SketchResponse Documentation

Release 1.0

Martin Segado

Nov 06, 2016

CONTENTS

This is the documentation for the SketchResponse Grader-Library API. The modules and methods documented here are the subset of modules and methods that are directly used to implement a SketchResponse grading scripts.

Contents:

CHAPTER

ONE

GRADER_LIB PACKAGE

1.1 Submodules

1.2 grader_lib.Asymptote module

class grader_lib.Asymptote.**Asymptotes**(*info*, *tolerance={}*)
Bases: grader_lib.Gradeable.Gradeable

Asymptote.

Note: Asymptotes is a generic class. You must instantiate either the VerticalAsymptotes or the HorizontalAsymptotes class to use the grading functions below.

---

closest_asym_to_value(*v*)
Return the absolute distance between v and the closest asymptote and the x or y axis value of that asymptote.

Parameters **v** – a value in the range of the x or y axis.

Returns

minDistance: the absolute difference between v and the asymptote, or float('inf') if no asymptote exists.

closestAsym: the value of the closest asymptote to the value v, or None if no asymptote exists.

Return type float, float

get_asym_at_value(*v*, *tolerance=None*)
Return the asymptote at the value v, or None.

Parameters

- **v** – a value in the range of the x or y axis.
- **tolerance(default** – None): pixel distance tolerance, if None is given 'asym_distance' constant is used.

Returns

the value of an asymptote that is within tolerances of the value v, or None if no such asymptote exists.

Return type float

get_number_of_asyms()
Return the number of asymptotes declared in the function.

Returns the number of asymptotes declared in the function.

Return type int

has_asym_at_value(*v*, *tolerance=None*)
Return whether an asymtote is declared at the given value.

Parameters

- **v** – a value in the range of the x or y axis.
- **tolerance(default** – None): pixel distance tolerance, if None is given 'asym_distance' constant is used.

Returns true if there is an asymptote declared within tolerances of the value v, or false otherwise.

Return type bool

class grader_lib.Asymptote.**HorizontalAsymptotes**(*info*)
Bases: *grader_lib.Asymptote.Asymptotes*

Horizontal Asymptote.

Note: Use this class to interact with any horizontal asymptotes in the function you are grading.

class grader_lib.Asymptote.**VerticalAsymptotes**(*info*)
Bases: *grader_lib.Asymptote.Asymptotes*

Vertical Asymptote.

Note: Use this class to interact with any vertical asymptotes in the function you are grading.

1.3 grader_lib.LineSegment module

class grader_lib.LineSegment.**LineSegment**(*point1*, *point2*)
A line segment wrapper class. Contains two Points defining the start point and the end point of the segment.

class grader_lib.LineSegment.**LineSegments**(*info*, *tolerance={}*)
Bases: grader_lib.Gradeable.Gradeable

Line Segments.

check_segment_endpoint(*segment*, *point*, *tolerance=None*)
Return whether the segment has its end point at the point (x,y).

Parameters

- **segment** – the line segment to check
- **point** – a list [x, y] defining the point to check
- **tolerance** – the square of the distance in pixels

Returns true if the segment's end point is at (x,y) within tolerances, otherwise false.

Return type bool

check_segment_endpoints(*segment*, *points*, *tolerance=None*)

Return whether the segment's start and end points are both in the list of points.

Parameters

- **segment** – the line segment to check
- **points** – a list of [x, y] coordiates
- **tolerance** – the square of the distance in pixels

Returns true if the segments start and end points are in points, otherwise false.

Return type bool

check_segment_startpoint(*segment*, *point*, *tolerance=None*)

Return whether the segment has its start point at the point (x,y).

Parameters

- **segment** – the line segment to check
- **point** – a list [x, y] defining the point to check
- **tolerance** – the square of the distance in pixels

Returns true if the segment's start point is at (x,y) within tolerances, otherwise false.

Return type bool

does_exist_between(*xmin*, *xmax*)

Return whether the function has values defined in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.

Returns true if at least one line segment overlaps with the range xmin to xmax within tolerances, otherwise false.

Return type bool

does_not_exist_between(*xmin*, *xmax*)

Return whether the function has no values defined in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.

Returns true if no line segments overlap with the range (xmin, xmax) within tolerances, otherwise false.

Return type bool

get_number_of_segments()

Return the number of line segments in this grader module.

Returns the number of line segments in this grader module

Return type int

get_segment_angle(*segment*)
Return the angle of the line segment in radians.

Parameters **segment** – the line segment to check

Returns the angle of the line segment in radians

Return type float

get_segment_length(*segment*)
Return whether the length of the line segment.

