COPADS: Collection of Python Algorithms and Data Structures

API Documentation

August 6, 2013

Contents

Contents **1**

1 Package copads **2**
- 1.1 Modules . . . 2
- 1.2 Variables . . . 4

2 Module copads.array **5**
- 2.1 Variables . . . 5
- 2.2 Class ParallelArray . . . 5
  - 2.2.1 Methods . . . 5
  - 2.2.2 Properties . . . 6

3 Module copads.bag **7**
- 3.1 Variables . . . 7
- 3.2 Class Bag . . . 7
  - 3.2.1 Methods . . . 7
  - 3.2.2 Properties . . . 9

4 Module copads.colormap **10**
- 4.1 Functions . . . 10
- 4.2 Variables . . . 10

5 Module copads.constants **11**
- 5.1 Variables . . . 11

6 Module copads.copadsexceptions **13**
- 6.1 Variables . . . 14
- 6.2 Class CopadsError . . . 14
  - 6.2.1 Methods . . . 14
  - 6.2.2 Properties . . . 14
- 6.3 Class MatrixError . . . 15
  - 6.3.1 Methods . . . 15
  - 6.3.2 Properties . . . 15
- 6.4 Class MatrixArithmeticError . . . 16
  - 6.4.1 Methods . . . 16
  - 6.4.2 Properties . . . 17
- 6.5 Class MatrixMultiplicationError . . . 17
  - 6.5.1 Methods . . . 17

6.5.2 Properties . . . 18
6.6 Class MatrixAdditionError . . . 18
6.6.1 Methods . . . 18
6.6.2 Properties . . . 19
6.7 Class MatrixSquareError . . . 19
6.7.1 Methods . . . 20
6.7.2 Properties . . . 20
6.8 Class MatrixTraceError . . . 21
6.8.1 Methods . . . 21
6.8.2 Properties . . . 21
6.9 Class MatrixMinorError . . . 22
6.9.1 Methods . . . 22
6.9.2 Properties . . . 23
6.10 Class MatrixDeterminantError . . . 23
6.10.1 Methods . . . 23
6.10.2 Properties . . . 24
6.11 Class GraphError . . . 24
6.11.1 Methods . . . 24
6.11.2 Properties . . . 25
6.12 Class EdgeNotFoundError . . . 25
6.12.1 Methods . . . 25
6.12.2 Properties . . . 26
6.13 Class VertexNotFoundError . . . 26
6.13.1 Methods . . . 26
6.13.2 Properties . . . 27
6.14 Class UnknownGraphMatrixError . . . 27
6.14.1 Methods . . . 28
6.14.2 Properties . . . 28
6.15 Class NotAdjacencyGraphMatrixError . . . 29
6.15.1 Methods . . . 29
6.15.2 Properties . . . 30
6.16 Class GraphEdgeSizeMismatchError . . . 30
6.16.1 Methods . . . 30
6.16.2 Properties . . . 31
6.17 Class GraphParameterError . . . 31
6.17.1 Methods . . . 31
6.17.2 Properties . . . 32
6.18 Class StatisticsError . . . 32
6.18.1 Methods . . . 33
6.18.2 Properties . . . 33
6.19 Class DistributionError . . . 33
6.19.1 Methods . . . 34
6.19.2 Properties . . . 34
6.20 Class NormalDistributionTypeError . . . 34
6.20.1 Methods . . . 35
6.20.2 Properties . . . 35
6.21 Class DistributionParameterError . . . 36
6.21.1 Methods . . . 36
6.21.2 Properties . . . 36
6.22 Class DistributionFunctionError . . . 37
6.22.1 Methods . . . 37

6.22.2 Properties . . . 38
6.23 Class DistanceError . . . 38
6.23.1 Methods . . . 38
6.23.2 Properties . . . 39
6.24 Class DistanceInputSizeError . . . 39
6.24.1 Methods . . . 39
6.24.2 Properties . . . 40
6.25 Class TreeError . . . 40
6.25.1 Methods . . . 40
6.25.2 Properties . . . 41
6.26 Class TreeNodeTypeError . . . 41
6.26.1 Methods . . . 41
6.26.2 Properties . . . 42
6.27 Class FunctionParameterTypeError . . . 42
6.27.1 Methods . . . 42
6.27.2 Properties . . . 43
6.28 Class FunctionParameterValueError . . . 43
6.28.1 Methods . . . 43
6.28.2 Properties . . . 44
6.29 Class ArrayError . . . 44
6.29.1 Methods . . . 45
6.29.2 Properties . . . 45
6.30 Class MaxIterationsException . . . 46
6.30.1 Methods . . . 46
6.30.2 Properties . . . 46

7 Module copads.dataframe 47
7.1 Variables . . . 47
7.2 Class dataframe . . . 47
7.2.1 Methods . . . 47
7.2.2 Properties . . . 49
7.2.3 Class Variables . . . 49

8 Module copads.dose_entities 50
8.1 Variables . . . 50
8.2 Class Organism . . . 50
8.2.1 Methods . . . 51
8.2.2 Properties . . . 52
8.2.3 Class Variables . . . 53
8.3 Class Population . . . 53
8.3.1 Methods . . . 53
8.3.2 Properties . . . 55
8.4 Class World . . . 56
8.4.1 Methods . . . 56
8.4.2 Properties . . . 58
8.4.3 Class Variables . . . 58

9 Module copads.dose_executor 59
9.1 Functions . . . 59
9.2 Variables . . . 60

10 Module copads.dose_parameters 61

10.1 Variables 61

11 Module copads.dose_world **64**
11.1 Variables 64
11.2 Class World 64
11.2.1 Methods 65
11.2.2 Properties 68
11.2.3 Class Variables 68

12 Module copads.genetic **69**
12.1 Functions 69
12.2 Variables 71
12.3 Class Chromosome 71
12.3.1 Methods 72
12.3.2 Properties 75
12.4 Class Organism 75
12.4.1 Methods 77
12.4.2 Properties 79
12.4.3 Class Variables 80
12.5 Class Population 80
12.5.1 Methods 81
12.5.2 Properties 84

13 Module copads.graph **85**
13.1 Variables 85
13.2 Class Graph 85
13.2.1 Methods 85
13.2.2 Class Variables 88

14 Module copads.hash **89**
14.1 Functions 89
14.2 Variables 91

15 Module copads.hypothesis **92**
15.1 Functions 93
15.2 Variables 114

16 Module copads.lc_bf **116**
16.1 Functions 116
16.2 Variables 117

17 Module copads.matrix **118**
17.1 Functions 118
17.2 Variables 120
17.3 Class Vector 120
17.3.1 Methods 121
17.3.2 Properties 122
17.3.3 Class Variables 122
17.4 Class Matrix 123
17.4.1 Methods 124
17.4.2 Class Variables 127
17.5 Class SparseMatrix 128
17.5.1 Methods 128

17.5.2 Properties . . . 129
17.5.3 Class Variables . . . 129

18 Module copads.n_bf **130**
18.1 Functions . . . 130
18.2 Variables . . . 130

19 Module copads.neural **131**
19.1 Functions . . . 131
19.2 Variables . . . 131
19.3 Class Neuron . . . 131
19.3.1 Methods . . . 132
19.3.2 Class Variables . . . 133
19.4 Class Brain . . . 134
19.4.1 Methods . . . 135
19.4.2 Class Variables . . . 137

20 Module copads.nrpy **138**
20.1 Functions . . . 139
20.2 Variables . . . 149

21 Module copads.objectdistance **150**
21.1 Functions . . . 150
21.2 Variables . . . 167

22 Module copads.operations **169**
22.1 Functions . . . 169
22.2 Variables . . . 172
22.3 Class Modulus2 . . . 172
22.3.1 Methods . . . 172
22.4 Class Boolean . . . 172
22.4.1 Methods . . . 173

23 Module copads.prioritydictionary **174**
23.1 Variables . . . 174
23.2 Class PriorityDictionary . . . 174
23.2.1 Methods . . . 174
23.2.2 Properties . . . 175
23.2.3 Class Variables . . . 175

24 Module copads.ragaraja **176**
24.1 Functions . . . 176
24.2 Variables . . . 188

25 Module copads.register_machine **189**
25.1 Functions . . . 189
25.2 Variables . . . 190

26 Module copads.ring **191**
26.1 Variables . . . 191
26.2 Class RingList . . . 191
26.2.1 Methods . . . 191

27 Module copads.samplestatistics **193**
27.1 Variables . . . 193
27.2 Class SingleSample . . . 193
27.2.1 Methods . . . 193
27.2.2 Class Variables . . . 195
27.3 Class SampleDistribution . . . 195
27.3.1 Methods . . . 195
27.4 Class TwoSample . . . 195
27.4.1 Methods . . . 196
27.4.2 Class Variables . . . 196

28 Module copads.set **198**
28.1 Variables . . . 198
28.2 Class Set . . . 198
28.2.1 Methods . . . 198

29 Module copads.statisticsdistribution **200**
29.1 Functions . . . 202
29.2 Variables . . . 204
29.3 Class Distribution . . . 205
29.3.1 Methods . . . 205
29.4 Class BetaDistribution . . . 206
29.4.1 Methods . . . 207
29.5 Class BinomialDistribution . . . 208
29.5.1 Methods . . . 209
29.6 Class CauchyDistribution . . . 210
29.6.1 Methods . . . 211
29.7 Class CosineDistribution . . . 212
29.7.1 Methods . . . 213
29.8 Class ExponentialDistribution . . . 214
29.8.1 Methods . . . 215
29.9 Class FDistribution . . . 217
29.9.1 Methods . . . 217
29.10Class GammaDistribution . . . 218
29.10.1Methods . . . 218
29.11Class ChiSquareDistribution . . . 220
29.11.1Methods . . . 220
29.12Class GeometricDistribution . . . 221
29.12.1Methods . . . 221
29.13Class HypergeometricDistribution . . . 222
29.13.1Methods . . . 223
29.14Class LogarithmicDistribution . . . 224
29.14.1Methods . . . 224
29.15Class NormalDistribution . . . 225
29.15.1Methods . . . 226
29.16Class PoissonDistribution . . . 228
29.16.1Methods . . . 228
29.17Class SemicircularDistribution . . . 229
29.17.1Methods . . . 230
29.18Class TDistribution . . . 231
29.18.1Methods . . . 232
29.19Class TriangularDistribution . . . 234

29.19.1 Methods . . . 234
29.20 Class UniformDistribution . . . 236
29.20.1 Methods . . . 236
29.21 Class WeiBullDistribution . . . 238
29.21.1 Methods . . . 238
29.22 Class BernoulliDistribution . . . 239
29.22.1 Methods . . . 240
29.23 Class BradfordDistribution . . . 241
29.23.1 Methods . . . 241
29.24 Class BurrDistribution . . . 243
29.24.1 Methods . . . 243
29.25 Class ChiDistribution . . . 245
29.25.1 Methods . . . 245
29.26 Class DoubleGammaDistribution . . . 245
29.26.1 Methods . . . 245
29.27 Class DoubleWeibullDistribution . . . 247
29.27.1 Methods . . . 247
29.28 Class ExtremeLBDistribution . . . 247
29.28.1 Methods . . . 247
29.29 Class FiskDistribution . . . 248
29.29.1 Methods . . . 248
29.30 Class FoldedNormalDistribution . . . 248
29.30.1 Methods . . . 248
29.31 Class GenLogisticDistribution . . . 249
29.31.1 Methods . . . 249
29.32 Class GumbelDistribution . . . 249
29.32.1 Methods . . . 249
29.33 Class HalfNormalDistribution . . . 251
29.33.1 Methods . . . 251
29.34 Class HyperbolicSecantDistribution . . . 253
29.34.1 Methods . . . 253
29.35 Class LaplaceDistribution . . . 254
29.35.1 Methods . . . 254
29.36 Class LogisticDistribution . . . 255
29.36.1 Methods . . . 255
29.37 Class LogNormalDistribution . . . 255
29.37.1 Methods . . . 255
29.38 Class MaxwellDistribution . . . 256
29.38.1 Methods . . . 256
29.39 Class NakagamiDistribution . . . 257
29.39.1 Methods . . . 258
29.40 Class NegativeBinomialDistribution . . . 258
29.40.1 Methods . . . 258
29.41 Class ParetoDistribution . . . 259
29.41.1 Methods . . . 260
29.42 Class PascalDistribution . . . 261
29.42.1 Methods . . . 262
29.43 Class PowerFunctionDistribution . . . 263
29.43.1 Methods . . . 263
29.44 Class RademacherDistribution . . . 264
29.44.1 Methods . . . 265

29.45Class RayleighDistribution . . . 266
29.45.1 Methods . . . 266
29.46Class ReciprocalDistribution . . . 267
29.46.1 Methods . . . 267

30 Module copads.tree **268**
30.1 Variables . . . 268
30.2 Class RBTreeIter . . . 268
30.2.1 Methods . . . 268
30.2.2 Properties . . . 269
30.3 Class OrderedBinaryTree . . . 269
30.3.1 Methods . . . 269
30.4 Class RBTree . . . 269
30.4.1 Methods . . . 270
30.4.2 Properties . . . 271
30.5 Class RBList . . . 271
30.5.1 Methods . . . 272
30.5.2 Properties . . . 273
30.6 Class RBDict . . . 273
30.6.1 Methods . . . 273
30.6.2 Properties . . . 275

31 Module copads.treenodes **276**
31.1 Variables . . . 276
31.2 Class RBNode . . . 276
31.2.1 Methods . . . 276
31.2.2 Properties . . . 277
31.3 Class BinaryNode . . . 277
31.3.1 Methods . . . 277
31.4 Class ThreeNode . . . 277
31.4.1 Methods . . . 277
31.5 Class FourNode . . . 277
31.5.1 Methods . . . 277

1 Package copads

COPADS (Collection of Python Algorithms and Data Structures).

The main aim of COPADS is to develop a compilation of of Python data structures and its algorithms, making it almost a purely developmental project. Personally, I look at this as a re-usable collection of tools that I can use in other projects. Therefore, this project is essentially "needs-driven", except a core subset of data structures and algorithms.

This project originated from 3 threads of thought. Firstly, while browsing through Mehta and Sahni's Handbook of Data Structures and Applications, I thought there might be utility to have a number of the listed data structures implemented in Python. Given my interest in biological data management, having a good set of data structures is always handy. The 2nd thread of thought came from Numerical Recipes. Again, I thought these algorithms will be handy to have and had started to translate some of them into Python during some overly energetic days. Finally, Python Cookbook had undergone 2 editions by 2008 and ActiveState had provided an online platform for Python Recipes which I found to be useful and can see how some of these recipes can be merged. Thus, COPADS is borned.

Project website: *http://copads.sf.net*

License: Unless specified otherwise, all parts of this package, except those adapted, are covered under Python Software Foundation License version 2.

Version: 0.4.1

Author: Maurice H.T. Ling <mauriceling@acm.org>

Copyright: (c) 2007-2013, Maurice H.T. Ling.

1.1 Modules

- **array**: Array Data Structures and Algorithms.
  (Section 2, p. 5)
- **bag**: Bag Data Structures and Algorithms.
  (Section 3, p. 7)
- **colormap**: These functions, when given a magnitude mag between cmin and cmax, return a colour tuple (red, green, blue).
  (Section 4, p. 10)
- **constants**: List of Constant Numbers
  (Section 5, p. 11)
- **copadsexceptions**: Exceptions Defined for COPADS.
  (Section 6, p. 13)
- **dataframe**: A generic dataframe to hold data and analysis Date created: 24th September 2012 Licence: Python Software Foundation License version 2
  (Section 7, p. 47)
- **dose_entities**: Boiler-plate codes for DOSE (digital organism simulation environment) entities Date created: 13th September 2012 Licence: Python Software Foundation License version 2
  (Section 8, p. 50)
- **dose_executor**: Default DOSE simulation runner in manuscript [1] Date created: 13th September 2012 Licence: Python Software Foundation License version 2
  (Section 9, p. 59)
- **dose_parameters**: Parameter file for DOSE simulation.

(Section 10, p. 61)
- **dose_world**: World structure for DOSE (digital organism simulation environment) Date created: 13th September 2012 Licence: Python Software Foundation License version 2
  (Section 11, p. 64)
- **genetic**: Framework for Genetic Algorithm Applications.
  (Section 12, p. 69)
- **graph**: Graph Data Structures and Algorithms.
  (Section 13, p. 85)
- **hash**: Hash Generators Date created: 16th April 2013 Licence: Python Software Foundation License version 2
  (Section 14, p. 89)
- **hypothesis**: Statistical Hypothesis Testing Routines.
  (Section 15, p. 92)
- **lc_bf**: Loose Circular Brainfuck (LCBF) Interpreter Date created: 15th August 2012 Licence: Python Software Foundation License version 2
  (Section 16, p. 116)
- **matrix**: Matrix Data Structures and Algorithms.
  (Section 17, p. 118)
- **n_bf**: NucleotideBF (nBF) Interpreter Date created: 15th August 2012 Licence: Python Software Foundation License version 2
  (Section 18, p. 130)
- **neural**: Framework for Neural Network Applications Date created: 10th May 2013 License: Python Software Foundation License version 2
  (Section 19, p. 131)
- **nrpy**: Numerical Recipes in Python.
  (Section 20, p. 138)
- **objectdistance**: Functions for Calculating Similarity Coefficients between 2 Objects.
  (Section 21, p. 150)
- **operations**: Mathematical Operation Routines.
  (Section 22, p. 169)
- **prioritydictionary**: Priority Dictionary and Algorithms.
  (Section 23, p. 174)
- **ragaraja**: Ragaraja Interpreter Date created: 16th August 2012 Licence: Python Software Foundation License version 2
  (Section 24, p. 176)
- **register_machine**: One dimensional tape/register machine Date created: 15th August 2012 Licence: Python Software Foundation License version 2
  (Section 25, p. 189)
- **ring**: Ring Data Structures and Algorithms.
  (Section 26, p. 191)
- **samplestatistics**: Data Structures and Algorithms for Data Collected from One or More Samples.
  (Section 27, p. 193)
- **set**: Sets Data Structure and Algorithms.
  (Section 28, p. 198)
- **statisticsdistribution**: Classes for Various Statistical Distributions.
  (Section 29, p. 200)
- **tree**: Tree Data Structures and Algorithms.
  (Section 30, p. 268)
- **treenodes**: Node Classes for Tree Data Structures.
  (Section 31, p. 276)

1.2 Variables

Name	Description
__package__	**Value:** `None`

2 Module copads.array

Array Data Structures and Algorithms.

Date created: 19th March 2008

2.1 Variables

Name	Description
__package__	**Value:** `'copads'`

2.2 Class ParallelArray

object
└ **copads.array.ParallelArray**

Parallel Array is an array whereby each data list in the array is of the same size. Ref: http://en.wikipedia.org/wiki/Parallel_arr

2.2.1 Methods

__init__(*self*, *fields*=`None`)

Constructor. Able to initiate the array with a list of fields names

Parameters

`fields:` list of field names to initiate. Default = None

Overrides: object.__init__

initFields(*self*, *fields*)

Initiates a list of fields into the array.

Parameters

`fields:` list of field names to initiate.

addField(*self*, *field*)

Method to add a new field into the array. Raises ArrayError if attempt to add an existing field.

Parameters

`field:` name of field

(type=string)

removeField(*self*, *field*)

Removes a field, together with its data, from the array.

Parameters

`field:` name of field

(type=string)

changeField(*self*, *oldname*, *newname*)

Change a field name.

Parameters

`oldname:` existing name of field

(type=string)

`newname:` new name to change to. A new field by this name will be created if 'oldname' is not found

(type=string)

addData(*self*, *fields*, *data*)

Add data into the array

Parameters

`fields:` ordered list of fields

`data:` ordered list of data to add

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

2.2.2 Properties

Name	Description
Inherited from object	
__class__	

3 Module copads.bag

Bag Data Structures and Algorithms.

3.1 Variables

Name	Description
__package__	**Value:** `'copads'`

3.2 Class Bag

object
└
copads.bag.Bag

Bag is also known as multi-set, or a set that can have duplicate elements

Adapted from: http://aspn.activestate.com/ASPN/Cookbook/ Python/Recipe/259174 Original author: Raymond Hettinger

3.2.1 Methods

__init__(*self*, *iterable*={})

Constructor.

Parameters

`iterable`: dictionary of elements to add

Overrides: object.__init__

update(*self*, *iterable*)

Update the bag

Parameters

`iterable`: dictionary if elements to add

__contains__(*self*, *elem*)

__getitem__(*self*, *elem*)

__setitem__(*self*, *elem*, *n*)

__delitem__(*self*, *elem*)

__len__(*self*)

__eq__(*self*, *other*)

__ne__(*self*, *other*)

__hash__(*self*)

hash(x)

Overrides: object.__hash__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

copy(*self*)

__copy__(*self*)

__deepcopy__(*self*, *memo*)

__getstate__(*self*)

__setstate__(*self*, *data*)

clear(*self*)

Remove all data elements in the bag

__iter__(*self*)

iterunique(*self*)

itercounts(*self*)

mostcommon(*self*, *n*=`None`)

Lists the most common elements of the bag and its counts

Return Value

list of (element, count of the element)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.2.2 Properties

Name	Description
Inherited from object	
__class__	

4 Module copads.colormap

These functions, when given a magnitude mag between cmin and cmax, return a colour tuple (red, green, blue). Light blue is cold (low magnitude) and yellow is hot (high magnitude).

Adapted from http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52273 Original author: Alexander Pletzer

4.1 Functions

floatRgb(*mag*, *cmin*, *cmax*)

Return a tuple of floats between 0 and 1 for the red, green and blue amplitudes.

strRgb(*mag*, *cmin*, *cmax*)

Return a tuple of strings to be used in Tk plots.

rgb(*mag*, *cmin*, *cmax*)

Return a tuple of integers to be used in AWT/Java plots.

htmlRgb(*mag*, *cmin*, *cmax*)

Return a tuple of strings to be used in HTML documents.

4.2 Variables

Name	Description
__package__	**Value:** `'copads'`

5 Module copads.constants

List of Constant Numbers

- CGOLD: 2 - GOLD
- FRODA: Froda Constant
- GAMMA: Euler's gamma constant
- GOLD: golden mean, also known as golden ratio ((1 + sqrt(5))/2)
- GOLOMB = Golomb Constant
- INVLN10: inverse of natural logarithm of ten (1/ln(10))
- INVLN2: inverse of natural logarithm of 2 (1/ln(2))
- INVPI180: inverse of PI180, also means 180 divide by π
- INVSQRT2PI: inverse of SQRT2PI (1/(SQRT(2π)))
- INV2PI: inverse of 2π
- LEHMER: Lehmer constant
- LN10: natural logarithm of 10
- LN2: natural logarithm of 2
- LNPI: natural logarithm of π
- PI: ratio of circumference of a circle to its diameter (π)
- PI2: 2π
- PI180: π divided by 180
- PIDIV2: π divided by 2
- RABBIT: Rabbit Constant
- SQRT2: square root of 2
- SQRT2PI: square root of 2π
- ZETA9: $\sum$(1/n^9, n=1..infinity)
- CFRACT: π^2/(6 (ln 2) (ln 10))
- PARIS: Paris Constant
- LENGYEL: Lengyel Constant
- BACKHOUSE: Backhouse Constant
- PORTER: Porter Constant

Date created: 19th March 2008

5.1 Variables

Name	Description
CGOLD	**Value:** `0.38196601125`
FRODA	**Value:** `6.58088599102`
GAMMA	**Value:** `0.577215664902`
GOLD	**Value:** `1.61803398875`
GOLOMB	**Value:** `0.624329988544`
INVLN10	**Value:** `0.434294481903`
INVLN2	**Value:** `1.44269504089`
INVPI180	**Value:** `57.2957795131`
INVSQRT2PI	**Value:** `0.398942280401`
INV2PI	**Value:** `0.159154943092`
LEHMER	**Value:** `0.592632718292`
LN10	**Value:** `2.30258509299`
LN2	**Value:** `0.69314718056`
LNPI	**Value:** `1.14472988585`
PI	**Value:** `3.14159265359`
PI2	**Value:** `6.28318530718`
PI180	**Value:** `0.0174532925199`
RABBIT	**Value:** `0.709803442861`
PIDIV2	**Value:** `1.57079632679`
SQRT2	**Value:** `1.41421356237`
SQRT2PI	**Value:** `2.50662827463`
ZETA9	**Value:** `1.00200839283`
CFRACT	**Value:** `1.0306408341`
PARIS	**Value:** `1.09864196439`
LENGYEL	**Value:** `1.09868580553`
BACKHOUSE	**Value:** `1.45607494858`
PORTER	**Value:** `1.46707807943`
__package__	**Value:** `None`

6 Module copads.copadsexceptions

Exceptions Defined for COPADS.

- CopadsError
  - MatrixError
    - MatrixArithmeticError
      - MatrixMultiplicationError
    - MatrixAdditionError
    - MatrixSquareError
    - MatrixTraceError
    - MatrixMinorError
    - MatrixDeterminantError
  - GraphError
    - EdgeNotFoundError
    - VertexNotFoundError
    - UnknownGraphMatrixError
      - NotAdjacencyGraphMatrixError
    - GraphEdgeSizeMismatchError
  - StatisticsError
    - DistributionError
      - NormalDistributionTypeError
    - DistributionParameterError
    - DistributionFunctionError
  - DistanceError
    - DistanceInputSizeError
  - TreeError
    - TreeNodeTypeError
  - FunctionParameterTypeError
  - FunctionParameterValueError
  - ArrayError
  - MaxIterationException

Credits

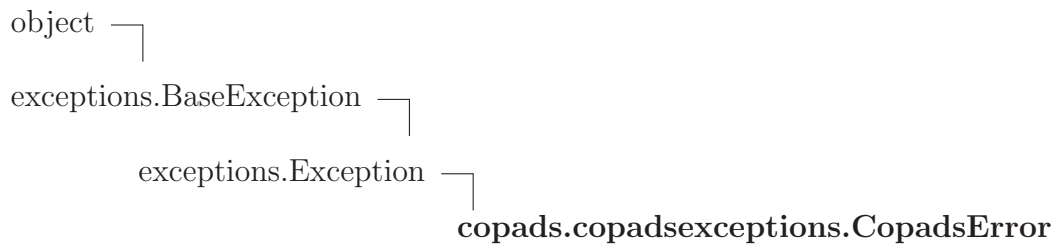
- MatrixError subclasses (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/189971)

Date created: 1st May 2005

6.1 Variables

Name	Description
__package__	**Value:** `None`

6.2 Class CopadsError



Known Subclasses: copads.copadsexceptions.ArrayError, copads.copadsexceptions.DistanceError, copads.copadsexceptions.StatisticsError, copads.copadsexceptions.GraphError, copads.copadsexceptions.F copads.copadsexceptions.FunctionParameterValueError, copads.copadsexceptions.MatrixError, copads.copadsexceptions.MaxIterationsException, copads.copadsexceptions.TreeError

Base class for all Copads-defined exceptions.

6.2.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

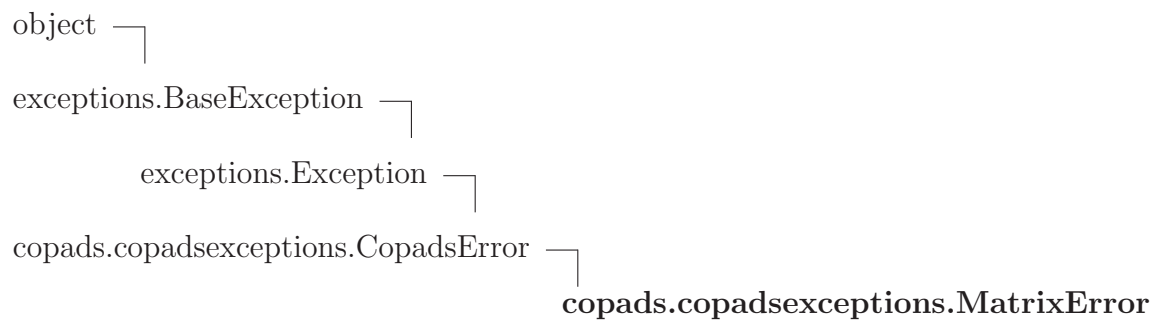
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.2.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.3 Class MatrixError



Known Subclasses: copads.copadsexceptions.MatrixArithmeticError, copads.copadsexceptions.MatrixSc

Abstract parent for all matrix exceptions

6.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

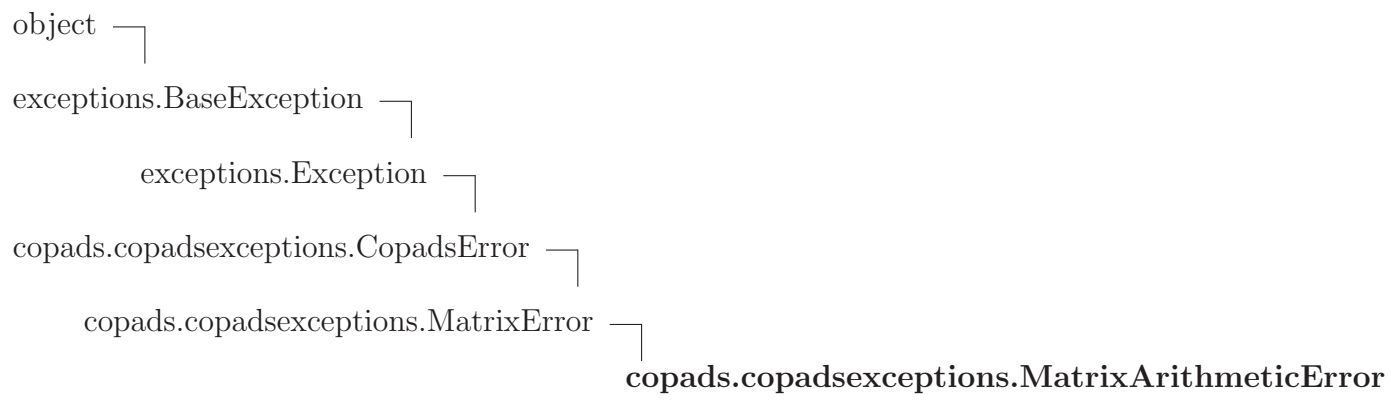
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.4 Class MatrixArithmeticError



Known Subclasses: copads.copadsexceptions.MatrixAdditionError, copads.copadsexceptions.MatrixMul

Incorrect dimensions for arithmetic

This exception is thrown when you try to add or multiply matricies of incompatible sizes.

6.4.1 Methods

__**init**__(*self*, *a*, *b*, *operation*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__**str**__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

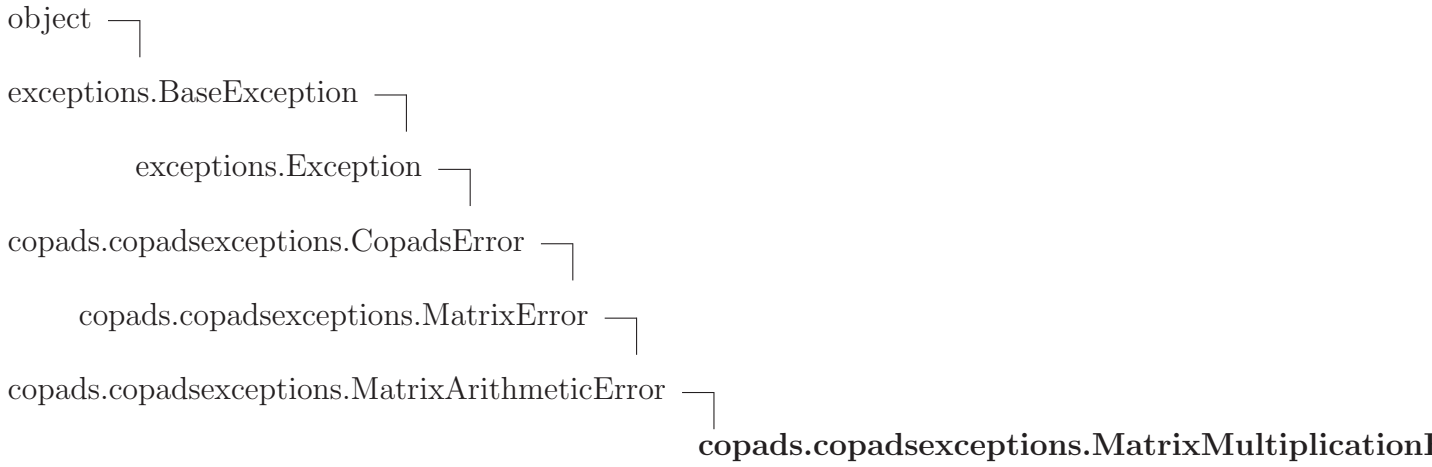
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.4.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.5 Class MatrixMultiplicationError



Thrown when you try to multiply matricies of incompatible dimensions.

