Introduction to Musical Corpus Studies

Release 0.0.1

Fabian C. Moss

Nov 12, 2020

CONTENT

Warning: This material is still (heavily) under construction and might change throughout the course!

You can help improving the course and let me know about any errors and inconsistencies that you find or suggest other ways of improving the course.

Welcome!

These pages present the content of the course "Introduction to Musical Corpus Studies" at the Institute of Musicology, given at University of Cologne in Fall 2020.

In the last two decades *Musical Corpus Studies* evolved from a niche discipline into a veritable research area. The growing availability of digital and digitized musical data as well as the application and development of modern methodologies from computer science, machine learning, and data science cast new light on old musicological questions and generate entirely novel approaches to empirical music research.

Moreover, the general methodological and epistemological approach of Musical Corpus Studies allows to transcend traditional intra-musicological boundaries between its sub-disciplintes (historical/systematic/ethnological/. . .) without sacrificing the respective specific viewpoints and perspectives.

This course offers a fundamental and practical introduction into these topics. It demonstrates, explores, and critically reflects central thematic areas and methods by means of a number of case studies. In the engagement with these topics the course also introduces elementary methods from natural language and music processing, as well as statistics, data analysis and visualization.

The course is aimed at students at the undergraduate level who have little or no empirical background and are curious about quantitative approaches to musicology.

CHAPTER

ONE

ORGANIZATION

1.1 About this course

This course aims at providing an example-based introduction to the rapidly developing field of Musical Corpus Studies (MCS). Introducing a field that relies equally on musicological domain knowledge and skills in computational and statistical methods faces obvious challenges: while most people interested in this field come with a background in either area, few people are versed in both, and it can take years to bridge the musicological-computational gap.

In particular, systematic introductions to programming or specific musicological topics can be at times quite arduous, even boring, because it takes a long time to proceed from learning basic concepts to acually interesting problems. The problems and "toy examples" that are presented to introduce the basic concepts are necessarily remote from real-world applications and challenging research problems.

This course takes an alternative route. It does not start with an introduction to the programming language Python (which will be used throughout to carry out the computational analyses) but rather showcases a number of recent corpus studies that take on musicological research questions. The focus thus lies in understanding how aspects of music can be studied with computational methods and by analyzing musical corpora.

If this sparks your interest, it will be much easier to pick up the basics for yourself, knowing what they are *for* and being motivated intrinsically. If you are not particularly interested in doing this kind of work yourself, you will still see a broad range of applications that are much more useful to you than learning (or not learning) programming basics.

1.2 Overview

This year's course takes place on two weekends (13-14 November and 11-12 December 2020), comprising twelve sessions in total. The topics cover a broad range of musicological topics, from folk melodies and Jazz solos, over harmonies in Beethoven's string quartets and 20th century Pop music, to Renaissance candences and metric patterns in Malian drum music (see Table 1.2).

No.	Date	Time	Topics
1	Fr., 13.11.2020	16:00-17:20 Uhr	Introduction / Background
2		17:40-19:00 Uhr	Melody I: Pitches, intervals, folk song melodies
3	Sa., 14.11.2020	09:00-10:20 Uhr	Melody II: Jazz solos
4		10:40-12:00 Uhr	Harmony I: Beethoven's string quartets
		12:00-13:00 Uhr	*Lunch Break*
5		13:00-14:20 Uhr	Group work
6		14:40-16:00 Uhr	Harmony II: Pop charts (Billboard 100)
7	Fr., 11.12.2020	10:00-11:20 Uhr	Harmony III: Cadences in Renaissance polyphony (guest: Richard Freedman)
8		11:40-13:00 Uhr	Harmony IV & Form: Brazilian Choro
9	Sa., 12.12.2020	09:00-10:20 Uhr	Rhythm & Meter: Malian percussion music
10		10:40-12:00 Uhr	Timbre: Electronic Music 1950-1990
		12:00-13:00 Uhr	*Lunch Break*
11		13:00-14:20 Uhr	Group work
12		14:40-16:00 Uhr	Recapitulation and conclusion

1.3 Credits

Active participation in this course is compensated with 3 credit points (CPs), equivalent to a work load of 90 hours. These are distributed as follows: 24 SWS (à 45 minutes) are allocated to presence in the block seminar. Additionally, 24 SWS are dedicated to the preparation and follow-up of the material. The remainder of 42 SWS goes to the reading of the relevant literature.

1.4 Deliverables and learning objectives

Apart from attending and following the presentations by the lecturer, course work consists of three main parts: preparing the relevant literature (reading), completing the assigned exercises (group work), and critically engaging with the course materials in the form of a report written together with your group (report).

These deliverables will broaden your knowledge and understanding of current musicological research, enhance your organizational and social skills, and help you to develop efficient work-load management strategies. Finally, compiling a report will advance your communication and writing abilities.

Reading

For each session, the relevant literature is cited in the text and provided on ILIAS. Careful preparation of the reading material is required in order to be able to follow the content of the course. Because the course will mainly talk about methods and general points of musical corpus research, the content (and musical topic) will mainly be introduced by the literature.

I am aware that the reading workload is relatively high since the course will be taught as a block seminar and doesn't spread out over the entire semester. I hope that the fact that the course is finished before the end of the year compensates for this.

Group work

At the beginning of the course, you will be randomly assigned to a group. Together with your group (which will stay fixed for the entire semester), you will work on a number of exercises during the course, e.g. in Zoom breakout rooms. You will collaborate on specific tasks related to the topic at hand, discuss methodological questions, and help each other in the understanding of some of the concepts that are introduced in the course.

Report

After the course has ended, your group will be randomly assigned a course topic (one of the twelve sessions in Table 1.2). It is your task to write a report on this theme. Questions that you could address are: What did you learn? Which concepts are not clear? Which methods did you (not) understand? What is missing? How can the textual descriptions be improved? Who in your group did what? Was the presentation of the material adequate? If not, what was missing? Please also write about the organization of your group, challenges and benefits.

Recommended structure for the report

1. **Introduction:** general description and summary of the course and your assigned session in particular.
2. **Discussion:** summarize the main discussion, open questions, and how you would continue this line or research.
3. **Issues:** describe in detail what was crucial for your understanding of the topic, what was missing, etc.
4. **Various:** anything that you would like to write in the report
5. **Author contributions:** describe briefly how each of you specifically contributed to the report.

Important: Submit your report by **31 January 2021, 23:59h** to fabian.moss@epfl.ch as a single PDF file per group, named *intro_corpusmus_<group_number>.pdf*, e.g. *intro_corpusmus_1.pdf*.

CHAPTER TWO

INTRODUCTION AND BACKGROUND

2.1 Introduction

- Me
- You
- Test breakout (exchange contact info)

2.1.1 About this course

2.1.2 Questions?

2.2 What are Musical Corpus Studies?

2.3 Some important issues

2.4 Examples and preview

2.5 Questions?

[ ]:
```

```

CHAPTER

THREE

MELODIES IN FOLK SONGS

On Jupyter Hub, change the kernel to Python 3.7!