Parameters **segment** – the line segment to check

Returns the length of the line segment

Return type int

get_segments_at(*point=False*, *x=False*, *y=False*, *distTolerance=None*, *squareDistTolerance=None*)
Return a list of line segments declared at the given value.

Parameters

- **point(default** – False): a Point instance at the value of interest.
- **x(default** – False): the x coordinate of interest.
- **y(default** – False): the y coordinate of interest.
- **distTolerance(default** – None): the pixel distance tolerance if only the x or y coordinate is given. If None default constant 'line_distance' is used.
- **squareDistTolerance(default** – None): the square pixel distance tolerance if point, or x and y are given. If None, default constant 'line_distance_squared' is used.

Note:

There are four use cases:

1. point not False: use the Point instance as the target to locate segments, returning a list of segments that pass through the Point.
2. x and y not False: use (x, y) as the target to locate segments, returning a list of segments that pass through the point (x, y).
3. x not False: use only the x coordinate to locate segments, returning a list of segments that pass through given x value.
4. y not False: use only the y coordinate to locate segments, returning a list of segments that pass through the given y value.

Returns a list of the line segments within tolerances of the given position arguments, or None

Return type list

has_angle_t_at_x(*t*, *x*, *tolerance=None*)

Return whether the line segment at position x has an angle of t wrt the x axis.

Parameters

- **t** – the angle in radians
- **x** – the position on the x-axis to test against.
- **tolerance** – the angle tolerance in degrees

Returns true if the function at value x has angle t within tolerances, otherwise false.

Return type bool

has_segments_at(*point=False*, *x=False*, *y=False*, *distTolerance=None*, *squareDistTolerance=None*)

Return a list of line segments declared at the given value.

Parameters

- **point(default** – False): a Point instance at the value of interest.
- **x(default** – False): the x coordinate of interest.
- **y(default** – False): the y coordinate of interest.
- **distTolerance(default** – None): the pixel distance tolerance if only the x or y coordinate is given. If None default constant 'line_distance' is used.
- **squareDistTolerance(default** – None): the square pixel distance tolerance if point, or x and y are given. If None, default constant 'line_distance_squared' is used.

Note:

There are four use cases:

1. point not False: use the Point instance as the target to locate segments, returning a list of segments that pass through the Point.
2. x and y not False: use (x, y) as the target to locate segments, returning a list of segments that pass through the point (x, y).
3. x not False: use only the x coordinate to locate segments, returning a list of segments that pass through given x value.
4. y not False: use only the y coordinate to locate segments, returning a list of segments that pass through the given y value.

Returns true if there is at least one line segment within tolerance of the given position, otherwise false.

Return type bool

has_slope_m_at_x(*m*, *x*, *tolerance=None*)

Return whether the function has slope m at the value x.

Parameters

- **m** – the slope value to test against.
- **x** – the position on the x-axis to test against.
- **tolerance** – the angle tolerance in degrees

Returns true if the function at value x has slope m within tolerances, otherwise false.

Return type bool

1.4 grader_lib.GradeableFunction module

class grader_lib.GradeableFunction.**GradeableFunction**(*gradeable*, *tolerance={}*)
Bases: grader_lib.MultipleSplinesFunction.MultipleSplinesFunction

GradeableFunction.

closest_point_to_point(*point*)
Return the square pixel distance to the closest point and a Point instance.

Parameters **point** – a Point instance

Returns

minDistanceSquared: the square of the pixel distance between point and the closest point, or float('inf') if no point exists.

minPoint: the closest Point to x, or None if no point exists.

Return type float, Point

closest_point_to_x(*x*)
Return the distance to the closest point and a Point instance.

Parameters **x** – a value in the range of the x axis.

Returns

minDistance: the absolute distance between x and the point, or float('inf') if no point exists.

minPoint: the closest Point to x, or None if no point exists.

Return type float, Point

does_exist_between(*xmin*, *xmax*, *end_tolerance=70*, *gap_tolerance=40*)
Return whether the function has values defined in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.
- **end_tolerance(default** – 70): the pixel tolerance for the endpoints of the range xmin to xmax.
- **gap_tolerance(default** – 40): the pixel tolerance for gaps in the function in the range xmin to xmax.

Returns true if the function is defined within tolerances over the range xmin to xmax, otherwise false.

Return type bool

does_not_exist_between(*xmin*, *xmax*)
Return whether the function has no values defined in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.

Returns true if the function has no values within tolerances in the range xmin to xmax, otherwise false.

Return type bool

get_horizontal_line_crossings(*yval*)
Return a list of the values where the function crosses the horizontal line y=yval.

