RP Scripts

Release 2.0

Marcos da Silva Sampaio

May 13, 2023

CONTENTS:

RP Scripts is an open-source command-line program designed to extract rhythmic partitioning information from given digital music scores files, plot partitiogram and indexogram, and create annotated digital music scores files.

This program reads MusicXML (compressed or not), Kern, and MIDI files generating rhythmic partitioning data in JSON files and partitioning charts. It uses the Music21 library for score parsing and Matplotlib for chart plotting.

Some published papers describe RP Scripts' features (See Section *Bibliography*). The software is available at https://github.com/msampaio/rpScripts.

This program is an HMB Project outcome.

CHAPTER

ONE

Installation

RP Scripts is a Python-3-based program. Python 3 non-users have to download and install it before running the following instructions (check https://www.python.org/downloads/).

There are multiple ways to install RP Scripts. All of them require dependencies installation:

```
pip install -r requirements.txt
```

1.1 Binary creation

The easiest way to use it is by creating a binary file (for instance, a `.exe` in Windows).

1. Build the binary:

```
pyinstaller rps_compile.spec
```

This command generates `build` and `dist` folders and saves the binary file into the `dist` folder.

2. Move the binary file to a folder in the user system's `PATH`.
3. We encourage the user to create a personal scripts folder and put it in the system's `PATH` environment variable.
4. Jump to *Finishing installation* instructions.

1.2 As a PIP package

To use rpscripts like a `PIP` package, run:

```
python -m build
```

Then, locate the `.whl` file inside the `dist` folder and run:

```
pip install -U dist/rpscripts-[version]-py3-none-any.whl
```

Linux and Mac users must rename the `rps_bin.py` file to `rpscripts` and move it to a binary folder listed at `PATH`, such as `~/.local/bin` (Linux) or `/usr/local/bin` (Mac).

1.3 Finishing installation

Finally, create a `~/rps_aux` folder and copy the provided `lattice_map.json` there.

CHAPTER

TWO

Usage

RP Scripts has a *Command Line Interface* and a structure for *Package usage*.

2.1 Components

RP Scripts comprises the following programs:

- Calculator
- Plotter
- Annotator
- Labeler
- Info
- Utils
- Stats
- Converter
- Trimmer

2.2 Command Line Interface

`rpscript` provides a high-level command line interface for the rhythmic partitioning tasks. Each program has a specific command line.

The command line below calls the help page.

```
rpscripts -h
```

This command outputs:

```
usage: rpscripts [-h] [-v]
                {calc,plot,annotate,label,info,utils,stats,convert,trim}
                ...

Rhythmic Partitioning Scripts.

options:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Subcommands:
  Available subcommands

  {calc,plot,annotate,label,info,utils,stats,convert,trim}
    calc                Calculator
    plot                Charts plotter
    annotate            Digital score annotator
    label               JSON file labeler. Annotate JSON file with given
                        labels
    info                JSON data info.
    utils               Auxiliary tools
    stats               Statistical tools
    convert             JSON file converter. Convert JSON to CSV file
    trim                JSON file trimmer. Trim given measures

Further information available at https://github.com/msampaio/rpScripts
```

For subcommands help, try `rpscripts 'subcommand' -h`. For instance:

```
rpscripts plot -h
```

For further information about these programs, check *Programs*.

2.2.1 Quick start

For a quick start, run the commands below. They will create `score.json` file and plot all available charts:

```
rpscripts calc score.mxl
rpscripts plot -a score.json
```

2.3 Package usage

The user can import the entire *rpscripts package* or its main classes:

```
# import rpscripts package
import rpscripts as rp

# import only the desired classes
from rpscripts import Partition, ParsemaeSegment, RPData
```

2.3.1 Main classes

The RPScripts main classes are:

class rpscripts.**Partition**(*parts=None*)

Main partition class.

class rpscripts.**ParsemaeSegment**(***kwargs*)

Parsema segment class.

class rpscripts.**RPData**(*path=None*)

Main Rhythmic Partitioning Data class.

2.3.2 Calculator module usage

The snippet below parses a given `score.mxl` digital score file, calculates the rhythmic partitions and saves the data into the `score.json` file.

```
import music21
import pandas

import rpscripts as rp

# Generate Music21 score stream
sco = music21.converter.parse('score.mxl')
segment = rp.ParsemaeSegment()
segment.make_from_music21_score(sco)

# Get partitions data as a dictionary
dic, values_ape = segment.get_data()

# Generate Pandas Dataframe object and export to CSV.

df = pandas.DataFrame(dic)
df.to_csv('score.csv')

# Save data as a JSON file.
rpdata = segment.make_rpdata('score.json')
rpdata.save_to_file()
```

2.3.3 Plotter module usage

The snippet below loads a given `score.json` (generated in the previous snippet) and plots simple and bubble partitiograms.

```
import pandas

import rpscripts as rp

# Load the JSON data generated in the previous snippet.
rpdata = rp.RPData('score.json')
```

(continues on next page)

(continued from previous page)

```
# Plot simple partitiogram (Jupyter).
# Jupyter needs %matplotlib inline to render the chart
chart = rp.plotter.SimplePartitiogramPlotter(rpdata)
chart.plot()

# Generate bubble partitiogram and save to file.
chart2 = rp.plotter.BubblePartitiogramPlotter(rpdata)
chart2.plot()
chart2.save()

# Change a default constant
rp.plotter.LABELS_SIZE = 20
```

2.3.4 Partition class usage

For `rpscripts.Partition` usage:

```
import rpscripts as rp

# Instantiate an object Partition
p = rp.Partition('1^2.2')

# Get agglomeration and dispersion indexes
p.get_agglomeration_index()
p.get_dispersion_index()
```

2.3.5 Hacking

The user can use RP Scripts' classes to calculate rhythmic partitioning data and plot charts.

For instance, the code below extends the *`AbstractTimePlotter`* to plot the number of parts of partitions in time.

```
import rpscripts as rp

def count_parts(rpdata: rp.RPData) -> int:
  return [rp.Partition(p).get_parts_size() for p in rpdata.partitions]

class PartsSizeTimePlot(rp.plotter.AbstractTimePlotter):
  def __init__(self, rpdata: rp.RPData,
               image_format='svg', close_bubbles=False,
               show_labels=False) -> None:

      self.name = 'parts_size'
      super().__init__(rpdata, image_format, show_labels)
      self.parts_size = count_parts(rpdata)

  def plot(self):
      plt.clf()
      plt.plot(self.x_values, self.parts_size)
      self.make_xticks()
```

(continues on next page)

(continued from previous page)

```
        plt.ylabel('Parts size')

        return super().plot()

rpdata = rp.RPData('score.json')
pstp = PartsSizeTimePlot(rpdata)
pstp.plot()
pstp.save()
```

CHAPTER

THREE

Programs

RP Scripts comprises several programs to calculate, plot, annotate, label, and trim rhythmic partitioning data.

3.1 Calculator

The Calculator program extracts rhythmic partitioning data from given digital scores and saves it into a JSON file.

Output JSON file contains:

1. **`texture_data` with events' data split in separate lists:**
    1. Index (`measure number + offset`)
    2. Measure number
    3. Offset
    4. Global offset
    5. Duration
    6. Partition
    7. Density number
    8. Agglomeration index
    9. Dispersion index
2. `offset_map` with a map of measure numbers and their global offsets
3. `values_map` with partitions and the values of their agglomeration and dispersion indexes
4. `partitions` with a single list of each event partitions

Its basic usage is:

```
rpscripts calc score.xml
```

The `-c` option also creates a CSV file with the events data (see *Converter* section).

The combined `-e` and `-c` options create a CSV file with equally-sized events. This procedure is helpful for statistical operations such as frequency analysis.

```
rpscripts calc -e score.xml
```

The option `-h` prints the program help:

```
rpscripts calc -h
```

Output:

```
usage: rpscripts calc [-h] [-c] [-e] filename

positional arguments:
filename             digital score filename (XML, MXL, MIDI and KRN)

options:
-h, --help           show this help message and exit
-c, --csv            output data in a CSV file.
-e, --equally_sized  generate equally-sized events
```

3.2 Plotter

Plotter returns partitiogram and indexogram charts from a given JSON file generated by *Calculator*. This command generates basic charts:

```
rpscripts plot score.json
```

Note: All the examples below are from Robert Schumann's op. 48, n. 2 (See the examples folder).

Plotter accepts multiple options to define image format, resolution, and indexogram type.

The option `-h` prints the command help:

```
usage: rpscripts plot [-h] [-f IMG_FORMAT] [-r RESOLUTION] [-a] [-u] [-w] [-m]
                     [-fl] [-c] [-e] [-t] [-p] [-b]
                     [--minimum_dispersion MINIMUM_DISPERSION]
                     [--maximum_dispersion MAXIMUM_DISPERSION]
                     [--minimum_agglomeration MINIMUM_AGGLOMERATION]
                     [--maximum_agglomeration MAXIMUM_AGGLOMERATION]
                     [--maximum_points_to_label MAXIMUM_POINTS_TO_LABEL]
                     [--dots_size DOTS_SIZE] [--labels_size LABELS_SIZE]
                     [--labels_distance LABELS_DISTANCE]
                     [--indexogram_slope INDEXOGRAM_SLOPE]
                     [--figure_dimensions FIGURE_DIMENSIONS]
                     filename

positional arguments:
filename              JSON filename (calc's output)

options:
-h, --help            show this help message and exit
-f IMG_FORMAT, --img_format IMG_FORMAT
                        Charts output format (svg, png, or jpg)
-r RESOLUTION, --resolution RESOLUTION
                        PNG image resolution. Default=300
-a, --all             Plot all available charts
-u, --bubble_partitiogram
                        Partitiogram as a bubble chart
```

(continues on next page)

