
pmp

Katarzyna Banaszak, Bartosz Kusek

Feb 10, 2020

CONTENTS:

1	Installation	1
2	Wiki	3
2.1	Quick Start	3
2.2	File Formats and Structure	4
2.3	Experiment Tutorial	4
2.4	Experiment Module	8
2.5	Rules Module	11
2.6	Preferences Module	16
2.7	Integer linear programming	17
2.8	Using pmp	18
2.9	Examples	18
2.10	About	19
	Python Module Index	21
	Index	23

INSTALLATION

```
pip install python-multiwinner-package
```


If you want to learn more go [here](#).

2.1 Quick Start

Here is short guide to start and run your first elections.

2.1.1 Instalation

Pmp can be obtained by pip.

```
pip install python-multiwinner-package
```

Note: **Don't worry, it's the only time you need to use this long name. In code use convenient abberiviation pmp**

Just in case, you can also install it directly from github source.

```
pip install git+https://github.com/Koozco/pmp.git
```

Development instalation guide can be found [here](#).

2.1.2 Basics - classes, methods, parameters.

Here are all entities necessary for running simple election.

1. Parameters naming Usually we characterize an election with:

- *n* - number of voters
- *m* - number of candidates
- *k* - size of committee

```
n = 5  
m = 3  
k = 2
```

2. Preferences Profile Depending on given rule, preferences are defined in an ordinal or approval model. Single preference always represents single voter.

- Ordinal preferences

```
from pmp.preferences import Ordinal
orders = [
    [1, 2, 0],
    [2, 1, 0],
    [2, 0, 1],
    [1, 2, 0],
    [1, 0, 2]
]

preferences = [Ordinal(o) for o in orders]
```

- Approval preferences

```
from pmp.preferences import Approval
approves = [
    [0, 1, 2],
    [0, 1],
    [1],
    [1, 2],
    [0, 2]
]

preferences = [Approval(a) for a in approves]
```

- Profile With defined preferences next step is to create preferences profile. It consists of candidates and preferences lists.

```
from pmp.preferences import Profile
candidates = [0, 1, 2]

profile = Profile(candidates, preferences)
```

3. Finding committee You always compute winning committee with an instance of a rule. It has method `find_committee` with two obligatory parameters - *k* and *profile*, both which we have defined earlier. With one instance you can compute results for different values of *k* or for different profiles.

```
from pmp.rules import SNTV

sntv = SNTV()
committee = sntv.find_committee(k, profile)
```

For more detailed information we encourage to further reading next sections or go straight to [Examples](#).

2.2 File Formats and Structure

[TBD] This section will describe all necessary conventions around file formats and structure.

2.3 Experiment Tutorial

This tutorial shows how to use main utility of pmp called `experiment` and should help you get familiar with `Experiment` class.

Tutorial goal: Compute election results for some set of scoring rules and visually compare their characteristics. Aggregate results of large number of elections on 2-dimensional histogram. Repeat this task for different points distributions.

2.3.1 Running experiment

We will try to lead you through your first experiment run with pmp based on use case of paper [What do multiwinner voting rules do? An experiment over the two-dimensional euclidean domain.](#) [E. Elkind, P. Faliszewski, J.-F. Laslier, P. Skowron, A. Slinko, and N. Talmon.]

We will focus on running elections in order to generate histograms like on 8th page of the paper. Please get familiar with mentioned plots.