This exception is also thrown when you try to right-multiply a row vector or left-multiply a column vector.

6.5.1 Methods

__init__(*self*, *a*, *b*)
x.__init__(...) initializes x; see help(type(x)) for signature
Overrides: object.__init__ extit(inherited documentation)

Inherited from copads.copadsexceptions.MatrixArithmeticError(Section 6.4)

__str__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

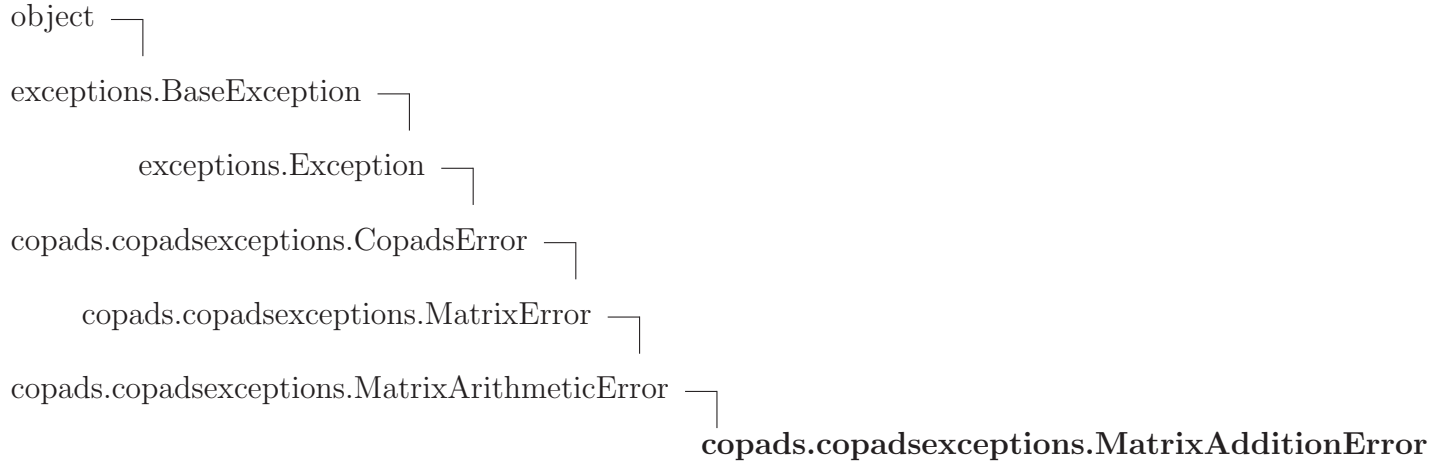
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.5.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.6 Class MatrixAdditionError



Thrown when you try to add matricies of incompatible dimensions.

6.6.1 Methods

__init__(*self*, *a*, *b*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from copads.copadsexceptions.MatrixArithmeticError(Section 6.4)

__str__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

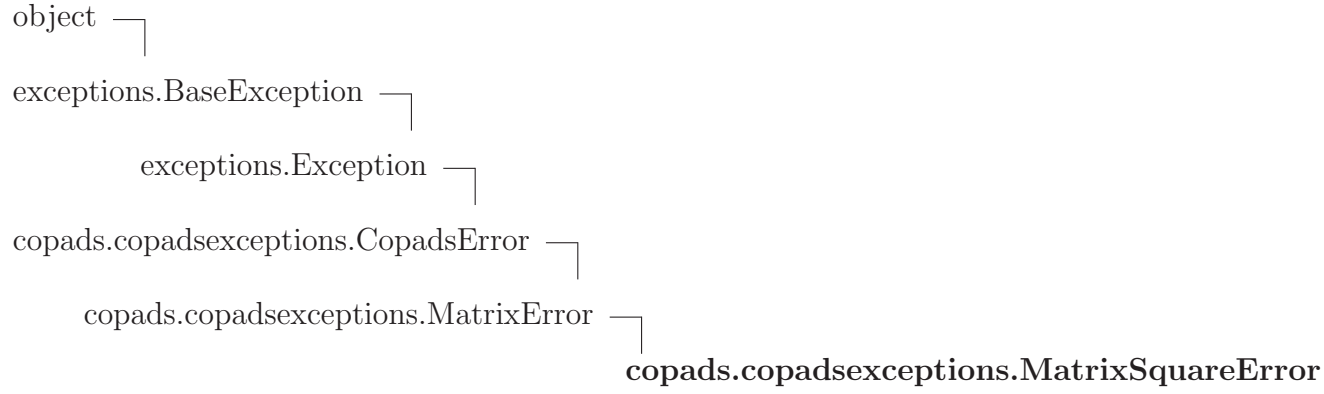
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.6.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.7 Class MatrixSquareError



Known Subclasses: copads.copadsexceptions.MatrixDeterminantError, copads.copadsexceptions.Matrix
copads.copadsexceptions.MatrixTraceError

Square-matrix only

This exception is thrown when you try to calculate a function that is only defined for square matricies on a non-square matrix.

6.7.1 Methods

__init__(*self*, *func*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.7.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.8 Class MatrixTraceError

object
exceptions.BaseException
exceptions.Exception
copads.copadsexceptions.CopadsError
copads.copadsexceptions.MatrixError
copads.copadsexceptions.MatrixSquareError
copads.copadsexceptions.MatrixTraceError

Thrown when you try to get the trace of a non-square matrix.

6.8.1 Methods

__init__(*self*)
x.__init__(...) initializes x; see help(type(x)) for signature
Overrides: object.__init__ extit(inherited documentation)

Inherited from copads.copadsexceptions.MatrixSquareError(Section 6.7)

__str__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.8.2 Properties

Name	Description
Inherited from exceptions.BaseException	

continued on next page

Name	Description
args, message	
Inherited from object __class__	

6.9 Class MatrixMinorError

object
exceptions.BaseException
exceptions.Exception
copads.copadsexceptions.CopadsError
copads.copadsexceptions.MatrixError
copads.copadsexceptions.MatrixSquareError
copads.copadsexceptions.MatrixMinorError

Thrown when you try to take a minor of a non-square matrix.

6.9.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from copads.copadsexceptions.MatrixSquareError(Section 6.7)

__str__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

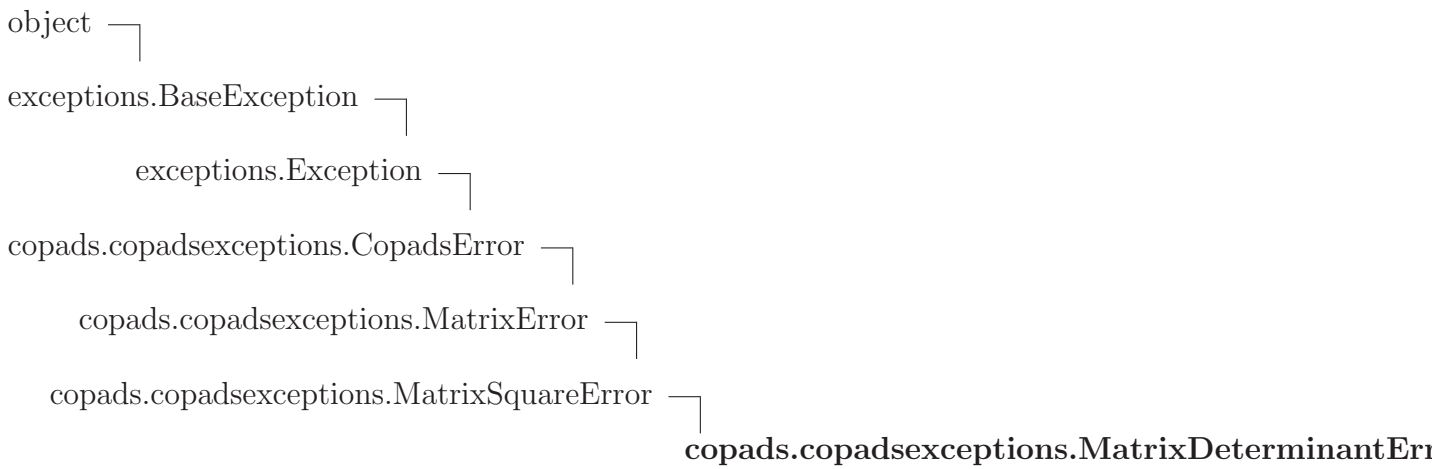
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.9.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.10 Class MatrixDeterminantError



Thrown when you try to take the determinant of a non-square matrix.

6.10.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from copads.copadsexceptions.MatrixSquareError(Section 6.7)

__str__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

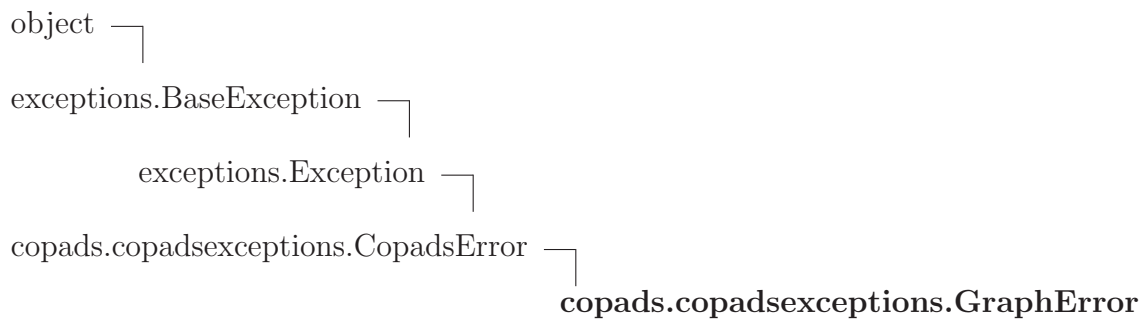
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.10.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.11 Class GraphError



Known Subclasses: copads.copadsexceptions.EdgeNotFoundError, copads.copadsexceptions.GraphEdge copads.copadsexceptions.GraphParameterError, copads.copadsexceptions.UnknownGraphMatrixError, copads.copadsexceptions.VertexNotFoundError

Abstract parent for all graph exceptions

6.11.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(),

__setattr__(), __setstate__(), __str__(), __unicode__()

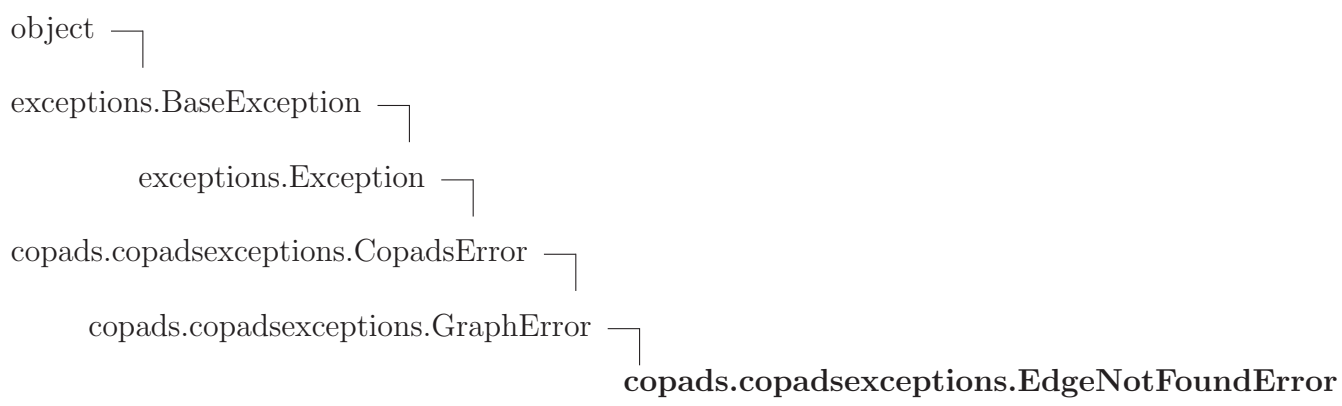
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.11.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.12 Class EdgeNotFoundError



Exception to be thrown when trying to retrieve an edge that is not found in the graph.

6.12.1 Methods

__init__(*self*, *edge*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

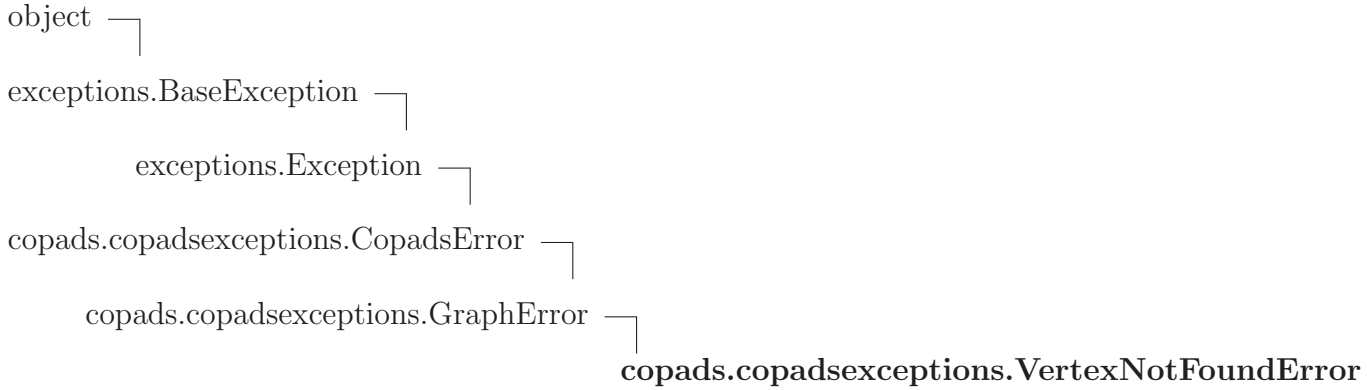
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.12.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.13 Class VertexNotFoundError



Exception to be thrown when trying to retrieve a vertex that is not found in the graph.

6.13.1 Methods

__init__(*self*, *vertex*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__**str**__(*self*)
str(x)
Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

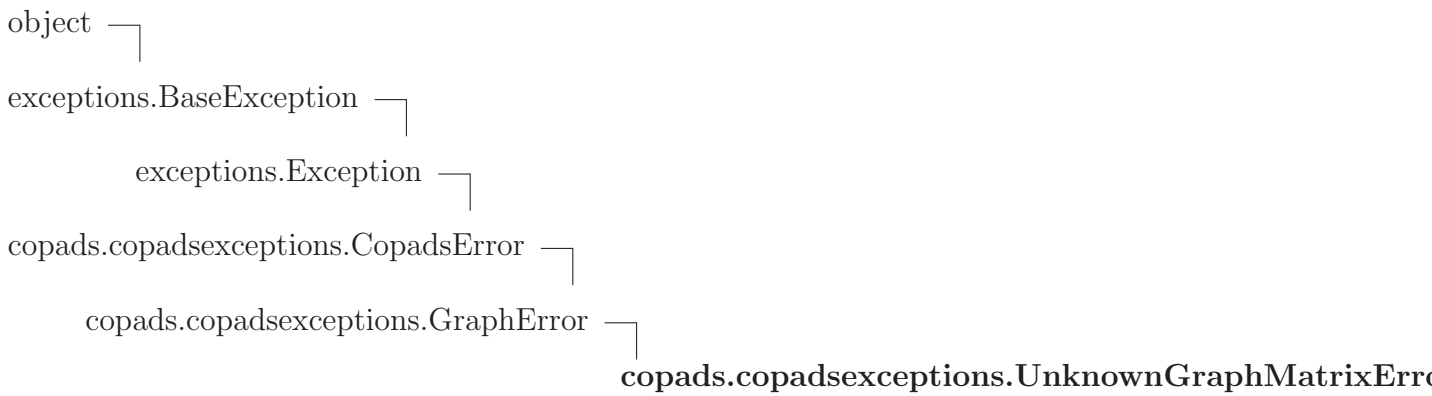
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.13.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.14 Class UnknownGraphMatrixError



Known Subclasses: copads.copadsexceptions.NotAdjacencyGraphMatrixError

Exception to be thrown when trying to use an unknown type of graph matrix.

6.14.1 Methods

__init__(*self*, *type*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

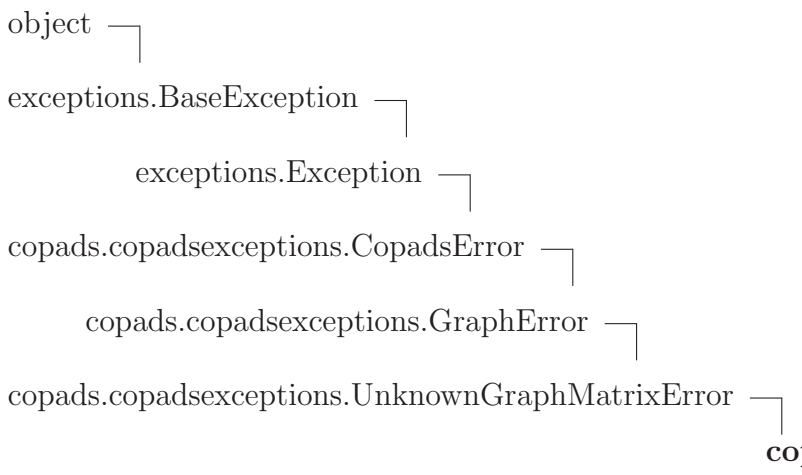
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.14.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.15 Class NotAdjacencyGraphMatrixError



Exception to be thrown when trying to enter a non-square matrix into Graph.GraphAdjacencyMatrix object or when not supplying an adjacency matrix when the calculation requires it.

6.15.1 Methods

__init__(*self*, *type*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

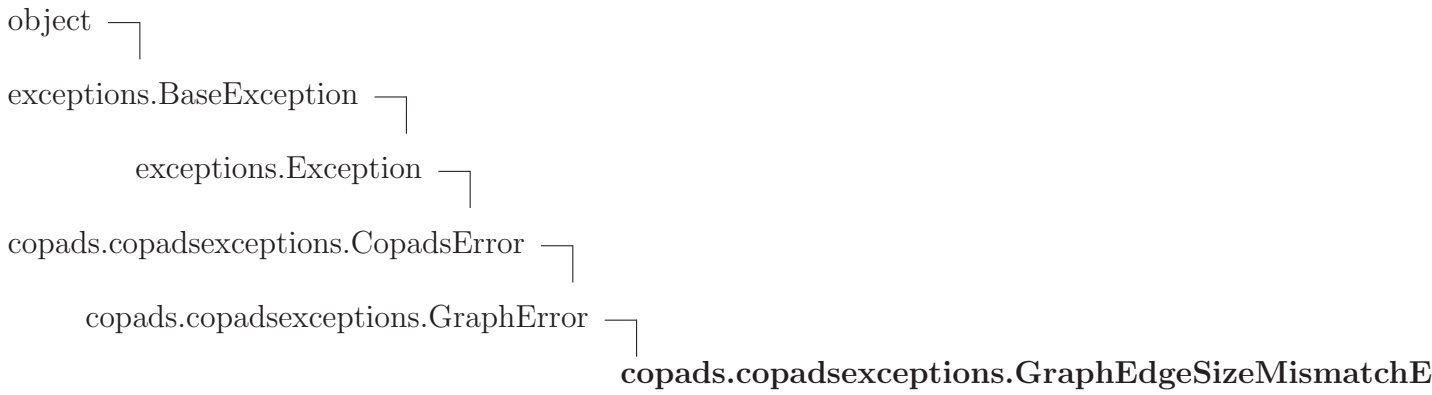
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.15.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.16 Class GraphEdgeSizeMismatchError



Exception for number of output edges do not match the number of input edges.

6.16.1 Methods

__init__(*self*, *index*, *edge*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.16.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.17 Class GraphParameterError

object
└ exceptions.BaseException
  └ exceptions.Exception
    └ copads.copadsexceptions.CopadsError
      └ copads.copadsexceptions.GraphError
        └ **copads.copadsexceptions.GraphParameterError**

Exception for parameter errors in Graph.Graph class.

6.17.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__**str**__(*self*) str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

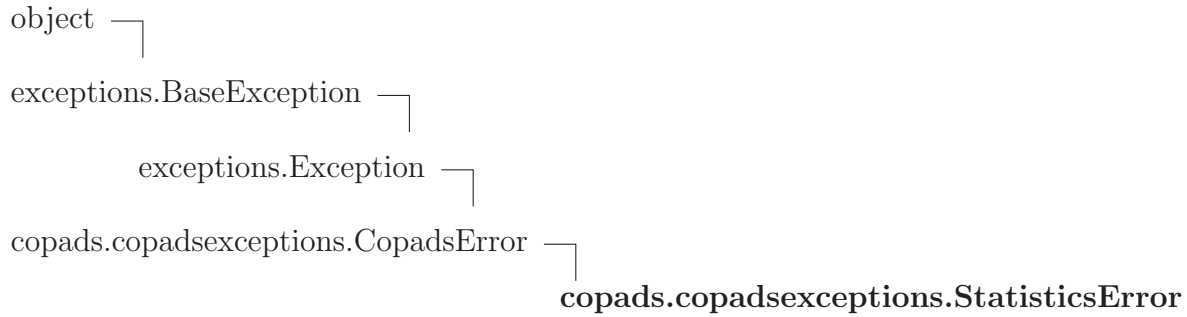
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.17.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.18 Class StatisticsError



Known Subclasses: copads.copadsexceptions.DistributionError

Abstract parent for all statistics exceptions

6.18.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

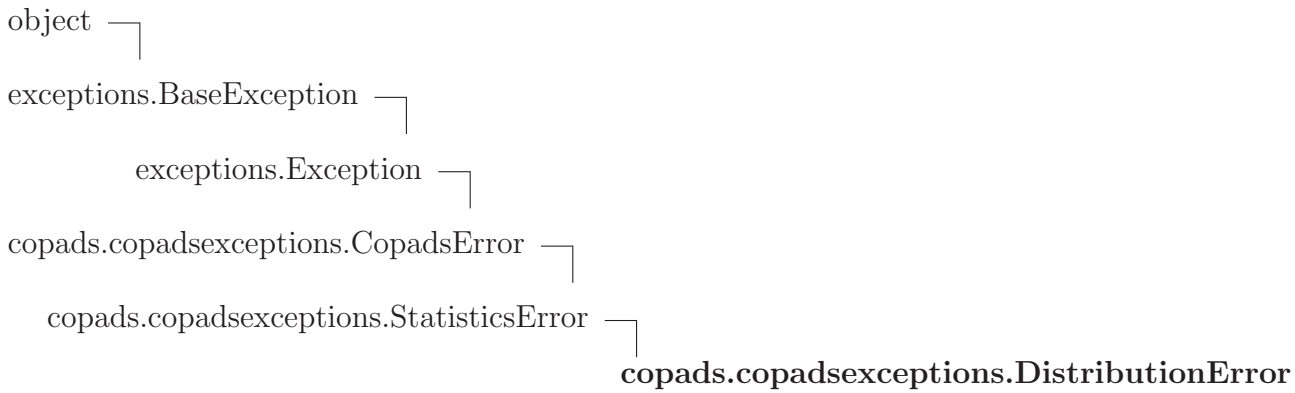
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.18.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.19 Class DistributionError



Known Subclasses: copads.copadsexceptions.DistributionFunctionError, copads.copadsexceptions.Distri
copads.copadsexceptions.NormalDistributionTypeError

Abstract parent for all exceptions pertaining to statistical distributions

6.19.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

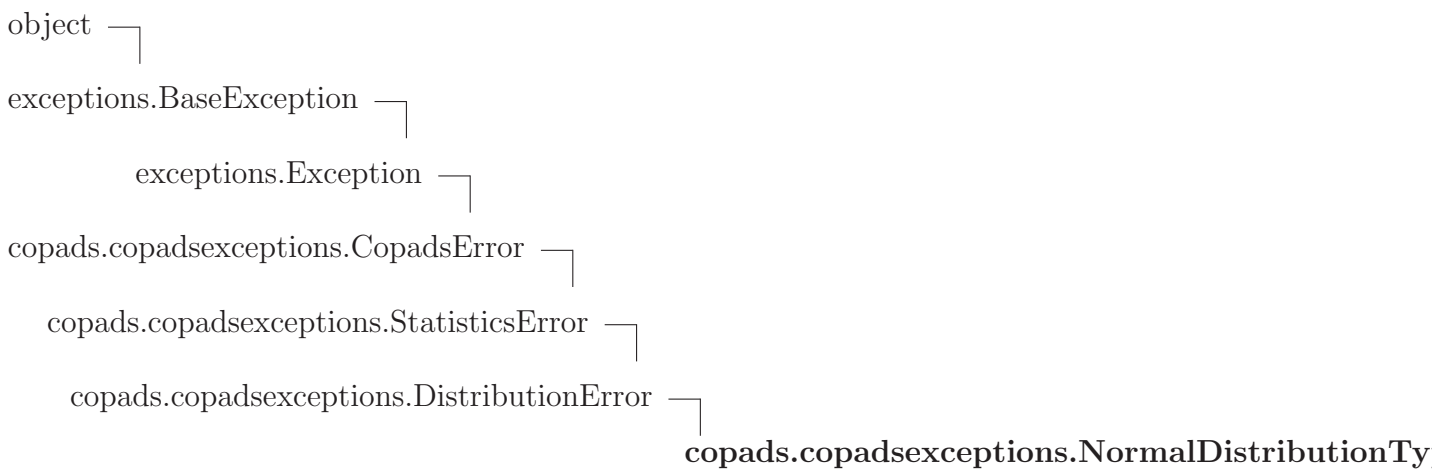
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.19.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.20 Class NormalDistributionTypeError



Exception for type errors in normal distribution (StatisticsDistribution.NormalDistribution).

6.20.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

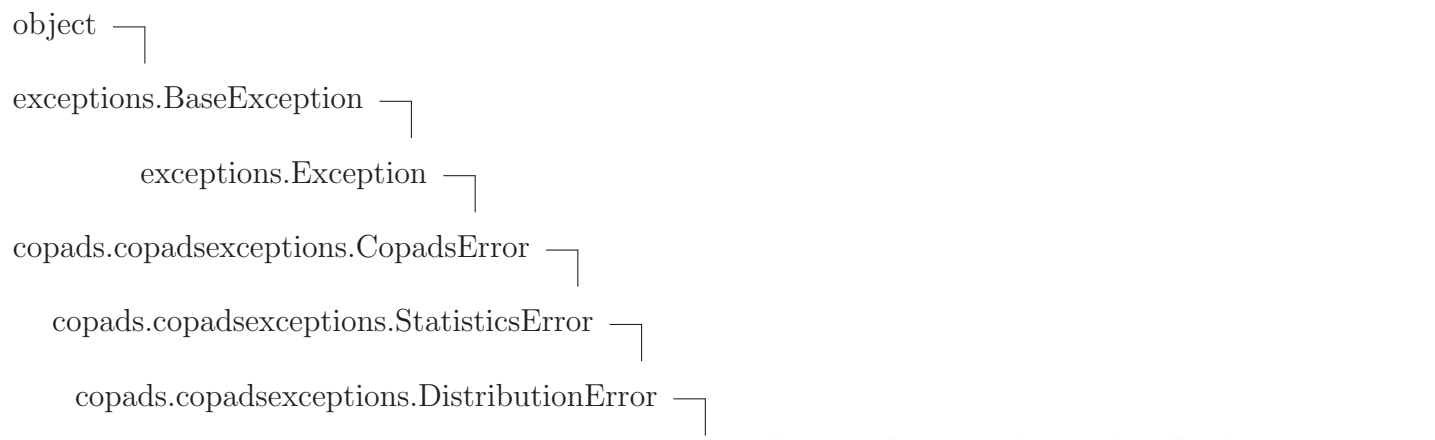
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.20.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.21 Class DistributionParameterError



Exception for parameter errors in distributions (StatisticsDistribution.*).

6.21.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

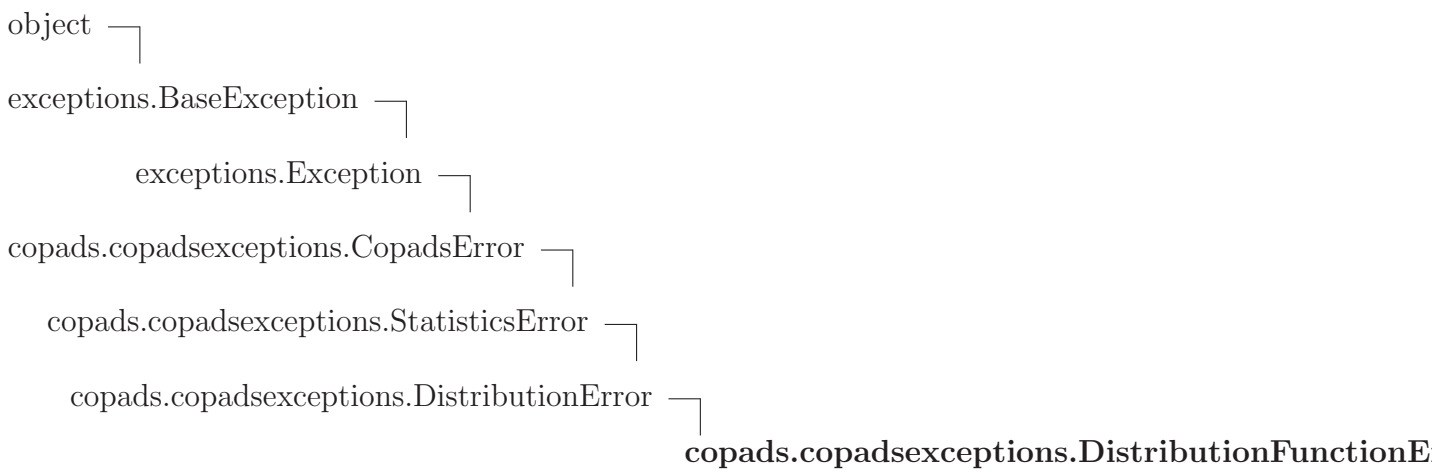
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.21.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.22 Class DistributionFunctionError



Exception for undefined functions in distributions (StatisticsDistribution.*).

6.22.1 Methods

__**init**__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__**str**__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

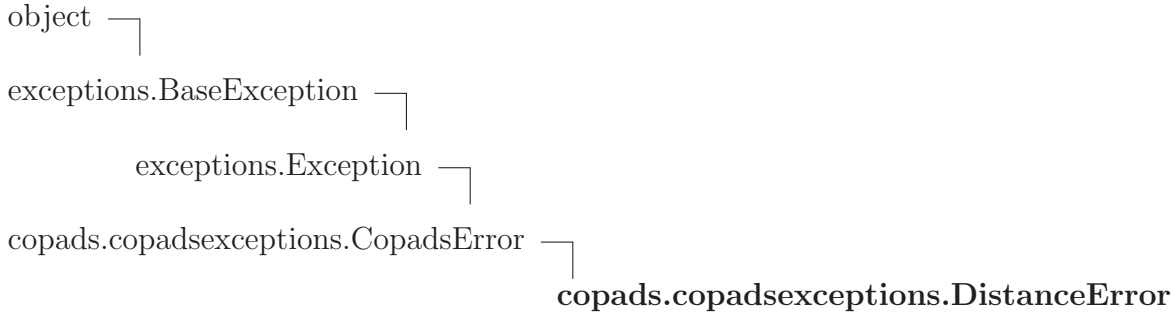
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.22.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.23 Class DistanceError



Known Subclasses: copads.copadsexceptions.DistanceInputSizeError

Abstract parent for all exceptions related to calculating distances between lists.