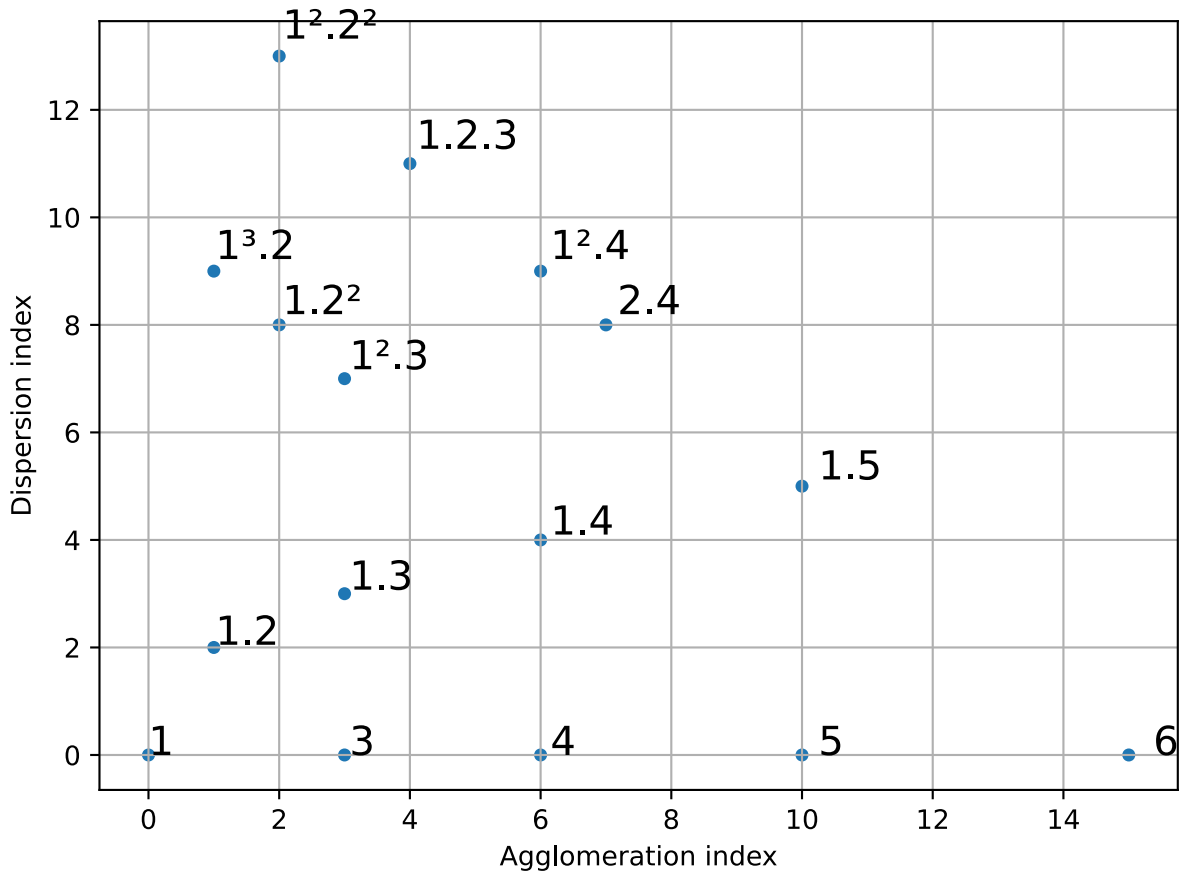

Fig. 1: Partitiogram

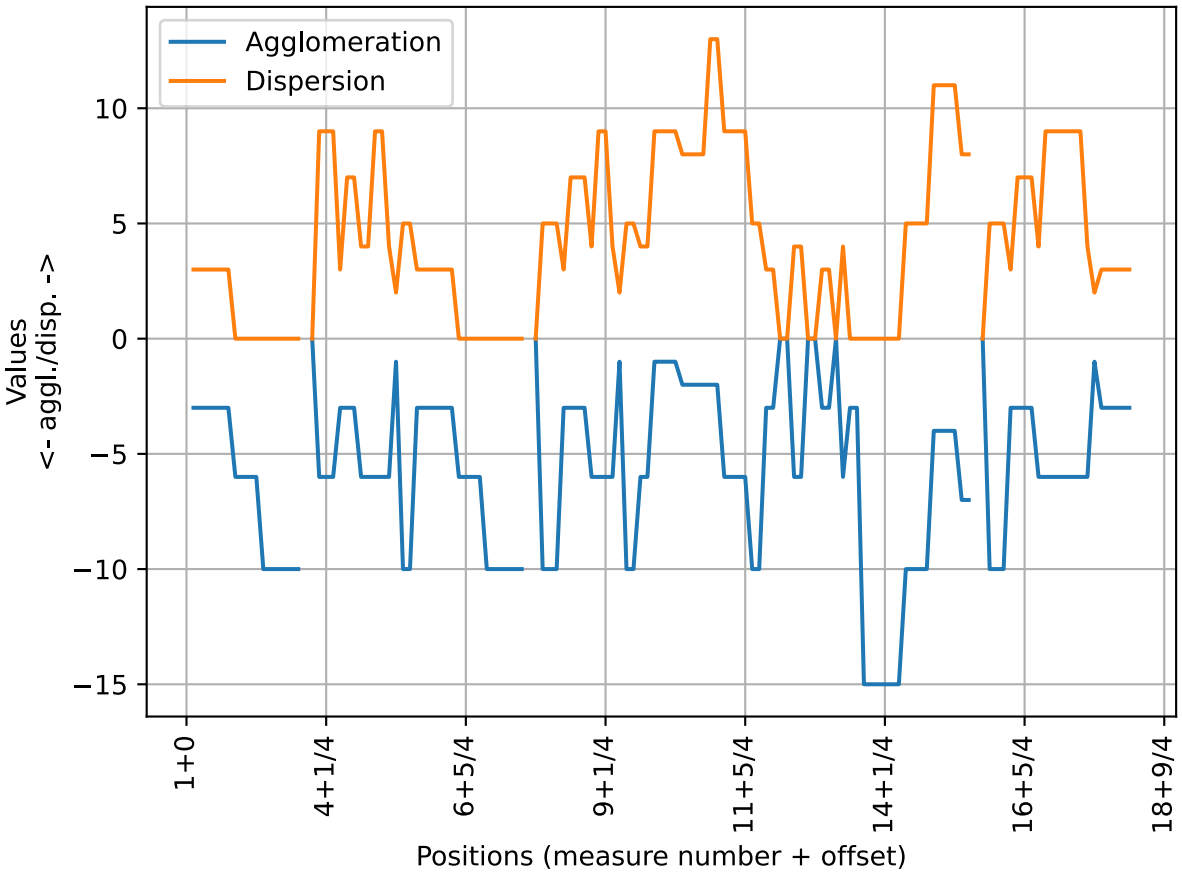

Fig. 2: Indexogram

(continued from previous page)

```
-w, --without_labels  Partitiogram as a bubble chart without labels
-m, --comparative_partitiogram
                        Comparative partitiogram. It demands a previous
                        labeled file. Check rpscripts labels -h' column
-fl, --show_form_labels
                        Draw vertical lines to display given form labels. It
                        demands a previous labeled file. Check rpscripts
                        labels -h' column
-c, --close_bubbles   Indexogram with bubbles' closing lines
-e, --stem            Indexogram as a stem chart
-t, --stairs          Indexogram as a stair chart
-p, --step            Indexogram as a step chart
-b, --combined        Indexogram as a combination of aglomeration and
                        dispersion
--minimum_dispersion MINIMUM_DISPERSION
                        Partitiogram minimum dispersion value to render
--maximum_dispersion MAXIMUM_DISPERSION
                        Partitiogram maximum dispersion value to render
--minimum_agglomeration MINIMUM_AGGLOMERATION
                        Partitiogram minimum agglomeration value to render
--maximum_agglomeration MAXIMUM_AGGLOMERATION
                        Partitiogram maximum agglomeration value to render
--maximum_points_to_label MAXIMUM_POINTS_TO_LABEL
                        Maximum number of points to label in bubble
                        partitiogram chart. Default=50
--dots_size DOTS_SIZE
                        Dots size in simple partitiogram chart. Default=15
--labels_size LABELS_SIZE
                        Labels size in partitiogram chart. Default=15
--labels_distance LABELS_DISTANCE
                        Distance between points and labels in partitiogram
                        chart. Default=1.025
--indexogram_slope INDEXOGRAM_SLOPE
                        Slope's X-distance. Default=1/4 (use always rational
                        numbers)
--figure_dimensions FIGURE_DIMENSIONS
                        Figure dimensions. Default=6.4,4.8 (comma separated
                        values)
```

3.2.1 Image format

Use the `-f` option to set the chart images format. Otherwise, Plotter generates them in svg format.

```
rpscripts plot -f svg score.json
rpscripts plot -f png score.json
rpscripts plot -f jpg score.json
```

3.2.2 Image resolution

Use the `-r` option to set JPG or PNG image resolution.

```
rpscripts plot -f png -r 300 score.json
```

3.2.3 Image dimensions

Use the `--figure_dimensions` option to set the image dimensions:

```
rpscripts plot --figure_dimensions 16.0,4.8 score.json
```

3.2.4 Partitiogram options

Use the `--maximum_points_to_label` option to set the maximum number of points to label in the partitiogram charts. The default value is 50:

```
rpscripts plot --maximum_points_to_label 20 -u score.json
```

Use the `--labels_size` option to set labels' size in the partitiogram charts. Default is 15:

```
rpscripts plot --labels_size 20 -u score.json
```

Use the `--labels_distance` option to set the distance between points and labels in the partitiogram charts. The default value is 1.025:

```
rpscripts plot --labels_distance 2 -u score.json
```

Use the `--dots_size` option to define the size of the dots in the partitiograms charts. The default value is 15 (the previous example's value is 10):

```
rpscripts plot --dots_size 10 -m score.json
```

Bubble partitiogram

Use the `-u` option to plot bubble partitiograms:

```
rpscripts plot -u score.json
```

Comparative partitiograms

Use the `-m` option to plot comparative partitiograms. It demands a labeled `JSON` file. The *Labeler* program generates the labeled file.

