tfields Documentation

Release 0.4.0

Daniel Böckenhoff

Jul 13, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO *TFIELDS*

Tensors, tensor fields, graphs, mesh manipulation, CAD and more on the basis of numpy.ndarrays. All objects keep track of their coordinate system. Symbolic math operations work for object manipulation.

Licensed under the `MIT License`

1.1 Resources

- Source code: https://gitlab.mpcdf.mpg.de/dboe/tfields
- Documentation: https://tfields.readthedocs.io
- Pypi: https://pypi.python.org/pypi/tfields

1.2 Features

The following features should be highlighted:

- Tensors
- TensorFields
- TensorMaps with fields
- Mesh manipulation by graph theory
- TODO

1.3 Review of alternative/supplementary APIs

- *sympy.diffgeom*: https://docs.sympy.org/latest/modules/diffgeom.html - Symbolic differential algebra implementation. It is not yet covering a broad variety of functionality and is not so well maintained.
- Units: You can find a comparison of modules handling units here: https://socialcompare.com/en/comparison/python-units-quantities-packages - astropy looks best to our understanding but sympy would integrated nicer without an additional package.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install tfields, run this command in your terminal:

```
$ pip install tfields
```

This is the preferred method to install tfields, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for tfields can be downloaded from the remote repo.

You can either clone the public repository:

```
$ git clone git://gitlab.mpcdf.mpg.de/dboe/tfields
```

Or download the tarball:

```
$ curl -OJL https://gitlab.mpcdf.mpg.de/dboe/tfields/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER THREE

USAGE

To use tfields in a project:

```
>>> import tfields
```

CHAPTER

FOUR

API DOCUMENTATION

4.1 tfields package

4.1.1 Subpackages

tfields.bases package

Submodules

tfields.bases.bases module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library Tools for sympy coordinate transformation

`tfields.bases.bases.`**`get_coord_system`**(*base*)

Parameters
base (*str or sympy.diffgeom.get_coord_system*) –

Returns
sympy.diffgeom.get_coord_system

`tfields.bases.bases.`**`get_coord_system_name`**(*base*)

Parameters
base (*str or sympy.diffgeom.get_coord_system*) –

Returns
name of base

Return type
str

`tfields.bases.bases.`**`lambdified_trafo`**(*base_old*, *base_new*)

Parameters

- **base_old** (*sympy.CoordSystem*) –
- **base_new** (*sympy.CoordSystem*) –

Examples

```
>>> import numpy as np
>>> import tfields
```

Transform cartesian to cylinder or spherical >>> a = np.array([[3,4,0]])

```
>>> trafo = tfields.bases.lambdified_trafo(tfields.bases.cartesian,
...                                        tfields.bases.cylinder)
>>> new = np.concatenate([trafo(*coords).T for coords in a])
>>> assert new[0, 0] == 5
```

```
>>> trafo = tfields.bases.lambdified_trafo(tfields.bases.cartesian,
...                                        tfields.bases.spherical)
>>> new = np.concatenate([trafo(*coords).T for coords in a])
>>> assert new[0, 0] == 5
```

`tfields.bases.bases.`**`transform`**(*array*, *base_old*, *base_new*, ***kwargs*)

Transform the input array in place :param array: :type array: np.ndarray :param base_old: :type base_old: str or sympy.CoordSystem :param base_new: :type base_new: str or sympy.CoordSystem

Examples

Cylindrical coordinates >>> import numpy as np >>> import tfields >>> cart = np.array([[0, 0, 0], … [1, 0, 0], … [1, 1, 0], … [0, 1, 0], … [-1, 1, 0], … [-1, 0, 0], … [-1, -1, 0], … [0, -1, 0], … [1, -1, 0], … [0, 0, 1]]) >>> cyl = tfields.bases.transform(cart, 'cartesian', 'cylinder') >>> cyl

Transform cylinder to spherical. No connection is defined so routing via cartesian >>> import numpy as np >>> import tfields >>> b = np.array([[5, np.arctan(4. / 3), 0]]) >>> newB = b.copy() >>> tfields.bases.transform(b, 'cylinder', 'spherical') >>> assert newB[0, 0] == 5 >>> assert round(newB[0, 1], 10) == round(b[0, 1], 10)

tfields.bases.manifold_3 module

`tfields.bases.manifold_3.`**`cartesian_to_cylinder`**(*array*)

`tfields.bases.manifold_3.`**`cartesian_to_spherical`**(*array*)

convert array to r, phi, theta r in [0, oo] phi in [-pi, pi] theta element [0, pi]: elevation angle defined from - Z-axis up

`tfields.bases.manifold_3.`**`cylinder_to_cartesian`**(*array*)

`tfields.bases.manifold_3.`**`physical_cylinder_to_natural_cartesian`**(*vector*, *position=None*)

We need to project from PHYSICAL cylinder coordinates to NATURAL cartesian coordinates

`tfields.bases.manifold_3.`**`spherical_to_cartesian`**(*array*)

Module contents

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library Tools for sympy coordinate transformation

tfields.lib package

Submodules

tfields.lib.decorators module

Function decoration

class tfields.lib.decorators.**cached_property**(*ttl=0*)

Bases: object

Decorator for read-only properties evaluated only once within TTL period.

It can be used to create a cached property like this:

```
import random

# the class containing the property must be a new-style class
class MyClass(object):
    # create property whose value is cached for ten minutes
    @cached_property(ttl=600)
    def randint(self):
        # will only be evaluated every 10 min. at maximum.
        return random.randint(0, 100)
```

The value is cached in the '_cache' attribute of the object instance that has the property getter method wrapped by this decorator. The '_cache' attribute value is a dictionary which has a key for every property of the object which is wrapped by this decorator. Each entry in the cache is created only when the property is accessed for the first time and is a two-element tuple with the last computed property value and the last time it was updated in seconds since the epoch.

The default time-to-live (TTL) is zero seconds. Set the TTL to zero for the cached value to never expire.

To expire a cached property value manually just do:

```
del instance._cache[<property name>]
```

tfields.lib.decorators.**once**(**args*, ***kwargs*)

tfields.lib.decorators.**parametrized**(*dec*)

possibility to parametrize a decorator

tfields.lib.grid module

tfields.lib.grid.**base_vectors**(*array*, *rtol=None*, *atol=None*)

describe the array in terms of base vectors Inverse function of igrid

Examples

```
>>> import tfields
>>> grid = tfields.lib.grid.igrid((3, 5, 5j))
>>> tfields.lib.grid.base_vectors(grid[:, 0])
(3.0, 5.0, 5j)
>>> grid2 = tfields.lib.grid.igrid((3, 5, 5j),
...                                (1, 2, 2j))
>>> grid_circle = tfields.lib.grid.igrid(
...     *tfields.lib.grid.base_vectors(grid2))
>>> assert tfields.Tensors(grid_circle).equal(grid2)
```

tfields.lib.grid.**change_iter_order**(*bv_lengths: List[int]*, *iter_order: List[int]*, *iter_order_new: List[int]*)

Parameters
bv_lengths – lenghts of basis vectors

Returns
indices for changeing fields generated with iter_order to iter_order_new

tfields.lib.grid.**compare_permutations**(*permut1*, *permut2*)

Return what rows you need to switch in order to make permut1 become permut2

Examples

```
>>> import tfields
>>> a = [1, 2, 0, 4, 3]
>>> b = [0, 1, 2, 3, 4]
>>> si = tfields.lib.grid.compare_permutations(a, b)
>>> si
[(0, 2), (1, 2), (3, 4)]
>>> tfields.lib.grid.swap_rows(a, *si)
>>> a
[0, 1, 2, 3, 4]
>>> a == b
True
```

tfields.lib.grid.**ensure_complex**(**base_vectors*) → List[Tuple[float, float, complex]]

Ensure, that the third entry in base_vector of type tuple becomes a complex type. The first two are mapped to float if they they are complex but with imag == 0.

tfields.lib.grid.**igrid**(**base_vectors*, ***kwargs*)

Parameters

- ***base_vectors** (*tuple, list or np.array*) – base vectors spaning the grid behaviour for different input types:

tuple: will be transformed to slices and given to np.mgrid list or np.array: arbitrary base vectors

- ****kwargs** –

  iter_order (list): order in which the iteration will be done.
  Frequency rises with position in list. default is [0, 1, 2] iteration will be done like:

  for v0 in base_vectors[iter_order[0]]:

  for v1 in base_vectors[iter_order[1]]:

  for v2 in base_vectors[iter_order[2]]:
  …

Examples

Initilaize using the mgrid notation >>> import numpy as np >>> import tfields >>> assert np.array_equal(tfields.lib.grid.igrid((0, 1, 2j), … (3, 4, 2j), … (6, 7, 2j)), … [[0., 3., 6.], … [0., 3., 7.], … [0., 4., 6.], … [0., 4., 7.], … [1., 3., 6.], … [1., 3., 7.], … [1., 4., 6.], … [1., 4., 7.]])

```
>>> assert np.array_equal(tfields.lib.grid.igrid([3, 4],
...                                              np.linspace(0, 1, 2),
...                                              (6, 7, 2),
...                                     iter_order=[1, 0, 2]),
...                       [[ 3.,  0.,  6.],
...                        [ 3.,  0.,  7.],
...                        [ 4.,  0.,  6.],
...                        [ 4.,  0.,  7.],
...                        [ 3.,  1.,  6.],
...                        [ 3.,  1.,  7.],
...                        [ 4.,  1.,  6.],
...                        [ 4.,  1.,  7.]])
>>> assert np.array_equal(tfields.lib.grid.igrid(np.linspace(0, 1, 2),
...                                              np.linspace(3, 4, 2),
...                                              np.linspace(6, 7, 2),
...                                              iter_order=[2, 0, 1]),
...                       [[ 0.,  3.,  6.],
...                        [ 0.,  4.,  6.],
...                        [ 1.,  3.,  6.],
...                        [ 1.,  4.,  6.],
...                        [ 0.,  3.,  7.],
...                        [ 0.,  4.,  7.],
...                        [ 1.,  3.,  7.],
...                        [ 1.,  4.,  7.]])
```

`tfields.lib.grid.`**`swap_columns`**(*array*, **index_tuples*)

Parameters

- **array** (*list or array*) –
- **arguments** (*expects tuples with indices to swap as*) –

Examples

```
>>> import numpy as np
>>> import tfields
>>> l = np.array([[3, 2, 1, 0], [6, 5, 4, 3]])
>>> tfields.lib.grid.swap_columns(l, (1, 2), (0, 3))
>>> l
array([[0, 1, 2, 3],
       [3, 4, 5, 6]])
```

`tfields.lib.grid.`**`swap_rows`**`(`*array*, **args*`)`

Parameters

- **array** (*list*) –
- **arguments** (*expects tuples with indices to swap as*) –

Examples

```
>>> import tfields
>>> l = [[3,3,3], [2,2,2], [1,1,1], [0, 0, 0]]
>>> tfields.lib.grid.swap_rows(l, (1, 2), (0, 3))
>>> l
[[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3]]
```

`tfields.lib.grid.`**`to_base_vectors`**`(`**base_vectors*`)`

Transform tuples to arrays with np.mgrid :param tuple of lenght 3 with complex third entry -> start: :param stop: :param n_steps:

Returns

list if np.array for each base

tfields.lib.io module

`tfields.lib.io.`**`bytes_to_numpy`**`(`*serialized_arr: bytearray*`)` → array

Convert back from numpy_to_bytes

`tfields.lib.io.`**`get_module_and_name`**`(`*type_*`)` → Tuple[str, str]

Examples

