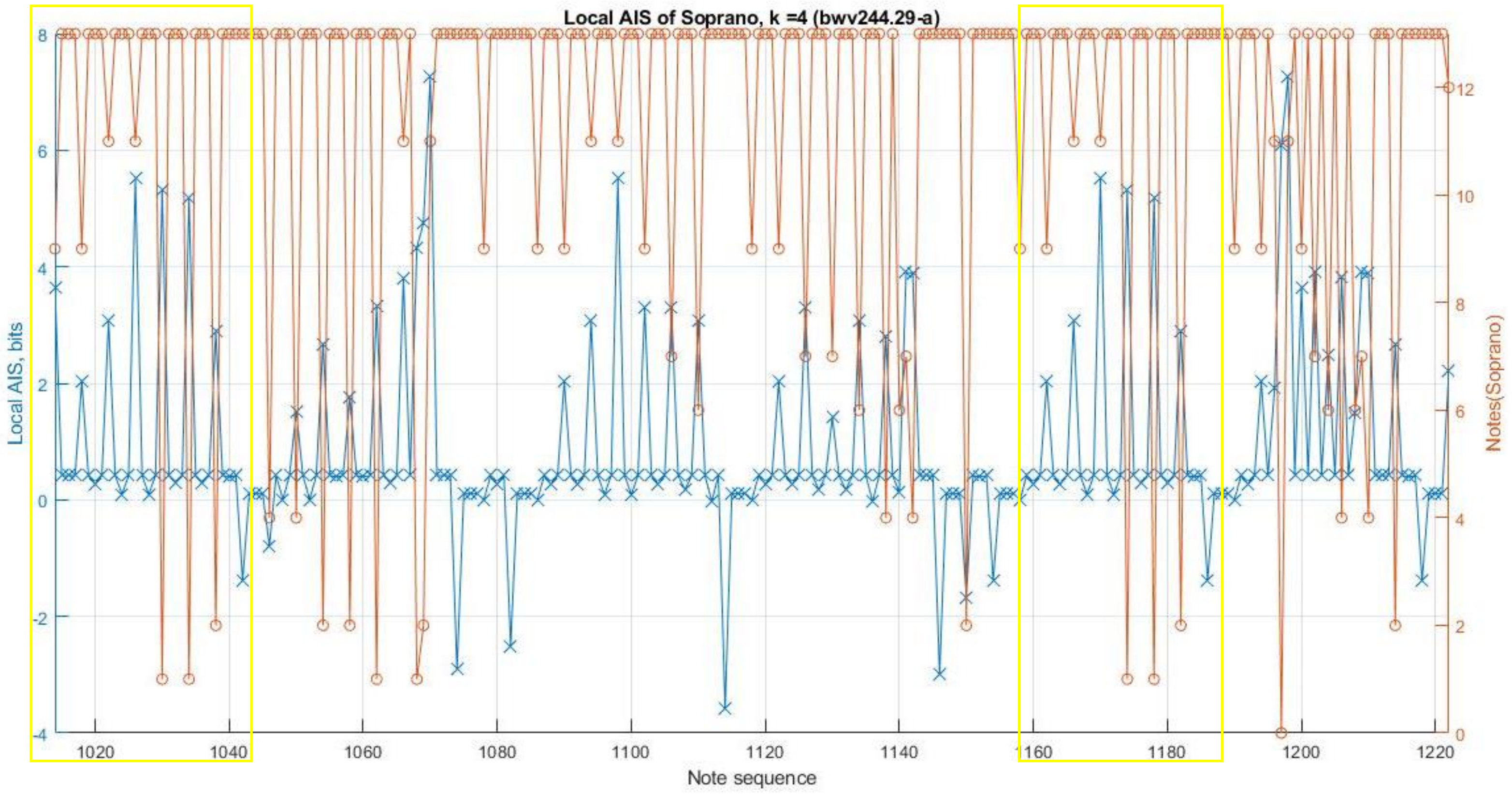


Figure 5-2 Local AIS of Soprano when k=4 (in blue) with its notes distribution (in red), for movement BWV244.29-a.

5.2 LTE

So, having known the past 4 notes, does one note in the past from Alto, Tenor or Bass help us to predict the following notes in Soprano? Local transfer entropies of three different voice types to Soprano are computed, and plotted in Figure 5-3, Figure 5-4, and Figure 5-5. The common trend observed from these three figures is that, non-zero values of LTE occur only when there is a transition of state (from no event to a note or vise versa) in the destination, Soprano. Most of the time, notes in Soprano and other voice types are distributed in a way that both of them sing and rest at the same time. When Soprano transits from long silence (no events) to a note, LTE is positive where the state of the source being 13 gives some information (surprises) to the transition. Contrastly, when Soprano stays longer in state of silence, LTE from source tends to misinform about this, giving a negative value. Other than that, it is noticed that information from source transferred to Soprano peaks when Soprano has two notes sung in a row.

Since LTEs from other sources are build on the LAIS of Soprano, it is seen that when past state is informative aka positive LAIS, the LTE is zero, no extra information from the source has helped in predicting the following state. This is because, as mentioned by Wibral et al. (2014), TE eliminates IS in the past of target process from being mistakenly considered as having been transferred.

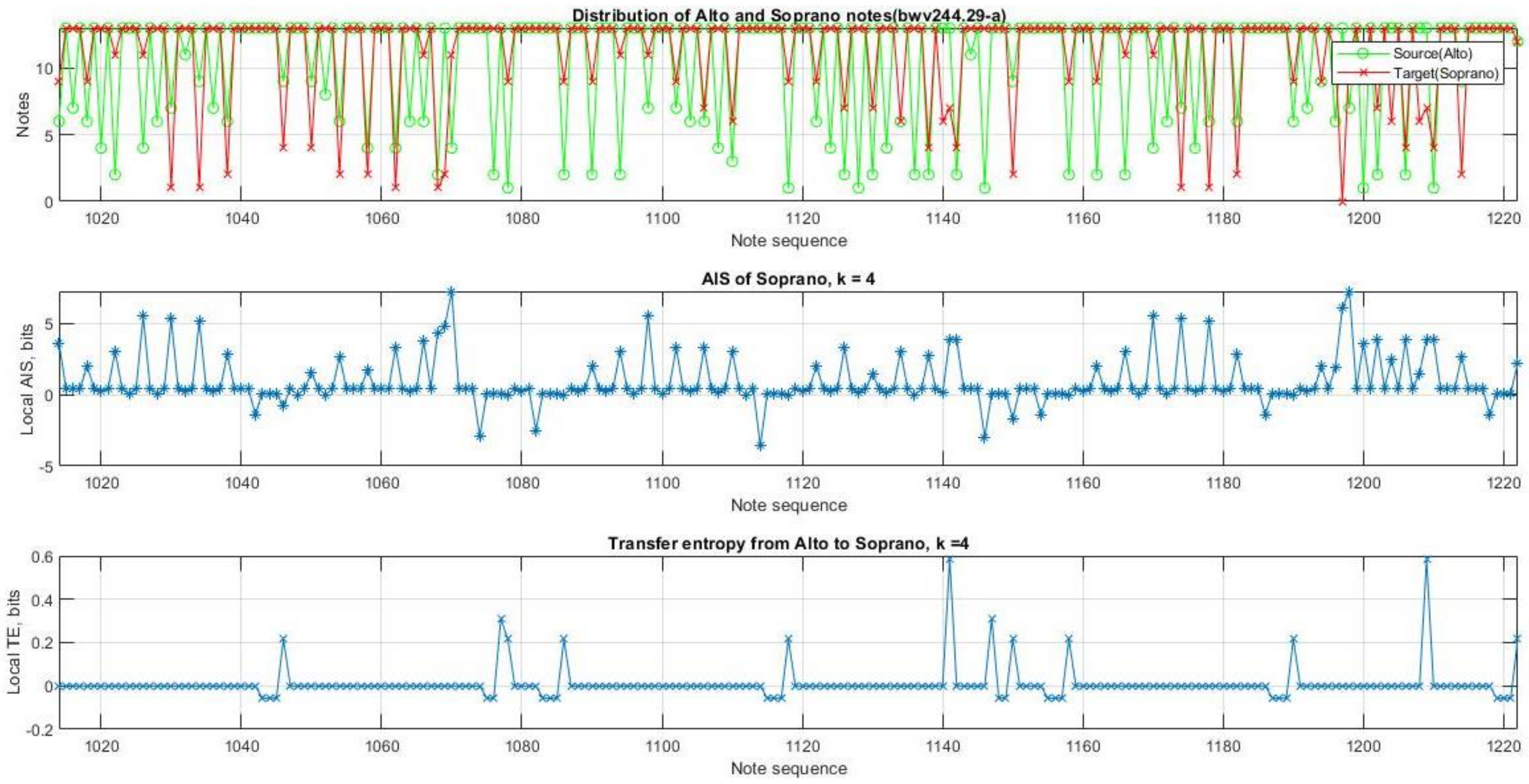


Figure 5-3 Distribution of Alto and Soprano notes (top), LAIS of Soprano with k =4 (middle) and Local Transfer Entropy from Alto to Soprano.