Let's recreate setup of the experiment. Our setup will be reduced version of the one presented in paper.

```
# 10k is a lot, let's start with 100, so we can save some time
experiments_num = 100
n = 200
m = 200
k = 20
```

Provide experiment configs. (see [ExperimentConfig](#)) One configuration corresponds to a column in the final plot.

```
from pmp.experiments import ExperimentConfig, generate_uniform, generate_gauss

uniform_config = ExperimentConfig('uniform')
uniform_config.add_candidates(lambda: generate_uniform(-3, -3, 3, 3, m, 'None'))
uniform_config.add_voters(lambda: generate_uniform(-3, -3, 3, 3, n, 'None'))

gaussian_config = ExperimentConfig('gaussian')
gaussian_config.add_candidates(lambda: generate_gauss(0.0, 0.0, 1.0, m, 'None'))
gaussian_config.add_voters(lambda: generate_gauss(0.0, 0.0, 1.0, n, 'None'))

configs = [uniform_config, gaussian_config]
```

Choose scoring rules. One rule corresponds to a row in the final plot.

```
from pmp.rules import SNTV, Borda

rules = [SNTV, Borda]
```

Last step is to run the experiment itself. Please notice how looks structure of generated directory.

```
for config in configs:
    experiment = Experiment(config)
    experiment.set_generated_dir_path('paper_generated')

    for rule in rules:
        election_name = "{}-{}".format(config.id, rule.__name__)
        experiment.add_election(rule, k, election_name)

    experiment.run(n=experiments_num, save_win=True, log_on=False)
```

Since we've run our experiment and store all necessary information in *.win* files, we can proceed to next part - processing results.

2.3.2 Processing experiment results

With all necessary files generated we can move forward. For this purpose we will use helper `process_win_dir`. It accepts argument `strategy`, which allows you to do anything-your-use-case require to do with computed results. It should look like:

```
def illustrate_sample_election(candidates, winners, election, voters, preferences):
    pass

process_win_dir('paper_generated', illustrate_sample_election)
```

1. Function Strategy

At first, let's aim to prepare sample election diagrams for each combination of distribution x rule. The simplest case for Callable `strategy` is to provide a function. Since it has to be a sample election, let's decide on which of `.win` files choose. For example, let it be 2nd election for each combination.

```
import os # we will use this later
from pmp.experiments import Histogram

# note: redundant parameters voters and preferences has to included
def illustrate_sample_election(candidates, winners, election, voters, preferences):
    # strategy has to pass all not 2nd elections
    election_n = election.split('_')[1]
    if int(election_n) != 2:
        return
```

We will use `Histogram`, although it is not exactly it's destination.

```
# create histogram object with scale parameters as in paper
histogram = Histogram(-3, 3, -3, 3, (0.33, 0.33, 0.33), 20)
```

Candidates are represented as list of their attributes. Accumulate them, just in case of high density, although drawing them as simple points should be enough.

```
# candidates, just like voters are given as tuples of coordinates and string property,
# ↪ party
candidates_cords = [(x, y) for (x, y, _party) in candidates]
histogram.accumulate(candidates_cords)
```

Winners contains list of candidate ids. Retrieve their coordinates and transform them to histogram bucket resolution.

```
winners_attributes = [candidates[i] for i in winners]
# transform winners coordinates from (-3, 3) to (0, 120)
winners_cords = [((x+3)*20, (y+3)*20) for (x, y, _party) in winners_attributes]

histogram.draw_fixed_points(winners_cords, (0, 0, 255), 1)
```

Create file in some pre-created directory.

```
dir = 'sample_elections'
filename = '{}-sample.png'.format(election)
path = os.path.join(dir, filename)

histogram.save_image(path)
```

Final implementation:

```
import os # we will use this later
from pmp.experiments import process_win_dir
from pmp.experiments import Histogram

# note: redundant parameters voters and preferences has to included
def illustrate_sample_election(candidates, winners, election, voters, preferences):
    # strategy has to pass all not 2nd elections
    election_n = election.split('_')[1]
    if int(election_n) != 2:
        return

    # create histogram object with scale parameters as in paper
    histogram = Histogram(-3, 3, -3, 3, (0.33, 0.33, 0.33), 20)

    # candidates, just like voters are given as tuples of coordinates and string_
    ↪property, party
    candidates_cords = [(x, y) for (x, y, _party) in candidates]
    histogram.accumulate(candidates_cords)

    winners_attributes = [candidates[i] for i in winners]
    # transform winners coordinates from (-3, 3) to (0, 120)
    winners_cords = [(x+3)*20, (y+3)*20 for (x, y, _party) in winners_attributes]

    histogram.draw_fixed_points(winners_cords, (0, 0, 255), 1)

    dir = 'sample_elections'
    filename = '{}-sample.png'.format(election)
    path = os.path.join(dir, filename)

    histogram.save_image(path)

process_win_dir('paper_generated', illustrate_sample_election)
```