```
>>> import numpy as np
>>> import tfields
```

This function can be used to ban your type to file as a string and (with the get_type method) get it back. >>> [tfields.lib.io.get_type(***tfields.lib.io.get_module_and_name(**type_**)) … for **type_** in (int, np.ndarray, str)] [<class 'int'>, <class 'numpy.ndarray'>, <class 'str'>]

`tfields.lib.io.`**`get_type`**`(`*module*, *name*`)`

Inverse to **:fun:`get_module_and_name`**

`tfields.lib.io.`**`numpy_to_bytes`**`(`*arr: array*`)` → bytearray

Convert to bytest array

Examples

```
>>> import numpy as np
>>> import tfields
>>> a = np.ones((23, 23), dtype = 'int')
>>> a_b = tfields.lib.io.numpy_to_bytes(a)
>>> a1 = tfields.lib.io.bytes_to_numpy(a_b)
>>> assert np.array_equal(a, a1) and a.shape == a1.shape and a.dtype == a1.dtype
```

tfields.lib.io.**numpy_to_str**(*arr*)

Convert an array to string representation

Examples

```
>>> import numpy as np
>>> import tfields
>>> arr = np.array([[1,2,3], [1,4,5]])
>>> enc = tfields.lib.io.numpy_to_str(arr)
>>> tfields.lib.io.str_to_numpy(enc)
array([[1, 2, 3],
       [1, 4, 5]])
```

tfields.lib.io.**str_to_numpy**(*str_*)

Convert back from numpy_to_str

tfields.lib.sets module

Algorithms around set operations

class tfields.lib.sets.**UnionFind**

Bases: object

Source:

http://code.activestate.com/recipes/215912-union-find-data-structure/

This algorithm and data structure are primarily used for Kruskal's Minimum Spanning Tree algorithm for graphs, but other uses have been found.

The Union Find data structure is not a universal set implementation, but can tell you if two objects are in the same set, in different sets, or you can combine two sets. ufset.find(obja) == ufset.find(objb) ufset.find(obja) != ufset.find(objb) ufset.union(obja, objb)

find(*obj*)

Find the root of the set that an object 'obj' is in. If the object was not known, will make it known, and it becomes its own set. Object must be Python hashable.'"

group_indices(*iterator*)

Return full groups from iterator

groups(*iterator*)

Return full groups from iterator

insert_objects(*objects*)

Insert a sequence of objects into the structure. All must be Python hashable.

union(*object1*, *object2*)

Combine the sets that contain the two objects given. Both objects must be Python hashable. If either or both objects are unknown, will make them known, and combine them.

tfields.lib.sets.**disjoint_group_indices**(*iterator*)

Examples

```
>>> import tfields
>>> tfields.lib.sets.disjoint_group_indices([[0, 0, 0, 'A'], [1, 2, 3],
...                                          [3, 0], [4, 4, 4], [5, 4], ['X', 0.
→42]])
[[0, 1, 2], [3, 4], [5]]
>>> tfields.lib.sets.disjoint_group_indices([[0], [1], [2], [3], [0, 1], [1, 2], [3,
→ 0]])
[[0, 1, 2, 3, 4, 5, 6]]
```

Returns
indices of iterator items grouped in disjoint sets

Return type
list

tfields.lib.sets.**disjoint_groups**(*iterator*)

Disjoint groups implementation

Examples

```
>>> import tfields
>>> tfields.lib.sets.disjoint_groups([[0, 0, 0, 'A'], [1, 2, 3], [3, 0],
...                                   [4, 4, 4], [5, 4], ['X', 0.42]])
[[[0, 0, 0, 'A'], [1, 2, 3], [3, 0]], [[4, 4, 4], [5, 4]], [['X', 0.42]]]
>>> tfields.lib.sets.disjoint_groups([[0], [1], [2], [3], [0, 1], [1, 2], [3, 0]])
[[[0], [1], [2], [3], [0, 1], [1, 2], [3, 0]]]
```

Returns
iterator items grouped in disjoint sets

Return type
list

tfields.lib.sets.**remap**(*arr: ndarray*, *keys: ndarray*, *values: ndarray*, *inplace=False*) → ndarray

Given an input array, remap its entries corresponding to 'keys' to 'values'

Parameters

- **input** – array to remap
- **keys** – values to be replaced
- **values** – values to replace 'keys' with

Returns
like 'input', but with elements remapped according to the mapping defined by 'keys' and 'values'

Return type
output

tfields.lib.stats module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library

`tfields.lib.stats.mode`(*array*, *axis=0*, *bins='auto'*, *range=None*)

generalisation of the scipy.stats.mode function for floats with binning .. rubric:: Examples

Forwarding usage: >>> import tfields # NOQA >>> import numpy as np >>> tfields.lib.stats.mode([[2,2,3], [4,5,3]]) array([[2, 2, 3]]) >>> tfields.lib.stats.mode([[2,2,3], [4,5,3]], axis=1) array([[2],

[3]])

Float usage: >>> np.random.seed(seed=0) # deterministic random >>> n = np.random.normal(3.1, 2., 1000) >>> assert np.isclose(tfields.lib.stats.mode(n), [2.30838613]) >>> assert np.isclose(tfields.lib.stats.mode(n, bins='sturges'), … [2.81321206]) >>> assert np.allclose(tfields.lib.stats.mode(np.array([n, n]), axis=1), … [[2.30838613], … [2.30838613]]) >>> tfields.lib.stats.mode(np.array([n, n]), axis=0).shape (1000, 1) >>> tfields.lib.stats.mode(np.array([n, n]), axis=1).shape (2, 1) >>> assert np.isclose(tfields.lib.stats.mode(np.array([n, n]), … axis=None), … [2.81321206])

`tfields.lib.stats.moment`(*a*, *moment=1*, *axis=0*, *weights=None*, *nan_policy='propagate'*)

Calculate the nth moment about the mean for a sample. A moment is a specific quantitative measure of the shape of a set of points. It is often used to calculate coefficients of skewness and kurtosis due to its close relationship with them. :param a: data :type a: array_like :param moment: order of central moment that is returned. Default is 1. :type moment: int or array_like of ints, optional :param axis: Axis along which the central moment is computed. Default is 0.

If None, compute over the whole array *a*.

Parameters

nan_policy (*{'propagate', 'raise', 'omit'}, optional*) – Defines how to handle when input contains nan. 'propagate' returns nan, 'raise' throws an error, 'omit' performs the calculations ignoring nan values. Default is 'propagate'.

Returns

n-th central moment – The appropriate moment along the given axis or over all values if axis is None. The denominator for the moment calculation is the number of observations, no degrees of freedom correction is done.

Return type

ndarray or float

See also:

`kurtosis`, `skew`, `describe`

Notes

The k-th weighted central moment of a data sample is: .. math:

```
m_k = \frac{1}{\sum_{j = 1}^n w_i} \sum_{i = 1}^n w_i (x_i - \bar{x})^k
```

Where n is the number of samples and x-bar is the mean. This function uses exponentiation by squares[1] for efficiency.

[1] http://eli.thegreenplace.net/2009/03/21/efficient-integer-exponentiation-algorithms

References

Examples

```
>>> from tfields.lib.stats import moment
>>> moment([1, 2, 3, 4, 5], moment=0)
1.0
>>> moment([1, 2, 3, 4, 5], moment=1)
0.0
>>> moment([1, 2, 3, 4, 5], moment=2)
2.0
```

Expansion of the scipy.stats moment function by weights: >>> moment([1, 2, 3, 4, 5], moment=1, weights=[-2, -1, 20, 1, 2]) 0.5

```
>>> moment([1, 2, 3, 4, 5], moment=2, weights=[5, 4, 3, 2, 1])
2.0
>>> moment([1, 2, 3, 4, 5], moment=2, weights=[5, 4, 3, 2, 1])
2.0
>>> assert moment([1, 2, 3, 4, 5], moment=2,
...               weights=[0.25, 1, 17.5, 1, 0.25]) == 0.2
>>> moment([1, 2, 3, 4, 5], moment=2, weights=[0, 0, 1, 0, 0])
0.0
```

tfields.lib.symbolics module

sympy helper functions

`tfields.lib.symbolics.`**`split_expression`**(*expr*)

Return the expression split up in the basic boolean functions.

`tfields.lib.symbolics.`**`to_plane`**(*expr*)

Tranlate the expression (coordinate form) to normal form and return as Plane .. rubric:: Examples

Get 3-d plane for linear equations >>> import sympy >>> from tfields.lib.symbolics import to_plane >>> x, y, z = sympy.symbols('x y z') >>> eq1 = 2*x - 4 >>> p1 = to_plane(eq1) >>> assert eq1, p1.equation()

multiple dimensions work >>> eq2 = x + 2*y + 3*z - 4 >>> p2 = to_plane(eq2) >>> assert eq2, p2.equation()

The base point is calculated independent of the coords >>> eq3 = 2*y + 3*z - 4 >>> p3 = to_plane(eq3) >>> assert eq3, p3.equation()

Inequalities will be treated like equations >>> ie1 = 2*y + 3*z > 4 >>> p4 = to_plane(ie1) >>> assert ie1.lhs - ie1.rhs, p4.equation()

Returns: sympy.Plane

`tfields.lib.symbolics.`**`to_planes`**(*expr*)

Resolve BooleanFunctions to retrieve multiple planes .. rubric:: Examples

```
>>> import sympy
>>> from tfields.lib.symbolics import to_planes, to_plane
>>> x, y, z = sympy.symbols('x y z')
>>> eq1 = 2*x > 0
>>> eq2 = 2*y + 3*z <= 4
```

(continues on next page)

(continued from previous page)

```
>>> p12 = to_planes(eq1 & eq2)
>>> p1 = to_plane(eq1)
>>> p12[0] == p1
True
>>> p2 = to_plane(eq2)
>>> p12[1] == p2
True
```

tfields.lib.util module

Various utility functions

tfields.lib.util.**argsort_unique**(*idx*)

https://stackoverflow.com/a/43411559/ @Divakar

tfields.lib.util.**convert_nan**(*arr*, *value=0.0*)

Replace all occuring NaN values by value

tfields.lib.util.**duplicates**(*arr*, *axis=None*)

View1D version of duplicate search Speed up version after https://stackoverflow.com/questions/46284660 /python-numpy-speed-up-2d-duplicate-search/46294916#46294916

Parameters

- **arr** (*array_like*) – array
- **args** (*other*) – see np.isclose

Examples

```
>>> import tfields
>>> import numpy as np
>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
>>> tfields.lib.util.duplicates(a, axis=0)
array([0, 0, 2])
```

An empty sequence will not throw errors >>> assert np.array_equal(tfields.lib.util.duplicates([], axis=0), [])

Returns

int is pointing to first occurence of unique value

Return type

list of int

tfields.lib.util.**flatten**(*seq*, *container=None*, *keep_types=None*, *key: Optional[Callable] = None*)

Approach to flatten a nested sequence.

Parameters

- **seq** (*iterable*) – iterable to be flattened
- **containter** (*iterable*) – iterable defining an append method. Values will be appended there
- **keep_types** (*list of type*) – types that should not be flattened but kept in nested form
- **key** (*callable*) – callable with the signature key(iterable) -> iterable

Examples