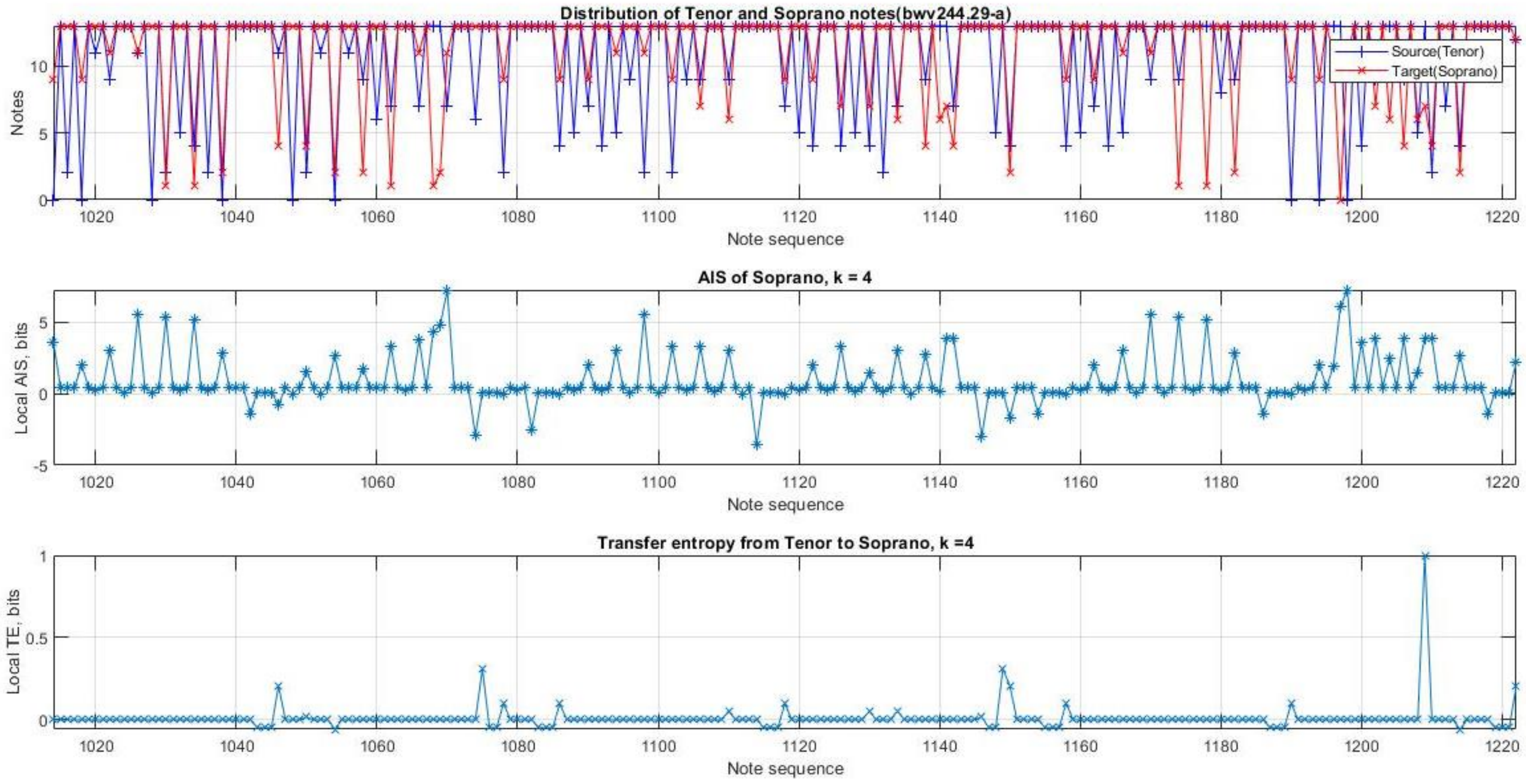


Figure 5-4 Distribution of Tenor and Soprano notes (top), LAIS of Soprano with k =4 (middle) and Local Transfer Entropy from Tenor to Soprano.

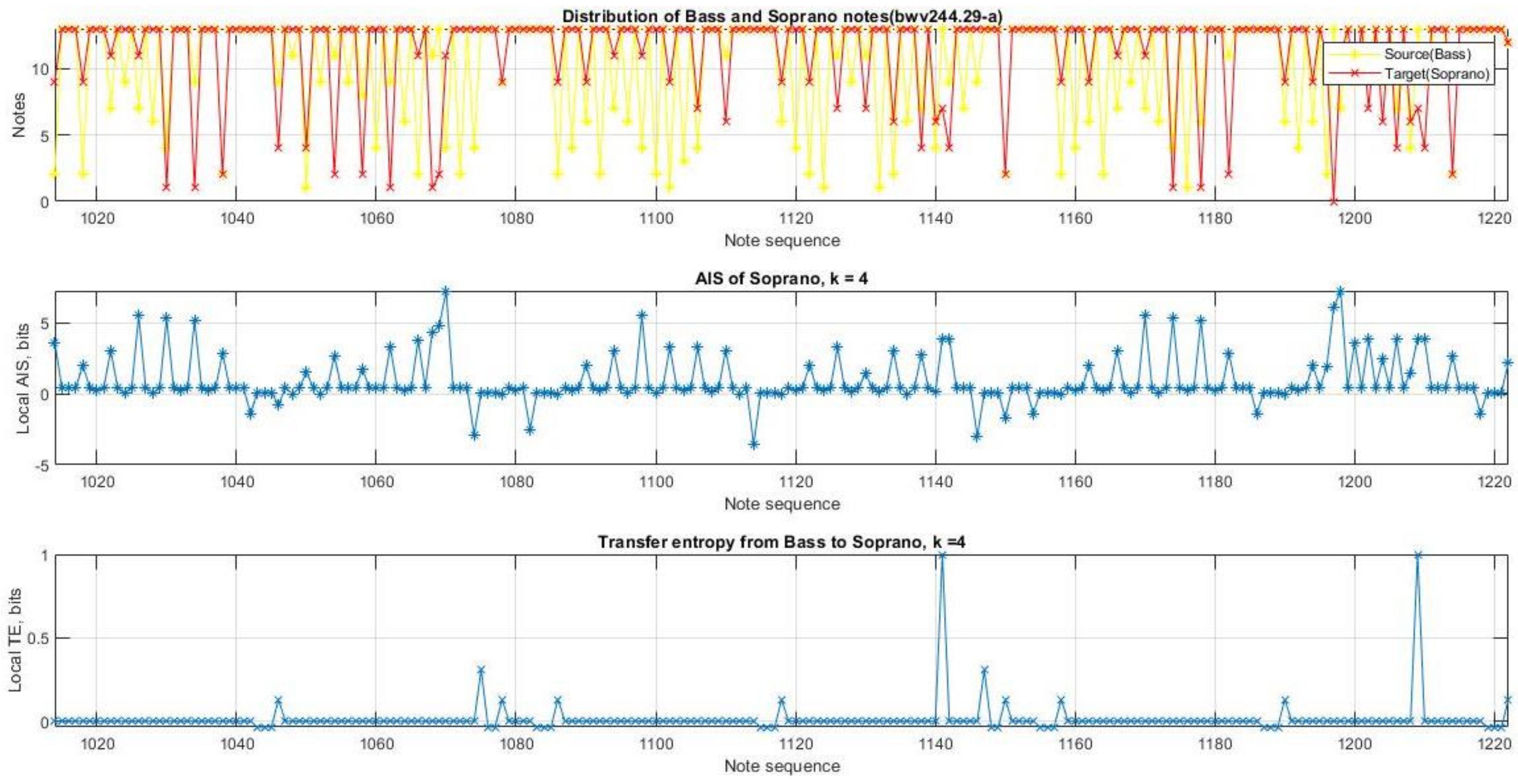


Figure 5-5 Distribution of Bass and Soprano notes (top), LAIS of Soprano with k =4 (middle) and Local Transfer Entropy from Bass to Soprano.

6 Discussion

It is interesting to model the dynamics of Soprano in St Matthew Passion and observe how its past state and neighbours would together reveal their contributions to its next state's prediction. From the results we verify the relationship between AIS and TE, where the latter provides a contrast to former. Also, the transitions in Soprano are examined in TE plots. Lastly, conclusion is drawn by looking at one movement and the fact that the music piece is studied in notes instead of pitches might end up in a fall short in ensuring the result's validity.

7 Conclusion

In conclusion, two sources of information making for the prediction of the next state in Soprano of BWV244.29-a are examined. Overall, AIS of Soprano with a history length of 4 has contributed far more in the prediction than the TE from other voice types, despite the latter gives more information in transitions. In future, it is suggested to include also more past state of sources in calculation, however it will need the use of machine with higher JAVA heap memory. Another possible extension of study is to compute the conditional TE for each variable.

References

Lizier, J. (2014). JIDT: An Information-Theoretic Toolkit for Studying the Dynamics of Complex Systems. *Frontiers in Robotics and AI*, *1*. doi:10.3389/frobt.2014.00011

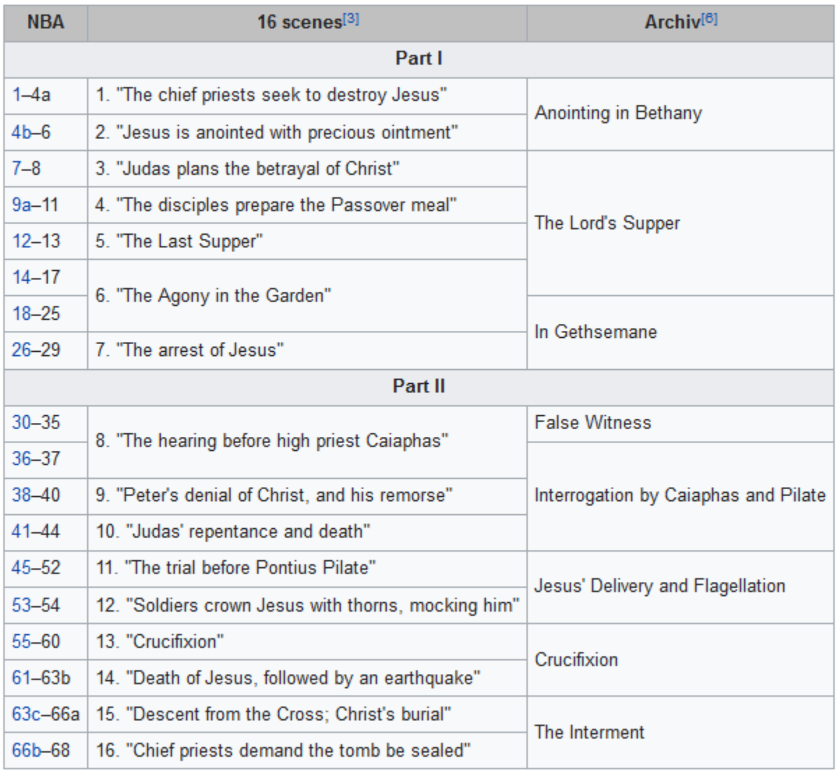
Wibral, M., Lizier, J., Vögler, S., Priesemann, V., & Galuske, R. (2014). Local active information storage as a tool to understand distributed neural information processing. *Frontiers in Neuroinformatics*, *8*(1). doi:10.3389/fninf.2014.00001

Wikipedia. (2020a). St Matthew Passion. Retrieved from https://en.wikipedia.org/wiki/St_Matthew_Passion

Wikipedia. (2020b). St Matthew Passion structure. Retrieved from https://en.wikipedia.org/wiki/St_Matthew_Passion_structure#Musical_structure