[10]:
```
import pandas as pd
import music21 as m21
import numpy as np
import statsmodels.api as sm

import matplotlib.pyplot as plt
import matplotlib as mpl

import seaborn as sns
sns.set_context("notebook")
```

[11]:
```
## Tragen Sie hier bitte Ihren username ein:
# USERNAME = "fmoss"

## for jupyter hubs
# %env QT_QPA_PLATFORM=offscreen
# # new user, create music21 environment variables.
# m21.environment.set('musicxmlPath', value='/usr/bin/mscore')
# m21.environment.set('musescoreDirectPNGPath', value='/usr/bin/mscore')
# m21.environment.set('graphicsPath', value=f'/home/{USERNAME}') # change accordingly␣
↪for your own username!
```

3.1 The *Essen Folksong Collection*

In this session, we work with a corpus of melodies, the *Essen Folksong Collection* (EFC). There are several ways to access this corpus, for example through the interface provided by the Center for Computer Assisted Research in the Humanities (CCARH) at Stanford University: http://essen.themefinder.org/ or via http://kern.ccarh.org/browse?l=essen.

A more convenient way to work with the pieces is by using the Python library `music21`. This library was developed and is maintaned my Mike Cuthbert at the MIT and is the most popular library for the computational analysis of symbolic music (i.e. scores). You can find its documentation here: http://web.mit.edu/music21/

However, using `music21` requires some training and getting used to its particular API (the way how to interact with its functions). We will not get into too many details here but rather showcase how it can be used for our purposes.

The first thing we do is to load the entire EFC and store it in a variable named `corpora`.

[12]:
```
# load corpus
corpora = m21.corpus.getComposer('essenFolksong')
```

Calling the variable `corpora` shows that it consists of a list of file paths. Using the `len()` function, we can find out how many corpora are stored in the variable `corpora`.

```
[13]: len(corpora)
```

```
[13]: 31
```

We can also directly call the variable `corpora` to see what it contains:

```
[14]: corpora
```

```
[14]: [WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/altdeu10.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/altdeu20.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad10.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad20.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad30.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad40.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad50.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad60.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad70.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/ballad80.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/boehme10.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/boehme20.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/dva0.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/erk10.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/erk20.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/erk30.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/erk5.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/fink0.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/folkHaydn.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/han1.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/han2.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/irl.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/kinder0.abc'),
       WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/lot.abc'),
```

(continues on next page)

(continued from previous page)

```
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/lux.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/test0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/test1.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/testd.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/teste.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/variant0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/zuccal0.abc')]
```

The variable `corpora` is a list of file paths, each of which points to a corpus in this collection. Note that the location depends on the location where `music21` is installed. If you would do this on your own computer, you would see different paths. The file names at the end of the file paths indicate what they contain, e.g. `altdeu10.abc` contains old German folksongs, `boehme10.abc` contains Czech folksongs, and `han1.abc` contains Chinese folksongs.

The `.abc` file ending refers to the ABC notation for encoding melodies. You find more information about the ABC encoding here: http://abcnotation.com/

For example, a song could be encoded like this:

[15]:
```
example_song = """
X:1
T:Speed the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA|
  GABc dedB|dedB dedB|c2ec B2dB|A2F2 G4:|
|:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df|
  g2gf g2Bd|g2f2 e2d2|c2ec B2dB|A2F2 G4:|
"""
```

The tripple quotes (`"""`) surrounding the ABC notation are used by Python to store multi-line text.

What can we already understand from this encoding?

`music21` can load this string and display a graphical output of the score. This is done by a **parser**. A parser is a program that reads a file and produces a structured output.

[16]:
```
parsed_example_song = m21.converter.parse(example_song)
```

We did not need to give it the entire string again because we have already saved it in the `example_song` variable. The purpose of variables is that you can refer to them later in your code without explicitly needing to state its value.

Calling the variable `parsed_example_song` now, however, does not really help us here…

[17]:
```
parsed_example_song
```

[17]:
```
<music21.stream.Score 0x1e4e4ea5a00>
```

It returns a somewhat cryptic statement that says that the variable countains a `music21.stream.Score` object. Understanding the internal organization of `music21` goes beyond this class. For us, it is suffient to know that these objects have certain associated functions, called **methods**, that we can use on them. To look at the score of this example song, we use the method `.show()`.

[18]:
```
parsed_example_song.show()
```

Voilà, this is much better! Now, let us compare the score output to the ABC encoding of the song:

[19]:
```
print(example_song)
```

```
X:1
T:Speed the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA|
  GABc dedB|dedB dedB|c2ec B2dB|A2F2 G4:|
|:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df|
  g2gf g2Bd|g2f2 e2d2|c2ec B2dB|A2F2 G4:|
```

Now the ABC notation makes already more sense. `T:Speed the Ploug` stands for the title, `M:4/4` for the meter, and `K:G` for the key of the song. The ABC documentation tells us that `X:1` encodes just a reference number, in case multiple pieces are stored in the same file (as in our case in the variable `corpora`, remember?). And the lines at the bottom encode the proper melody, where the letters represent note names that are organized into bars with or without repetition signs.

`music21` even gives us the option to listen to the song if we path the `midi` argument to the `.show()` method:

[20]:
```
parsed_example_song.show("midi")
```

```
<IPython.core.display.HTML object>
```

Now, what happens if we try to parse one of the corpora in the EFC? We can select a specific corpus by its **index** in the list. Python starts counting at 0, so the first file in the list corresponds to

[21]:
```
corpora[0]
```

[21]:
```
WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/altdeu10.abc')
```

As you can see, this is just the first file path in the variable `corpora`. Let's try to parse it!

[22]:
```
first_corpus = m21.converter.parse(corpora[0])
```

Looking at the new variable `first_corpus` shows a difference to the example song before; we don't have a `music21.stream.Score` object but a `music21.stream.Opus` object.

[23]:
```
first_corpus
```

[23]:
```
<music21.stream.Opus 0x1e4e52bfdf0>
```

If we would call the `.show()` method on `first_corpus`, we would see the scores of all pieces that are in this particular corpus. But we don't know how many these are. It there are only three songs, it would not be a problem, but if there were thousands of songs, it could take a very long time to parse and display them all. Fortunately, all pieces in the collection have the `X:n` line that we saw above, so that we can directly reference them. With which number would we have to replace `n` if we wanted to look at the 7tst piece? Remember that Python starts counting at 0.

[24]:
```
first_corpus[70].show()
```

[25]:
```
first_corpus[70].show("midi")
```

```
<IPython.core.display.HTML object>
```

We have seen that we can select items from lists by **indexing** them, `list[i]`. We can get ranges of lists by using the `:` character. For example, `list[:10]` shows the first ten elements, `list[10:]` shows everything after the ninth element, and `list[3:6]` shows elements 3, 4, and 5 (not 6!) of the list.

3.2 Comparing songs

Looking at individual songs is interesting for music analysis but for that the computational approach is not really necessary. We could as easily do the same by just looking at a book of scores. The power of computational methods becomes clearer when we start comparing different songs, potentially in a large number.

To facilitate this comparison, we will first load all songs in all corpora of the EFC into a single list, called `songs` (this might take a couple of minutes).

[26]:
```
songs = [s for i in range(len(corpora)) for s in m21.converter.parse(corpora[i]) ]
```

This looks a bit complicated but all it does is to go through all corpora and extract all songs into a new list. The way we did it is called **list comprehension** in Python. It is not important if you don't understand this now but feel free to look it up!

Using the `len()` function again, we see how many songs we have in total.