6.23.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

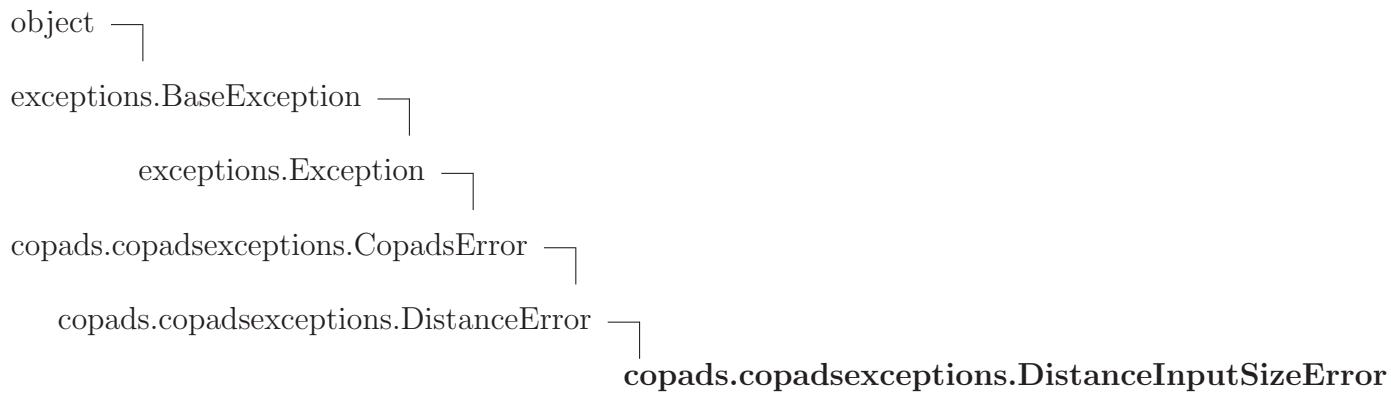
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.23.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.24 Class DistanceInputSizeError



Exception for input parameter size errors for list (object) distance routines that have specific requirements for the size of inputs.

6.24.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

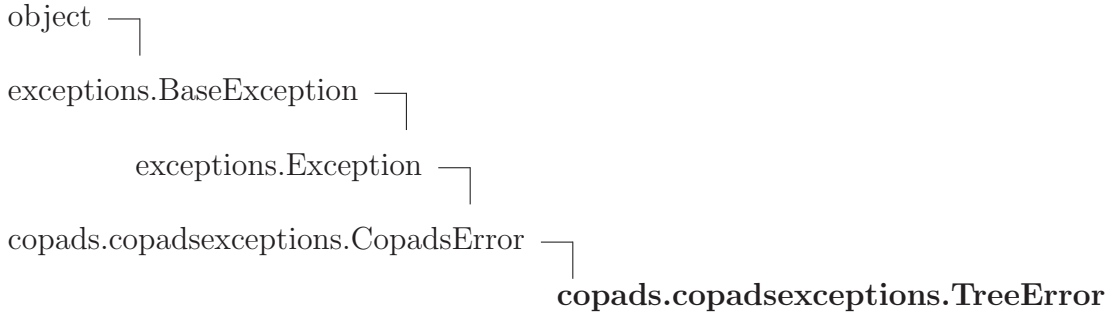
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.24.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.25 Class TreeError



Known Subclasses: copads.copadsexceptions.TreeNodeTypeError

Abstract parent for all tree exceptions

6.25.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

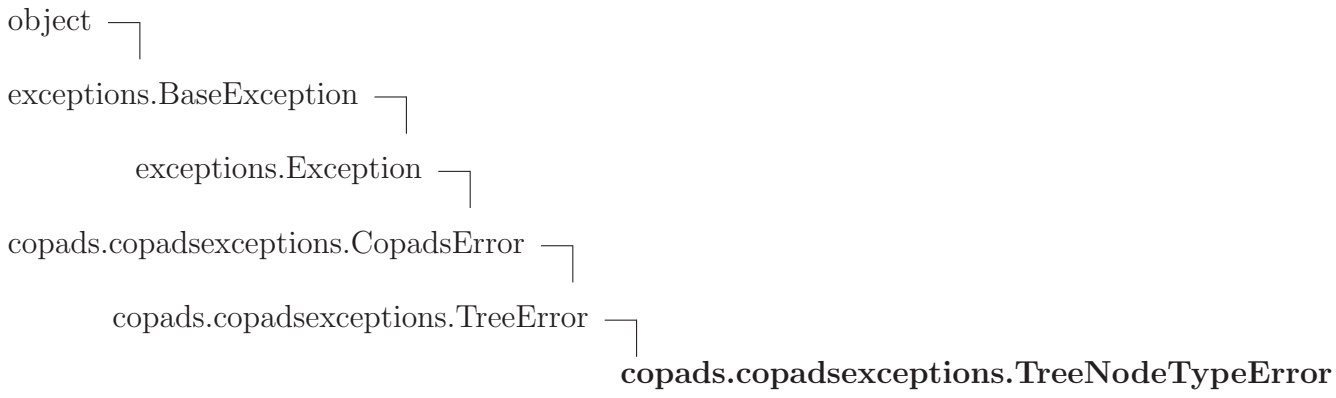
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.25.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.26 Class TreeNodeTypeError



Exception to be thrown when trying to add a non-Node class object (Tree.Node) into a Tree object (Tree.BinaryTree)

6.26.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

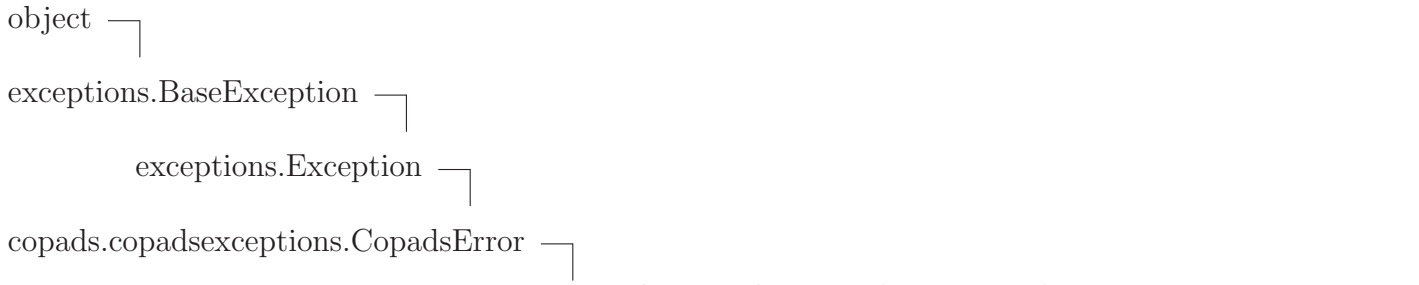
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.26.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

6.27 Class FunctionParameterTypeError



Exception to be thrown when trying a function parameter is of the wrong data type

6.27.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

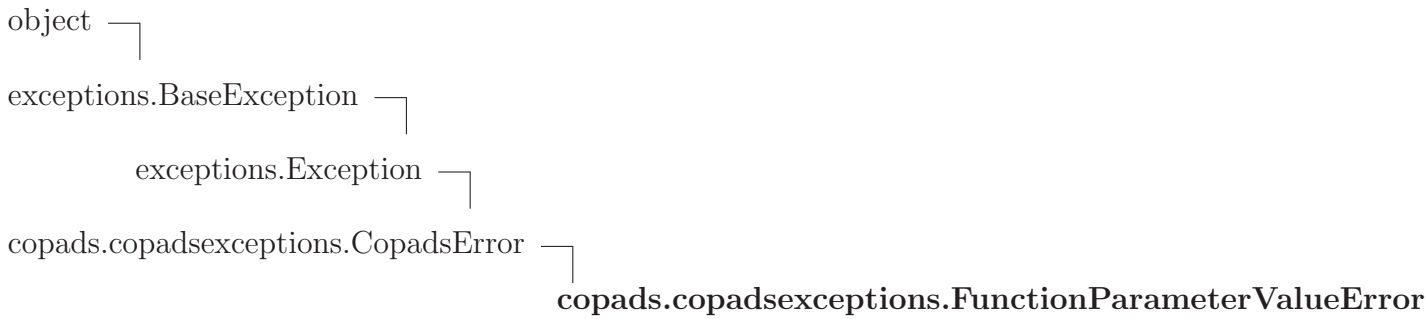
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.27.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.28 Class FunctionParameterValueError



Exception to be thrown when trying a function parameter is of wrong value

6.28.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__**str**__(*self*) str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

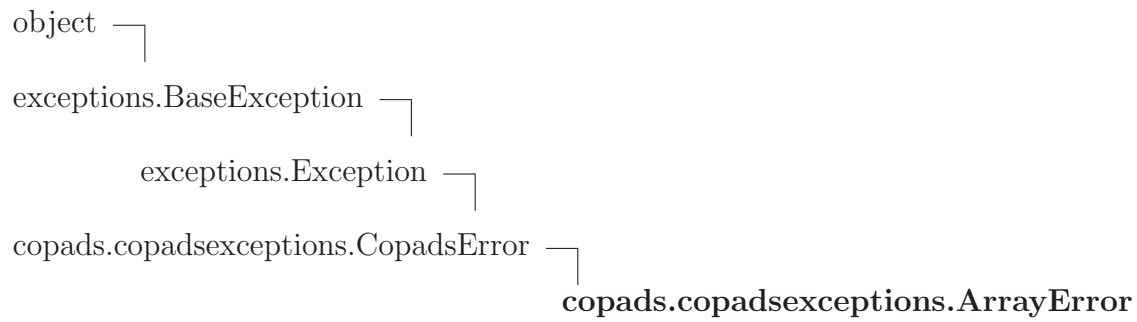
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.28.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.29 Class ArrayError



Exception to be thrown when encountered error in Array type

6.29.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

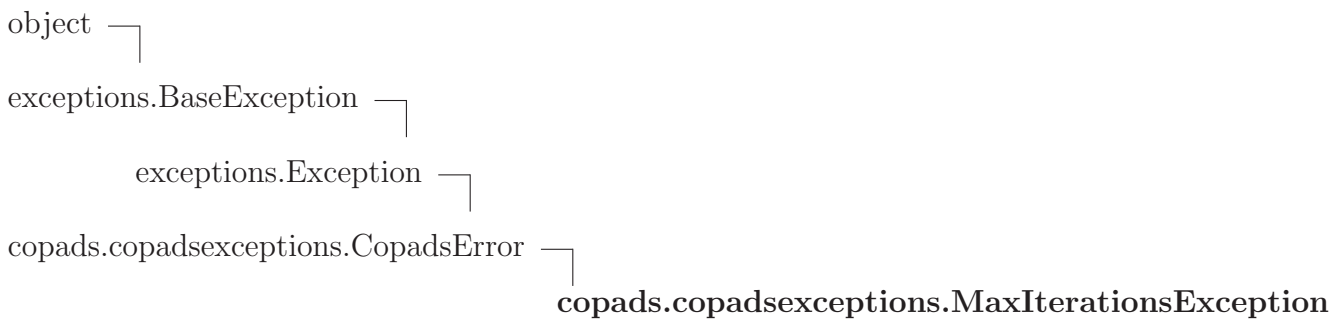
Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.29.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

6.30 Class MaxIterationsException



Exception to catch maximum looping.

6.30.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

6.30.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

7 Module copads.dataframe

A generic dataframe to hold data and analysis Date created: 24th September 2012 Licence: Python Software Foundation License version 2

7.1 Variables

Name	Description
__package__	**Value:** `'copads'`

7.2 Class dataframe

object
└
copads.dataframe.dataframe

A data frame is an encapsulation of one or more data series and its associated analyses.

7.2.1 Methods

__init__(*self*, *name*=`''`)

Constructor. Initialize data frame with a name.

Parameters

`name`: Name of this data frame

(type=string)

Overrides: object.__init__

addSeries(*self*, *name*, *data*=`[]`)

Add a data series (vector/column) to the frame.

Parameters

`name`: Name of data series

(type=string)

`data`: Data to be added

(type=list)

changeDataType(*self*, *name*, *type*='float')

Change the data type for a data series

Parameters

`name`: Name of data series

(type=string)

`type`: Type of data to cast data series into. Allowable = {float | integer | string | long}. Default = float

getSeries(*self*, *name*)

Get values of a data series.

Parameters

`name`: Name of data series

(type=string)

Return Value

List of data

has_series(*self*, *name*)

Check whether data frame has a certain data series (by name).

Parameters

`name`: Name of data series

(type=string)

Return Value

True, if the data series is present; False, if the data series is absent

changeDatum(*self*, *name*, *observation*, *newvalue*)

Change the value of a datum in a data series.

Parameters

`name`: Name of data series

(type=string)

`observation`: Observation number to change (starting count = 1)

(type=integer)

`newvalue`: new value for the observation

getObservation(*self*, *observation*)

Get values of an observation

Parameters

`observation`: Observation number to change (starting count = 1)

(type=integer)

Return Value

Dictionary of data

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.2.2 Properties

Name	Description
Inherited from object	
__class__	

7.2.3 Class Variables

Name	Description
data	**Value:** {}
analyses	**Value:** {}
data_length	**Value:** `0`
frame_name	**Value:** ''

8 Module copads.dose_entities

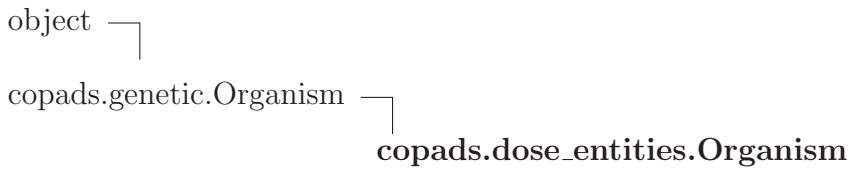
Boiler-plate codes for DOSE (digital organism simulation environment) entities Date created: 13th September 2012 Licence: Python Software Foundation License version 2

Reference: Ling, MHT. 2012. An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy. The Python Papers 7: 5.

8.1 Variables

Name	Description
Chromosome	**Value:** `g.Chromosome(initial_chromosome, ['0', '1', '2', '3', '4'...`

8.2 Class Organism



8.2.1 Methods

__init__(*self*)

Sets up a new organism with default status (age = 0, vitality = 100, lifespan = 100, fitness = 100, alive = True)

Parameters

`genome:`	list of chromosomes to inherit. Default = 'default', which will set up one default chromosome. It also allows a 'dummy' chromosome which is basically a one-base chromosome - this is for applications which does not utilize the chromosome.
`mutation_type:`	type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = point.
`additional_mutation_rate:`	probability of mutation per base above background mutation rate. Default = 0.01 (1%). No mutation event will ever happen if (rate + background_mutation) is less than zero.
`gender:`	establishes the gender of the organism which may be used for mating routines.

Overrides: object.__init__ extit(inherited documentation)

get_cytoplasm(*self*)

fitness(*self*)

Function to calculate the fitness of the current organism. **This function MUST be over-ridden by the inherited class or substituted as fitness function is highly dependent on utility.** The only requirement is that a fitness score must be returned.

Here, the sample implementation calculates fitness as proportion of the genome with '1's.

Return Value

fitness score or fitness list

Overrides: copads.genetic.Organism.fitness extit(inherited documentation)

mutation_scheme(*self*)

Function to trigger mutation events in each chromosome. **This function may be over-ridden by the inherited class or substituted to cater for specific mutation schemes but not an absolute requirement to do so.**

Both type and rate must be defined at the same time, otherwise the initiated mutation_type and additional_mutation_rate will be used.

Parameters

`type`: type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = None.

`rate`: probability of mutation per base above background mutation rate. Default = None. No mutation event will ever happen if (rate + background_mutation) is less than zero.

Overrides: copads.genetic.Organism.mutation_scheme extit(inherited documentation)

Inherited from copads.genetic.Organism(Section 12.4)

__str__(), clone(), getStatus(), setStatus()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

8.2.2 Properties

Name	Description
Inherited from object __class__	

8.2.3 Class Variables

Name	Description
cytoplasm	**Value:** `[0]* cytoplasm_size`
Inherited from copads.genetic.Organism (Section 12.4) genome, status	

8.3 Class Population

object
└
copads.genetic.Population
└
copads.dose_entities.Population

8.3.1 Methods

__**init**__(*self*, *pop_size*=`population_size`, *max_gen*=`maximum_generations`)

Establishes a population of organisms.

Parameters

`goal:`	the goal to be reached by the population.
`maxgenerations:`	maximum number of generations to evolve. Default = 'infinite'.
`agents:`	organisms making up the initial population.

Overrides: object.__init__ extit(inherited documentation)

prepopulation_control(*self*)

Function to trigger population control events before mating event in each generation (For example, to simulate pre-puberty death). **This function may be over-ridden by the inherited class or substituted to cater for specific events.** Although this is not an absolute requirement, it is extremely encouraged to prevent exhaustion of memory space. Without population control, it will seems like a reproducing immortal population.

Here, the sample implementation eliminates the bottom half of the population based on fitness score unless the number of organisms is less than 20. This is to prevent extinction. However, if the number of organisms is more than 2000 (more than 100x initial population size, a random selection of 2000 will be used for the next generation.

Overrides: copads.genetic.Population.prepopulation_control extit(inherited documentation)

mating(*self*)

Function to trigger mating events in each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific mating schemes but not an absolute requirement to do so.** Mating schemes should include

- selection of mating partners using Organism.fitness() function and/or other status
- processes and actions of mating

Here, the sample implementation randomly selects any 2 organisms from the culled list and generate a new genome for the progeny organism by single crossover.

Overrides: copads.genetic.Population.mating extit(inherited documentation)

postpopulation_control(*self*)

Function to trigger population control events after mating event in each generation(For example, to simulate old-age death). **This function may be over-ridden by the inherited class or substituted to cater for specific events.** Although this is not an absolute requirement, it is extremely encouraged to prevent exhaustion of memory space. Without population control, it will seems like a reproducing immortal population.

Overrides: copads.genetic.Population.postpopulation_control extit(inherited documentation)

generation_events(*self*)

Function to trigger other defined events in each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific events but not an absolute requirement to do so.** Events and controls may include

- processes simulating disaster or other catastrophic events
- changes in mutations

Overrides: copads.genetic.Population.generation_events extit(inherited documentation)

report(*self*)

Function to report the status of each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific reporting schemes but not an absolute requirement to do so.** At the very least, this function should report whether the goal is reached.

Return Value
dictionary of status describing the current generation

Overrides: copads.genetic.Population.report extit(inherited documentation)

Inherited from copads.genetic.Population(Section 12.5)

add_organism(), freeze(), generation_step(), revive()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.3.2 Properties

Name	Description
Inherited from object	
__class__	

8.4 Class World

object

copads.dose_world.World

copads.dose_entities.World

8.4.1 Methods

__init__(*self*, *world_x*=`world_x`, *world_y*=`world_y`, *world_z*=`world_z`)

Setting up the world and ecosystem

Parameters

`world_x`: number of ecological cells on the x-axis

`world_y`: number of ecological cells on the y-axis

`world_z`: number of ecological cells on the z-axis

Overrides: object.__init__ extit(inherited documentation)

organism_movement(*self*, *x*, *y*, *z*)

Function to trigger organism movement from current ecological cell to an adjacent ecological cell. **This function may be over-ridden by the inherited class or substituted to cater for mobility schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

`y`: location of current ecological cell on the y-axis

`z`: location of current ecological cell on the z-axis

Overrides: copads.dose_world.World.organism_movement extit(inherited documentation)

organism_location(*self*, *x*, *y*, *z*)

Function to trigger organism movement from current ecological cell to a distant ecological cell. **This function may be over-ridden by the inherited class or substituted to cater for mobility schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

`y`: location of current ecological cell on the y-axis

`z`: location of current ecological cell on the z-axis

Overrides: copads.dose_world.World.organism_location extit(inherited documentation)

ecoregulate(*self*)

Function to simulate events to the entire ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

Overrides: copads.dose_world.World.ecoregulate extit(inherited documentation)

update_ecology(*self*, *x*, *y*, *z*)

Function to process temporary_input and temporary_output from the activities of the organisms in the current ecological cell into a local ecological cell condition, and update the ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

`y`: location of current ecological cell on the y-axis

`z`: location of current ecological cell on the z-axis

Overrides: copads.dose_world.World.update_ecology extit(inherited documentation)

update_local(*self*, *x*, *y*, *z*)

Function to update local ecological cell condition from the ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

`y`: location of current ecological cell on the y-axis

`z`: location of current ecological cell on the z-axis

Overrides: copads.dose_world.World.update_local extit(inherited documentation)

report(*self*)

Function to report the status of the world and ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for specific reporting schemes but not an absolute requirement to do so.**

Return Value

dictionary of status describing the current generation

Overrides: copads.dose_world.World.report extit(inherited documentation)

Inherited from copads.dose_world.World(Section 11.2)

eco_burial(), eco_excavate()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.4.2 Properties

Name	Description
Inherited from object	
__class__	

8.4.3 Class Variables

Name	Description
Inherited from copads.dose_world.World (Section 11.2)	
ecosystem	

9 Module copads.dose_executor

Default DOSE simulation runner in manuscript [1] Date created: 13th September 2012 Licence: Python Software Foundation License version 2

[1] Ling, MHT. 2012. An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy. The Python Papers 7: 5.

9.1 Functions

set_instruction_version(*ragaraja_version*, *instruction_set*='ragaraja_instructions.txt')

Set active / usable set of Ragaraja instructions / operations for the current simulation.

Parameters

`ragaraja_version:` defines the version of Ragaraja instructions to be used. Allowed values are {0 | 0.1 | 1}

`instruction_set:` file name for user-defined instruction set usage. This will only be used when ragaraja_version=0. Default = ragaraja_instructions.txt. Please see ragaraja_instructions.txt for format.

(type=string)

Return Value

dictionary of ragaraja instructions as keys and instruction execution functions as values.

Since: version 0.4.1

write_parameters()

Write parameters into file.

Since: version 0.4.1

simulate(*entity_module*)

Simulate the entities in DOSE.

Parameters

`entity_module`: python module of entities to execute. Required classes in entity_module are Chromosome (inherited from genetic.Chromosome), Organism (inherited from genetic.Organism), Population (inherited from genetic.Population) and World (inherited from dose_world.World).

Since: version 0.4.1

9.2 Variables

Name	Description
__package__	**Value:** `'copads'`

10 Module copads.dose_parameters

Parameter file for DOSE simulation. Date created: 13th September 2012 Licence: Python Software Foundation License version 2

This file contains the default parameters for DOSE simulation.

10.1 Variables

Name	Description
initial_chromosome	Probability of number of mutations per base. For example, 0.1 means 10% of the chromosome will be mutated per generation. **Value:** `['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '...`
background_mutation_rate	Probability of mutation per base above additional_mutation_rate. No mutation event will ever happen if (additional_mutation_rate + additional_mutation_rate) is less than zero. **Value:** `0.1`
additional_mutation_rate	Size of cytoplasm (length of list) for the ancestor organism. **Value:** `0`
cytoplasm_size	Maximum size of cytoplasm (length of list) allowable. Some Ragaraja instructions can increase cytoplasms size, equivalent to cell enlargement. **Value:** `50`
max_cytoplasm_size	If 'True', a clean cytoplasm will be used to execute the genome. If 'False', the cytoplasm will be reused in the next generation. **Value:** `200`
clean_cytoplasm	Maximum number of codons to evaluate. Therfore, the maximum evaluated genome is 3*max_codon bases. **Value:** `True`
max_codon	List containing name(s) of population. The number of names = the number of populations to create. **Value:** `2000`
population_names	Number of organism per population. **Value:** `['pop1', 'pop2']`

continued on next page

Name	Description
population_size	Number of ecological cells in the world in (x,y,z) coordinate as (world_x, world_y, world_z). **Value:** `100`
world_x	**Value:** `5`
world_y	**Value:** `5`
world_z	List of tuple indicating the location on ecosystem to map the population(s) and will be mapped in the same order as population_names. For example, if population_names = ['pop1', 'pop2'] and population_locations = [(0,0,0), (4,4,4)], population 'pop1' will be in (0,0,0) and population 'pop2' will be in (4,4,4). **Value:** `5`
population_locations	Maximum number of generations to simulate. **Value:** `[(0, 0, 0), (4, 4, 4)]`
maximum_generations	Number of generations between each freezing/fossilization event. Freezing/fossilization event is similar to glycerol stocking in microbiology. **Value:** `500`
fossilized_frequency	Proportion of population to freeze/fossilize. If the population size or the preserved proportion is below 100, the entire population will be preserved. **Value:** `100`
fossilized_ratio	Dictionary containing prefix of file names for freezing/fossilization. The preserved sample will be written into a file with name in the following format - <prefix>_<generation count>_<sample size>.gap **Value:** `0.01`
fossil_files	Number of generations between printing of reports (based on Population.report function) into files. The result file format is <UTC date time stamp> <current generation count> <output from Population.report function>. **Value:** `{'pop1': 'pop1', 'pop2': 'pop2'}`
print_frequency	Dictionary containing file names for result files. File name will be <file names>.result **Value:** `10`

continued on next page

Name	Description
result_files	Version of Ragaraja instruction set to use. **Value:** {'pop1': 'pop1', 'pop2': 'pop2'}
ragaraja_version	File containing Ragaraja instruction set to be used. This option is only effective when ragaraja_version = 0. Format of file is <instruction>={Y|N} where "Y" = instruction to be used and "N" = instruction not to be used. **Value:** 0.1
user_defined_instructions	Number of generations between each burial/preservation of ecosystem. **Value:** 'ragaraja_instructions.txt'
eco_buried_frequency	Prefix of file name for ecosystem burial/preservation. The preserved ecosystem will be written into a file with name in the following format - <prefix>_<generation count>.eco **Value:** 500
eco_burial_file	**Value:** 'eco'
__package__	**Value:** None

11 Module copads.dose_world

World structure for DOSE (digital organism simulation environment) Date created: 13th September 2012 Licence: Python Software Foundation License version 2

Reference: Ling, MHT. 2012. An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy. The Python Papers 7: 5.

11.1 Variables

Name	Description
__package__	**Value:** `'copads'`

11.2 Class World

object
└
copads.dose_world.World

Representation of a 3-dimensional ecological world.

The ecosystem is made up of ecological cells. Each ecological cell is modelled as a dictionary of

- local_input: A list containing processed input, representing the partial local ecological condition, to be used as input to the organisms in the current ecological cell. This is updated by World.update_local function.
- local_output: A list containing processed output, representing the partial local ecological condition. This is updated by World.update_local function.
- temporary_input: A list acting as temporary holding for input after being fed to the organisms in the current ecological cell, which is to be used to update local_input and local_output lists by World.update_local and World.update_ecology functions.
- temporary_output: A list acting as temporary holding for output from the organisms in the current ecological cell, which is to be used to update local_input and local_output lists by World.update_local and World.update_ecology functions.
- organisms: The number of organisms in the current ecological cell which is updated by World.organism_movement and World.organism_location functions.

See Also: Ling, MHT. 2012. An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy. The Python Papers 7: 5.

11.2.1 Methods

__init__(*self*, *world_x*, *world_y*, *world_z*)

Setting up the world and ecosystem

Parameters

`world_x`: number of ecological cells on the x-axis
(type=integer)

`world_y`: number of ecological cells on the y-axis
(type=integer)

`world_z`: number of ecological cells on the z-axis
(type=integer)

Overrides: object.__init__

eco_burial(*self*, *filename*)

Function to preserve the entire ecosystem.

Parameters

`filename`: file name of preserved ecosystem.

eco_excavate(*self*, *filename*)

Function to excavate entire ecosystem.

Parameters

`filename`: file name of preserved ecosystem.

ecoregulate(*self*)

Function to simulate events to the entire ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

organism_movement(*self*, *x*, *y*, *z*)

Function to trigger organism movement from current ecological cell to an adjacent ecological cell. **This function may be over-ridden by the inherited class or substituted to cater for mobility schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

(type=integer)

`y`: location of current ecological cell on the y-axis

(type=integer)

`z`: location of current ecological cell on the z-axis

(type=integer)

organism_location(*self*, *x*, *y*, *z*)

Function to trigger organism movement from current ecological cell to a distant ecological cell. **This function may be over-ridden by the inherited class or substituted to cater for mobility schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

(type=integer)

`y`: location of current ecological cell on the y-axis

(type=integer)

`z`: location of current ecological cell on the z-axis

(type=integer)

update_ecology(*self*, *x*, *y*, *z*)

Function to process temporary_input and temporary_output from the activities of the organisms in the current ecological cell into a local ecological cell condition, and update the ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

(type=integer)

`y`: location of current ecological cell on the y-axis

(type=integer)

`z`: location of current ecological cell on the z-axis

(type=integer)

update_local(*self*, *x*, *y*, *z*)

Function to update local ecological cell condition from the ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for ecological schemes but not an absolute requirement to do so.**

Parameters

`x`: location of current ecological cell on the x-axis

(type=integer)

`y`: location of current ecological cell on the y-axis

(type=integer)

`z`: location of current ecological cell on the z-axis

(type=integer)

report(*self*)

Function to report the status of the world and ecosystem. **This function may be over-ridden by the inherited class or substituted to cater for specific reporting schemes but not an absolute requirement to do so.**

Return Value

dictionary of status describing the current generation

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.2.2 Properties

Name	Description
Inherited from object __class__	

11.2.3 Class Variables

Name	Description
ecosystem	**Value:** {}

12 Module copads.genetic

Framework for Genetic Algorithm Applications. Date created: 23rd February 2010 Licence: Python Software Foundation License version 2

See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

12.1 Functions

crossover(*chromosome1*, *chromosome2*, *position*)

Cross-over operator - swaps the data on the 2 given chromosomes after a given position.

Parameters

`chromosome1:` Chromosome object

`chromosome2:` Chromosome object

`position:` base position of the swap over
(type=integer)

Return Value

(resulting chromosome1, resulting chromosome2)

Since: version 0.4

population_constructor(*data*={'additional_mutation_rate': 0.01, 'background_mutation':...})

Function to construct a population based on a dictionary of population data.

Population data contains the following keys:

- 'nucleotide_list' = List of allowable nucleotides (bases). Default = [1, 2, 3, 4].
- 'chromosome_length' = Length of a chromosome. Default = 200.
- 'chromosome_type' = Type of chromosome. Default = 'defined'.
- 'chromosome' = Initial chromosome. Default = [1] * 200.
- 'background_mutation' = Background mutation rate. Default = 0.0001 (0.01%).
- 'genome_size' = Number of chromosomes per organism. Default = 1.
- 'population_size' = Size of initial population (number of organisms). Default = 200.
- 'fitness_function' = Fitness evaluation function. Accepts 'default' or a function. Please refer to Organism. Default = 'default'.
- 'mutation_scheme' = Function simulating the mutation scheme of an organism. Accepts 'default' or a function. Please refer to Organism. Default = 'default'.
- 'additional_mutation_rate' = Mutation rate on top of background mutation rate. Default = 0.01 (1%).
- 'mutation_type' = Type of default mutation. Default = 'point'.
- 'goal' = Goal of the population, as evaluated by fitness function. Default = 4.
- 'maximum_generation' = Number of generations to simulate. Accepts an integer or 'infinite'. Default = 'infinite'.
- 'prepopulation_control' = Function simulating pre-mating population control. Please refer to Population. Default = 'default'.
- 'mating' = Function simulating mating procedure (mate selection and act of mating control. Please refer to Population. Default = 'default'.
- 'postpopulation_control' = Function simulating post-mating population control. Please refer to Population. Default = 'default'.
- 'generation_events' = Function simulating other possible (usually rare or random) events in the generation. Please refer to Population. Default = 'default'.
- 'report' = Function to generate the status report of the generation. Please refer to Population. Default = 'default'.