```
>>> from tfields.lib.util import flatten
>>> import numpy as np
>>> flatten([[1,2,3],4,[[5,[6]]]])
[1, 2, 3, 4, 5, 6]
>>> flatten([[1,2,3],4,[[5,[{6:1}]]]], keep_types=[dict])
[1, 2, 3, 4, 5, {6: 1}]
>>> flatten([[1,2,3],4,[[5,[np.array([6])]]]], keep_types=[np.ndarray])
[1, 2, 3, 4, 5, array([6])]
```

Strings work although they have the __iter__ attribute in python3 >>> flatten([[0, 0, 0, 'A'], [1, 2, 3]]) [0, 0, 0, 'A', 1, 2, 3]

Dictionaries will return flattened keys >>> flatten({"a": 1, "b": 2}) ['a', 'b']

You can use the key keyword to specify a transformation on the iterable: >>> flatten({"a": {"a1": 1, "a2": 4}, "b": 2}, key=dict.values) [1, 4, 2]

```
>>> def dict_flat_key(item):
...     if isinstance(item, dict):
...         return item.values()
...     return item
>>> flatten({"a": {"a1": 1, "a2": [3, 4]}, "b": 2}, key=dict_flat_key)
[1, 3, 4, 2]
```

`tfields.lib.util.`**`index`**(*arr*, *entry*, *rtol=0*, *atol=0*, *equal_nan=False*, *axis=None*)

Examples

```
>>> import numpy as np
>>> import tfields
>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
>>> tfields.lib.util.index(a, [2, 3, 4], axis=0)
2
```

```
>>> a = np.array([[1, 0, 0], [2, 3, 4]])
>>> tfields.lib.util.index(a, 4)
5
```

Returns
index of entry in arr

Return type
int

`tfields.lib.util.`**`is_full_slice`**(*index*, *shape*)

Determine if an index is the full slice (i.e. __getitem__ with this index returns the full array) w.r.t the shape given.

Examples

```
>>> import numpy as np
>>> import tfields
>>> class index_getter:
...     def __getitem__(self, index):
...         return index
>>> get_index = index_getter()
>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
>>> shape = a.shape
>>> tfields.lib.util.is_full_slice(get_index[:], shape)
True
>>> tfields.lib.util.is_full_slice(get_index[:, :], shape)
True
>>> tfields.lib.util.is_full_slice(get_index[:, 1], shape)
False
>>> tfields.lib.util.is_full_slice(get_index[1:, :], shape)
False
>>> tfields.lib.util.is_full_slice(get_index[:1, :], shape)
False
>>> tfields.lib.util.is_full_slice(get_index[:, 1:], shape)
False
>>> tfields.lib.util.is_full_slice(get_index[:, :1], shape)
False
>>> tfields.lib.util.is_full_slice(get_index[:, :-1], shape)
True
>>> tfields.lib.util.is_full_slice(get_index[np.array([True, True, True])], shape)
True
>>> tfields.lib.util.is_full_slice(get_index[np.array([True, True, False])], shape)
False
```

`tfields.lib.util.`**`multi_sort`**(*array*, **others*, ***kwargs*)

Sort all given lists parralel with array sorting, ie rearrange the items in the other lists in the same way, you rearrange them for array due to array sorting

Parameters

- **array** (*iterable*) –
- ***others** (*iterable*) –
- ****kwargs** – method (function): sorting function. Default is 'sorted' ...: further arguments are passed to method. Default rest is

  'key=array[0]'

  reversed (bool): wether to reverse the results or not cast_type (type): type of returned iterables

Examples

```
>>> from tfields.lib.util import multi_sort
>>> multi_sort([1,2,3,6,4], [1,2,3,4,5])
([1, 2, 3, 4, 6], [1, 2, 3, 5, 4])
>>> a, b = multi_sort([1,2,3,6,4], [1,2,3,4,5])
>>> b
[1, 2, 3, 5, 4]
```

Expanded to sort as many objects as needed >>> multi_sort([1,2,3,6,4], [1,2,3,4,5], [6,5,4,3,2]) ([1, 2, 3, 4, 6], [1, 2, 3, 5, 4], [6, 5, 4, 2, 3])

Reverse argument >>> multi_sort([1,2,3,6,4], [1,2,3,4,5], [6,5,4,3,2], reverse=True) ([6, 4, 3, 2, 1], [4, 5, 3, 2, 1], [3, 2, 4, 5, 6])

Returns
One iterable for each >>> multi_sort([], [], []) ([], [], []) >>> multi_sort([], [], [], cast_type=tuple) ((), (), ())

Return type
tuple(cast_type)

`tfields.lib.util.`**`pairwise`**(*iterable*)

iterator s -> (s0,s1), (s1,s2), (s2, s3), … Source:
https://stackoverflow.com/questions/5434891/iterate-a-list-as-pair-current-next-in-python

Returns
two iterators, one ahead of the other

`tfields.lib.util.`**`view_1d`**(*arr*)

Delete duplicate columns of the input array https://stackoverflow.com/a/44999009/ @Divakar

Module contents

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

Collection of additional numpy functions part of tfields library

4.1.2 Submodules

4.1.3 tfields.bounding_box module

class `tfields.bounding_box.`**`Node`**(*mesh*, *cuts*, *coord_sys=None*, *at_intersection='split'*, *delta=0.0*, *parent=None*, *box=None*, *internal_template=None*, *cut_expr=None*)

Bases: `object`

This class allows to increase the performance with cuts in x,y and z direction An extension to arbitrary cuts might be possible in the future

Parameters

- **parent** – Parent node of self
- **mesh** – Mesh corresponding to the node
- **cut_expr** – Cut that determines the seperation in left and right node
- **cuts** – List of cuts for the children nodes

Attrs:

parent (Node) remaining_cuts (dict): key specifies dimension, value the cuts that

are still not done

cut_expr (dict): part of parents remaining_cuts. The dimension defines
what is meant by left and right

Examples

```
>>> import tfields
>>> mesh = tfields.Mesh3D.grid((5.6, 6.2, 3),
...                            (-0.25, 0.25, 4),
...                            (-1, 1, 10))
```

```
>>> cuts = {'x': [5.7, 6.1],
...         'y': [-0.2, 0, 0.2],
...         'z': [-0.5, 0.5]}
```

```
>>> tree = tfields.bounding_box.Node(mesh,
...                                  cuts,
...                                  at_intersection='keep')
>>> leaves = tree.leaves()
>>> leaves = tfields.bounding_box.Node.sort_leaves(leaves)
>>> meshes = [leaf.mesh for leaf in leaves]
>>> templates = [leaf.template for leaf in leaves]
>>> special_leaf = tree.find_leaf([5.65, -0.21, 0])
```

find_leaf(*point*, *_in_recursion=False*)

Returns
Node: leaf note, containinig point None: point outside root box

Return type
Node / None

in_box(*point*)

is_last_cut()

is_leaf()

is_root()

leaves()

Recursive function to create a list of all leaves

Returns
of all leaves descending from this node

Return type
list

property root

classmethod sort_leaves(*leaves_list*)

sorting the leaves first in x, then y, then z direction

property template

Get the global template for a leaf. This can be applied to the root mesh with the cut method to retrieve exactly this leaf mesh again.

Returns

mesh with first scalars as an instruction on how to build
this cut (scalars point to faceIndices on mother mesh). Can be used with Mesh3D.cut

Return type
tfields.Mesh3D

class tfields.bounding_box.Searcher(*mesh*, *n_sections=None*, *delta=0.0*, *cut_length=None*)

Bases: *Node*

in_faces(*tensors*, *delta=-1*, *assign_multiple=False*)

TODO-0:

- check case of point+-delta outside box!

Examples

```
>>> import tfields
>>> import numpy as np
>>> mesh = tfields.Mesh3D.grid((0, 1, 2), (1, 2, 2), (2, 3, 2))
>>> tree = tfields.bounding_box.Searcher(mesh)
>>> points = tfields.Tensors([[0.5, 1, 2.1],
...                           [0.5, 0, 0],
...                           [0.5, 2, 2.1],
...                           [0.5, 1.5, 2.5]])
>>> box_res = tree.in_faces(points, delta=0.0001)
>>> usual_res = mesh.in_faces(points, delta=0.0001)
>>> assert np.array_equal(box_res, usual_res)
```

4.1.4 tfields.core module

Author: Daniel Boeckenhoff Mail: dboe@ipp.mpg.de

core of tfields library contains numpy ndarray derived bases of the tfields package

Notes

noqa:E501 pylint:disable=line-too-long, * It could be worthwhile concidering np.li.mixins.NDArrayOperatorsMixin

`class tfields.core.AbstractFields(iterable=(), /)`

Bases: `list`, `AbstractObject`

Extension of the list to tfields polymorphism. Allow setitem and getitem by object name.

`class tfields.core.AbstractNdarray(array, **kwargs)`

Bases: `ndarray`, `AbstractObject`

All tensors and subclasses should derive from AbstractNdarray. AbstractNdarray implements all the inheritance specifics for np.ndarray Whene inheriting, three attributes are of interest:

`__slots__`

If you want to add attributes to your AbstractNdarray subclass, add the attribute name to __slots__

Type
List(str)

`__slot_defaults__`

if __slot_defaults__ is None, the defaults for the attributes in __slots__ will be None other values will be treaded as defaults to the corresponding arg at the same position in the __slots__ list.

Type
list

`__slot_dtypes__`

for the conversion of the args in __slots__ to numpy arrays. None values mean no conversion.

Type
List(dtypes)

`__slot_setters__`

Because __slots__ and properties are mutually exclusive this is a possibility to take care of proper attribute handling. None will be passed for 'not set'.

Type
List(callable)

Parameters

- **array** (*array-like*) – input array
- ****kwargs** – arguments corresponding to __slots__

`property bulk`

The pure ndarray version of the actual state -> nothing attached

`copy(**kwargs)`

The standard ndarray copy does not copy slots. Correct for this.

Examples

```
>>> import tfields
>>> m = tfields.TensorMaps(
...     [[1,2,3], [3,3,3], [0,0,0], [5,6,7]],
...     [[1], [3], [0], [5]],
...     maps=[
...         ([[0, 1, 2], [1, 2, 3]], [21, 42]),
...         [[1]],
...         [[0, 1, 2, 3]]
...     ])
>>> mc = m.copy()
>>> mc.equal(m)
True
>>> mc is m
False
>>> mc.fields is m.fields
False
>>> mc.fields[0] is m.fields[0]
False
>>> mc.maps[3].fields[0] is m.maps[3].fields[0]
False
```

class tfields.core.**AbstractObject**

Bases: Storable

Abstract base class for all tfields objects implementing polymorphisms

class tfields.core.**Container**(**items*, *labels=None*)

Bases: *AbstractFields*

Store lists of tfields objects. Save mechanisms are provided

Examples

```
>>> import numpy as np
>>> import tfields
>>> sphere = tfields.Mesh3D.grid(
...     (1, 1, 1),
...     (-np.pi, np.pi, 3),
...     (-np.pi / 2, np.pi / 2, 3),
...     coord_sys='spherical')
>>> sphere2 = sphere.copy() * 3
>>> c = tfields.Container([sphere, sphere2])
```