[27]:
```
len(songs)
```

[27]:
```
8514
```

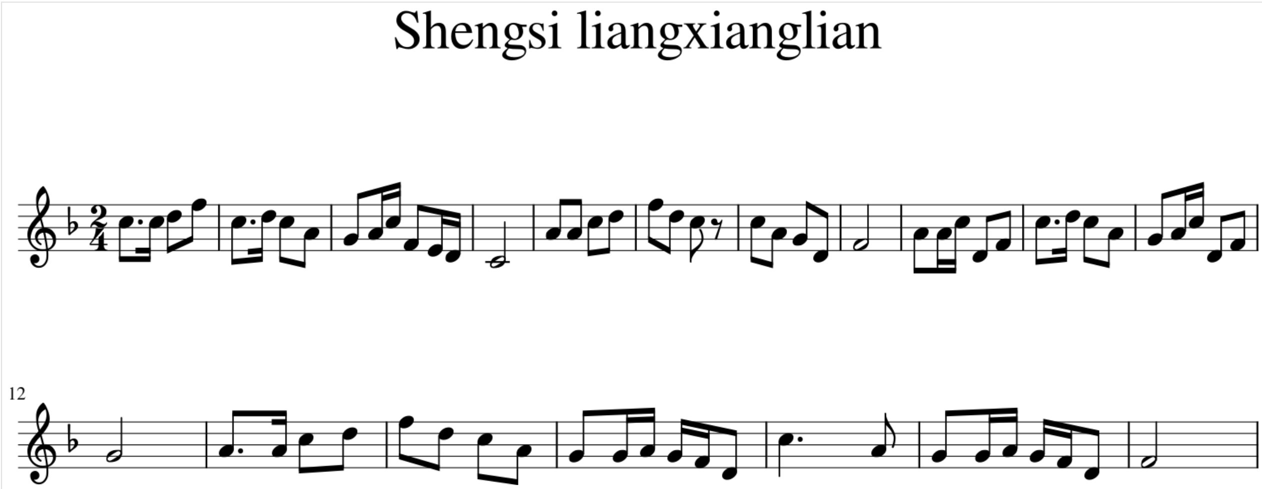
We can now use the list `songs` to compare two different songs. Again, we load the 71st song of the first corpus and store it now in a variable `german_song`, and we load chinese song with index 6200 into the variable `chinese_song`.

[28]:
```
german_song = songs[70]
chinese_song = songs[6200]
```

It is easy to display these songs now:

[31]:
```
german_song.show()
```

[32]:
```
chinese_song.show()
```




```
[33]: chinese_song.show("midi")
```

```
<IPython.core.display.HTML object>
```

Analysis of songs...

3.3 Computational analysis

We now go on to a computational analysis of these two and all the other songs. Specifically, we wil compare their **melodic profiles**. To make things a bit simpler, we will just look at the notes.

A note can be easily represented as a pair of **pitch** (its height) and its **duration**. For example, the first note of the *Die plappernden Junggesellen* could be represented as `(D4, 1/4)`; it is a quarter note on the pitch D4 (the 4 indicates the octave in which the note is).

Another way to represent the pitch of notes is using **MIDI numbers**. MIDI stands for *Musical Instrument Digital Interface* and was developed for the communication between different electronic instruments such as keyboards. In MIDI, each note is simply associated with a number:

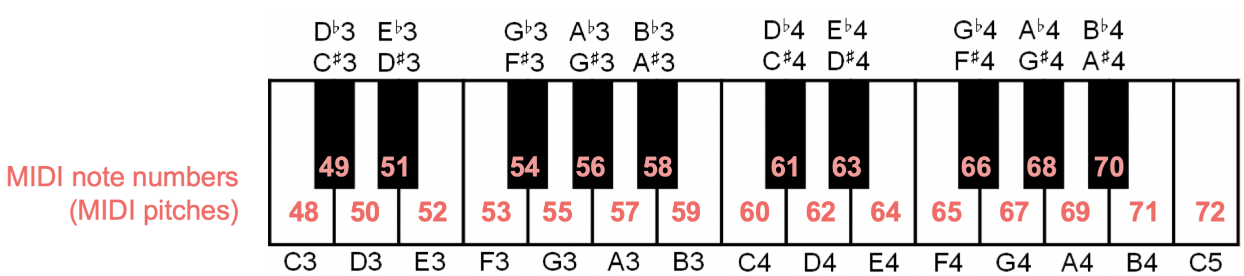


Image from https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2_MIDI.html.

We can see that D4 is associated with the number 62. The second note, the G4, is associated with 62+5=67 because G is five semitones above D.

To make it easier to work with pieces in this way, we define a **function** that gives us a list of notes for each piece.

```
[34]: def notelist(piece):
          """
          This function takes a song as input and returns a list of (pitch, duration) pairs,
          where the duration is given in quarter notes.
```

(continues on next page)

(continued from previous page)

```
    """

    df = pd.DataFrame([ (note.pitch.midi, note.quarterLength) for note in piece.flat.
↪notes ], columns=["MIDI Pitch", "Duration"])
    df["Onset"] = df["Duration"].cumsum()

    return df
```

Note that the duration of a note is given in quarter notes, i.e. a quarter note has a duration of 1, a half note has a duration of 2, and an eighth note has a duration of 0.5.

Let's display the first phrase (the first eight notes) of the German song:

[35]:
```
notelist(german_song)[:8]
```

[35]:

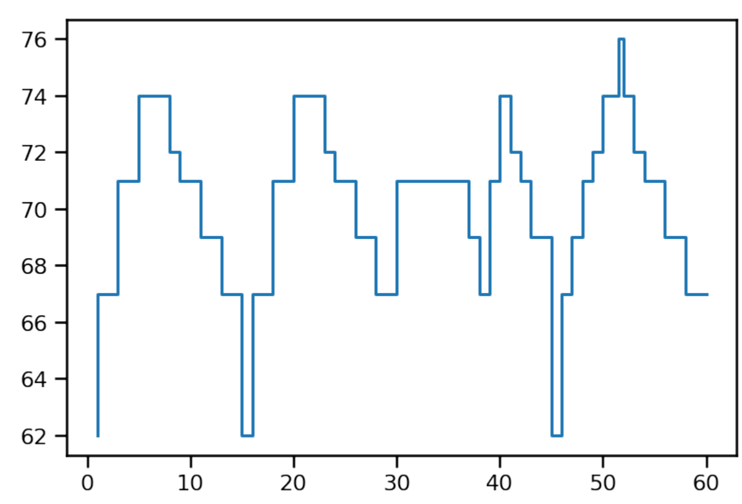
	MIDI Pitch	Duration	Onset
0	62	1.0	1.0
1	67	2.0	3.0
2	71	2.0	5.0
3	74	3.0	8.0
4	72	1.0	9.0
5	71	2.0	11.0
6	69	2.0	13.0
7	67	2.0	15.0

Note that we added another column, "Onset". What does it represent?

This allows us now to look at the **melodic profile** of a particular song.