Parameters

`data`: population data

(type=dictionary)

Return Value

Population object

See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL,

population_simulate(*population*, *printfreq*=100, *freezefreq*='never', *freezefile*='pop', *freezeproportion*=0.01, *resultfile*='result.txt')

Function to simulate the population - start the GA.

Parameters

`population`: Population to run.

(type=Population object)

`printfreq`: Reporting intervals on screen. Default = 100.

`freezefreq`: Generation intervals to freeze population into a file. See Population.freeze method. Accepts an integer or 'never'. Default = 'never'.

`freezefile`: Prefix of file name of frozen population. See Population.freeze method. Default = 'pop'.

`freezeproportion`: Proportion of population to be preserved. See Population.freeze method. Default = 0.01, preserves 1% of the population.

`resultfile`: Name of file to print out results of each generation. Format of output is dependent on reporting method of the population (Population.report). If 'None', it will print out the results into a file. Default = 'result.txt'.

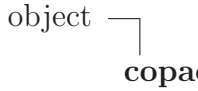
See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

Since: version 0.4

12.2 Variables

Name	Description
population_data	**Value:** {'additional_mutation_rate': 0.01, 'background_mutation':...
__package__	**Value:** 'copads'

12.3 Class Chromosome



copads.genetic.Chromosome

Representation of a linear chromosome.

See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

Since: version 0.4

12.3.1 Methods

__init__(*self*, *sequence*, *base*, *background_mutation*=`0.0001`)

Sets up a chromosome.

Parameters

`sequence:`	a subscriptable object (list or string) representing the sequence of the chromosome.
`base:`	a subscriptable object (list or string) representing allowable entities in the sequence.
`background_mutation:`	background mutation rate represented as the probability of number of mutations per base. Default = 0.0001 (0.01%).

Overrides: object.__init__

Since: version 0.4

rmutate(*self*, *type*='point', *rate*=0.01, *start*=0, *end*=-1)

Random Mutation operator - to simulate random point, insertion, deletion, inversion, gene translocation and gene duplication events.

The start and end parameters are useful for simulating mutational hotspots in the genome.

Parameters

`type:` type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = point.

`rate:` probability of mutation per base above background mutation rate. Default = 0.01 (1%). No mutation event will ever happen if (rate + background_mutation) is less than zero.

`start:` starting base on the sequence for mutation. Default = 0, start of the genome.

`end:` last base on the sequence for mutation. Default = -1, end of the genome.

Since: version 0.4

kmutate(*self*, *type*='point', *start*=0, *end*=0, *sequence*=None, *tpos*=0)

Known Mutation operator - to simulate a known point, insertion, deletion, inversion, gene translocation or gene duplication event.

The required parameters will be determined by the type of mutation:

- point requires
  - start (the position of the base to mutate)
  - sequence (the new base)
- delete requires
  - start (the position of the base to delete)
- insert requires
  - start (the position of the base to begin insertion)
  - sequence (list of sequence or a base to insert)
- invert requires
  - start (the position of the base to start inversion)
  - end (the position of the base to end inversion)
- duplicate requires
  - start (the position of the starting base to duplicate)
  - end (the position of the last base to duplicate)
- translocate requires
  - start (the position of the base to start translocation)
  - end (the position of the base to end translocation)
  - tpos (the position of the base insert the translocated sequence)

Parameters

`type:` type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = point.

`start:` starting base on the chromosome for mutation. Default = 0, start of the genome.

`end:` last base on the chromosome for mutation. Default = 0.

`sequence:` list of bases to change to (point mutation) or to insert (insertion mutation)

`tpos:` position of the chromosome to insert the translocated sequence.

Since: version 0.4

replicate(*self*)

Replicates (deep copy) the chromosome.

Return Value
a copy of current chromosome.

Since: version 0.4

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.3.2 Properties

Name	Description
Inherited from object	
__class__	

12.4 Class Organism

object
└ **copads.genetic.Organism**

An organism represented by a list of chromosomes and a status table. This class should almost never be instantiated on its own but acts as an ancestor class as some functions need to be over-ridden in the inherited class for the desired use.

Each organism is identifiable by a randomly generated 32-character 'name' as Organism.identity.

Methods to be over-ridden in the inherited class or substituted are

- fitness
- mutation_scheme

Pre-defined status are

1. alive - is the organism alive? True or False.
2. vitality - percentage of maximum vitality.
3. age - the current age of the organism.
4. lifespan - pre-defined maximum lifespan to be set based on scenario.
5. fitness - how fit the organism is? Set to maximum fitness.
6. death - reason of death (as death code).

List of defined death codes

1. death01 - zero vitality
2. death02 - maximum age reached
3. death03 - unknown death cause

See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

Since: version 0.4

12.4.1 Methods

__init__(*self*, *genome*='default', *mutation_type*='point', *additional_mutation_rate*=0.01, *gender*=None)

Sets up a new organism with default status (age = 0, vitality = 100, lifespan = 100, fitness = 100, alive = True)

Parameters

genome:	list of chromosomes to inherit. Default = 'default', which will set up one default chromosome. It also allows a 'dummy' chromosome which is basically a one-base chromosome - this is for applications which does not utilize the chromosome.
mutation_type:	type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = point.
additional_mutation_rate:	probability of mutation per base above background mutation rate. Default = 0.01 (1%). No mutation event will ever happen if (rate + background_mutation) is less than zero.
gender:	establishes the gender of the organism which may be used for mating routines.

Overrides: object.__init__

Since: version 0.4

fitness(*self*)

Function to calculate the fitness of the current organism. **This function MUST be over-ridden by the inherited class or substituted as fitness function is highly dependent on utility.** The only requirement is that a fitness score must be returned.

Here, the sample implementation calculates fitness as proportion of the genome with '1's.

Return Value

fitness score or fitness list

Since: version 0.4

mutation_scheme(*self*, *type*=`None`, *rate*=`None`)

Function to trigger mutation events in each chromosome. **This function may be over-ridden by the inherited class or substituted to cater for specific mutation schemes but not an absolute requirement to do so.**

Both type and rate must be defined at the same time, otherwise the initiated mutation_type and additional_mutation_rate will be used.

Parameters

`type`: type of mutation. Accepts 'point' (point mutation), 'insert' (insert a base), 'delete' (delete a base), 'invert' (invert a stretch of the chromosome), 'duplicate' (duplicate a stretch of the chromosome), 'translocate' (translocate a stretch of chromosome to another random position). Default = None.

`rate`: probability of mutation per base above background mutation rate. Default = None. No mutation event will ever happen if (rate + background_mutation) is less than zero.

Since: version 0.4

setStatus(*self*, *variable*, *value*)

Sets new status or change status of the organism. However, the following status change will result in death of the organism

1. 'alive' to False
2. 'vitality' to or below zero
3. 'age' to or above lifespan

Parameters

`variable:` name of status to change

`value:` new value of the status

Since: version 0.4

getStatus(*self*, *variable*)

Returns a status variable of the current organism

Parameters

`variable:` name of the status variable

Return Value

status or a KeyError if status is not found

Since: version 0.4

__**str**__(*self*)

Returns the genome of the organism

Overrides: object.__str__

clone(*self*)

Cloness (deep copy) the organism.

Return Value

a copy of current organism.

Since: version 0.4

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

12.4.2 Properties

continued on next page

Name	Description

Name	Description
Inherited from object	
__class__	

12.4.3 Class Variables

Name	Description
status	**Value:** {'age': 0.0, 'alive': True, 'death': None, 'fitness': 100...
genome	**Value:** []

12.5 Class Population

object
└
copads.genetic.Population

Representation of a population as a list of organisms. The entire population is in memory.

Methods to be over-ridden in the inherited class or substituted are

- prepopulation_control
- mating
- postpopulation_control
- generation_events
- report

See Also: Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

Since: version 0.4

12.5.1 Methods

__init__(*self*, *goal*, *maxgenerations*=`'infinite'`, *agents*=`[]`)

Establishes a population of organisms.

Parameters

`goal:`	the goal to be reached by the population. *(type=the return type of Organism.fitness())*
`maxgenerations:`	maximum number of generations to evolve. Default = 'infinite'.
`agents:`	organisms making up the initial population. *(type=list of Organism objects)*

Overrides: object.__init__

Since: version 0.4

prepopulation_control(*self*)

Function to trigger population control events before mating event in each generation (For example, to simulate pre-puberty death). **This function may be over-ridden by the inherited class or substituted to cater for specific events.** Although this is not an absolute requirement, it is extremely encouraged to prevent exhaustion of memory space. Without population control, it will seems like a reproducing immortal population.

Here, the sample implementation eliminates the bottom half of the population based on fitness score unless the number of organisms is less than 20. This is to prevent extinction. However, if the number of organisms is more than 2000 (more than 100x initial population size, a random selection of 2000 will be used for the next generation.

Since: version 0.4

mating(*self*)

Function to trigger mating events in each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific mating schemes but not an absolute requirement to do so.** Mating schemes should include

- selection of mating partners using Organism.fitness() function and/or other status
- processes and actions of mating

Here, the sample implementation randomly selects any 2 organisms from the culled list and generate a new genome for the progeny organism by single crossover.

Since: version 0.4

postpopulation_control(*self*)

Function to trigger population control events after mating event in each generation(For example, to simulate old-age death). **This function may be over-ridden by the inherited class or substituted to cater for specific events.** Although this is not an absolute requirement, it is extremely encouraged to prevent exhaustion of memory space. Without population control, it will seems like a reproducing immortal population.

Since: version 0.4

generation_events(*self*)

Function to trigger other defined events in each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific events but not an absolute requirement to do so.** Events and controls may include

- processes simulating disaster or other catastrophic events
- changes in mutations

Since: version 0.4

report(*self*)

Function to report the status of each generation. **This function may be over-ridden by the inherited class or substituted to cater for specific reporting schemes but not an absolute requirement to do so.** At the very least, this function should report whether the goal is reached.

Return Value

dictionary of status describing the current generation

Since: version 0.4

generation_step(*self*)

Function to simulate events for one generation. These includes

- mating (according to the mating scheme or function)
- mutating each organism in the population
- population size control
- other events defined under generation_events function
- increment of generation count
- reporting the population status

Return Value

information returned from report function.

Since: version 0.4

add_organism(*self*, *organism*)

Add a new organism(s) to the population.

Parameters

`organism`: list of new Organism object(s)

Since: version 0.4

freeze(*self*, *prefix*=`'pop'`, *proportion*=`0.01`)

Preserves part or the entire population. If the population size or the preserved proportion is below 100, the entire population will be preserved. The preserved sample will be written into a file with name in the following format - <prefix><generation count>_<sample size>.gap

Parameters

`prefix`: prefix of file name. Default = 'pop'.

`proportion`: proportion of population to be preserved. Default = 0.01, preserves 1% of the population.

Since: version 0.4

revive(*self*, *filename*, *type*='replace')

Revives a frozen population.

Parameters

`filename`: file name of frozen population (generated by Population.freeze() function)

`type`: type of revival. Allows 'replace' (replace the current population with the revived population) or 'add' (add the revived population to the current population). Default = 'replace'.

Since: version 0.4

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.5.2 Properties

Name	Description
Inherited from object	
__class__	

13 Module copads.graph

Graph Data Structures and Algorithms.

Date created: 17th August 2005

13.1 Variables

Name	Description
__package__	**Value:** `'copads'`

13.2 Class Graph

Graph data structure

13.2.1 Methods

__init__(*self*, ***kwarg*)

Initialization method. It can accept the following keyword parameters:

Parameters

`adjacency:` adjacency matrix, with vertices as the first row - row count is 1 more than column count

(type=list of list)

`digraph:` state whether the input values is to construct a graph (True for directional graph). Default = False.

(type=boolean)

`edges:` list of edges where each tuple is (<source>, <destination>). Uses digraph parameter.

(type=list of 2-element tuples)

`graph:` graph as {<source> : <destination dictionary>} where <destination dictionary> ::= {<destination> : <attribute>}.

(type=dictionary of dictionary)

`vertices:` vertices (nodes).

(type=list)

makeGraphFromAdjacency(*self*, *adj*)

Constructs a graph from an adjacency (adj) matrix, which is given as a list of list (rows). The first row of the matrix contains a list of the vertices; hence, there will be n+1 rows and n-columns in the given matrix.

Parameters

`adj`: adjacency matrix, with vertices as the first row - row count is 1 more than column count

(type=list of list)

makeGraphFromVertices(*self*, *vertices*)

Initialize a list of nodes (vertices) without edges.

Parameters

`vertices`: list of vertices

Status: Tested method

Since: version 0.1

makeGraphFromEdges1(*self*, *edges*)

Constructs a directional graph from edges (a list of tuple). Each tuple contains 2 vertices. For example, P -> Q is written as ('P', 'Q').

Parameters

`edges`: edges

(type=list of 2-element tuple)

Status: Tested method

Since: version 0.1

makeGraphFromEdges2(*self*, *edges*)

Constructs an un-directional graph from edges (a list of tuple). Each tuple contains 2 vertices. An un-directional graph is implemented as a directional graph where each edges runs both directions.

Parameters

`edges`: list of edges

(type=list of 2-element tuples)

isVertices(*self*, *vlist*)

Checks whether each element in vlist is a vertex (node) of the graph.

Parameters

`vlist:` list of vertices

Return Value

dictionary of <element of vlist> : <True | False>

Dijkstra(*self*, *start*, *end*=`None`)

Find shortest paths from the start vertex to all vertices nearer than or equal to the end.

Dijkstra's algorithm is only guaranteed to work correctly when all edge lengths are positive. This code does not verify this property for all edges (only the edges seen before the end vertex is reached), but will correctly compute shortest paths even for some graphs with negative edges, and will raise an exception if it discovers that a negative edge has caused it to make a mistake.

Parameters

`start:` vertex of starting point

`end:` vertex of ending point

Status: Tested method (by proxy from testing shortestPath method)

Since: version 0.1

shortestPath(*self*, *start*, *end*)

Find a single shortest path from the given start vertex to the given end vertex. The output is a list of the vertices in order along the shortest path.

Parameters

`start:` vertex of starting point

`end:` vertex of ending point

Status: Tested method

Since: version 0.1

RandomGraph(*self*, *nodes*, *edges*, *maxweight*=`100.0`)

Generates a graph of random edges.

Parameters

`nodes:` list of nodes or number of nodes in the random graph

`edges:` number of edges to generate in the random graph
(type=integer)

`maxweight:` maximum weight of each edge. default = 100.0
(type=float)

13.2.2 Class Variables

Name	Description
graph	**Value:** {}

14 Module copads.hash

Hash Generators Date created: 16th April 2013 Licence: Python Software Foundation License version 2

14.1 Functions

forward_file_hash(*f*, *fsize*, *start*, *blocksize*, *algorithm*)

Forward file hashing. It takes file and generate repeated hashes to the end of the file in blocks.

Algorithm:

1. block <- file from position N to N + blocksize
2. hash <- generate hash from block
3. N = N + blocksize
4. block <- file from position N to N + blocksize
5. block = block + hash
6. hash <- generate hash from block
7. Repeat steps 3 to 6 until end of file
8. Return hash

Parameters

`f:` file handle of file to hash

`fsize:` file size of f

(type=integer)

`start:` file location to start hashing

(type=integer)

`blocksize:` block size for hash generation

(type=integer)

`algorithm:` algorithm to generate hash. Allowable options are {md5|sha1|sha244|sha256|sha384|sha512}

(type=strong)

Return Value

generated hash in string

backward_file_hash(*f*, *end*, *blocksize*, *algorithm*)

Forward file hashing. It takes file and generate repeated hashes to the end of the file in blocks.

Algorithm:

1. block <- file from position N - blocksize to N
2. hash <- generate hash from block
3. N = N - blocksize
4. block <- file from position N - blocksize to N
5. block = block + hash
6. hash <- generate hash from block
7. Repeat steps 3 to 6 until start of file
8. Return hash

Parameters

`f:` file handle of file to hash

`end:` file location to start hashing

(type=integer)

`blocksize:` block size for hash generation

(type=integer)

`algorithm:` algorithm to generate hash. Allowable options are {md5|sha1|sha244|sha256|sha384|sha512}

(type=strong)

Return Value

generated hash in string

cfh(*filename*, *blocksize*=`1024`, *startpoints*=`10`, *algorithm*=`'md5'`)

Circular file hasher. This function uses repeated forward and backward hash functions to generate hash for the entire file. Hence, a byte in the file will be hashed multiple times. The length of hash generated will be dependent on the block size and number of start points.

Algorithm:

1. forward hash <- hash(start) + [hash(start + 1) for start = 0 to startpoints]
2. backward hash <- hash(start) + [hash(start - 1) for start = startpoints to 0]
3. Return forward hash + backward hash

Parameters

`filename:` name of file to hash

(type=string)

`blocksize:` block size for hash generation. Default = 1024

(type=integer)

`startpoints:` defines the number of start point on the file for hash generation. Default = 10.

(type=integer)

`algorithm:` algorithm to generate hash. Allowable options are {md5|sha1|sha244|sha256|sha384|sha512}

(type=strong)

Return Value

generated hash in string

14.2 Variables

Name	Description
__package__	**Value:** `'copads'`

15 Module copads.hypothesis

Statistical Hypothesis Testing Routines.

Each routine will Returns a 5-element list

ritical, right result

where

- left result = True (statistic in lower critical region) or False (statistic not in lower critical region)
- left critical = lower critical value generated from 1 - confidence
- statistic = calculated statistic value
- right critical = upper critical value generated from confidence
- right result = True (statistic in upper critical region) or False (statistic not in upper critical region)

References

- Test 1-100: Gopal K. Kanji. 2006. 100 Statistical Tests, 3rd edition. Sage Publications.

Date created: 1st September 2008

15.1 Functions

test(*statistic*, *distribution*, *confidence*)

Generates the critical value from distribution and confidence value using the distribution's inverseCDF method and performs 1-tailed and 2-tailed test by comparing the calculated statistic with the critical value.

Returns a 5-element list [left result, left critical, statistic, right critical, right result] where

- left result = True (statistic in lower critical region) or False (statistic not in lower critical region)
- left critical = lower critical value generated from 1 - confidence
- statistic = calculated statistic value
- right critical = upper critical value generated from confidence
- right result = True (statistic in upper critical region) or False (statistic not in upper critical region)

Therefore, null hypothesis is accepted if left result and right result are both False in a 2-tailed test.

Parameters

`statistic:` calculated statistic (float)

`distribution:` distribution to calculate critical value

(type=instance of a statistics distribution)

`confidence:` confidence level of a one-tail test (usually 0.95 or 0.99), use 0.975 or 0.995 for 2-tail test

(type=float of less than 1.0)

Z1Mean1Variance(*smean*, *pmean*, *pvar*, *ssize*, *confidence*)

Test 1: Z-test for a population mean (variance known)

To investigate the significance of the difference between an assumed population mean and sample mean when the population variance is known.

Limitations

1. Requires population variance (use Test 7 if population variance unknown)

Parameters

`smean:`	sample mean
`pmean:`	population mean
`pvar:`	population variance
`ssize:`	sample size
`confidence:`	confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Z2Mean1Variance(*smean1*, *smean2*, *pvar*, *ssize1*, *ssize2*, *confidence*, *pmean1*=`0.0`, *pmean2*=`0.0`)

Test 2: Z-test for two population means (variances known and equal)

To investigate the significance of the difference between the means of two samples when the variances are known and equal.

Limitations

1. Population variances must be known and equal (use Test 8 if population variances unknown

Parameters

`smean1:`	sample mean of sample #1
`smean2:`	sample mean of sample #2
`pvar:`	variances of both populations (variances are equal)
`ssize1:`	sample size of sample #1
`ssize2:`	sample size of sample #2
`confidence:`	confidence level
`pmean1:`	population mean of population #1 (optional)
`pmean2:`	population mean of population #2 (optional)

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Z2Mean2Variance(*smean1*, *smean2*, *pvar1*, *pvar2*, *ssize1*, *ssize2*, *confidence*, *pmean1*=`0.0`, *pmean2*=`0.0`)

Test 3: Z-test for two population means (variances known and unequal)

To investigate the significance of the difference between the means of two samples when the variances are known and unequal.

Limitations

1. Population variances must be known(use Test 9 if population variances unknown

Parameters

`smean1:`	sample mean of sample #1
`smean2:`	sample mean of sample #2
`pvar1:`	variance of population #1
`pvar2:`	variance of population #2
`ssize1:`	sample size of sample #1
`ssize2:`	sample size of sample #2
`confidence:`	confidence level
`pmean1:`	population mean of population #1 (optional)
`pmean2:`	population mean of population #2 (optional)

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.4

Z1Proportion(*spro*, *ppro*, *ssize*, *confidence*)

Test 4: Z-test for a proportion (binomial distribution)

To investigate the significance of the difference between an assumed proportion and an observed proportion.

Limitations

1. Requires sufficiently large sample size to use Normal approximation to binomial

Parameters

`spro:` sample proportion

`ppro:` population proportion

`ssize:` sample size

`confidence:` confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Z2Proportion(*spro1*, *spro2*, *ssize1*, *ssize2*, *confidence*)

Test 5: Z-test for the equality of two proportions (binomial distribution) To investigate the assumption that the proportions of elements from two populations are equal, based on two samples, one from each population.

Limitations

1. Requires sufficiently large sample size to use Normal approximation to binomial

Parameters

`spro1:` sample proportion #1

`spro2:` sample proportion #2

`ssize1:` sample size #1

`ssize2:` sample size #2

`confidence:` confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Z2Count(*time1*, *time2*, *count1*, *count2*, *confidence*)

Test 6: Z-test for comparing two counts (Poisson distribution)

To investigate the significance of the differences between two counts.

Limitations

1. Requires sufficiently large sample size to use Normal approximation to binomial

Parameters

`time1:` first measurement time

`time2:` second measurement time

`count1:` counts at first measurement time

`count2:` counts at second measurement time

`confidence:` confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

t1Mean(*smean*, *pmean*, *svar*, *ssize*, *confidence*)

Test 7: t-test for a population mean (population variance unknown)

To investigate the significance of the difference between an assumed population mean and a sample mean when the population variance is unknown and cannot be assumed equal or not equal.

Limitations

1. Weaker form of Test 1

Parameters

`smean:`	sample mean
`pmean:`	population mean
`svar:`	sample variance
`ssize:`	sample size
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

t2Mean2EqualVariance(*smean1*, *smean2*, *svar1*, *svar2*, *ssize1*, *ssize2*, *confidence*, *pmean1*=`0.0`, *pmean2*=`0.0`)

Test 8: t-test for two population means (population variance unknown but equal)

To investigate the significance of the difference between the means of two populations when the population variances are unknown but assumed equal.

Limitations

1. Weaker form of Test 2

Parameters

`smean1:`	sample mean of sample #1
`smean2:`	sample mean of sample #2
`svar1:`	variances of sample #1
`svar2:`	variances of sample #2
`ssize1:`	sample size of sample #1
`ssize2:`	sample size of sample #2
`confidence:`	confidence level
`pmean1:`	population mean of population #1 (optional)
`pmean2:`	population mean of population #2 (optional)

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

t2Mean2UnequalVariance(*smean1*, *smean2*, *svar1*, *svar2*, *ssize1*, *ssize2*, *confidence*, *pmean1*=`0.0`, *pmean2*=`0.0`)

Test 9: t-test for two population means (population variance unknown and unequal)

To investigate the significance of the difference between the means of two populations when the population variances are unknown and unequal.

Limitations

1. Weaker form of Test 3

Parameters

`smean1:`	sample mean of sample #1
`smean2:`	sample mean of sample #2
`svar1:`	variances of sample #1
`svar2:`	variances of sample #2
`ssize1:`	sample size of sample #1
`ssize2:`	sample size of sample #2
`confidence:`	confidence level
`pmean1:`	population mean of population #1 (optional)
`pmean2:`	population mean of population #2 (optional)

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

tPaired(*smean1*, *smean2*, *svar*, *ssize*, *confidence*)

Test 10: t-test for two population means (method of paired comparisons)

To investigate the significance of the difference between two population means when no assumption is made about the population variances.

Parameters

`smean1:` sample mean of sample #1

`smean2:` sample mean of sample #2

`svar:` variance of differences between pairs

`ssize:` sample size

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

tRegressionCoefficient(*variancex*, *varianceyx*, *b*, *ssize*, *confidence*)

Test 11: t-test of a regression coefficient

To investigate the significance of the regression coefficient.

Limitations

1. Homoedasticity of values

Parameters

`variancex:` variance of x

`varianceyx:` variance of yx

`b:` calculated Regression Coefficient

`ssize:` sample size

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

tPearsonCorrelation(*r*, *ssize*, *confidence*)

Test 12: t-test of a correlation coefficient

To investigate whether the difference between the sample correlation coefficient and zero is statistically significant.

Limitations

1. Assumes population correlation coefficient to be zero (use Test 13 for testing other population correlation coefficient
2. Assumes a linear relationship (regression line as Y = MX + C)
3. Independence of x-values and y-values

Use Test 59 when these conditions cannot be met

Parameters

`r:`	calculated Pearson's product-moment correlation coefficient
`ssize:`	sample size
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

ZPearsonCorrelation(*sr*, *pr*, *ssize*, *confidence*)

Test 13: Z-test of a correlation coefficient

To investigate the significance of the difference between a correlation coefficient and a specified value.

Limitations

1. Assumes a linear relationship (regression line as Y = MX + C)
2. Independence of x-values and y-values

Use Test 59 when these conditions cannot be met

Parameters

`sr:`	calculated sample Pearson's product-moment correlation coefficient
`pr:`	specified Pearson's product-moment correlation coefficient to test
`ssize:`	sample size
`confidence:`	confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Z2PearsonCorrelation(*r1*, *r2*, *ssize1*, *ssize2*, *confidence*)

Test 14: Z-test for two correlation coefficients

To investigate the significance of the difference between the correlation coefficients for a pair variables occurring from two difference populations.

Parameters

`r1:`	Pearson correlation coefficient of sample #1
`r2:`	Pearson correlation coefficient of sample #2
`ssize1:`	Sample size #1
`ssize2:`	Sample size #2
`confidence:`	confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

ChiSquarePopVar(*values*, *ssize*, *pv*, *confidence*=`0.95`)

Test 15: Chi-square test for a population variance

To investigate the difference between a sample variance and an assumed population variance.

Parameters

`values:`	sample values
`ssize:`	sample size
`pv:`	population variance
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

FVarianceRatio(*var1*, *var2*, *ssize1*, *ssize2*, *confidence*)

Test 16: F-test for two population variances (variance ratio test)

To investigate the significance of the difference between two population variances.

Parameters

`var1:`	variance #1
`var2:`	variance #2
`ssize1:`	sample size #1
`ssize2:`	sample size #2
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

F2CorrelatedObs(*r*, *var1*, *var2*, *ssize1*, *ssize2*, *confidence*)

Test 17: F-test for two population variances (with correlated observations)

To investigate the difference between two population variances when there is correlation between the pairs of observations.

Parameters

`r:`	Sample correlation value
`var1:`	variance #1
`var2:`	variance #2
`ssize1:`	sample size #1
`ssize2:`	sample size #2
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

ZCorrProportion(*ssize*, *ny*, *yn*, *confidence*)

Test 23: Z-test for correlated proportions

To investigate the significance of the difference between two correlated proportions in opinion surveys. It can also be used for more general applications.

Limitations

1. The same people are questioned both times (correlated property).
2. Sample size must be quite large.

Parameters

`ssize:`	sample size
`ny:`	number answered 'no' in first poll and 'yes' in second poll
`yn:`	number answered 'yes' in first poll and 'no' in second poll
`confidence:`	confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

Chisq2Variance(*ssize*, *svar*, *pvar*, *confidence*)

Test 24: Chi-square test for an assumed population variance

To investigate the significance of the difference between a population variance and an assumed variance value.

Limitations

1. Sample from normal distribution

Parameters

`ssize:` sample size

`svar:` sample variance

`pvar:` population variance (assumed)

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

F2Count(*count1*, *count2*, *confidence*, *time1*=0, *time2*=0, *repeat*=False)

Test 25: F-test for two counts (Poisson distribution)

To investigate the significance of the difference between two counted results (based on a Poisson distribution).

Limitations

1. Counts must satisfy a Poisson distribution
2. Samples obtained under same conditions.

Parameters

`count1:`	count of first sample
`count2:`	count of second sample
`repeat:`	flag for repeated sampling (default = False)
`time1:`	time at which first sample is taken (only needed if repeat = True)
`time2:`	time at which second sample is taken (only needed if repeat = True)
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

ChisqFit(*observed*, *expected*, *confidence*)

Test 37: Chi-square test for goodness of fit

To investigate the significance of the differences between observed data arranged in K classes, and the theoretical expected frequencies in the K classes.

Limitations

1. Observed and theoretical distributions should have same number of elements
2. Same class division for both distributions
3. Expected frequency of each class should be at least 5

Parameters

`observed:` list of observed frequencies (index matched with expected)

`expected:` list of expected frequencies (index matched with observed)

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

tx2testofKcounts(*T*, *V*, *confidence*)

Test 38: The x2-test for compatibility of K counts

To investigate the significance of the differences between K counts.

Limitations:

1. The counts must be obtained under comparable conditions

Parameters

`T:` list of time under K counts

`V:` list of values of K counts

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

Chisq2x2(*s1*, *s2*, *ssize*, *confidence*)

Test 40: Chi-square test for consistency in 2x2 table

To investigate the significance of the differences between observed frequencies for two dichotomous distributions.

Limitations

1. Total sample size (sample 1 + sample 2) must be more than 20
2. Each cell frequency more than 3

Parameters

`s1:` 2-element list or tuple of frequencies for sample #1

`s2:` 2-element list or tuple of frequencies for sample #2

`ssize:` sample size

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

ChisquareKx2table(*c1*, *c2*, *k*, *confidence*)

Test 41: The x2-test for consistency in a K x 2 table

To investigate the significance of the differences between K observed frequency distributions with a dichotomous classification.

Limitations:

1. K sample sizes must be large enough.
2. It is usually assumed to be satisfied if the cell frequencies are equal to 5

Parameters

`c1:`	class #1 values of sample k
`c2:`	class #2 values of sample k
`k:`	number of samples
`confidence:`	confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

Chisquare2xKtable(*s1*, *s2*, *k*, *confidence*)

Test 43: The x2-test for consistency in a 2 x K table

To investigate the significance of the differences between two distributions based on two samples spread over K classes

Limitations:

1. The two samples are sufficiently large
2. The K classes when put together form a complete series

Parameters

`s1:` sample #1 of class k

`s2:` sample #2 of class k

`k:` number of classes

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

MedianTestfor2Pop(*s1*=`(9, 6)`, *s2*=`(6, 9)`, *confidence*=`0.95`)

Test 50: The median test of two populations

To test if two random samples could have come from two populations with same frequency distribution

Limitations:

1. The two samples are assumed to be reasonably large

Parameters

`s1:` 2-element list or tuple of frequencies for sample #1

`s2:` 2-element list or tuple of frequencies for sample #2

`confidence:` confidence level

See Also: Chay, ZE, Ling, MHT. 2010. COPADS, II: Chi-Square test, F-Test and t-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 2:3

Status: Tested function

Since: version 0.4

SpearmanCorrelation(*ssize*, *confidence*, *R*=`None`, *series1*=`[]`, *series2*=`[]`)

Test 58: Spearman rank correlation test (paired observations) To investigate the significance of the correlation between two series of observations obtained in pairs.