Plotter generates comparative partitiograms for the combination of pairs of all available labels.

```
rpscripts plot -m score.json
```

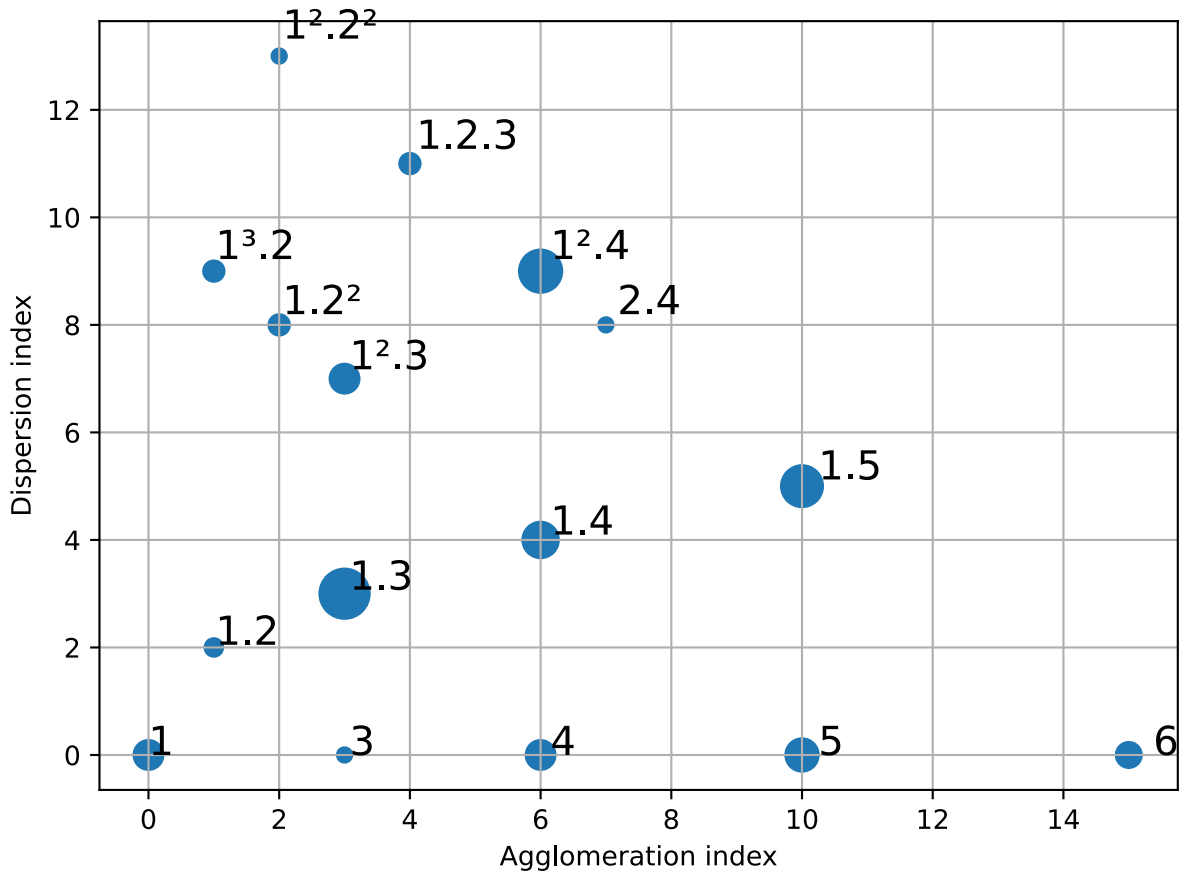

Fig. 3: Bubble partitiogram

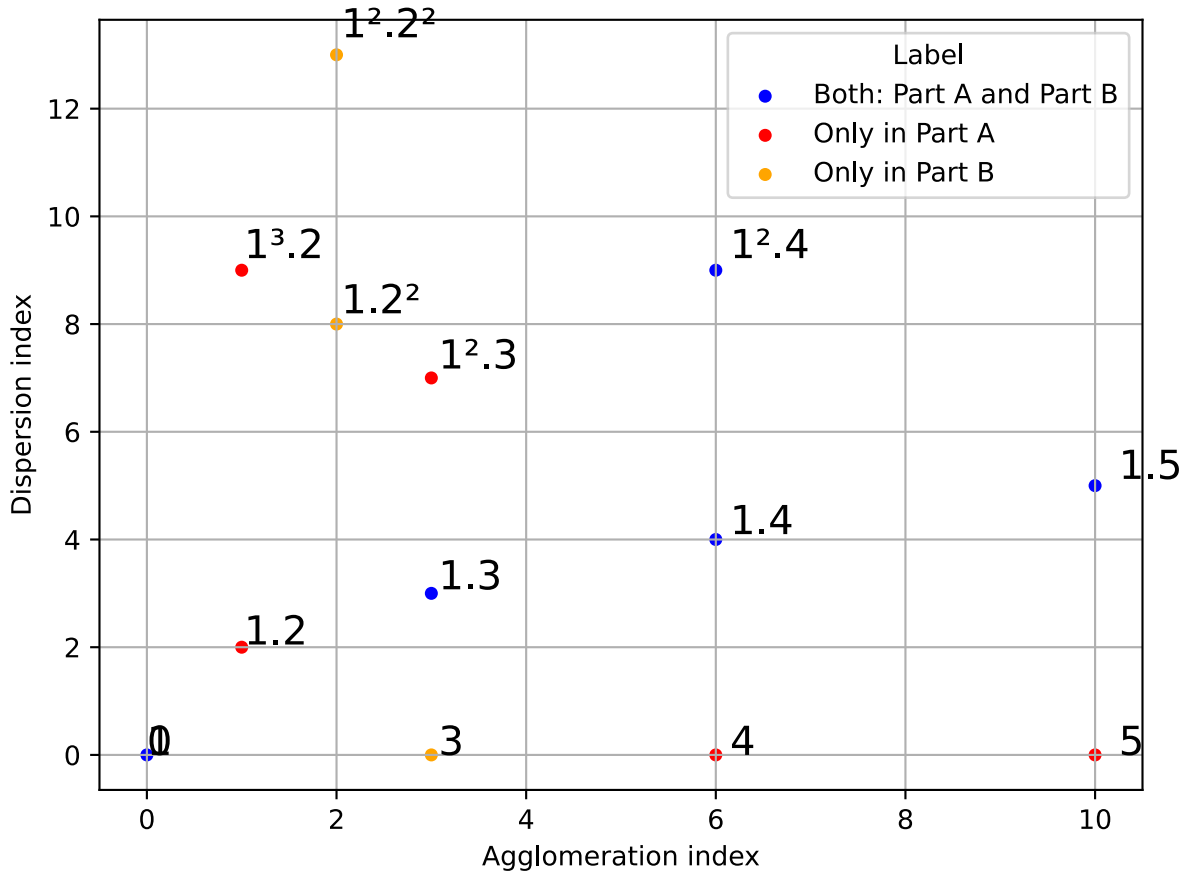

Fig. 4: Comparative partitiogram: parts A and B

Filters

Use the `--minimum_dispersion`, `--maximum_dispersion`, `--minimum_agglomeration`, and `--maximum_agglomeration` options to filter the plotted partitiogram data.

```
rpscripts plot --minimum_dispersion 6 --maximum_agglomeration 8 score.json
```

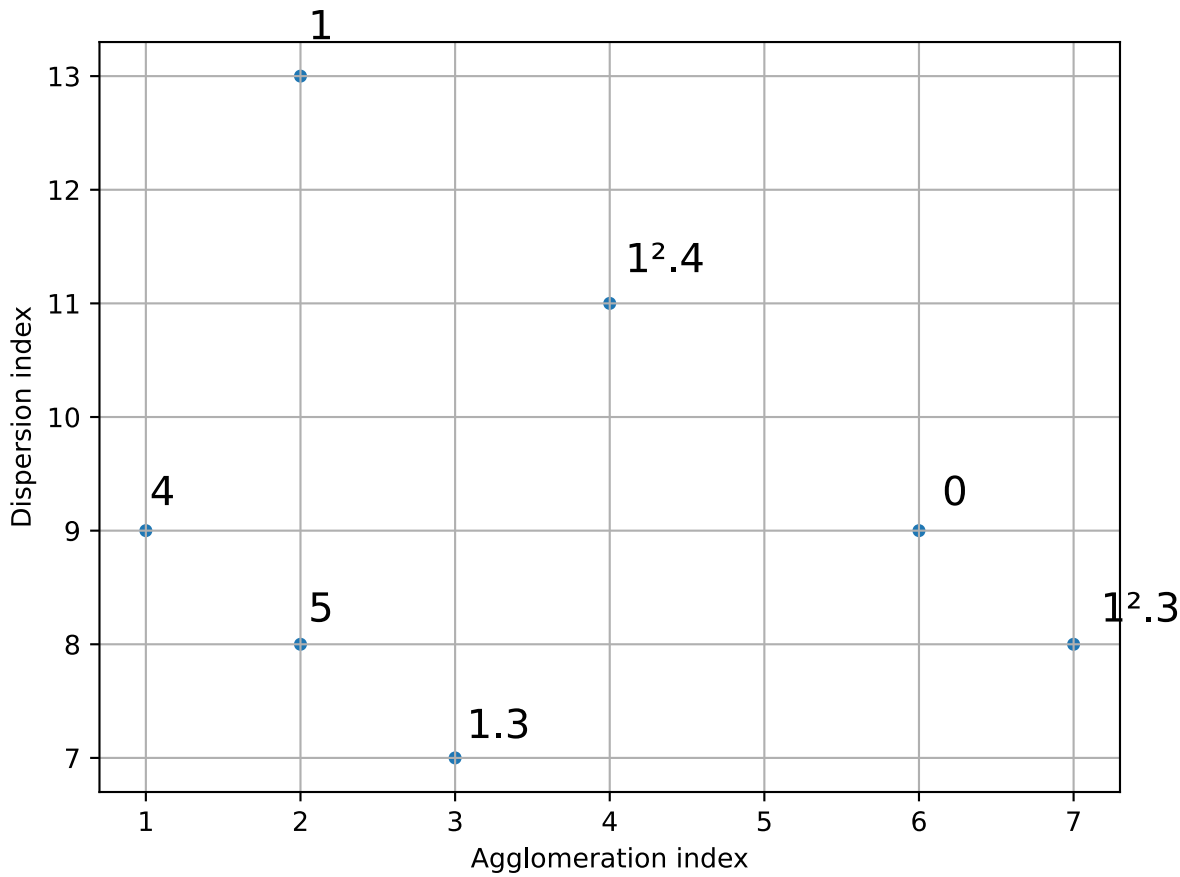

Fig. 5: Filtered partitiogram

3.2.5 Indexogram options

Stem style

Use the `-e` option to plot indexogram in stem style:

```
rpscripts plot -e score.json
```