```
>>> c.save("~/tmp/spheres.npz")
>>> c1 = tfields.Container.load("~/tmp/spheres.npz")
```

copy()

Return a shallow copy of the list.

property items

items of the container as a list

class tfields.core.**Fields**(**items*)

Bases: *AbstractFields*

Container for fields which should be attached to tensors with the <>.fields attribute to make them tensor fields

static to_field(*field*, *copy=False*, ***kwargs*)

Parameters

- **field** (Tensors) –
- **copy** (*bool*) –
- ****kwargs** – passed to Tensors constructor

class tfields.core.**Maps**(**args*, ***kwargs*)

Bases: SortedDict, *AbstractObject*

Container for TensorFields sorted by dimension, i.e indexing by dimension

Parameters

- **(** (**args*) – List(TensorFields): | List(Tuple(int, TensorFields)): | TensorFields: | Tuple(Tensors, *Fields)): TODO: document
- **)** –
- ****kwargs** – forwarded to SortedDict

TODO: further documentation

equal(*other*, ***kwargs*)

Test equality with other object. :param **kwargs: passed to each item on equality check

static to_map(*map_*, **fields*, *copy=False*, ***kwargs*)

Parameters

- **map** (TensorFields) –
- ***fields** (Tensors) –
- **copy** (*bool*) –
- ****kwargs** – passed to TensorFields constructor

class tfields.core.**TensorFields**(*tensors*, **fields*, ***kwargs*)

Bases: *Tensors*

Discrete Tensor Field

Parameters

- **tensors** (*array*) – base tensors
- ***fields** (*array*) – multiple fields assigned to one base tensor. Fields themself are also of type tensor
- ****kwargs** – rigid (bool): demand equal field and tensor lenght … : see tfields.Tensors

Examples

```
>>> import tfields
>>> from tfields import Tensors, TensorFields
>>> scalars = Tensors([0, 1, 2])
>>> vectors = Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> scalar_field = TensorFields(vectors, scalars)
>>> scalar_field.rank
1
>>> scalar_field.fields[0].rank
0
>>> vectorField = TensorFields(vectors, vectors)
>>> vectorField.fields[0].rank
1
>>> vectorField.fields[0].dim
3
>>> multiField = TensorFields(vectors, scalars, vectors)
>>> multiField.fields[0].dim
1
>>> multiField.fields[1].dim
3
```

Empty initialization

```
>>> empty_field = TensorFields([], dim=3)
>>> assert empty_field.shape == (0, 3)
>>> assert empty_field.fields == []
```

Directly initializing with lists or arrays

```
>>> vec_field_raw = tfields.TensorFields([[0, 1, 2], [3, 4, 5]],
...                                       [1, 6], [2, 7])
>>> assert len(vec_field_raw.fields) == 2
```

Copying

```
>>> cp = TensorFields(vectorField)
>>> assert vectorField.equal(cp)
```

Copying takes care of coord_sys

```
>>> cp.transform(tfields.bases.CYLINDER)
>>> cp_cyl = TensorFields(cp)
>>> assert cp_cyl.coord_sys == tfields.bases.CYLINDER
```

Copying with changing type

```
>>> tcp = TensorFields(vectorField, dtype=int)
>>> assert vectorField.equal(tcp)
>>> assert tcp.dtype == int
```

Raises

- **TypeError** –
- `>>> import tfields` –

- `>>> tfields.TensorFields([1, 2, 3], [3]) # doctest` – +ELLIPSIS
- `Traceback (most recent call last)` –
- `...` –
- `ValueError` – Length of base (3) should be the same as the length of all fields ([1]).
- `This error can be suppressed by setting rigid=False` –
- `>>> loose = tfields.TensorFields([1, 2, 3], [3], rigid=False)` –
- `>>> assert len(loose) != 1` –

coord_sys

equal(*other*, ***kwargs*)

Test, whether the instance has the same content as other.

Parameters

- **other** (*iterable*) –
- ****kwargs** – see Tensors.equal

fields

classmethod merged(**objects*, ***kwargs*)

Factory method Merges all input arguments to one object

Parameters

- **return_templates** (*bool*) – return the templates which can be used together with cut to retrieve the original objects
- **dim** (*int*) –
- ****kwargs** – passed to cls

Examples

```
>>> import numpy as np
>>> import tfields
>>> import tfields.bases
```

The new object with turn out in the most frequent coordinate system if not specified explicitly

```
>>> vec_a = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vec_b = tfields.Tensors([[5, 4, 1]],
...     coord_sys=tfields.bases.cylinder)
>>> vec_c = tfields.Tensors([[4, 2, 3]],
...     coord_sys=tfields.bases.cylinder)
>>> merge = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, [[2, 0, 1]])
>>> assert merge.coord_sys == 'cylinder'
>>> assert merge.equal([[0, 0, 0],
...                     [0, 0, 1],
...                     [1, -np.pi / 2, 0],
...                     [5, 4, 1],
```

(continues on next page)

(continued from previous page)

```
...                     [4, 2, 3],
...                     [2, 0, 1]])
```

Merge also shifts the maps to still refer to the same tensors

```
>>> tm_a = tfields.TensorMaps(merge, maps=[[[0, 1, 2]]])
>>> tm_b = tm_a.copy()
>>> assert tm_a.coord_sys == 'cylinder'
>>> tm_merge = tfields.TensorMaps.merged(tm_a, tm_b)
>>> assert tm_merge.coord_sys == 'cylinder'
>>> assert tm_merge.maps[3].equal([[0, 1, 2],
...                               list(range(len(merge),
...                                          len(merge) + 3,
...                                          1))])
```

```
>>> obj_list = [tfields.Tensors([[1, 2, 3]],
...             coord_sys=tfields.bases.CYLINDER),
...             tfields.Tensors([[3] * 3]),
...             tfields.Tensors([[5, 1, 3]])]
>>> merge2 = tfields.Tensors.merged(
...     *obj_list, coord_sys=tfields.bases.CARTESIAN)
>>> assert merge2.equal([[-0.41614684, 0.90929743, 3.],
...                      [3, 3, 3], [5, 1, 3]], atol=1e-8)
```

The return_templates argument allows to retrieve a template which can be used with the cut method.

```
>>> merge, templates = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, return_templates=True)
>>> assert merge.cut(templates[0]).equal(vec_a)
>>> assert merge.cut(templates[1]).equal(vec_b)
>>> assert merge.cut(templates[2]).equal(vec_c)
```

name

property names

Retrive the names of the fields as a list

Examples

```
>>> import tfields
>>> s = tfields.Tensors([1,2,3], name=1.)
>>> tf = tfields.TensorFields(s, *[s]*10)
>>> assert len(tf.names) == 10
>>> assert set(tf.names) == {1.}
>>> tf.names = range(10)
>>> tf.names
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

plot(**args*, ***kwargs*)

Generic plotting method of TensorFields.

Parameters

- **field_index** – index of the field to plot (as quiver by default)
- **normalize** – If True, normalize the field vectors to show only the direction
- **color** – additional str argument 'norm' added. If color="norm", color with the norm.

transform_field(*coord_sys*, *field_index=0*)

Transform the field to the coordinate system of choice.

NOTE: This is not yet any way generic!!! Have a look at Einsteinpy and actual status of
sympy for further implementation

class tfields.core.**TensorMaps**(*tensors*, **fields*, ***kwargs*)

Bases: *TensorFields*

Parameters

- **tensors** – see Tensors class
- ***fields** (Tensors) – see TensorFields class
- ****kwargs** – coord_sys ('str'): see Tensors class maps (array-like): indices indicating a connection between the

  tensors at the respective index positions

Examples

```
>>> import tfields
>>> scalars = tfields.Tensors([0, 1, 2])
>>> vectors = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> maps = [tfields.TensorFields([[0, 1, 2], [0, 1, 2]], [42, 21]),
...         tfields.TensorFields([[1], [2]], [-42, -21])]
>>> mesh = tfields.TensorMaps(vectors, scalars,
...                           maps=maps)
>>> assert isinstance(mesh.maps, tfields.Maps)
>>> assert len(mesh.maps) == 2
>>> assert mesh.equal(tfields.TensorFields(vectors, scalars))
```

Copy constructor

```
>>> mesh_copy = tfields.TensorMaps(mesh)
```

Copying takes care of coord_sys

```
>>> mesh_copy.transform(tfields.bases.CYLINDER)
>>> mesh_cp_cyl = tfields.TensorMaps(mesh_copy)
>>> assert mesh_cp_cyl.coord_sys == tfields.bases.CYLINDER
```

cleaned(*stale=True*, *duplicates=True*)

Parameters

- **stale** (bool) – remove stale vertices
- **duplicates** (bool) – replace duplicate vertices by originals

Examples

```
>>> import numpy as np
>>> import tfields
>>> mp1 = tfields.TensorFields([[0, 1, 2], [3, 4, 5]],
...                             *zip([1,2,3,4,5], [6,7,8,9,0]))
>>> mp2 = tfields.TensorFields([[0], [3]])
```

```
>>> tm = tfields.TensorMaps([[0,0,0], [1,1,1], [2,2,2], [0,0,0],
...                           [3,3,3], [4,4,4], [5,6,7]],
...                          maps=[mp1, mp2])
```

```
>>> c = tm.cleaned()
>>> assert c.equal([[0., 0., 0.],
...                 [1., 1., 1.],
...                 [2., 2., 2.],
...                 [3., 3., 3.],
...                 [4., 4., 4.]])
>>> assert np.array_equal(c.maps[3], [[0, 1, 2], [0, 3, 4]])
>>> assert np.array_equal(c.maps[1], [[0], [0]])
```

Returns
copy of self without stale vertices and duplicat points (depending on arguments)

coord_sys

disjoint_map(*map_dim*)

Find the disjoint sets of map = self.maps[map_dim] As an example, this method is interesting for splitting a mesh consisting of seperate parts

Parameters
map_dim (*int*) – reference to map position used like: self.maps[map_dim]

Returns
map description(tuple): see self.parts

Return type
Tuple(int, List(List(int)))

Examples

```
>>> import tfields
>>> a = tfields.TensorMaps(
...     [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0]],
...     maps=[[[0, 1, 2], [0, 2, 3]]])
>>> b = a.copy()
```

```
>>> b[:, 0] += 2
>>> m = tfields.TensorMaps.merged(a, b)
>>> mp_description = m.disjoint_map(3)
>>> parts = m.parts(mp_description)
>>> aa, ba = parts
```

(continues on next page)

(continued from previous page)

```
>>> assert aa.maps[3].equal(ba.maps[3])
>>> assert aa.equal(a)
>>> assert ba.equal(b)
```

equal(*other*, ***kwargs*)

Test, whether the instance has the same content as other.

Parameters

- **other** (*iterable*) –
- **optional** – see TensorFields.equal

Examples

```
>>> import tfields
>>> maps = [tfields.TensorFields([[1]], [42])]
>>> tm = tfields.TensorMaps(maps[0], maps=maps)
```

>>> assert tm.equal(tm)

```
>>> cp = tm.copy()
```

>>> assert tm.equal(cp)

```
>>> cp.maps[1].fields[0] = -42
>>> assert tm.maps[1].fields[0] == 42
>>> assert not tm.equal(cp)
```

fields

keep(*keep_condition*)

Return copy of self with vertices where keep_condition is True Copy because self is immutable

Examples

```
>>> import numpy as np
>>> import tfields
>>> m = tfields.TensorMaps(
...     [[0,0,0], [1,1,1], [2,2,2], [0,0,0],
...      [3,3,3], [4,4,4], [5,5,5]],
...     maps=[tfields.TensorFields([[0, 1, 2], [0, 1, 3],
...                                 [3, 4, 5], [3, 4, 1],
...                                 [3, 4, 6]],
...                                 [1, 3, 5, 7, 9],
...                                 [2, 4, 6, 8, 0])])
>>> c = m.removed([True, True, True, False, False, False, False])
>>> c.equal([[0, 0, 0],
...          [3, 3, 3],
...          [4, 4, 4],
...          [5, 5, 5]])
True
```

(continues on next page)

(continued from previous page)

```
>>> assert c.maps[3].equal(np.array([[0, 1, 2], [0, 1, 3]]))
>>> assert c.maps[3].fields[0].equal([5, 9])
>>> assert c.maps[3].fields[1].equal([6, 0])
```

maps

classmethod merged(**objects*, ***kwargs*)