Limitations

1. Assumes the two population distributions to be continuous
2. Sample size must be more than 10

Parameters

`R:`	sum of squared ranks differences
`ssize:`	sample size
`series1:`	ranks of series #1 (not used if R is given)
`series2:`	ranks of series #2 (not used if R is given)
`confidence:`	confidence level

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested function

Since: version 0.1

ZtestLogOddsRatio(*group1*, *group2*, *confidence*)

Test 84: Z-test for comparing sequential contingencies acoss two groups using the 'log odds ratio'

To test the significance of the difference in sequential connections across groups

Limitations:

1. This test is applicable when a logit transformation can be used and 2 X 2 contingency tables are available

Parameters

`group1:`	group #1 values (row = x, y) & (column a, b) Provide values in xa, xb, ya, yb format
`group2:`	group #2 values (row = x, y) & (column a, b) Provide values in xa, xb, ya, yb format
`confidence:`	confidence level

15.2 Variables

Name	Description
BACKHOUSE	**Value:** `1.45607494858`
CFRACT	**Value:** `1.0306408341`
CGOLD	**Value:** `0.38196601125`
FRODA	**Value:** `6.58088599102`
GAMMA	**Value:** `0.577215664902`
GOLD	**Value:** `1.61803398875`
GOLOMB	**Value:** `0.624329988544`
INV2PI	**Value:** `0.159154943092`
INVLN10	**Value:** `0.434294481903`
INVLN2	**Value:** `1.44269504089`
INVPI180	**Value:** `57.2957795131`
INVSQRT2PI	**Value:** `0.398942280401`
LEHMER	**Value:** `0.592632718292`
LENGYEL	**Value:** `1.09868580553`
LN10	**Value:** `2.30258509299`
LN2	**Value:** `0.69314718056`
LNPI	**Value:** `1.14472988585`
PARIS	**Value:** `1.09864196439`
PI	**Value:** `3.14159265359`
PI180	**Value:** `0.0174532925199`
PI2	**Value:** `6.28318530718`
PIDIV2	**Value:** `1.57079632679`
PORTER	**Value:** `1.46707807943`
RABBIT	**Value:** `0.709803442861`
SQRT2	**Value:** `1.41421356237`
SQRT2PI	**Value:** `2.50662827463`
ZETA9	**Value:** `1.00200839283`
__package__	**Value:** `'copads'`

16 Module copads.lc_bf

Loose Circular Brainfuck (LCBF) Interpreter Date created: 15th August 2012 Licence: Python Software Foundation License version 2

LCBF uses all 8 operations in standard Brainfuck with the following differences:

1. The tape or array is circular (a ring list) instead of linear. When the pointer is at the "end" of the tape, an increment (">") will move the tape to the start. Similarly, when the pointer is decremented at the "beginning" of the tape, the pointer goes to the end.
2. Operations after a start loop operator ("[") will only be executed provided the loop(s) are properly closed. If the loops are open, the program will terminate.
3. However, it is possible to have an end loop operator ("]") without a preceding start loop operator ("["). In this case, the end loop operator ("]") will be ignored and execution continues.
4. Unclosed or unopened loops may result in non-deterministic behaviour.
5. All inputs are pre-defined at the start of the program.

See Also: http://esolangs.org/wiki/Loose_Circular_Brainfuck_(LCBF)

16.1 Functions

increment(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Increase value of cell by 1. Equivalent to "+" in Brainfuck.

decrement(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Decrease value of cell by 1. Equivalent to "-" in Brainfuck.

forward(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Move forward by one cell on tape. Equivalent to ">" in Brainfuck.

backward(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Move backward by one cell on tape. Equivalent to "<" in Brainfuck.

call_out(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Output current tape cell value and append to the end of the output list. Equivalent to "." in Brainfuck.

accept_predefined(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Writes the first value of the input list into the current cell and remove the value from the input list. If input list is empty, "0" will be written

cbf_start_loop(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Start loop. Operations after a start loop operator ("[") will only be executed provided the loop(s) are properly closed. If the loops are open, the program will terminate. Note that unclosed or unopened loops may result in non-deterministic behaviour.

cbf_end_loop(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

End loop. However, it is possible to have an end loop operator ("]") without a preceding start loop operator ("["). In this case, the end loop operator ("]") will be ignored and execution continues. Note that unclosed or unopened loops may result in non-deterministic behaviour.

16.2 Variables

Name	Description
LCBF	**Value:** {'+': increment, '-': decrement, '>': forward, '<': backw...
__package__	**Value:** 'copads'

17 Module copads.matrix

Matrix Data Structures and Algorithms.

Date created: 19th March 2008

17.1 Functions

isVector(x)

Determines if the argument is a vector class object.

vZeros(n)

Returns a vector of length n with all ones.

vOnes(n)

Returns a vector of length n with all ones.

vRandom(n, $lmin$=`0.0`, $lmax$=`1.0`)

Returns a random vector of length n.

vDot(a, b)

dot product of two vectors.

vNorm(a)

Computes the norm of vector a.

vSum(a)

Returns the sum of the elements of a.

vLog10(a)

log10 of each element of a.

vLog(a)

log of each element of a.

vExp(a)

Elementwise exponential.

vSin(a)

Elementwise sine.

vTan(a)

Elementwise tangent.

vCos(a)

Elementwise cosine.

vAsin(a)

Elementwise inverse sine.

vAtan(a)

Elementwise inverse tangent.

vAcos(a)

Elementwise inverse cosine.

vSqrt(a)

Elementwise sqrt.

vSinh(a)

Elementwise hyperbolic sine.

vTanh(a)

Elementwise hyperbolic tangent.

vCosh(a)

Elementwise hyperbolic cosine.

vPow(a, b)

Takes the elements of a and raises them to the b-th power

vAtan2(*a*, *b*)

Arc tangent

isSparse(*x*)

smTranspose(*a*)

transpose

smDotDot(*y*, *a*, *x*)

double dot product y^+ *A*x

smDot(*a*, *b*)

vector-matrix, matrix-vector or matrix-matrix product

smDiag(*b*)

smIdentity(*n*)

17.2 Variables

Name	Description
__package__	**Value:** `'copads'`

17.3 Class Vector

object
  list
    copads.matrix.Vector

A list based vector class that supports elementwise mathematical operations

Adapted from http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52272 Original author: A. Pletzer

17.3.1 Methods

__getslice__(*self*, *i*, *j*)

x[i:j]

Use of negative indices is not supported.

Overrides: list.__getslice__ extit(inherited documentation)

__add__(*self*, *other*)

Addition of current matrix to "other" matrix.

Overrides: list.__add__

Status: Tested method

Since: version 0.1

__neg__(*self*)

Status: Tested method

Since: version 0.1

__sub__(*self*, *other*)

Status: Tested method

Since: version 0.1

__mul__(*self*, *other*)

Element by element multiplication

Overrides: list.__mul__

Status: Tested method

Since: version 0.1

__rmul__(*self*, *other*)

n*x

Overrides: list.__rmul__ extit(inherited documentation)

__div__(*self*, *other*)

Element by element division.

Status: Tested method

Since: version 0.1

__rdiv__(*self*, *other*)

The same as __div__

size(*self*)

conjugate(*self*)

ReIm(*self*)

Return the real and imaginary parts

AbsArg(*self*)

Return modulus and phase parts

out(*self*)

Prints out the vector.

Inherited from list

__contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getattribute__(), __getitem__(), __gt__(), __iadd__(), __imul__(), __init__(), __iter__(), __le__(), __len__(), __lt__(), __ne__(), __new__(), __repr__(), __reversed__(), __setitem__(), __setslice__(), __sizeof__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from object

__delattr__(), __format__(), __reduce__(), __reduce_ex__(), __setattr__(), __str__(), __subclasshook__()

17.3.2 Properties

Name	Description
Inherited from object	
__class__	

17.3.3 Class Variables

Name	Description
Inherited from list	
__hash__	

17.4 Class Matrix

A linear algebra matrix

This class defines a generic matrix and the basic matrix operations from linear algebra. An instance of this class is a single matrix with particular values.

Arithmetic operations, trace, determinant, and minors are defined. This is a lightweight alternative to a numerical Python package for people who need to do basic linear algebra.

Vectors are implemented as 1xN and Nx1 matricies. There is no separate vector class. This implementation enforces the distinction between row and column vectors.

Indexing is zero-based, i.e. the upper left-hand corner of a matrix is element (0,0), not element (1,1).

Matricies are stored as a list of lists, where the top level lists are the rows and the sub-lists are the columns. Because of the way Python handles list references, you have be careful when copying matrix objects. If you have a matrix a, assign b=a, and then change values in b, you will change values in a as well. Matrix copying should be done with copy.deepcopy.

This implementation has no memory-saving optimization for sparse matricies. A derived class may implement a more sophisticated storage method by overriding the __getitem__ and __setitem__ functions.

Determinants are taken by expanding by minors on the top row. The private functions supplied for expansion by minors are more generic than what is needed by this implementation. They may be used by a derived class that wishes to do more efficient expansion of sparse matricies.

By default, Matrix elements are members of the complex field, but if you want to perform linear algebra on something other than numbers you may redefine Matrix.null_element, Matrix.identity_element, and Matrix.inverse_element and override the is_scalar_element function.

References George Arfken, "Mathematical Methods for Physicists", 3rd ed. San Diego. Academic Press Inc. (1985)

Adapted from: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/189971 Original author: Bill McNeill <billmcn@speakeasy.net>

Maintainer: Maurice H.T. Ling <mauriceling@acm.org> Copyright (c) 2005 Maurice H.T.

Ling <mauriceling@acm.org> Date: 1st May 2005

17.4.1 Methods

__init__(*self*, **args*)

Matrix constructor

A matrix can be created in three ways.

1. A single integer argument is supplied. The constructor creates a null square matrix of that size. For example
   Matrix(2)
   creates the following matrix
   0 0 0 0
2. Two integer arguments are supplied. The constructor creates a null matrix of size first argument x second argument. For example
   Matrix(2, 3)
   creates the following matrix
   0 0 0 0 0 0
3. A list of lists is supplied. It represents a set of initial matrix values. Each element is a row and each sub-list is a column. For example
   Matrix([[1,2,3], [4,5,6], [7,8,9]])
   creates the following matrix
   1 2 3 4 5 6 7 8 9

createNullMatrix(*self*, *row*, *col*)

Create a matrix using the null value

This is a private function called by __init__.

__str__(*self*)

__cmp__(*self*, *other*)

__getitem__(*self*, *(row, col)*)

The value at (row, col)

For example, to get the value of element 1,3 say

m[(1,3)]

__setitem__(*self*, *(row, col)*, *value*)

Sets the value at (row, col)

For example, to set the value of element 1,3 to 5 say

m[(1,3)] = 5

rows(*self*)

The number of rows in the matrix

cols(*self*)

The number of columns in the matrix

row(*self*, *i*)

The ith row of the matrix

col(*self*, *j*)

The jth row of the matrix

__add__(*self*, *other*)

Add matrix self + other

__neg__(*self*)

Negate the current matrix

__sub__(*self*, *other*)

Subtract matrix self - other

__mul__(*self*, *other*)

Multiply matrix self*other

other can be another matrix or a scalar.

__rmul__(*self*, *other*)

Multiply other*self

This is only called if other.__add__ is not defined, so assume that other is a scalar.

scalarMultiply(*self*, *scalar*)

Multiply the matrix by a scalar value.

This is a private function called by __mul__ and __rmul__.

matrixMultiply(*self*, *other*)

Multiply the matrix by another matrix.

This is a private function called by __mul__.

isRowVector(*self*)

Is the matrix a row vector?

isColumnVector(*self*)

Is the matrix a column vector?

isSquare(*self*)

Is the matrix square?

transpose(*self*)

The transpose of the matrix

trace(*self*)

The trace of the matrix

determinant(*self*)

The determinant of the matrix

expandByMinorsOnRow(*self*, *row*)

Calculates the determinant by expansion of minors

This function returns the determinant of the matrix by doing an expansion of minors on the specified row.

expandByMinorsOnColumn(*self*, *col*)

Calculates the determinant by expansion of minors

This function returns the determinant of the matrix by doing an expansion of minors on the specified column.

minor(*self*, *i*, *j*)

A minor of the matrix

This function returns the minor given by striking out row i and column j of the matrix.

minorMatrix(*self*)

Calculates minor matrix

adjoint(*self*)

Calculates adjoint matrix

inverse(*self*)

Calculates inverse matrix

vectorInnerProduct(*self*, *a*, *b*)

Takes the inner product of vectors a and b

a and b are lists. This is a private function called by matrix_multiply.

isScalarElement(*self*, *x*)

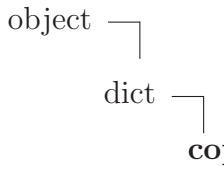
Is x a scalar

By default a scalar is an element in the complex number field. A class that wants to perform linear algebra on things other than numbers may override this function.

17.4.2 Class Variables

Name	Description
null_element	**Value:** `0`
identity_element	**Value:** `1`
inverse_element	**Value:** `-1`

17.5 Class SparseMatrix



copads.matrix.SparseMatrix

A sparse matrix class based on a dictionary, supporting matrix (dot) product and a conjugate gradient solver.

In this version, the sparse class inherits from the dictionary; this requires Python 2.2 or later.

Adapted from http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52275 Original author: Alexander Pletzer

Dictionary storage format { (i,j): value, ... } where (i,j) are the matrix indices

17.5.1 Methods

size(*self*)

returns # of rows and columns

__**add**__(*self*, *other*)

__**neg**__(*self*)

__**sub**__(*self*, *other*)

__**mul**__(*self*, *other*)

element by element multiplication: other can be scalar or sparse

__**rmul**__(*self*, *other*)

__**div**__(*self*, *other*)

element by element division self/other: other is scalar

__**rdiv**__(*self*, *other*)

element by element division other/self: other is scalar

abs(*self*)

out(*self*)

plot(*self*, *width_in*=400, *height_in*=400)

CGsolve(*self*, *x0*, *b*, *tol*=1e-10, *nmax*=1000, *verbose*=1)

Solve self*x = b and return x using the conjugate gradient method

biCGsolve(*self*, *x0*, *b*, *tol*=1e-10, *nmax*=1000)

Solve self*x = b and return x using the bi-conjugate gradient method

save(*self*, *filename*, *OneBased*=0)

Save matrix in file <filaname> using format OneBased, nrow, ncol, nnonzeros [ii, jj, data]

Inherited from dict

__cmp__(), __contains__(), __delitem__(), __eq__(), __ge__(), __getattribute__(), __getitem__(), __gt__(), __init__(), __iter__(), __le__(), __len__(), __lt__(), __ne__(), __new__(), __repr__(), __setitem__(), __sizeof__(), clear(), copy(), fromkeys(), get(), has_key(), items(), iteritems(), iterkeys(), itervalues(), keys(), pop(), popitem(), setdefault(), update(), values(), viewitems(), viewkeys(), viewvalues()

Inherited from object

__delattr__(), __format__(), __reduce__(), __reduce_ex__(), __setattr__(), __str__(), __subclasshook__()

17.5.2 Properties

Name	Description
Inherited from object	
__class__	

17.5.3 Class Variables

Name	Description
Inherited from dict	
__hash__	

18 Module copads.n_bf

NucleotideBF (nBF) Interpreter Date created: 15th August 2012 Licence: Python Software Foundation License version 2

NucleotideBF is a derivative of Brainfuck based on IUPAC nucleotide code. It uses only 5 of the 8 operations in Brainfuck, and there is no loop operations.

The commands for nBF can be divided into 2 classes - deterministic operations and random operations. The deterministic operations are A (increment, equivalent to '+'), T (decrement, equivalent to '-'), C (backward, equivalent to '<'), G (forward, equivalent to '>') and '.' (call out).

Based on the same interpreter as Loose Circular Brainfuck (LCBF), the tape or array is circular (a ring list) instead of linear. When the pointer is at the "end" of the tape, an increment ("A") will move the tape to the start. Similarly, when the pointer is decremented at the "beginning" of the tape, the pointer goes to the end.

See Also: http://esolangs.org/wiki/NucleotideBF_(nBF)

18.1 Functions

random_op(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Random operations / commands to simulate ambiguous DNA nucleotides. Allowable ambiguous nucleotides are: R: Random between A or G Y: Random between C or T S: Random between G or C W: Random between A or T K: Random between G or T M: Random between A or C B: Random between C or G or T D: Random between A or G or T H: Random between A or C or T V: Random between A or C or G N: Random between A or T or C or G

18.2 Variables

Name	Description
nBF	**Value:** {'A': increment, 'T': decrement, 'G': forward, 'C': backw...
__package__	**Value:** 'copads'

19 Module copads.neural

Framework for Neural Network Applications Date created: 10th May 2013 License: Python Software Foundation License version 2

19.1 Functions

tf_linear(x)

Linear transfer function. Equation: y = x

itf_linear(y)

Linear transfer function - inverse. Equation: x = y

dtf_linear(x)

Linear transfer function - derivative. Equation: df = 1.0

19.2 Variables

Name	Description
__package__	**Value:** `'copads'`

19.3 Class Neuron

Class for a Neuron.

1. Each neuron is identified by a unique name, which will be used for mapping between different neurons.
2. A dictionary, named "cellbody", is provided to contain any other information needed. For example, it can be used to contain a timer for time-delayed neuron activation or it can be used for neuronal memory.

19.3.1 Methods

__init__(*self*, *name*=`None`, *transfer_function*=<`function tf_linear at 0x1051c1a28`>, *itransfer_function*=<`function itf_linear at 0x1051c1b18`>, *dtransfer_function*=<`function dtf_linear at 0x1051c1b90`>)

Constructor method for Neuron class.

Parameters

`name:` unique name of the neuron. Default = a string of 60 to 75 numbers.

(type=string)

`transfer_function:` a function to convert the consolidated weighted input for the neuron and generate an output signal. Default = tf_linear; that is, output signal = consolidated weighted input.

(type=function)

`itransfer_function:` inverse of transfer_function, used by some learning algorithms, Default = itf_linear

(type=function)

`dtransfer_function:` differential of transfer_function, used by some learning algorithms, Default = dtf_linear

(type=function)

generate_name(*self*)

Generate a new random name (a string of 60 to 75 numbers) for the current neuron.

Return Value

new name of neuron.

set_transfer_functions(*self*, *transfer_function*, *itransfer_function*=`None`, *dtransfer_function*=`None`)

Set transfer function, its inverse and differential functions.

Parameters

`transfer_function:` a function to convert the consolidated weighted input for the neuron and generate an output signal.

(type=function)

`itransfer_function:` inverse of transfer_function, used by some learning algorithms, Default = None

(type=function)

`dtransfer_function:` differential of transfer_function, used by some learning algorithms, Default = None

(type=function)

execute(*self*, *synapses*, *activations*)

Execute/activate the neuron into action.

Parameters

`synapses:` dictionary of reverse synaptic connections and its synaptic weights in the format of {name of signal receiving neuron>: {<name of signal receiving neuron>: <synaptic weight>}}

(type=dictionary)

`activations:` a dictionary of current activations of the entire network where keys are the names of neurons and values are the activation state or current output signal.

(type=dictionary)

Return Value

activations dictionary with the updated output signal or activation state from the current neuron.

19.3.2 Class Variables

Name	Description
name	**Value:** `None`
weights	**Value:** {}
cellbody	**Value:** {}

continued on next page

Name	Description
transfer_function	**Value:** `None`
itransfer_function	**Value:** `None`
dtransfer_function	**Value:** `None`

19.4 Class Brain

Class for the neural network.

1. A brain consists of a set of neurons (uniquely named or uniquely randomly named) held in "neuron_pool" dictionary where key is the name and value is the neuron object.
2. The corresponding names of neurons are entered into "activations" dictionary as keys. This dictionary holds the current activation states or output signals (as values) of all neurons.
3. A sequence of neuronal activity is given as activation_sequence, which is a list of list. This suggests that neurons listed in activation_sequence[0] will be activated before those neurons listed in activation_sequence[1], and neuron in activation_sequence[0][0] will be activated before neuron in activation_sequence[0][1]. Inherited class can over-ride Brain.run() method to cater for parallel activations of all neurons in activation_sequence[0] or even parallel activation of all neurons regardless of sequence.
4. In sequential activation (that is activation_sequence[0] before activation_sequence[1]), neurons in activation_sequence[0] will be the input neurons to traditional neural networks.
5. The connections and synaptic weights are registered in "synapses" dictionary as a reverse connection - {<name of signal receiving neuron>: <dictionary of signal receiving neurons and its weights>} where <dictionary of signal receiving neurons and its weights> is {<name of signal receiving neuron>: <synaptic weight>}. Hence, synpases dictionary is a nested dictionary of {name of signal receiving neuron>: {<name of signal receiving neuron>: <synaptic weight>}}.
6. A dictionary, called "brainmatter", is provided to contain any other information needed.
7. The brain will also a learning algorithm to train / learn by itself.

19.4.1 Methods

__init__(*self*, *number_of_neurons*=0, *original_neuron*=None, *list_of_neuron_names*=[], *learning_algorithm*=None)

Constructor method for Brain class.

A brain or neural network can be constructed by a predefined list of unique names for neurons or by stating the number of neurons required. In both cases, a pre-created neuron may be used but is not mandatory. However, in event where both the list of unique names for neurons and the number of neurons required are given, the list of unique names for neurons takes precedence - for example, if 5 neuron names but 10 neurons are required, then only 5 neurons (which are named) will be created.

Parameters

`number_of_neurons:` number of neurons requested (takes a lower precedence than the names of neurons), default = 0.

(type=integer)

`original_neuron:` a pre-created neuron to duplicate, default = None

(type=Neuron object)

`list_of_neuron_names:` names for neurons to be created (takes a higher precedence than the number of neurons requested), default = [] (empty list).

(type=list)

`learning_algorithm:` learning mechanism for brain

(type=function)

connect_neurons(*self*, *originating_neuron*, *destination_neuron*)

Establish synaptic connection between 2 existing neurons.

Parameters

`originating_neuron:` name of neuron to connect from

(type=string)

`destination_neuron:` name of neuron to connect to

(type=string)

disconnect_neurons(*self*, *originating_neuron*, *destination_neuron*)

Disconnect between 2 connected neurons.

Parameters

`originating_neuron:` name of neuron to disconnect from
(type=string)

`destination_neuron:` name of neuron to disconnect to
(type=string)

set_activation_sequence(*self*, *activation_sequence*)

Set neuron activation sequence or replace current neuron activation sequence.

Parameters

`activation_sequence:` list of list of activation sequence

add_neuron_to_activation_sequence(*self*, *neuron_name*, *position*)

Add neuron to activation sequence.

Parameters

`neuron_name:` name of neuron to add
(type=string)

`position:` position to add neuron, where position > 0
(type=integer)

remove_neuron_from_activation_sequence(*self*, *neuron_name*, *position*=`'all'`)

Remove neuron from activation sequence.

Parameters

`neuron_name:` name of neuron to remove
(type=string)

`position:` position to add neuron, where position > 0 (integer) or 'all' (string). If position = 'all', this method will remove neuron from the entire activation sequence

add_neuron(*self*, *neuron*=`None`)

Add a neuron into the brain.

Parameters

`neuron:` the neuron to be added. If None, then a generic neuron will be created and added. Default = None.

(type=Neuron object)

remove_neuron(*self*, *neuron_name*)

Removing a neuron from the brain.

Parameters

`neuron_name:` name of neuron to remove

(type=string)

empty_brain(*self*)

Empty/clear the entire brain of all neurons, connections and states.

19.4.2 Class Variables

Name	Description
activations	**Value:** {}
neuron_pool	**Value:** {}
synapses	**Value:** {}
activation_sequence	**Value:** `[]`
brainmatter	**Value:** {}
learning_algorithm	**Value:** `None`

20 Module copads.nrpy

Numerical Recipes in Python.

References:

- Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T. 1989. Numerical Recipes in Pascal. Cambridge University Press, Cambridge (ISBN 978-0521375160)
- Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T. 1992. Numerical Recipes in C, 2nd edition. Cambridge University Press, Cambridge (ISBN 978-0521431088)

Numerical Recipes in C, 2nd edition is freely browsable online at http://www.nrbook.com/a/bookcpdf.php but is not intended as a substitution for purchasing the book.

Functions will be named as in the references and will be referred to section number. For example, the reference "NRP 5.2" refers to Numerical Recipes in Pascal chapter 5 section 2.

The authors of Numerical Recipes in Pascal (NRP) and Numerical Recipes in C, 2nd edition (NRC2) explicitly allows the reader to analyze the mathematical ideas in the codes within the book and owns the re-implemented functions as stated in NRP and NRC2 that "If you analyze the ideas contained in a program, and then express those ideas in your own distinct implementation, then that new program implementation belongs to you" (page xv of NRP; page xviii of NRC2). Not mentioned in NRP, NRC2 allows the reader a "free licence" (page xviii of NRC2) which allows the reader to make one machine-readable copy of the C codes in the book for his/her own use (not distribution) in his/her work, provided that the source codes are not distributed. As such, the codes in this file will be called by other functions in COPADS but not for direct use by users of COPADS - if you intend to call these functions directly, the simplest way is to own a copy of both NRP and NRC2.

Date created: 19th March 2008

20.1 Functions

bessi(*n*, *x*)

Modified Bessel function I-sub-n(x).

Parameters

`n`: integer, more than 1 - modified n-th Bessel function

`x`: positive integer

Return Value

modified n-th Bessel function of x

See Also: NRP 6.5

Status: Tested function

Since: version 0.1

bessi0(*x*)

Modified Bessel function I-sub-0(x).

Parameters

`x`: float number

Return Value

modified Bessel function base 0 of x

See Also: NRP 6.5

Status: Tested function

Since: version 0.1

bessi1(*x*)

Bessel function I-sub-1(x).

Parameters

`x`: float number

Return Value

float number

See Also: NRP 6.5

Status: Tested function

Since: version 0.1

bessj(*n*, *x*)

Bessel function J-sub-n(x).

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.5

Status: Tested function

Since: version 0.1

bessj0(*x*)

Bessel function J-sub-0(x).

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.4

Status: Tested function

Since: version 0.1

bessj1(*x*)

Bessel function J-sub-1(x).

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.4

Status: Tested function

Since: version 0.1

bessk(*n*, *x*)

Bessel function K-sub-n(x). @see: NRP 6.5

Parameters

`n`: integer, more than 1 - modified n-th Bessel function

`x`: positive integer

Return Value

modified n-th Bessel function of x

bessk0(*x*)

Bessel function K-sub-0(x).

Parameters

`x`: positive integer

Return Value

n-th Bessel function of x

See Also: NRP 6.5

bessk1(*x*)

Bessel function K-sub-1(x).

Parameters

`x`: positive integer

Return Value

n-th Bessel function of x

See Also: NRP 6.5

bessy(*n*, *x*)

Bessel function Y-sub-n(x).

Parameters

`n`: integer, more than 1 - n-th Bessel function

`x`: positive integer

Return Value

n-th Bessel function of x

See Also: NRP 6.4

Status: Tested function

Since: version 0.1

bessy0(x)

Bessel function Y-sub-0(x).

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.4 Depend: bessj0

Status: Tested function

Since: version 0.1

bessy1(x)

Bessel function Y-sub-1(x).

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.4 Depend: bessj1

Status: Tested function

Since: version 0.1

beta(z, w)

Beta function. Depend: gammln

Parameters
`z`: float number

`w`: float number

Return Value
float number

See Also: NRP 6.1, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

betacf(*a*, *b*, *x*)

Continued fraction for incomplete beta function. Adapted from salstat_stats.py of SalStat (www.sf.net/projects/salstat)

See Also: NRP 6.3, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

betai(*a*, *b*, *x*)

Incomplete beta function

I-sub-x(a,b) = 1/B(a,b)*(Integral(0,x) of tˆ(a-1)(1-t)ˆ(b-1) dt)

where a,b>0 and B(a,b) = G(a)*G(b)/(G(a+b)) where G(a) is the gamma function of a.

Adapted from salstat_stats.py of SalStat (www.sf.net/projects/salstat)
Depend: betacf, gammln

See Also: NRP 6.3, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

bico(*n*, *k*)

Binomial coefficient. Returns n!/(k!(n-k)!) Depend: factln, gammln

Parameters

`n`: total number of items

`k`: required number of items

Return Value

floating point number

See Also: NRP 6.1, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

chebev(*a*, *b*, *c*, *m*, *x*)

Chebyshev evaluation.

Parameters

`a`: float number

`b`: float number

`c`: list of Chebyshev coefficients produced by chebft with the same 'a' and 'b'

`m`:

`x`:

Return Value

float number - function value

See Also: NRP 5.6

erf(*x*)

Error function (a special incomplete gamma function) equivalent to gammp(0.5, xˆ2) for x => 0. In this routine, gammp is by-passed and gser and gcf are used directly. Depend: gser. gcf, gammln

Parameters

`x`: float number

Return Value

float number

See Also: NRP 6.2

erfc(*x*)

Complementary error function (a special incomplete gamma function) equivalent to gammq(0.5, xˆ2) which is equivalent to 1 - gammp(0.5, xˆ2) for x => 0.0 Depend: gammp, gammq, gser, gcf, gammln

Parameters

`x`: float number

Return Value

float number

See Also: NRP 6.2

Status: Tested function

Since: version 0.1

erfcc(x)

Complementary error function similar to erfc(x) but with fractional error lesser than 1.2e-7.

Parameters
`x`: float number

Return Value
float number

See Also: NRP 6.2

Status: Tested function

Since: version 0.1

expdev(x)

Depends: ran3

See Also: NRP 7.2

factln(n)

Natural logarithm of factorial: ln(n!)

Parameters
`n`: positive integer

Return Value
natural logarithm of factorial of n

See Also: NRP 6.1, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

gammln(n)

Complete Gamma function.

Parameters

`n`: float number

Return Value

float number

See Also: NRP 6.1, http://mail.python.org/pipermail/python-list/2000-June/671838.html, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

gammp(a, x)

Gamma incomplete function, P(a,x). P(a,x) = (1/gammln(a)) * integral(0, x, (eˆ-t)*(tˆ(a-1)), dt) Depend: gser, gcf, gammln

Parameters

`a`: float number

`x`: float number

Return Value

float number

See Also: NRP 6.2, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

gammq(*a*, *x*)

Incomplete gamma function: Q(a, x) = 1 - P(a, x) = 1 - gammp(a, x) Also commonly known as Q-equation.

See Also:
http://mail.python.org/pipermail/python-list/2000-June/671838.html, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

gcf(*a*, *x*, *itmax*=`200`, *eps*=`3e-07`)

Continued fraction approx'n of the incomplete gamma function.

See Also:
http://mail.python.org/pipermail/python-list/2000-June/671838.html, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

gser(*a*, *x*, *itmax*=`700`, *eps*=`3e-07`)

Series approximation to the incomplete gamma function.

See Also:
http://mail.python.org/pipermail/python-list/2000-June/671838.html, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested function

Since: version 0.1

mdian1(*data*)

Calculates the median of a list of numerical values using sorting.

Parameters
`data`: a 1-dimensional list of numerical data

Return Value
value of median

See Also: NRP 13.2

moment(*data*)

Calculates moment from a list of numerical data. @see: NRP 13.1

Parameters

`data:` a 1-dimensional list of numerical values

Return Value

(ave, adev, sdev, var, skew, kurt) where

- ave = mean
- adev = average deviation
- sdev = standard deviation
- var = variance
- skew = skew
- kurt = kurtosis

qgaus(*a*, *b*, *func*)

See Also: NRP 4.5

cdf_binomial(*k*, *n*, *p*)

Cummulative density function of Binomial distribution. No reference implementation. Depend: betai, betacf, gammln

Parameters

`k:` number of times of event occurrence in n trials

`n:` total number of trials

`p:` probability of event occurrence per trial

Return Value

float number - Binomial probability

See Also: NRP 6.3, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

cdf_poisson(k, x)

Cummulative density function of Poisson distribution from 0 to k - 1 inclusive. No reference implementation. Depend: gammq, gser, gcf, gammln

Parameters

`k`: number of times of event occurrence

`x`: mean of Poisson distribution

Return Value

float number - Poisson probability of k - 1 times of occurrence with the mean of x

See Also: NRP 6.2, Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

20.2 Variables

Name	Description
__package__	**Value:** `'copads'`

21 Module copads.objectdistance

Functions for Calculating Similarity Coefficients between 2 Objects.