2. Callable Object Strategy

Second goal is to generate the histograms themselves. We can not use as simple function as above. All files referring one experiment and election configuration need to be aggregated to one histogram. Naive solution is to create object collecting *Histogram* object per each configuration (in simplest case, e.g. dict) and inject it's reference in function we provide as a strategy (for example, using higher-order function or decorator) and in the strategy function itself conditionally access right histogram depending on election id.

To achieve this goal and keep code clean we will conclude above logic in a class.

```
class DistributionHistograms:
    def __call__(self, candidates, winners, election, voters, preferences):
        pass

process_win_dir('paper_generated', illustrate_sample_election)
```

Let's sum up all logic required to handle single *.win* file:

- identify histogram where to add points
- retrieve winners coordinates
- accumulate points

After finish of processing:

- draw all histograms

Without breaking down into details, below class satisfy all above requirements while keeping histograms in instances state.

```
import os
from pmp.experiments import Histogram

class DistributionHistograms:
    def __init__(self):
        self.histograms = {}

    def __call__(self, candidates, winners, election, voters, preferences):
        election_id = election.split('_')[0]
        histogram = self._get_histogram(election_id)

        winners_attributes = [candidates[i] for i in winners]
        winners_cords = [(x, y) for (x, y, _party) in winners_attributes]
        histogram.accumulate(winners_cords)

    def draw_all_histograms(self):
        dir = 'distributions'

        for election in self.histograms.keys():
            histogram = self._get_histogram(election)

            filename = '{}-distribution.png'.format(election)
            path = os.path.join(dir, filename)

            histogram.save_image(path)

    def _get_histogram(self, election_id):
        # if it's first time election_id is met
        if election_id not in self.histograms.keys():
            self.histograms[election_id] = Histogram(-3, 3, -3, 3, (0, 0, 1), 20)
        return self.histograms[election_id]
```

And everything that should be called presents:

```
from pmp.experiments import process_win_dir

histograms = DistributionHistograms()

process_win_dir('paper', histograms)

histograms.draw_all_histograms()
```

Last step is to use ‘pmp’ worthily with your use case :)

2.4 Experiment Module

Experiment is a convenient way to configure and run your computations.

2.4.1 Experiment

class pmp.experiments.**Experiment** (*config=None*)

Experiment to run

add_election (*rule, k, id*)

Parameters

- **rule** ([Rule](#)) – Scoring rule for added election
- **k** (*int*) – Size of the winning committee for added election
- **id** (*str*) – Text id for the election. Builds up first part of output filenames

For multi-rule experiments. It's preferred way of defining elections in experiments.

get_generated_dir_path ()

Returns Path to the root directory where files are generated

Return type *str*

run (*visualization=False, n=1, save_win=False, save_in=False, save_out=False, log_on=True, elect_configs=None, split_dirs=True*)

Parameters

- **visualization** (*bool*) –
- **n** (*int*) –
- **save_win** (*bool*) –
- **save_in** (*bool*) –
- **save_out** (*bool*) –
- **log_on** (*bool*) –
- **elect_configs** (*List[ElectionConfig]*) – Election configs. If given, experiment ignores it's one-rule configuration
- **split_dirs** (*bool*) – When True create separate directory for each election related files

Run experiment. Experiment runs elections configured in following precedence: * from *elect_configs* parameter, if present * set up from *election_configurations* added by *add_election*, if added * set up from *set_election*

set_election (*rule, k*)