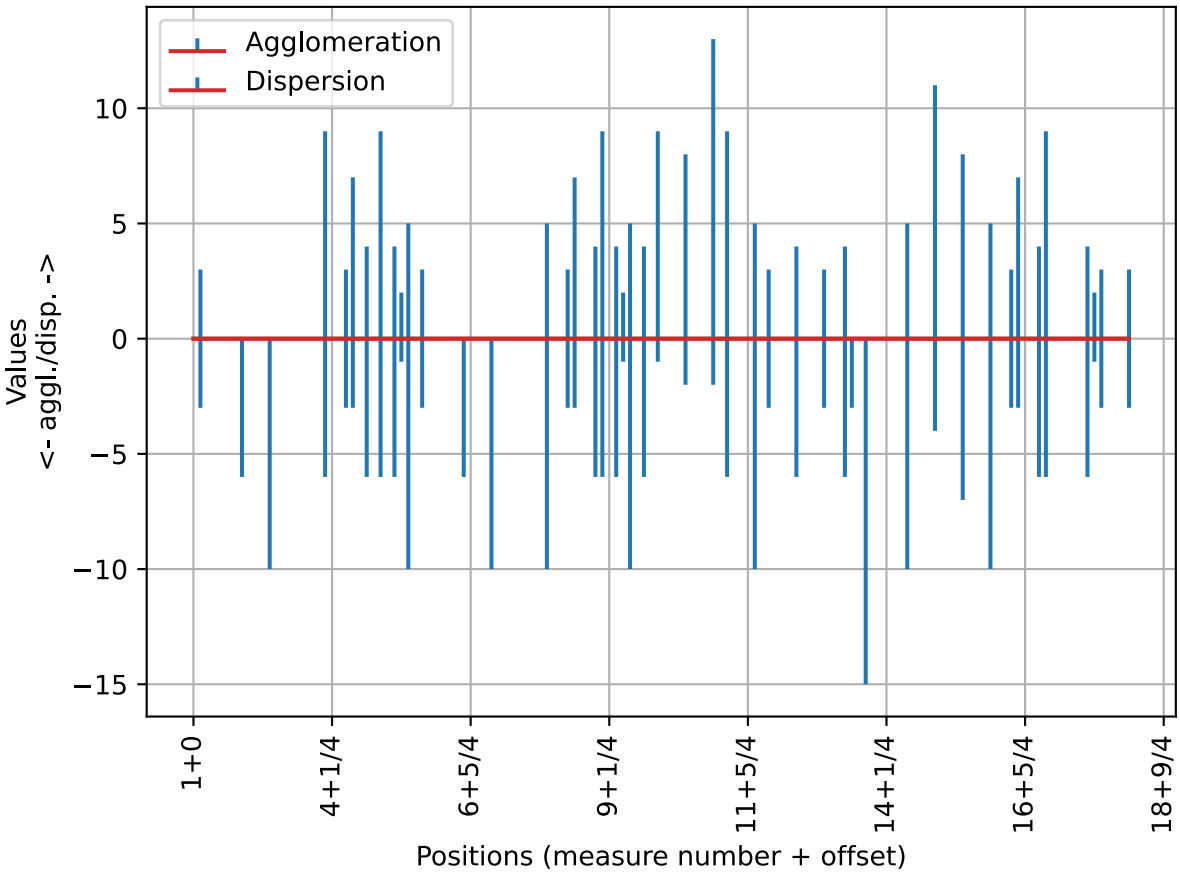

Fig. 6: Indexogram as stem chart

Step style

Use the `-p` option to plot indexogram in stem style:

```
rpscripts plot -e score.json
```

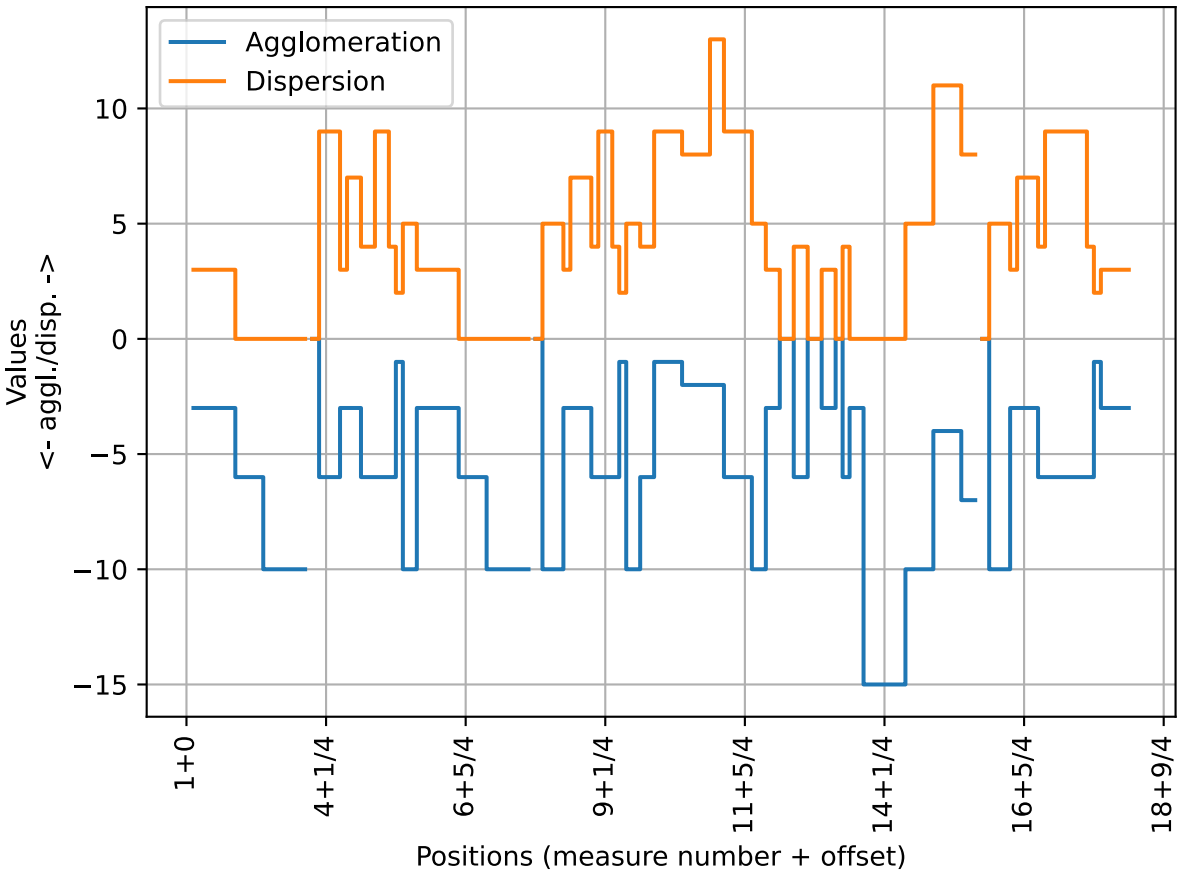

Fig. 7: Indexogram as step chart

Stairs style

Use the `-t` option to plot indexogram in stairs style:

```
rpscripts plot -t score.json
```

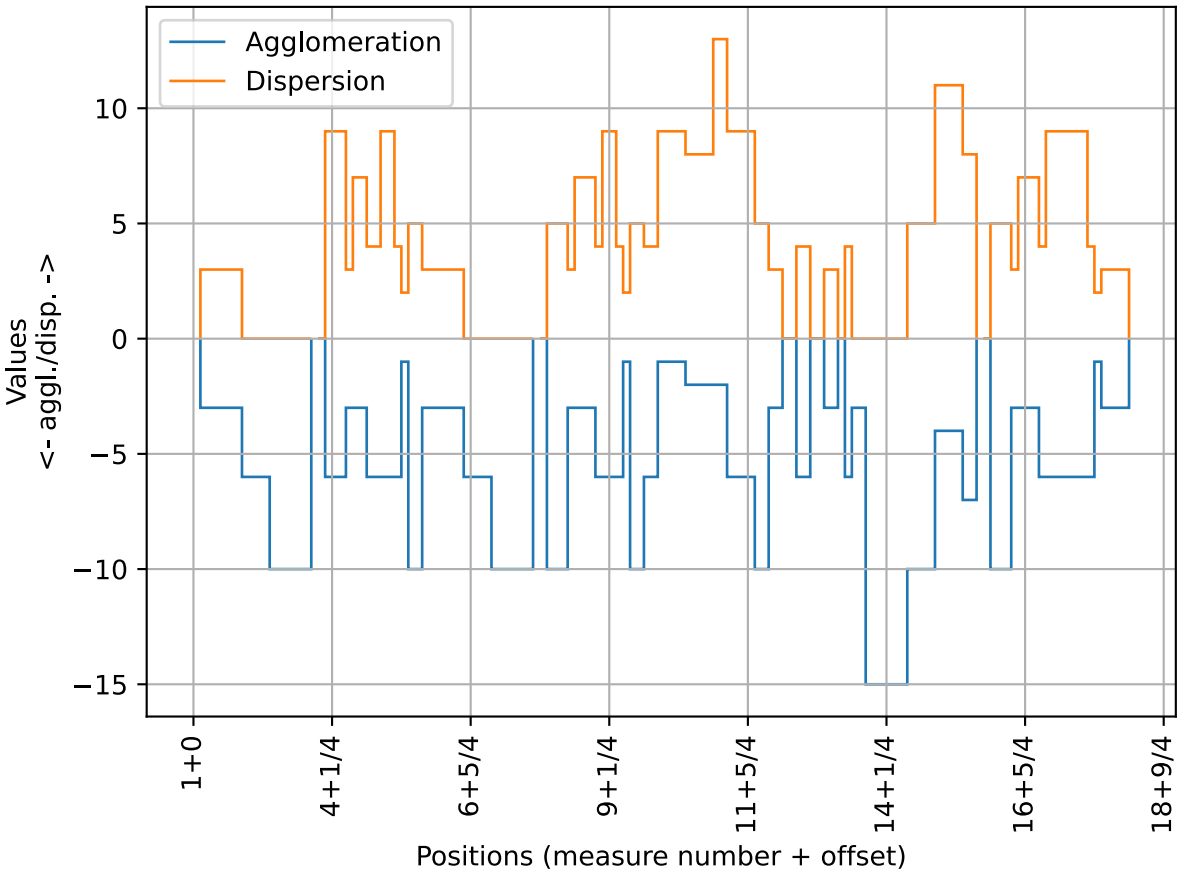

Fig. 8: Indexogram as stair chart

Combined style

Use the `-b` option to plot indexogram in combined style:

```
rpscripts plot -b score.json
```