Parameters **yval** – the y-axis value of the horizontal line.

Returns the list of values where the function crosses the line y=yval.

Return type [float]

get_number_of_points()
Return the number of points declared in the function.

get_point_at(*point=False*, *x=False*, *y=False*, *distTolerance=None*, *squareDistTolerance=None*)
Return a reference to the Point declared at the given value.

Parameters

- **point(default** – False): a Point instance at the value of interest.
- **x(default** – False): the x coordinate of interest.
- **y(default** – False): the y coordinate of interest.
- **distTolerance(default** – None): the pixel distance tolerance if only the x coordinate is given. If None default constant 'point_distance' is used.
- **squareDistTolerance(default** – None): the square pixel distance tolerance if point, or x and y are given. If None, default constant 'point_distance_squared' is used.

Note:

There are three use cases:

1. point not False: use the Point instance as the target to locate a point in the function.
2. x and y not False: use (x, y) as the target to locate a point in the function.
3. x not False: use only the x coordinate to locate a point in the function, returning the first Point with the given x value.

Returns the first Point instance within tolerances of the given arguments, or None

Return type *Point*

get_vertical_line_crossings(*xval*)
Return a list of the values where the function crosses the horizontal line x=xval.

Parameters **xval** – the x-axis value of the vertical line.

Returns the list of values where the function crosses the line x=xval.

Return type [float]

has_constant_value_y_between(*y*, *xmin*, *xmax*)
Return whether the function has a constant value y over the range xmin to xmax.

Parameters

- **y** – the constant value to check.
- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.

Returns true if the function has the value y at both xmin and xmax and the function is straight in the range xmin to xmax, otherwise false.

Return type bool

has_max_at(*x*, *delta=False*, *xmin=False*, *xmax=False*)
Return if the function has a local maximum at the value x.

Parameters

- **x** – the x-axis value to test.
- **delta(default** – False): the delta value to sample on either side of x (not setting it uses a default value).
- **xmin(default** – False): the position of the value left of x to compare (not setting it uses the value x - delta).
- **xmax(default** – False): the position of the value right of x to compare (not setting it uses the value x + delta).

Returns true if the value of the function at x is greater than both the values at xmin and xmax, otherwise false.

Return type bool

has_min_at(*x*, *delta=False*, *xmin=False*, *xmax=False*)
Return if the function has a local minimum at the value x.

Parameters

- **x** – the x-axis value to test.
- **delta(default** – False): the delta value to sample on either side of x (not setting it uses a default value).
- **xmin(default** – False): the position of the value left of x to compare (not setting it uses the value x - delta).
- **xmax(default** – False): the position of the value right of x to compare (not setting it uses the value x + delta).

Returns true if the value of the function at x is less than both the values at xmin and xmax, otherwise false.

Return type bool

has_negative_curvature_between(*xmin*, *xmax*, *numSegments=5*, *failureTolerance=None*)
Return whether the function has negative curvature in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.
- **numSegments(default** – 5): the number of segments to divide the function into to individually test for negative curvature.
- **failureTolerance(default** – None): the number of segments that can fail the negative curvature test before test failure. If None given uses default constant 'curve_failure'.

Returns true if all segments, in the range xmin to xmax, have negative curvature within tolerances, otherwise false.

Return type bool

has_point_at(*point=False*, *x=False*, *y=False*, *distTolerance=None*, *squareDistTolerance=None*)
Return whether a point is declared at the given value.

Parameters

- **point(default** – False): a Point instance at the value of interest.
- **x(default** – False): the x coordinate of interest.
- **y(default** – False): the y coordinate of interest.
- **distTolerance(default** – None): the pixel distance tolerance if only the x coordinate is given. If None default constant 'point_distance' is used.
- **squareDistTolerance(default** – None): the square pixel distance tolerance if point, or x and y are given. If None, default constant 'point_distance_squared' is used.

Note:

There are three use cases:

1. point not False: use the Point instance as the target to locate a point in the function.
2. x and y not False: use (x, y) as the target to locate a point in the function.
3. x not False: use only the x coordinate to locate a point in the function, returning the first Point with the given x value.

Returns true if there is a Point declared within tolerances of the given argument(s), false otherwise.

Return type bool

has_positive_curvature_between(*xmin*, *xmax*, *numSegments=5*, *failureTolerance=None*)
Return whether the function has positive curvature in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.
- **numSegments(default** – 5): the number of segments to divide the function into to individually test for positive curvature.
- **failureTolerance(default** – None): the number of segments that can fail the positive curvature test before test failure. If None given uses default constant 'curve_failure'.

