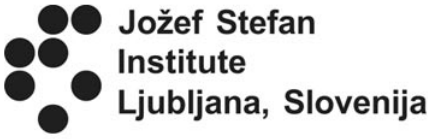
IJS delovno poročilo
IJS DP-13758

2022

Marko Bohanec

DEXiPy: A Package for Using DEXi Models in Python



CONTENTS:

CHAPTER

ONE

DEXIPY: BASIC CONCEPTS

DEXiPy is a software package for using DEXi models in Python. The main function is evaluating decision alternatives using a model previously developed by DEXi software.

1.1 DEXi Models

DEXi models are hierarchical qualitative rule-based multi-criteria decision models developed according to the method DEX (Decision EXpert), using the program DEXi.

In general, a DEXi model consists of a hierarchy of qualitative (symbolic linguistic, discrete) variables, called *attributes*. Each attribute represents some observable property (such as Price or Performance) of decision alternatives under study. An attribute can take values from a set of words, such as {"low", "medium", "high"} or {"unacc", "acc", "good", "exc"}. Value sets are usually small (up to five elements) and preferentially ordered from "bad" to "good" values.

The *hierarchy* of attributes represents a decomposition of a decision problem into sub-problems, so that higher-level attributes depend on lower-level ones. Consequently, terminal nodes represent inputs, and non-terminal attributes represent outputs of the model. Among these, the most important are one or more root attributes, which represent the final evaluation(s) of the alternatives.

The *evaluation* of decision alternatives takes place as a hierarchical aggregation of values from model inputs to outputs. Evaluation is governed by *decision rules*, defined for each non-terminal attribute by the creator of the model (usually referred to as a "decision maker").

1.2 Terminological remarks

DEX DEX (Decision EXpert) refers to a general multi-attribute decision modeling method, characterized by using qualitative attribute hierarchies and decision tables. For further information, see (Trdin, Bohanec, 2018) and (Bohanec, 2022).

DEXi DEXi ("DEX for instruction") refers to DEXi software. DEXi implements a subset of DEX, for instance, it is restricted to set-based evaluation methods. DEXi supports the creation and editing of *DEXi Models*, which are saved on `.dxi` files and subsequently read by DEXiPy for processing in Python. For further information on DEXi, see http://kt.ijs.si/MarkoBohanec/dexi.html and (Bohanec, 2020).

DEXiPy DEXiPy is this package. It is capable of reading and processing DEXi models with some extensions towards the full DEX (for example, using value distributions).

1.3 DEXiPy Functionality

Models developed using the DEXi software are stored in XML-formatted `.dxi` files. In order to use DEXi models in Python, DEXiPy supports the following tasks:

1. Reading DEXi models from `.dxi` files, using *dexipy.dexi.read_dexi()*.
2. Defining data (both input and output) about considered decision alternatives, using *dexipy.dexi.alternative()*.
3. Evaluating decision alternatives, using *dexipy.dexi.evaluate()*.

By default, evaluation is based on sets, which is a standard evaluation procedure of DEXi. DEXiPy extends this by supporting:

- evaluations using probabilistic and fuzzy value distributions;
- "pruned" evaluation, when the evaluation starts from selected non-terminal attribute(s) upwards.

1.4 Limitations

DEXiPy has been designed to facilitate *using* DEXi models in Python, produced externally by the DEXi software. DEXiPy does not provide any explicit means for creating and/or editing DEXi models.

1.5 References

1. *Decision EXpert*. Wikipedia, https://en.wikipedia.org/wiki/Decision_EXpert.
2. *DEXi: A Program for Multi-Attribute Decision Making*. http://kt.ijs.si/MarkoBohanec/dexi.html.
3. Bohanec, M.: *DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04*. IJS Report DP-13100, Jožef Stefan Institute, Ljubljana, 2020. http://kt.ijs.si/MarkoBohanec/pub/DEXiManual504.pdf.
4. Trdin, N., Bohanec, M.: Extending the multi-criteria decision making method DEX with numeric attributes, value distributions and relational models. *Central European Journal of Operations Research*, 1-24, 2018. https://doi.org/10.1007/s10100-017-0468-9.
5. Bohanec, M.: DEX (Decision EXpert): A Qualitative Hierarchical Multi-criteria Method. In: Kulkarni, A.J. (ed.): Multiple Criteria Decision Making: Techniques, Analysis and Applications. Singapore: Springer, 39-78, 2022. https://doi.org/10.1007/978-981-16-7414-3_3.

CHAPTER

TWO

DEXIPY CLASSES AND DATA TYPES

2.1 Classes

DEXiModel

`dexipy.dexi.DEXiModel` is a top-level DEXiPy class that represents a whole DEXi model. The hiearchy of model attributes starts at `DEXiModel.root`. A `DEXiModel` object generally contains its `name` and `description` strings, lists of attribute IDs and a list of alternatives.

DEXiAttribute

`dexipy.dexi.DEXiAttribute` is a variable representing some measurable property of decision alternatives. Attributes are structired hierarchically, therefore each attribute contains the list `DEXiAttribute.inputs` of its input attributes, i.e., immediate descendants in the hierarchy. When fully defined, each attribute has an associated `DEXiAttribute.scale`, which is an object of the `DEXiScale` class. Furthermore, each aggregate attribute (non-terminal node in the hierarchy) is normally associated with a `DEXiFunction` object `DEXiAttribute.funct`, which governs the aggregation of input values to the value of that attribute. Also, an attribute has a `name`, optional `description` and an ID string that is unique in the model context.

DEXiScale

The `dexipy.dexi.DEXiScale` class defines the set of values that can be assigned to the corresponding attribute. A scale might, but need not be preferentially ordered.

While `DEXiScale` is a base class, there are two derived scale types that are actually used in DEXi models:

DEXiContinousScale

This scale can be associated with basic attributes, i.e., terminal nodes of the model. Only `float` values can be assigned to such attributes.

DEXiDiscreteScale

A discrete scale defines an ordered set of discrete values that can be assigned to the corresponding attribute, for instance `{"low", "medium", "high"}`. This scale type can be associated with both basic and aggregate attributes.

DEXiFunction

`dexipy.dexi.DEXiFunction` is a base class for functions that aggregate or discretize the values of some attribute's inputs to the value of that attribute.

Two types of functions are used in DEXiPy:

DEXiDiscretizeFunction

This function type is used to map numeric values of a continuous basic attribute to discrete values of a single discrete parent attribute.

DEXiTabularFunction

This is the main DEXi aggregation function type that aggregates multiple discrete inputs to a single discrete parent attribute. Essentially, a `DEXiTabularFunction` consists of a lookup table that defines the output value for all combinations of input values. Each table entry is referred to as an *elementary decision rule*.

2.2 DEXi values

DEXi values are used throughout DEXi models. They provide input values and carry results of evaluations of decision alternatives. DEXi values are also used in definitions of `dexipy.dexi.DEXiFunction` and are returned by `dexipy.dexi.DEXiFunction.evaluate()` when evaluating some function for a given set of arguments.

In DEXi, values are always bound to the context provided by a `dexipy.dexi.DEXiScale`. Since each fully defined `dexipy.dexi.DEXiAttribute` is associated with some scale, we can generalize the scale context to attributes and speak about "assigning some value to an attribute".

In DEXiPy, DEXi values are *not* represented by classes, but rather by different data types that are interpreted in the context of given attributes and their scales. *dexipy.types* defines two type hints that define admissible value types:

```
DexiValue = Union[None, str, float, Tuple[int], Set[int],
            List[float], Dict[int, float]]

DexiScaleValue = Union[None, str, float, Tuple[Union[int, str]], Set[Union[int, str]],
                 List[float], Dict[Union[int, str], float]]
```

`DEXiValue` defines data types that prevalently use numeric data and do not require a scale in order to be interpreted. `DEXiScaleValue` additionally includes `str` type in tuples, sets and dictionaries to facilitate refering to discrete scale values by their names rather than indices.

For any scale type, the admissible DEXi values are `None`, `""` or any string starting with `"undef"`, indicating an unavailable or unknown value. All evaluations involving `None` result in `None`.

`dexipy.dexi.DEXiContinousScale` allows only floating-point numbers.

`dexipy.dexi.DEXiDiscreteScale` is the main scale type used throughout DEXi models and supports a wider range of value types.

The "normal" and most common discrete value is a "single qualitative value". For illustration, let us use the scale composed of four qualitative values: `{"unacc", "acc", "good", "exc"}`. Then, "a single qualitative value" denotes one of these words. Internally in DEXiPy, such values are not represented by strings, but rather by ordinal numbers, so that `ord("unacc") == 0`, `ord("acc") == 1`, etc. Some DEXiPy functions can convert between the two representations, for example `dexipy.dexi.DEXiModel.evaluate()` and *dexipy.dexi.alternative()*.

In order to cope with missing, incomplete or uncertain data, DEX extends the concept of single values to value *sets* and *distributions*. In DEXiPy, wherever it is possible to use a single qualitative value, it is also possible to use a value set or distribution. This includes all data representing *alternatives* and all functions that return qualitative values. Also note that while sets are fully implemented in the current DEXi software, distributions are not and are thus considered extensions towards the full DEX method.

A *DEXi value set* is a subset of the full range of a `dexipy.dexi.DEXiDiscreteScale` values. For the above scale example, the full range of ordinal values is `(0, 1, 2, 3)`, and some possible subsets are `{1}`, `(1, 3)`, and `(1, 2, 3)`. Both tuples and sets can be used, however tuples are internally converted to sets. The string `"*"` can be used to indicate a full set of values of the corresponding discrete scale.

When a value set is used in a scale context, it can be specified also in terms of value names, for instance `{"acc"}`, `{"acc", "exc"}`, and `("acc", "good", "exc")`. Mixed format is acceptable, too: `("acc", 2, "exc")`. This format can be used in dictionaries that contain data about alternatives, and is also produced by `dexipy.dexi.DEXiModel.alt_text()`.

A *DEXi value distribution* associates each `DexiDiscreteScale` value with some number, generally denoted p and normally expected to ben the [0,1] interval. Depending on the context and used evaluation method (see *Evaluation of Alternatives*), p can be interpreted as a *probability* or *fuzzy set membership*. In DEXiPy, value distributions are internally represented by a list of floating-point numbers. For example, `[0.5, 0, 0.2, 0.3]` represents a value distribution over the above scale example, assigning

- $p = 0.5$ to `"unacc"`,
- $p = 0.0$ to `"acc"`,
- $p = 0.2$ to `"good"` and
- $p = 0.3$ to `"exc"`.

An alternative and possibly more readable representation of distributions uses the dictionary format, for instance:

- using value indices: `{0: 0.5, 2: 0.2, 2: 0.3}`
- using value names: `{"unacc": 0.5, "good": 0.2, "exc": 0.3}`
- mixed: `{"unacc": 0.5, 2: 0.2, "exc": 0.3}`

2.3 Alternatives

Alternatives (more specifically, *decision alternatives*) are objects evaluated by DEXi models. In DEXiPy, an alternative is represented by a dictionary whose:

- *keys* denote attributes, and
- *values* contain *DEXi values* of the corresponding attributes.

In general, *keys* can be either string attribute IDs or integer attribute indices. In addition, there are two special keys `"name"` and `"description"`, providing a name and description string of the alternative; both are optional. Attribute keys can refer to any attribute in the model. However, an alternative that is to be evaluated by *`dexipy.dexi.evaluate()`* is expected to contain key/value pairs of all basic attributes of the model. After this alternative has been evaluated, calculated values that correspond to aggregate attributes are added to the dictionary, possibly overwriting previous values.

`dexipy.types` defines type hints for admissible represenations of alternatives:

```
DexiAlternative = Dict[Union[str, int], DexiValue]
DexiAlternatives = List[DexiAlternative]
DexiAltData = Union[DexiAlternative, DexiAlternatives]
```

`DexiAlternative` represents a single alternative. `DexiAlternatives` represents a list of alternatives. `DexiAltData` denotes a union of both representations.