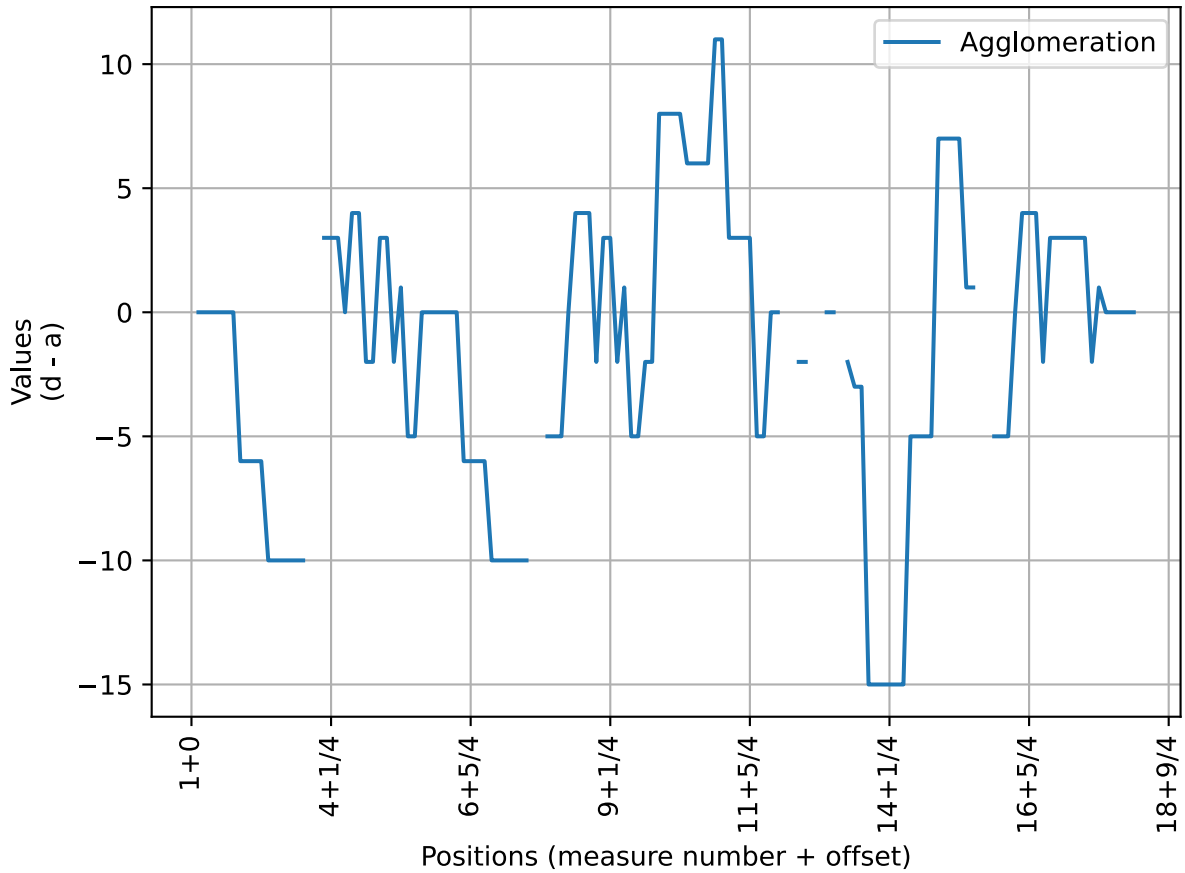

Fig. 9: Indexogram as combined chart

Closing bubbles

For the Simple indexogram, use the `-c` option to plot vertical lines closing indexogram bubbles:

```
rpscripts plot -c score.json
```

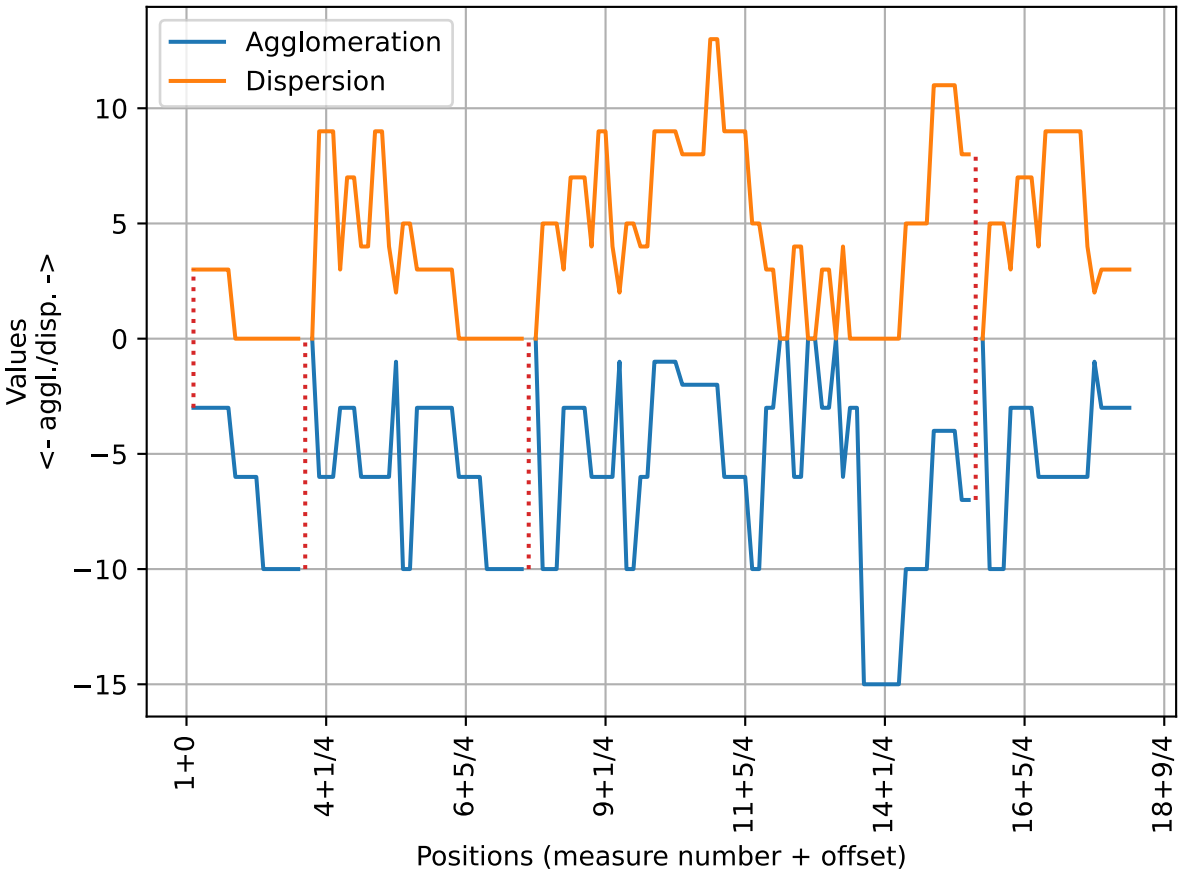

Fig. 10: Indexogram with closing lines

Form labels displaying

For the labels displaying, use the `-fl` option. It adds vertical lines in the indexogram. It demands a labeled `JSON` file. The *Labeler* program generates the labeled file.

```
rpscripts plot -fl score.json
```

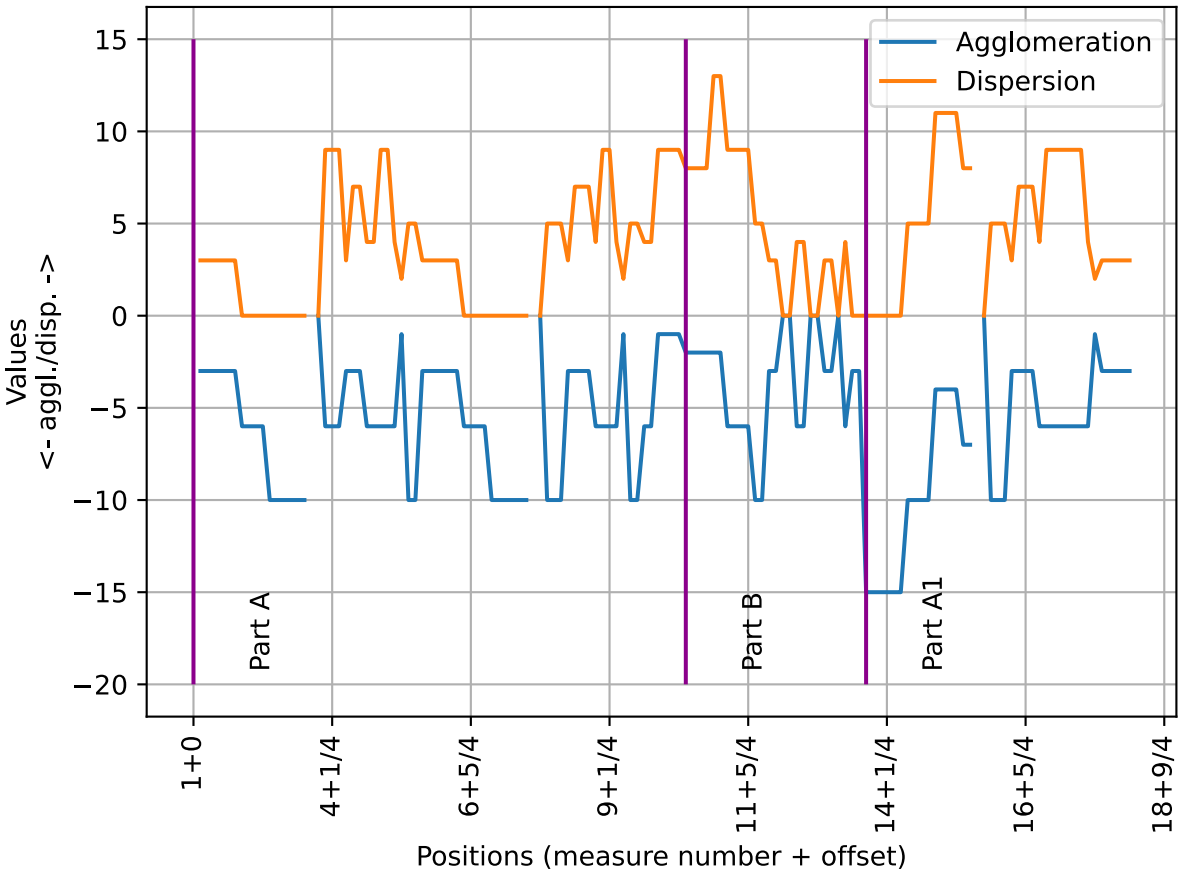

Fig. 11: Indexogram with form labels indications

Sloping X-distance

For the Simple and Combined indexograms, use `--indexogram_slope` option to set the slope's X-distance. The slopes between adjacent points is helpful in partitioning operations identification.

A low value generates a chart similar to *Stairs style* and *Step style*.

