Mathtoolspy Documentation

Release 0.3 [4 - Beta]

sonntagsgesicht, based on a fork of Deutsche Postbank [pbrisk

Wednesday, 18 September 2019

Contents

CHAPTER 1

Introduction

1.1 Python library *mathtoolspy*

A fast, efficient Python library for mathematically operations, like integration, solver, distributions and other useful functions.

1.2 Example Usage

```
>>> from mathtoolspy.integration import gauss_kronrod

>>> fct = lambda x:exp(-x*x)
>>> integrator = gauss_kronrod()
>>> integrator(fct, -1.0, 2.0)
1.62890552357
```

1.3 Install

The latest stable version can always be installed or updated via pip:

```
$ pip install mathtoolspy
```

If the above fails, please try easy_install instead:

```
$ easy_install mathtoolspy
```

1.4 Examples

```
# Simplest example possible
        a, b, c, d, e = 1, 4, -6, -6, 1
    fct = lambda x : a*x*x*x*x + b*x*x*x + c*x*x + d*x + e
    opt = Optimizer1Dim(minimize_algorithm=brent)
    result = opt.optimize(fct, constraint=Constraint(-10.0, -2.0), initila_value=1.
↪0)
    >>> result.xmin
    -3.70107061641
    >>> result.fmin
    -74.1359364077
    >>> result.number_of_function_calls
    40
```

1.5 Development Version

The latest development version can be installed directly from GitHub:

```
$ pip install --upgrade git+https://github.com/pbrisk/mathtoolspy.git
```

1.6 Contributions

Issues and Pull Requests are always welcome.

1.7 License

Code and documentation are available according to the Apache Software License (see LICENSE).

CHAPTER 2

Tutorial

. . . will come soon.

CHAPTER 3

API Documentation

3.1 Distributions

mathtoolspy.distribution.normal_distribution.**density_normal_dist**(*x*)
Density function for normal distribution @param x: float value @return value of normal density function

mathtoolspy.distribution.normal_distribution.**normal_density**(*x*)

mathtoolspy.distribution.normal_distribution.**cdf_abramowitz_stegun**(*x*)
The cumulative distribution function of the standard normal distribution. The standard implementation, following Abramowitz/Stegun, (26.2.17).

mathtoolspy.distribution.normal_distribution.**normal_cdf**(*x*)

3.2 Interpolation

mathtoolspy.interpolation.linear.**interpolation_linear**(*x*, *x1*, *x2*, *y1*, *y2*)
Linear interpolation returns (y2 - y1) / (x2 - x1) * (x - x1) + y1

class mathtoolspy.interpolation.spline.**base_interpolation**(*x_list=[]*, *y_list=[]*)
Bases: object

Basic class to interpolate given data.

update(*x_list=[]*, *y_list=[]*)
update interpolation data :param list(float) x_list: x values :param list(float) y_list: y values

classmethod from_dict(*xy_dict*)

class mathtoolspy.interpolation.spline.**spline**(*x_list=[]*, *y_list=[]*, *boundary_condition=None*)
Bases: *mathtoolspy.interpolation.spline.base_interpolation*

interpolates the data with cubic splines.

Parameters

- **x_list** – data
- **y_list** – data

- **boundary_condition** – Either a tuple (l, r) of values for the slope or None. If the argument is not specified then None will be taken as boundary conditions, which leads to the so called not-a-knot method for splines. Not-a-knot will determine the boundary conditions by also requiring that the third derivatives of the two most left and the two most right interpolation polynomials agree. The boundary condition (0,0) will lead to the so called natural spline

class mathtoolspy.interpolation.spline.**natural_spline**(*x_list=[]*, *y_list=[]*)
Bases: *mathtoolspy.interpolation.spline.spline*

class mathtoolspy.interpolation.spline.**nak_spline**(*x_list=[]*, *y_list=[]*)
Bases: *mathtoolspy.interpolation.spline.spline*

mathtoolspy.interpolation.bilinear.**interpolation_bilinear**(*x*, *y*, *x1*, *x2*, *y1*, *y2*, *z11*, *z21*, *z22*, *z12*)

The points (x_i, y_i) and values z_ij are connected as follows: Starting from lower left going in mathematically positive direction, i.e. counter clockwise. Therefore: (x1,y1,z11), (x2,y1,z21), (x2,y2,z22), (x1,y2,z12).