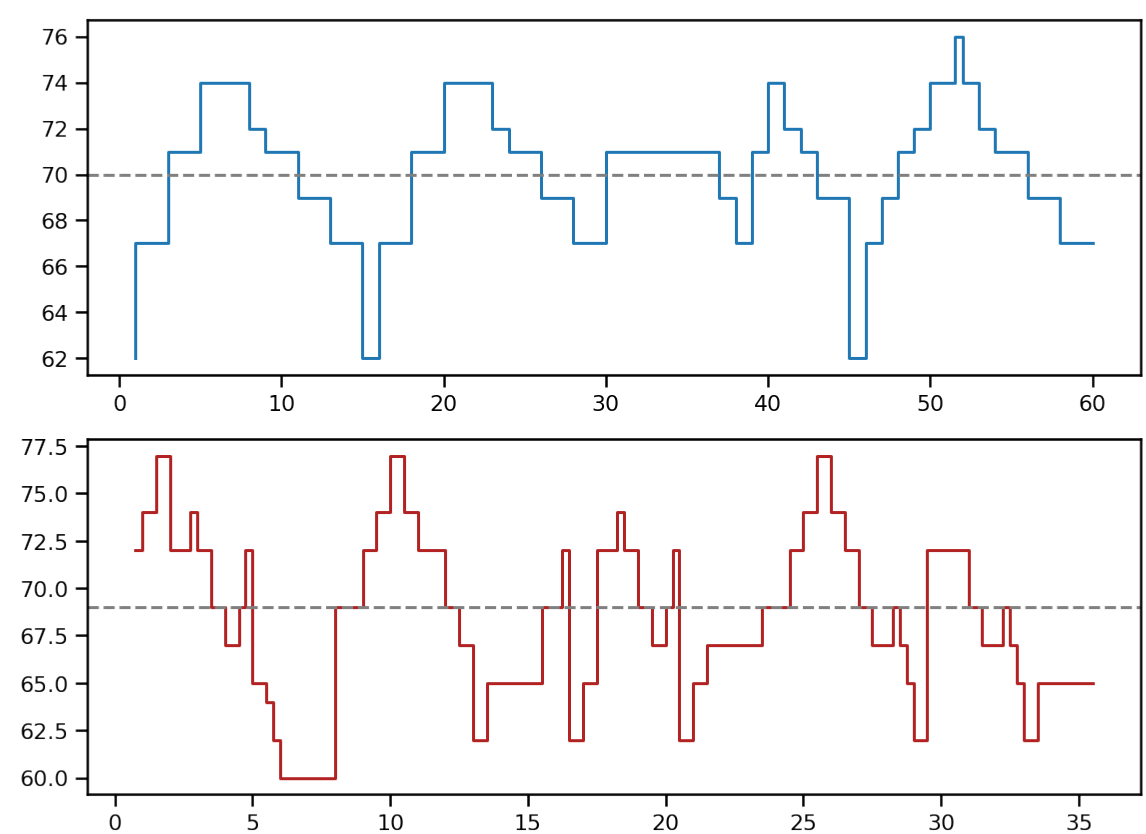
[36]:
```
def plot_melodic_profile(notelist, ax=None, c=None, mean=False, Z=False,␣
↪sections=False, standardized=False):

    if ax == None:
        ax = plt.gca()

    if standardized:
        x = notelist["Rel. Onset"]
        y = notelist["Rel. MIDI Pitch"]
    else:
        x = notelist["Onset"]
        y = notelist["MIDI Pitch"]

    ax.step(x,y, color=c)

    if mean:
        ax.axhline(y.mean(), color="gray", linestyle="--")

    if sections:
        for l in [ x.max() * i for i in [ 1/4, 1/2, 3/4] ]:
            ax.axvline(l, color="gray", linewidth=1, linestyle="--")
```

[37]:
```
plot_melodic_profile(notelist(german_song))
```



Likewise, we can as easily plot the melodic contour of the Chinese song (we will use a different color).

[38]:
```
fig, axes = plt.subplots(2,1, figsize=(8,6))

plot_melodic_profile(notelist(german_song), ax=axes[0], mean=True)
plot_melodic_profile(notelist(chinese_song), ax=axes[1], c="firebrick", mean=True)

plt.tight_layout()
plt.savefig("img/melodic_profiles.png")
```



The dashed grey lines in both plots show the average MIDI pitch of the song.

But still, it is quite difficult to compare them directly. They differ both with respect to their length (see the numbers on the "Onset" axis) and their pitches (see "MIDI Pitch" axis).

We need to transform them in a way that makes them directly comparable. To that end, we define a new function `standardize()`.

[39]:
```
def standardize(notelist):
    """
    Takes a notelist as input and returns a standardized version.
    """

    notelist["Rel. MIDI Pitch"] = (notelist["MIDI Pitch"] - notelist["MIDI Pitch"].
↪mean()) / notelist["MIDI Pitch"].std()
    notelist["Rel. Duration"] = notelist["Duration"] / notelist["Duration"].sum()
    notelist["Rel. Onset"] = notelist["Onset"] / notelist["Onset"].max()

    return notelist
```

[46]:
```
standardize(notelist(german_song))
```

[46]:

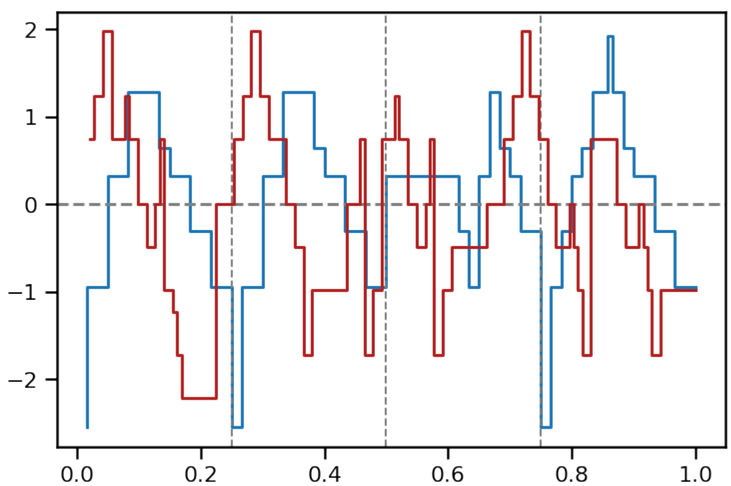
	MIDI Pitch	Duration	Onset	Rel. MIDI Pitch	Rel. Duration	Rel. Onset
0	62	1.0	1.0	-2.543827	0.016667	0.016667
1	67	2.0	3.0	-0.949300	0.033333	0.050000
2	71	2.0	5.0	0.326322	0.033333	0.083333
3	74	3.0	8.0	1.283038	0.050000	0.133333
4	72	1.0	9.0	0.645227	0.016667	0.150000
5	71	2.0	11.0	0.326322	0.033333	0.183333
6	69	2.0	13.0	-0.311489	0.033333	0.216667
7	67	2.0	15.0	-0.949300	0.033333	0.250000
8	62	1.0	16.0	-2.543827	0.016667	0.266667
9	67	2.0	18.0	-0.949300	0.033333	0.300000
10	71	2.0	20.0	0.326322	0.033333	0.333333
11	74	3.0	23.0	1.283038	0.050000	0.383333
12	72	1.0	24.0	0.645227	0.016667	0.400000
13	71	2.0	26.0	0.326322	0.033333	0.433333
14	69	2.0	28.0	-0.311489	0.033333	0.466667
15	67	2.0	30.0	-0.949300	0.033333	0.500000
16	71	1.0	31.0	0.326322	0.016667	0.516667
17	71	1.0	32.0	0.326322	0.016667	0.533333
18	71	1.0	33.0	0.326322	0.016667	0.550000
19	71	1.0	34.0	0.326322	0.016667	0.566667
20	71	1.0	35.0	0.326322	0.016667	0.583333
21	71	2.0	37.0	0.326322	0.033333	0.616667
22	69	1.0	38.0	-0.311489	0.016667	0.633333
23	67	1.0	39.0	-0.949300	0.016667	0.650000
24	71	1.0	40.0	0.326322	0.016667	0.666667
25	74	1.0	41.0	1.283038	0.016667	0.683333
26	72	1.0	42.0	0.645227	0.016667	0.700000
27	71	1.0	43.0	0.326322	0.016667	0.716667
28	69	2.0	45.0	-0.311489	0.033333	0.750000
29	62	1.0	46.0	-2.543827	0.016667	0.766667
30	67	1.0	47.0	-0.949300	0.016667	0.783333
31	69	1.0	48.0	-0.311489	0.016667	0.800000
32	71	1.0	49.0	0.326322	0.016667	0.816667
33	72	1.0	50.0	0.645227	0.016667	0.833333
34	74	1.5	51.5	1.283038	0.025000	0.858333
35	76	0.5	52.0	1.920849	0.008333	0.866667
36	74	1.0	53.0	1.283038	0.016667	0.883333