Returns true if all segments, in the range xmin to xmax, have positive curvature within tolerances, otherwise false.

Return type bool

has_slope_m_at_x(*m*, *x*, *tolerance=None*)
Return whether the function has slope m at the value x.

Parameters

- **m** – the slope value to test against.
- **x** – the position on the x-axis to test against.

- **tolerance(default** – None): angle tolerance in degrees. If None given uses default constant 'angle'.

Returns true if the function at value x has slope m within tolerances, otherwise false.

Return type bool

has_value_y_at_x(*y*, *x*, *yTolerance=None*, *xTolerance=None*)
Return whether the function has the value y at x.

Parameters

- **y** – the target y value.
- **x** – the x value.
- **yTolerance(default** – None): the y-axis pixel distance within which the function value is accepted.
- **xTolerance(default** – None): the x-axis pixel distance within which the function value is accepted.

Returns true if the function value at x is y within tolerances, otherwise false

Return type bool

is_always_decreasing(*failureTolerance=None*)
Return whether the function is decreasing over its entire domain.

Returns true if the function is decreasing within tolerances over the entire domain, otherwise false.

Return type bool

is_always_increasing(*failureTolerance=None*)
Return whether the function is increasing over its entire domain.

Returns true if the function is increasing within tolerances over the entire domain, otherwise false.

Return type bool

is_decreasing_between(*xmin*, *xmax*, *numPoints=10*, *failureTolerance=None*)
Return whether the function is decreasing in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.
- **numPoints(default** – 10): the number of points to test along the range.
- **failureTolerance(default** – None): the number of pairwise point decrease comparisons that can fail before the test fails. If None give, default constant 'inc_dec_failure' is used.

Returns true if all sequential pairs of points have decreasing values within tolerances for the range xmin to xmax, otherwise false.

Return type bool

is_greater_than_y_between(*y*, *xmin*, *xmax*, *tolerance=None*)
Return whether function is always greater than y in the range xmin to xmax.

Parameters

- **y** – the target y value.
- **xmin** – the minimum x range value.
- **xmax** – the maximum x range value.
- **tolerance(default** – None): pixel distance tolerance. If None given uses default constant 'comparison'.

Returns true if the minimum value of the function in the range (xmin,xmax) is greater than y within tolerances, otherwise false.

Return type bool

is_increasing_between(*xmin*, *xmax*, *numPoints=10*, *failureTolerance=None*)

Return whether the function is increasing in the range xmin to xmax.

Parameters

- **xmin** – the minimum x-axis value of the range to test.
- **xmax** – the maximum x-axis value of the range to test.
- **numPoints(default** – 10): the number of points to test along the range.
- **failureTolerance(default** – None): the number of pairwise point increase comparisons that can fail before the test fails. If None give, default constant 'inc_dec_failure' is used.

Returns true if all sequential pairs of points have increasing values within tolerances for the range xmin to xmax, otherwise false.

Return type bool

is_less_than_y_between(*y*, *xmin*, *xmax*, *tolerance=None*)

Return whether function is always less than y in the range xmin to xmax.

Parameters

- **y** – the target y value.
- **xmin** – the minimum x range value.
- **xmax** – the maximum x range value.
- **tolerance(default** – None): pixel distance tolerance. If None given uses default constant 'comparison'.

Returns true if the maximum value of the function in the range (xmin,xmax) is less than y within tolerances, otherwise false.

Return type bool

is_straight()

Return whether the function is straight over its entire domain.

Returns true if the function is straight within tolerances over the entire domain, otherwise false.

Return type bool

is_straight_between(*xmin*, *xmax*)

Return whether the function is straight within the range xmin to xmax. An alternate approximate implementation until we sort out some issues above

Parameters

- **xmin** – the minimum x-axis value of the range to check.

- **xmax** – the maximum x-axis value of the range to check.

Returns true if the function is straight within tolerances between xmin and xmax, otherwise false

Return type bool

is_zero_at_x_equals_zero(*yTolerance=None*, *xTolerance=None*)

Return whether the function is zero at x equals zero.

Parameters

- **yTolerance(default** – None): the y-axis pixel distance within which the function value is accepted.
- **xTolerance(default** – None): the x-axis pixel distance within which the function value is accepted.

Returns true if the function value at x equals zero is zero within tolerances, otherwise false

Return type bool

1.5 grader_lib.Point module

class grader_lib.Point.**Point**(*parent_function*, *x*, *y*, *pixel=True*)

get_px_distance_squared(*point*)

get_x_distance(*x*)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

g

A

C

D

G

H

I

L

P

V