Factory method Merges all input arguments to one object

Parameters

- **return_templates** (*bool*) – return the templates which can be used together with cut to retrieve the original objects
- **dim** (*int*) –
- ****kwargs** – passed to cls

Examples

```
>>> import numpy as np
>>> import tfields
>>> import tfields.bases
```

The new object with turn out in the most frequent coordinate system if not specified explicitly

```
>>> vec_a = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vec_b = tfields.Tensors([[5, 4, 1]],
...     coord_sys=tfields.bases.cylinder)
>>> vec_c = tfields.Tensors([[4, 2, 3]],
...     coord_sys=tfields.bases.cylinder)
>>> merge = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, [[2, 0, 1]])
>>> assert merge.coord_sys == 'cylinder'
>>> assert merge.equal([[0, 0, 0],
...                     [0, 0, 1],
...                     [1, -np.pi / 2, 0],
...                     [5, 4, 1],
...                     [4, 2, 3],
...                     [2, 0, 1]])
```

Merge also shifts the maps to still refer to the same tensors

```
>>> tm_a = tfields.TensorMaps(merge, maps=[[[0, 1, 2]]])
>>> tm_b = tm_a.copy()
>>> assert tm_a.coord_sys == 'cylinder'
>>> tm_merge = tfields.TensorMaps.merged(tm_a, tm_b)
>>> assert tm_merge.coord_sys == 'cylinder'
>>> assert tm_merge.maps[3].equal([[0, 1, 2],
...                               list(range(len(merge),
...                                          len(merge) + 3,
...                                          1))])
```

```
>>> obj_list = [tfields.Tensors([[1, 2, 3]],
...             coord_sys=tfields.bases.CYLINDER),
...             tfields.Tensors([[3] * 3]),
...             tfields.Tensors([[5, 1, 3]])]
>>> merge2 = tfields.Tensors.merged(
...     *obj_list, coord_sys=tfields.bases.CARTESIAN)
>>> assert merge2.equal([[-0.41614684, 0.90929743, 3.],
...                      [3, 3, 3], [5, 1, 3]], atol=1e-8)
```

The return_templates argument allows to retrieve a template which can be used with the cut method.

```
>>> merge, templates = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, return_templates=True)
>>> assert merge.cut(templates[0]).equal(vec_a)
>>> assert merge.cut(templates[1]).equal(vec_b)
>>> assert merge.cut(templates[2]).equal(vec_c)
```

name

parts(**map_descriptions*)

Parameters

***map_descriptions** (*Tuple(int, List(List(int)))*) – tuples of map_dim (int): reference to map position

used like: self.maps[map_dim]

map_indices_list (List(List(int))): each int refers
to index in a map.

Returns

One TensorMaps or TensorMaps subclass per
map_description

Return type
List(cls)

paths(*map_dim*)

Find the minimal amount of graphs building the original graph with maximum of two links per node i.e.

“o—–o o—–o” ” / /” “” / /”” “o–o–o o–o 8–o”

|

= | + +

o o o

/ /

/ /

o o o o

where 8 is a duplicated node (one has two links and one has only one.)

Examples

```
>>> import tfields
```

Ascii figure above: >>> a = tfields.TensorMaps([[1, 0], [3, 0], [2, 2], [0, 4], [2, 4], … [4, 4], [1, 6], [3, 6], [2, 2]], … maps=[[[0, 2], [2, 4], [3, 4], [5, 4], … [1, 8], [6, 4], [6, 7], [7, 4]]])

```
>>> paths = a.paths(2)
>>> assert paths[0].equal([[ 1.,  0.],
...                        [ 2.,  2.],
...                        [ 2.,  4.],
...                        [ 0.,  4.]])
>>> assert paths[0].maps[4].equal([[ 0.,  1.,  2.,  3.]])
>>> assert paths[1].equal([[ 4.,  4.],
...                        [ 2.,  4.],
...                        [ 1.,  6.],
...                        [ 3.,  6.],
...                        [ 2.,  4.]])
>>> assert paths[2].equal([[ 3.,  0.],
...                        [ 2.,  2.]])
```

Note: The Longest path problem is a NP-hard problem.

`plot`(**args*, ***kwargs*)

Generic plotting method of TensorMaps.

Parameters

- ***args** – Depending on Positional arguments passed to the underlying `rna.plotting.plot_tensor_map()` function for arbitrary .
- **dim** (*int*) – dimension of the plot representation (axes).
- **map** (*int*) – index of the map to plot (default is 3).
- **edgecolor** (*color*) – color of the edges (dim = 3)

`removed`(*remove_condition*)

Return copy of self without vertices where remove_condition is True Copy because self is immutable

Examples

```
>>> import tfields
>>> m = tfields.TensorMaps(
...     [[0,0,0], [1,1,1], [2,2,2], [0,0,0],
...      [3,3,3], [4,4,4], [5,5,5]],
...     maps=[tfields.TensorFields([[0, 1, 2], [0, 1, 3],
...           [3, 4, 5], [3, 4, 1],
...           [3, 4, 6]],
...           [1, 3, 5, 7, 9],
...           [2, 4, 6, 8, 0])])
>>> c = m.keep([False, False, False, True, True, True, True])
>>> c.equal([[0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
...           [3, 3, 3],
...           [4, 4, 4],
...           [5, 5, 5]])
True
>>> assert c.maps[3].equal([[0, 1, 2], [0, 1, 3]])
>>> assert c.maps[3].fields[0].equal([5, 9])
>>> assert c.maps[3].fields[1].equal([6, 0])
```

stale()

Returns
Mask for all vertices that are stale i.e. are not refered by maps

Examples

```
>>> import numpy as np
>>> import tfields
>>> vectors = tfields.Tensors(
...     [[0, 0, 0], [0, 0, 1], [0, -1, 0], [4, 4, 4]])
>>> tm = tfields.TensorMaps(
...     vectors,
...     maps=[[[0, 1, 2], [0, 1, 2]], [[1, 1], [2, 2]]])
>>> assert np.array_equal(tm.stale(), [False, False, False, True])
```

class tfields.core.**Tensors**(*tensors*, ***kwargs*)

Bases: *AbstractNdarray*

Set of tensors with the same basis.

Parameters

- **tensors** – np.ndarray or AbstractNdarray subclass
- ****kwargs** – name: optional - custom name, can be anything

Examples

```
>>> import numpy as np
>>> import tfields
```

Initialize a scalar range

```
>>> scalars = tfields.Tensors([0, 1, 2])
>>> scalars.rank == 0
True
```

Initialize vectors

```
>>> vectors = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vectors.rank == 1
True
>>> vectors.dim == 3
```

(continues on next page)

(continued from previous page)

```
True
>>> assert vectors.coord_sys == 'cartesian'
```

Initialize the Levi-Zivita Tensor

```
>>> matrices = tfields.Tensors([[[0, 0, 0], [0, 0, 1], [0, -1, 0]],
...                              [[0, 0, -1], [0, 0, 0], [1, 0, 0]],
...                              [[0, 1, 0], [-1, 0, 0], [0, 0, 0]]])
>>> matrices.shape == (3, 3, 3)
True
>>> matrices.rank == 2
True
>>> matrices.dim == 3
True
```

Initializing in different start coordinate system

```
>>> cyl = tfields.Tensors([[5, np.arctan(4. / 3.), 42]],
...                       coord_sys='cylinder')
>>> assert cyl.coord_sys == 'cylinder'
>>> cyl.transform('cartesian')
>>> assert cyl.coord_sys == 'cartesian'
>>> cart = cyl
>>> assert round(cart[0, 0], 10) == 3.
>>> assert round(cart[0, 1], 10) == 4.
>>> assert cart[0, 2] == 42
```

Initialize with copy constructor keeps the coordinate system

```
>>> with vectors.tmp_transform('cylinder'):
...     vect_cyl = tfields.Tensors(vectors)
...     assert vect_cyl.coord_sys == vectors.coord_sys
>>> assert vect_cyl.coord_sys == 'cylinder'
```

You can demand a special dimension.

```
>>> _ = tfields.Tensors([[1, 2, 3]], dim=3)
>>> _ = tfields.Tensors([[1, 2, 3]], dim=2)
Traceback (most recent call last):
    ...
ValueError: Incorrect dimension: 3 given, 2 demanded.
```

The dimension argument (dim) becomes necessary if you want to initialize an empty array

```
>>> _ = tfields.Tensors([])
Traceback (most recent call last):
    ...
ValueError: Empty tensors need dimension parameter 'dim'.
>>> tfields.Tensors([], dim=7)
Tensors([], shape=(0, 7), dtype=float64)
```

closest(*other*, ***kwargs*)

Parameters

- **other** (Tensors) – closest points to what? -> other
- ****kwargs** – forwarded to scipy.spatial.cKDTree.query

Returns

Indices of other points that are closest to
own points

Return type
array shape(len(self))

Examples

```
>>> import tfields
>>> m = tfields.Tensors([[1,0,0], [0,1,0], [1,1,0], [0,0,1],
...                      [1,0,1]])
>>> p = tfields.Tensors([[1.1,1,0], [0,0.1,1], [1,0,1.1]])
>>> p.closest(m)
array([2, 3, 4])
```

contains(*other*)

Inspired by a speed argument @ stackoverflow.com/questions/14766194/testing-whether-a-numpy-array-contains-a-given-row

Examples

```
>>> import tfields
>>> p = tfields.Tensors([[1,2,3], [4,5,6], [6,7,8]])
>>> p.contains([4,5,6])
True
```

coord_sys

cov_eig(*weights=None*)

Calculate the covariance eigenvectors with lenghts of eigenvalues

Parameters
weights (*np.array | int | None*) – index to scalars to weight with

cut(*expression, coord_sys=None, return_template=False, **kwargs*)

Extract a part of the object according to the logic given by <expression>.

Parameters

- **expression** (*sympy logical expression|tfields.TensorFields*) – logical expression which will be evaluated. use symbols x, y and z. If tfields.TensorFields or subclass is given, the expression refers to a template.
- **coord_sys** (*str*) – coord_sys to evaluate the expression in. Only active for template expression

Examples

```
>>> import tfields
>>> import sympy
>>> x, y, z = sympy.symbols('x y z')
>>> p = tfields.Tensors([[1., 2., 3.], [4., 5., 6.], [1, 2, -6],
...                      [-5, -5, -5], [1,0,-1], [0,1,-1]])
>>> p.cut(x > 0).equal([[1, 2, 3],
...                     [4, 5, 6],
...                     [1, 2, -6],
...                     [1, 0, -1]])
True
```

combinations of cuts

```
>>> cut_expression = (x > 0) & (z < 0)
>>> combi_cut = p.cut(cut_expression)
>>> combi_cut.equal([[1, 2, -6], [1, 0, -1]])
True
```

Templates can be used to speed up the repeated cuts on the same underlying tensor with the same expression but new fields. First let us cut a but request the template on return: >>> field1 = list(range(len(p))) >>> tf = tfields.TensorFields(p, field1) >>> tf_cut, template = tf.cut(cut_expression, … return_template=True)

Now repeat the cut with a new field: >>> field2 = p >>> tf.fields.append(field2) >>> tf_template_cut = tf.cut(template) >>> tf_template_cut.equal(combi_cut) True >>> tf_template_cut.fields[0].equal([2, 4]) True >>> tf_template_cut.fields[1].equal(combi_cut) True

Returns
copy of self with cut applied [optional: template - requires <return_template> switch]

property dim

Manifold dimension

distances(*other*, ***kwargs*)

Parameters

- **other** (*Iterable*) –
- ****kwargs** – … is forwarded to scipy.spatial.distance.cdist

Examples

```
>>> import tfields
>>> p = tfields.Tensors.grid((0, 2, 3j),
...                          (0, 2, 3j),
...                          (0, 0, 1j))
>>> p[4,2] = 1
>>> p.distances(p)[0,0]
0.0
>>> p.distances(p)[5,1]
1.4142135623730951
>>> p.distances([[0,1,2]])[-1][0] == 3
True
```

dot(*b*, *out=None*)

Computes the n-d dot product between self and other defined as in mathematica by summing over the last dimension. When self and b are both one-dimensional vectors, this is just the "usual" dot product; when self and b are 2D matrices, this is matrix multiplication.