(continues on next page)

(continued from previous page)

```
37          72       1.0    54.0          0.645227       0.016667     0.900000
38          71       1.0    55.0          0.326322       0.016667     0.916667
39          71       1.0    56.0          0.326322       0.016667     0.933333
40          69       1.0    57.0         -0.311489       0.016667     0.950000
41          69       1.0    58.0         -0.311489       0.016667     0.966667
42          67       2.0    60.0         -0.949300       0.033333     1.000000
```

[47]:
```
plot_melodic_profile(standardize(notelist(german_song)), mean=True, sections=True,␣
↪standardized=True)
plot_melodic_profile(standardize(notelist(chinese_song)), c="firebrick",␣
↪standardized=True)
```



Standardizing the songs makes it possible to compare them directly: They have now the same length 1 and their pitches are centered around the mean 0 with a standard deviation of 1.

However, already with two pieces this plot is quite crowded.

[48]:
```
dfs = []

for i, song in enumerate(songs):
    df = standardize(notelist(song))
    df["Song ID"] = i
    dfs.append(df)

big_df = pd.concat(dfs).reset_index(drop=True)
```

[49]:
```
big_df
```

[49]:
```
        MIDI Pitch  Duration  Onset  Rel. MIDI Pitch  Rel. Duration  \
0               67      2.00   2.00        -1.819039       0.013158
1               70      2.00   4.00        -0.741977       0.013158
2               71      2.00   6.00        -0.382956       0.013158
3               72      2.00   8.00        -0.023935       0.013158
4               72      2.00  10.00        -0.023935       0.013158
...            ...       ...    ...              ...            ...
450591          71      0.25  28.50         0.691456       0.008197
450592          69      0.25  28.75         0.098779       0.008197
450593          73      0.25  29.00         1.284133       0.008197
450594          71      1.00  30.00         0.691456       0.032787
```

(continues on next page)

(continued from previous page)

```
450595          69       0.50   30.50          0.098779       0.016393

        Rel. Onset  Song ID
0         0.013158        0
1         0.026316        0
2         0.039474        0
3         0.052632        0
4         0.065789        0
...            ...      ...
450591    0.934426     8513
450592    0.942623     8513
450593    0.950820     8513
450594    0.983607     8513
450595    1.000000     8513

[450596 rows x 7 columns]
```

[50]:
```
big_df.to_csv("data/big_df.csv")
```

[51]:
```
big_df.sample(10)
```

[51]:
```
        MIDI Pitch  Duration   Onset  Rel. MIDI Pitch  Rel. Duration  \
109749          69      1.00   16.00        -0.109682       0.021739
434396          64      0.50   18.50        -0.632118       0.016949
117809          71      0.50   19.50         0.902822       0.020833
218843          72      0.50   29.00         0.216612       0.012195
298799          74      0.50  174.00         0.787618       0.002725
440611          76      0.25   29.00         1.004299       0.003906
360749          70      0.50    5.00         0.326851       0.016129
71559           62      0.50   10.00        -1.786648       0.007812
204570          74      0.50   28.50         1.214628       0.011111
446299          79      0.25   46.75         1.409059       0.004762

        Rel. Onset  Song ID
109749    0.347826     2395
434396    0.627119     8218
117809    0.812500     2535
218843    0.707317     4656
298799    0.948229     5745
440611    0.453125     8337
360749    0.161290     6788
71559     0.156250     1703
204570    0.633333     4357
446299    0.890476     8439
```

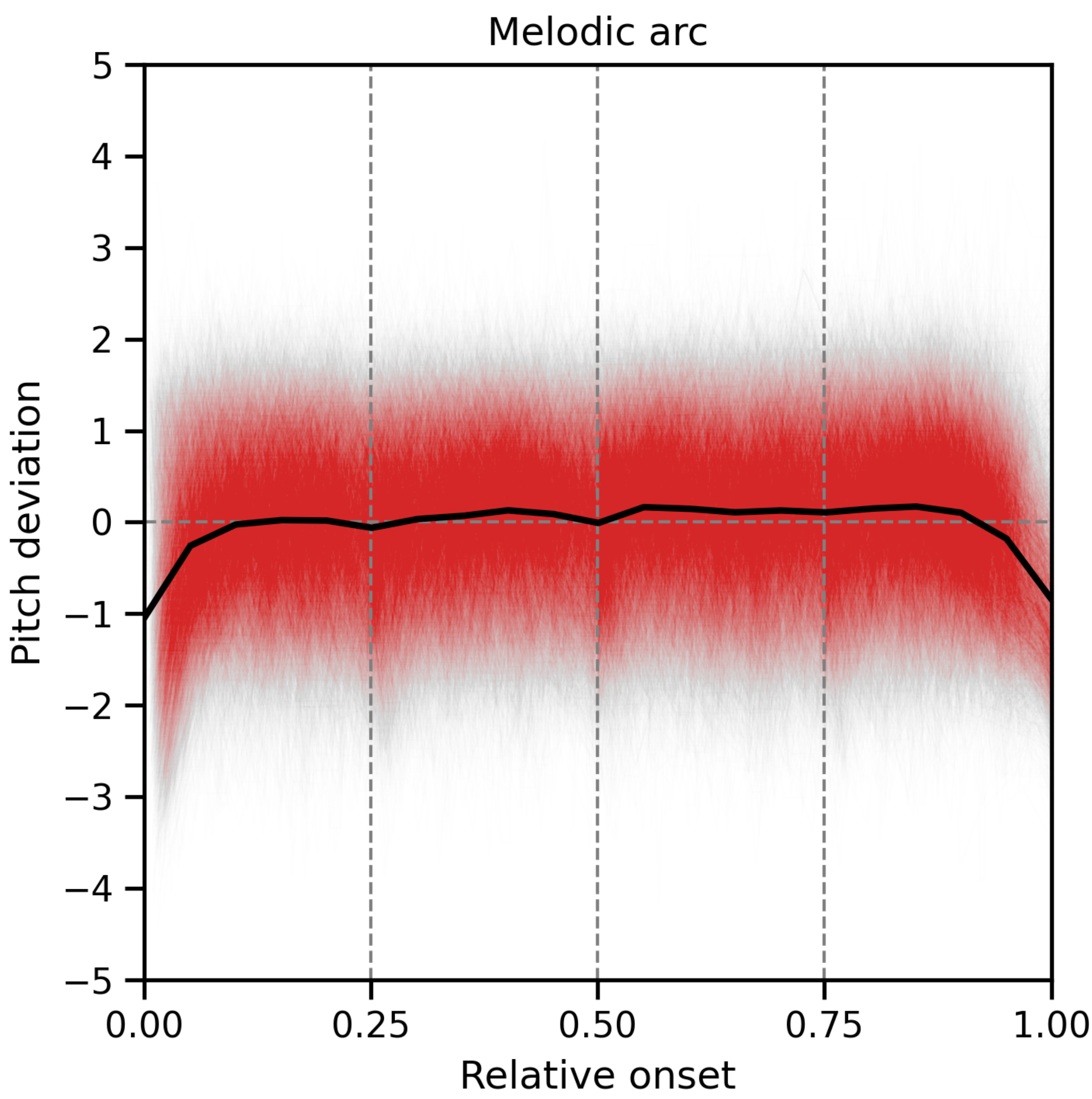
3.4 The melodic arc

[296]:
```
%%time

fig, ax = plt.subplots(figsize=(5,5))

grouped = big_df.groupby("Song ID")
```