Parameters

- **rule** ([Rule](#)) – Scoring rule being set
- **k** (*int*) – Size of the committee being set

[Deprecated] Set election parameters the rule and the size of the committee. Overrides previous experiment setup. Only for setting up one-rule experiments.

set_generated_dir_path (*dir_path*)

Parameters **dir_path** (*str*) – Path to the root directory where files are generated

set_inout_filename (*name*)

Parameters **name** (*str*) – Inout filename

Set filename of files containing generated candidates and voters. Overrides previous value. Only for setting up one-rule experiments.

set_result_filename (*name*)

Parameters *name* (*str*) – Result filename

[Deprecated] Set result name. It builds up first part of names of all files being generated during experiment. Overrides previous value. Only for setting up one-rule experiments.

2.4.2 ExperimentConfig

class pmp.experiments.**ExperimentConfig** (*id=""*)

Store candidates and voters configuration. Above properties can be stored as static values or generating functions, so different setup is generated each time.

get_candidates ()

Returns Copy of candidates

Return type List

impartial (*m, n*)

Candidates in impartial only as a list of consecutive integers starting from 0

set_candidates (*list_of_candidates*)

Parameters *list_of_candidates* (*List*) – List of candidates

2.4.3 ElectionConfig

class pmp.experiments.**ElectionConfig** (*rule, k, id*)

Election configuration used by an Experiment

2.4.4 Histogram

class pmp.experiments.**Histogram** (*minx, maxx, miny, maxy, opacity_mask=(1.0, 1.0, 1.0),
ppu=None, W=256, H=256*)

Two-dimensional histogram.

accumulate (*points*)

Parameters *points* (*List [Tuple [float, float]]*) – List of points to be accumulated into histogram.

draw_fixed_points (*points, color, size=0*)

Parameters

- **points** (*List [Tuple [float, float]]*) – List of points to be draw.
- **color** (*Tuple [int, int, int]*) – RGB color of points.
- **size** (*int*) – Size of points. When equal 0 draw single pixel.

Draw fixed points over the main histogram layer.

save_image (*path*)

Saves the histogram image. Fixed points are drawn above the histogram itself.

2.4.5 Generating Functions

`pmp.experiments.generating_functions.random()` → x in the interval $[0, 1)$.

2.4.6 Helpers

`pmp.experiments.helpers.process_win_dir(path, strategy)`

Parameters

- **path** (*str*) – path of processed directory
- **strategy** (*Callable[List, List, List, List, str]*) – Run with {candidates, voters, preferences, winners, election}

Helper for processing experiment-generated directories. Visits all election directories stored in path directory. After loading .win file runs strategy with candidates, voters, preferences, winners, election args. Election is a string id.

2.5 Rules Module

Module containing classes representing all supported voting rules. For a general rule interface see Rule class.

`pmp.rules.rules_list = ['Bloc', 'Borda', 'ChamberlinCourant', 'PAV', 'Rule', 'SNTV', 'WeakPareto']`

Variable containing names of all provided rules. Initialized during import.

2.5.1 Rule

class `pmp.rules.Rule` (*tie_break=<function any_winner>*)

Scoring rule class.

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill profile.scores member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Returns Committee winning under given rule

Return type List[int]

2.5.2 WeaklySeparable

class pmp.rules.**WeaklySeparable** (*weights=None, tie_break=<function any_winner>*)

Weakly Separable scoring rule This is base class for all weakly separable scoring rules

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill profile.scores member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Returns Committee winning under given rule

Return type List[int]

get_committees (*k, candidates_with_score*)

Parameters

- **k** (*Number*) – Size of committee
- **candidates_with_score** (*Dict[Number, List[Number]]*) – Dictionary with lists of candidates who achieved given score

Returns List[List]

Find all winning committees

2.5.3 SNTV

class pmp.rules.**SNTV** (*tie_break=<function any_winner>*)