3.2.6 All charts

Use the `-a` option to plot all available indexogram types charts:

```
rpscripts plot -a score.json
```

3.3 Annotator

Annotator generates an annotated digital score from a given score and a JSON file generated by *Calculator*.

The basic command line is:

```
rpscripts annotate -s score.xml score.json
```

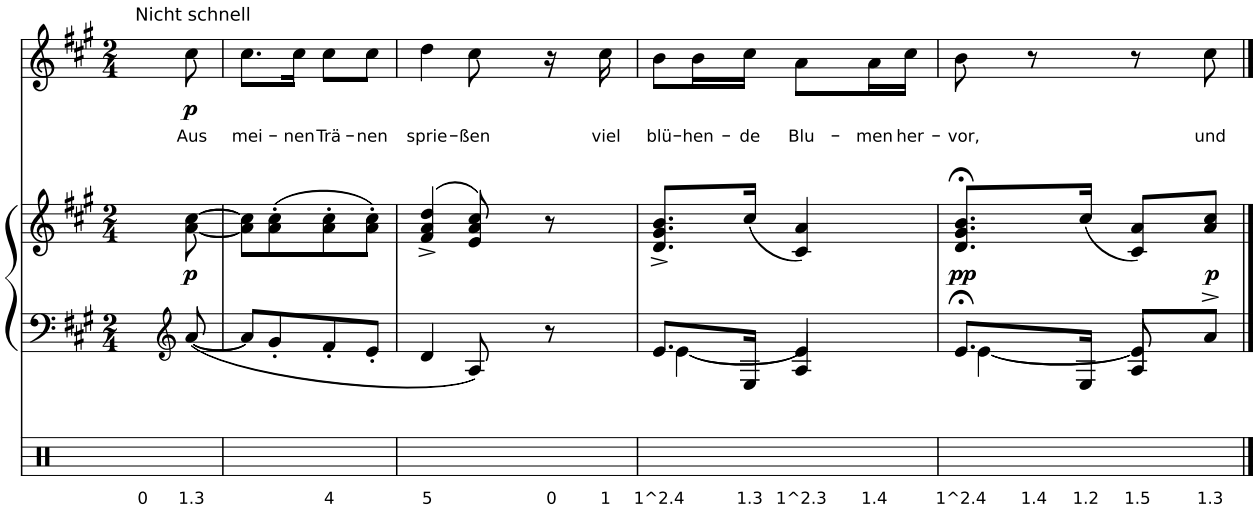

Fig. 12: Annotated score

Annotator does not work with trimmed scores *Trimmer*.

Option `-h` returns the program's help:

```
usage: rpscripts annotate [-h] -s SCORE_FILENAME [-t TYPE] filename

positional arguments:
filename              JSON filename (calc's output)

options:
-h, --help            show this help message and exit
-s SCORE_FILENAME, --score_filename SCORE_FILENAME
```

(continues on next page)

(continued from previous page)

```
                        digital score filename (XML, MXL, MIDI and KRN)
-t TYPE, --type TYPE  type of annotation (partitions)
```

3.3.1 Types

Annotator program can annotate multiple types of data into the given digital score. The default type is `partitions`. Future versions will provide other data types to annotate.

The `-t` option defines the type of annotation:

```
rpscripts annotate -t partitions -s score.xml score.json
```

3.4 Labeler

Labeler adds `label` data into the given `JSON` file (the one generated by *Calculator*). One can add labels such as "Exposition", "First Theme", and so on. *Plotter* needs labeled `JSON` to render comparative partitiograms.

The command below parses the given `score-labels.txt` file and adds it to the `score.json` file.

```
rpscripts label -t score-labels.txt score.json
```

Labeler demands a TXT file with annotations with the label name and its start point, one in each line and separated by commas:

```
One label,1+0
Another label,2+1/4
One more label,2+3/2
```

Option `-h` returns the program's help:

```
usage: rpscripts label [-h] [-t TXT_FILENAME] filename

positional arguments:
filename              JSON filename (calc's output)

options:
-h, --help            show this help message and exit
-t TXT_FILENAME, --txt_filename TXT_FILENAME
                        TXT filename (labels map)
```

See *Comparative partitiograms* and *Form labels displaying* for some labels applications.

3.5 Info

Info command shows a few pieces of information about the JSON data, such as:

1. The number of distinct partitions.
2. The number of distinct density numbers.
3. Highest values of agglomeration and dispersion index.

```
rpscripts info score.json
```

Command's output:

```
This file contains labels data: True
Number of events: 55
Number of distinct partitions: 17
Number of distinct density numbers: 6
Ratio partitions/dn: 2.83
Highest dispersion index: 13
Highest agglomeration index: 15
```

3.6 Stats

Stats command plots agglomeration and dispersion indexes histograms and prints statistical summary about the given JSON data.

```
rpscripts stats score.json
```

Command's output:

```
Statistical summary: full
    Agglomeration  Dispersion
count          55.00       55.00
mean            4.47        3.84
std             3.56        3.39
min             0.00        0.00
25%             1.50        0.00
50%             3.00        4.00
75%             6.00        5.00
max            15.00       13.00
```

3.7 Utils

Utils command provides additional tools for rhythmic partitioning analysis. For instance, for lattice map creation with cardinality 50, run:

```
rpscripts utils -sm 50
```

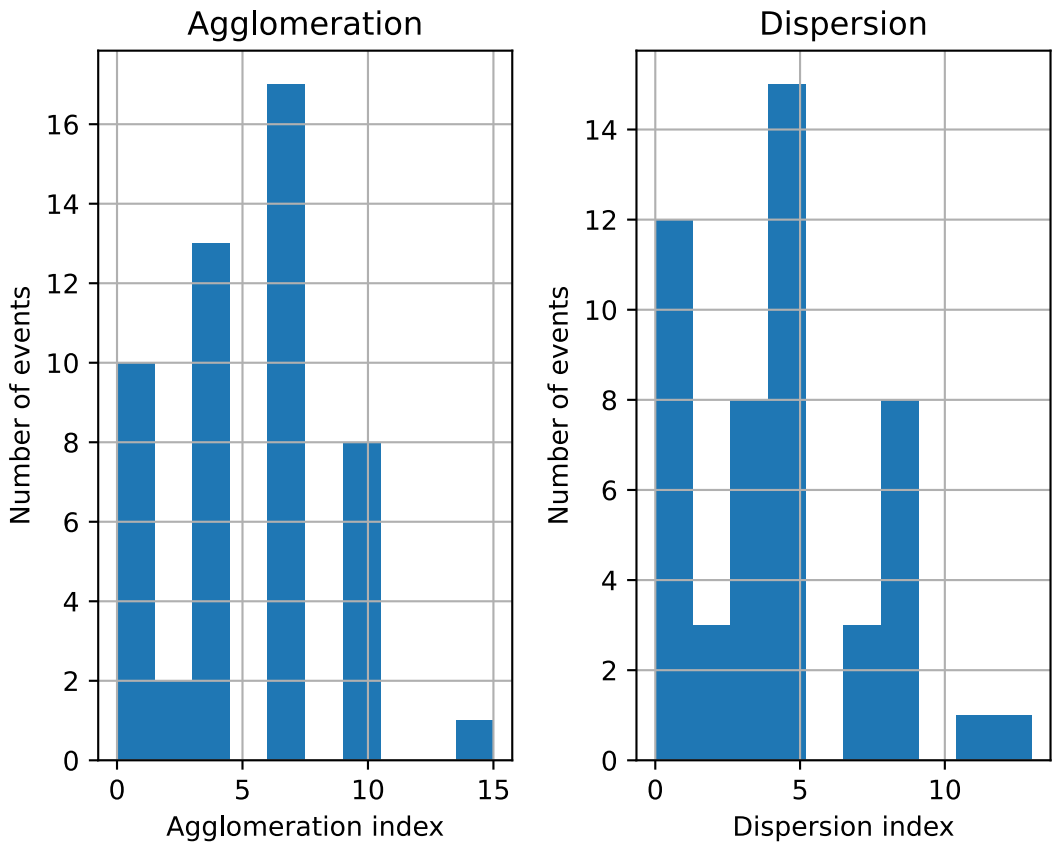

Fig. 13: Histograms

3.8 Converter

The converter command saves a given JSON data into a CSV file.

Option `-h` returns program's help:

```
usage: rpscripts convert [-h] [-e] filename

positional arguments:
filename             JSON filename (calc's output)

options:
-h, --help           show this help message and exit
-e, --equally_sized  generate equally-sized events
```

Option `-e` creates a CSV file with equally-sized events. This procedure is helpful for statistical operations such as frequency analysis.

3.9 Trimmer

Trimmer command returns a sliced JSON data with the content between given measure numbers:

```
rpscripts trim -s 10 -e 20 score.json
```

Option `-h` prints program's help:

```
usage: rpscripts trim [-h] [-s START] [-e END] filename

positional arguments:
filename              JSON filename (calc's output)

options:
-h, --help            show this help message and exit
-s START, --start START
                        Start measure. Blank means "from the beginning"
-e END, --end END     End measure. Blank means "to the end"
```

Note: Trimmed JSON data is not suitable for the *Annotator* processing.

CHAPTER

FOUR

Modules

4.1 rpscripts package

4.1.1 Subpackages

rpscripts.lib package

Submodules

rpscripts.lib.base module

This module provides RPScripts' basic classes and methods

exception rpscripts.lib.base.**CustomException**

Bases: Exception

Generic custom exception.

class rpscripts.lib.base.**EventLocation**(***kwargs*)

Bases: object

Temporal event location.

make_str_index()

Return string index (measure number + offset).

parse_str_index()

Parse string index and store in the class attributes.

class rpscripts.lib.base.**GeneralSubparser**(*subparser: ArgumentParser*)

Bases: object

Argparse subparser abstract class.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*) → None

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

class rpscripts.lib.base.**RPData**(*path=None*)

Bases: object

Main Rhythmic Partitioning Data class.

get_agglomeration_dispersion(*partition_str: str*) → list

Find and return a given partition's agglomeration and dispersion values as a two-element list.

These values must be present in the *values_map* class attribute and the partition value must be in string format.

get_events_location(*attribute: str*) → dict

Return a dictionary with the event locations where the measure number is the dictionary key, and the pair "offset, element", the dictionary value. The *element* is the value in the *attribute* list.