Note:

- This is not the same as the numpy.dot function.

Examples

```
>>> import tfields
>>> import numpy as np
```

Scalar product by transposed dot product >>> a = tfields.Tensors([[4, 0, 4]]) >>> b = tfields.Tensors([[10, 0, 0.5]]) >>> c = a.t.dot(b) >>> assert c.equal([42]) >>> assert c.equal(np.dot(a[0], b[0])) >>> assert c.rank == 0

To get the angle between a and b you now just need >>> angle = np.arccos(c)

Matrix vector multiplication >>> a = tfields.Tensors([[[1, 20, 0], [2, 18, 1], [1, 5, 10]]]) >>> b = tfields.Tensors([[1, 2, 3]]) >>> c = a.dot(b) >>> assert c.equal([[41,41,41]])

TODO: generalize dot product to inner

Matrix matrix multiplication can not be done like this. It requires # >>> a = tfields.Tensors([[[1, 8], [2, 4]]]) # >>> b = tfields.Tensors([[[1, 2], [1/2, 1/4]]]) # >>> c = a.dot(b) # >>> c # >>> assert c.equal([[[5, 4], [4, 5]]])

TODO: handle types, fields and maps (which fields etc to choose for the output?)

epsilon_neighbourhood(*epsilon*)

Returns

indices for those sets of points that lie within epsilon around the other

Examples

Create mesh grid with one extra point that will have 8 neighbours within epsilon >>> import tfields >>> p = tfields.Tensors.grid((0, 1, 2j), … (0, 1, 2j), … (0, 1, 2j)) >>> p = tfields.Tensors.merged(p, [[0.5, 0.5, 0.5]]) >>> [len(en) for en in p.epsilon_neighbourhood(0.9)] [2, 2, 2, 2, 2, 2, 2, 2, 9]

equal(*other*, *rtol=None*, *atol=None*, *equal_nan=False*, *return_bool=True*)

Evaluate, whether the instance has the same content as other.

Parameters

- **optional** – rtol (float) atol (float) equal_nan (bool)
- **numpy.isclose** (*see*) –

evalf(*expression=None*, *coord_sys=None*)

Parameters

- **expression** (`sympy logical expression`) –
- **coord_sys** (`str`) – coord_sys to evalfuate the expression in.

Returns

mask of dtype bool with lenght of number of points in
self. This array is True, where expression evalfuates True.

Return type
np.ndarray

Examples

```
>>> import tfields
>>> import numpy as np
>>> import sympy
>>> x, y, z = sympy.symbols('x y z')
>>> p = tfields.Tensors([[1., 2., 3.], [4., 5., 6.], [1, 2, -6],
...                      [-5, -5, -5], [1,0,-1], [0,1,-1]])
>>> np.array_equal(p.evalf(x > 0),
...                [True, True, True, False, True, False])
True
>>> np.array_equal(p.evalf(x >= 0),
...                [True, True, True, False, True, True])
True
```

And combination

```
>>> np.array_equal(p.evalf((x > 0) & (y < 3)),
...                [True, False, True, False, True, False])
True
```

Or combination

```
>>> np.array_equal(p.evalf((x > 0) | (y > 3)),
...                [True, True, True, False, True, False])
True
```

classmethod grid(**base_vectors*, ***kwargs*)

Parameters

- ***base_vectors** (*Iterable*) – base coordinates. The amount of base vectors defines the dimension
- ****kwargs** –

  iter_order (list): order in which the iteration will be done.
  Frequency rises with position in list. default is [0, 1, 2] iteration will be done like:

  for v0 in base_vectors[iter_order[0]]:

  for v1 in base_vectors[iter_order[1]]:

  for v2 in base_vectors[iter_order[2]]:
  coords0.append(locals()['v%i' % iter_order[0]]) coords1.append(locals()['v%i' % iter_order[1]]) coords2.append(locals()['v%i' % iter_order[2]])

Examples

Initilaize using the mgrid notation

```
>>> import numpy as np
>>> import tfields
>>> mgrid = tfields.Tensors.grid((0, 1, 2j), (3, 4, 2j), (6, 7, 2j))
>>> mgrid.equal([[0, 3, 6],
...              [0, 3, 7],
...              [0, 4, 6],
...              [0, 4, 7],
...              [1, 3, 6],
...              [1, 3, 7],
...              [1, 4, 6],
...              [1, 4, 7]])
True
```

Lists or arrays are accepted also. Furthermore, the iteration order can be changed

```
>>> lins = tfields.Tensors.grid(
...     np.linspace(3, 4, 2), np.linspace(0, 1, 2),
...     np.linspace(6, 7, 2), iter_order=[1, 0, 2])
>>> lins.equal([[3, 0, 6],
...             [3, 0, 7],
...             [4, 0, 6],
...             [4, 0, 7],
...             [3, 1, 6],
...             [3, 1, 7],
...             [4, 1, 6],
...             [4, 1, 7]])
True
>>> lins2 = tfields.Tensors.grid(np.linspace(0, 1, 2),
...                              np.linspace(3, 4, 2),
...                              np.linspace(6, 7, 2),
...                              iter_order=[2, 0, 1])
>>> lins2.equal([[0, 3, 6],
...              [0, 4, 6],
...              [1, 3, 6],
...              [1, 4, 6],
...              [0, 3, 7],
...              [0, 4, 7],
...              [1, 3, 7],
...              [1, 4, 7]])
True
```

When given the coord_sys argument, the grid is performed in the given coorinate system:

```
>>> lins3 = tfields.Tensors.grid(np.linspace(4, 9, 2),
...                              np.linspace(np.pi/2, np.pi/2, 1),
...                              np.linspace(4, 4, 1),
...                              iter_order=[2, 0, 1],
...                              coord_sys=tfields.bases.CYLINDER)
>>> assert lins3.coord_sys == 'cylinder'
>>> lins3.transform('cartesian')
```

(continues on next page)

(continued from previous page)

```
>>> assert np.array_equal(lins3[:, 1], [4, 9])
```

index(*tensor*, ***kwargs*)

Parameters
tensor –

Returns
index of tensor occuring

Return type
int

indices(*tensor*, *rtol=None*, *atol=None*)

Returns
indices of tensor occuring

Return type
list of int

Examples

Rank 1 Tensors

```
>>> import tfields
>>> p = tfields.Tensors([[1,2,3], [4,5,6], [6,7,8], [4,5,6],
...                      [4.1, 5, 6]])
>>> p.indices([4,5,6])
array([1, 3])
>>> p.indices([4,5,6.1], rtol=1e-5, atol=1e-1)
array([1, 3, 4])
```

Rank 0 Tensors

```
>>> p = tfields.Tensors([2, 3, 6, 3.01])
>>> p.indices(3)
array([1])
>>> p.indices(3, rtol=1e-5, atol=1e-1)
array([1, 3])
```

main_axes(*weights=None*)

Returns
Main Axes eigen-vectors

classmethod merged(**objects*, ***kwargs*)

Factory method Merges all input arguments to one object

Parameters

- **return_templates** (*bool*) – return the templates which can be used together with cut to retrieve the original objects
- **dim** (*int*) –
- ****kwargs** – passed to cls

Examples

```
>>> import numpy as np
>>> import tfields
>>> import tfields.bases
```

The new object with turn out in the most frequent coordinate system if not specified explicitly

```
>>> vec_a = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vec_b = tfields.Tensors([[5, 4, 1]],
...     coord_sys=tfields.bases.cylinder)
>>> vec_c = tfields.Tensors([[4, 2, 3]],
...     coord_sys=tfields.bases.cylinder)
>>> merge = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, [[2, 0, 1]])
>>> assert merge.coord_sys == 'cylinder'
>>> assert merge.equal([[0, 0, 0],
...                     [0, 0, 1],
...                     [1, -np.pi / 2, 0],
...                     [5, 4, 1],
...                     [4, 2, 3],
...                     [2, 0, 1]])
```

Merge also shifts the maps to still refer to the same tensors

```
>>> tm_a = tfields.TensorMaps(merge, maps=[[[0, 1, 2]]])
>>> tm_b = tm_a.copy()
>>> assert tm_a.coord_sys == 'cylinder'
>>> tm_merge = tfields.TensorMaps.merged(tm_a, tm_b)
>>> assert tm_merge.coord_sys == 'cylinder'
>>> assert tm_merge.maps[3].equal([[0, 1, 2],
...                               list(range(len(merge),
...                                          len(merge) + 3,
...                                          1))])
```

```
>>> obj_list = [tfields.Tensors([[1, 2, 3]],
...             coord_sys=tfields.bases.CYLINDER),
...             tfields.Tensors([[3] * 3]),
...             tfields.Tensors([[5, 1, 3]])]
>>> merge2 = tfields.Tensors.merged(
...     *obj_list, coord_sys=tfields.bases.CARTESIAN)
>>> assert merge2.equal([[-0.41614684, 0.90929743, 3.],
...                      [3, 3, 3], [5, 1, 3]], atol=1e-8)
```

The return_templates argument allows to retrieve a template which can be used with the cut method.

```
>>> merge, templates = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, return_templates=True)
>>> assert merge.cut(templates[0]).equal(vec_a)
>>> assert merge.cut(templates[1]).equal(vec_b)
>>> assert merge.cut(templates[2]).equal(vec_c)
```

min_dists(*other=None*, ***kwargs*)

Parameters

- **other** (*array | None*) – if None: closest distance to self
- ****kwargs** –

  memory_saving (bool): for very large array comparisons
  default False

  … rest is forwarded to scipy.spatial.distance.cdist

Returns
minimal distances of self to other

Return type
np.array

Examples

```
>>> import tfields
>>> import numpy as np
>>> p = tfields.Tensors.grid((0, 2, 3),
...                          (0, 2, 3),
...                          (0, 0, 1))
>>> p[4,2] = 1
>>> dMin = p.min_dists()
>>> expected = [1] * 9
>>> expected[4] = np.sqrt(2)
>>> np.array_equal(dMin, expected)
True
```

```
>>> dMin2 = p.min_dists(memory_saving=True)
>>> bool((dMin2 == dMin).all())
True
```

mirror(*coordinate*, *condition=None*)

Reflect/Mirror the entries meeting <condition> at <coordinate> = 0

Parameters
coordinate (*int*) – coordinate index

Examples

```
>>> import tfields
>>> p = tfields.Tensors([[1., 2., 3.], [4., 5., 6.], [1, 2, -6]])
>>> p.mirror(1)
>>> assert p.equal([[1, -2, 3], [4, -5,  6], [1, -2, -6]])
```

multiple coordinates can be mirrored at the same time i.e. a point mirrorion would be

```
>>> p = tfields.Tensors([[1., 2., 3.], [4., 5., 6.], [1, 2, -6]])
>>> p.mirror([0,2])
>>> assert p.equal([[-1, 2, -3], [-4, 5, -6], [-1, 2., 6.]])
```

You can give a condition as mask or as str. The mirroring will only be applied to the points meeting the condition.

```
>>> import sympy
>>> x, y, z = sympy.symbols('x y z')
>>> p.mirror([0, 2], y > 3)
>>> p.equal([[-1, 2, -3], [4, 5, 6], [-1, 2, 6]])
True
```

moment(*moment*, *weights=None*)

Returns
Moments of the distribution.