Generally, the lower boundary of similar coefficient signifies complete difference (no similarity) while the upper boundary (if any) signifies complete similarity (no difference).

In the following formulae, the following notations will be used

- A = found in both 'original' and 'test'
- B = found in 'original' only
- C = found in 'test' only
- D = not found in either 'original' or 'test'

Date created: 17th August 2005

21.1 Functions

binarize(*data*, *absent*=`0`)

Converts input data in a list of presence or absence of values. For example, binarize([1, 2, 0, 3, 4, 0], 0) –> [1, 1, 0, 1, 1, 0] binarize([1, 2, 0, 3, 4, 0], 2) –> [1, 0, 1, 1, 1, 1]

compare(*original*, *test*, *absent*, *type*=`'Set'`)

Used for processing list-based (positional) or set-based (non-positional) distance of categorical data.

Parameters

`original:` list of original data

`test:` list of data to test against original

`absent:` indicator to define absent data

`type:` {List | Set}

Jaccard(*original*, *test*, *absent*=0, *type*='Set')

Jaccard coefficient for nominal or ordinal data.

Coefficient: *A / (A + B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Sokal_Michener(*original*, *test*, *absent*=0, *type*='Set')

Sokal and Michener coefficient for nominal or ordinal data.

Coefficient: *(A + D) / (A + B + C + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Matching(*original*, *test*, *absent*=0, *type*='Set')

Matching coefficient for nominal or ordinal data

Coefficient: *(A + D) / (2A + B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Dice(*original*, *test*, *absent*=0, *type*='Set')

Dice coefficient for nominal or ordinal data.

Coefficient: *2A / (2A + B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Ochiai(*original*, *test*, *absent*=0, *type*='Set')

Ochiai coefficient for nominal or ordinal data.

Coefficient: *2A / sqrt((A + B)(A + C)))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Ochiai2(*original*, *test*, *absent*=0, *type*='Set')

Ochiai 2 coefficient for nominal or ordinal data, and requires the presence of regions whereby both original and test are not present.

Coefficient: *(A * D) / sqrt((A + B)(A + C)(D + B)(D + C))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Anderberg(*original*, *test*, *absent*=0, *type*='Set')

Anderberg coefficient for nominal or ordinal data.

Coefficient: *A / (A + 2(B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Kulczynski2(*original*, *test*, *absent*=0, *type*='Set')

Second Kulczynski coefficient for nominal or ordinal data.

Coefficient: *((A / (A + B)) + (A / (A + C))) / 2*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Kulczynski(*original*, *test*, *absent*=0, *type*='Set')

First Kulczyski coefficient for nominal or ordinal data.

Coefficient: *A / (B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Forbes(*original*, *test*, *absent*=0, *type*='Set')

Forbes coefficient for nominal or ordinal data.

Coefficient: *A(A + B + C + D) / ((A + B)(A + C))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Hamann(*original*, *test*, *absent*=`0`, *type*=`'Set'`)

Hamann coefficient for nominal or ordinal data.

Coefficient: *((A + D) - (B + C)) / (A + B + C + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Simpson(*original*, *test*, *absent*=`0`, *type*=`'Set'`)

Simpson coefficient for nominal or ordinal data.

Coefficient: *A / min(A + B, A + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Russel_Rao(*original*, *test*, *absent*=0, *type*='Set')

Russel and Rao coefficient for nominal or ordinal data.

Coefficient: *A / (A + B + C + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Roger_Tanimoto(*original*, *test*, *absent*=0, *type*='Set')

Roger and Tanimoto coefficient for nominal or ordinal data.

Coefficient: *(A + D) / (A + 2B + 2C + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Sokal_Sneath(*original*, *test*, *absent*=0, *type*='Set')

Sokal and Sneath coefficient for nominal or ordinal data.

Coefficient: *A / (A + 2B + 2C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Sokal_Sneath2(*original*, *test*, *absent*=0, *type*='Set')

Sokal and Sneath 2 coefficient for nominal or ordinal data.

Coefficient: *(2A + 2D) / (2A + B + C + 2D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Sokal_Sneath3(*original*, *test*, *absent*=0, *type*='Set')

Sokal and Sneath 3 coefficient for nominal or ordinal data.

Coefficient: *(A + D) / (B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Buser(*original*, *test*, *absent*=0, *type*='Set')

Buser coefficient (also known as Baroni-Urbani coefficient) for nominal or ordinal data.

Coefficient: *(sqrt(A * D) + A) / (sqrt(A * D) + A + B + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Fossum(*original*, *test*, *absent*=0, *type*='Set')

Fossum coefficient for nominal or ordinal data.

Coefficient: *((A + B + C + D)(A - 0.5)^2) / ((A + B)(A + C))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

YuleQ(*original*, *test*, *absent*=0, *type*='Set')

Yule Q coefficient (also known as First Yule coefficient) for nominal or ordinal data.

Coefficient: *((A * D) - (B * C)) / ((A * D) + (B * C))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

YuleY(*original*, *test*, *absent*=0, *type*='Set')

Yule Y coefficient (also known as Second Yule coefficient) for nominal or ordinal data.

Coefficient: *(sqrt(A * D) - sqrt(B * C)) / (sqrt(A * D) + sqrt(B * C))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Mcconnaughey(*original*, *test*, *absent*=0, *type*='Set')

McConnaughey coefficient for nominal or ordinal data.

Coefficient: *(A^2 - (B * C)) / (A + B)(A + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Stiles(*original*, *test*, *absent*=0, *type*='Set')

Stiles coefficient for nominal or ordinal data.

Coefficient: *log10(((A + B + C + D)(|(A * D) - (B * C)| - ((A + B + C + D) / 2)) ^ 2) / (A + B)(A + C)(B + D)(C + D))*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Pearson(*original*, *test*, *absent*=0, *type*='Set')

Pearson coefficient for nominal or ordinal data.

Coefficient: *((A * D) - (B * C)) / (A + B)(A + C)(B + D)(C + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Dennis(*original*, *test*, *absent*=0, *type*='Set')

Dennis coefficient for nominal or ordinal data.

Coefficient: *((A * D) - (B * C)) / (A + B + C + D)(A + B)(A + C)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Gower_Legendre(*original*, *test*, *absent*=0, *type*='Set')

Gower and Legendre coefficient for nominal or ordinal data.

Coefficient: *(A + D) / ((0.5 * (B + C)) + A + D)*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Tulloss(*original*, *test*, *absent*=0, *type*='Set')

Tulloss coefficient for nominal or ordinal data.

Coefficient: *sqrt(U * S * R)*

- *U = log(1 + ((min(B, C) + A) / (max(B, C) + A))) / log2*
- *S = 1 / sqrt(log(2 + (min(B, C) / (A + 1))) / log2)*
- *R = log(1 + A / (A + B))log(1 + A / (A + C)) / log2log2*

Parameters

`original`: list of original data

`test`: list of data to test against original

`absent`: user-defined identifier for absent of region, default = 0

`type`: {Set | List}, define whether use Set comparison (non-positional) or list comparison (positional), default = Set

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.3

Hamming(*original*, *test*)

Hamming coefficient for ordinal data - only for positional data.

Coefficient: number of mismatches with respect to position

Parameters

`original`: list of original data

`test`: list of data to test against original

See Also: Ling, MHT. 2010. COPADS, I: Distances Measures between Two Lists or Sets. The Python Papers Source Codes 2:2.

Status: Tested function

Since: version 0.1

Euclidean(*original*, *test*)

Euclidean coefficient for interval or ratio data.

Coefficient: *sqrt(∑(((A + B)(i) - (A + C)(i)) ˆ 2))*

euclidean(original, test) -> euclidean distance between original and test.
Adapted from BioPython

Parameters

`original:` list of original data

`test:` list of data to test against original

Status: Tested function

Since: version 0.1

Minkowski(*original*, *test*, *power*=`3`)

Minkowski coefficient for interval or ratio data.

Coefficient: *power-th root(∑(((A + B)(i) - (A + C)(i)) ˆ power))*

Minkowski Distance is a generalized absolute form of Euclidean Distance.
Minkowski Distance = Euclidean Distance when power = 2

Parameters

`original:` list of original data

`test:` list of data to test against original

`power:` expontential variable

(type=integer)

Status: Tested function

Since: version 0.4

Manhattan(*original*, *test*)

Manhattan coefficient for interval or ratio data.

Coefficient: $\sum$*(abs((A + B)(i) - (A + C)(i)))*

Manhattan Distance is also known as City Block Distance. It is essentially summation of the absolute difference between each element.

Parameters

`original:` list of original data

`test:` list of data to test against original

See Also: Krause, Eugene F. 1987. Taxicab Geometry. Dover. ISBN 0-486-25202-7.

Status: Tested function

Since: version 0.4

Canberra(*original*, *test*)

Canberra coefficient for interval or ratio data.

Coefficient: $\sum$*(abs((A + B)(i) - (A + C)(i)) / abs((A + B)(i) + (A + C)(i)))*

Parameters

`original:` list of original data

`test:` list of data to test against original

See Also: Lance GN and Williams WT. 1966. Computer programs for hierarchical polythetic classification. Computer Journal 9: 60-64.

Status: Tested function

Since: version 0.4

Bray_Curtis(*original*, *test*)

Complement Bray and Curtis coefficient for interval or ratio data. Lower boundary of Bray and Curtis coefficient represents complete similarity (no difference).

Coefficient: *1 - ∑(abs((A + B)(i) - (A + C)(i))) / (∑((A + B)(i)) + ∑((A + C)(i)))*

Parameters

`original`: list of original data

`test`: list of data to test against original

See Also: Bray JR and Curtis JT. 1957. An ordination of the upland forest communities of S. Winconsin. Ecological Monographs 27: 325-349.

Status: Tested function

Since: version 0.4

Cosine(*original*, *test*)

Cosine coefficient for interval or ratio data.

Coefficient: *∑(abs((A + B)(i) * (A + C)(i))) / (∑((A + B) ^ 2) * ∑((A + C) ^ 2))*

Parameters

`original`: list of original data

`test`: list of data to test against original

Status: Tested function

Since: version 0.4

Tanimoto(*original*, *test*)

Tanimoto coefficient for interval or ratio data.

Coefficient: *∑(abs((A + B)(i) * (A + C)(i))) / (∑((A + B) ^ 2) + ∑((A + C) ^ 2) - ∑(abs((A + B)(i) * (A + C)(i))))*

Parameters

`original`: list of original data

`test`: list of data to test against original

Status: Tested function

Since: version 0.4

21.2 Variables

Name	Description
__package__	**Value:** `'copads'`

22 Module copads.operations

Mathematical Operation Routines.

Date created: 19th March 2008

22.1 Functions

asdict(*l*)

Return a dictionary where the keys are the items in the list, with arbitrary values. This is useful for quick testing of membership. Adapted from BioPython.

Return Value
dictionary

items(*l*)

Generate a list of one of each item in l. The items are returned in arbitrary order. Adapted from BioPython.

Return Value
list of items

count(*items*)

Count the number of times each item appears in a list of data. Adapted from BioPython.

Return Value
dict of counts of each item

contents(*items*)

Summarize the contents of the list in terms of the percentages of each item. For example, if an item appears 3 times in a list with 10 items, it is in 0.3 of the list. Adapted from BioPython.

Return Value
dict of item:percentage

itemindex(*l*)

Make an index of the items in the list. The dictionary contains the items in the list as the keys, and the index of the first occurrence of the item as the value. Adapted from BioPython.

Return Value
dict of item : index of item

indexesof(*l*, *fn*)

Return a list of indexes i where fn(l[i]) is true. Adapted from BioPython.

Return Value
list of indexes

take(*l*, *indexes*)

Adapted from BioPython.

Return Value
list of just the indexes from l

take_byfn(*l*, *fn*, *opposite*=`0`)

Adapted from BioPython.

fcmp(*x*, *y*, *precision*)

Return Value
-1, 0, or 1

intd(*x*, *digits_after_decimal*=...)

Represent a floating point number with some digits after the decimal point as an integer. This is useful when floating point comparisons are failing due to precision problems. e.g. intd(5.35, 1) -> 54. Adapted from BioPython.

Return Value
int x, rounded

safe_log(*n*, *zero*=`None`, *neg*=`None`)

Calculate the log of n. If n is 0, returns the value of zero. If n is negative, returns the value of neg. Adapted from BioPython.

Return Value
log(n)

safe_log2(*n*, *zero*=None, *neg*=None)

Calculate the log base 2 of n. If n is 0, returns the value of zero. If n is negative, returns the value of neg. Adapted from BioPython.

Return Value
log(n)

safe_exp(*n*, *under*=None, *over*=None)

Guaranteed not to overflow. Instead of overflowing, it returns the values of 'under' for underflows or 'over' for overflows. Adapted from BioPython.

Return Value
e**n

factorial(*n*)

Calculates and return n! where n is an integer, or will be casted as an integer.

fibonacci(*n*)

Calculates and return the sum of the first n-th Fibonacci number

permutation(*items*, *n*=None)

Generates permutation of n-elements from the list of input items. For example, if n = 2, this function will generate permutations of 2-elements.

Adapted from Raymond Hettinger's comment to
http://code.activestate.com/recipes/474124/

combination(*items*, *n*=None)

Generates combination of n-elements from the list of input items. For example, if n = 2, this function will generate combinations of 2-elements.

Adapted from Raymond Hettinger's comment to
http://code.activestate.com/recipes/474124/

sample_wr(*population*, *k*)

Chooses k random elements (with replacement) from a population. Adapted from Raymond Hettinger's comment to
http://code.activestate.com/recipes/273085/

Since: version 0.2

sample(*population*, *k*)

Chooses k random elements (without replacement) from a population.

Since: version 0.2

summation(*x*)

Sum of all the elements of x

product(*x*)

Product of all the elements of x, also known as series product

22.2 Variables

Name	Description
LOG2	**Value:** `0.69314718056`
__package__	**Value:** `'copads'`

22.3 Class Modulus2

Class for Modulus 2 arithmetics

Status: Tested methods

Since: version 0.1

22.3.1 Methods

__**init**__(*self*, *input*=`0`)

__**add**__(*self*, *other*)

__**mul**__(*self*, *other*)

__**str**__(*self*)

22.4 Class Boolean

Class for Boolean arithmetics

Status: Tested methods

Since: version 0.1

22.4.1 Methods

__**init**__(*self*, *input*=0)

__**add**__(*self*, *other*)

__**mul**__(*self*, *other*)

__**str**__(*self*)

23 Module copads.prioritydictionary

Priority Dictionary and Algorithms.

Date created: 19th March 2008

23.1 Variables

Name	Description
__package__	**Value:** `'copads'`

23.2 Class PriorityDictionary

object
  dict
    copads.prioritydictionary.PriorityDictionary

This data structure acts almost like a dictionary, with two modifications: First, D.smallest() returns the value x minimizing D[x]. For this to work correctly, all values D[x] stored in the dictionary must be comparable. Second, iterating 'for x in D' finds and removes the items from D in sorted order. Each item is not removed until the next item is requested, so D[x] will still return a useful value until the next iteration of the for-loop.

Adapted from: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/117228 Original author: David Eppstein

23.2.1 Methods

__init__(*self*)

Initialize priorityDictionary by creating binary heap of pairs (value,key). Note that changing or removing a dict entry will not remove the old pair from the heap until it is found by smallest() or until the heap is rebuilt.

Return Value

```
new empty dictionary
```

Overrides: object.__init__

smallest(*self*)

Find smallest item after removing deleted items from heap.

__iter__(*self*)

Create destructive sorted iterator of priorityDictionary.

Overrides: dict.__iter__

__setitem__(*self*, *key*, *val*)

Change value stored in dictionary and add corresponding pair to heap. Rebuilds the heap if the number of deleted items grows too large, to avoid memory leakage.

Overrides: dict.__setitem__

setDefault(*self*, *key*, *val*)

Reimplement setdefault to call our customized __setitem__.

Inherited from dict

__cmp__(), __contains__(), __delitem__(), __eq__(), __ge__(), __getattribute__(), __getitem__(), __gt__(), __le__(), __len__(), __lt__(), __ne__(), __new__(), __repr__(), __sizeof__(), clear(), copy(), fromkeys(), get(), has_key(), items(), iteritems(), iterkeys(), itervalues(), keys(), pop(), popitem(), setdefault(), update(), values(), viewitems(), viewkeys(), viewvalues()

Inherited from object

__delattr__(), __format__(), __reduce__(), __reduce_ex__(), __setattr__(), __str__(), __subclasshook__()

23.2.2 Properties

Name	Description
Inherited from object	
__class__	

23.2.3 Class Variables

Name	Description
Inherited from dict	
__hash__	

24 Module copads.ragaraja

Ragaraja Interpreter Date created: 16th August 2012 Licence: Python Software Foundation License version 2

Ragaraja is a derivative and massive extension of Brainfuck. This work is influenced by a large number of Brainfuck derivatives, other esoteric programming languages, and even assembly languages. Probably the most critical difference between Ragaraja and other Brainfuck derivatives is the large number of commands / instructions - 1000 possible commands / instructions, inspired by Nandi (follower of Lord Shiva) who was supposed to be the first author of Kama Sutra and wrote it in 1000 chapters.

Etymology: Ragaraja is the name of a Mahayana Buddhist deity from Esoteric traditions. The Chinese calls him Ai Ran Ming Wang. Ragaraja is one of the Wisdom Kings (a group of Bodhisattvas) and represents the state at which sexual excitement or agitation can be channeled towards enlightenment and passionate love can become compassion for all living things. Hence, I name this compilation/derivative/extension of Brainfuck in 1000 commands/instructions/opcode to signify the epitome, a channeling of raw urge to the love and compassion for and towards every being. May really be viewed as Brainfuck attaining enlightenment or Nirvana. Whoever that can remember all 1000 commands and use it, really deserves an award.

The interpreter environment consists of the following elements:

1. Array/Tape: A circular tape initialized with 30 thousand cells each with zero. This can be visualized as a 30,000 cell register machine. The number of cells can increase or decrease during runtime.
2. Source: The program
3. Input List: A list of data given to the execution environment at initialization.
4. Output List: A list of output from the execution. This may also be used as a secondary tape.

When the program terminates, all 4 elements are returned, and the interpreter terminates itself.

See Also: http://esolangs.org/wiki/Ragaraja

24.1 Functions

instruction_padding(*inst*)

loop_start(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Start loop. Will only enter loop if current cell is more than "0". If current cell is "0" or less, it will go to the end of the loop (command 015). if the loop is not closed, it will go to the end of the source.

loop_end(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

End loop. However, it is possible to have an end loop operator (command 015) without a preceding start loop operator (command 014). In this case, the end loop operator (command 015) will be ignored and execution continues.

tape_move(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Moving tape pointer for more than one increment or decrement.

Instructions handled: 001: Move forward by 5 cells on tape. Equivalent to 5 times of "000". 002: Move forward by 10 cells on tape. Equivalent to 10 times of "000". 003: Move forward by NxN cells on tape where N is the value of the current cell. If N is a decimal, it will move forward by the floor of NxN. For example, if N is 4.2, this operation will tape pointer forward by 17 cells. As NxN is always a positive number, it does not matter if the value of the current cell is positive or negative. 005: Move backward by 5 cells on tape. Equivalent to 5 times of "004". 006: Move backward by 10 cells on tape. Equivalent to 10 times of "004". 007: Move backward by NxN cells on tape where N is the value of the current cell. If N is a decimal, it will move backward by the floor of NxN. For example, if N is 4.2, this operation will tape pointer backward by 17 cells. As NxN is always a positive number, it does not matter if the value of the current cell is positive or negative. 043: Move the tape cell pointer to the first cell. 044: Move the tape cell pointer to the last cell. 045: Move the tape cell pointer to the location determined by the last value of the output list. If the last value of the output list is more than the length of the tape, it will take the modulus of the length of the tape. For example, the last value of the output list is 5, the tape cell pointer will point to the 5th cell on the tape. 061: Move forward by the number of cells signified by the current cell. 062: Move backward by the number of cells signified by the current cell. 140: Move tape pointer to the centre of the tape. If the tape has odd number cells, it will move to the lower cell. For example, this instruction will move the tape pointer to the 500th cell of a 1000-cell tape, or 142nd of a 285-cell tape. 141: Move tape pointer to 1/4 the of the tape. If the tape has odd number cells, it will move to the lower cell. For example, this instruction will move the tape pointer to the 250th cell of a 1000-cell tape, or 71st of a 285-cell tape. 142: Move tape pointer to 3/4 the of the tape. If the tape has odd number cells, it will move to the lower cell. For example, this instruction will move the tape pointer to the 750th cell of a 1000-cell tape, or 213rd of a 285-cell tape. 143: Move tape pointer to the position as the integer value in the current cell. If the value of the cell is larger than the length of the tape, it will move to the modulus of the integer value in the current cell.

accumulations(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Accumulate the tape cell by more than one increment or decrement.

Instructions handled: 009: Increase value of cell by 5. Equivalent to 5 times of "008". 010: Increase value of cell by 10. Equivalent to 10 times of "008". 012: Decrease value of cell by 5. Equivalent to 5 times of "011". 013: Decrease value of cell by 10. Equivalent to 10 times of "011". 032: Double current tape cell value. 033: Half current tape cell value.

nBF_random_op(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

NucleotideBF (nBF) random operations - to simulate ambiguous DNA bases.

Instructions handled: 050: Randomly execute "008" (increment by 1) or "000" (move forward by 1). Equivalent to "R" in NucleotideBF (nBF). 051: Randomly execute "011" (decrement by 1) or "004" (move backward by 1). Equivalent to "Y" in NucleotideBF (nBF). 052: Randomly execute "000" (move forward by 1) or "004" (move backward by 1). Equivalent to "S" in NucleotideBF (nBF). 053: Randomly execute "008" (increment by 1) or "011" (decrement by 1). Equivalent to "W" in NucleotideBF (nBF). 054: Randomly execute "000" (move forward by 1) or "011" (decrement by 1). Equivalent to "K" in NucleotideBF (nBF). 055: Randomly execute "004" (move backward by 1) or "008" (increment by 1). Equivalent to "M" in NucleotideBF (nBF). 056: Randomly execute "000" (move forward by 1) or "004" (move backward by 1) or "011" (decrement by 1). Equivalent to "B" in NucleotideBF (nBF). 057: Randomly execute "000" (move forward by 1) or "008" (increment by 1) or "011" (decrement by 1). Equivalent to "D" in NucleotideBF (nBF). 058: Randomly execute "004" (move backward by 1) or "008" (Increment by 1) or "011" (decrement by 1). Equivalent to "H" in NucleotideBF (nBF). 059: Randomly execute "000" (move forward by 1) or "004" (move backward by 1) or "008" (increment by 1). Equivalent to "V" in NucleotideBF (nBF). 060: Randomly execute "000" (move forward by 1) or "004" (move backward by 1) or "008" (increment by 1) or "011" (decrement by 1). Equivalent to "N" in NucleotideBF (nBF)

tape_size(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Change the length of the tape during runtime.

Instructions handled: 016: Add one cell to the end of the tape. 017: Add 10 cells to the end of the tape. 018: Remove one cell from the end of the tape. If original tape pointer is at the last cell before removal operation, the tape pointer will point to the last cell after removal. 019: Remove 10 cells from the end of the tape. If original tape pointer is at the last cell before removal operation, the tape pointer will point to the last cell after removal. 034: Insert a cell after the current tape cell. For example, if current tape cell is 35, a cell initialized to zero will be added as cell 36. As a result, the tape is 1 cell longer. 035: Delete the current cell. As a result, the tape is 1 cell shorter. 036: Delete the current and append to the end of the output list. As a result, the tape is 1 cell shorter.

source_move(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Moving the source without execution.

Instructions handled: 023: Move source pointer forward by one instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 024: Move source pointer forward by 5 instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 025: Move source pointer forward by 10 instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 026: Move source pointer backward by one instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 027: Move source pointer backward by 5 instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 028: Move source pointer backward by 10 instruction without execution if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. 082: Skip next instruction if current cell is "0". Equivalent to "/" in [[Minimal]]. However, this operation will only execute if there is at least 1 more instruction from the current instruction. 083: Skip the number of instructions equivalent to the absolute integer value of the current cell if the source pointer does not point beyond the length of the source after the move, otherwise, does not move the source pointer. For example, if current cell is "5.6" or "5", the next 5 instructions will be skipped.

set_tape_value(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Set values into tape cell by over-writing the original value.

Instructions handled: 084: Set current tape cell to "0". 085: Set current tape cell to "-1". 086: Set current tape cell to "1". 097: Set the value of the current cell to pi (3.14159265358979323846) 098: Set the value of the current cell to e (2.718281828459045) 187: Set all the cell values in the cells after the current cell to "0" (current cell is not affected). 188: Set all the cell values in the cells before the current cell to "0" (current cell is not affected). 189: Set all values in tape to "0". 190: Set all the cell values in the tape to the value of current cell. 191: Set all the cell values in the tape to the tape position of current cell. 192: Set all the cell values in the cells after the current cell to the value of current cell. 193: Set all the cell values in the cells before the current cell to the value of current cell. 194: Set all the cell values in the cells after the current cell to the tape position of current cell. 195: Set all the cell values in the cells before the current cell to the tape position of current cell.

mathematics(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Performs mathematical and arithmetical operations.

Instructions handled: 065: Add the value of the current cell (n) and (n+1)th cell, and store the value in the current cell. Array[n] = Array[n] + Array[n+1] 066: Add the value of the current cell (n) and first of the input list, and store the value in the current cell. 067: Add the value of the current cell (n) and last of the input list, and store the value in the current cell. 068: Subtract the value of the current cell (n) from (n+1)th cell, and store the value in the current cell. Array[n] = Array[n+1] - Array[n] 069: Subtract the value of the current cell (n) from the first of the input list, and store the value in the current cell. Array[n] = InputList[0] - Array[n] 070: Subtract the value of the current cell (n) from the last of the input list, and store the value in the current cell. Array[n] = InputList[-1] - Array[n] 071: Multiply the value of the current cell (n) and (n+1)th cell, and store the value in the current cell. Array[n] = Array[n+1] * Array[n] 072: Multiply the value of the current cell (n) and first of the input list, and store the value in the current cell. 073: Multiply the value of the current cell (n) and last of the input list, and store the value in the current cell. 074: Divide the value of the current cell (n) from (n+1)th cell, and store the value in the current cell. Array[n] = Array[n+1] / Array[n] 075: Divide the value of the current cell (n) from the first of the input list, and store the value in the current cell. Array[n] = InputList[0] / Array[n] 076: Divide the value of the current cell (n) from the last of the input list, and store the value in the current cell. Array[n] = InputList[-1] - Array[n] 077: Modulus (remainder after division) the value of the current cell (n) from (n+1)th cell, and store the value in the current cell. Array[n] = Array[n+1] % Array[n] 078: Modulus (remainder after division) the value of the current cell (n) from the first of the input list, and store the value in the current cell. Array[n] = InputList[0] % Array[n] 079: Modulus (remainder after division) the value of the current cell (n) from the last of the input list, and store the value in the current cell. Array[n] = InputList[-1] % Array[n] 080: Floor the value of the current cell. For example, if the value of the current cell is 6.7, it will becomes 6. 087: Negate the value of the current cell. Positive value will be negative. Negative value will be positive. Equivalent to "_" in L00P 088: Calculate the sine of the value of the current cell (measured in radians) and replace. Equivalent to "s" in Grin. Array[n] = sine(Array[n]) 089: Calculate the cosine of the value of the current cell (measured in radians) and replace. Equivalent to "c" in Grin. Array[n] = cosine(Array[n]) 090: Calculate the tangent of the value of the current cell (measured in radians) and replace. Equivalent to "t" in Grin. Array[n] = tangent(Array[n]) 091: Calculate the arc sine of the value of the current cell (measured in radians) and replace. Equivalent to "S" in Grin. Array[n] = arcsine(Array[n]) 092: Calculate the arc cosine of the value of the current cell (measured in radians) and replace. Equivalent to "C" in Grin. Array[n] = arccosine(Array[n]) 093: Calculate the arc tangent of the value of the current cell (measured in radians) and replace. Equivalent to "T" in Grin. Array[n] = arctangent(Array[n]) 094: Calculate the reciprocal of the value of the current cell (measured in radians) and replace. Equivalent to "1" in Grin. Array[n] = 1/Array[n] 095: Calculate the square root of the value of the current cell (measured in radians) and replace. Equivalent to "q" in Grin. Array[n] = sqrt(Array[n]) 096: Calculate the natural logarithm of the value of

output_IO(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Using output list as output storage or secondary tape, write and accept values from output list.

Instructions handled: 021: Output current tape cell location and append to the end of the output list. 022: Output current source location and append to the end of the output list. 037: Replace the current tape cell value with the last value of the output list, and delete the last value from the output list. 038: Replace the current tape cell value with the last value of the output list, without deleting the last value from the output list. 039: Replace the current tape cell value with the first value of the output list, and delete the first value from the output list. 040: Replace the current tape cell value with the first value of the output list, without deleting the first value from the output list. 041: Remove first value from the output list. 042: Remove last value from the output list. 172: Append all the cell values in the cells after the current cell to the end of output list (current cell is not appended to output list). 173: Append all the cell values in the cells after the current cell to the end of output list (current cell is not appended to output list), then set the values of all the cells after the current cell to "0" (current cell value is not affected). 174: Append all the cell values in the cells to the end of output list. 175: Append all the cell values in the cells to the end of output list and set the tape to "0".

logic(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Logical operations

Instructions handled: 120: AND operator: Given positive numbers (>0) as True and zero or negative numbers (<=0) as False, store Array[current] AND Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 121: OR operator: Given positive numbers (>0) as True and zero or negative numbers (<=0) as False, store Array[current] OR Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 122: NOT operator: Given positive numbers (>0) as True and zero or negative numbers (<=0) as False, store NOT Array[current] in the current cell (Array[current]) where "0" is False and "1" is True. 123: LESS-THAN operator: Store Array[current] < Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 124: MORE-THAN operator: Store Array[current] > Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 125: EQUAL operator: Store Array[current] = Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 126: NOT-EQUAL operator: Store Array[current] != Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 127: LESS-THAN-OR-EQUAL operator: Store Array[current] <= Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 128: MORE-THAN-OR-EQUAL operator: Store Array[current] => Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. 129: NAND operator: Given positive numbers (>0) as True and zero or negative numbers (<=0) as False, store Array[current] NAND Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. Array[current] NAND Array[current+1] is equivalent to NOT (Array[current] AND Array[current+1]) 130: NOR operator: Given positive numbers (>0) as True and zero or negative numbers (<=0) as False, store Array[current] NOR Array[current+1] in the current cell (Array[current]) where "0" is False and "1" is True. Array[current] NOR Array[current+1] is equivalent to NOT (Array[current] OR Array[current+1])

flipping(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Flipping of execution elements.

Instructions handled: 046: Flip the tape. The original first cell becomes the last cell but the tape pointer does not flip in location. 047: Flip the output list. 048: Flip the instruction list (source) but the source pointer does not flip in location.

input_IO(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Write to and accept values from input list.

Instructions handled: 064: Writes the first value of the input list into the current cell and without removing the value from the input list. If input list is empty, "0" will be written.

tape_manipulate(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Manipulating the tape.