Single non-transferable vote scoring rule

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find

- **profile** (*Profile*) – Preferences profile object

If it is possible, fill `profile.scores` member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Returns Committee winning under given rule

Return type List[int]

get_committees (*k, candidates_with_score*)

Parameters

- **k** (*Number*) – Size of committee
- **candidates_with_score** (*Dict[Number, List[Number]]*) – Dictionary with lists of candidates who achieved given score

Returns List[List]

Find all winning committees

2.5.4 Bloc

class `pmp.rules.Bloc` (*weights=None, tie_break=<function any_winner>*)

Bloc vote scoring rule

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill `profile.scores` member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Returns Committee winning under given rule

Return type List[int]

get_committees (*k, candidates_with_score*)

Parameters

- **k** (*Number*) – Size of committee
- **candidates_with_score** (*Dict[Number, List[Number]]*) – Dictionary with lists of candidates who achieved given score

Returns List[List]

Find all winning committees

2.5.5 Borda

class pmp.rules.**Borda** (*weights=None, tie_break=<function any_winner>*)

Borda vote scoring rule

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill profile.scores member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Returns Committee winning under given rule

Return type List[int]

get_committees (*k, candidates_with_score*)

Parameters

- **k** (*Number*) – Size of committee

- **candidates_with_score** (*Dict[Number, List[Number]]*) – Dictionary with lists of candidates who achieved given score

Returns List[List]

Find all winning committees

2.5.6 ChamberlinCourant

class pmp.rules.**ChamberlinCourant** (*weights=None*)

Chamberlin-Courant vote scoring rule

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill profile.scores member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates
- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile, method=None*)

Parameters

- **k** (*String*) – size of committee to find
- **profile** (*Profile*) – preferences profile object
- **method** (*String*) – Method of computation - Bruteforce/ILP

Returns committee winning under given rule

2.5.7 PAV

class pmp.rules.**PAV** (*alpha=None*)

Proportional Approval Voting scoring rule.

compute_candidate_scores (*k, profile*)

Parameters

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

If it is possible, fill profile.scores member dictionary with scores of all committees

compute_committee_score (*committee, k, profile*)

Parameters

- **committee** (*List*) – List of candidates

- **k** (*str*) – Size of committee to find
- **profile** (*Profile*) – Preferences profile object

Find score assigned to given committee

find_committee (*k, profile, method=None*)

Parameters

- **k** (*String*) – size of committee to find
- **profile** (*Profile*) – preferences profile object
- **method** (*String*) – Method of computation - Bruteforce/ILP

Returns committee winning under given rule

2.6 Preferences Module

2.6.1 Preference

class pmp.preferences.**Preference** (*order=[], weights=None*)

Single voter preference.

is_valid (*num_cand*)

Parameters **num_cand** (*Number*) – Number of candidates

Returns Boolean

Check if Preference is valid under profile with given candidates number

2.6.2 Profile

class pmp.preferences.**Profile** (*candidates=None, preferences=None*)

Profile of voters' preferences

add_preference (*preference*)

Parameters **preference** (*Preference*) – Added preference

Add single preference to the profile. Works only if preference is valid.

add_preferences (*preferences*)

Parameters **preferences** (*List (Preference)*) – Added preferences

Add preferences to the profile. Adds only the valid ones.

clean_scores ()

Clear candidates scores cached in profile

2.6.3 Ordinal

class pmp.preferences.**Ordinal** (*order, weights=None*)

Ordinal preference profile.

better_candidates_count (*candidate*)

Candidate's rank - from how many candidates the candidate is worse.

compare_candidates (*candidate_a, candidate_b*)

Returns the better one.

is_valid (*num_cand*)

Parameters **num_cand** (*Number*) – Number of candidates

Returns Boolean

Check if Preference is valid under profile with given candidates number

worse_candidates_count (*candidate*)

Candidate's rank - from how many candidates the candidate is better.