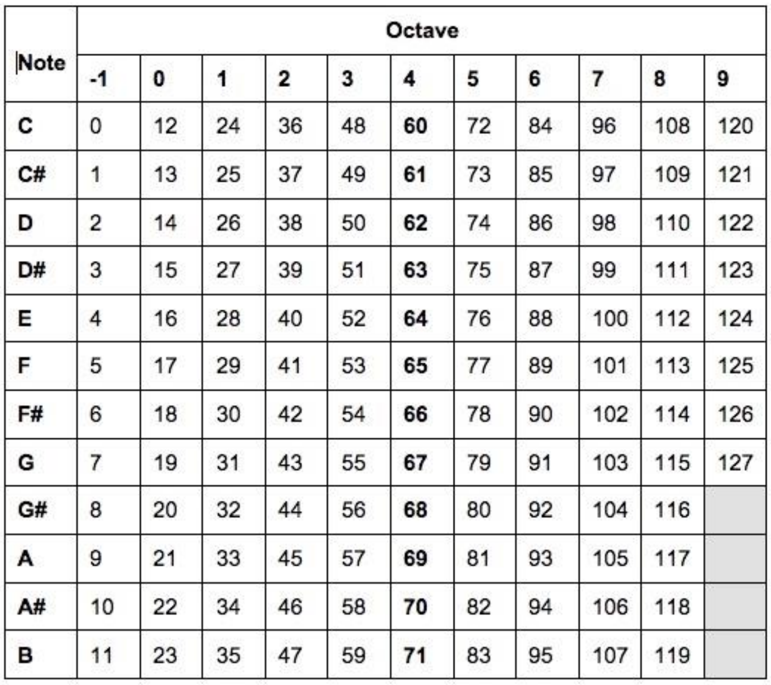
A. Appendix

Report writing



NBA	16 scenes[3]	Archiv[6]
	Part I	
1–4a	1. "The chief priests seek to destroy Jesus"	Anointing in Bethany
4b–6	2. "Jesus is anointed with precious ointment"	
7–8	3. "Judas plans the betrayal of Christ"	The Lord's Supper
9a–11	4. "The disciples prepare the Passover meal"	
12–13	5. "The Last Supper"	
14–17	6. "The Agony in the Garden"	
18–25		In Gethsemane
26–29	7. "The arrest of Jesus"	
	Part II	
30–35	8. "The hearing before high priest Caiaphas"	False Witness
36–37		Interrogation by Caiaphas and Pilate
38–40	9. "Peter's denial of Christ, and his remorse"	
41–44	10. "Judas' repentance and death"	
45–52	11. "The trial before Pontius Pilate"	Jesus' Delivery and Flagellation
53–54	12. "Soldiers crown Jesus with thorns, mocking him"	
55–60	13. "Crucifixion"	Crucifixion
61–63b	14. "Death of Jesus, followed by an earthquake"	
63c–66a	15. "Descent from the Cross; Christ's burial"	The Interment
66b–68	16. "Chief priests demand the tomb be sealed"	

Figure A-1 Scene division based on the Gospel text. Retrieved from: https://en.wikipedia.org/wiki/St_Matthew_Passion_structure#Musical_structure

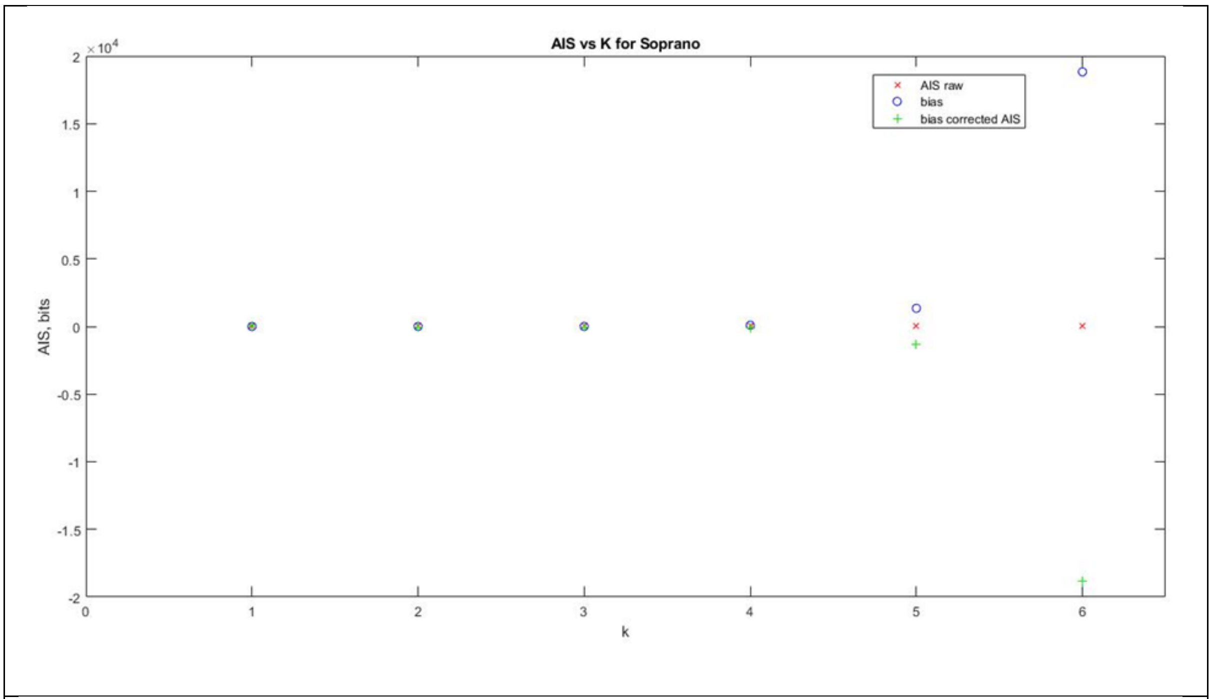


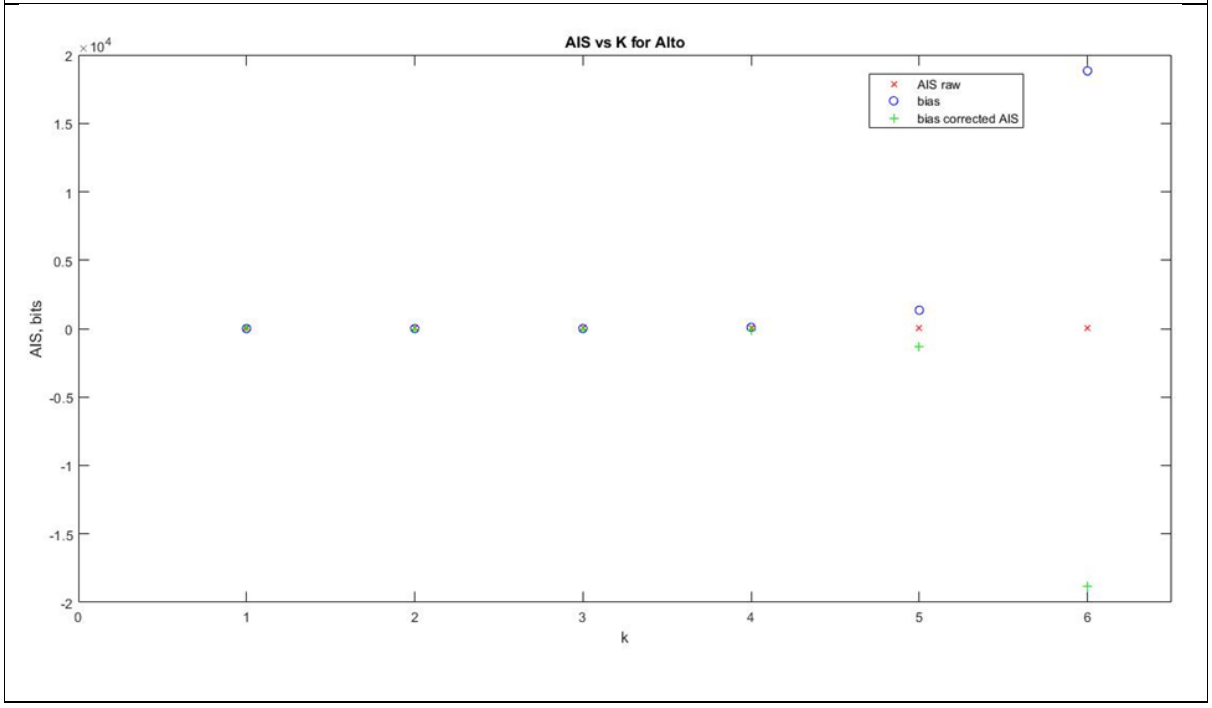
Note	**Octave**										
	-1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
C	0	12	24	36	48	**60**	72	84	96	108	120
C#	1	13	25	37	49	**61**	73	85	97	109	121
D	2	14	26	38	50	**62**	74	86	98	110	122
D#	3	15	27	39	51	**63**	75	87	99	111	123
E	4	16	28	40	52	**64**	76	88	100	112	124
F	5	17	29	41	53	**65**	77	89	101	113	125
F#	6	18	30	42	54	**66**	78	90	102	114	126
G	7	19	31	43	55	**67**	79	91	103	115	127
G#	8	20	32	44	56	**68**	80	92	104	116	
A	9	21	33	45	57	**69**	81	93	105	117	
A#	10	22	34	46	58	**70**	82	94	106	118	
B	11	23	35	47	59	**71**	83	95	107	119	

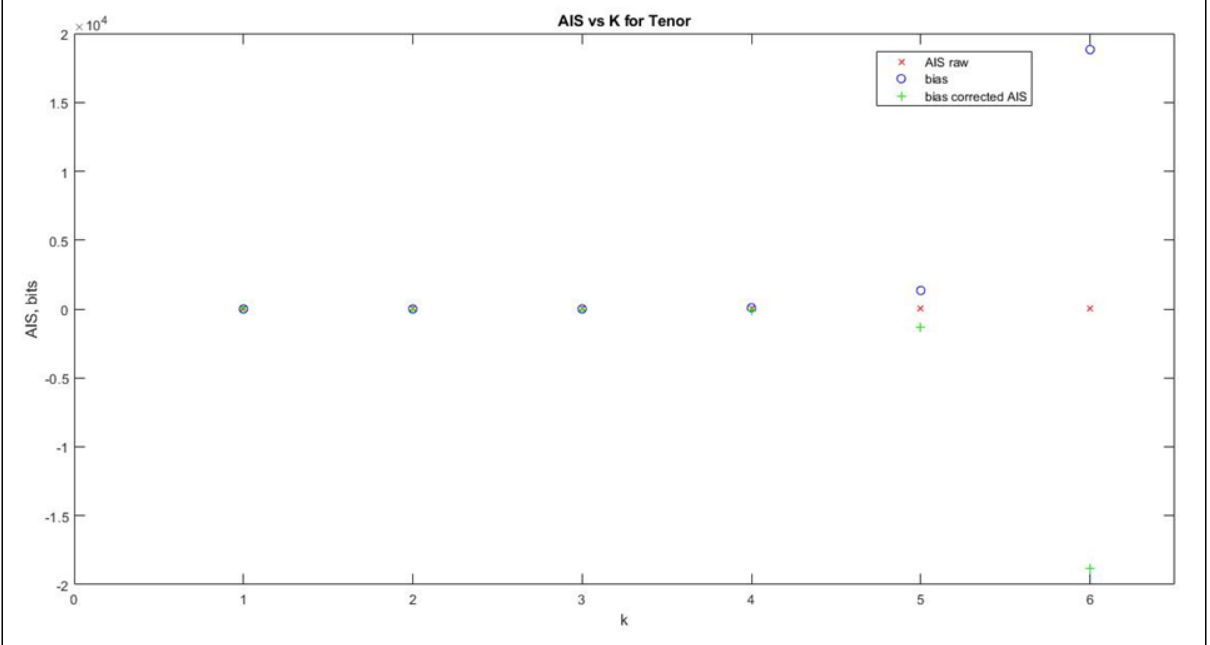
Figure A-2 MIDI notes number chart

Table A-1 Symbols used to represent notes in dataset

Integers occurred in data	Corresponding note	Assigned symbol
36, 48, 60, 72	C	0
37, 49, 61, 73	C#	1
38, 50, 62, 74	D	2
39, 51, 63	D#	3
40, 52, 64, 76	E	4
41, 53, 65, 77	F	5
42, 54, 66	F#	6
43, 55, 67, 79	G	7
44, 56, 68	G#	8
33, 45, 57, 69	A	9
34, 46, 58, 70	A#	10
35, 47, 59, 71	B	11
128	(Stop)	12
129	(No event)	13