If the *attribute* value is *partitions*, the *element* is the partition as string. For instance, *1.3*, *2^2* and so on.

get_frequency_counter(*attribute: str*, *proportional=True*) → dict

Return a frequency counter of the values of a given attribute.

get_probability_counter(*attribute: str*, *exclude_repeats=True*, *proportional=True*) → dict

Return a probability counter of adjacent values of a given attribute.

load_from_file() → None

Load data from the file set into *path* attribute and fill the other class attributes with this loaded data.

make_class_graph(*attribute: str*) → Digraph

Return a *graphviz.Digraph* with the adjacent values of a list of values stored in the given *attribute*.

save_to_csv(*equally_sized=False*) → None

Save the data into a CSV file.

If equally_sized parameter is true, the events are proportionally divided into smaller events of a unique duration.

save_to_file(*filename=None*) → None

Save the class data into a JSON file.

to_json() → dict

Return the data as a dictionary with fractions formated to json.

trim(*start_pointer: int*, *end_pointer: int*)

Return a new *RPData* object with trimmed attributes' data.

Only values_map is kept as the original *RPData* object.

rpscripts.lib.base.**aux_find_next_measure_number**(*global_offset: Fraction*, *offset_map: dict*, *index=0*) → tuple

Return the nearest measure number that the global offset value is higher than the given one as parameter.

This function also updates and returns the given index.

rpscripts.lib.base.**aux_sum_if_none**(*a*, *b*)

Return the sum if one or two given values is *None*.

rpscripts.lib.base.**clean_filename**(*filename: str*) → str

Clean the given filename.

Remove the special characters and spaces.

`rpscripts.lib.base.`**`convert_texture_data_from_json`**(*data: dict*) → dict

Convert texture data from JSON.

The fractions are converted from string to Fraction objects.

`rpscripts.lib.base.`**`convert_texture_data_to_json`**(*data: dict*) → dict

Convert texture data to JSON.

The fractions are converted from Fraction objects to strings.

`rpscripts.lib.base.`**`convert_to_equal_durations`**(*dataframe: DataFrame*, *offset_map: dict*) → dict

Convert texture data from a given DataFrame to dictionary format with events of the same duration.

For instance, a dataframe with two events with durations 1/2 and 1/3 are converted in 3 events and 2 events of duration 1/6.

`rpscripts.lib.base.`**`dump_json_data`**(*filename: str*, *data*) → None

Dump data to json file.

`rpscripts.lib.base.`**`file_rename`**(*filename: str*, *extension: str*, *suffix=None*) → str

Rename files to change extension value and add optional suffix.

`rpscripts.lib.base.`**`find_nearest_smaller`**(*value*, *seq: list*)

Find the smaller nearest value of a given value from a given sequence.

`rpscripts.lib.base.`**`fraction_to_string`**(*value*) → str

Return a given value as a fraction represented as a string format.

`rpscripts.lib.base.`**`get_diff_lcm`**(*seq: list*) → Fraction

Return the lowest common multiple of the differences between the adjacent values of a given list.

`rpscripts.lib.base.`**`get_fractions_denominator_lcm`**(*fractions_lst: list*)

Return the lowest common multiple value of a list of fractions denominators.

`rpscripts.lib.base.`**`get_number_combinations_pairs`**(*n: int*) → float

Return the number of pair combinations of a list of *n* elements.

Binomial coefficient.

`rpscripts.lib.base.`**`load_json_file`**(*filename: str*) → dict

Load JSON file.

`rpscripts.lib.base.`**`make_fraction`**(*value*) → Fraction

Return a Fraction object from a given Fraction or float value.

`rpscripts.lib.base.`**`make_general_graph`**(*labels: list*, *name: str*) → Digraph

Return a *graphviz.Digraph* with the adjacent values of a given list of labels.

`rpscripts.lib.base.`**`parse_fraction`**(*value*) → Fraction

Return a Fraction object from a given value.

`rpscripts.lib.base.`**`parse_pow`**(*partition*) → str

Superscript pow values in given partitions.

`rpscripts.lib.base.`**`save_dict_into_csv_file`**(*dic: dict*, *filename: str*) → None

Save a given dictionary into a CSV file.

rpscripts.lib.partition module

This module provides classes and functions for rhythmic partitioning analysis. For further information, see Gentil-Nunes 2009.

Gentil-Nunes, Pauxy. 2009. "Análise Particional: uma Mediação entre Composição Musical e a Teoria das Partições." Ph.D. Dissertation, Universidade Federal do Estado do Rio de Janeiro.

class rpscripts.lib.partition.**Partition**(*parts=None*)

Bases: `object`

Main partition class.

as_string() → str

Return the partition as a string such as "1^3.2".

count_binary_relations()

Count binary relations of partition's parts.

get_agglomeration_index() → int

Return the partition's agglomeration index.

get_density_number() → int

Return the partition's density number.

get_dispersion_index() → int

Return the partition's dispersion index.

get_parts_size() → int

Count the partition's number of parts.

resize() → list

Return the list of current's resized partitions.

revariate() → list

Return the list of current's revariated partitions.

transfer(*default=True*) → list

Return the list of current's transfered partitions.

class rpscripts.lib.partition.**PartitionComparator**(*partition_1:* Partition, *partition_2:* Partition)

Bases: `object`

get_jacquard_similarity()

Return Jacquard similarity value. See Moreira, 2019.

class rpscripts.lib.partition.**PartitionLattice**(*cardinality=10*)

Bases: `object`

Partition Lattice class. It helps in lattice creating and saving.

save_file()

Save lattice map into file.

rpscripts.lib.partition.**get_lexset**(*number: int*) → list

Return the lexical set of a given *number*. The lexical set is the list of all partitions from 1 to number.

rpscripts.lib.partition.**get_partitions**(*number: int*) → list

Return the partitions of a given *number* as a list of lists.

`rpscripts.lib.partition.make_ryp_map(higher_cardinality: int) → dict`

Make a rhythmic partitioning Young lattice of a given cardinality.

`rpscripts.lib.partition.make_subseq(partition, n) → list`

Return a subset array from a given partition from 0 to *n* element.

Module contents

4.1.2 Submodules

4.1.3 rpscripts.annotator module

This module provides annotation features. It adds data such as partition into the XML score data and saves as a MXL file.

`class rpscripts.annotator.Subparser(subparser: ArgumentParser)`

Bases: `GeneralSubparser`

Implements argparser.

`add_arguments() → None`

Add *argparse.ArgumentParser* arguments.

`handle(args)`

Parse arguments and run the module functions.

`setup() → None`

Basic setup.

Set program_name and help message.

`rpscripts.annotator.main(m21_score: Score, rpdata: RPData, outfilename: str, labels_name='partitions') → None`

Add data from given RPData into the given Music21 score and save its in a new outfile.

4.1.4 rpscripts.calculator module

This module provides classes and functions to calculate the rhythmic partitions from a given digital score.

`class rpscripts.calculator.MusicalEvent(**kwargs)`

Bases: `object`

Auxiliary musical event class.

This class has only the needed attributes of Music21's Note, Rest and Chord classes.

`is_rest()`

Check if the current object represents a music21' rest object.

`set_data_from_m21_obj(m21_obj, measure_number, measure_offset, offset=None)`

Get data from given Music21 object, set as current object's attributes, and return offset and duration.

class rpscripts.calculator.**Parsema**(***kwargs*)

Bases: object

Auxiliary Parsema class.

Parsema is the set of adjacent equal partitions. See Gentil-Nunes 2009 for further information.

add_single_events(*single_events: list*) → None

Set single events, calculate their durations and set partition.

set_partition() → None

Create *Partition* object from single events attribute and set it to *partition* attribute.

class rpscripts.calculator.**ParsemaeSegment**(***kwargs*)

Bases: object

Parsema segment class.

get_data() → tuple

Get partitions, agglomeration, and dispersion data and their locations.

make_from_music21_score(*m21_score: Score*) → None

Create *Parsema* objects fom given Music21 Score object and store at *parsemae* class attribute.

make_rpdata(*filename: str*) → *RPData*

Return *RPData* object from *parsemae* class attribute.

class rpscripts.calculator.**PartSoundingMap**(***kwargs*)

Bases: object

Sounding Map class of a musical part.

get_single_event_by_location(*global_offset: Fraction*) → *SingleEvent*

Return a *SingleEvent* object from its location.

set_from_m21_part(*m21_part: Part*) → None

Set Music21's part elements into *single_events* attribute.

Create *MusicEvent* objects for each event and then, create *SingleEvent* objects to add to *single_events* attribute.

class rpscripts.calculator.**ScoreSoundingMap**(***kwargs*)

Bases: object

Sounding Map class of a musical score. Individual parts are sounding maps.

add_part_sounding_map(*m21_part: Part*) → None

Creates a *PartSoundingMap* from a given Music21 Part and add it to sounding_maps and attacks attributes.

add_score_sounding_maps(*m21_score: Score*) → None

Create *PartSoundingMap* objects from each part of a given Music21 Score.

This method also get measure offsets and explodes voices into parts.

get_single_events_by_location(*global_offset: Fraction*) → list

Return a list of *SingleEvent* objects in different sounding_maps from their locations.

make_parsemae() → list

Return a list of *Parsema* objects from *attacks* attribute.

This method also handles merged parsemae.

class rpscripts.calculator.**SingleEvent**(***kwargs*)

Bases: object

Auxiliary single event. It's more simple than Music21's note and rest objects and has useful attributes such ass number of pitches and sounding.

class rpscripts.calculator.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.calculator.**aux_join_music_events**(*events: dict*) → dict

Join *MusicalEvent*

This methods join adjacent tied objects as well as adjacent rests.

rpscripts.calculator.**aux_make_events_from_part**(*m21_part: Part*) → dict

Return a dictionary with Musical Events and their locations from a given Music21 part object.

rpscripts.calculator.**auxiliary_get_duration**(*m21_obj*) → Fraction

Return the duration of the given Music21 object as a Fraction object.

rpscripts.calculator.**make_music_events_from_part**(*m21_part: Part*) → dict

Return a dictionary with location and Musical Events from a given Music21 part object. Adjacent rests and tied notes are joined.

rpscripts.calculator.**make_offset_map**(*m21part: Part*) → dict

Create map with measure number and global offset value.

rpscripts.calculator.**split_part_chords**(*m21_part: Part*) → Part

Return a new Music21 Part object with pitches extracted from chords of a given Music21 Part object.

This function splits chords with notes of distinct durations.

rpscripts.calculator.**split_score**(*filename: str*) → Score

Parse a given digital score file, split chords, convert voices to parts and returns a new Music21 Score object.