Instructions handled: 081: Swap the value of the current cell (n) and (n+1)th cell. 133: Flip the tape from the cell after the current cell to the end of the tape (temporarily breaking the circularity of the tape). 161: Cut the tape before the current cell (n) and append it to the end of the tape and set tape pointer to 0. <—A—>n<—B—> ==> n<—B—><—A—> 162: Cut the tape after the current cell (n) and append it to the start of the tape and set tape pointer to the last cell. <—A—>n<—B—> ==> <—B—><—A—>n 163: Cut out the current cell and append it to the front of the tape and set tape pointer to 0. <—A—>n<—B—> ==> n<—A—><—B—> 164: Cut out the current cell and append it to the end of the tape and set tape pointer to the last cell. <—A—>n<—B—> ==> <—A—><—B—>n

register_IO(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

```
Read from and write to register from tape cell.

Instructions handled:
201: Store value of current tape cell to register #1
202: Store value of current tape cell to register #2
203: Store value of current tape cell to register #3
204: Store value of current tape cell to register #4
205: Store value of current tape cell to register #5
206: Store value of current tape cell to register #6
207: Store value of current tape cell to register #7
208: Store value of current tape cell to register #8
209: Store value of current tape cell to register #9
210: Store value of current tape cell to register #10
211: Store value of current tape cell to register #11
212: Store value of current tape cell to register #12
213: Store value of current tape cell to register #13
214: Store value of current tape cell to register #14
215: Store value of current tape cell to register #15
216: Store value of current tape cell to register #16
217: Store value of current tape cell to register #17
218: Store value of current tape cell to register #18
219: Store value of current tape cell to register #19
220: Store value of current tape cell to register #20
221: Store value of current tape cell to register #21
222: Store value of current tape cell to register #22
223: Store value of current tape cell to register #23
224: Store value of current tape cell to register #24
225: Store value of current tape cell to register #25
226: Store value of current tape cell to register #26
227: Store value of current tape cell to register #27
228: Store value of current tape cell to register #28
229: Store value of current tape cell to register #29
230: Store value of current tape cell to register #30
231: Store value of current tape cell to register #31
232: Store value of current tape cell to register #32
233: Store value of current tape cell to register #33
234: Store value of current tape cell to register #34
235: Store value of current tape cell to register #35
236: Store value of current tape cell to register #36
237: Store value of current tape cell to register #37
238: Store value of current tape cell to register #38
239: Store value of current tape cell to register #39
240: Store value of current tape cell to register #40
241: Store value of current tape cell to register #41
242: Store value of current tape cell to register #42
243: Store value of current tape cell to register #43
244: Store value of current tape cell to register #44
```

jump_identifier(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Defines jump location within the source tape. These instructions acts as pure identifiers and do not perform any operations.

Instructions handled: 200, 300, 400, 500, 600, 700, 800, 900

not_used(*array*, *apointer*, *inputdata*, *output*, *source*, *spointer*)

Default do-nothing handler for not implemented instructions.

source_filter(*source*, *sfilter*=`['000', '001', '002', '003', '004', '005', '006', '007', ...]`)

Checks a Ragaraja source code string and removes any instructions are not found in the filter

Parameters

`source`: Ragaraja source code string

(type=string)

`sfilter`: list of instructions allowed. Default = ragaraja_v1. Other defined list are nBF_instructions (NucleotideBF) and tested_ragaraja_instructions.

Return Value

Ragaraja source code string

Since: version 0.4

nBF_to_Ragaraja(*source*)

Converts NucleotideBF (nBF) source code to Ragaraja source code

Parameters

`source`: NucleotideBF (nBF) source code

(type=string)

Return Value

Ragaraja source code string

Since: version 0.4

activate_version(*version*=1)

Function to only set tested instructions as usable.

Parameters

`version`: Define the version to activate. Default = 1. Allowable versions are

- 0 (all currently tested instructions)
- 0.1 (using NucleotideBF instructions)
- 1 (as defined in Ling, MHT. 2012. An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy. The Python Papers.)

Since: version 0.4

24.2 Variables

Name	Description
register	**Value:** `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...`
ragaraja	**Value:** `{'000': forward, '001': tape_move, '002': tape_move, '003...`
ragaraja_v1	**Value:** `['000', '001', '002', '003', '004', '005', '006', '007', ...`
ragaraja_v2	**Value:** `['000', '001', '002', '003', '004', '005', '006', '007', ...`
tested_ragaraja_instructions	**Value:** `['000', '001', '002', '003', '004', '005', '006', '007', ...`
nBF_instructions	**Value:** `['000', '004', '008', '011', '020', '050', '051', '052', ...`
__package__	**Value:** `'copads'`

25 Module copads.register_machine

One dimensional tape/register machine Date created: 15th August 2012 Licence: Python Software Foundation License version 2

The machine consists of the following elements:

1. Array/Tape: A circular tape for operations to occur
2. Source: The program
3. Input List: A list of data given to the machine at initialization.
4. Output List: A list of output from the execution. This may also be used as a secondary tape.

When the program terminates, all 4 elements are returned, and the machine terminates itself.

25.1 Functions

interpret(*source*, *functions*, *function_size*=`1`, *inputdata*=`[]`, *array*=`None`, *size*=`30`, *max_instructions*=`1000`)

Interpreter loop.

Parameters

`source:`	Instructions to execute. *(type=string)*
`functions:`	Dictionary of functions / operations.
`function_size:`	Length of each instruction. Default = 1 *(type=integer)*
`inputdata:`	Any input data that the function may need. *(type=list)*
`array:`	The endless tape in a Turing machine which is implemented as a circular list, making it virtually limitless. *(type=list)*
`size:`	Length of the type (array). Default = 30 *(type=integer)*
`max_instructions:`	The maximum number of instructions to execute. Default = 1000 *(type=integer)*

25.2 Variables

Name	Description
__package__	**Value:** `None`

26 Module copads.ring

Ring Data Structures and Algorithms.

Date created: 19th March 2008

26.1 Variables

Name	Description
__package__	**Value:** `None`

26.2 Class RingList

The RingList is a class implementing a circular list. The ring have a fixed size and when it is full and you append a new element, the first one will be deleted. The class lets you access to the data like a python list or like a string.

Adapted from: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/435902 Original author: Flavio Catalani

26.2.1 Methods

__init__(*self*, *length*)

Initializes an empty RingList of maximum given length as given in length parameter

append(*self*, *x*)

Append parameter x to the data structure.

get(*self*)

Retrieves the data.

remove(*self*)

Removes the last element of the ring.

size(*self*)

Returns the current size of the ring.

maxSize(*self*)

Returns the maximum allowed size of the ring.

__**str**__(*self*)

Returns the ring as a string.

27 Module copads.samplestatistics

Data Structures and Algorithms for Data Collected from One or More Samples.

The following functions were adapted from http://www.nmr.mgh.harvard.edu/ Neural_Systems_Group/gary
(assumes 1-dimensional list as input):

- geometricMean
- harmonicMean
- arithmeticMean
- median
- medianScore
- mode
- moment
- variation
- skew
- kurtosis

27.1 Variables

Name	Description
__package__	**Value:** `'copads'`

27.2 Class SingleSample

Class to hold a single sample, and provides calculations on the sample

27.2.1 Methods

__init__(*self*, *data*, *name*=`'Sample 1'`)

geometricMean(*self*)

Calculates the geometric mean of the data

Status: Tested method

Since: version 0.1

harmonicMean(*self*)

Calculates the harmonic mean of the data

Status: Tested method

Since: version 0.1

arithmeticMean(*self*)

Returns the arithematic mean of the data

Status: Tested method

Since: version 0.1

moment(*self*, *moment*=1)

Calculates the nth moment about the mean for the data

skew(*self*)

Returns the skewness of the data, as defined in Numerical Recipies (alternate defn in CRC Standard Probability and Statistics, p.6.)

Status: Tested method

Since: version 0.1

kurtosis(*self*)

Returns the kurtosis of the data, as defined in Numerical Recipies (alternate defn in CRC Standard Probability and Statistics, p.6.)

Status: Tested method

Since: version 0.1

variation(*self*)

Returns the coefficient of variation in percentage, as defined in CRC Standard Probability and Statistics, p.6. Ref:
http://en.wikipedia.org/wiki/Coefficient_of_variation

Status: Tested method

Since: version 0.1

range(*self*)

Returns the range of the data (maximum - minimum)

Status: Tested method

Since: version 0.1

variance(*self*)

Returns the variance of the data

Status: Tested method

Since: version 0.1

__**str**__(*self*)

fullSummary(*self*)

27.2.2 Class Variables

Name	Description
data	**Value:** `None`
rowcount	**Value:** `0`
name	**Value:** `None`
summary	**Value:** {}

27.3 Class SampleDistribution

copads.statisticsdistribution.Distribution
└ **copads.samplestatistics.SampleDistribution**

27.3.1 Methods

__**init**__(*self*, *sampleData*)

Constructor method. The parameters are used to construct the probability distribution.

Overrides: copads.statisticsdistribution.Distribution.__init__ extit(inherited documentation)

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), inverseCDF(), kurtosis(), mean(), mode(), skew(), variance()

27.4 Class TwoSample

Class to hold a two samples, and provides calculations on the samples

27.4.1 Methods

__init__(*self*, *data1*, *name1*, *data2*, *name2*)

getSample(*self*, *name*)

listSamples(*self*)

covariance(*self*)

Calculates covariance using the formula: Cov(xy) = E(xy) - E(x)E(y)

Status: Tested method

Since: version 0.3

linear_regression(*self*)

Calculates the first order linear regression model in the form of "y = mx + c" from the 2 samples where the first sample (data1 and name1 in initialization method) is taken as "X" and the second sample (data2 and name2 in initialization method is taken as "Y".

Return Value

Tuple of (gradient, intercept)

Status: Tested method

Since: version 0.1

mlr(*self*, *order*=2)

pearson(*self*)

Calculates the Pearson's product-moment coefficient by the formula

(N * sum_xy) - (sum_x * sum_y)
———————————————————— ((N * sum_x2 - (sum_x)**2) *
(N * sum_y2 - (sum_y)**2)) ** 0.5

Status: Tested method

Since: version 0.1

27.4.2 Class Variables

Name	Description
sample	**Value:** {}

continued on next page

Name	Description
sample_name	**Value:** []

28 Module copads.set

Sets Data Structure and Algorithms.

Date created: 19th March 2008

28.1 Variables

Name	Description
__package__	**Value:** `None`

28.2 Class Set

The set class. It can contain mutable objects.

Adapted from: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/106469

28.2.1 Methods

__init__(*self*, *seq*=`None`)

The constructor. It can take any object giving an iterator as an optional argument to populate the new set.

__str__(*self*)

copy(*self*)

Shallow copy of a set object.

__contains__(*self*, *elem*)

__len__(*self*)

items(*self*)

Returns a list of the elements in the set.

add(*self*, *elem*)

Add one element to the set.

remove(*self*, *elem*)

Remove an element from the set. Return an error if elem is not in the set.

discard(*self*, *elem*)

Remove an element from the set. Do nothing if elem is not in the set.

__**iter**__(*self*)

__**or**__(*self*, *other*)

Union of two sets.

__**sub**__(*self*, *other*)

Difference of two sets.

__**and**__(*self*, *other*)

Intersection of two sets.

__**add**__(*self*, *other*)

Symmetric difference of two sets.

__**mul**__(*self*, *other*)

Cartesian product of two sets.

__**lt**__(*self*, *other*)

Returns 1 if the lhs set is contained but not equal to the rhs set.

__**le**__(*self*, *other*)

Returns 1 if the lhs set is contained in the rhs set.

__**eq**__(*self*, *other*)

Returns 1 if the sets are equal.

29 Module copads.statisticsdistribution

Classes for Various Statistical Distributions.

References:

- Regress+ A compendium of common probability distributions (version 2.3) by Michael P. McLaughlin (mpmcl@mitre.org) http://www.causascientia.org/math_stat/Dists/Compendium.pdf
- Hand-book on statistical distributions for experimentalists Internal report SUF-PFY/96-01. University of Stockholms by Christian Walck (walck@physto.se)

Distributions:

- BetaDistribution(location, scale, p, q)
  - PowerFunctionDistribution(shape)
- BinomialDistribution(success, trial)
  - BernoulliDistribution(success)
- BradfordDistribution
- BurrDistribution
- CauchyDistribution(location=0.0, scale=1.0)
  - LorentzDistribution (alias of CauchyDistribution)
- ChiDistribution
  - HalfNormalDistribution(location, scale)
  - MaxwellDistribution(scale)
  - RayleighDistribution(scale)
- CosineDistribution(location=0.0, scale=1.0)
- DoubleGammaDistribution
- DoubleWeibullDistribution
- ExponentialDistribution(location=0.0, scale=1.0)
  - NegativeExponentialDistribution (alias of ExponentialDistribution)
- ExtremeLBDistribution
- FDistribution
- FiskDistribution
  - LogLogisticDistribution (alias of FiskDistribution)
- FoldedNormalDistribution
- GammaDistribution
  - ChiSquareDistribution(df)
  - ErlangDistribution(shape)
  - FurryDistribution (alias of GammaDistribution)

- GenLogisticDistribution
- GeometricDistribution(success=0.5)
- GumbelDistribution(location, scale)
  - FisherTippettDistribution (alias of GumbelDistribution)
  - GompertzDistribution (alias of GumbelDistribution)
  - LogWeibullDistribution (alias of GumbelDistribution)
- HyperbolicSecantDistribution
- HypergeometricDistribution
- InverseNormalDistribution
  - WaldDistribution (alias of InverseNormalDistribution)
- LaplaceDistribution
  - BilateralExponentialDistribution (alias of LaplaceDistribution)
  - DoubleExponentialDistribution (alias of LaplaceDistribution)
- LogarithmicDistribution(shape)
- LogisticDistribution
  - SechSquaredDistribution (alias of LogisticDistribution)
- LogNormalDistribution
  - AntiLogNormalDistribution (alias of LogNormalDistribution)
  - CobbDouglasDistribution (alias of LogNormalDistribution)
- NakagamiDistribution
- NegativeBinomialDistribution(success, target)
  - PascalDistribution(success, target)
  - PolyaDistribution (alias of NegativeBinomialDistribution)
- NormalDistribution()
- ParetoDistribution(location=1.0, shape=1.0)
- PoissonDistribution(expectation)
- RademacherDistribution()
- ReciprocalDistribution
- SemicircularDistribution(location=0.0, scale=1.0)
- TDistribution(location=0.0, scale=1.0, shape=2)
- TriangularDistribution
- UniformDistribution(location, scale)
  - RectangularDistribution (alias of UniformDistribution)
- WeibullDistribution
  - FrechetDistribution (alias of WeibullDistribution)

Date created: 17th August 2005

29.1 Functions

ErlangDistribution(*location*, *scale*, *shape*)

Erlang distribution is an alias of Gamma distribution where the shape parameter is an integer.

Parameters

`location:`

`scale:`

`shape:`

Status: Tested method

Since: version 0.2

FurryDistribution(*location*, *scale*, *shape*)

Furry distribution is an alias of Gamma distribution.

Parameters

`location:`

`scale:`

`shape:`

Status: Tested method

Since: version 0.2

FrechetDistribution(***parameters*)

Frechet distribution is an alias of Weibull distribution.

AntiLogNormalDistribution(***parameters*)

Anti-Lognormal distribution is an alias of Lognormal distribution.

BilateralExponentialDistribution(***parameters*)

Bilateral Exponential distribution is an alias of Laplace distribution.

CobbDouglasDistribution(***parameters*)

Cobb-Douglas distribution is an alias of Lognormal distribution.

DoubleExponentialDistribution(***parameters*)

Double Exponential distribution is an alias of Laplace distribution.

FisherTippettDistribution(*location*, *scale*)

Fisher-Tippett distribution is an alias of Gumbel distribution.

Parameters

`location:` η

`scale:` θ

GompertzDistribution(*location*, *scale*)

Gompertz distribution is an alias of Gumbel distribution.

Parameters

`location:` η

`scale:` θ

LogLogisticDistribution(***parameters*)

Log-Logistic distribution is an alias of Fisk distribution.

LogWeibullDistribution(*location*, *scale*)

Log-Weibull distribution is an alias of Gumbel distribution.

Parameters

`location:` η

`scale:` θ

LorentzDistribution(***parameters*)

Lorentz distribution is an alias of Cauchy distribution.

NegativeExponentialDistribution(***parameters*)

Negative-exponential distribution is an alias of Exponential distribution.

PolyaDistribution(*success*, *target*)

Polya distribution is an alias of Negative Binomial distribution.

Parameters

`success:` probability of success; 0 <= success <= 1

`target:` a constant, target number of successes

RectangularDistribution(***parameters*)

Rectangular distribution is an alias of Uniform distribution.

SechSquaredDistribution(***parameters*)

Sech-squared distribution is an alias of Logistic distribution.

WaldDistribution(***parameters*)

Wald distribution is an alias of Inverse Normal distribution.

29.2 Variables

Name	Description
BACKHOUSE	**Value:** `1.45607494858`
CFRACT	**Value:** `1.0306408341`
CGOLD	**Value:** `0.38196601125`
FRODA	**Value:** `6.58088599102`
GAMMA	**Value:** `0.577215664902`
GOLD	**Value:** `1.61803398875`
GOLOMB	**Value:** `0.624329988544`
INV2PI	**Value:** `0.159154943092`
INVLN10	**Value:** `0.434294481903`
INVLN2	**Value:** `1.44269504089`
INVPI180	**Value:** `57.2957795131`
INVSQRT2PI	**Value:** `0.398942280401`
LEHMER	**Value:** `0.592632718292`
LENGYEL	**Value:** `1.09868580553`
LN10	**Value:** `2.30258509299`
LN2	**Value:** `0.69314718056`
LNPI	**Value:** `1.14472988585`
PARIS	**Value:** `1.09864196439`
PI	**Value:** `3.14159265359`
PI180	**Value:** `0.0174532925199`
PI2	**Value:** `6.28318530718`
PIDIV2	**Value:** `1.57079632679`
PORTER	**Value:** `1.46707807943`
RABBIT	**Value:** `0.709803442861`
SQRT2	**Value:** `1.41421356237`
SQRT2PI	**Value:** `2.50662827463`
ZETA9	**Value:** `1.00200839283`
__package__	**Value:** `'copads'`

29.3 Class Distribution

Known Subclasses: copads.statisticsdistribution.BernoulliDistribution, copads.statisticsdistribution.Bet
copads.statisticsdistribution.BinomialDistribution, copads.statisticsdistribution.BradfordDistribution,
copads.statisticsdistribution.BurrDistribution, copads.statisticsdistribution.CauchyDistribution,
copads.statisticsdistribution.ChiDistribution, copads.statisticsdistribution.GammaDistribution,
copads.statisticsdistribution.CosineDistribution, copads.statisticsdistribution.DoubleGammaDistribution,
copads.statisticsdistribution.DoubleWeibullDistribution, copads.statisticsdistribution.ExponentialDistribu
copads.statisticsdistribution.ExtremeLBDistribution, copads.statisticsdistribution.FDistribution,
copads.statisticsdistribution.FiskDistribution, copads.statisticsdistribution.FoldedNormalDistribution,
copads.statisticsdistribution.GenLogisticDistribution, copads.statisticsdistribution.GeometricDistribution,
copads.statisticsdistribution.GumbelDistribution, copads.statisticsdistribution.HalfNormalDistribution,
copads.statisticsdistribution.HyperbolicSecantDistribution, copads.statisticsdistribution.HypergeometricD
copads.statisticsdistribution.LaplaceDistribution, copads.statisticsdistribution.LogNormalDistribution,
copads.statisticsdistribution.LogarithmicDistribution, copads.statisticsdistribution.LogisticDistribution,
copads.statisticsdistribution.MaxwellDistribution, copads.statisticsdistribution.NakagamiDistribution,
copads.statisticsdistribution.NegativeBinomialDistribution, copads.statisticsdistribution.NormalDistributi
copads.statisticsdistribution.ParetoDistribution, copads.statisticsdistribution.PascalDistribution,
copads.statisticsdistribution.PoissonDistribution, copads.statisticsdistribution.PowerFunctionDistribution,
copads.statisticsdistribution.RademacherDistribution, copads.statisticsdistribution.RayleighDistribution,
copads.statisticsdistribution.ReciprocalDistribution, copads.statisticsdistribution.SemicircularDistribution
copads.statisticsdistribution.TDistribution, copads.statisticsdistribution.TriangularDistribution,
copads.statisticsdistribution.UniformDistribution, copads.statisticsdistribution.WeiBullDistribution,
copads.samplestatistics.SampleDistribution

Abstract class for all statistical distributions. Due to the large variations of parameters for each distribution, it is unlikely to be able to standardize a parameter list for each method that is meaningful for all distributions. Instead, the parameters to construct each distribution is to be given as keyword arguments.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

29.3.1 Methods

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability. CDF is also known as density function.

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

__init__(*self*, ***parameters*)

Constructor method. The parameters are used to construct the probability distribution.

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

kurtosis(*self*)

Gives the kurtosis of the sample.

mean(*self*)

Gives the arithmetic mean of the sample.

mode(*self*)

Gives the mode of the sample, if closed-form is available.

skew(*self*)

Gives the skew of the sample.

variance(*self*)

Gives the variance of the sample.

29.4 Class BetaDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.BetaDistribution**

Class for Beta Distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.4.1 Methods

__init__(*self*, *location*, *scale*, *p*, *q*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:` upper bound

`p:` shape parameter. Although no upper bound but seldom exceed 10.

`q:` shape parameter. Although no upper bound but seldom exceed 10.

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

moment(*self*, *r*)

Gives the r-th moment of the sample.

random(*self*)

Gives a random number based on the distribution.

29.5 Class BinomialDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.BinomialDistribution

Class for Binomial Distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.5.1 Methods

__init__(*self*, *success*=0.5, *trial*=1000)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`success:` probability of success; 0 <= success <= 1

`trial:` number of Bernoulli trials

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=0, *step*=1)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.6 Class CauchyDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.CauchyDistribution**

Class for Cauchy Distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.6.1 Methods

__init__(*self*, *location*=`0.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` the mean; default = 0.0

`scale:` spread of the distribution, λ; default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

median(*self*)

Gives the median of the sample.

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*, *seed*)

Gives a random number based on the distribution.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew(), variance()

29.7 Class CosineDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.CosineDistribution**

Cosine distribution is sometimes used as a simple approximation to Normal distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.7.1 Methods

__init__(*self*, *location*=`0.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` the mean; default = 0.0

`scale:` spread of the distribution, λ; default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

median(*self*)

Gives the median of the sample.

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 13rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

29.8 Class ExponentialDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.ExponentialDistribution

Exponential distribution is the continuous version of Geometric distribution. It is also a special case of Gamma distribution where shape = 1

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and

Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.8.1 Methods

__init__(*self*, *location*=`0.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` position of the distribution, default = 0.0

`scale:` spread of the distribution, λ; default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

median(*self*)

Gives the median of the sample.

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*)

Gives a random number based on the distribution.

29.9 Class FDistribution

copads.statisticsdistribution.Distribution
└
copads.statisticsdistribution.FDistribution

Class for F Distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.9.1 Methods

__init__(*self*, *df1*=1, *df2*=1)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`df1`: degrees of freedom for numerator

`df2`: degrees of freedom for denorminator

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), mode(), skew(), variance()

29.10 Class GammaDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.GammaDistribution**

Known Subclasses: copads.statisticsdistribution.ChiSquareDistribution

Class for Gamma Distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.10.1 Methods

__init__(*self*, *location*, *scale*, *shape*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:`

`shape:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

PDF()

29.11 Class ChiSquareDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.GammaDistribution

copads.statisticsdistribution.ChiSquareDistribu

Chi-square distribution is a special case of Gamma distribution where location = 0, scale = 2 and shape is twice that of the degrees of freedom.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.11.1 Methods

__init__(*self*, *df*=2)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`df`: degrees of freedom

Overrides: copads.statisticsdistribution.Distribution.__init__

Inherited from copads.statisticsdistribution.GammaDistribution(Section 29.10)

CDF(), inverseCDF(), kurtosis(), mean(), mode(), qmean(), qmode(), skew(), variance()

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

PDF()

29.12 Class GeometricDistribution

copads.statisticsdistribution.Distribution
└
copads.statisticsdistribution.GeometricDistribution

Geometric distribution is the discrete version of Exponential distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.12.1 Methods

__init__(*self*, *success*=`0.5`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`success`: probability of success; 0 <= success <= 1; default = 0.5

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=1, *step*=1)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew()

29.13 Class HypergeometricDistribution

copads.statisticsdistribution.Distribution ┐
copads.statisticsdistribution.HypergeometricDistribut

Class for Hypergeometric distribution

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.13.1 Methods

__init__(*self*, *sample_size*, *population_size*=100, *population_success*=50)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`sample_size:` sample size (not more than population size)
(type=integer)

`population_size:` population size; default = 100
(type=integer)

`population_success:` number of successes in the population (cannot be more than population size); default = 10
(type=integer)

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *sample_success*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value (sample_success, an integer that is not more than sample size) on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *sample_success*)

Partial Distribution Function, which gives the probability for the particular value of x (sample_success, an integer that is not more than sample size), or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=1, *step*=1)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew()

29.14 Class LogarithmicDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.LogarithmicDistribution**

Class for Logarithmic Distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.14.1 Methods

__init__(*self*, *shape*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`shape`: the spread of the distribution

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew()

29.15 Class NormalDistribution

copads.statisticsdistribution.Distribution ┐

copads.statisticsdistribution.NormalDistribution

Class for standardized normal distribution (area under the curve = 1)

See Also: Ling, MHT. 2009. Ten Z-Test Routines from Gopal Kanji's 100 Statistical Tests. The Python Papers Source Codes 1:5

Status: Tested method

Since: version 0.1

29.15.1 Methods

__init__(*self*)

Constructor method. The parameters are used to construct the probability distribution.

Overrides: copads.statisticsdistribution.Distribution.__init__ extit(inherited documentation)

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Calculates the density (probability) at x by the formula f(x) = 1/(sqrt(2 pi) sigma) eˆ-((xˆ2/(2 sigmaˆ2)) where mu is the mean of the distribution and sigma the standard deviation.

Parameters

`x`: probability at x

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`-10.0`, *end*=`10.0`, *error*=`1e-07`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis, together with the cumulative probability.

Parameters

`probability:` probability under the curve from -infinity

`start:` lower boundary of calculation (default = -10)

`end:` upper boundary of calculation (default = 10)

`error:` error between the given and calculated probabilities (default = 10e-8)

Return Value

Returns a tuple (start, cprob) where 'start' is the standard deviation for the area under the curve from -infinity to the given 'probability' (+/- step). 'cprob' is the calculated area under the curve from -infinity to the returned 'start'.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean extit(inherited documentation)

mode(*self*)

Gives the mode of the sample, if closed-form is available.

Overrides: copads.statisticsdistribution.Distribution.mode extit(inherited documentation)

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis extit(inherited documentation)

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew extit(inherited documentation)

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance extit(inherited documentation)

random(*self*)

Gives a random number based on the distribution.

29.16 Class PoissonDistribution

copads.statisticsdistribution.Distribution ┐
copads.statisticsdistribution.PoissonDistribution

Class for Poisson Distribution. Poisson distribution is binomial distribution with very low success - that is, for rare events.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.16.1 Methods

__init__(*self*, *expectation*=`0.001`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`expectation`: mean success probability; λ

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.001`, *step*=`1`)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew()

29.17 Class SemicircularDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.SemicircularDistribution**

Class for Semicircular Distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.17.1 Methods

__init__(*self*, *location*=`0.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` mean of the distribution, default = 0.0

`scale:` spread of the distribution, default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`-10.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

29.18 Class TDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.TDistribution**

Class for Student's t-distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.18.1 Methods

__init__(*self*, *location*=0.0, *scale*=1.0, *shape*=2)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` default = 0.0

`scale:` default = 1.0

`shape:` degrees of freedom; default = 2

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Calculates the density (probability) at x with n-th degrees of freedom as *f(x) = Γ((n+1)/2) / (sqrt(n * pi) Γ(n/2)) (1 + x^2/n)^-((n+1)/2)*

for all real x. It has mean 0 (for n > 1) and variance n/(n-2) (for n > 2).

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`-10.0`, *end*=`10.0`, *error*=`1e-07`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis, together with the cumulative probability.

Parameters

`probability:` probability under the curve from -infinity

`start:` lower boundary of calculation (default = -10)

`end:` upper boundary of calculation (default = 10)

`error:` error between the given and calculated probabilities (default = 10e-8)

Return Value

Returns a tuple (start, cprob) where 'start' is the standard deviation for the area under the curve from -infinity to the given 'probability' (+/- step). 'cprob' is the calculated area under the curve from -infinity to the returned 'start'.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.19 Class TriangularDistribution

copads.statisticsdistribution.Distribution ┐
copads.statisticsdistribution.TriangularDistribution

Class for Triangular Distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.19.1 Methods

__init__(*self*, *upper_limit*, *peak*, *lower_limit*=0)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`upper_limit:` upper limit of the distrbution
(type=float)

`peak:` peak of the distrbution, which has to be between the lower and upper limits of the distribution
(type=float)

`lower_limit:` lower limit of the distrbution, default = 0
(type=float)

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

29.20 Class UniformDistribution

copads.statisticsdistribution.Distribution ┐
copads.statisticsdistribution.UniformDistribution

Class for Uniform distribution.

See Also: Ling, MHT. 2009. Compendium of Distributions, I: Beta, Binomial, Chi- Square, F, Gamma, Geometric, Poisson, Student's t, and Uniform. The Python Papers Source Codes 1:4

Status: Tested method

Since: version 0.2

29.20.1 Methods

__init__(*self*, *location*, *scale*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*)

Partial Distribution Function, which gives the probability for particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

median(*self*)

Gives the median of the sample.

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

random(*self*, *lower*, *upper*)

Gives a random number based on the distribution.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

mode()

29.21 Class WeiBullDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.WeiBullDistribution**

Class for Weibull distribution.

See Also: Chen, KFQ, Ling, MHT. 2013. COPADS III (Compendium of Distributions II): Cauchy, Cosine, Exponential, Hypergeometric, Logarithmic, Semicircular, Triangular, and Weibull. The Python Papers Source Codes 5: 2.

Status: Tested method

Since: version 0.4

29.21.1 Methods

__init__(*self*, *location*=`1.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` position of the distribution, default = 1.0

`scale:` shape of the distribution, default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under trobability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

median(*self*)

Gives the median of the sample.

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

random(*self*)

Gives a random number based on the distribution.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), mean(), skew(), variance()

29.22 Class BernoulliDistribution

copads.statisticsdistribution.Distribution ┐

copads.statisticsdistribution.BernoulliDistribution

Bernoulli distribution is a special case of Binomial distribution where where number of trials = 1

29.22.1 Methods

__init__(*self*, *success*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`success`: probability of success; 0 <= success <= 1

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0`, *step*=`1`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.23 Class BradfordDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.BradfordDistribution**

Class for Bradford distribution.

29.23.1 Methods

__init__(*self*, *location*, *scale*, *shape*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:` upper bound

`shape:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*, *seed*)

Gives a random number based on the distribution.

29.24 Class BurrDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.BurrDistribution**

Burr distribution is the generalization of Fisk distribution. Burr distribution with D = 1 becomes Fisk distribution.

29.24.1 Methods

__init__(*self*, *location*, *scale*, *C*, *D*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:`

`C:` shape

`D:` shape

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*, *seed*)

Gives a random number based on the distribution.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew()

29.25 Class ChiDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.ChiDistribution**

Class for Chi distribution.

29.25.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.26 Class DoubleGammaDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.DoubleGammaDistributi**

Double Gamma distribution is the signed version of Gamma distribution.

29.26.1 Methods

__init__(*self*, *location*, *scale*, *shape*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:`

`shape:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), mode()

29.27 Class DoubleWeibullDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.DoubleWeibullDistributi**

Double Weibull distribution is the signed version of Weibull distribution.