AIS vs K for Soprano
×10^4
AIS, bits
k
AIS raw
bias
bias corrected AIS


AIS vs K for Alto
×10^4
AIS, bits
k
AIS raw
bias
bias corrected AIS



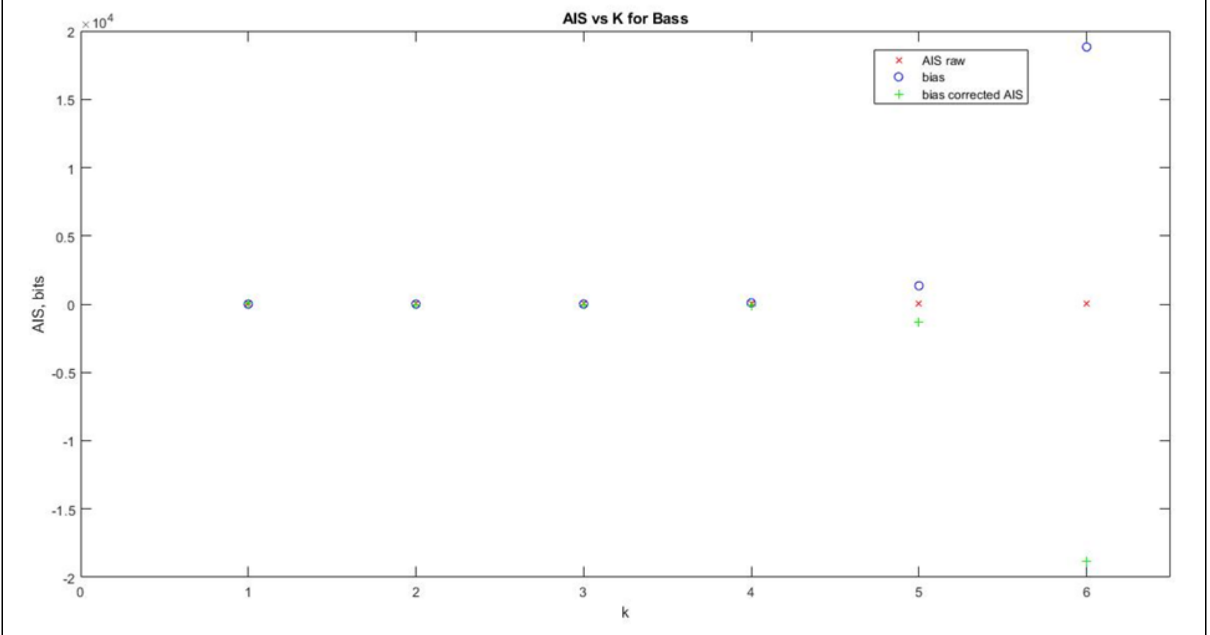


Figure A-3 AIS vs k graphs for Soprano, Alto, Tenor and Bass (top to bottom respectively).