3.3 Integration

class mathtoolspy.integration.gauss_kronrod_integrator.**GaussKronrodIntegrator**(*max_number_ini-tial_order=3*, *min_number_ abs_tolerance 10*, *rel_tolerance= check_abs_tol check_rel_tole*

Bases: object

integrate(*function*, *lower_bound*, *upper_bound*)

class mathtoolspy.integration.gauss_kronrod_integrator.**GaussKronrodConstants**
Bases: object

Constants for Gauss Kronrod Integrator

min_max_iter = 7

gaussKronrodPattersonRule = [3, 7, 15, 31, 63, 127, 255]

FL = [1, 2, 4, 8, 11, 13, 0]

FH = [1, 3, 7, 15, 16, 16, -1]

KL = [0, 0, 0, 0, 0, 2, 4, 8, 4, 8, 11]

KH = [0, 1, 3, 7, 15, 2, 5, 16, 4, 8, 16]

KX = [0, 1, 2, 3, 4, 5, 8, 11]

coefficients = [-0.1111111111111111, 0.22540333075851662, 0.5555555555555556, 0.0064

class mathtoolspy.integration.gauss_legendre_integrator.**GaussLegendreIntegrator**(*steps=100*)
Bases: object

Gauss Legendre integrator, which uses the Gauss Legendre quadratures

integrate(*function*, *lower_bound*, *upper_bound*)

class mathtoolspy.integration.gauss_legendre_integrator.**GaussLegendreQuadratures**
Bases: object

static get_values(*nsteps*)

class mathtoolspy.integration.gauss_lobatto_integrator.**GaussLobattoStep**(*function*, *alpha*, *beta*, *Is*, *max_number_of_iterati*

Bases: object

adaptive_step(*a*, *b*, *fa*, *fb*)

class mathtoolspy.integration.gauss_lobatto_integrator.**GaussLobattoIntegrator**(*max_number_*, *abs_tolerance* *10*)

Bases: object

integrate(*function*, *lower_bound*, *upper_bound*)

class mathtoolspy.integration.simplex_integrator.**SimplexIntegrator**(*steps=100*, *log_info=None*)

Bases: object

logNone(*x*)

integrate(*function*, *lower_bound*, *upper_bound*)

Calculates the integral of the given one dimensional function in the interval from lower_bound to upper_bound, with the simplex integration method.

3.4 Solver

mathtoolspy.solver.analytic_solver.**roots_of_cubic_polynom**(*a1*, *a2*, *a3*)

Finds the roots of a 3 dim polymon of the form x^3 + a1 * x^2 + a2 * x + a3. The roots are returned as complex numbers.

mathtoolspy.solver.minimize_algorithm_1dim_brent.**shift**(*a*, *b*, *c*)

The shift function returns the values as tuple.

mathtoolspy.solver.minimize_algorithm_1dim_brent.**minimize_algorithm_1dim_brent**(*fct*, *_a*, *_b*, *_c*, *tolerance=1e* *13*)

Finds the minimum of the given function f. The arguments are the given function f, and given a bracketing triplet of abscissas A, B, C (such that B is between A and C, and f(B) is less than both f(A) and f(C)) and the Tolerance. This routine isolates the minimum to a fractional precision of about tol using Brent's method. The abscissa of of the minimum is returned as xmin, and the minimum value is returned as brent, the returned function value.

mathtoolspy.solver.minimize_algorithm_1dim_golden.**minimize_algorithm_1dim_golden**(*function*, *a*, *b*, *c*, *tolerance=* *13*)