Example of an evaluated alternative:

```
{'name': 'MyCar1a',
'CAR': {0, 3}, 'PRICE': {0, 2}, 'BUY.PRICE': 2, 'MAINT.PRICE': {0, 1, 2},
'TECH.CHAR.': 2, 'COMFORT': 2, '#PERS': 2, '#DOORS': 2, 'LUGGAGE': 2,
'SAFETY': 1}
```

Representation using attribute indices is also possible:

```
{'name': 'MyCar1a',
1: {0, 3}, 2: {0, 2}, 3: 2, 4: {0, 1, 2}, 5: 2, 6: 2, 7: 2, 8: 2, 9: 2, 10: 1}
```

Multiple alternatives can be combined together in a list. The method `dexipy.dexi.DEXiFunction.evaluate()`, which evaluates alternatives, accepts `DexiAltData`, i.e., both a single alternative or a list of alternatives.

2.4 Evaluation of Alternatives

In DEXiPy, decision alternatives can be evaluated using the method *dexipy.dexi.DexiModel.evaluate()* or function *dexipy.dexi.evaluate()*. They accept almost the same arguments and are actually just suitable aliases for calling the main implementation at *dexipy.eval.evaluate()*.

Essentially, *evaluation of alternatives* in DEX is a bottom-up aggregation method: starting with basic attributes (or "pruned" aggregate attributes), values of each alternative are gradually aggregated towards the root attribute. The aggregation at each individual *dexipy.dexi.DexiAttribute* is governed by the corresponding *dexipy.dexi.DexiFunction*. When alternative values are sets or distributions (see *DEXi values*), then `evaluate()` methods try all possible combinations of values of the input attributes.

Four aggregation methods are supported: "set", "prob", "fuzzy" and "fuzzynorm".

The "set" method interprets DEXi values as sets. The output value assigned to some `attribute` is composed of the union of all `attribute.funct` evaluations for all possible combinations of values of `attribute.inputs`.

The remaining three methods interpret DEXi values as value distributions. They follow the same algorithm, but use different methods (see *dexipy.eval.DexiEvalParameters*) in three algorithmic steps: normalization, and conjunctive and disjunctive aggregation of values. All values distributions involved in calculations are normalized by the method `dexipy.eval.DexiEvalParameters.norm`. All combinations of `attribute.inputs` values are individually evaluated by the corresponding tabular function `attribute.funct`. The value p of each set of `attribute.funct` arguments is determined by the conjunctive aggregation function `dexipy.eval.DexiEvalParameters.and_op` over p's of individual arguments. Finally, the p of some output value `val` is determined by the disjunctive aggregation function `dexipy.eval.DexiEvalParameters.or_op`, applied on the p's of all partial evaluations that map to `val`.

For mathematical background and more details about aggregation in DEX, please see (Trdin, Bohanec, 2018).

CHAPTER

THREE

EXAMPLES

3.1 A typical DEXiPy workflow

This example uses a simple DEXi model for evaluating cars, which is distributed together with the DEXi software (including DEXiPy) and is used throughout DEX literature to illustrate the methodological approach (https://en.wikipedia.org/wiki/Decision_EXpert).

First, this model is loaded and printed as follows:

```
>>> import dexipy.dexi as dxi
>>> car = dxi.read_dexi("data/car.dxi")
>>> print(car)
DEXi Model: CAR_MODEL
Description: Car demo
index id          structure        scale                     funct
    0 CAR_MODEL   CAR_MODEL
    1 CAR         +- CAR           unacc; acc; good; exc (+) 12 3x4
    2 PRICE       |- PRICE         high; medium; low (+)     9 3x3
    3 BUY.PRICE   | |- BUY.PRICE   high; medium; low (+)
    4 MAINT.PRICE | +- MAINT.PRICE high; medium; low (+)
    5 TECH.CHAR.  +- TECH.CHAR.    bad; acc; good; exc (+)   9 3x3
    6 COMFORT     |- COMFORT       small; medium; high (+)   36 3x4x3
    7 #PERS       | |- #PERS       to_2; 3-4; more (+)
    8 #DOORS      | |- #DOORS      2; 3; 4; more (+)
    9 LUGGAGE     | +- LUGGAGE     small; medium; big (+)
   10 SAFETY      +- SAFETY        small; medium; high (+)
```

Rows in the printout correspond to individual attributes. The columns display:

- `index`: Indices of attributes.
- `id`: Unique attribute IDs, generated by DEXiPy from original DEXi names, in order to assure unambiguous referencing of attributes.
- `structure`: The hierarchical structure of attributes, named as in the original DEXi model.
- `scale`: Value scales associated with each attribute. The symbol `(+)` indicates that the corresponding scale is ordered preferentially in an increasing order.
- `funct`: Information about the size (number of rules) and dimensions of the corresponding decision tables.

Looking at the structure of attributes, please notice that the attribute at index 0 is virtual and does not appear in the original DEXi model. In DEXiPy it allows using models that have multiple root attributes; these models appear as subtrees of the virtual root.

The "real" root of `CAR_MODEL` is actually `CAR` at index 1. It depends on two lower-level attributes, `PRICE` and `TECH.CHAR.` These are decomposed further. Overall, the model consists of:

1. six input (*basic*) attributes: `BUY.PRICE`, `MAINT.PRICE`, `#PERS`, `#DOORS`, `LUGGAGE` and `SAFETY`, and

2. four output (*aggregate*) attributes: `CAR`, `PRICE`, `TECH.CHAR.` and `COMFORT`.

Among the latter, `CAR` is the most important and represents the overall evaluation of cars.

The next step usually consists of defining a decision alternative or a list of alternatives (i.e., cars in this case). The Car model already comes with a list of two cars, accessible using *dexipy.dexi.DexiModel.alternatives*. Each alternative is represented as a dictionary:

```
>>> car.alternatives[0]
{'name': 'Car1', 'CAR': 3, 'PRICE': 2, 'BUY.PRICE': 1, 'MAINT.PRICE': 2,
'TECH.CHAR.': 3, 'COMFORT': 2, '#PERS': 2, '#DOORS': 2, 'LUGGAGE': 2,
'SAFETY': 2}
```

Alternatives can be printed in a tabular form:

```
>>> print(car.alt_table())
alternative Car1 Car2
CAR            3    2
PRICE          2    1
BUY.PRICE      1    1
MAINT.PRICE    2    1
TECH.CHAR.     3    2
COMFORT        2    2
#PERS          2    2
#DOORS         2    2
LUGGAGE        2    2
SAFETY         2    1
```

In this printout, attribute values are shown using the internal DEXiPy representation, i.e., using ordinal value numbers. A more readable output can be obtained by *dexipy.dexi.DexiModel.alt_text()*:

```
>>> print(car.alt_text())
alternative   Car1   Car2
CAR            exc   good
PRICE          low medium
BUY.PRICE   medium medium
MAINT.PRICE    low medium
TECH.CHAR.     exc   good
COMFORT       high   high
#PERS         more   more
#DOORS           4      4
LUGGAGE        big    big
SAFETY        high medium
```

This data can be edited using common Python list and dictionary functions.

Additionally, DEXiPy provides the method *dexipy.dexi.DexiModel.alternative()* for defining a single decision alternative, for example:

```
>>> alt = car.alternative("MyCar1", values =
          {'BUY.PRICE': "low", 'MAINT.PRICE': 2, '#PERS': "more", '#DOORS': "4",
           'LUGGAGE': 2, 'SAFETY': "medium"})
>>> print(car.alt_table(alt))
alternative MyCar1
CAR            None
PRICE          None
BUY.PRICE       low
MAINT.PRICE       2
TECH.CHAR.     None
COMFORT        None
```

(continues on next page)

(continued from previous page)

```
#PERS          more
#DOORS            4
LUGGAGE           2
SAFETY       medium
```

Finally, alternatives can be evaluated using *dexipy.dexi.DexiModel.evaluate()*:

```
>>> eval_alt = car.evaluate(alt)
>>> print(car.alt_text(eval_alt))
alternative MyCar1
CAR             exc
PRICE           low
BUY.PRICE       low
MAINT.PRICE     low
TECH.CHAR.     good
COMFORT        high
#PERS          more
#DOORS            4
LUGGAGE         big
SAFETY       medium
```

3.2 Examples of using value sets and distributions

For example, let us consider a car for which we have no evidence about its possible maintenance costs. For the value of `MAINT.PRICE`, we may use `"*"` to denote the full range of the corresponding attribute values (in this case equivalent to `{0, 1, 2}` or `('high', 'medium', 'low')`. Notice how the evaluation method considers all the possible values of `MAINT.PRICE` and propagates them upwards the model structure.

```
>>> alt = car.alternative("MyCar1a", values =
          {'BUY.PRICE': "low", 'MAINT.PRICE': "*", '#PERS': "more", '#DOORS': "4",
           'LUGGAGE': 2, 'SAFETY': "medium"})
>>> eval_alt = car.evaluate(alt)
>>> print(car.alt_text(eval_alt))
alternative                    MyCar1a
CAR                   ('unacc', 'exc')
PRICE                  ('high', 'low')
BUY.PRICE                          low
MAINT.PRICE ('high', 'medium', 'low')
TECH.CHAR.                        good
COMFORT                           high
#PERS                             more
#DOORS                               4
LUGGAGE                           big
SAFETY                         medium
```

The above result is not really useful, as the car turns out to be `('unacc', 'exc')`, that is, either `"unacc"` or `"exc"`, depending on maintenance costs. Thus, let us try using value distribution for `MAINT.PRICE`, telling DEXiPy that high maintenance costs are somewhat unexpected (with probability $p = 0.1$) and that medium costs ($p = 0.6$) are more likely than low ($p = 0.3$). The evaluation method `"prob"` gives the following results:

```
>>> alt = car.alternative("MyCar1b", values =
          {'BUY.PRICE': "low", 'MAINT.PRICE': {"low": 0.3, "medium": 0.6, "high": 0.1}
↪,
```

(continues on next page)

(continued from previous page)

```
          '#PERS': "more", '#DOORS': "4", 'LUGGAGE': 2, 'SAFETY': "medium"})
>>> eval_alt = car.evaluate(alt, method = "prob")
>>> print(car.alt_text(eval_alt, decimals = 2))
alternative                                  MyCar1b
CAR                        {'unacc': 0.1, 'exc': 0.9}
PRICE                       {'high': 0.1, 'low': 0.9}
BUY.PRICE                                        low
MAINT.PRICE {'high': 0.1, 'medium': 0.6, 'low': 0.3}
TECH.CHAR.                                      good
COMFORT                                         high
#PERS                                           more
#DOORS                                             4
LUGGAGE                                          big
SAFETY                                        medium
```

In this case, the final evaluation of `CAR` is `{'unacc': 0.1, 'exc': 0.9}`, that is, it is much more likely that `MyCar1b` is `"exc"` than `"unacc"`.

CHAPTER

FOUR

DEXIPY

4.1 dexipy package

4.1.1 dexipy.types module

Module `dexipy.types` defines type aliases and enumeration classess that are used throughout DEXiPy.

`class dexipy.types.BoundAssoc(value)`
Bases: `enum.Enum`

Enumeration associated with bounds that discretize continuous scales.

Parameters `Enum` (*int*) – indicates the interval to which some corresponding bound b belongs.

`down = -1`
Indicates that b belongs to the interval $<= b$.

`up = 1`
Indicates that b belongs to the interval $b >=$.

`dexipy.types.CallableNorm`
Callable normalization functions, that accept and return a list of floats.

Type Type alias

alias of `Callable`[[...], `List`[`float`]]

`dexipy.types.CallableOperator`
Callable `and_op` and `or_op` operator functions that accept a list of floats and return a single float.

Type Type alias

alias of `Callable`[[`List`[`float`]], `float`]

`dexipy.types.DexiAltData`
General representation of alternatives: `Union[DexiAlternative, DexiAlternatives]`.