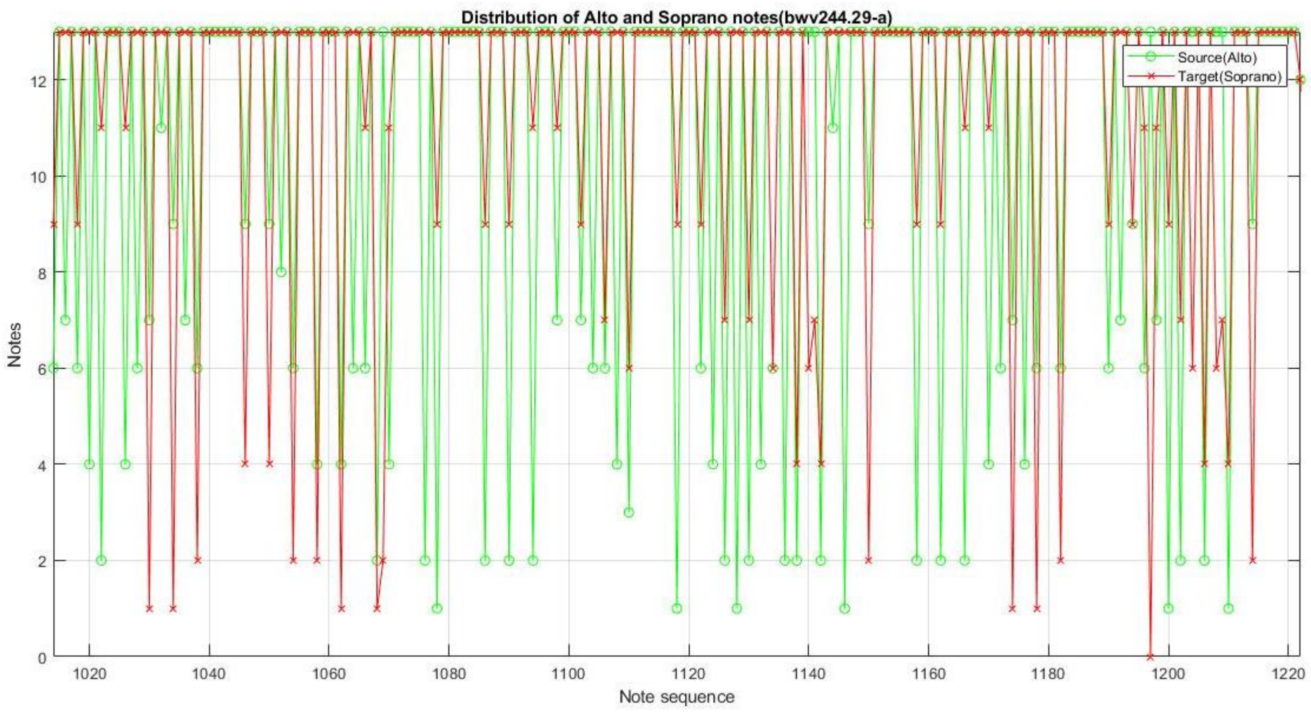


Figure A-4 Distribution of notes of Alto and Soprano in BWV244.29-a

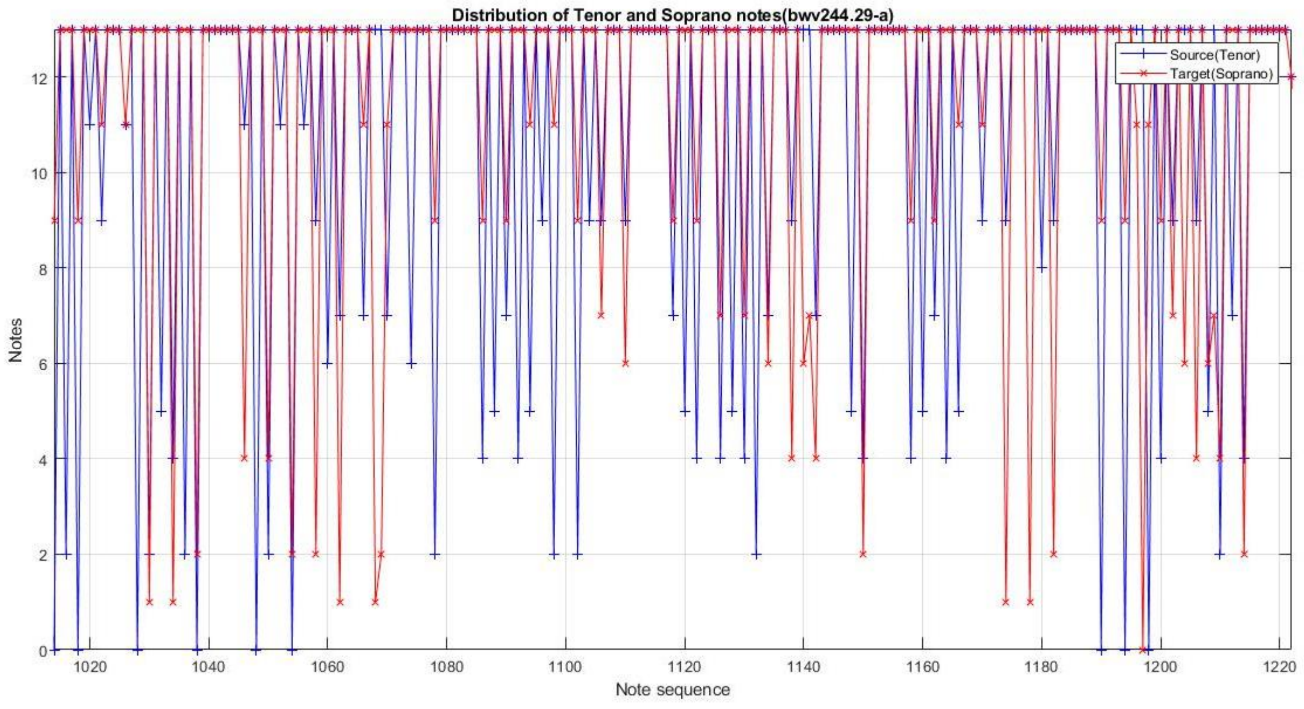


Figure A-5 Distribution of notes of Tenor and Soprano in BWV244.29-a

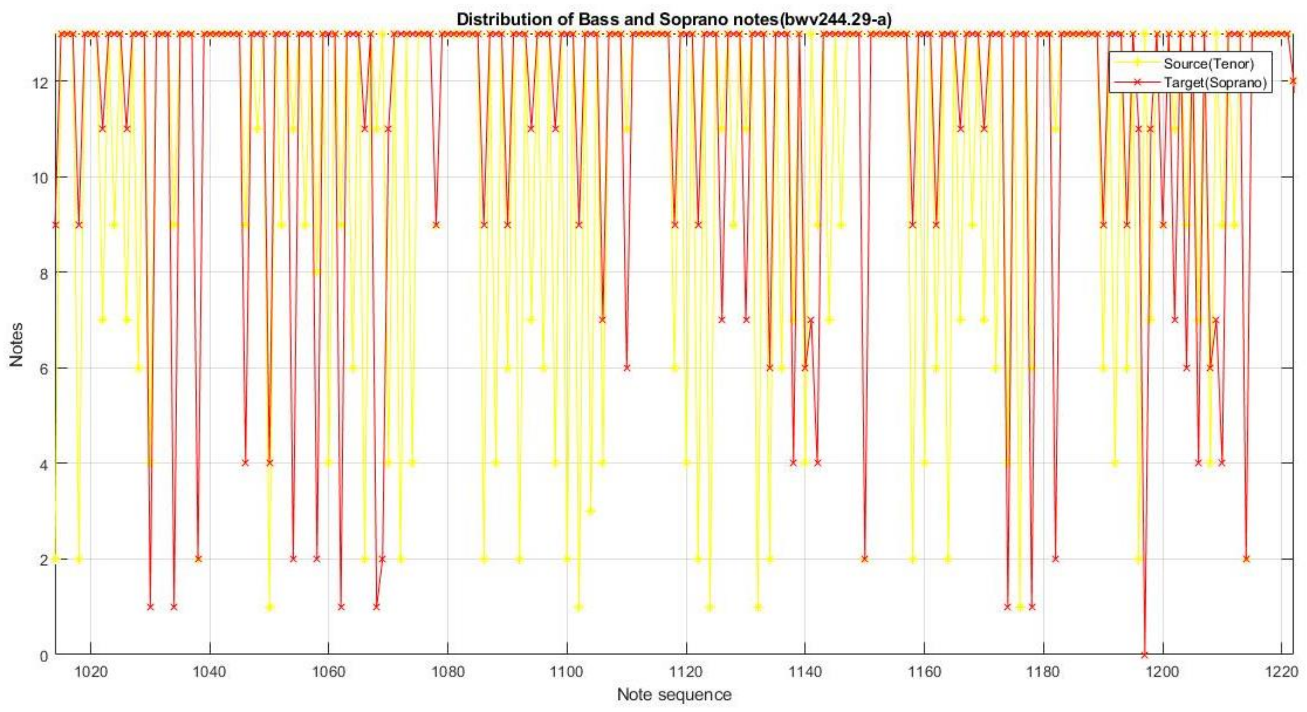


Figure A-6 Distribution of notes of Bass and Soprano in BWV244.29-a

MATLAB code to construct AIS vs K graphs

```
% Add JIDT jar library to the path, and disable warnings that it's already there:
warning('off','MATLAB:Java:DuplicateClass');
javaaddpath('C:\Users\maria\OneDrive\Documents\infodynamics-dist-1.5\infodynamics.jar');
% Add utilities to the path
addpath('C:\Users\maria\OneDrive\Documents\infodynamics-dist-1.5\demos\octave');

data = load('C:\Users\maria\OneDrive\Documents\1USyd\2nd_sem\CSYS5030\assessment\Assignment3-
4\bwv244_D_notes.csv');
% Column indices start from 1 in Matlab:
variable = octaveToJavaIntArray(data(:,4));

for k = 1:6
% 1. Construct the calculator:
calc = javaObject('infodynamics.measures.discrete.ActiveInformationCalculatorDiscrete', 14,
k);
% 2. No other properties to set for discrete calculators.
% 3. Initialise the calculator for (re-)use:
calc.initialise();
% 4. Supply the sample data:
calc.addObservations(variable);
% 5. Compute the estimate:
result = calc.computeAverageLocalOfObservations();
results(k) = result;
% 6. Compute the (statistical significance via) null distribution analytically:
measDist = calc.computeSignificance();
```

```
bias(k) = measDist.getMeanOfDistribution();

fprintf('AIS_Discrete(all cols, k=%d) = %.4f bits from %d samples (bias %.4f,bias
corrected %.4f)\n', ...
    k, result, calc.getNumObservations(), bias(k), result - bias(k));

end

figure(7);
plot(results, 'rx');
hold on;
plot(bias, 'bo');
biasCorrectedAIS = results - bias;
plot(biasCorrectedAIS, 'g+');
hold off;
xlabel('k');
ylabel('AIS, bits');
% ylim([-4.5 4.5]);
xlim([0 6.5]);
legend('AIS raw', 'bias', 'bias corrected AIS')
title('AIS vs K for Bass')

% [value index] = max(biasCorrectedAIS)
% calc.getNumObservations()

% Result for Soprano
% AIS_Discrete(all cols, k=1) = 0.1215 bits from 2604 samples (bias 0.0325,bias corrected
0.0891)
% AIS_Discrete(all cols, k=2) = 0.2490 bits from 2603 samples (bias 0.4869,bias corrected -
0.2380)
% AIS_Discrete(all cols, k=3) = 0.4368 bits from 2602 samples (bias 6.8522,bias corrected -
6.4155)
% AIS_Discrete(all cols, k=4) = 0.8291 bits from 2601 samples (bias 96.0006,bias corrected -
95.1715)
% AIS_Discrete(all cols, k=5) = 0.8561 bits from 2600 samples (bias 1344.5575,bias corrected
-1343.7014)
% AIS_Discrete(all cols, k=6) = 0.9656 bits from 2599 samples (bias 18831.0802,bias corrected
-18830.1146)

% Result for Alto
% AIS_Discrete(all cols, k=1) = 0.1788 bits from 2604 samples (bias 0.0325,bias corrected
0.1464)
% AIS_Discrete(all cols, k=2) = 0.3367 bits from 2603 samples (bias 0.4869,bias corrected -
0.1502)
% AIS_Discrete(all cols, k=3) = 0.5319 bits from 2602 samples (bias 6.8522,bias corrected -
6.3203)
% AIS_Discrete(all cols, k=4) = 0.9491 bits from 2601 samples (bias 96.0006,bias corrected -
95.0515)
% AIS_Discrete(all cols, k=5) = 0.9678 bits from 2600 samples (bias 1344.5575,bias corrected
-1343.5897)
% AIS_Discrete(all cols, k=6) = 1.1408 bits from 2599 samples (bias 18831.0802,bias corrected
-18829.9394)

% Result for Tenor
% AIS_Discrete(all cols, k=1) = 0.1790 bits from 2604 samples (bias 0.0325,bias corrected
0.1465)
% AIS_Discrete(all cols, k=2) = 0.3403 bits from 2603 samples (bias 0.4869,bias corrected -
0.1466)
```

```
% AIS_Discrete(all cols, k=3) = 0.5226 bits from 2602 samples (bias 6.8522,bias corrected -
6.3297)
% AIS_Discrete(all cols, k=4) = 0.9229 bits from 2601 samples (bias 96.0006,bias corrected -
95.0777)
% AIS_Discrete(all cols, k=5) = 0.9451 bits from 2600 samples (bias 1344.5575,bias corrected
-1343.6124)
% AIS_Discrete(all cols, k=6) = 1.1688 bits from 2599 samples (bias 18831.0802,bias corrected
-18829.9114)

% Result for Bass
% AIS_Discrete(all cols, k=1) = 0.1792 bits from 2604 samples (bias 0.0325,bias corrected
0.1467)
% AIS_Discrete(all cols, k=2) = 0.3765 bits from 2603 samples (bias 0.4869,bias corrected -
0.1105)
% AIS_Discrete(all cols, k=3) = 0.5537 bits from 2602 samples (bias 6.8522,bias corrected -
6.2985)
% AIS_Discrete(all cols, k=4) = 0.9808 bits from 2601 samples (bias 96.0006,bias corrected -
95.0198)
% AIS_Discrete(all cols, k=5) = 0.9966 bits from 2600 samples (bias 1344.5575,bias corrected
-1343.5609)
% AIS_Discrete(all cols, k=6) = 1.1759 bits from 2599 samples (bias 18831.0802,bias corrected
-18829.9043)
```

MATLAB code for LAIS and LTE

Section 1 setup and load data

```
% Add JIDT jar library to the path, and disable warnings that it's already there:
warning('off','MATLAB:Java:DuplicateClass');
javaaddpath('C:\Users\maria\OneDrive\Documents\infodynamics-dist-1.5\infodynamics.jar');
% Add utilities to the path
addpath('C:\Users\maria\OneDrive\Documents\infodynamics-dist-1.5\demos\octave');

% 0. Load/prepare the data:
data = load('C:\Users\maria\OneDrive\Documents\1USyd\2nd_sem\CSYS5030\assessment\Assignment3-
4\bwv244_D_notes.csv');
```

Section 2 Active Information Storage

```
variable = octaveToJavaIntArray(data(:,1)); % 1 = Soprano, 2 = Alto, 3 = Tenor, 4 = Bass

k = 4;
% 1. Construct the calculator:
calc1 = javaObject('infodynamics.measures.discrete.ActiveInformationCalculatorDiscrete', 14,
k);
% 2. No other properties to set for discrete calculators.
% 3. Initialise the calculator for (re-)use:
calc1.initialise();
% 4. Supply the sample data:
calc1.addObservations(variable);
% 5. Compute the estimate:
result1 = calc1.computeAverageLocalOfObservations();
% 6. Compute the (statistical significance via) null distribution empirically (e.g. with 100
permutations):
measDist1 = calc1.computeSignificance(100);

fprintf('AIS_Discrete(col_0) = %.4f bits (null: %.4f +/- %.4f std dev.; p(surrogate >
measured)=%.5f from %d surrogates)\n', ...
       result1, measDist1.getMeanOfDistribution(), measDist1.getStdOfDistribution(),
measDist1.pValue, 100);

% Pull out the local AIS values for each point in the time series:
localAISValues = calc1.computeLocalFromPreviousObservations(variable);
```