(continues on next page)

(continued from previous page)

```
for i, g in grouped:
    x = g["Rel. Onset"]
    y = g["Rel. MIDI Pitch"]
    ax.plot(x,y, lw=.5, c="tab:red", alpha=1/300)

ax.axvline(.25, lw=1, ls="--", c="gray")
ax.axvline(.5, lw=1, ls="--", c="gray")
ax.axvline(.75, lw=1, ls="--", c="gray")
ax.axhline(0, lw=1, ls="--", c="gray")

lowess = sm.nonparametric.lowess
big_x = big_df["Rel. Onset"]
big_y = big_df["Rel. MIDI Pitch"]
big_z = lowess(big_y, big_x, frac=5/100, delta=1/20)
ax.plot(big_z[:,0], big_z[:,1], c="black", lw=2)

plt.title("Melodic arc")
plt.xlabel("Relative onset")
plt.ylabel("Pitch deviation")
plt.xticks(np.linspace(0,1,5))
plt.yticks(np.linspace(-5,5,11))
plt.xlim(0,1)

plt.tight_layout()
plt.savefig("img/melodic_arc.png")
plt.show()
```




```
Wall time: 31 s
```

3.5 Intervals

We have seen that the melodic arc emerges as a stable shape over the entire EFC, and that sub-phrases of the songs likewise have an arc-like shape. In the remainder of this section, we look at **intervals**, the distance between two notes.

Let's come back to the song *Die plappernden Junggesellen*

[29]:
```
german_song.show()
```

We have already extracted its notes and stored them in a DataFrame:

```
[55]: big_df[big_df["Song ID"] == 70].head(8)
```

```
[55]:       MIDI Pitch  Duration  Onset  Rel. MIDI Pitch  Rel. Duration  Rel. Onset  \
2969          62       1.0    1.0        -2.543827       0.016667    0.016667
2970          67       2.0    3.0        -0.949300       0.033333    0.050000
2971          71       2.0    5.0         0.326322       0.033333    0.083333
2972          74       3.0    8.0         1.283038       0.050000    0.133333
2973          72       1.0    9.0         0.645227       0.016667    0.150000
2974          71       2.0   11.0         0.326322       0.033333    0.183333
2975          69       2.0   13.0        -0.311489       0.033333    0.216667
2976          67       2.0   15.0        -0.949300       0.033333    0.250000

      Song ID
2969       70
2970       70
2971       70
2972       70
2973       70
2974       70
2975       70
2976       70
```

The code above reads as "Select all rows in `big_df` for which the column `Song ID` is equal to 70". The `.head()` method displays the first 5 rows by default but you can specify the number of rows you want to be displayed (here 8).

Focusing on the "MIDI Pitch" column, the notes in the first phrase have MIDI pitch 62, 67, 71, 74, 72. Since intervals correspond to the difference between notes, the intervals for the beginning of this song are:

- +5 (67-62)
- +4 (71-67)
- +3 (74-71)
- -2 (72-74)
- -1 (71-72)
- -2 (69-71)
- -2 (67-69)

The sequence of intervals in this phrase is thus `[+5, +4, +3, -2, -1, -2, -2]`. The signs (+ or -) also reflect the arc-like shape of this first phrase, but the sizes of the intervals are not perfecly balanced. Note that `-2` (two descending semitones, or one descending whole tone) is the most frequent interval.

[110]:
```
all_ints = [ p2 - p1 for i, g in big_df.groupby("Song ID") for p1, p2 in zip(g["MIDI␣
↪Pitch"], g["MIDI Pitch"][1:]) ]
min_int = min(all_ints)
max_int = max(all_ints)
```

[111]:
```
min_int, max_int
```

[111]:
```
(-25, 25)
```

[ ]:
```
len(all_ints)
```

[125]:
```
ints_df = pd.DataFrame(0, index=np.arange(min_int,max_int), columns=np.arange(min_int,
↪max_int+1))

for i, g in big_df.groupby("Song ID"):
    intervals = [ p2 - p1 for p1, p2 in zip(g["MIDI Pitch"], g["MIDI Pitch"][1:])]

    for i1, i2 in zip(intervals, intervals[1:]):
        ints_df.loc[i1,i2] += 1
```

[126]:
```
ints_df
```

[126]:
```
     -25  -24  -23  -22  -21  -20  -19  -18  -17  -16  ...   16   17   18  \
-25    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-24    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-23    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-22    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-21    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-20    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-19    0    0    0    0    0    0    0    0    0    0  ...    3    0    0
-18    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-17    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-16    0    0    0    0    0    0    0    0    0    0  ...    4    0    0
-15    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-14    0    0    0    0    0    0    0    0    0    0  ...    0    1    0
-13    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-12    0    0    0    0    0    0    0    0    0    0  ...   13    2    0
-11    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-10    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-9     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-8     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-7     0    0    0    0    0    0    0    0    0    0  ...    5    0    0
-6     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-5     0    0    0    0    0    0    0    0    0    0  ...    0    4    0
-4     0    0    0    0    0    0    0    0    0    0  ...    7    0    0
-3     0    0    0    0    0    0    0    0    0    0  ...    1    4    0
-2     0    0    0    0    0    0    0    0    1    0  ...   11   13    0
-1     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 0     1    0    0    0    0    0   13    0    4    0  ...    4    4    0
 1     0    0    0    0    0    0    0    0    4    0  ...    0    0    0
 2     0    0    0    0    4    0    0    0    0    6  ...    0    2    0
 3     0    0    0    0    0    0    4    0    0    0  ...    0    2    0
 4     0    0    0    0    1    0    0    0    0    2  ...    0    0    0
```

(continues on next page)

(continued from previous page)