Type Type alias

alias of `Union`[`Dict`[`Union`[`str`, `int`], `Union`[`None`, `str`, `float`, `Tuple`[`int`], `Set`[`int`], `List`[`float`], `Dict`[`int`, `float`]]], `List`[`Dict`[`Union`[`str`, `int`], `Union`[`None`, `str`, `float`, `Tuple`[`int`], `Set`[`int`], `List`[`float`], `Dict`[`int`, `float`]]]]]

`dexipy.types.DexiAlternative`
Representation of a single decision alternative: `Dict[Union[str, int], DexiValue]`.

Type Type alias

alias of `Dict`[`Union`[`str`, `int`], `Union`[`None`, `str`, `float`, `Tuple`[`int`], `Set`[`int`], `List`[`float`], `Dict`[`int`, `float`]]]

dexipy.types.DexiAlternatives
Representation of multiple decision alternatives as List[DexiAlternative].

Type Type alias

alias of List[Dict[Union[str, int], Union[None, str, float, Tuple[int], Set[int], List[float], Dict[int, float]]]]

class dexipy.types.DexiEvalMethod(*value*)
Bases: enum.Enum

Enumeration od DEXiPy evaluation methods.

Parameters Enum (*int*) – Evaluation method.

fuzzy = 3
Evaluation interpreting DEXi values as fuzzy set memberships (possibility distributions).

fuzzynorm = 4
Similar to fuzzy, but enforcing fuzzy normalization (the maximum distribution element must equal to 1.0).

prob = 2
Evaluation interpreting DEXi values as probability distributions.

set = 1
Evaluation using sets (default).

class dexipy.types.DexiOrder(*value*)
Bases: enum.Enum

Enumeration of DexiScale preferential order.

Parameters Enum (*int*) – Preferential order.

ascending = 1
Scale values are ordered from "bad" to "good" ones (default).

descending = -1
Scale values are ordered from "good" to "bad" ones.

none = 0
Scale values are not ordered by preference.

class dexipy.types.DexiQuality(*value*)
Bases: enum.Enum

Enumeration of DexiScale scale value quality classes.

Parameters Enum (*int*) – Scale value quality class.

bad = -1
A "bad" value class, indicating an undesired value.

good = 1
A "good" value class, indicating a highly desired, ideal, value.

none = 0
A "neutral" value class, neither particularly "good" nor "bad".

dexipy.types.DexiScaleValue
Admissible DEXi values that can be interpreted in a given DexiScale context.

Type Type alias

alias of Union[None, str, float, Tuple[Union[int, str]], Set[Union[int, str]], List[float], Dict[Union[int, str], float]]

dexipy.types.DexiValue
Admissible DEXi values that can be interpreted without knowing the DexiScale context.

Type Type alias

alias of `Union[None, str, float, Tuple[int], Set[int], List[float], Dict[int, float]]`

class dexipy.types.**DexiValueType**(*value*)

Bases: `enum.Enum`

Enumeration of `DexiValue` data types.

dict = 7

float = 3

int = 2

list = 6

none = 0

set = 4

str = 1

tuple = 5

4.1.2 dexipy.values module

Module `dexipy.values` contains helper classes and functions for handling DEXi values.

class dexipy.values.**DexiValues**(*value*)

Bases: `object`

A wrapper class around a `DexiValue` data element. An object of this class contains a DEXi value `dexipy.values.DexiValues.value`, on which methods operate.

Parameters **value** (*Any*) – A `DexiValue` object stored internally and operated upon by methods.

as_distr()

Convert `self.value` to a value distribution, if possible.

Return type *Optional*[*List*[float]]

as_set(*strict=False*)

Convert `self.value` to a set, if possible. See *dexipy.values.dexi_value_as_set()* for details.

Parameters **strict** (*bool, optional*) – Defines the conversion when `self.value` is a value distribution. When True, only distributions that clearly represent sets, i.e., contain only 0.0 or 1.0 elements, are converted. When False, all elements with non-zero values are considered set members, too. Defaults to False.

Return type *Optional*[*Set*[int]]

check_scale_value()

Check whether or not `self.value` contains valid `DexiScaleValue` data.

Return type bool

check_value()

Check whether or not `self.value` contains valid `DexiValue` data.

Return type bool

reduce()

Reduces the data representation of `self.value`, if possible.

Return type DexiValue

`reduce_value()`
Returns a reduced data representation of `self.value`.

Return type None

`val_str(scale, none=None, reduce=False, decimals=None, use_dict=True)`
Returns a string representation of `self.value`. See *dexipy.dexi.value_text()* for more details.

Parameters

- **scale** (*Any*) – Expected a *dexipy.dexi.DexiScale* object.
- **none** (*Optional[str], optional*) – An optional string that is returned when the value cannot be interpreted. Defaults to None.
- **reduce** (*bool, optional*) – Whether or not the value is reduced (see *reduce_dexi_value()*) prior to processing. Defaults to False.
- **decimals** (*Optional[int], optional*) – The number of decimals used to display float numbers. Defaults to None.
- **use_dict** (*bool, optional*) – Whether or not the dictionary-form is used for displaying value distributions (rather than list-form). Defaults to True.

Return type *Optional*[str]

`value_type()`
Determine the value type of `self.value`.

Return type *Optional*[*dexipy.types.DexiValueType*]

`dexipy.values.check_dexi_scale_value(value)`
Checks the data object and determines whether or not it represents a `DexiScaleValue`.

Operation is similar to *dexipy.values.check_dexi_value()*, except that it additionally allows using name strings to indicate scale values.

Only the data structure is checked. Even if the structure is correct, `value` can still contain elements that may not be correct in the context of some specific `DexiScale`. For instance, the object may contain value names or indices not found in the scale definition.

Parameters **value** (*Any*) – Value object to be checked.

Returns Whether or not the object's structure is valid for representing a `DexiScaleValue`.

Return type bool

`dexipy.values.check_dexi_value(value)`
Checks the data object and determines whether or not it represents a `DexiValue`.

Only the data structure is checked. Even if the structure is correct, `value` can still contain elements that may not be correct in the context of some specific `DexiScale`. For instance, the object may contain value indices not found in the scale definition.

Parameters **value** (*Any*) – Value object to be checked.

Returns Whether or not the object's structure is valid for representing a `DexiValue`.

Return type bool

`dexipy.values.dexi_value_as_distr(value)`
Converts a `DexiValue` object to a value distribution.

Parameters **value** (*DexiValue*) – A DEXi value object.

Returns `value` represented in terms of a value distribution, or None if it cannot be interpreted.

Return type Optional[List[float]]

Examples

```
>>> dexi_value_as_distr(2)
[0.0, 0.0, 1.0]
>>> dexi_value_as_distr({1, 2})
[0.0, 1.0, 1.0]
>>> dexi_value_as_distr({0: 0.5, 2: 1.0})
[0.5, 0.0, 1.0]
>>> dexi_value_as_distr((1, 1, 2, 2))
[0.0, 1.0, 1.0]
```

dexipy.values.**dexi_value_as_set**(*value*, *strict=False*)

Converts a `DexiValue` object to a set.

Parameters

- **value** (*DexiValue*) – A DEXi value object.
- **strict** (*bool, optional*) – Defines the conversion from value distributions. When True, only distributions that clearly represent sets, i.e., contain only 0.0 or 1.0 elements, are converted. When False, all elements with non-zero values are considered set members, too. Defaults to False.

Returns Resulting value set. None if `value` cannot be interpreted as a set.

Return type Optional[Set[int]]

Examples

```
>>> dexi_value_as_set([0, 1, 0.5, 1], strict = True) # returns None
>>> dexi_value_as_set([0, 1, 0.0, 1], strict = True)
{1, 3}
>>> dexi_value_as_set([0, 1, 0.5, 1], strict = False)
{1, 2, 3}
```

dexipy.values.**dexi_value_type**(*value*)

Determines the `DexiValueType` of the argument.

Parameters **value** (*Any*) – Value object to be checked.

Returns Enumeration of the argument's DEXi value type.

Return type Optional[*DexiValueType*]

dexipy.values.**reduce_dexi_value**(*value*)

Reduce a `DexiValue` to a smaller and possibly more comprehensible data representation, if possible.

Typical reductions:

- a tuple to set: `(1, 1, 2, 2) -> {1, 2}`
- a single-element tuple or set to int: `(1,) -> {1} -> 1`
- a distribution to set, if possible: `[1.0, 0.0, 1.0] -> {0, 2}`

Parameters **value** (*DexiValue*) – A DEXi value object.

Returns Reduced representation of `value`, or `value` itself if no reduction is possible.

Return type DexiValue

Examples

```
>>> reduce_dexi_value((1, 1, 2, 2))
{1, 2}
>>> reduce_dexi_value({1})
1
>>> reduce_dexi_value([1.0, 0.0, 1.0])
{0, 2}
>>> reduce_dexi_value([1.0, 0.5, 1.0]) # no reduction
[1.0, 0.5, 1.0]
>>> reduce_dexi_value({1: 1.0})
1
```

dexipy.values.**reduce_set**(*value*)

Reduces a `DexiValue`, represented as a set, to a smaller data representation, if possible

Typical reductions:

- an empty set to None: `{} -> None`
- a single-element tuple or set to int: `(1,) -> {1} -> 1` or `{"low"} -> "low"`

Parameters `value` (*Any*) – A DEXi value object.

Returns Reduced representation of `value`, or `value` itself if no reduction is possible.

Return type DexiValue

Examples

```
>>> reduce_set(set()) # returns None
>>> reduce_set({1})
1
>>> reduce_set({1, 2}) # no reduction
{1, 2}
>>> reduce_set(0.1) # no reduction
0.1
```

4.1.3 dexipy.dexi module

Module `dexipy.dexi` is the main DEXiPy module. It defines all the main classess that constiute a DEXi model and exposes all functions that are usually used to read DEXi models and evaluate decision alternatives.

class dexipy.dexi.**DexiAttribute**(*name=''*, *description=''*, *id=None*, *inputs=None*, *parent=None*, *link=None*, *scale=None*, *funct=None*)

Bases: `object`

`DexiAttribute` is a class representing DEXi attributes.

In a DEXi model, attributes are variables that represent observed properties of decision alternatives. Attributes are structured in a tree, so each attribute may, but need not, have one or more direct descendants (lower-level attributes) in the tree. Attributes without descendants are called *basic* and serve as model inputs. Attributes with one or more descendants are called *aggregate* and represent model outputs. In order to represent attribute hierarchies rather than plain trees, some attributes may be *linked*: two attributes of which one links to another one collectively represent, in a conceptual sense, a single attribute in the hierarchy.

When completely defined, each attribute is associated with a value scale represented by a *dexipy.dexi.DexiScale* object. It is also expected that a *dexipy.dexi.DexiFunction* is defined for each

aggregate attribute, where it serves for the aggregation or discretization of the attribute's inputs to values of that attribute.

Parameters

- **name** (*str, optional*) – Attribute name. Defaults to "".
- **description** (*str, optional*) – Attribute description. Defaults to "".
- **id** (*Optional [str] optional*) – Attribute ID. By default equals to `name`, but may be adjusted for uniqueness in the model context. Defaults to None.
- **inputs** (*Sequence [DexiAttribute], optional*) – A vector of this attribute's input attributes. Defaults to None.
- **parent** (*Union [DexiAttribute, DexiiModel], optional*) – This attribute's parent attribute in the DEXi mopdel tree. By convention, `model.root.parent == model`. Defaults to None.
- **link** (*DexiAttribute, optional*) – A link to an alias of this attribute. Defaults to None.
- **scale** (*DexiScale, optional*) – A scale associated with this attribute. Defaults to None.
- **funct** (*DexiFunction, optional*) – An aggregation/discretization function associated with this attribute. Defaults to None.

affects(*attribute*)

Checks if this attribute affects `attribute`. An attribute is affected if it lies on the path from the affecting attribute toward the model root.