Section 3 Transfer Entropy

```
% Column indices start from 1 in Matlab:
% 1 = Soprano, 2 = Alto, 3 = Tenor, 4 = Bass
source = octaveToJavaIntArray(data(:,4));
destination = octaveToJavaIntArray(data(:,1));

% 1. Construct the calculator:
calc2 = javaObject('infodynamics.measures.discrete.TransferEntropyCalculatorDiscrete', 14, k,
1, 1, 1, 1);
% 2. No other properties to set for discrete calculators.
% 3. Initialise the calculator for (re-)use:
calc2.initialise();
% 4. Supply the sample data:
calc2.addObservations(source, destination);
% 5. Compute the estimate:
result2 = calc2.computeAverageLocalOfObservations();
% 6. Compute the (statistical significance via) null distribution empirically (e.g. with 100
permutations):
measDist2 = calc2.computeSignificance(100);

fprintf('TE_Discrete(col_0 -> col_1) = %.4f bits (null: %.4f +/- %.4f std dev.; p(surrogate >
measured)=%.5f from %d surrogates)\n', ...
       result2, measDist2.getMeanOfDistribution(), measDist2.getStdOfDistribution(),
measDist2.pValue, 100);

% Pull out the local TE values for each point in the time series:
localTE = calc2.computeLocalFromPreviousObservations(source, destination);
localTE = javaMatrixToOctave(localTE);
```

Section 4 Plot graphs

```
% Combined plot
figure(10);

tiledlayout(3,1);
nexttile
% plot(data(:,2), '-go');
% plot(data(:,3), '-b+');
plot(data(:,4), '-y*');
hold on;
plot(data(:,1), '-rx');
hold off;
title('Distribution of Bass and Soprano notes(bwv244.29-a)');
```

```
xlabel('Note sequence');
ylabel('Notes');
legend('Source(Bass)', 'Target(Soprano)');
xlim([1014 1222]);
ylim([0 13]);
grid;

nexttile
plot(k+1:length(localAISValues), localAISValues(k+1:end), '-*');
title('AIS of Soprano, k = 4')
ylabel('Local AIS, bits');
xlabel('Note sequence');
xlim([1014 1222]);
grid;

nexttile
plot(k+1:length(localTE), localTE(k+1:end), '-x');
% BreakPlot(k+1:length(localTE),localTE(k+1:end),0.35,0.9,'Line');
title('Transfer entropy from Bass to Soprano, k =4')
ylabel('Local TE, bits');
xlabel('Note sequence');
xlim([1014 1222]);
grid;

% Individual plots

figure(11);
% plot(data(:,2), '-go');
% plot(data(:,3), '-b+');
plot(data(:,4), '-y*');
hold on;
plot(data(:,1), '-rx');
hold off;
title('Distribution of Bass and Soprano notes(bwv244.29-a)');
xlabel('Note sequence');
ylabel('Notes');
legend('Source(Tenor)', 'Target(Soprano)');
xlim([1014 1222]);
ylim([0 13]);
grid;

figure(12);
plot(k+1:length(localAISValues), localAISValues(k+1:end), '-*');
title('AIS of Soprano, k = 4')
ylabel('Local AIS, bits');
xlabel('Note sequence');
xlim([1014 1222]);
grid;
% Result: AIS_Discrete(col_0) = 0.8291 bits (null: 0.2184 +/- 0.0088 std dev.; p(surrogate >
measured)=0.00000 from 100 surrogates)

figure(13);
plot(k+1:length(localTE), localTE(k+1:end), '-x');
% BreakPlot(k+1:length(localTE),localTE(k+1:end),0.4,0.8,'Line');
title('Transfer entropy from Bass to Soprano, k =4')
ylabel('Local TE, bits');
xlabel('Note sequence');
xlim([1014 1222]);
```

```
grid;
% Result: TE_Discrete(col_0 -> col_1) = 0.0075 bits (null: 0.1280 +/- 0.0069 std dev.;
p(surrogate > measured)=1.00000 from 100 surrogates)
```

Python code for data pre-processing and preparation:

(refer to the following page)

CSYS5030_500386873

December 1, 2020

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1 Imports

```
[ ]: from itertools import groupby
     import music21 as m21
     import pandas as pd
     import numpy as np
     import csv
```

2 List of movements of St Matthew Passion

```
[ ]: bwv244_list = ['bach/bwv244.3.mxl','bach/bwv244.10.mxl','bach/bwv244.15.
      ↪mxl','bach/bwv244.17.mxl',\
                    'bach/bwv244.25.mxl','bach/bwv244.29-a.mxl','bach/bwv244.32.
      ↪mxl','bach/bwv244.37.mxl',\
                    'bach/bwv244.40.mxl','bach/bwv244.44.mxl','bach/bwv244.46.mxl',\
                    'bach/bwv244.54.mxl','bach/bwv244.62.mxl']
```

3 Important function

```
[ ]: ### Code adapted from
     ### https://colab.research.google.com/github/cpmpercussion/creative-prediction/
      ↪blob/master/notebooks/3-zeldic-musical-RNN.ipynb
     ### It is modified to have transpose function

     MELODY_NOTE_OFF = 128 # (stop playing all previous notes)
     MELODY_NO_EVENT = 129 # (no change from previous event)
     def streamToNoteArray(stream, transpose=False):
         """
         Convert a Music21 sequence to a numpy array of int8s into Melody-RNN format:
             0-127 - note on at specified pitch
```

```
        128   - note off
        129   - no event
    """
    # Part 0, transpose the notes
    if transpose:
      k = stream.analyze('key')
      # print("Expected key: ",k)
      # print("Alternative key(s):")
      # for analysis in k.alternateInterpretations:
      #   if (analysis.correlationCoefficient > 0.5):
      #       print(analysis)
      i = m21.interval.Interval(k.tonic, m21.pitch.Pitch('D'))
      stream = stream.transpose(i)

    # Part one, extract from stream
    total_length = np.int(np.round(stream.flat.highestTime / 0.25)) # in␣
↪semiquavers
    stream_list = []
    for element in stream.flat:
        if isinstance(element, m21.note.Note):
            stream_list.append([np.round(element.offset / 0.25), np.
↪round(element.quarterLength / 0.25), element.pitch.midi])
        elif isinstance(element, m21.chord.Chord):
            stream_list.append([np.round(element.offset / 0.25), np.
↪round(element.quarterLength / 0.25), element.sortAscending().pitches[-1].
↪midi])
    np_stream_list = np.array(stream_list, dtype=np.int)
    df = pd.DataFrame({'pos': np_stream_list.T[0], 'dur': np_stream_list.T[1],␣
↪'pitch': np_stream_list.T[2]})
    df = df.sort_values(['pos','pitch'], ascending=[True, False]) # sort the␣
↪dataframe properly
    df = df.drop_duplicates(subset=['pos']) # drop duplicate values
    # part 2, convert into a sequence of note events
    output = np.zeros(total_length+1, dtype=np.int16) + np.
↪int16(MELODY_NO_EVENT)  # set array full of no events by default.
    # Fill in the output list
    for i in range(total_length):
        if not df[df.pos==i].empty:
            n = df[df.pos==i].iloc[0] # pick the highest pitch at each␣
↪semiquaver
            output[i] = n.pitch # set note on
            output[i+n.dur] = MELODY_NOTE_OFF
    return output
```

4 Original sequence extracting

4.1 Soprano

```
[ ]: # Get a list of pitch arrays
     bwv244_sop_pitch_arrays = []
     for bwv_part in bwv244_list:
       piece = m21.corpus.parse(bwv_part)
       sop = piece.parts[0]
       pitch_array = streamToNoteArray(sop)
       bwv244_sop_pitch_arrays.append(pitch_array)

     # Check the length of each part in the list
     for i in bwv244_sop_pitch_arrays:
       print(i.shape)

     # Concatenate the arrays into one and get some info about the array
     bwv244_sop_pitch_whole = np.concatenate(np.array(bwv244_sop_pitch_arrays))
     print("Number of notes:")
     print(bwv244_sop_pitch_whole.shape)
     bwv244_sop_pitch_whole_df = pd.DataFrame(bwv244_sop_pitch_whole)
     print("Notes that do appear:")
     unique_sop, counts_sop = np.unique(bwv244_sop_pitch_whole, return_counts=True)
     print(unique_sop)
     print("with occurence:")
     print(counts_sop)
     print("Notes that don't appear:")
     print(np.setdiff1d(np.arange(0,129),unique_sop))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 60  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
  79 128 129]
with occurence:
```

```
[   2    2    2   16   14   26   45   19   70   37   75   57   72   95
   31   42   15   13    7   13 1952]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  61  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127]
```

4.2 Alto

```
[ ]: bwv244_alto_pitch_arrays = []
     for bwv_part in bwv244_list:
       piece = m21.corpus.parse(bwv_part)
       alto = piece.parts[1]
       pitch_array = streamToNoteArray(alto)
       bwv244_alto_pitch_arrays.append(pitch_array)

     for i in bwv244_alto_pitch_arrays:
       print(i.shape)

     bwv244_alto_pitch_whole = np.concatenate(np.array(bwv244_alto_pitch_arrays))
     print("Number of notes:")
     print(bwv244_alto_pitch_whole.shape)
     bwv244_alto_pitch_whole_df = pd.DataFrame(bwv244_alto_pitch_whole)
     print("Notes that do appear:")
     unique_alto, counts_alto = np.unique(bwv244_alto_pitch_whole,␣
      ↪return_counts=True)
     print(unique_alto)
     print("with occurence:")
     print(counts_alto)
     print("Notes that don't appear:")
     print(np.setdiff1d(np.arange(0,129),unique_alto))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
```

```
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 55  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73
  74 128 129]
with occurence:
[   1    2    2   10   25   15   71   53   99   73  100   77   60   75
   22   35   11    3    1   13 1857]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  56  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127]
```

4.3 Tenor

[ ]:
```
bwv244_tenor_pitch_arrays = []
for bwv_part in bwv244_list:
  piece = m21.corpus.parse(bwv_part)
  tenor = piece.parts[2]
  pitch_array = streamToNoteArray(tenor)
  bwv244_tenor_pitch_arrays.append(pitch_array)

for i in bwv244_tenor_pitch_arrays:
  print(i.shape)

bwv244_tenor_pitch_whole = np.concatenate(np.array(bwv244_tenor_pitch_arrays))
print("Number of notes:")
print(bwv244_tenor_pitch_whole.shape)
bwv244_tenor_pitch_whole_df = pd.DataFrame(bwv244_tenor_pitch_whole)
print("Notes that do appear:")
unique_tenor, counts_tenor = np.unique(bwv244_tenor_pitch_whole,␣
 ↪return_counts=True)
print(unique_tenor)
print("with occurence:")
print(counts_tenor)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique_tenor))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
```

```
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
  69 128 129]
with occurence:
[   1    5   14   12   45   31   94   76   90   77   71   93   29   59
   20   26   13    3    2   13 1831]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127]
```