Parameters
moment (*int*) – n-th moment

Examples

```
>>> import tfields
```

Skalars

```
>>> t = tfields.Tensors(range(1, 6))
>>> assert t.moment(1) == 0
>>> assert t.moment(1, weights=[-2, -1, 20, 1, 2]) == 0.5
>>> assert t.moment(2, weights=[0.25, 1, 17.5, 1, 0.25]) == 0.2
```

Vectors

```
>>> t = tfields.Tensors(list(zip(range(1, 6), range(1, 6))))
>>> assert tfields.Tensors([0.5, 0.5]).equal(
...     t.moment(1, weights=[-2, -1, 20, 1, 2]))
>>> assert tfields.Tensors([1. , 0.5]).equal(
...     t.moment(1, weights=list(zip([-2, -1, 10, 1, 2],
...                                  [-2, -1, 20, 1, 2]))))
```

name

norm(*ord=None*, *axis=None*, *keepdims=False*)

Calculate the norm up to rank 2

Parameters

- **axis** (*See numpy.linal.norm except redefinition in*) –
- **axis** – by default omitting first axis

Examples

```
>>> import tfields
>>> a = tfields.Tensors([[1, 0, 0]])
>>> assert a.norm().equal([1])
```

normalized(**args*, ***kwargs*)

Return the self / norm(self)

Parameters
to (*forwarded*) – meth:norm

Examples

```
>>> import tfields
>>> a = tfields.Tensors([[1, 4, 3]])
>>> assert not a.norm().equal([1])
>>> a = a.normalized()
>>> assert a.norm().equal([1])
```

```
>>> a = tfields.Tensors([[1, 0, 0],
...                      [0, 2, 0],
...                      [0, 0, 3]])
>>> assert a.norm().equal([1, 2, 3])
>>> a = a.normalized()
>>> assert a.equal([
...     [1, 0, 0],
...     [0, 1, 0],
...     [0, 0, 1],
... ])
>>> assert a.norm().equal([1, 1, 1])
```

plot(**args*, ***kwargs*)

Generic plotting method of Tensors.

Forwarding to rna.plotting.plot_tensor

property rank

Tensor rank

property t

Same as self.T but for tensor dimension only. Keeping the order of stacked tensors.

Examples

```
>>> import tfields
>>> a = tfields.Tensors([[[1,2,3,4],[5,6,7,8]]])
>>> assert a.t.equal([a[0].T])
```

tmp_transform(*coord_sys*)

Temporarily change the coord_sys to another coord_sys and change it back at exit This method is for cleaner code only. No speed improvements go with this.

Parameters

transform (*see*) –

Examples

```
>>> import tfields
>>> p = tfields.Tensors([[1,2,3]], coord_sys=tfields.bases.SPHERICAL)
>>> with p.tmp_transform(tfields.bases.CYLINDER):
...     assert p.coord_sys == tfields.bases.CYLINDER
>>> assert p.coord_sys == tfields.bases.SPHERICAL
```

to_segment(*segment*, *num_segments*, *coordinate*, *periodicity=6.283185307179586*, *offset=0.0*, *coord_sys=None*)

For circular (close into themself after <periodicity>) coordinates at index <coordinate> assume <num_segments> segments and transform all values to segment number <segment>

Parameters

- **segment** (*int*) – segment index (starting at 0)
- **num_segments** (*int*) – number of segments
- **coordinate** (*int*) – coordinate index
- **periodicity** (*float*) – after what lenght, the coordiante repeats
- **offset** (*float*) – offset in the mapping
- **coord_sys** (*str or sympy.CoordinateSystem*) – in which coord sys the transformation should be done

Examples

```
>>> import tfields
>>> import numpy as np
>>> pStart = tfields.Points3D([[6, 2 * np.pi, 1],
...                            [6, 2 * np.pi / 5 * 3, 1]],
...                           coord_sys='cylinder')
>>> p = tfields.Points3D(pStart)
>>> p.to_segment(0, 5, 1, offset=-2 * np.pi / 10)
>>> assert np.array_equal(p[:, 1], [0, 0])
```

```
>>> p2 = tfields.Points3D(pStart)
>>> p2.to_segment(1, 5, 1, offset=-2 * np.pi / 10)
>>> assert np.array_equal(np.round(p2[:, 1], 4), [1.2566] * 2)
```

transform(*coord_sys*, ***kwargs*)

Parameters

coord_sys (*str*) –

Examples

```
>>> import numpy as np
>>> import tfields
```

CARTESIAN to SPHERICAL >>> t = tfields.Tensors([[1, 2, 2], [1, 0, 0], [0, 0, -1], … [0, 0, 1], [0, 0, 0]]) >>> t.transform('spherical')

r

```
>>> assert t[0, 0] == 3
```

phi

```
>>> assert t[1, 1] == 0.
>>> assert t[2, 1] == 0.
```

theta is 0 at (0, 0, 1) and pi / 2 at (0, 0, -1)

```
>>> assert round(t[1, 2], 10) == round(0, 10)
>>> assert t[2, 2] == -np.pi / 2
>>> assert t[3, 2] == np.pi / 2
```

theta is defined 0 for R == 0

```
>>> assert t[4, 0] == 0.
>>> assert t[4, 2] == 0.
```

CARTESIAN to CYLINDER

```
>>> tCart = tfields.Tensors([[3, 4, 42], [1, 0, 0], [0, 1, -1],
...                          [-1, 0, 1], [0, 0, 0]])
>>> t_cyl = tCart.copy()
>>> t_cyl.transform('cylinder')
>>> assert t_cyl.coord_sys == 'cylinder'
```

R

```
>>> assert t_cyl[0, 0] == 5
>>> assert t_cyl[1, 0] == 1
>>> assert t_cyl[2, 0] == 1
>>> assert t_cyl[4, 0] == 0
```

Phi

```
>>> assert round(t_cyl[0, 1], 10) == round(np.arctan(4. / 3), 10)
>>> assert t_cyl[1, 1] == 0
>>> assert round(t_cyl[2, 1], 10) == round(np.pi / 2, 10)
>>> assert t_cyl[1, 1] == 0
```

Z

```
>>> assert t_cyl[0, 2] == 42
>>> assert t_cyl[2, 2] == -1
```

```
>>> t_cyl.transform('cartesian')
>>> assert t_cyl.coord_sys == 'cartesian'
>>> assert round(t_cyl[0, 0], 10) == 3
```

`tfields.core.as_fields`(*fields*)

Setter for TensorFields.fields Copies input .. rubric:: Examples

```
>>> import tfields
>>> scalars = tfields.Tensors([0, 1, 2])
>>> vectors = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> maps = [tfields.TensorFields([[0, 1, 2], [0, 1, 2]]),
...         tfields.TensorFields([[1], [2]], [-42, -21])]
>>> mesh = tfields.TensorMaps(vectors, scalars,
...                           maps=maps)
>>> mesh.maps[3].fields = [[42, 21]]
>>> assert len(mesh.maps[3].fields) == 1
>>> assert mesh.maps[3].fields[0].equal([42, 21])
```

`tfields.core.as_maps`(*maps*)

Setter for TensorMaps.maps Copies input

`tfields.core.dim`(*tensor*)

Manifold dimension

`tfields.core.rank`(*tensor*)

Tensor rank

4.1.5 tfields.mask module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library contains interaction methods for sympy and numpy

`tfields.mask.evalf`(*array*, *cut_expression=None*, *coords=None*)

Linking sympy and numpy by retrieving a mask according to the cut_expression

Parameters

- **array** (*numpy ndarray*) –
- **cut_expression** (*sympy logical expression*) –
- **coord_sys** (*str*) – coord_sys to evalfuate the expression in.

Returns

mask which is True, where cut_expression evalfuates True.

Return type

np.array

Examples

```
>>> import sympy
>>> import numpy as np
>>> import tfields
>>> x, y, z = sympy.symbols('x y z')
```

```
>>> a = np.array([[1., 2., 3.], [4., 5., 6.], [1, 2, -6],
...               [-5, -5, -5], [1,0,-1], [0,1,-1]])
>>> assert np.array_equal(
...     tfields.evalf(a, x > 0),
...     np.array([ True, True, True, False, True, False]))
```

And combination >>> assert np.array_equal(… tfields.evalf(a, (x > 0) & (y < 3)), … np.array([True, False, True, False, True, False]))

Or combination >>> assert np.array_equal(… tfields.evalf(a, (x > 0) | (y > 3)), … np.array([True, True, True, False, True, False]))

If array of other shape than (?, 3) is given, the coords need to be specified >>> a0, a1 = sympy.symbols('a0 a1') >>> assert np.array_equal(… tfields.evalf([[0., 1.], [-1, 3]], a1 > 2, coords=[a0, a1]), … np.array([False, True], dtype=bool))

>= is taken care of >>> assert np.array_equal(… tfields.evalf(a, x >= 0), … np.array([True, True, True, False, True, True]))

4.1.6 tfields.mesh_3d module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

Triangulated mesh class and methods

class tfields.mesh_3d.**Mesh3D**(*tensors*, **fields*, ***kwargs*)

Bases: *TensorMaps*

Points3D child used as vertices combined with faces to build a geometrical mesh of triangles .. rubric:: Examples

```
>>> import tfields
>>> import numpy as np
>>> m = tfields.Mesh3D([[1,2,3], [3,3,3], [0,0,0], [5,6,7]], faces=[[0, 1, 2], [1,
↪2, 3]])
>>> m.equal([[1, 2, 3],
...          [3, 3, 3],
...          [0, 0, 0],
...          [5, 6, 7]])
True
>>> np.array_equal(m.faces, [[0, 1, 2], [1, 2, 3]])
True
```

conversion to points only >>> tfields.Points3D(m).equal([[1, 2, 3], … [3, 3, 3], … [0, 0, 0], … [5, 6, 7]]) True

Empty instances >>> m = tfields.Mesh3D([]);

going from Mesh3D to Triangles3D instance is easy and will be cached. >>> m = tfields.Mesh3D([[1,0,0], [0,1,0], [0,0,0]], faces=[[0, 1, 2]]); >>> assert m.triangles().equal(tfields.Triangles3D([[1., 0., 0.], … [0., 1., 0.], … [0., 0., 0.]]))

a list of scalars is assigned to each face >>> mScalar = tfields.Mesh3D([[1,0,0], [0,1,0], [0,0,0]], … faces=([[0, 1, 2]], [.5])); >>> np.array_equal(mScalar.faces.fields, [[0.5]]) True

adding together two meshes: >>> m2 = tfields.Mesh3D([[1,0,0],[2,0,0],[0,3,0]], … faces=([[0,1,2]], [.7])) >>> msum = tfields.Mesh3D.merged(mScalar, m2) >>> msum.equal([[1., 0., 0.], … [0., 1., 0.], … [0., 0., 0.], … [1., 0., 0.], … [2., 0., 0.], … [0., 3., 0.]]) True >>> assert np.array_equal(msum.faces, [[0, 1, 2], [3, 4, 5]])

Saving and reading >>> from tempfile import NamedTemporaryFile >>> outFile = NamedTemporaryFile(suffix='.npz') >>> m.save(outFile.name) >>> _ = outFile.seek(0) >>> m1 = tfields.Mesh3D.load(outFile.name, allow_pickle=True) >>> bool(np.all(m == m1)) True >>> assert np.array_equal(m1.faces, np.array([[0, 1, 2]]))

centroids()

coord_sys

cut(**args*, ***kwargs*)

cut method for Mesh3D. :param expression:

sympy locical expression: Sympy expression that defines planes
in 3D

Mesh3D: A mesh3D will be interpreted as a template, i.e. a
fast instruction of how to cut the triangles. It is the second part of the tuple, returned by a previous cut with a sympy locial expression with 'return_template=True'. We use the vertices and maps of the Mesh as the skelleton of the returned mesh. The fields are mapped according to indices in the template.maps[i].fields.