2.6.4 Approval

class pmp.preferences.**Approval** (*approved*)

Approval preference profile.

is_valid (*num_cand*)

Parameters **num_cand** (*Number*) – Number of candidates

Returns Boolean

Check if Preference is valid under profile with given candidates number

2.7 Integer linear programming

PMP allows you to use Cplex and Gurobi as ILP backends.

2.7.1 Gurobi instalation tips

Visit [official Gurobi site](#), get proper license and download installator dedicated for your operating system.

In order to install required python interface gurobipy run it's setup script:

```
cd GUROBI_INSTALLATION_DIR
python setup.py install
```

Please note it is important to run this script inside gurobi's directory.

2.7.2 Cplex instalation tips

Visit [official IBM Cplex site](#), get proper license and download installator dedicated for your operating system.

Now install Cplex python interface.

```
pip install cplex
```

Please note: **If you are using virtualenv, you may need to recreate your venv after installing this dependencies.**

2.8 Using pmp

You can use pmp twofold. If core functionalities satisfy your needs, simply install pmp with pip. Use it as other python modules. For more details please check instructions.

2.8.1 Developing pmp

During development of your projects you may need to extend or modify pmp code.

We strongly encourage you to use virtualenv during development.

Install virtualenv and test installation.

```
pip install virtualenv
virtualenv --version
```

Clone project and enter it's directory, then create virtual environment for pmp.

```
git clone https://github.com/Koozco/pmp.git
cd pmp
virtualenv pmp_venv
```

In order to use this venv please remember to use: `source pmp_venv/bin/activate`

And for deactivation `deactivate`

Install pmp in `editable` mode, so all changes you make in code will be reflected in module's behaviour.

```
pip install -e .
```

Virtual env will allow you to separate this installation and it's dependencies from your globally installed python.

Enjoy!

2.8.2 Working with docs

As a documentation tool we use Sphinx, however we stay open for any ideas suggestions, which may improve our current setup. Main file format of Sphinx is *reStructuredText*, but it is also possible to render *Markdown* in `.rst` files.

If you want to extend documentation, a place to start is `docs/source`, where are stored all sources. After adding sources, compile them with Sphinx Makefile.

```
cd docs
make clean html
```

Check the result in `_build/html/index.html`

Note: Depending on your OS it may be necessary not only to install sphinx by pip, but also by OS package manager.

2.9 Examples

While getting started or looking for some feature please feel free to check out [examples](#).

2.10 About

The purpose of Python Multiwinner Package project is to deliver a library of algorithms computing results of multi-winner elections, where elections mean any election where voters specify their preferences regarding candidates that they could choose. Motivation of the library is to create a mean which will ease sharing and exchanging results of work connected with this domain.

Python Multiwinner Package development started in April 2018 at AGH University of Science and Technology at the initiative of Piotr Faliszewski. Up to December 2018 pmp growth was driven as engineering thesis of two AGH students, Katarzyna Banaszak and Bartosz Kusek. Now package is still being expanded. Please feel free to contact as about collaboration.

PYTHON MODULE INDEX

p

`pmp.experiments.generating_functions`,
 [11](#)
`pmp.experiments.helpers`, [11](#)
`pmp.preferences`, [16](#)
`pmp.rules`, [11](#)

A

accumulate() (*pmp.experiments.Histogram method*), 10
 add_election() (*pmp.experiments.Experiment method*), 9
 add_preference() (*pmp.preferences.Profile method*), 16
 add_preferences() (*pmp.preferences.Profile method*), 16
 Approval (*class in pmp.preferences*), 17

B

better_candidates_count() (*pmp.preferences.Ordinal method*), 16
 Bloc (*class in pmp.rules*), 13
 Borda (*class in pmp.rules*), 14