4.4 Bass

[ ]:
```
bwv244_bass_pitch_arrays = []
for bwv_part in bwv244_list:
  piece = m21.corpus.parse(bwv_part)
  bass = piece.parts[3]
  pitch_array = streamToNoteArray(bass)
  bwv244_bass_pitch_arrays.append(pitch_array)

for i in bwv244_bass_pitch_arrays:
  print(i.shape)

bwv244_bass_pitch_whole = np.concatenate(np.array(bwv244_bass_pitch_arrays))
print("Number of notes:")
print(bwv244_bass_pitch_whole.shape)
bwv244_bass_pitch_whole_df = pd.DataFrame(bwv244_bass_pitch_whole)
print("Notes that do appear:")
unique_bass, counts_bass = np.unique(bwv244_bass_pitch_whole,␣
 ↪return_counts=True)
print(unique_bass)
print("with occurence:")
print(counts_bass)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique_bass))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 38  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
  57  58  59  60  61  62  63  64 128 129]
with occurence:
[   1    2    2    4   11    9   29   22   34   35   34   70   38   76
   48   72   64   44   62   20   36   10    6    5    1    1   13 1856]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  39  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
 116 117 118 119 120 121 122 123 124 125 126 127]
```

4.5 whole analysis

```
[ ]: # Pool all together to check the distribution of pitches
     all_notes = np.concatenate((bwv244_sop_pitch_whole,bwv244_alto_pitch_whole, \
                                 bwv244_tenor_pitch_whole, bwv244_bass_pitch_whole))
     print("Number of notes:")
     print(all_notes.shape)
     all_notes_df = pd.DataFrame(all_notes)
     print("Notes that do appear:")
     unique, counts = np.unique(all_notes, return_counts=True)
     print(unique)
     print("with occurence:")
     print(counts)
     print("Notes that don't appear:")
     print(np.setdiff1d(np.arange(0,129),unique))
```

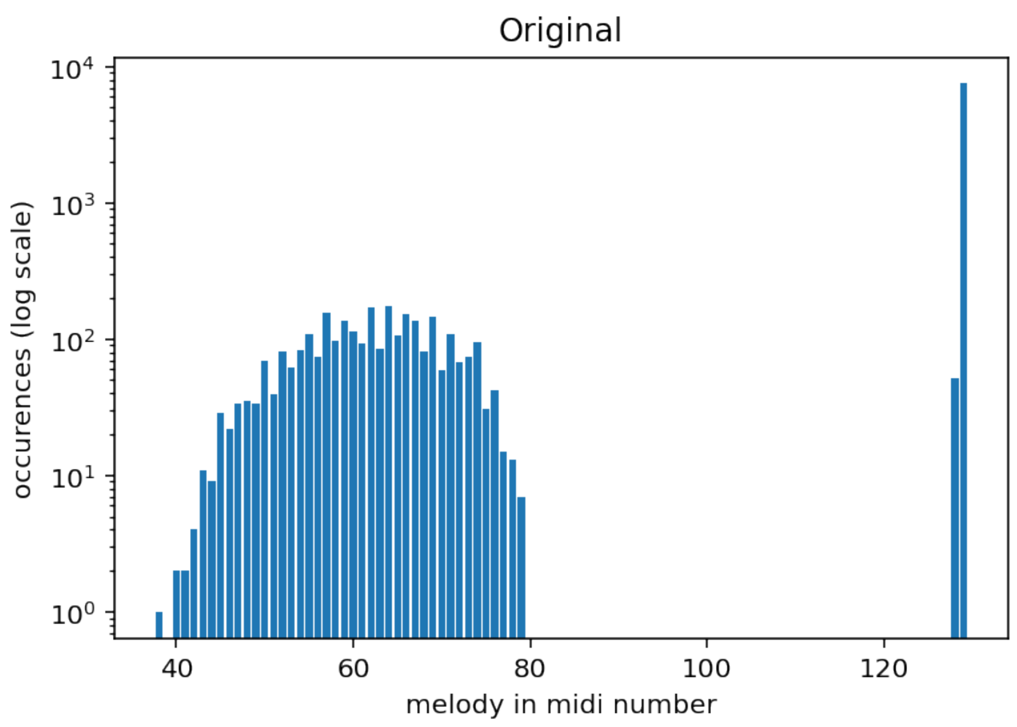
```
Number of notes:
(10420,)
Notes that do appear:
[ 38  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
  75  76  77  78  79 128 129]
with occurence:
[   1    2    2    4   11    9   29   22   34   35   34   70   39   81
   62   84  110   75  158   98  136  114   92  171   85  175  107  152
  135   82  147   59  110   68   75   96   31   42   15   13    7   52
 7496]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  39  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94
  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112
 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127]
```

[ ]:
```
print("Plot the relative occurences of each note:")
import matplotlib.pyplot as plt
%matplotlib inline

#plt.style.use('dark_background')
plt.bar(unique, counts)
plt.yscale('log')
plt.xlabel('melody in midi number')
plt.ylabel('occurences (log scale)')
plt.title('Original')
```

```
Plot the relative occurences of each note:
```

[ ]:
```
Text(0.5, 1.0, 'Original')
```



```
[ ]: pitch_original = unique.tolist()
     pitch_sym = [i for i in range(len(unique))]
     sym_dict = dict(zip(pitch_original, pitch_sym))
```

```
[ ]: sym_dict
```

```
[ ]: {38: 0,
      40: 1,
      41: 2,
      42: 3,
      43: 4,
      44: 5,
      45: 6,
      46: 7,
      47: 8,
      48: 9,
      49: 10,
      50: 11,
      51: 12,
      52: 13,
      53: 14,
      54: 15,
      55: 16,
      56: 17,
      57: 18,
      58: 19,
      59: 20,
      60: 21,
      61: 22,
      62: 23,
      63: 24,
      64: 25,
      65: 26,
      66: 27,
      67: 28,
      68: 29,
      69: 30,
      70: 31,
      71: 32,
      72: 33,
      73: 34,
      74: 35,
      75: 36,
      76: 37,
      77: 38,
      78: 39,
      79: 40,
      128: 41,
      129: 42}
```

```
[ ]: # 51-59, 60-71, 71-79, 128, 129
     note_sym = [2]+[i for i in range(4,12)]+[i for i in range(12)]+[i for i in␣
      ↪range(12)]+[i for i in range(8)]+[12,13]
     note_dict = dict(zip(pitch_original, note_sym))
```

```
[ ]: note_dict
```

```
[ ]: {38: 2,
      40: 4,
      41: 5,
      42: 6,
      43: 7,
      44: 8,
      45: 9,
      46: 10,
      47: 11,
      48: 0,
      49: 1,
      50: 2,
      51: 3,
      52: 4,
      53: 5,
      54: 6,
      55: 7,
      56: 8,
      57: 9,
      58: 10,
      59: 11,
      60: 0,
      61: 1,
      62: 2,
      63: 3,
      64: 4,
      65: 5,
      66: 6,
      67: 7,
      68: 8,
      69: 9,
      70: 10,
      71: 11,
      72: 0,
      73: 1,
      74: 2,
      75: 3,
      76: 4,
      77: 5,
      78: 6,
      79: 7,
      128: 12,
      129: 13}
```

[ ]:
```
# Downsize the symbol so that only count the pitches that appear
pitch_series1 = list(map(sym_dict.get, bwv244_sop_pitch_whole))
pitch_series2 = list(map(sym_dict.get, bwv244_alto_pitch_whole))
pitch_series3 = list(map(sym_dict.get, bwv244_tenor_pitch_whole))
pitch_series4 = list(map(sym_dict.get, bwv244_bass_pitch_whole))

# Write the 4 variables into one single csv
with open('bwv244.csv', mode='w') as csv_file:
  writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.
 ↪QUOTE_MINIMAL)

  for i in range(len(pitch_series1)):
    writer.
 ↪writerow([pitch_series1[i],pitch_series2[i],pitch_series3[i],pitch_series4[i]])
```

```
[ ]: # Further Downsize the symbol to 14 note types
pitch_series12 = list(map(note_dict.get, bwv244_sop_pitch_whole))
pitch_series22 = list(map(note_dict.get, bwv244_alto_pitch_whole))
pitch_series32 = list(map(note_dict.get, bwv244_tenor_pitch_whole))
pitch_series42 = list(map(note_dict.get, bwv244_bass_pitch_whole))

# Write the 4 variables into one single csv
with open('bwv244_notes.csv', mode='w') as csv_file:
  writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.
 ↪QUOTE_MINIMAL)

  for i in range(len(pitch_series12)):
    writer.
 ↪writerow([pitch_series12[i],pitch_series22[i],pitch_series32[i],pitch_series42[i]])
```

5 Sequence extracting after transposing to D

5.1 Soprano

```
[ ]: bwv244_sop_pitch_arrays = []
for bwv_part in bwv244_list:
  piece = m21.corpus.parse(bwv_part)
  sop = piece.parts[0]
  pitch_array = streamToNoteArray(sop, transpose=True)
  bwv244_sop_pitch_arrays.append(pitch_array)

for i in bwv244_sop_pitch_arrays:
  print(i.shape)

bwv244_sop_pitch_whole = np.concatenate(np.array(bwv244_sop_pitch_arrays))
print("Number of notes:")
print(bwv244_sop_pitch_whole.shape)
bwv244_sop_pitch_whole_df = pd.DataFrame(bwv244_sop_pitch_whole)
print("Notes that do appear:")
unique_sop, counts_sop = np.unique(bwv244_sop_pitch_whole, return_counts=True)
print(unique_sop)
print("with occurence:")
print(counts_sop)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique_sop))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
```

```
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 55  57  58  59  60  61  62  64  65  66  67  69  70  71  72  73  74  76
  77  79 128 129]
with occurence:
[   6   32   17    3   25    7   75   96   41   77   79   73   13   14
   13    6   22   21   18    2   13 1952]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  56  63  68  75  78  80  81  82  83  84  85  86  87  88  89  90  91
  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109
 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127]
```

5.2 Alto

```
[ ]: bwv244_alto_pitch_arrays = []
     for bwv_part in bwv244_list:
       piece = m21.corpus.parse(bwv_part)
       alto = piece.parts[1]
       pitch_array = streamToNoteArray(alto, transpose=True)
       bwv244_alto_pitch_arrays.append(pitch_array)

     for i in bwv244_alto_pitch_arrays:
       print(i.shape)

     bwv244_alto_pitch_whole = np.concatenate(np.array(bwv244_alto_pitch_arrays))
     print("Number of notes:")
     print(bwv244_alto_pitch_whole.shape)
     bwv244_alto_pitch_whole_df = pd.DataFrame(bwv244_alto_pitch_whole)
     print("Notes that do appear:")
     unique_alto, counts_alto = np.unique(bwv244_alto_pitch_whole,␣
      ↪return_counts=True)
     print(unique_alto)
     print("with occurence:")
     print(counts_alto)
     print("Notes that don't appear:")
     print(np.setdiff1d(np.arange(0,129),unique_alto))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 50  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
  69  70  71  72  73  74 128 129]
with occurence:
[   1    4   25    2   28    2   47   11   19   18   29  122   10   97
   26  114   92   12   58    4    9    1    2    2   13 1857]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  51  75  76  77
  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119 120 121 122 123 124 125 126 127]
```