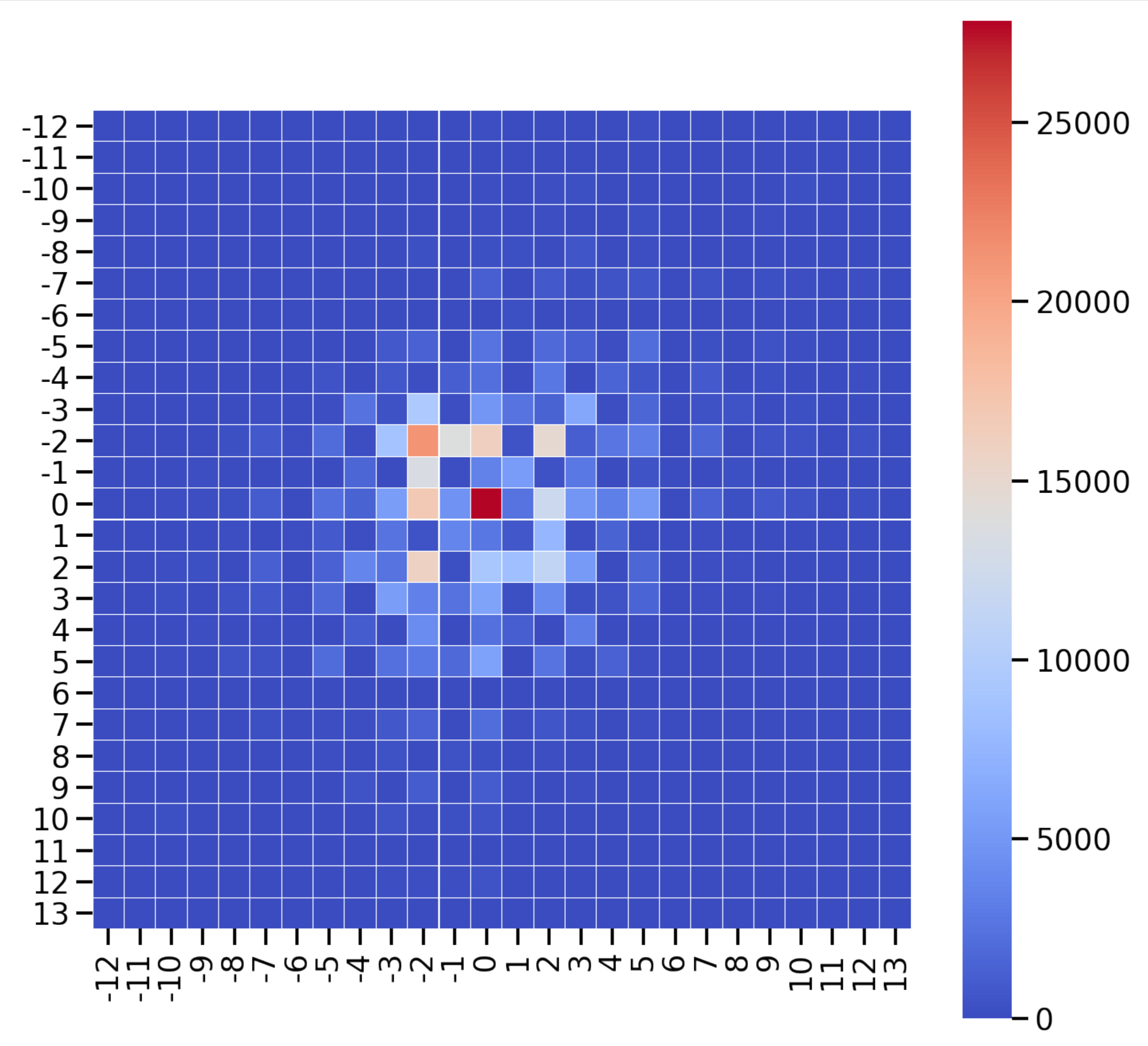
```
 5     0    0    0    0    0    0    0    0    0    1  ...    0    1    0
 6     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 7     0    0    0    0    0    0    0    0    0    1  ...    0    0    0
 8     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 9     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 10    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 11    0    0    0    0    0    0    0    0    0    1  ...    0    0    0
 12    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 13    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 14    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 15    0    0    0    0    0    0    0    0    1    0  ...    0    0    0
 16    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 17    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 18    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 19    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 20    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 21    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 22    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 23    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 24    0    0    0    0    0    0    0    0    0    0  ...    0    0    0

      19   20   21   22   23   24   25
-25    0    0    0    0    0    0    0
-24    0    0    0    0    0    0    0
-23    0    0    0    0    0    0    0
-22    0    0    0    0    0    0    0
-21    0    0    4    0    0    0    0
-20    0    0    0    0    0    0    0
-19   10    0    0    0    0    0    0
-18    0    0    0    0    0    0    0
-17    0    0    0    0    0    0    0
-16    0    0    0    0    0    0    0
-15    0    0    0    0    0    0    0
-14    0    0    0    0    0    0    0
-13    0    0    0    0    0    0    0
-12    1    0    0    0    0    0    0
-11    0    0    0    0    0    0    0
-10    0    0    0    2    0    0    0
-9     0    0    0    0    0    0    0
-8     0    0    0    0    0    0    0
-7     5    4    1    1    0    0    0
-6     0    0    0    0    0    0    0
-5     5    0    0    0    0    0    0
-4     0    0    0    0    0    0    0
-3     1    0    0    0    0    0    1
-2     1    0    1    1    0    1    0
-1     0    0    0    0    0    0    0
 0     0    0    1    0    0    0    0
 1     0    0    0    0    0    0    0
 2     0    0    0    0    0    0    0
 3     0    0    0    0    0    0    0
 4     0    0    0    0    0    0    0
 5     0    0    0    0    0    0    0
 6     0    0    0    0    0    0    0
 7     0    0    0    0    0    0    0
 8     0    0    0    0    0    0    0
 9     0    0    0    0    0    0    0
```

(continues on next page)

(continued from previous page)

```
10    0    0    0    0    0    0    0
11    0    0    0    0    0    0    0
12    0    0    0    0    0    0    0
13    0    0    0    0    0    0    0
14    0    0    0    0    0    0    0
15    0    0    0    0    0    0    0
16    0    0    0    0    0    0    0
17    0    0    0    0    0    0    0
18    0    0    0    0    0    0    0
19    0    0    0    0    0    0    0
20    0    0    0    0    0    0    0
21    0    0    0    0    0    0    0
22    0    0    0    0    0    0    0
23    0    0    0    0    0    0    0
24    0    0    0    0    0    0    0

[50 rows x 51 columns]
```

[131]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(ints_df.loc[-12:13,-12:13], cmap="coolwarm", square=True, linewidths=0.01,
↪ax=ax)
plt.show()
```



The two most common interval pairs are `(0,0)` and `(-2,-2)`. A much less frequent pair of intervals is `(5,0)`, but this is still much more frequent than, for example, `(9,9)`.

To which melodic fragments do these correspond?

[57]:
```
big_df["Avg. MIDI Pitch"] = 0

for i, group in big_df.groupby("Song ID"):
    grp_mean_pitch = int(group["MIDI Pitch"].mean())
    big_df.loc[big_df["Song ID"] == i, "Avg. MIDI Pitch"] = grp_mean_pitch
```

[58]:
```
big_df["shifted_pitch"] = big_df["MIDI Pitch"] - big_df["Avg. MIDI Pitch"]
```

[59]:
```
big_df.tail()
```

[59]:
```
        MIDI Pitch  Duration  Onset  Rel. MIDI Pitch  Rel. Duration  \
450591          71      0.25  28.50         0.691456       0.008197
```

(continues on next page)

(continued from previous page)

```
450592          69      0.25   28.75         0.098779       0.008197
450593          73      0.25   29.00         1.284133       0.008197
450594          71      1.00   30.00         0.691456       0.032787
450595          69      0.50   30.50         0.098779       0.016393