Parameters **attribute** (*DexiAttribute*) – An attribute checked for being affected.

Returns Is `attribute` affected by this attribute?

Return type bool

att_str()

Returns a string representation of the main non-empty attributes of this object.

Returns A string representation of a dictionary containing the non-empty values of `name`, `id`, `description`, `inputs`, `link`, `scale` and `funct`.

Return type str

count()

Returns the number of input attributes of this attribute.

Returns The number of input attributes.

Return type int

dim()

Returns dimensions of this attribute's input attributes.

Returns A list of all input attribute's scale sizes. May contain None elements for undefined input scales.

Return type List[Optional[int]]

inp_index(*inp*)

Returns the index of `inp` in `self.inputs`.

Parameters **inp** (*DexiAttribute*) – An attribute.

Returns Index of `inp`, or None if not found.

Return type Optional[int]

is_aggregate()

Checks if this attribute is aggregate.

Returns Is this attribute aggregate?

Return type bool

`is_basic`(*include_linked=True*)
Checks if this attribute is basic.

Parameters `include_linked` (*bool, optional*) – Whether or not a linked attribute is considered basic. Defaults to True.

Returns Is this attribute basic?

Return type bool

`is_link`()
Checks if this attribute is linked.

Returns Is this attribute linked, i.e., has a defined `self.link`?

Return type bool

`level`()
Returns the level of this attribute in the DEXi model. The level of `dexipy,dexi.DexiModel.root` is 0.

Returns The level of this attribute.

Return type int

`model`()
Returns a `dexipy,dexi.DexiModel` to which this attribute belongs.

Returns The model containing this attribute. None might be returned if this attribute is improperly "wired" (using `self.parent`) in the enclosing model.

Return type Optional[*DexiModel*]

`ninp`()
Returns the number of input attributes of this attribute.

Returns The number of input attributes.

Return type int

`structure`()
Makes a specific indentation string used in DEXiPy to print DEXi model structure.

This method calls *dexipy.dexi.DexiAttribute.tree_indent()* with arguments that indent attributes by two characters per level.

Returns An DEXiPy indentation string, including connections with other attributes.

Return type str

`tree_indent`(*none=' '*, *thru='|'*, *link='*'*, *last='+'*, *line='-'*)
Creates a string indicating the level of this attribute and its connections with other attributes in the model. In model printouts, such a string can be used in front of the attribute name or ID, achieving an indented output.

Parameters

- `none` (*str, optional*) – No connection at a given tree level. Defaults to " ".
- `thru` (*str, optional*) – Vertical connection to attributes displayed below this one. Defaults to "|".
- `link` (*str, optional*) – Horizontal connection to this attribute and vertical connection to attributes displayed below. Defaults to "*".
- `last` (*str, optional*) – Horizontal connection to the attribute that occurs as last child of the parent attribute. Defaults to "+".

- **line** (*str, optional*) – A horizontal line pointing to this attribute. Defaults to "-".

Returns An indentation string, including connections with other attributes.

Return type str

class dexipy.dexi.DexiContinuousScale(*order=DexiOrder.ascending*, *lpoint=- inf*, *hpoint=inf*)

Bases: *dexipy.dexi.DexiScale*

DexiContinuousScale is a scale class representing continuous scales in DEXi.

Parameters

- **order** (DexiOrder, *optional*) – Preferential order of the scale. Defaults to DexiOrder.ascending.
- **lpoint** (*float*) – Defines the interval [-Infinity, lpoint]. All float values lying in this interval are considered to be of a DexiQuality.bad quality (for order = DexiOrder.ascending) or DexiQuality.good quality (for order = DexiOrder.descending).
- **hpoint** (*float*) – Defines the interval [hpoint, +Infinity]. All float values lying in this interval are considered to be of a DexiQuality.good quality (for order = DexiOrder.ascending) or DexiQuality.bad quality (for order = DexiOrder.descending).

equal(*scl*)

Checks whether or not this scale is equal to another scale scl.

Parameters **scl** (*Any*) – Expected a *dexipy.dexi.DexiScale* object.

Returns Is this scale equal to scl?

Return type bool

is_continuous()

Returns True is this scale is continuous.

Returns Is this scale continous?

Return type bool

scale_str()

Returns a short string representing the scale to be used in DEXi model printouts.

Returns A short string representation of this scale.

Return type str

value_quality(*value*)

Returns the quality (value class) of value.

Parameters **value** (*Any*) – Some scale value, usually a number or string.

Returns Value quality.

Return type Union[None, *DexiQuality*]

class dexipy.dexi.DexiDiscreteScale(*values*, *order=DexiOrder.ascending*, *descriptions=[]*, *quality=[]*)

Bases: *dexipy.dexi.DexiScale*

DexiDiscreteScale is a scale class representing qualitative (symbolic, discrete, verbal) scales in DEXi. Such scales are typical for DEXi models and are the only scale type currently supported by the DEXi software.

An attribute associated with a discrete scale can take values from a finite (and usually small) set of string values contained in self.values. Additionally, each of these values is associated *dexipy.types.DexiQuality*; the latter are contained in the list self.quality, which is of the same length as self.values.

Parameters

- **order** (DexiOrder, *optional*) – Preferential order of the scale. Defaults to DexiOrder.ascending.
- **values** (*List[str]*) – A list of qualitative scale values. Example: `self.values = ["low", "medium", "high"]`.
- **descriptions** (*List[str], optional*) – A list of textual descriptions of the corresponding `self.values`. If necessary, the list is internally paddedd to the same length as `self.values`.
- **quality** (*List[DexiQuality], optional*) – A list of qualities, corresponding to `self.values`. Should be of the same length as `self.values`. Defaults to [], which assigns default qualities using *dexipy.dexi.DexiDiscreteScale.default_quality()*.

`count()`

Returns the number of discrete values of this scale.

Returns The number of values or 0 for non-discrete scales.

Return type int

`classmethod default_quality(order, nvals)`

Makes a default list of value qualities.

Parameters

- **order** (DexiOrder) – Preferential order of the scale.
- **nvals** (*int*) – The number of discrete scale values.

Returns

A list of length `nvals` of value qualities. The list depends on `order` as follows:

- DexiOrder.ascending: Returns [DexiQuality.bad, . . . , DexiQuality.good]
- DexiOrder.descending: Returns [DexiQuality.good, . . . , DexiQuality.bad]
- DexiOrder.none: Returns [. . .]

Here, ". . . " denotes a sufficiently long sequence of DexiQuality.none.

Return type List[*DexiQuality*]

`equal(scl)`

Checks whether or not this scale is equal to another scale `scl`.

Parameters **scl** (*Any*) – Expected a *dexipy.dexi.DexiScale* object.

Returns Is this scale equal to `scl`?

Return type bool

`full_range()`

Returns the full range of admissible values for this scale.

Returns A set of all values or None for non-discrete scales.

Return type Union[None, Set[int]]

`is_discrete()`

Returns True is this scale is discrete.

Returns Is this scale discrete?

Return type bool

`scale_str()`

Returns a short string representing the scale to be used in DEXi model printouts.

Returns A short string representation of this scale.

Return type str

`value_index(value)`
Returns the index of value in the scale.

Parameters `value` (*str*) – Some string.

Raises `ValueError` – When `value` was not found.

Returns Index of `value` in `self.values`.

Return type int

`value_index_or_none(value)`
Returns the index of value in the scale or None if not found.

Parameters `value` (*str*) – Some string.

Returns Index of `value` in `self.values` or None if `value` was not found.

Return type Optional[int]

`value_quality(value)`
Returns the quality (value class) of `value`.

Parameters `value` (*Any*) – Some scale value, usually a number or string.

Returns Value quality.

Return type Union[None, *DexiQuality*]

`class dexipy.dexi.DexiDiscretizeFunction(attribute=None, bounds=[], assoc=[], values=[])`
Bases: *dexipy.dexi.DexiFunction*

`DexiDiscretizeFunction` represents DEXi functions that discretize numerical values of continuous attributes to qualitative values of discrete attributes. A `DexiDiscretizeFunction` can be associated only with a discrete attribute that has exactly one continuous input. Then, the function discretizes numeric values of the input attribute and maps them to discrete values of the parent attribute.

Objects of class `DexiDiscretizeFunction` define discretization rules in terms of three lists: `values`, `bounds` and `assoc`. Using `n = self.nvals()` to denote the length of `values`, the required lengths of `bounds` and `assoc` are `n - 1`,

The list `bounds` refers to values of the input attribute and partitions its scale in `n` intervals [-Infinity, bound[0]], [bound[0], bound[1]], ..., [bound[n - 1], +Infinity].

The list `values` then defines the output DEXi values for each interval.

The list `assoc` contains *dexipy.types.BoundAssoc* elements that indicate to which interval, lower or higher, belong the corresponding `bounds`.

When creating a `DexiDiscretizeFunction`, the determining argument is `bounds`. The remaining two arguments, `values` and `assoc` are padded to the right length, possibly inserting None and `BoundAssoc.down`, respectively.

Parameters

- `attribute` (*Optional[DexiAttribute], optional*) – A *dexipy.dexi.DexiAttribute* to which this function is assigned. The attribute is required to be discrete (i.e., associated with *dexipy.dexi.DexiDiscreteScale*) and must have exactly one continuous attribute (i.e., associated with *dexipy.dexi.DexiContinuousScale*). Defaults to None.
- `bounds` (*List[float], optional*) – List of bounds. Defaults to [].
- `assoc` (*list, optional*) – List of bound associations. Defaults to [].

- **values** (*List [DexiValue], optional*) – List of output DEXi values corresponding to each interval induced by **bounds**. Defaults to [].

bound_assoc(*idx*, *default=None*)
Returns association of the **idx**-th bound.

Parameters

- **idx** (*int*) – A bound index.
- **default** (BoundAssoc, *optional*) – An optional association returned for out-of-bound indices. Defaults to None.

Returns Association of the **idx**-th bound or **default** when **idx** is out of bounds.

Return type Optional[*BoundAssoc*]

funct_str()
Creates a short string for displaying information about the function in printouts.

Returns A short string, whose contents depends on this function's type.

Return type str

nargs()
Returns the number of function arguments.

Returns The number of this function's arguments.

Return type int

nvals()
Returns this function size.

Returns The number of this function's decision rules. Returns 0 for function types that do not have rules.

Return type int

value(*args*)
Returns the function value for given arguments.

Parameters **args** (*float*) – A single float function argument.

Returns Function value for the given argument.

Return type DexiValue

class dexipy.dexi.DexiFunction(*attribute=None*)
Bases: object

DexiFunction is a base class for representing DEXi aggregation and discretization functions.

Parameters **attribute** (*Optional [DexiAttribute], optional*) – A *dexipy.dexi.DexiAttribute* to which this function is assigned. Defaults to None.

evaluate(*args*)
A wrapper around *dexipy.dexi.DexiFunction.value()* that returns None when **value()** raises an error.

Parameters **args** (*Any*) – Expected is a single float number or a list of integer arguments passed to the function.

Returns Function value for the given **args**, or None when the value cannot be determined for the given **args**.

Return type DexiValue

funct_str()
Creates a short string for displaying information about the function in printouts.

Returns A short string, whose contents depends on this function's type.

Return type str

`nargs()`
Returns the number of function arguments.

Returns The number of this function's arguments.

Return type int

`nvals()`
Returns this function size.

Returns The number of this function's decision rules. Returns 0 for function types that do not have rules.

Return type int

`value(`*args*`)`
Returns the function value for given arguments.

Parameters **`args`** (*`Any`*) – Expected is a single float number or a list of integer arguments passed to the function.

Raises `ValueError` – When function value cannot be determined for the given `args`.

Returns Function value for the given `args`.

Return type DexiValue

`class dexipy.dexi.DexiModel(`*name=''*, *description=''*, *root=None*, *linking=False*`)`
Bases: `object`

`DexiModel` is the class that represents a DEXi model in Python.