5.3 Tenor

```
[ ]: bwv244_tenor_pitch_arrays = []
     for bwv_part in bwv244_list:
       piece = m21.corpus.parse(bwv_part)
       tenor = piece.parts[2]
       pitch_array = streamToNoteArray(tenor, transpose=True)
       bwv244_tenor_pitch_arrays.append(pitch_array)

     for i in bwv244_tenor_pitch_arrays:
       print(i.shape)

     bwv244_tenor_pitch_whole = np.concatenate(np.array(bwv244_tenor_pitch_arrays))
     print("Number of notes:")
     print(bwv244_tenor_pitch_whole.shape)
     bwv244_tenor_pitch_whole_df = pd.DataFrame(bwv244_tenor_pitch_whole)
     print("Notes that do appear:")
```

```
unique_tenor, counts_tenor = np.unique(bwv244_tenor_pitch_whole,␣
 ↪return_counts=True)
print(unique_tenor)
print("with occurence:")
print(counts_tenor)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique_tenor))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 43  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61
  62  64  65 128 129]
with occurence:
[   2    9    4   11   11   26   84    4  108   60   76  127   12  136
    5   40   22    4   14    5    1   13 1831]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  44  63  66  67  68  69  70  71  72  73  74
  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92
  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110
 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127]
```

5.4 Bass

[ ]:
```
bwv244_bass_pitch_arrays = []
for bwv_part in bwv244_list:
  piece = m21.corpus.parse(bwv_part)
  bass = piece.parts[3]
  pitch_array = streamToNoteArray(bass, transpose=True)
  bwv244_bass_pitch_arrays.append(pitch_array)

for i in bwv244_bass_pitch_arrays:
  print(i.shape)

bwv244_bass_pitch_whole = np.concatenate(np.array(bwv244_bass_pitch_arrays))
print("Number of notes:")
print(bwv244_bass_pitch_whole.shape)
bwv244_bass_pitch_whole_df = pd.DataFrame(bwv244_bass_pitch_whole)
print("Notes that do appear:")
unique_bass, counts_bass = np.unique(bwv244_bass_pitch_whole,␣
 ↪return_counts=True)
print(unique_bass)
print("with occurence:")
print(counts_bass)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique_bass))
```

```
(177,)
(193,)
(193,)
(257,)
(193,)
(209,)
(177,)
(193,)
(257,)
(193,)
(177,)
(193,)
(193,)
Number of notes:
(2605,)
Notes that do appear:
[ 33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60  61  62 128 129]
with occurence:
[   5    1    1    6    5   29    1   22   27   22   41   12   89   29
   47   28   32   87   10   64   30   44   31   12   36    6   12    3
    1    3   13 1856]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  63  64  65
  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83
  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101
 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
 120 121 122 123 124 125 126 127]
```

5.5 whole analysis

```
[ ]: all_notes2 = np.concatenate((bwv244_sop_pitch_whole,bwv244_alto_pitch_whole, \
                              bwv244_tenor_pitch_whole, bwv244_bass_pitch_whole))
     print("Number of notes:")
     print(all_notes2.shape)
     all_notes2_df = pd.DataFrame(all_notes2)
     print("Notes that do appear:")
     unique2, counts2 = np.unique(all_notes2, return_counts=True)
     print(unique2)
     print("with occurence:")
     print(counts2)
     print("Notes that don't appear:")
     print(np.setdiff1d(np.arange(0,129),unique2))
```

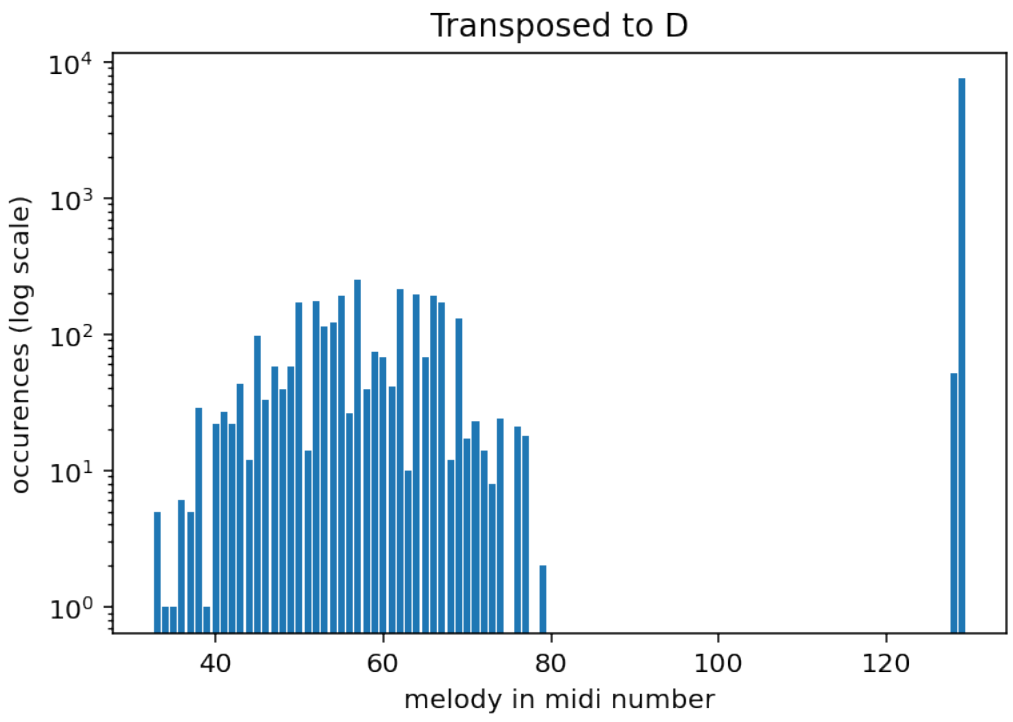
```
Number of notes:
(10420,)
Notes that do appear:
[ 33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
  69  70  71  72  73  74  76  77  79 128 129]
with occurence:
[   5    1    1    6    5   29    1   22   27   22   43   12   98   33
   58   39   58  172   14  176  115  122  192   26  251   39   74   68
   41  214   10  198   68  191  171   12  131   17   23   14    8   24
   21   18    2   52 7496]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  75  78  80
  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98
  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
 117 118 119 120 121 122 123 124 125 126 127]
```

```
[ ]: print("Plot the relative occurences of each note:")
     import matplotlib.pyplot as plt
     %matplotlib inline

     #plt.style.use('dark_background')
     plt.bar(unique2, counts2)
     plt.yscale('log')
     plt.xlabel('melody in midi number')
     plt.ylabel('occurences (log scale)')
     plt.title('Transposed to D')
```

```
Plot the relative occurences of each note:
```

```
[ ]: Text(0.5, 1.0, 'Transposed to D')
```



```
[ ]: unique2.shape
```

```
[ ]: (47,)
```

```
[ ]: pitch_original2 = unique2.tolist()
     pitch_sym2 = [i for i in range(47)]
     sym_dict2 = dict(zip(pitch_original2, pitch_sym2))
```

```
[ ]: sym_dict2
```

```
[ ]: {33: 0,
      34: 1,
      35: 2,
      36: 3,
      37: 4,
      38: 5,
      39: 6,
      40: 7,
      41: 8,
      42: 9,
      43: 10,
      44: 11,
      45: 12,
      46: 13,
      47: 14,
      48: 15,
      49: 16,
      50: 17,
      51: 18,
      52: 19,
      53: 20,
      54: 21,
      55: 22,
      56: 23,
      57: 24,
      58: 25,
      59: 26,
      60: 27,
      61: 28,
      62: 29,
      63: 30,
      64: 31,
      65: 32,
      66: 33,
      67: 34,
      68: 35,
      69: 36,
      70: 37,
      71: 38,
      72: 39,
      73: 40,
      74: 41,
      76: 42,
      77: 43,
      79: 44,
      128: 45,
      129: 46}
```

[ ]:
```
# 33-35, 36-47, 48-59, 60-71, 72-74,76,77,79,128,129
note_sym2 = [i for i in range(9,12)]+[i for i in range(12)]+[i for i in␣
 ↪range(12)]+ [i for i in range(12)] + [i for i in range(3)]+[4,5,7,12,13]
print(len(note_sym2))
note_dict2 = dict(zip(pitch_original2, note_sym2))
```

```
47
```

[ ]:
```
note_dict2
```

[ ]:
```
{33: 9,
 34: 10,
35: 11,
36: 0,
37: 1,
38: 2,
39: 3,
40: 4,
41: 5,
42: 6,
43: 7,
44: 8,
45: 9,
46: 10,
47: 11,
48: 0,
49: 1,
50: 2,
51: 3,
52: 4,
53: 5,
54: 6,
55: 7,
56: 8,
57: 9,
58: 10,
59: 11,
60: 0,
61: 1,
62: 2,
63: 3,
64: 4,
65: 5,
66: 6,
67: 7,
68: 8,
69: 9,
70: 10,
71: 11,
72: 0,
73: 1,
74: 2,
76: 4,
77: 5,
79: 7,
128: 12,
129: 13}
```

```
[ ]: pitch_series21 = list(map(sym_dict2.get, bwv244_sop_pitch_whole))
     pitch_series22 = list(map(sym_dict2.get, bwv244_alto_pitch_whole))
     pitch_series23 = list(map(sym_dict2.get, bwv244_tenor_pitch_whole))
     pitch_series24 = list(map(sym_dict2.get, bwv244_bass_pitch_whole))

     with open('bwv244_D.csv', mode='w') as csv_file:
       writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.
      ↪QUOTE_MINIMAL)

       for i in range(len(pitch_series21)):
         writer.
      ↪writerow([pitch_series21[i],pitch_series22[i],pitch_series23[i],pitch_series24[i]])
```

```
[ ]: pitch_series212 = list(map(note_dict2.get, bwv244_sop_pitch_whole))
     pitch_series222 = list(map(note_dict2.get, bwv244_alto_pitch_whole))
     pitch_series232 = list(map(note_dict2.get, bwv244_tenor_pitch_whole))
     pitch_series242 = list(map(note_dict2.get, bwv244_bass_pitch_whole))

     with open('bwv244_D_notes.csv', mode='w') as csv_file:
       writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.
      ↪QUOTE_MINIMAL)

       for i in range(len(pitch_series212)):
         writer.
      ↪writerow([pitch_series212[i],pitch_series222[i],pitch_series232[i],pitch_series242[i]])
```