C

ChamberlinCourant (*class in pmp.rules*), 15
 clean_scores() (*pmp.preferences.Profile method*), 16
 compare_candidates() (*pmp.preferences.Ordinal method*), 16
 compute_candidate_scores() (*pmp.rules.Bloc method*), 13
 compute_candidate_scores() (*pmp.rules.Borda method*), 14
 compute_candidate_scores() (*pmp.rules.ChamberlinCourant method*), 15
 compute_candidate_scores() (*pmp.rules.PAV method*), 15
 compute_candidate_scores() (*pmp.rules.Rule method*), 11
 compute_candidate_scores() (*pmp.rules.SNTV method*), 12
 compute_candidate_scores() (*pmp.rules.WeaklySeparable method*), 12
 compute_committee_score() (*pmp.rules.Bloc method*), 13
 compute_committee_score() (*pmp.rules.Borda method*), 14

compute_committee_score() (*pmp.rules.ChamberlinCourant method*), 15
 compute_committee_score() (*pmp.rules.PAV method*), 15
 compute_committee_score() (*pmp.rules.Rule method*), 11
 compute_committee_score() (*pmp.rules.SNTV method*), 13
 compute_committee_score() (*pmp.rules.WeaklySeparable method*), 12

D

draw_fixed_points() (*pmp.experiments.Histogram method*), 10

E

ElectionConfig (*class in pmp.experiments*), 10
 Experiment (*class in pmp.experiments*), 9
 ExperimentConfig (*class in pmp.experiments*), 10

F

find_committee() (*pmp.rules.Bloc method*), 13
 find_committee() (*pmp.rules.Borda method*), 14
 find_committee() (*pmp.rules.ChamberlinCourant method*), 15
 find_committee() (*pmp.rules.PAV method*), 16
 find_committee() (*pmp.rules.Rule method*), 11
 find_committee() (*pmp.rules.SNTV method*), 13
 find_committee() (*pmp.rules.WeaklySeparable method*), 12

G

get_candidates() (*pmp.experiments.ExperimentConfig method*), 10
 get_committees() (*pmp.rules.Bloc method*), 14
 get_committees() (*pmp.rules.Borda method*), 14
 get_committees() (*pmp.rules.SNTV method*), 13
 get_committees() (*pmp.rules.WeaklySeparable method*), 12
 get_generated_dir_path() (*pmp.experiments.Experiment method*), 9

H

Histogram (*class in pmp.experiments*), 10

I

impartial() (*pmp.experiments.ExperimentConfig method*), 10

is_valid() (*pmp.preferences.Approval method*), 17

is_valid() (*pmp.preferences.Ordinal method*), 17

is_valid() (*pmp.preferences.Preference method*), 16

O

Ordinal (*class in pmp.preferences*), 16

P

PAV (*class in pmp.rules*), 15

pmp.experiments.generating_functions
(*module*), 11

pmp.experiments.helpers (*module*), 11

pmp.preferences (*module*), 16

pmp.rules (*module*), 11

Preference (*class in pmp.preferences*), 16

process_win_dir() (in *module*
pmp.experiments.helpers), 11

Profile (*class in pmp.preferences*), 16

R

random() (in *module*
pmp.experiments.generating_functions),
11

Rule (*class in pmp.rules*), 11

rules_list (in *module pmp.rules*), 11

run() (*pmp.experiments.Experiment method*), 9

S

save_image() (*pmp.experiments.Histogram method*),
10

set_candidates() (*pmp.experiments.ExperimentConfig method*), 10

set_election() (*pmp.experiments.Experiment method*), 9

set_generated_dir_path()
(*pmp.experiments.Experiment method*), 9

set_inout_filename()
(*pmp.experiments.Experiment method*), 9

set_result_filename()
(*pmp.experiments.Experiment method*), 10

SNTV (*class in pmp.rules*), 12

W

WeaklySeparable (*class in pmp.rules*), 12

worse_candidates_count()
(*pmp.preferences.Ordinal method*), 17