Given a function f, and given a bracketing triplet of abscissas ax, bx, cx (such that bx is between ax and cx, and f(bx) is less than both f(ax) and f(cx)), this routine performs a golden section search for the minimum, isolating it to a fractional precision of about tol. The abscissa of the minimum is returned as xmin, and the minimum function value is returned as Golden, the returned function value. See Press, et al. (1992) "Numerical recipes in C", 2nd ed., p.401.

class mathtoolspy.solver.minimize_algorithm_ndim_powell.**MinimizeAlgorithmNDimPowell**

Bases: object

The minimazation with the Powell algorithm. Press, et al., 'Numerical Recipes in C', 2nd ed, Powell (p.412).

Minimization of a function func of n variables. Input consists of an initial starting point p[0..n-1]; an inital matrix xi[0..n-1,0..n-1], whose columns contain the initial set of directions (usually the n unit vectors); and ftol, the fractional tolerance in the function value such that failure to decrease by more than this amount on one iteration signals doneness. On output, p is set to best point found, xi is the then-current direction set, fret is the returned function value at p. The routine linmin is used. *** Initial matrix xi[0..n-1,0..n-1], whose columns contain the initial set of directions (usually the n unit vectors). On output xi is the then-current direction set. *** Initial starting point p[0..n-1]. On output p is set to best point found.

TINY = 1e-25

TOL = 0.0002

FindMinimun(*fct*, *initial_point*, *tol*)

powell(*initial_xvalues*, *initial_fvalues*, *ndim*, *tol*, *fct*)

mathtoolspy.solver.minimum_bracketing.**GLIMIT = 100.0**
Used to prevent any possible division by zero.

mathtoolspy.solver.minimum_bracketing.**mn_brak**(*a*, *b*, *fct*)

mathtoolspy.solver.minimum_bracketing.**mn_brak_**(*a*, *b*, *fct*)

mathtoolspy.solver.minimum_bracketing.**minimum_bracketing**(*fct*, *initial_value=0.0*, *natural_length=1.0000000000001*)
Given a function func, and given distinct inital points ax and bx, this routine searches in the downhill direction (defined by the function as evaluated at the initial points) and returns new points at ax, bx, cx that bracket a minimum of the function. Also returned are the function values at the three points. See Press, et al. (1992) "Numerical recipes in C", 2nd ed., p.400.

mathtoolspy.solver.minimum_bracketing.**simple_bracketing**(*func*, *a*, *b*, *precision=1e-20*)
find root by simple_bracketing an interval

Parameters

- **func** (*callable*) – function to find root
- **a** (*float*) – lower interval boundary
- **b** (*float*) – upper interval boundary
- **precision** (*float*) – max accepted error

Return type tuple

Returns `(a, m, b)` of last recursion step with `m = a + (b-a) *.5`

class mathtoolspy.solver.optimizer.**FctWithCount**(*fct*)
Bases: object

class mathtoolspy.solver.optimizer.**DeviationFct**(*fct*, *rescale=None*, *max_number_of_function_calls=5000*)
Bases: object

can_be_called()

class mathtoolspy.solver.optimizer.**Constraint**(*lower_bound*, *upper_bound*)
Bases: object

is_a_number()

mid_point()

contains(*x*)

static create_constraint(*constraint_tupel*)

class mathtoolspy.solver.optimizer.**InfinityConstraint**
Bases: object

is_a_number()

mid_point()

contains(*x*)

class mathtoolspy.solver.optimizer.**Optimizer1Dim**(*minimize_algorithm*)
Bases: object

optimize(*function*, *constraint=None*, *initial_value=0.0*, *tolerance=1e-08*)

class mathtoolspy.solver.optimizer.**Optimizer**(*minimizeAlgorithm*)
Bases: object

optimize(*function*, *constraint*, *initialValues*, *tolerance=1e-08*)

class mathtoolspy.solver.optimizer.**OptimizerResult**
Bases: object

static create_succesful(*xmin*, *fmin*, *numberOfFunctionCalls*)

static create_not_succesful(*errorMsg*, *numberOfFunctionCalls*)

class mathtoolspy.solver.optimizer.**TangentsInverseTransformation**(*constraint*)
Bases: object

class mathtoolspy.solver.optimizer.**TangentsTransformation**(*constraint*)
Bases: object

class mathtoolspy.solver.optimizer.**OptimizerInitialValuesSearchOnFixedGrid**(*minimizeAlgorithm*, *stepsPerAxis*)
Bases: object