In DEXiPy, DexiModel objects are normally created by reading from a `.dxi` file, previously developed by the DEXi software. Reading models from files ensures that they are properly "wired" into a consistent structure of attributes, scales and functions. In principle, all fields of a DexiModel should be thus considered read-only. DEXiPy does not provide any explicit functionality for creating and changing DEXi models. Of course, models can still be created and modified in Python, but without any integrity or consistency guarantees.

Parameters

- **`name`** (*`str, optional`*) – Model name. Defaults to "".
- **`description`** (*`str, optional`*) – An optional textual description of the model. Defaults to "".
- **`root`** (*`Optional[DexiAttribute], optional`*) – The virtual root of all subtrees/hierarchies of attributes in the model. Defaults to None.
- **`linking`** (*`bool, optional`*) – Indicates whether or not the model uses linked attributes, which are used in DEXi to represent hierarchies of attributes (i.e., directed acyclic graphs) rather than trees. Defaults to False.

`attributes`
List of all model attributes, using the depth-first order.

Type List[*DexiAttribute*]

`natt`
Length of `self.attributes`.

Type int

`att_names`
List of attribute names in the model, in the order of `self.attributes`.

Type List[str]

`att_ids`
List of attribute IDs in the model, in the order of `self.attributes`.

Type List[str]

`basic`
List of all basic attributes.

Type List[*DexiAttribute*]

`aggregate`
List of all aggregate attributes.

Type List[*DexiAttribute*]

`links`
List of all linked attributes.

Type List[*DexiAttribute*]

`non_root`
List of all `self.attributes` without `self.root`.

Type List[*DexiAttribute*]

`basic_names`
List of all basic attributes's names.

Type List[str]

`aggregate_names`
List of all aggregate attributes' names.

Type List[str]

`links_names`
List of all linked attributes' names.

Type List[str]

`non_root_names`
List of all `self.non_root` attributes' names.

Type List[str]

`basic_ids`
List of all basic attributes's IDs.

Type List[str]

`aggregate_ids`
List of all aggregate attributes' IDs.

Type List[str]

`links_ids`
List of all linked attributes' IDs.

Type List[str]

`non_root_ids`
List of all `self.non_root` attributes' IDs.

Type List[str]

`alternatives`
A list of DEXi decision alternatives defined as part of the model.

Type DexiAlternatives

`alt_table`(*alternatives=None*, *alt_head='alternative'*, *sel_att=[]*, *basic=False*, *aggregate=False*, *decimals=None*, *transpose=False*, *use_dict=True*, *text=False*, *default=''*)

Creates a string that, when printed out, displays `alternatives` in a tabular form.

Parameters

- **`alternatives`** (*Optional [DexiAltData], optional*) – A single alterative or a list of alternatives. Defaults to None, which selects `self.alternatives`.
- **`alt_head`** (*str, optional*) – Text to be displayed in the topmost-left cell. Defaults to "alternative".
- **`sel_att`** (*List [str], optional*) – A list of attribute IDs to be displayed. Defaults to [], which selects all non-root attributes, unless overriden by `basic` and `aggregate` arguments.
- **`basic`** (*bool, optional*) – Selects all basic attributes for display. Considered only when `sel_att == []`. Defaults to False.
- **`aggregate`** (*bool, optional*) – Selects all aggregate attributes for display. Considered only when `sel_att == []`. Defaults to False.
- **`decimals`** (*Optional [int], optional*) – The number of decimal places used to display floating-point numbers. Defaults to None.
- **`transpose`** (*bool, optional*) – Normally, the table displays attributes in rows and alternatives in columns. Setting this argument to True transposes the table. Defaults to False.
- **`use_dict`** (*bool, optional*) – Whether or not the dictionary-form is used for displaying value distributions (rather than list-form). Defaults to True.
- **`text`** (*bool, optional*) – Whether of not value cells are textualized using `dexipy.dexi.textualize_alternatives()`. Defaults to False.
- **`default`** (*str, optional*) – The string used to display None values. Defaults to "".

Returns A multi-line string representation of `alternatives`.

Return type str

`alt_text`(*alternatives=None*, ***kwargs*)

Creates a string that, when printed out, displays `alternatives` in a tabular and textualized form. It is meant as a convenient abbreviation for calling `self.alt_table(alternatives, text = True, **kwargs)`.

Parameters

- **`alternatives`** (*Optional [DexiAltData], optional*) – A single alterative or a list of alternatives. Defaults to None, which selects `self.alternatives`
- **`**kwargs`** – Other arguments passed to *dexipy.dexi.DexiModel.alt_table()*.

Returns A multi-line string representation of `alternatives`.

Return type str

`alternative`(*name=None*, *description=None*, *alt=None*, *sel_att=[]*, *basic=False*, *aggregate=False*, *default=None*, *deindex=False*, *values=None*, ***kwargs*)

Defines a new decision alternative.

Parameters

- **`name`** (*Optional [str], optional*) – Name of the alternative. Defaults to None.
- **`description`** (*Optional [str], optional*) – Textual description of the alternative. Defaults to None.

- **alt** (*Optional[DexiAlternative], optional*) – A base alternative. If specified, all the values of `alt` are copied to the resulting alternative prior to adding or overwriting them with values specified in other arguments. Defaults to None.
- **sel_att** (*List[str], optional*) – The list of attributes whose values are added to the resulting alternative and initialized to the None value. Defaults to [].
- **basic** (*bool, optional*) – If True, add all basic attributes. Considered only when `sel_att == []`. Defaults to False.
- **aggregate** (*bool, optional*) – If True, add all aggregate attributes. Considered only when `sel_att == []`. Defaults to False.
- **default** (*DexiValue, optional*) – Default attribute value that is assigned unless it has been specified in other arguments. Defaults to None.
- **deindex** (*bool, optional*) – Whether or not the resulting alternative data is deindexed after setup, using `dexipy.dexi.deindex_alternative()`. Defaults to False.
- **values** (*Optional[DexiAlternative], optional*) – A dictionary of `attribute_id: attribute_value` or `attribute_index: attribute_value` elements, added to the resulting dictionary in addition to or overwriting values set by other arguments. Defaults to None.
- ****kwargs** – Keyword method arguments in the form `attribute_id = attribute_value`. This form is possible only with attributes whose IDs match the Python's syntax for variable names.

Returns Alternative data represented as a dictionary consisting of `attribute_id: attribute_value` or `attribute_index: attribute_value` elements.

Return type DexiAlternative

`att_index`(*attname*, *use_id=True*)

Given an ID or name of an attribute, find its index in `self.attributes`.

Parameters

- **attname** (*str*) – Attribute ID or name.
- **use_id** (*bool, optional*) – Whether or not to search by attribute ID (True) or name (False). Defaults to True.

Returns Attribute index or None if not found. When searching by name, attribute names may not be unique and only the first index is returned in this case.

Return type Optional[int]

`att_indices`(*attname*)

Returns the index of attribute named `attname` in `self.attributes`.

Parameters **attname** (*str*) – Attribute name.

Returns A list of attributes. Notice that attribute names may not be unique, resulting in a list containing more that one attribute.

Return type List[int]

`att_stat()`

Counts attributes of different types in the model.

Returns A dictionary of the form `{"all": int, "basic": int, "aggregate": int, "link": int}`, containing counts of the corresponding attribute types.

Return type Dict[str, int]

`attrib`(*find*)
A general method for finding an attribute in the model.

Parameters `find` (*Union[DexiAttribute, str, int]*) – An attribute object, ID or index.

Returns DexiAttribute object if found in the model, or None otherwise.

Return type Optional[*DexiAttribute*]

`check_alternative`(*alt*, *aggregate=False*)
Checks the data representing alternative `alt`, and reports found errors and warnings.

Parameters

- `alt` (*DexiAlternative*) – A DEXi alternative.
- `aggregate` (*bool, optional*) – Whether values of aggregate attributes are checked or not. The rationale for this argument is that the evaluation, which typically follows checking, overwrites values of aggregate attributes. Defaults to False.

Returns A dictionary of the form `{"errors": <list of error strings>, "warnings": <list of warning strings>}`.

Return type Dict[str, List[str]]

`check_alternatives`(*alternatives=None*, *aggregate=False*)
A vectorized version of `dexipy.dexi.check_alternative()`. May check a single alternative or a list of alternatives.

Parameters

- `alternatives` (*Optional[DexiAltData], optional*) – A single alterative or a list of alternatives. Defaults to None, which selects `self.alternatives`.
- `aggregate` (*bool, optional*) – Whether values of aggregate attributes are checked or not. Defaults to False.

Raises `ValueError` – When `alternatives` is not of a DexiAltData type.

Returns A dictionary of the form `{"errors": <list of error strings>, "warnings": <list of warning strings>}`.

Return type Dict[str, List[str]]

`collect_attributes`(*att*)
Returns a list of attributes, obtained by a recursive depth-first travesal of the subtree rooted at `att`.

Parameters `att` (DexiAttribute) – The root attribute of traversal.

Returns List of attributes found in the subtree rooted at `att`, including `att`.

Return type List[*DexiAttribute*]

`deindex_alternative`(*alt*)
Converts a `DexiAlternative` dictionary replacing all attribute indices with the corresponding attribute IDs.

Args: alt (DexiAlternative): An alternative, represented by a dictionary.

Returns: DexiAlternative: A dictionary with all numeric indices replaced with string IDs.

Parameters `alt` (*DexiAlternative*) –

Return type DexiAlternative

`evaluate`(*alternatives=None*, *method='set'*, *root=None*, *prune=[]*, *pre_check=False*, *bounding=False*, *in_place=False*, *eval_param=None*)

Evaluates decision alternative(s).

Please see *Evaluation of Alternatives* for more information about the evaluation process and evaluation methods used in DEXiPy.

Parameters

- **alternatives** (*Optional[DexiAltData], optional*) – A single DexiAlternative or a list of alternatives (DexiAlternatives). Defaults to None, which selects `self.alternatives`.
- **method** (*str, optional*) – One of the strings "set", "prob", "fuzzy", "fuzzynorm" that select the evaluation method. Defaults to "set".
- **root** (*Optional[DexiAttribute], optional*) – The topmost (root) attribute of the evaluation. Defaults to None, which selects `self.root`.
- **prune** (*List[str], optional*) – List of attribute IDs at which the evaluation is "pruned". This means that input attributes below some pruned attribute are not evaluated and the pruned attribute is treated as an input (basic) attribute for this evaluation. Defaults to [].
- **pre_check** (*bool, optional*) – Whether of not `alternatives` are checked by `dexipy,dexi.DexiModel.check_alternatives()` prior to evaluation. Defaults to False.
- **bounding** (*bool, optional*) – Whether or not the evaluation keeps calculated values within bounds prescribed by the corresponding scales. Defaults to False.
- **in_place** (*bool, optional*) – If True, evaluation modifies `alternatives` in place, otherwise a copy is made and returned by the method . Defaults to False.
- **eval_param** (*Optional[eval.DexiEvalParameters], optional*) – Optional *dexipy.eval.DexiEvalParameters*, that may customize the normalization and aggregation methods used in the evaluation. Defaults to None.

Returns Returns an evaluated alternative or list of alternatives, depending on the type of the `alternatives` argument.

Return type DexiAltData

`find_attributes`(*select*)

A vectorized version of *dexipy.dexi.DexiModel.attrib()*. Returns a list of attributes found in the model according to the `select` specification.

Parameters **select** (*List[Union[DexiAttribute, str, int]]*) – A list of DEXiAttributes, attribute IDs or attribute indices.

Returns A list of DexiAttributes w.r.t. the correspoding

Return type

List[Optional[DexiAttribute]]

`select` criteria. May contain None elements for non-found attributes.

`link_attributes`()

Carries out the linking of attributes.

DEXi attributes that have the same names and value scales, and satisfy some other constraints to prevent making cycles in the model, are linked together so that they logically represent a single attribute. In this way, a tree of attributes is conceptually turned in a hierarchy (directed acyclic graph).