4.1.5 rpscripts.cli module

This module provides the command line interface.

rpscripts.cli.**main**() → None

Parse the given command line arguments.

4.1.6 rpscripts.config module

This module contains RP Scripts' global variables.

These variables are used in multiple modules.

4.1.7 rpscripts.converter module

This module converts calculator's JSON output into CSV file with or without intermediary equally-sized events.

class rpscripts.converter.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*) → None

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.converter.**main**(*filename: str*, *equally_sized=False*) → None

Create *RPData* object from given filename and save the data into a csv file.

If *equally_sized* parameter is true, the events are proportionally divided into smaller events of a unique duration.

4.1.8 rpscripts.info module

This shows information about data.

class rpscripts.info.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.info.**main**(*filename: str*) → None

Print basic information about the given filename.

4.1.9 rpscripts.labeler module

This module parses TXT label file and adds this information into the JSON file.

class rpscripts.labeler.**Edge**(*label=None*, *index=None*, *global_offset=None*)

Bases: `object`

This class represents the point where a excerpt or section starts.

set_global_offset(*offset_map: dict*) → None

Set the edge global offset from a given offset map.

set_index(*offset_map: dict*) → None

Set the edge index (measure_number + global_offset).

class rpscripts.labeler.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.labeler.**main**(*json_filename: str*, *txt_fname: str*) → None

Add the given TXT file labels into the given JSON file.

rpscripts.labeler.**parse_txt**(*filename: str*) → list

Return list of *Edge* objects created from extracted labels information given in TXT file.

4.1.10 rpscripts.plotter module

class rpscripts.plotter.**AbstractIndexogramPlotter**(*rpdata: RPData*, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractTimePlotter*

Abstract indexogram plotter class.

This class extends AbstractTimePlotter and has dispersion and inverted agglomeration data as attributes to help indexograms plotting.

class rpscripts.plotter.**AbstractPartitiogramPlotter**(*rpdata: RPData*, *image_format='svg'*, *with_labels=True*, ***kwargs*)

Bases: *AbstractPlotter*

Abstract partitiogram plotter class.

This class extends AbstractPlotter and has agglomeration, dispersion and quantity attributes to help Partitiogram scatter plotting.

check_given_limits() → bool

Return True if there are value filters defined.

check_labels()

data_filter() → None

Filter the dispersion and agglomeration indexes values for a narrowed plot.

plot()

Extend AbstractPlotter's method. Set X- and Y-axis labels.

class rpscripts.plotter.**AbstractPlotter**(*rpdata:* RPData, *image_format='svg'*)

Bases: `object`

Main abstract plotter class.

This class has useful attributes and methods to handle RPData objects.

get_subplots()

Return subplots dumped into pickle file.

Useful for multiple chart creation.

plot() → None

Plot Setup.

Depends on the extended classes implementation.

save(*close_figure=True*) → None

Saves the chart into the RPData outname.

xticks_adjust() → None

Apply tight layout method to the plot object.

It is useful for rotated xlabels charts.

class rpscripts.plotter.**AbstractTimePlotter**(*rpdata:* RPData, *image_format='svg'*, *show_labels=False*)

Bases: *AbstractPlotter*

Abstract time plotter class.

This class extends AbstractPlotter and handles X-axis to show measure numbers and offsets.

make_xticks() → None

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

xticks_adjust()

Extend AbstractPlotter's xticks_adjust method.

Rotates xticks.

class rpscripts.plotter.**BubblePartitiogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *with_labels=True*, *bubble_size=2000*, ***kwargs*)

Bases: *AbstractPartitiogramPlotter*

plot()

Extend AbstractPartitiogramPlotter's method. Create scatter plot.

class rpscripts.plotter.**CombinedIndexogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractIndexogramPlotter*

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

class rpscripts.plotter.**ComparativePartitiogramMaker**(*rpdata:* RPData, *image_format='svg'*, *with_labels=True*)

Bases: *AbstractPartitiogramPlotter*

plot()

Call ComparativePartitiogramPlotter's plot method for each plotter object available at *plotters* attribute.

save()

Saves the chart into the RPData outname.

class rpscripts.plotter.**ComparativePartitiogramPlotter**(*rpdata:* RPData, *column_1*, *column_2*, *image_format='svg'*, *with_labels=True*, ***kwargs*)

Bases: *AbstractPartitiogramPlotter*

plot()

Extend AbstractPartitiogramPlotter's method. Create multiple superimposed scatter plots.

class rpscripts.plotter.**SimpleIndexogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractIndexogramPlotter*

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

class rpscripts.plotter.**SimplePartitiogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *with_labels=True*, ***kwargs*)

Bases: *AbstractPartitiogramPlotter*

plot()

Extend AbstractPartitiogramPlotter's method. Create scatter plot.

class rpscripts.plotter.**StairsIndexogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractIndexogramPlotter*

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

class rpscripts.plotter.**StemIndexogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractIndexogramPlotter*

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

class rpscripts.plotter.**StepIndexogramPlotter**(*rpdata:* RPData, *image_format='svg'*, *close_bubbles=False*, *show_labels=False*)

Bases: *AbstractIndexogramPlotter*

plot()

Extend AbstractPlotter's method. Set X-axis label and add optional vertical lines for labels if they are available in RPData.

class rpscripts.plotter.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

4.1.11 rpscripts.stats module

This module provides a few statistical data about the given filename.

class rpscripts.stats.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.stats.**main**(*filename: str*, *no_plot: bool*, *split_labels: bool*) → None

Make histogram and print statistical summary of agglomeration and dispersion indexes.

4.1.12 rpscripts.trimmer module

This module cuts off a given calc's JSON output file by given start and end measure numbers and return a new JSON file.

class rpscripts.trimmer.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.trimmer.**main**(*rpdata:* RPData, *start_measure*, *end_measure*) → None

Trim the given RPData between the given start and end measures.

4.1.13 rpscripts.utils module

This module contains helpful tools for the rhythmic partitioning task.

class rpscripts.utils.**Subparser**(*subparser: ArgumentParser*)

Bases: *GeneralSubparser*

Implements argparser.

add_arguments() → None

Add *argparse.ArgumentParser* arguments.

handle(*args*)

Parse arguments and run the module functions.

setup() → None

Basic setup.

Set program_name and help message.

rpscripts.utils.**copy_default_lattice_map**() → None

Copy lattice map to the default path.

rpscripts.utils.**save_map**(*lexset: int*) → None

Create a partition lattice of *lexset* size and save into the program's default path.

4.1.14 Module contents

Provide main modules and classes.

CHAPTER

FIVE

Bibliography

1. Gentil-Nunes, Pauxy. 2009. "Análise Particional: uma Mediação entre Composição Musical e a Teoria das Partições." Ph.D. Dissertation, Universidade Federal do Estado do Rio de Janeiro.
2. Sampaio, Marcos da Silva, and Pauxy Gentil-Nunes. 2022. "Python Scripts for Rhythmic Partitioning Analysis." *MusMat - Brazilian Journal of Music and Mathematics* 6 (2): 17-55. https://musmat.org/wp-content/uploads/2022/12/02-Sampaio-Gentil-Nunes-V6N2_2022.pdf.
3. Sampaio, Marcos da Silva, Pauxy Gentil-Nunes, Sidnei Marques de Oliveira, Vicente Sanches de Oliveira, Jaderson Cardona de Oliveira. "New Visual Tools for Rhythmic Partitioning Analysis of Musical Texture." *Musica Theorica* 7 (2). https://revistamusicatheorica.tema.mus.br/index.php/musica-theorica/article/view/240/.

5.1 How to cite

Sampaio, Marcos da Silva and Pauxy Gentil-Nunes. RP Scripts: Rhythmic Partitioning Scripts, release 2.0. Available at https://github.com/msampaio/rpScripts. Accessed on May 10, 2023, 2023.

```bibtex
@Misc{              sampaio.ea2023-rpscripts,
    author       = {Sampaio, Marcos da Silva and Gentil-Nunes, Pauxy},
    year         = {2023},
    title        = {{RP Scripts}: {Rhythmic} {Partitioning} {Scripts}, Release 1.1},
    howpublished = "Available at \url{https://github.com/msampaio/rpScripts}. Accessed␣
↪on Mar 1, 2023"
}
```

Bibtex (download)

CHAPTER

SIX

Authors

6.1 Developer

- Marcos da Silva Sampaio

6.2 Contributors

- Pauxy Gentil-Nunes - Rhythmic Partitioning Theory support
- Sidnei Marques de Oliveira - Features testing

CHAPTER

SEVEN

Changelog

7.1 Release 2.0 (released May 13, 2023)

7.1.1 Deprecated

- Moved `RPC` to `calculator`
- Moved `RPP` to `plotter`
- Moved `RPA` to `annotator`
- Moved `RPL` to `labeler`
- Moved multi-`CSV`-based to `JSON`-based

7.1.2 Features added

- New Command line interface
- Trimmer
- Converter (to `CSV` format)
- Statistical data printer
- Lattice map generator
- Sphinx documentation
- PIP package generator
- Binary generator
- *partition* module
- New modules with hacking-easy classes
- Memory optimization: position and duration events replace equally-size based events
- Frequency analysis calculator
- Probability calculator
- Docstrings
- New chart settings options

7.2 Release 1.1 (released Mar 1, 2023)

7.2.1 Features added

- Rhythmic Partitioning Labeler script
- RPP refactoring (class based)
- Bubble partitiogram
- Comparative partitiogram
- Partitiogram dimensions as script options
- Documentation improvement
- Abstract classes for charts
- Optional CSV rendering without equal durations
- Functional tests (RPC)
- Annotation from MIDI files.

7.2.2 Bugs fixed

- Parsing of chords with notes with distinct tie values
- Kern parsing
- Events' duration calculating
- CSV format reading
- Example image's legends

7.3 Release 1.0 (released Dec 29, 2022)

7.3.1 Features added

- RP Scripts documentation (README)
- Standalone RPC Script - MusicXML, KRN and MIDI parser (Music21 based). - Rhythmic Partitioning calculator. - Output containing events with equal durations.
- Standalone RPP Script - Partitiogram - Multiple indexogram types: stairs, stem, combined, and standard (with and without bubble closing' vertical lines) - Image format selection (svg, png, jpg)
- Standalone RPA Script - Annotation in new MusicXML file. - Generation from given MusicXML and Kern files. It doesn't work with MIDI.
- Usage examples

CHAPTER

EIGHT

Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

F

G

H

I

L

M

P

R

S

T

X