thompsons_v Documentation

Author

Aug 27, 2018

Contents

CHAPTER 1

Piecewise-linear maps

class plmaps.plmap.**PLMap**(*domain*, *range*)

Let I and J be closed intervals. We define $\mathrm{PL}_{S,G}(I, J)$ to be the set of functions $\alpha\colon I \to J$ with the following properties.

- α is an orientation-preserving bijection.
- α is piecewise-linear, with finitely many linear segments.
- The coordinates of each breakpoint of α belong to S.
- The gradients of each linear section of α belong to G.

We do not require that the endpoints of I or J belong to S.

This class represents functions of this form where $S = \mathbb{Q}$ and $G = \mathbb{Q}_{>0}$.

Variables

- ***domain*** – see *__init__()*.
- ***range*** – see *__init__()*.
- ***gradients*** – The ith entry of this list is the gradient of the ith linear segment.

__init__(*domain*, *range*)

Create a new `PLMap` given the breakpoints' coordinates. Breakpoint lists are normalised in memory: redudant breakpoints (where the the gradient does not change) are removed. Coordinates are provided via two lists *domain* and *range* of `Fraction` s.

Raises

- **ValueError** – if `len(domain) != len(range)`.
- **ValueError** – if `len(domain) < 2`.
- **ValueError** – if *domain* or *range* are not increasing sequences.
- **ValueError** – if *domain* or *range* contain invalid breakpoints. This requirement does **not** apply the first or last element of each list.
- **ValueError** – if *domain* or *range* describe linear segments with invalid gradients.

```
>>> PLMap([0, 1], [0, Fraction(1, 2), 1])
Traceback (most recent call last):
...
ValueError: Domain and range lengths differ
>>> PLMap([], [])
Traceback (most recent call last):
...
ValueError: Domain must be defined by at least two points
>>> PLMap([0, 0], [1, 0])
Traceback (most recent call last):
...
ValueError: domain is not an increasing sequence
>>> PLMap([0, 1], [1, 0])
Traceback (most recent call last):
...
ValueError: range is not an increasing sequence
```

domain

range

gradients

classmethod identity(*t0*, *t1*)

__iter__()
Iterating over a `PLMap` yields its breakpoints.

classmethod from_aut(*aut*)
Creates a new PLMap using a `Homomorphism`.

classmethod from_stream(*stream*)

save_to_file(*filename*)

__mul__(*other*)
Postcompose on the right.

image(x)
Where does the current PLMap send the point x?

Raises **ValueError** – if x is not in the current map's domain.

inverse_image(y)
Where is mapped by the current PLMap to the point y?

Raises **ValueError** – if y is not in the current map's range.

rgradient_at(x)

is_identity()

dump(*short=True*)

format_pl_segments(***kwargs*)

tikz_path()

commutes(*other*)

centralise_in_F()

restriction(*t0*, *t1*)
Produce a copy of the current PLMap restricted to the interval $[t_0, t_1]$.

Parameters **target** (*iterable*) – A sequence of at least two integers. The first and last entries are the start and end of the interval onto which we restrict.

Raises

- **ValueError** – if `t0 >= t1`.
- **ValueError** – if *t0* and *t1* do not describe a subinterval of `self.domain`.

restriction_of_range(*t0*, *t1*, *raw=False*)

Produce a copy of the current PLMap restricted to the interval $[t_0, t_1]$.

Parameters **target** (*iterable*) – A sequence of at least two integers. The first and last entries are the start and end of the interval onto which we restrict.

Raises

- **ValueError** – if `t0 >= t1`.
- **ValueError** – if *t0* and *t1* do not describe a subinterval of `self.domain`.

is_permutation()

fixed_points(*raw=False*)

fixed_point_boundary()

is_one_bump()

one_bump_test_conjugate_with(*other*, *initial_gradient*, *verbose=False*)

one_bump_linearity_boxes(*other*, *initial_gradient*)

class plmaps.plmap.**PL2**(*domain*, *range*)

PL_2 is shorthand for the set of `PLMap` s with $S = \mathbb{Z}[1/2]$ and $G = 2^{\mathbb{Z}}$. As *noted above <plmaps.plmaps.PLMap>*, we don't insist that the endpoints of the domain and range lie in S: we're working with what my thesis calls $\mathrm{PL}_2^{\mathrm{rest}}$ rather than $\mathrm{PL}_2^{\mathrm{flat}}$.

```
>>> PL2([0, Fraction(1, 3)], [1, Fraction(5, 3)])
<PL2: [0, 1/3] -> [1, 5/3]>
>>> PL2([0, Fraction(1, 2), 1], [1, Fraction(5, 3), 2])
Traceback (most recent call last):
...
ValueError: range contains an invalid breakpoint
>>> PL2([0, 1], [0, 3])
Traceback (most recent call last):
...
ValueError: Invalid gradient
```

one_bump_test_conjugate(*other*)

one_bump_cent_gen(*verbose=False*)

If the current PLMap is a one-bump function DoD, produce an element which generates its centraliser in $\mathrm{PL}(D)$. The generator's initial gradient will be above 1 if and only if the the current PLMap's initial gradient is above 1.

Raises **ValueError** – if the current PLMap is not a one-bump function.

Python Module Index

p

Index

Symbols

C

D

F

G

I

O

P

R

S

T