An optimizer, which searches in a fix grid for the best initial value.

optimize(*function*, *constraint*, *initialValues*, *tolerance=1e-08*)

class mathtoolspy.solver.optimizer.**combinations**(*listOfTupel*)
Bases: object

3.5 Utilities

The MathFct contains some mathematically method, which are not supported by the Python lib.

mathtoolspy.utils.math_fcts.**abs_sign**(*a*, *b*)
The absolute value of A with the sign of B.

mathtoolspy.utils.math_fcts.**sign**(*x*)
Returns the sign of the double number x. -1 if x < 0; 1 if x > 0 and 0 if x == 0

mathtoolspy.utils.math_fcts.**float_equal**(*x*, *y*, *tol=1e-13*)

mathtoolspy.utils.math_fcts.**prod**(*factors*)
The product of the given factors (iterable) :param factors: :return:

mathtoolspy.utils.math_fcts.**get_grid**(*start*, *end*, *nsteps=100*)
Generates a equal distanced list of float values with nsteps+1 values, begining start and ending with end.

Parameters **start** – the start value of the generated list.

:type float

Parameters **end** – the end value of the generated list.

:type float

Parameters **nsteps** – optional the number of steps (default=100), i.e. the generated list contains nstep+1 values.

:type int

class mathtoolspy.utils.math_fcts.**FctWithCount**(*fct*)
Bases: object

class mathtoolspy.utils.math_fcts.**CompositionFct**(**fcts*)
Bases: object

mathtoolspy.utils.mathconst.**DOUBLE_TOL = 1e-13**
PI

mathtoolspy.utils.mathconst.**PI = 3.141592653589793**
The factor $sqrt{2}$.

mathtoolspy.utils.mathconst.**SQRT_OF_TWO = 1.414213562373095**
$sqrt{Pi}$.

Type The factor

mathtoolspy.utils.mathconst.**SQRT_OF_PI = 1.772453850905516**
$sqrt{2*Pi}$.

Type The factor

mathtoolspy.utils.mathconst.**SQRT_OF_TWO_PI = 2.506628274631**
$1/sqrt{2Pi}$.

Type The prefactor of the normal density

mathtoolspy.utils.mathconst.**ONE_OVER_SQRT_OF_TWO_PI = 0.398942280401433**
The factor $1/{2Pi}$.

mathtoolspy.utils.mathconst.**ONE_OVER_TWO_PI = 0.15915494309189532**
The golden ratio.

mathtoolspy.utils.mathconst.**GOLD = 0.61803399**
One minus GOLD

mathtoolspy.utils.mathconst.**ONE_MINUS_GOLD = 0.38196601**
One over GOLD

class mathtoolspy.utils.surface.**Surface**(*xaxis*, *yaxis*, *values*)
Bases: object

A matrix with interpolation and extrapolation.

The values has to be a float matrix implementing the method get_item(i, j). @params xaxis: list of float values. @params yaxis: list of float values. @params values: some object implementing a get_item(i, j) method or nested list.

get_value(*x*, *y*)

CHAPTER 4

Releases

These changes are listed in decreasing version number order.

4.1 Release 0.3

Release date was Wednesday, 18 September 2019

4.2 Release 0.2

December 31th, 2017

4.3 Release 0.1

Release date was July 7th, 2017

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

I

K

L

M

N

O

P

R

S

T

U