Return type None

`make_ids`(*max_len=None*, *var_names=False*)
A helper method for creating attribute IDs from attribute names. Generated IDs are assigned to `self.att_ids` and propagated through the model.

Parameters

- **max_len** (*Optional[int], optional*) – Maximum length of IDs (excuding `"_<idx>"` strings added to ensure the uniqueness of IDs). Defaults to None, which leaves the attribute name length intact.
- **var_names** (*bool, optional*) – Whether or not IDs should be generated so that they conform with Python's syntax for variable names. This allows using attribute IDs as function arguments, for example with *dexipy.dexi.DexiModel.alternative()*. Defaults to False.

Return type None

`parent_attributes`(*att*)
Traverses the subtree of attributes rooted at `att` and sets the `parent` field of each attribute.

Parameters **att** (DexiAttribute) – The root attribute of traversal.

Return type None

`propagate_ids`()
Propagates `self.att_ids` to other `self.*_ids` lists and to IDs of individual attributes in the model. Should be explicitly called after a modification of `self.att_ids`.

Return type None

`setup`()
A helper method called as the last step when creating a DexiModel.

This method assigns `self.root`, sets attributes' parents, generates attribute IDs and creates all model-level attribute lists. If `self.linking` is True, it also traverses the model and links attributes.

When manually changing elements of a DexiModel, calling this method might (but need not) reestablish a consistent model structure.

Raises **ValueError** – When `self.root` is undefined or not of the DexiAttribute class.

Return type None

`textualize_alternative`(*alt*, *decimals=None*, *use_dict=True*)
Converts an internal representation of alternative `alt`, which typically uses numerical attribute and value indices, to a more comprehensible dictionary that uses string attribute IDs and value names.

Parameters

- **alt** (*DexiAlternative*) – An alternative, represented by a dictionary.
- **decimals** (*Optional[int], optional*) – The number of decimal places used to display floating-point numbers. Defaults to None.
- **use_dict** (*bool, optional*) – Whether or not the dictionary-form is used for displaying value distributions (rather than list-form). Defaults to True.

Returns `alt` converted to an equivalent dictionary that uses strings instead of numbers.

Return type DexiAlternative

`textualize_alternatives`(*alts*, *decimals=None*, *use_dict=True*)
A vectorized version of `dexipy.dexi.textualize_alternative()`. May textualize a single alternative or a list of alternatives.

Parameters

- **alts** (*DexiAltData*) – A single alterative or a list of alternatives. Defaults to None, which selects `self.alternatives`.
- **decimals** (*Optional[int], optional*) – The number of decimal places used to display floating-point numbers. Defaults to None.
- **use_dict** (*bool, optional*) – Whether or not the dictionary-form is used for displaying value distributions (rather than list-form). Defaults to True.

Returns `alts` converted to an equivalent representation that uses strings instead of numbers.

Return type DexiAltData

`class dexipy.dexi.DexiScale(order=DexiOrder.ascending)`

Bases: `object`

`DexiScale` is a base class for representing DEXi value scales.

A value scale defines the type and set of values that can be assigned to some *dexipy.dexi.DexiAttribute*.

dexipy.dexi.DexiScale defines attributes and methods common to all scales. Normally, this class should not be created itself, but only through derived scale classes, *dexipy.dexi.DexiContinuousScale* and *dexipy.dexi.DexiDiscreteScale*.

Parameters **order** (DexiOrder, *optional*) – Preferential order of the scale. Defaults to DexiOrder.ascending.

`count()`

Returns the number of discrete values of this scale.

Returns The number of values or 0 for non-discrete scales.

Return type int

`equal(scl)`

Checks whether or not this scale is equal to another scale `scl`.

Parameters **scl** (*Any*) – Expected a *dexipy.dexi.DexiScale* object.

Returns Is this scale equal to `scl`?

Return type bool

`classmethod equal_scales(scl1, scl2)`

Checks wheter of not two scales are equal.

Parameters

- **scl1** (*Any*) – First scale.
- **scl2** (*Any*) – Second scale.

Returns Returns True if both scales are None or both are not None and `scl1.equals(scl2)`. Returns False for non-scale arguments.

Return type bool

`full_range()`

Returns the full range of admissible values for this scale.

Returns A set of all values or None for non-discrete scales.

Return type Union[None, Set[int]]

`is_continuous()`

Returns True is this scale is continuous.

Returns Is this scale continous?

Return type bool

`is_discrete()`
Returns True is this scale is discrete.

Returns Is this scale discrete?

Return type bool

`scale_str()`
Returns a short string representing the scale to be used in DEXi model printouts.

Returns A short string representation of this scale.

Return type str

`value_index(value)`
Returns the index of `value` in the scale.

Returns Integer index or None for non-discrete scales.

Return type Union[None, int]

Parameters `value` (*Any*) –

`value_quality(value)`
Returns the quality (value class) of `value`.

Parameters `value` (*Any*) – Some scale value, usually a number or string.

Returns Value quality.

Return type Union[None, *DexiQuality*]

`class dexipy.dexi.DexiTabularFunction(attribute=None, dim=None, values=None, low=None, high=None)`

Bases: *dexipy.dexi.DexiFunction*

`DexiTabularFunction` represents the most common function type used in DEXi models. Functions of this type aggregate discrete attribute values according to *decision rules*, defined in terms of a *decision table*.

A decision table contains as many decision rules as there are possible combinations of input attributes' values. For instance, if some attribute has two inputs whose discrete scales have three and four values, respectively, then the number of rules is equal to $3 * 4 = 12$. Each rule defines the value of the attribute for one of the possible combinations of values of the attribute's inputs. For example, a decision rule `(0, 1): 2` maps the zeroth and first value of the first and second input attribute to value 2 of the output attribute. A decision table is represented as a dictionary of such rules. Thus, it can be interpreted as a lookup table that, given a vector of values of `attribute.inputs`, (i.e., function arguments) returns the corresponding output attribute value. In general, the output value can be any *DEXi value*.

Notes

- In order to properly define the context in which the newly created function operates, at least one of the arguments, `attribute` or `dim`, must be given.
- Decision rules must be defined either directly by the `values` argument, or indirectly using the `low` and `high` strings.

Parameters

- `attribute` (*Optional[DexiAttribute], optional*) – A *dexipy.dexi.DexiAttribute* to which this function is assigned. Both the attribute and its inputs are required to be discrete (i.e., associated with *dexipy.dexi.DexiDiscreteScale*. Defaults to None.
- `dim` (*Any, optional*) – A list of integers, representing the size of the corresponding input attribute dimensions. Defaults to None.

- **values** (*Optional[List[DexiValue]], optional*) – A decision table. Represented in the form of a dictionary that maps function arguments (represented as integer tuples) to function values (represented as *DEXi values*). Defaults to None.
- **low** (*Optional[str], optional*) – A DEXi string representing lower bounds of function values in `.dxi` files. Defaults to None.
- **high** (*Optional[str], optional*) – A DEXi string representing upper bounds of function values in `.dxi` files. When None, it is assumed that `low == high`. Defaults to None.

Raises

- **ValueError** – When at least one of the arguments `attribute` or `dim` is not given.
- **ValueError** – When `dim` is not a list of integers, has length different from `attribute.count()` or contains dimensions incompatible with input attributes.
- **ValueError** – When `low` and/or `high` are of length different than the expected number of decision rules.

`funct_str()`
Creates a short string for displaying information about the function in printouts.

Returns A short string, whose contents depends on this function's type.

Return type str

`nargs()`
Returns the number of function arguments.

Returns The number of this function's arguments.

Return type int

`nvals()`
Returns the function space size (the total number of input attributes' value combinations).

Returns Size of the space defined by this function's input attributes.

Return type int

`value(args)`
Returns the function value for given arguments.

Parameters **args** (*Iterable[int]*) – A vector of integer arguments passed to the function.

Raises **ValueError** – When function value cannot be determined for the given `args`.

Returns Function value for the given `args`.

Return type DexiValue

`value_vector()`
Returns the vector of this function's values.

Returns A list of DEXi values, extracted from `self.values`, in the order determined using *dexipy.utils.cartesian_product()*.

Return type List[DexiValue]

`dexipy.dexi.alt_lines(alternatives, alt_head='alternative', sel_att=[], transpose=False, default='')`
Makes a list of strings that represent `alternative` in a tabular form.

Parameters

- **alternatives** (*DexiAltData*) – A DEXi alternative or a list of DEXi alternatives.

- **alt_head** (*str, optional*) – Text to be included as the top-left element of the table. Defaults to "alternative".
- **sel_att** (*List[str], optional*) – A list of selected attribute IDs. Any remaining IDs in `alternatives` are ignored. Defaults to [], which selects all attributes.
- **transpose** (*bool, optional*) – Normally, table columns correspond to alternatives. Setting `transpose` to True makes columns corresponding to attributes. Defaults to False.
- **default** (*str, optional*) – A string to display None values. Defaults to "".

Returns A list of lines that can be joined together and printed.

Return type List[str]

`dexipy.dexi.alt_name`(*alt*, *default=''*, *keyword='name'*)

Extracts alternative name from DexiAlternative data.

Parameters

- **alt** (*DexiAlternative*) – A DEXi alternative.
- **default** (*str, optional*) – Name returned when no name has been found. Defaults to "".
- **keyword** (*str, optional*) – Key string for finding alternative name. Defaults to "name".

Returns Name of the alternative, or `default` if not found.

Return type str

`dexipy.dexi.alternative`(*name=None*, *description=None*, *alt=None*, *values=None*, ***kwargs*)

Defines a new decision alternative.

Parameters

- **name** (*Optional[str], optional*) – Name of the alternative. Defaults to None.
- **description** (*Optional[str], optional*) – Textual description of the alternative. Defaults to None.
- **alt** (*Optional[DexiAlternative], optional*) – A base alternative. If specified, all the values of `alt` are copied to the resulting alternative prior to adding or overwriting them with `values`. Defaults to None.
- **values** (*Optional[DexiAlternative], optional*) – A dictionary of `attribute_id: attribute_value` or `attribute_index: attribute_value` elements, added to the resulting dictionary. Defaults to None.
- ****kwargs** – Keyword function arguments in the form `attribute_id = attribute_value`. This form is possible only with attributes whose IDs match the Python's syntax for variable names.

Raises `ValueError` – When `alt` is not a dictionary.

Returns Alternative data represented as a dictionary consisting of `attribute_id: attribute_value` or `attribute_index: attribute_value` elements.

Return type DexiAlternative

`dexipy.dexi.att_names`(*atts*, *use_id=True*)

Given a sequence of attributes, returns the list of names of IDs of that attribute.

Parameters

- **atts** (*Iterable[DexiAttribute]*) – A sequence of attributes.

- **use_id** (*bool, optional*) – Whether or not to use attribute IDs (True) or names (False). Defaults to True.

Returns A list of attributes' IDs or names.

Return type List[str]

dexipy.dexi.**bounded_scale_value**(*value*, *scale*)

A wrapper around *dexipy.dexi.scale_value()* that ensures that the resulting value lies within the bounds set up by `scale`.

Parameters

- **value** (*Any*) – Normally, a *DEXi value* is expected.
- **scale** (*Any*) – A *dexipy.dexi.DexiScale* object.

Raises `ValueError` – When `value` cannot be interpreted or bound to `scale` limits.

Returns An interpreted value (see *dexipy.dexi.scale_value()*) and reduced to `scale` bounds.

Return type DexiValue

dexipy.dexi.**columnize**(*alternatives*, *alt_head='alternative'*, *sel_att=[]*, *transpose=False*, *default=''*)

A helper function that "columnizes" DexiAltData. A suitable list of columns is produced for further processing.