        Rel. Onset  Song ID  Avg. MIDI Pitch  shifted_pitch
450591    0.934426     8513               68              3
450592    0.942623     8513               68              1
450593    0.950820     8513               68              5
450594    0.983607     8513               68              3
450595    1.000000     8513               68              1
```

[60]:
```
idx = np.arange(big_df["shifted_pitch"].min(), big_df["shifted_pitch"].max() + 1)
idx
```

[60]:
```
array([-16, -15, -14, -13, -12, -11, -10,  -9,  -8,  -7,  -6,  -5,  -4,
        -3,  -2,  -1,   0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
        10,  11,  12,  13,  14,  15,  16,  17])
```

[61]:
```
transitions_df = pd.DataFrame(0, index=idx, columns=idx)
transitions_df
```

[61]:
```
     -16  -15  -14  -13  -12  -11  -10  -9   -8   -7   ...   8    9    10  \
-16    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-15    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-14    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-13    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-12    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-11    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-10    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-9     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-8     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-7     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-6     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-5     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-4     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-3     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-2     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
-1     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 0     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 1     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 2     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 3     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 4     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 5     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 6     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 7     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 8     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 9     0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 10    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 11    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 12    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 13    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 14    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 15    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 16    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
 17    0    0    0    0    0    0    0    0    0    0  ...    0    0    0
```

(continues on next page)

(continued from previous page)

```
      11   12   13   14   15   16   17
-16    0    0    0    0    0    0    0
-15    0    0    0    0    0    0    0
-14    0    0    0    0    0    0    0
-13    0    0    0    0    0    0    0
-12    0    0    0    0    0    0    0
-11    0    0    0    0    0    0    0
-10    0    0    0    0    0    0    0
-9     0    0    0    0    0    0    0
-8     0    0    0    0    0    0    0
-7     0    0    0    0    0    0    0
-6     0    0    0    0    0    0    0
-5     0    0    0    0    0    0    0
-4     0    0    0    0    0    0    0
-3     0    0    0    0    0    0    0
-2     0    0    0    0    0    0    0
-1     0    0    0    0    0    0    0
 0     0    0    0    0    0    0    0
 1     0    0    0    0    0    0    0
 2     0    0    0    0    0    0    0
 3     0    0    0    0    0    0    0
 4     0    0    0    0    0    0    0
 5     0    0    0    0    0    0    0
 6     0    0    0    0    0    0    0
 7     0    0    0    0    0    0    0
 8     0    0    0    0    0    0    0
 9     0    0    0    0    0    0    0
 10    0    0    0    0    0    0    0
 11    0    0    0    0    0    0    0
 12    0    0    0    0    0    0    0
 13    0    0    0    0    0    0    0
 14    0    0    0    0    0    0    0
 15    0    0    0    0    0    0    0
 16    0    0    0    0    0    0    0
 17    0    0    0    0    0    0    0

[34 rows x 34 columns]
```

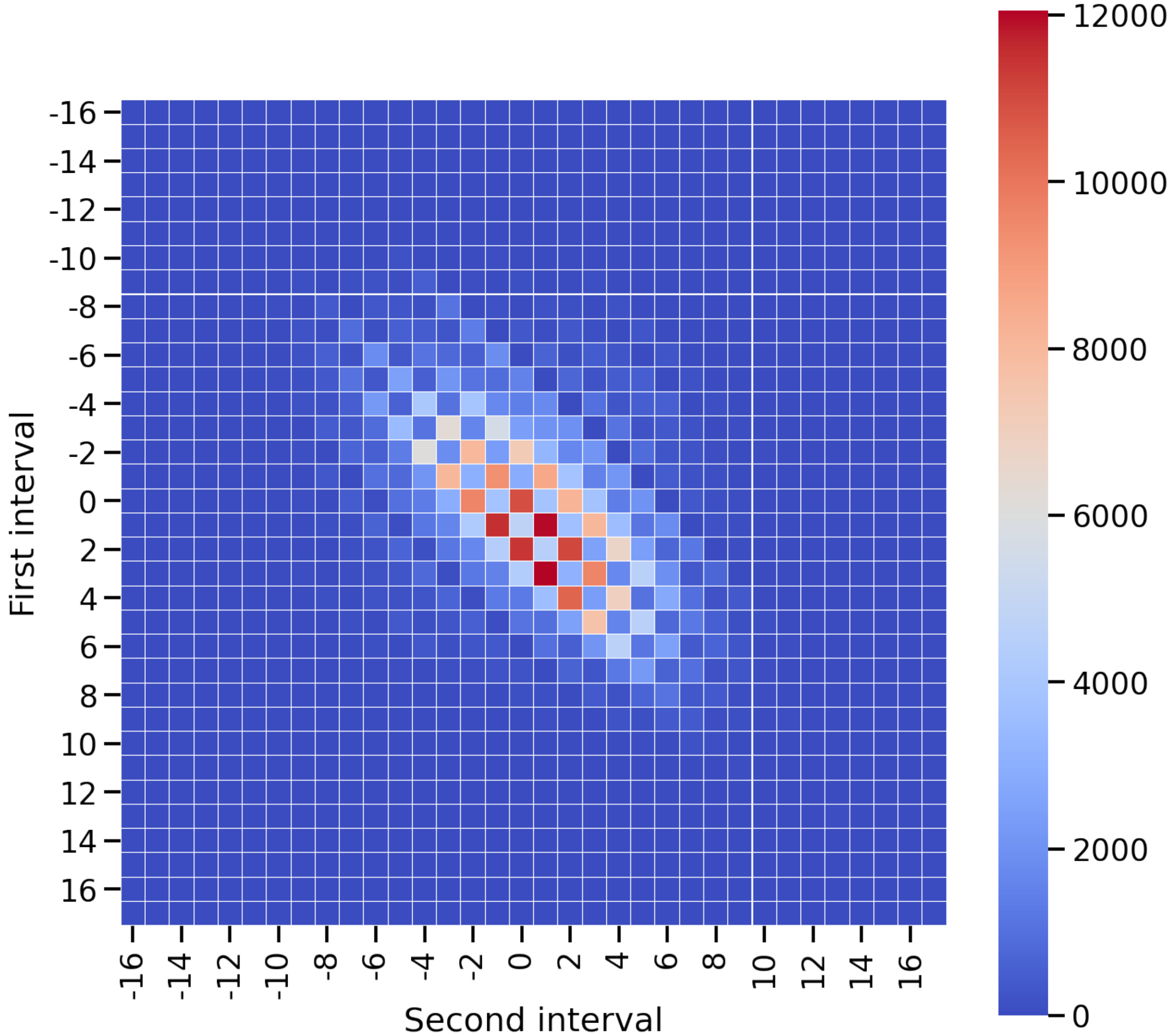
[62]:
```
%%time

for i, group in big_df.groupby("Song ID"):
    for bg in zip(group["shifted_pitch"], group["shifted_pitch"][1:]):
        transitions_df.loc[bg[0],bg[1]] +=1
```

```
Wall time: 1min 7s
```

[63]:
```
print(f"There are {transitions_df.sum().sum()} intervals in total in the corpus.")
```

```
There are 442082 intervals in total in the corpus.
```

[106]:
```
fig, ax = plt.subplots(figsize=(10,10))

g = sns.heatmap(transitions_df, cmap="coolwarm", linewidths=.01, square=True)
plt.ylabel("First interval")
plt.xlabel("Second interval")
plt.show()
```


First interval
Second interval
-16
-14
-12
-10
-8
-6
-4
-2
0
2
4
6
8
10
12
14
16
12000
10000
8000
6000
4000
2000
0

```
[ ]:
```