29.27.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.28 Class ExtremeLBDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.ExtremeLBDistribution**

Class for Extreme LB distribution.

29.28.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.29 Class FiskDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.FiskDistribution**

Class for Fisk distribution.

29.29.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.30 Class FoldedNormalDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.FoldedNormalDistributio**

Class for Folded Normal distribution.

29.30.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.31 Class GenLogisticDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.GenLogisticDistribution**

Generalized Logistic distribution is a generalization of Logistic distribution. It becomes Logistic distribution when shape = 1

29.31.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.32 Class GumbelDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.GumbelDistribution**

Class for Gumbel Distribution.

29.32.1 Methods

__**init**__(*self*, *location*, *scale*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` η

`scale:` θ

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

median(*self*)

Gives the median of the sample.

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*, *seed*)

Gives a random number based on the distribution.

29.33 Class HalfNormalDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.HalfNormalDistribution

Half Normal distribution is a special case of Chi distribution where shape (also degrees of freedom) = 1, and Folded Normal distribution where location = 0

29.33.1 Methods

__init__(*self*, ***parameters*)

Constructor method. The parameters are used to construct the probability distribution.

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.34 Class HyperbolicSecantDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.HyperbolicSecantDistrib**

Class for Hyperbolic Secant Distribution.

29.34.1 Methods

__init__(*self*, *location*, *scale*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:`

`scale:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

29.35 Class LaplaceDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.LaplaceDistribution

29.35.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.36 Class LogisticDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.LogisticDistribution**

29.36.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.37 Class LogNormalDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.LogNormalDistribution**

29.37.1 Methods

__init__(*self*, *a*, *b*)

Constructor method. The parameters are used to construct the probability distribution.

Overrides: copads.statisticsdistribution.Distribution.__init__

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

random(*self*)

Gives a random number based on the distribution.

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), kurtosis(), mode(), skew(), variance()

29.38 Class MaxwellDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.MaxwellDistribution**

Maxwell distribution is a special case of Chi distribution where location = 0 and shape (degrees of freedom) = 3

29.38.1 Methods

__init__(*self*, *scale*)

Constructor method.

Parameters

`scale:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.39 Class NakagamiDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.NakagamiDistribution

29.39.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

29.40 Class NegativeBinomialDistribution

copads.statisticsdistribution.Distribution ┐

copads.statisticsdistribution.NegativeBinomialDistrib

Class for Negative Binomial Distribution.

29.40.1 Methods

__**init**__(*self*, *success*, *target*)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`success:` probability of success; 0 <= success <= 1

`target:` a constant, target number of successes

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=0, *step*=1)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), skew(), variance()

29.41 Class ParetoDistribution

copads.statisticsdistribution.Distribution ┐

copads.statisticsdistribution.ParetoDistribution

Class for Pareto Distribution.

29.41.1 Methods

__init__(*self*, *location*=`1.0`, *scale*=`1.0`)

Constructor method. The parameters are used to construct the probability distribution.

Parameters

`location:` also the scale; default = 1.0

`scale:` λ; default = 1.0

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

median(*self*)

Gives the median of the sample.

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

quantile1(*self*)

Gives the 1st quantile of the sample.

quantile3(*self*)

Gives the 3rd quantile of the sample.

qmean(*self*)

Gives the quantile of the arithmetic mean of the sample.

qmode(*self*)

Gives the quantile of the mode of the sample.

random(*self*)

Gives a random number based on the distribution.

29.42 Class PascalDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.PascalDistribution

Class for Pascal Distribution. Pascal Distribution is a form of Negative Binomial Distribution

where the 'target' is an integer

29.42.1 Methods

__init__(*self*, *success*, *target*)

Constructor method.

Parameters

`success`: probability of success; 0 <= success <= 1

`target`: a constant, target number of successes

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.43 Class PowerFunctionDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.PowerFunctionDistributi**

Class for Power Function Distribution. It is a form of Beta Distribution.

29.43.1 Methods

__init__(*self*, *shape*)

Constructor method.

Parameters

`shape:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.44 Class RademacherDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.RademacherDistribution

Class for Rademacher Distribution.

29.44.1 Methods

__init__(*self*)

Constructor method.

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

kurtosis(), mode()

29.45 Class RayleighDistribution

copads.statisticsdistribution.Distribution
└ **copads.statisticsdistribution.RayleighDistribution**

Rayleigh distribution is a special case of Chi distribution where location = 0 and shape (degrees of freedom) = 2

29.45.1 Methods

__init__(*self*, *scale*)

Constructor method.

Parameters

`scale:`

Overrides: copads.statisticsdistribution.Distribution.__init__

CDF(*self*, *x*)

Cummulative Distribution Function, which gives the cummulative probability (area under the probability curve) from -infinity or 0 to a give x-value on the x-axis where y-axis is the probability.

Overrides: copads.statisticsdistribution.Distribution.CDF

PDF(*self*, *x*)

Partial Distribution Function, which gives the probability for the particular value of x, or the area under probability distribution from x-h to x+h for continuous distribution.

Overrides: copads.statisticsdistribution.Distribution.PDF

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

mean(*self*)

Gives the arithmetic mean of the sample.

Overrides: copads.statisticsdistribution.Distribution.mean

mode(*self*)

Gives the mode of the sample.

Overrides: copads.statisticsdistribution.Distribution.mode

kurtosis(*self*)

Gives the kurtosis of the sample.

Overrides: copads.statisticsdistribution.Distribution.kurtosis

skew(*self*)

Gives the skew of the sample.

Overrides: copads.statisticsdistribution.Distribution.skew

variance(*self*)

Gives the variance of the sample.

Overrides: copads.statisticsdistribution.Distribution.variance

29.46 Class ReciprocalDistribution

copads.statisticsdistribution.Distribution

copads.statisticsdistribution.ReciprocalDistribution

29.46.1 Methods

inverseCDF(*self*, *probability*, *start*=`0.0`, *step*=`0.01`)

It does the reverse of CDF() method, it takes a probability value and returns the corresponding value on the x-axis.

Overrides: copads.statisticsdistribution.Distribution.inverseCDF

Inherited from copads.statisticsdistribution.Distribution(Section 29.3)

CDF(), PDF(), __init__(), kurtosis(), mean(), mode(), skew(), variance()

30 Module copads.tree

Tree Data Structures and Algorithms.

Date created: 19th March 2008

30.1 Variables

Name	Description
BLACK	**Value:** `0`
RED	**Value:** `1`
__package__	**Value:** `'copads'`

30.2 Class RBTreeIter

object
└ **copads.tree.RBTreeIter**

30.2.1 Methods

__init__(*self*, *tree*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__iter__(*self*)

Return the current item in the container

next(*self*)

Return the next item in the container Once we go off the list we stay off even if the list changes

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

30.2.2 Properties

Name	Description
Inherited from object	
__class__	

30.3 Class OrderedBinaryTree

Adapted from http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/286239 Original author: Lawrence Oluyede

30.3.1 Methods

__init__(*self*)

Initializes the root member

addNode(*self*, *data*)

insert(*self*, *root*, *data*)

lookup(*self*, *root*, *target*)

minValue(*self*, *root*)

maxDepth(*self*, *root*)

size(*self*, *root*)

printTree(*self*, *root*)

printRevTree(*self*, *root*)

30.4 Class RBTree

object
└ **copads.tree.RBTree**

Known Subclasses: copads.tree.RBDict, copads.tree.RBList

30.4.1 Methods

__init__(*self*, *cmpfn*=<built-in function cmp>, *unique*=True)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__len__(*self*)

__del__(*self*)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

__iter__(*self*)

rotateLeft(*self*, *x*)

rotateRight(*self*, *x*)

insertFixup(*self*, *x*)

insertNode(*self*, *key*, *value*)

deleteFixup(*self*, *x*)

deleteNode(*self*, *z*, *all*=True)

findNode(*self*, *key*)

traverseTree(*self*, *f*)

nodesByTraversal(*self*)

return all nodes as a list

nodes(*self*)

return all nodes as a list

firstNode(*self*)

lastNode(*self*)

nextNode(*self*, *prev*)

returns None if there isn't one

prevNode(*self*, *next*)

returns None if there isn't one

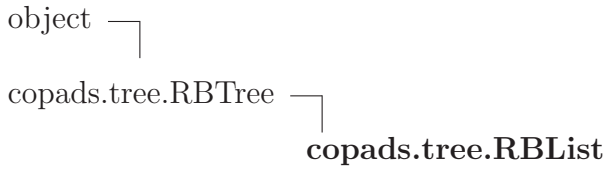
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

30.4.2 Properties

Name	Description
Inherited from object	
__class__	

30.5 Class RBList



List class uses same object for key and value Assumes you are putting sortable items into the list.

30.5.1 Methods

__init__(*self*, *list*=[], *cmpfn*=<built-in function cmp>, *unique*=True)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__getitem__(*self*, *index*)

__delitem__(*self*, *index*)

__contains__(*self*, *item*)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

findNodeByIndex(*self*, *index*)

insert(*self*, *item*)

append(*self*, *item*)

count(*self*, *item*)

index(*self*, *item*)

extend(*self*, *otherList*)

pop(*self*, *index*=None)

remove(*self*, *item*, *all*=True)

reverse(*self*)

sort(*self*)

clear(*self*)

delete all entries

values(*self*)

reverseValues(*self*)

Inherited from copads.tree.RBTree(Section 30.4)

__del__(), __iter__(), __len__(), __repr__(), deleteFixup(), deleteNode(), findNode(), firstNode(), insertFixup(), insertNode(), lastNode(), nextNode(), nodes(), nodesByTraversal(), prevNode(), rotateLeft(), rotateRight(), traverseTree()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

30.5.2 Properties

Name	Description
Inherited from object	
__class__	

30.6 Class RBDict

object

copads.tree.RBTree

copads.tree.RBDict

30.6.1 Methods

__init__(*self*, *dict*={}, *cmpfn*=<`built-in function cmp`>)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

__getitem__(*self*, *key*)

__setitem__(*self*, *key*, *value*)

__delitem__(*self*, *key*)

get(*self*, *key*, *default*=`None`)

keys(*self*)

values(*self*)

items(*self*)

has_key(*self*, *key*)

clear(*self*)

delete all entries

copy(*self*)

return shallow copy

update(*self*, *other*)

Add all items from the supplied mapping to this one.

Will overwrite old entries with new ones.

setdefault(*self*, *key*, *value*=`None`)

Inherited from copads.tree.RBTree(Section 30.4)

__del__(), __iter__(), __len__(), deleteFixup(), deleteNode(), findNode(), firstNode(), insertFixup(), insertNode(), lastNode(), nextNode(), nodes(), nodesByTraversal(), prevNode(), rotateLeft(), rotateRight(), traverseTree()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

30.6.2 Properties

Name	Description
Inherited from object	
__class__	

31 Module copads.treenodes

Node Classes for Tree Data Structures.

Date created: 19th March 2008

31.1 Variables

Name	Description
BLACK	**Value:** `0`
RED	**Value:** `1`
__package__	**Value:** `None`

31.2 Class RBNode

object
└
copads.treenodes.RBNode

31.2.1 Methods

__init__(*self*, *key*=`None`, *value*=`None`, *color*=`1`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

__nonzero__(*self*)

__len__(*self*)

imitate sequence

__**getitem**__(*self*, *index*)
imitate sequence

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

31.2.2 Properties

Name	Description
Inherited from object	
__class__	

31.3 Class BinaryNode

31.3.1 Methods

__**init**__(*self*, *data*)

31.4 Class ThreeNode

31.4.1 Methods

__**init**__(*self*, *data*)

31.5 Class FourNode

31.5.1 Methods

__**init**__(*self*, *data*)

Index

copads *(package)*, 2–4
  copads.array *(module)*, 5–6
    copads.array.ParallelArray *(class)*, 5–6
  copads.bag *(module)*, 7–9
    copads.bag.Bag *(class)*, 7–9
  copads.colormap *(module)*, 10
    copads.colormap.floatRgb *(function)*, 10
    copads.colormap.htmlRgb *(function)*, 10
    copads.colormap.rgb *(function)*, 10
    copads.colormap.strRgb *(function)*, 10
  copads.constants *(module)*, 11–12
  copads.copadsexceptions *(module)*, 13–46
    copads.copadsexceptions.ArrayError *(class)*, 44–45
    copads.copadsexceptions.CopadsError *(class)*, 14–15
    copads.copadsexceptions.DistanceError *(class)*, 38–39
    copads.copadsexceptions.DistanceInputSizeError *(class)*, 39–40
    copads.copadsexceptions.DistributionError *(class)*, 33–34
    copads.copadsexceptions.DistributionFunctionError *(class)*, 37–38
    copads.copadsexceptions.DistributionParameterError *(class)*, 35–37
    copads.copadsexceptions.EdgeNotFoundError *(class)*, 25–26
    copads.copadsexceptions.FunctionParameterTypeError *(class)*, 42–43
    copads.copadsexceptions.FunctionParameterValueError *(class)*, 43–44
    copads.copadsexceptions.GraphEdgeSizeMismatchError *(class)*, 30–31
    copads.copadsexceptions.GraphError *(class)*, 24–25
    copads.copadsexceptions.GraphParameterError *(class)*, 31–32
    copads.copadsexceptions.MatrixAdditionError *(class)*, 18–19
    copads.copadsexceptions.MatrixArithmeticError *(class)*, 15–17
    copads.copadsexceptions.MatrixDeterminantError *(class)*, 23–24
    copads.copadsexceptions.MatrixError *(class)*, 15
    copads.copadsexceptions.MatrixMinorError *(class)*, 22–23
    copads.copadsexceptions.MatrixMultiplicationError *(class)*, 17–18
    copads.copadsexceptions.MatrixSquareError *(class)*, 19–20
    copads.copadsexceptions.MatrixTraceError *(class)*, 20–22
    copads.copadsexceptions.MaxIterationsException *(class)*, 45–46
    copads.copadsexceptions.NormalDistributionTypeErr *(class)*, 34–35
    copads.copadsexceptions.NotAdjacencyGraphMatrixE *(class)*, 28–30
    copads.copadsexceptions.StatisticsError *(class)*, 32–33
    copads.copadsexceptions.TreeError *(class)*, 40–41
    copads.copadsexceptions.TreeNodeTypeError *(class)*, 41–42
    copads.copadsexceptions.UnknownGraphMatrixError *(class)*, 27–28
    copads.copadsexceptions.VertexNotFoundError *(class)*, 26–27
  copads.dataframe *(module)*, 47–49
    copads.dataframe.dataframe *(class)*, 47–49
  copads.dose_entities *(module)*, 50–59
    copads.dose_entities.Organism *(class)*, 50–53
    copads.dose_entities.Population *(class)*, 53–55
    copads.dose_entities.World *(class)*, 55–59
  copads.dose_executor *(module)*, 59–60
    copads.dose_executor.set_instruction_version *(function)*, 59
    copads.dose_executor.simulate *(function)*,

59
copads.dose_executor.write_parameters *(function)*, 59
copads.dose_parameters *(module)*, 61–63
copads.dose_world *(module)*, 64–68
copads.dose_world.World *(class)*, 64–68
copads.genetic *(module)*, 69–84
copads.genetic.Chromosome *(class)*, 71–75
copads.genetic.crossover *(function)*, 69
copads.genetic.Organism *(class)*, 75–80
copads.genetic.Population *(class)*, 80–84
copads.genetic.population_constructor *(function)*, 69
copads.genetic.population_simulate *(function)*, 70
copads.graph *(module)*, 85–88
copads.graph.Graph *(class)*, 85–88
copads.hash *(module)*, 89–91
copads.hash.backward_file_hash *(function)*, 89
copads.hash.cfh *(function)*, 90
copads.hash.forward_file_hash *(function)*, 89
copads.hypothesis *(module)*, 92–115
copads.hypothesis.Chisq2Variance *(function)*, 107
copads.hypothesis.Chisq2x2 *(function)*, 111
copads.hypothesis.ChisqFit *(function)*, 109
copads.hypothesis.Chisquare2xKtable *(function)*, 112
copads.hypothesis.ChisquareKx2table *(function)*, 111
copads.hypothesis.ChiSquarePopVar *(function)*, 104
copads.hypothesis.F2CorrelatedObs *(function)*, 105
copads.hypothesis.F2Count *(function)*, 108
copads.hypothesis.FVarianceRatio *(function)*, 105
copads.hypothesis.MedianTestfor2Pop *(function)*, 113
copads.hypothesis.SpearmanCorrelation *(function)*, 113
copads.hypothesis.t1Mean *(function)*, 98
copads.hypothesis.t2Mean2EqualVariance *(function)*, 99
copads.hypothesis.t2Mean2UnequalVariance *(function)*, 100
copads.hypothesis.test *(function)*, 93
copads.hypothesis.tPaired *(function)*, 101
copads.hypothesis.tPearsonCorrelation *(function)*, 102
copads.hypothesis.tRegressionCoefficient *(function)*, 102
copads.hypothesis.tx2testofKcounts *(function)*, 110
copads.hypothesis.Z1Mean1Variance *(function)*, 93
copads.hypothesis.Z1Proportion *(function)*, 96
copads.hypothesis.Z2Count *(function)*, 97
copads.hypothesis.Z2Mean1Variance *(function)*, 94
copads.hypothesis.Z2Mean2Variance *(function)*, 95
copads.hypothesis.Z2PearsonCorrelation *(function)*, 104
copads.hypothesis.Z2Proportion *(function)*, 97
copads.hypothesis.ZCorrProportion *(function)*, 106
copads.hypothesis.ZPearsonCorrelation *(function)*, 103
copads.hypothesis.ZtestLogOddsRatio *(function)*, 114
copads.lc_bf *(module)*, 116–117
copads.lc_bf.accept_predefined *(function)*, 116
copads.lc_bf.backward *(function)*, 116
copads.lc_bf.call_out *(function)*, 116
copads.lc_bf.cbf_end_loop *(function)*, 117
copads.lc_bf.cbf_start_loop *(function)*, 117

copads.lc_bf.decrement *(function)*, 116
copads.lc_bf.forward *(function)*, 116
copads.lc_bf.increment *(function)*, 116
copads.matrix *(module)*, 118–129
copads.matrix.isSparse *(function)*, 120
copads.matrix.isVector *(function)*, 118
copads.matrix.Matrix *(class)*, 123–127
copads.matrix.smDiag *(function)*, 120
copads.matrix.smDot *(function)*, 120
copads.matrix.smDotDot *(function)*, 120
copads.matrix.smIdentity *(function)*, 120
copads.matrix.smTranspose *(function)*, 120
copads.matrix.SparseMatrix *(class)*, 127–129
copads.matrix.vAcos *(function)*, 119
copads.matrix.vAsin *(function)*, 119
copads.matrix.vAtan *(function)*, 119
copads.matrix.vAtan2 *(function)*, 119
copads.matrix.vCos *(function)*, 119
copads.matrix.vCosh *(function)*, 119
copads.matrix.vDot *(function)*, 118
copads.matrix.Vector *(class)*, 120–123
copads.matrix.vExp *(function)*, 118
copads.matrix.vLog *(function)*, 118
copads.matrix.vLog10 *(function)*, 118
copads.matrix.vNorm *(function)*, 118
copads.matrix.vOnes *(function)*, 118
copads.matrix.vPow *(function)*, 119
copads.matrix.vRandom *(function)*, 118
copads.matrix.vSin *(function)*, 119
copads.matrix.vSinh *(function)*, 119
copads.matrix.vSqrt *(function)*, 119
copads.matrix.vSum *(function)*, 118
copads.matrix.vTan *(function)*, 119
copads.matrix.vTanh *(function)*, 119
copads.matrix.vZeros *(function)*, 118
copads.n_bf *(module)*, 130
copads.n_bf.random_op *(function)*, 130
copads.neural *(module)*, 131–137
copads.neural.Brain *(class)*, 134–137
copads.neural.dtf_linear *(function)*, 131
copads.neural.itf_linear *(function)*, 131
copads.neural.Neuron *(class)*, 131–134
copads.neural.tf_linear *(function)*, 131
copads.nrpy *(module)*, 138–149
copads.nrpy.bessi *(function)*, 139
copads.nrpy.bessi0 *(function)*, 139
copads.nrpy.bessi1 *(function)*, 139
copads.nrpy.bessj *(function)*, 139
copads.nrpy.bessj0 *(function)*, 140
copads.nrpy.bessj1 *(function)*, 140
copads.nrpy.bessk *(function)*, 140
copads.nrpy.bessk0 *(function)*, 141
copads.nrpy.bessk1 *(function)*, 141
copads.nrpy.bessy *(function)*, 141
copads.nrpy.bessy0 *(function)*, 141
copads.nrpy.bessy1 *(function)*, 142
copads.nrpy.beta *(function)*, 142
copads.nrpy.betacf *(function)*, 142
copads.nrpy.betai *(function)*, 143
copads.nrpy.bico *(function)*, 143
copads.nrpy.cdf_binomial *(function)*, 148
copads.nrpy.cdf_poisson *(function)*, 148
copads.nrpy.chebev *(function)*, 143
copads.nrpy.erf *(function)*, 144
copads.nrpy.erfc *(function)*, 144
copads.nrpy.erfcc *(function)*, 144
copads.nrpy.expdev *(function)*, 145
copads.nrpy.factln *(function)*, 145
copads.nrpy.gammln *(function)*, 145
copads.nrpy.gammp *(function)*, 146
copads.nrpy.gammq *(function)*, 146
copads.nrpy.gcf *(function)*, 147
copads.nrpy.gser *(function)*, 147
copads.nrpy.mdian1 *(function)*, 147
copads.nrpy.moment *(function)*, 147
copads.nrpy.qgaus *(function)*, 148
copads.objectdistance *(module)*, 150–168
copads.objectdistance.Anderberg *(function)*, 153
copads.objectdistance.binarize *(function)*, 150
copads.objectdistance.Bray_Curtis *(function)*, 166
copads.objectdistance.Buser *(function)*, 159
copads.objectdistance.Canberra *(function)*,

166
copads.objectdistance.compare *(function)*, 150
copads.objectdistance.Cosine *(function)*, 167
copads.objectdistance.Dennis *(function)*, 162
copads.objectdistance.Dice *(function)*, 152
copads.objectdistance.Euclidean *(function)*, 164
copads.objectdistance.Forbes *(function)*, 155
copads.objectdistance.Fossum *(function)*, 159
copads.objectdistance.Gower_Legendre *(function)*, 163
copads.objectdistance.Hamann *(function)*, 155
copads.objectdistance.Hamming *(function)*, 164
copads.objectdistance.Jaccard *(function)*, 150
copads.objectdistance.Kulczynski *(function)*, 154
copads.objectdistance.Kulczynski2 *(function)*, 154
copads.objectdistance.Manhattan *(function)*, 165
copads.objectdistance.Matching *(function)*, 151
copads.objectdistance.Mcconnaughey *(function)*, 161
copads.objectdistance.Minkowski *(function)*, 165
copads.objectdistance.Ochiai *(function)*, 152
copads.objectdistance.Ochiai2 *(function)*, 153
copads.objectdistance.Pearson *(function)*, 162
copads.objectdistance.Roger_Tanimoto *(function)*, 157
copads.objectdistance.Russel_Rao *(function)*, 156
copads.objectdistance.Simpson *(function)*, 156
copads.objectdistance.Sokal_Michener *(function)*, 151
copads.objectdistance.Sokal_Sneath *(function)*, 157
copads.objectdistance.Sokal_Sneath2 *(function)*, 158
copads.objectdistance.Sokal_Sneath3 *(function)*, 158
copads.objectdistance.Stiles *(function)*, 161
copads.objectdistance.Tanimoto *(function)*, 167
copads.objectdistance.Tulloss *(function)*, 163
copads.objectdistance.YuleQ *(function)*, 160
copads.objectdistance.YuleY *(function)*, 160
copads.operations *(module)*, 169–173
copads.operations.asdict *(function)*, 169
copads.operations.Boolean *(class)*, 172–173
copads.operations.combination *(function)*, 171
copads.operations.contents *(function)*, 169
copads.operations.count *(function)*, 169
copads.operations.factorial *(function)*, 171
copads.operations.fcmp *(function)*, 170
copads.operations.fibonacci *(function)*, 171
copads.operations.indexesof *(function)*, 170
copads.operations.intd *(function)*, 170
copads.operations.itemindex *(function)*, 169
copads.operations.items *(function)*, 169
copads.operations.Modulus2 *(class)*, 172
copads.operations.permutation *(function)*, 171
copads.operations.product *(function)*, 172
copads.operations.safe_exp *(function)*, 171
copads.operations.safe_log *(function)*, 170

copads.operations.safe_log2 *(function)*, 170
copads.operations.sample *(function)*, 171
copads.operations.sample_wr *(function)*, 171
copads.operations.summation *(function)*, 172
copads.operations.take *(function)*, 170
copads.operations.take_byfn *(function)*, 170
copads.prioritydictionary *(module)*, 174–175
copads.prioritydictionary.PriorityDictionary *(class)*, 174–175
copads.ragaraja *(module)*, 176–188
copads.ragaraja.accumulations *(function)*, 178
copads.ragaraja.activate_version *(function)*, 187
copads.ragaraja.flipping *(function)*, 184
copads.ragaraja.input_IO *(function)*, 184
copads.ragaraja.instruction_padding *(function)*, 176
copads.ragaraja.jump_identifier *(function)*, 186
copads.ragaraja.logic *(function)*, 183
copads.ragaraja.loop_end *(function)*, 177
copads.ragaraja.loop_start *(function)*, 176
copads.ragaraja.mathematics *(function)*, 181
copads.ragaraja.nBF_random_op *(function)*, 179
copads.ragaraja.nBF_to_Ragaraja *(function)*, 187
copads.ragaraja.not_used *(function)*, 187
copads.ragaraja.output_IO *(function)*, 182
copads.ragaraja.register_IO *(function)*, 185
copads.ragaraja.set_tape_value *(function)*, 180
copads.ragaraja.source_filter *(function)*, 187
copads.ragaraja.source_move *(function)*, 180
copads.ragaraja.tape_manipulate *(function)*, 185
copads.ragaraja.tape_move *(function)*, 177
copads.ragaraja.tape_size *(function)*, 179
copads.register_machine *(module)*, 189–190
copads.register_machine.interpret *(function)*, 189
copads.ring *(module)*, 191–192
copads.ring.RingList *(class)*, 191–192
copads.samplestatistics *(module)*, 193–197
copads.samplestatistics.SampleDistribution *(class)*, 195
copads.samplestatistics.SingleSample *(class)*, 193–195
copads.samplestatistics.TwoSample *(class)*, 195–197
copads.set *(module)*, 198–199
copads.set.Set *(class)*, 198–199
copads.statisticsdistribution *(module)*, 200–267
copads.statisticsdistribution.AntiLogNormalDistribut *(function)*, 202
copads.statisticsdistribution.BernoulliDistribution *(class)*, 239–241
copads.statisticsdistribution.BetaDistribution *(class)*, 206–208
copads.statisticsdistribution.BilateralExponentialDist *(function)*, 202
copads.statisticsdistribution.BinomialDistribution *(class)*, 208–210
copads.statisticsdistribution.BradfordDistribution *(class)*, 241–243
copads.statisticsdistribution.BurrDistribution *(class)*, 243–245
copads.statisticsdistribution.CauchyDistribution *(class)*, 210–212
copads.statisticsdistribution.ChiDistribution *(class)*, 245
copads.statisticsdistribution.ChiSquareDistribution *(class)*, 220
copads.statisticsdistribution.CobbDouglasDistributio *(function)*, 202
copads.statisticsdistribution.CosineDistribution *(class)*, 212–214

copads.statisticsdistribution.Distribution *(class)*, 204–206
copads.statisticsdistribution.DoubleExponentialDistribution *(function)*, 202
copads.statisticsdistribution.DoubleGammaDistribution *(class)*, 245–246
copads.statisticsdistribution.DoubleWeibullDistribution *(class)*, 246–247
copads.statisticsdistribution.ErlangDistribution *(function)*, 202
copads.statisticsdistribution.ExponentialDistribution *(class)*, 214–216
copads.statisticsdistribution.ExtremeLBDistribution *(class)*, 247
copads.statisticsdistribution.FDistribution *(class)*, 216–218
copads.statisticsdistribution.FisherTippettDistribution *(function)*, 203
copads.statisticsdistribution.FiskDistribution *(class)*, 247–248
copads.statisticsdistribution.FoldedNormalDistribution *(class)*, 248
copads.statisticsdistribution.FrechetDistribution *(function)*, 202
copads.statisticsdistribution.FurryDistribution *(function)*, 202
copads.statisticsdistribution.GammaDistribution *(class)*, 218–220
copads.statisticsdistribution.GenLogisticDistribution *(class)*, 248–249
copads.statisticsdistribution.GeometricDistribution *(class)*, 220–222
copads.statisticsdistribution.GompertzDistribution *(function)*, 203
copads.statisticsdistribution.GumbelDistribution *(class)*, 249–251
copads.statisticsdistribution.HalfNormalDistribution *(class)*, 251–252
copads.statisticsdistribution.HyperbolicSecantDistribution *(class)*, 252–254
copads.statisticsdistribution.HypergeometricDistribution *(class)*, 222–224
copads.statisticsdistribution.LaplaceDistribution *(class)*, 254–255
copads.statisticsdistribution.LogarithmicDistribution *(class)*, 224–225
copads.statisticsdistribution.LogisticDistribution *(class)*, 255
copads.statisticsdistribution.LogLogisticDistribution *(function)*, 203
copads.statisticsdistribution.LogNormalDistribution *(class)*, 255–256
copads.statisticsdistribution.LogWeibullDistribution *(function)*, 203
copads.statisticsdistribution.LorentzDistribution *(function)*, 203
copads.statisticsdistribution.MaxwellDistribution *(class)*, 256–257
copads.statisticsdistribution.NakagamiDistribution *(class)*, 257–258
copads.statisticsdistribution.NegativeBinomialDistrib *(class)*, 258–259
copads.statisticsdistribution.NegativeExponentialDist *(function)*, 203
copads.statisticsdistribution.NormalDistribution *(class)*, 225–228
copads.statisticsdistribution.ParetoDistribution *(class)*, 259–261
copads.statisticsdistribution.PascalDistribution *(class)*, 261–263
copads.statisticsdistribution.PoissonDistribution *(class)*, 228–229
copads.statisticsdistribution.PolyaDistribution *(function)*, 203
copads.statisticsdistribution.PowerFunctionDistributi *(class)*, 263–264
copads.statisticsdistribution.RademacherDistribution *(class)*, 264–266
copads.statisticsdistribution.RayleighDistribution *(class)*, 266–267
copads.statisticsdistribution.ReciprocalDistribution *(class)*, 267
copads.statisticsdistribution.RectangularDistribution *(function)*, 203
copads.statisticsdistribution.SechSquaredDistribution *(function)*, 204
copads.statisticsdistribution.SemicircularDistribution *(class)*, 229–231

copads.statisticsdistribution.TDistribution *(class)*, 231–233
copads.statisticsdistribution.TriangularDistribution *(class)*, 233–236
copads.statisticsdistribution.UniformDistribution *(class)*, 236–238
copads.statisticsdistribution.WaldDistribution *(function)*, 204
copads.statisticsdistribution.WeiBullDistribution *(class)*, 238–239
copads.tree *(module)*, 268–275
copads.tree.OrderedBinaryTree *(class)*, 269
copads.tree.RBDict *(class)*, 273–275
copads.tree.RBList *(class)*, 271–273
copads.tree.RBTree *(class)*, 269–271
copads.tree.RBTreeIter *(class)*, 268–269
copads.treenodes *(module)*, 276–277
copads.treenodes.BinaryNode *(class)*, 277
copads.treenodes.FourNode *(class)*, 277
copads.treenodes.RBNode *(class)*, 276–277
copads.treenodes.ThreeNode *(class)*, 277