Parameters

- **alternatives** (*DexiAltData*) – A DEXi alternative or a list of DEXi alternatives.
- **alt_head** (*str, optional*) – Text to be included as the first element of the first column. Defaults to "alternative".
- **sel_att** (*List[str], optional*) – A list of selected attribute IDs. Any remaining IDs in `alternatives` are ignored. Defaults to [].
- **transpose** (*bool, optional*) – Normally, columns correspond to alternatives. Setting `transpose` to True makes columns corresponding to attributes. Defaults to False.
- **default** (*str, optional*) – A string to display None values. Defaults to "".

Returns List of columns. Each column is a list of strings. All columns are padded to the same length.

Return type List[List[str]]

dexipy.dexi.**evaluate**(*model*, *alternatives=None*, *method='set'*, *root=None*, *prune=[]*, *pre_check=False*, *bounding=False*, *in_place=False*, *eval_param=None*)

Evaluates decision alternative(s).

Please see *Evaluation of Alternatives* for more information about the evaluation process and evaluation methods used in DEXiPy.

Parameters

- **model** (DexiModel) – A DexiModel. Required.
- **alternatives** (*Optional[DexiAltData], optional*) – A single DexiAlternative or a list of alternatives (DexiAlternatives). Defaults to None, which selects `model.alternatives`. alternatives (DexiAlternatives). Defaults to None, which selects `model.alternatives`.
- **method** (*str, optional*) – One of the strings "set", "prob", "fuzzy", "fuzzynorm" that select the evaluation method. Defaults to "set".
- **root** (*Optional[DexiAttribute], optional*) – The topmost (root) attribute of the evaluation. Defaults to None, which selects `model.root`.

- **prune** (*List [str], optional*) – List of attribute IDs at which the evaluation is "pruned". This means that input attributes below some pruned attribute are not evaluated and the pruned attribute is treated as an input (basic) attribute for this evaluation. Defaults to [].
- **pre_check** (*bool, optional*) – Whether of not `alternatives` are checked by `dexipy,dexi.DexiModel.check_alternatives()` prior to evaluation. Defaults to False.
- **bounding** (*bool, optional*) – Whether or not the evaluation keeps calculated values within bounds prescribed by the corresponding scales. Defaults to False.
- **in_place** (*bool, optional*) – If True, evaluation modifies `alternatives` in place, otherwise a copy is made and returned by the method. Defaults to False.
- **eval_param** (*Optional [eval.DexiEvalParameters], optional*) – Optional *dexipy.eval.DexiEvalParameters*, that may customize the normalization and aggregation methods used in the evaluation. Defaults to None.

Returns Returns an evaluated alternative or list of alternatives, depending on the type of the `alternatives` argument.

Return type DexiAltData

`dexipy.dexi.read_dexi(`*filename*`)`
Reads a DEXi model from a `.dxi` file.

Parameters **filename** (*str*) – File name.

Returns DEXi model read from the file.

Return type *DexiModel*

`dexipy.dexi.read_dexi_from_string(`*xml*`)`
Reads a DEXi model from a `xml` string.

Parameters **xml** (*str*) – XML string representation of a DEXi model, conforming to the format used in `.dxi` files.

Returns DEXi model read from the string.

Return type *DexiModel*

`dexipy.dexi.scale_of(`*obj*`)`
Returns a DEXi scale related to `obj`.

Parameters **obj** (*Any*) – Expected is a *dexipy.dexi.DexiScale* or *dexipy.dexi.DexiAttribute* object.

Returns Returns `obj` if `obj` is a *dexipy.dexi.DexiScale* object. Returns `obj.scale` if `obj` is a *dexipy.dexi.DexiAttribute* object. Returns None for other object types.

Return type Optional[*DexiScale*]

`dexipy.dexi.scale_value(`*value*, *scale*`)`
Checks and interprets `value` on `scale`.

Parameters

- **value** (*Any*) – Normally, a *DEXi value* is expected.
- **scale** (*Any*) – A *dexipy.dexi.DexiScale* object.

Raises **ValueError** – When scale is undefined, when `value` does not contain a valid DEXi value or the value cannot be interpreted on `scale`.

Returns

An interpreted `value`. The interpretation includes:

- the value "*" is replaced by the actual *dexipy.dexi.DexiScale.full_range()* of the scale,
- the strings "" and "undef..." are interpreted as None,
- float values are admissible only with continuous scales, otherwise they are interpreted as None,
- all value-name strings that occur in value sets, lists and dictionaries, are replaced by the corresponding numeric indices.

Return type DexiValue

dexipy.dexi.value_text(*value*, *scale*, *none=None*, *reduce=False*, *decimals=None*, *use_dict=True*)

Represents value by a human-readable string that can be printed.

Parameters

- **value** (*DexiValue*) – A DEXi value.
- **scale** (*Any*) – Expected a *dexipy.dexi.DexiScale* object.
- **none** (*Optional[str], optional*) – An optional string that is returned when the value cannot be interpreted. Defaults to None.
- **reduce** (*bool, optional*) – Whether or not the value is reduced (see *dexipy.values.reduce_dexi_value()*) prior to processing. Defaults to False.
- **decimals** (*Optional[int], optional*) – The number of decimals used to display float numbers. Defaults to None.
- **use_dict** (*bool, optional*) – Whether or not the dictionary-form is used for displaying value distributions (rather than list-form). Defaults to True.

Returns A string representation of value.

Return type Optional[str]

4.1.4 dexipy.eval module

The module dexipy.eval implements classess and functions for the evaluation of decision alternatives.

class dexipy.eval.DexiEvalMethods

Bases: object

A class defining default *dexipy.eval.DexiEvalParameters* for the evaluation methods implemented in DEXiPy.

The default parameters are set as follows:

```
import dexipy.utils as utl
self.set_method(DexiEvalParameters("set", lambda x: 0, lambda x: 1, utl.
↪norm_none))
self.set_method(DexiEvalParameters("prob", utl.prod, sum, utl.norm_sum))
self.set_method(DexiEvalParameters("fuzzy", min, max, utl.norm_none))
self.set_method(DexiEvalParameters("fuzzynorm", min, max, utl.norm_max))
```

classmethod get_method(*method*)

Gets default evaluation parameters for method.

Parameters **method** (*str*) – Method name.

Raises **ValueError** – When method parameters have not been previously defined for the given method name.

Returns Default parameters of the given method.

Return type *DexiEvalParameters*

classmethod set_method(*method*)
Sets default evaluation parameters for `method`.

Parameters `method` (DexiEvalParameters) – Evaluation parameters with defined method name `method.method`.

Return type None

class dexipy.eval.DexiEvalParameters(*method*, *and_op*, *or_op*, *norm*)
Bases: `object`

A class defining evaluation parameters.

Please see *Evaluation of Alternatives* for more information about the evaluation process and evaluation methods used in DEXiPy.

Parameters

- `method` (*str*) – Method name. One of the strings "set", "prob", "fuzzy", "fuzzynorm".
- `and_op` (*CallableOperator*) – Conjunctive aggregation function.
- `or_op` (*CallableOperator*) – Disjunctive aggregation function.
- `norm` (*CallableNorm*) – Normalization function.

dexipy.eval.EvalMethods = <dexipy.eval.DexiEvalMethods object>
A *dexipy.eval.DexiEvalMethods* object containing default evaluation parameters for all methods implemented in DEXiPy.

dexipy.eval.eval_parameters(*method*, *and_op=None*, *or_op=None*, *norm=None*)
Fetches default evaluation parameters from `EvalMethods` and optionally modifies them considering non-None arguments of this function.

Parameters

- `method` (*str*) – Method name. Required.
- `and_op` (*Optional [CallableOperator], optional*) – If not None, set the conjuntive aggregation function. Defaults to None.
- `or_op` (*Optional [CallableOperator], optional*) – If not None, set the disjuntive aggregation function. Defaults to None.
- `norm` (*Optional [CallableNorm], optional*) – If not None, set the normalization function. Defaults to None.

Returns Fetched and optionally modified evaluation parameters.

Return type *DexiEvalParameters*

dexipy.eval.evaluate(*model*, *alternatives=None*, *method='set'*, *root=None*, *prune=[]*, *pre_check=False*, *bounding=False*, *in_place=False*, *eval_param=None*)
This is the main implementation of the evaluation method in DEXiPy.

While it is possible to call this function directly, it is also possible to avoid importing *dexipy.eval* and run evaluations using *dexipy.dexi.evaluate()* or *dexipy.dexi.DexiModel.evaluate()*.

Please see *Evaluation of Alternatives* for more information about the evaluation process and evaluation methods used in DEXiPy.

Parameters

- `model` (DexiModel) – A DexiModel. Required.
- `alternatives` (*Optional [DexiAltData], optional*) – A single DexiAlternative or a list of alternatives (DexiAlternatives). Defaults to None, which selects `model.alternatives`.

- **method** (*str, optional*) – One of the strings "set", "prob", "fuzzy", "fuzzynorm" that select the evaluation method. Defaults to "set".
- **root** (*Optional [DexiAttribute], optional*) – The topmost (root) attribute of the evaluation. Defaults to None, which selects `model.root`.
- **prune** (*List [str], optional*) – List of attribute IDs at which the evaluation is "pruned". This means that input attributes below some pruned attribute are not evaluated and the pruned attribute is treated as an input (basic) attribute for this evaluation. Defaults to [].
- **pre_check** (*bool, optional*) – Whether of not `alternatives` are checked by `dexipy,dexi.DexiModel.check_alternatives()` prior to evaluation. Defaults to False.
- **bounding** (*bool, optional*) – Whether or not the evaluation keeps calculated values within bounds prescribed by the corresponding scales. Defaults to False.
- **in_place** (*bool, optional*) – If True, evaluation modifies `alternatives` in place, otherwise a deep copy is made and returned by the method. Defaults to False.
- **eval_param** (*Optional [eval.DexiEvalParameters], optional*) – Optional *dexipy.eval.DexiEvalParameters*, which may customize the normalization and aggregation methods used in the evaluation. Defaults to None.

Returns Returns an evaluated alternative or a list of alternatives, depending on the type of the `alternatives` argument.

Return type DexiAltData

dexipy.eval.**evaluation_order**(*att*, *prune=[]*)

Determine the evaluation order of attributes. Interpreted as a sequence, the order guarantees that whenever some attribute is reached as a next candidate for evaluation, all the affecting attributes have already been evaluated.

Parameters

- **att** (DexiAttribute) – The starting point of evaluation.
- **prune** (*List [str], optional*) – A list of attribute IDs at which to prune the evaluatiuon. The evaluation will treat them as if they were basic attributes, not looking to any descendant attributes. Defaults to [].

Returns A list of attrribute IDs in the evaluation order.

Return type List[str]

dexipy.eval.**get_alt_value**(*alt*, *id*)

Returns `alt[id]`.

Parameters

- **alt** (*Any*) – Expected a DexiAlternative.
- **id** (*str*) – Value ID, a key in the `alt` dictionary.

Returns Alternative value corresponding to ID, or None if not found.

Return type DexiValue

4.1.5 dexipy.parse module

The module `dexipy.parse` implements parsing and reading of `.dxi` files.

The only functions interesting for public use are `dexipy.parse.read_dexi()` and `dexipy.parse.read_dexi_from_string()`; both are aliased in the module *dexipy.dexi*.

For more information, please refer to *dexipy.dexi.read_dexi()* and *dexipy.dexi.read_dexi_from_string()* .

4.1.6 dexipy.utils module

Module `dexipy.utils` contains a collection of helper functions used in DEXiPy.

`dexipy.utils.aligned`(*s*, *width=- 1*, *align='l'*)
Pads and/or aligns the string.

Parameters

- **s** (*str*) – Some input string.
- **width** (*int , optional*) – The required length of the resulting string. Defaults to -1, not affecting string length.
- **align** (*str , optional*) – A one-character string in `("l", "c", "r")`, requesting a left, centered or right justification, respectively.

Returns Padded and justified string.

Return type str

`dexipy.utils.cartesian_product`(**dimensions*)
Constructs the cartesian product of ranges, tuples or sets submitted as the function arguments.

Uses `itertools.product()`.

Returns List of all possible combinations of values of `dimensions`.

Return type List

Examples

```
>>> cartesian_product((0, 1), (2, 3, 4))
[(0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4)]
>>> cartesian_product({"a", "b"}, (2, 3, 4))
[('a', 2), ('a', 3), ('a', 4), ('b', 2), ('b', 3), ('b', 4)]
>>> cartesian_product(range(2), range(3))
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

`dexipy.utils.check_str`(*check*, *errors=True*, *warnings=False*)
Makes a string, suitable for printing, from an error/warning dictionary.

Parameters

- **check** (*Dict[str, List[str]]*) – A dictionary in the form `{"errors": [...], "warnings": [...]}`.
- **errors** (*bool , optional*) – Should this function consider errors? Defaults to True.
- **warnings** (*bool , optional*) – Should this function consider warnings? Defaults to False.

Returns A string containing error and warning messages, formatted for printing.

Return type str

dexipy.utils.**col_width**(*column*)

Calculates the maximum width of strings in `column`.

Parameters `column` (*List[str]*) – A list of strings.

Returns Maximum string length.

Return type int

dexipy.utils.**dict_to_list**(*distr*)

Converts a dictionary-form value distribution to a list-form one. Example: `{1: 0.7, 2: 0.2}` is converted to `[0.0, 0.7, 0.2]`.

Parameters `distr` (*Dict[int, float]*) – A dictionary-form value distribution to be converted.

Returns A list-form value distribution.

Return type List[float]

Examples

```
>>> dict_to_list({1: 0.7, 2: 0.2})
[0.0, 0.7, 0.2]
```

dexipy.utils.**distr_to_set**(*distr*, *eps=2.220446049250313e-16*)

Converts a DEXi value distribution to a DEXi value set.

Parameters

- `distr` (*List[float]*) – A distribution to be converted.
- `eps` (*float, optional*) – Threshold for determining whether a distribution element can be considered a member of the resulting set. Defaults to float_info.epsilon.

Returns The set, composed of indices of `distr` elements greater than `eps`.

Return type Set[int]

Examples

```
>>> distr_to_set([0.0, 0.5, 1.0, 0.0])
{1, 2}
>>> distr_to_set([0.0, 0.5, 1.0, 0.0], 0.5)
{2}
```

dexipy.utils.**distr_to_strict_set**(*distr*, *eps=2.220446049250313e-16*)

Converts a DEXi value distribution to a DEXi value set. Only distributions that strictly represent sets, i.e., they contain only 0.0 and 1.0 entries, are converted.

Parameters

- `distr` (*List[float]*) – A distribution to be converted.
- `eps` (*float, optional*) – Allowed tolerance around distribution values, so that they can be considered 0.0 or 1.0. Defaults to float_info.epsilon.

Returns The set, composed of indices of `distr` elements that differ from 0.0 or 1.0 for at most `eps`. None is returned if `distr` contains values that are not sufficiently close to 0.0 or 1.0.

Return type Set[int]

Examples

```
>>> distr_to_strict_set([0.0, 0.5, 1.0, 0.0]) # returns None
>>> distr_to_strict_set([0.0, 1.0, 1.0, 0.0])
{1, 2}
>>> distr_to_strict_set([0.0, 0.9, 1.1, 0.0]) # returns None
>>> distr_to_strict_set([0.0, 0.9, 1.1, 0.0], 0.1)
{1, 2}
```

dexipy.utils.**is_in_range**(*x*, *lb*, *hb*, *la=BoundAssoc.up*, *ha=BoundAssoc.down*)

Checks whether or not the argument `x` lies in the interval bounded by `lb` and `hb`, considering the corresponding bound associations `la` and `ha`.

Parameters

- **x** (*float*) – An integer or floating point value.
- **lb** (*float*) – Lower interval bound.
- **hb** (*float*) – Upper interval bound.
- **la** (BoundAssoc, *optional*) – Bound association of `lb`. Defaults to BoundAssoc.up.
- **ha** (BoundAssoc, *optional*) – Bound association of `hb`. Defaults to BoundAssoc.down.

Returns Whether or not `x` lies in the specified interval.

Return type bool

Examples

```
>>> is_in_range(0.5, 0, 1)
True
>>> is_in_range(0.0, 0, 1, BoundAssoc.up, BoundAssoc.down)
True
>>> is_in_range(0.0, 0, 1, BoundAssoc.down, BoundAssoc.down)
False
>>> is_in_range(1.0, 0, 1, BoundAssoc.down, BoundAssoc.down)
True
>>> is_in_range(1.0, 0, 1, BoundAssoc.down, BoundAssoc.up)
False
```

dexipy.utils.**name_to_id**(*name*, *replace='_'*)

Replaces all non-alphanumeric characters in `name` with `replace`.

Parameters

- **name** (*str*) – Some string.
- **replace** (*str*, *optional*) – Replacement string. Defaults to "_".

Returns Converted string.

Return type str

Example

```
>>> name_to_id("Some #string 1")
'Some__string_1'
```

dexipy.utils.**names_to_ids**(*names*, *replace='_'*)
A vectorised version of *dexipy.utils.name_to_id()*.

Parameters

- **names** (*List[str]*) – List of strings.
- **replace** (*str, optional*) – Replacement string. Defaults to "_".

Returns Converted list of strings, of the same length as `names`.

Return type List[str]

dexipy.utils.**norm_max**(*vals*, *req_max=1.0*)
Normalizes a list of float values so that `max(values) == req_max`.

Parameters

- **vals** (*List[float]*) – A list of values.
- **req_max** (*float, optional*) – The required maximum of the resulting list. Defaults to 1.0.

Returns Normalized list. Returns the original list if `max(vals) == 0`.

Return type List[float]

Examples

```
>>> norm_max([0.1, 0.2, 0.4])
[0.25, 0.5, 1.0]
>>> norm_max([0.1, -0.2, 0.4])
[0.25, -0.5, 1.0]
>>> norm_max([0.1, -0.2, 0.4], 2)
[0.5, -1.0, 2.0]
```

dexipy.utils.**norm_none**(*vals*)
A no-normalization function that can be used in place of other normalization functions.

Parameters **vals** (*List[float]*) – A list of values.

Returns The original list of values. No normalization is performed.

Return type List[float]

dexipy.utils.**norm_sum**(*vals*, *req_sum=1.0*)
Normalizes a list of float values so that `sum(vals) == req_sum`.

Parameters

- **vals** (*List[float]*) – A list of values.
- **req_sum** (*float, optional*) – The required sum of the resulting list. Defaults to 1.0.

Returns Normalized list. Returns the original list if `sum(vals) == 0`.

Return type List[float]

Examples

```
>>> norm_sum([0.1, 0.2, 0.5])
[0.125, 0.25, 0.625]
>>> norm_sum([0.1, -0.2, 0.5])
[0.25, -0.5, 1.25]
>>> norm_sum([0.1, 0.2, 0.5], 2)
[0.25, 0.5, 1.25]
```

dexipy.utils.**objlen**(*obj*)

Returns length of any object type.

Parameters `obj` (*Any*) – An object.

Returns `len(obj)` if object's length is defined, or 0 otherwise.

Return type int

dexipy.utils.**pad_list**(*lst*, *newlen*, *pad*)

Pads a list to the required length, adding `pad` elements if necessary.

Parameters

- `lst` (*List[Any]*) – List of objects of any type.
- `newlen` (*int*) – The required length of the resulting list.
- `pad` (*Any*) – Elements to be added if necessary.

Returns A list obtained from `lst`, padded to the required length.

Return type List[Any]

dexipy.utils.**prod**(*iterable*)

Calculates the product of arguments.

Parameters `iterable` – A sequence of integer or float numbers.

Returns Product of arguments.

Return type int or float

dexipy.utils.**round_float**(*val*, *decimals=None*)

Rounds a float number to the required number of decimals.

Parameters

- `val` (*float*) – An int or float number.
- `decimals` (*Optional[int], optional*) – The required number of decimals. No rounding takes place if None. Defaults to None.

Raises `ValueError` – If the `val` argument is not an integer or float.

Returns Rounded float value.

Return type float

dexipy.utils.**round_float_list**(*values*, *decimals=None*)

Rounds all list elements to the required number of decimals. A vectorised version of `dexipy.util.round_float()`.

Parameters

- `values` (*List[float]*) – List of floats.
- `decimals` (*Optional[int], optional*) – The required number of decimals. No rounding takes place if None. Defaults to None.

Returns A list of rounded values.

Return type List[float]

dexipy.utils.**rule_values**(*vals*, *add=0*)

Convert a DEXi rule values string to a tuple of integers to be used in DEXiPy.

In `.dxi` files, values of decision rules are encoded using character strings, where each individual character encodes some function value. The encoding is zero-based, so that the character `"0"` represents the lowest ordinal number on the corresponding discrete scale.

Parameters

- **vals** (*str*) – A value string as used in `.dxi` files.
- **add** (*int, optional*) – An optional integer value to be added to elements of the resulting tuple. Defaults to 0.

Returns A tuple of integers, of the same length as `vals`.

Return type Tuple[int, ...]

Example

```
>>> rule_values("05A")
(0, 5, 17)
```

dexipy.utils.**set_to_distr**(*valset*, *length=0*)

Converts a set to a value distribution.

Parameters

- **valset** (*Union[int, Set[int]]*) – A value to be converted.
- **length** (*int, optional*) – The required length of the distribution. Defaults to 0.

Returns Set converted to a list of floats. The minimal length of the list is `length`, but it may be extended if `valset` contains elements larger than `length - 1`.

Return type Optional[List[float]]

Examples

```
>>> set_to_distr(1)
[0.0, 1.0]
>>> set_to_distr(1, 5)
[0.0, 1.0, 0.0, 0.0, 0.0]
>>> set_to_distr({1,2}, 4)
[0.0, 1.0, 1.0, 0.0]
```

dexipy.utils.**table_lines**(*columns*, *align=''*, *def_align='l'*)

A general-purpose function for making a table from a list of column strings.

Parameters

- **columns** (*List[List[str]]*) – A list of columns. Each column is a list of strings.
- **align** (*str, optional*) – A string consisting of letters in `("l", "c", "r")` that indicate the justification of the corresponding columns . Defaults to “”.
- **def_align** (*str, optional*) – Default aligning character for columns not specified in `align`. Defaults to `"l"`.

Returns A list of table lines that can be joined for printing.

Return type List[str]

dexipy.utils.**unique_names**(*names*, *reserved=[]*, *start=0*)

Converts a list of strings to a list of unique ID strings.

Parameters

- **names** (*List[str]*) – A list of strings to be converted to IDs.
- **reserved** (*List[str], optional*) – Reserved strings that should not be used as IDs. Defaults to [].
- **start** (*int, optional*) – To make IDs unique, indices of the form `_<int>` are added to the original strings. This argument defines the starting index, which corresponds to the first appearance of some string and is incremented before each subsequent occurence. Defaults to 0.

Returns A list of unique IDs, of the same length as **names**.

Return type List[str]

Examples

```
>>> unique_names(["name", "state", "name", "city", "name", "zip", "zip"])
['name', 'state', 'name_1', 'city', 'name_2', 'zip', 'zip_1']
>>> unique_names(["name", "state", "name", "city", "name", "zip", "zip"],␣
↪reserved = ["name"])
['name_1', 'state', 'name_2', 'city', 'name_3', 'zip', 'zip_1']
```

dexipy.utils.**values_to_str**(*vals*, *add=0*)

Converts numbers to a DEXi string. A reverse operation of *rule_values()*.

Parameters

- **vals** (*Iterable[int]*) – An iterable of integers to be converted to characters.
- **add** (*int, optional*) – An optional integer value to be added to **vals** before conversion. Defaults to 0.

Returns A string of the same length as **vals**.

Return type str

Example

```
>>> values_to_str((0, 5, 17))
'05A'
```

PYTHON MODULE INDEX

d