Parameters

- **coord_sys** (*coordinate system to cut in*) –
- **at_intersection** (*str*) – instruction on what to do, when a cut will intersect a triangle. Options: 'remove' (Default) - remove the faces that are on the edge

  'keep' - keep the faces that are on the edge 'split' - Create new triangles that make up the old one.
- **return_template** (*bool*) – If True: return the template to redo the same cut fast

Examples

define the cut >>> import numpy as np >>> import tfields >>> from sympy.abc import x,y,z >>> cut_expr = x > 1.5

```
>>> m = tfields.Mesh3D.grid((0, 3, 4),
...                         (0, 3, 4),
...                         (0, 0, 1))
>>> m.fields.append(tfields.Tensors(np.linspace(0, len(m) - 1,
...                                             len(m))))
>>> m.maps[3].fields.append(
...     tfields.Tensors(np.linspace(0,
...                                 len(m.maps[3]) - 1,
...                                 len(m.maps[3]))))
>>> mNew = m.cut(cut_expr)
>>> len(mNew)
8
>>> mNew.nfaces()
```

(continues on next page)

(continued from previous page)

```
6
>>> float(mNew[:, 0].min())
2.0
```

Cutting with the 'keep' option will leave triangles on the edge untouched: >>> m_keep = m.cut(cut_expr, at_intersection='keep') >>> float(m_keep[:, 0].min()) 1.0 >>> m_keep.nfaces() 12

Cutting with the 'split' option will create new triangles on the edge: >>> m_split = m.cut(cut_expr, at_intersection='split') >>> float(m_split[:, 0].min()) 1.5 >>> len(m_split) 15 >>> m_split.nfaces() 15

Cut with 'return_template=True' will return the exact same mesh but additionally an instruction to conduct the exact same cut fast (template) >>> m_split_2, template = m.cut(cut_expr, at_intersection='split', … return_template=True) >>> m_split_template = m.cut(template) >>> assert m_split.equal(m_split_2, equal_nan=True) >>> assert m_split.equal(m_split_template, equal_nan=True) >>> assert len(template.fields) == 1 >>> assert len(m_split.fields) == 1 >>> assert len(m_split_template.fields) == 1 >>> assert m_split.fields[0].equal(… list(range(8, 16)) + [np.nan] * 7, equal_nan=True) >>> assert m_split_template.fields[0].equal(… list(range(8, 16)) + [np.nan] * 7, equal_nan=True)

This seems irrelevant at first but consider, the map field or the tensor field changes: >>> m_altered_fields = m.copy() >>> m_altered_fields[0] += 42 >>> assert not m_split.equal(m_altered_fields.cut(template)) >>> assert tfields.Tensors(m_split).equal(… m_altered_fields.cut(template)) >>> assert tfields.Tensors(m_split.maps[3]).equal(… m_altered_fields.cut(template).maps[3])

The cut expression may be a sympy.BooleanFunction: >>> cut_expr_bool_fun = (x > 1.5) & (y < 1.5) & (y >0.2) & (z > -0.5) >>> m_split_bool = m.cut(cut_expr_bool_fun, … at_intersection='split')

Returns

copy of cut mesh * optional: template

disjoint_parts(*return_template=False*)

Returns

disjoint_parts(List(cls)), templates(List(cls))

```
>>> import tfields
>>> a = tfields.Mesh3D(
...     [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0]],
...     maps=[[[0, 1, 2], [0, 2, 3]]])
>>> b = a.copy()
```

```
>>> b[:, 0] += 2
>>> m = tfields.Mesh3D.merged(a, b)
>>> parts = m.disjoint_parts()
>>> aa, ba = parts
>>> assert aa.maps[3].equal(ba.maps[3])
>>> assert aa.equal(a)
>>> assert ba.equal(b)
```

property faces

fields

classmethod grid(**base_vectors*, ***kwargs*)

Construct 'cuboid' along base_vectors .. rubric:: Examples

Building symmetric geometries were never as easy:

Approximated sphere with radius 1, translated in y by 2 units >>> import numpy as np >>> import tfields >>> sphere = tfields.Mesh3D.grid((1, 1, 1), … (-np.pi, np.pi, 12), … (-np.pi / 2, np.pi / 2, 12), … coord_sys='spherical') >>> sphere.transform('cartesian') >>> sphere[:, 1] += 2

Oktaeder >>> oktaeder = tfields.Mesh3D.grid((1, 1, 1), … (-np.pi, np.pi, 5), … (-np.pi / 2, np.pi / 2, 3), … coord_sys='spherical') >>> oktaeder.transform('cartesian')

Cube with edge length of 2 units >>> cube = tfields.Mesh3D.grid((-1, 1, 2), … (-1, 1, 2), … (-5, -3, 2))

Cylinder >>> cylinder = tfields.Mesh3D.grid((1, 1, 1), … (-np.pi, np.pi, 12), … (-5, 3, 12), … coord_sys='cylinder') >>> cylinder.transform('cartesian')

`in_faces`(*points*, *delta*, ***kwargs*)

Check whether points lie within triangles with Barycentric Technique see Triangles3D.in_triangles. If multiple requests are done on huge meshes, this can be hugely optimized by requesting the property self.tree or setting it to self.tree = <saved tree> before calling in_faces.

`maps`

`name`

`nfaces`()

`classmethod plane`(**base_vectors*, ***kwargs*)

Alternative constructor for creating a plane from :param *base_vectors: see grid constructors in core. One base_vector has
to be one-dimensional

Parameters
****kwargs** – forwarded to __new__

`planes`()

`project`(*tensor_field*, *delta=None*, *merge_functions=None*, *point_face_assignment=None*, *return_point_face_assignment=False*)

Project the points of the tensor_field to a copy of the mesh and set the face values accord to the field to the maps field. If no field is present in tensor_field, the number of points in a mesh is counted.

Parameters

- **tensor_field** (Tensors | TensorFields) –
- **delta** (*float | None*) – forwarded to Mesh3D.in_faces
- **merge_functions** (*callable*) – if multiple Tensors lie in the same face, they are mapped with the merge_function to one value
- **point_face_assignment** (*np.array, dtype=int*) – array assigning each point to a face. Each entry position corresponds to a point of the tensor, each entry value is the index of the assigned face
- **return_point_face_assignment** (*bool*) – if true, return the point_face_assignment

Examples

```
>>> import tfields
>>> import numpy as np
>>> mp = tfields.TensorFields([[0,1,2],[2,3,0],[3,2,5],[5,4,3]],
...                           [1, 2, 3, 4])
>>> m = tfields.Mesh3D([[0,0,0], [1,0,0], [1,1,0], [0,1,0], [0,2,0], [1,2,0]],
...                    maps=[mp])
>>> points = tfields.Tensors([[0.5, 0.2, 0.0],
...                           [0.5, 0.02, 0.0],
...                           [0.5, 0.8, 0.0],
...                           [0.5, 0.8, 0.1]])  # not contained
```

Projecting points onto the mesh gives the count >>> m_points = m.project(points, delta=0.01) >>> list(m_points.maps[3].fields[0]) [2, 1, 0, 0]

TensorFields with arbitrary size are projected, combinging the fields automatically >>> fields = [tfields.Tensors([1,3,42, -1]), … tfields.Tensors([[0,1,2], [2,3,4], [3,4,5], [-1] * 3]), … tfields.Tensors([[[0, 0]] * 2, … [[2, 2]] * 2, … [[3, 3]] * 2, … [[9, 9]] * 2])] >>> tf = tfields.TensorFields(points, *fields) >>> m_tf = m.project(tf, delta=0.01) >>> assert m_tf.maps[3].fields[0].equal([2, 42, np.nan, np.nan], equal_nan=True) >>> assert m_tf.maps[3].fields[1].equal([[1, 2, 3], … [3, 4, 5], … [np.nan] * 3, … [np.nan] * 3], … equal_nan=True) >>> assert m_tf.maps[3].fields[2].equal([[[1, 1]] * 2, … [[3, 3]] * 2, … [[np.nan, np.nan]] * 2, … [[np.nan, np.nan]] * 2], … equal_nan=True)

Returning the calculated point_face_assignment can speed up multiple results >>> m_tf, point_face_assignment = m.project(tf, delta=0.01, … return_point_face_assignment=True) >>> m_tf_fast = m.project(tf, delta=0.01, point_face_assignment=point_face_assignment) >>> assert m_tf.equal(m_tf_fast, equal_nan=True)

remove_faces(*face_delete_mask*)

Remove faces where face_delete_mask is True

template(*sub_mesh*)

'Manual' way to build a template that can be used with self.cut :returns:

template (see cut), can be used as template to retrieve
sub_mesh from self instance

Return type
Mesh3D

Examples

```
>>> import tfields
>>> from sympy.abc import y
>>> mp = tfields.TensorFields([[0,1,2],[2,3,0],[3,2,5],[5,4,3]],
...                           [1, 2, 3, 4])
>>> m = tfields.Mesh3D([[0,0,0], [1,0,0], [1,1,0], [0,1,0], [0,2,0], [1,2,0]],
...                    maps=[mp])
>>> m_cut = m.cut(y < 1.5, at_intersection='split')
>>> template = m.template(m_cut)
>>> assert m_cut.equal(m.cut(template))
```

property tree

Cached property to retrieve a bounding_box Searcher. This searcher can hugely optimize 'in_faces' searches

Examples

```
>>> import numpy as np
>>> import tfields
>>> mesh = tfields.Mesh3D.grid((0, 1, 3), (1, 2, 3), (2, 3, 3))
>>> _ = mesh.tree
>>> assert hasattr(mesh, '_cache')
>>> assert 'mesh_tree' in mesh._cache
>>> face_indices = mesh.in_faces(tfields.Points3D([[0.2, 1.2, 2.0]]),
...                              0.00001)
```

You might want to know the number of points per face >>> unique, counts = np.unique(face_indices, return_counts=True) >>> dict(zip(unique, counts)) # one point on triangle number 16 {16: 1}

triangles()

Cached method to retrieve the triangles, belonging to this mesh .. rubric:: Examples

```
>>> import tfields
>>> mesh = tfields.Mesh3D.grid((0, 1, 3), (1, 2, 3), (2, 3, 3))
>>> assert mesh.triangles() is mesh.triangles()
```

4.1.7 tfields.planes_3d module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library

class tfields.planes_3d.**Planes3D**(*tensors*, **fields*, ***kwargs*)

Bases: *TensorFields*

Point-NormVector representaion of planes

Examples

```
>>> import tfields
>>> points = [[0, 1, 0]]
>>> norms = [[0, 0, 1]]
>>> plane = tfields.Planes3D(points, norms)
>>> plane.symbolic()[0]
Plane(Point3D(0, 1, 0), (0, 0, 1))
```

coord_sys

fields

name

plot(***kwargs*)

forward to Mesh3D plotting

symbolic()

Returns
list with sympy.Plane objects

Return type
list

4.1.8 tfields.points_3d module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

basic threedimensional tensors

class tfields.points_3d.**Points3D**(*tensors*, ***kwargs*)

Bases: *Tensors*

Points3D is a general class for 3D Point operations and storage. Points are stored in np.arrays of shape (len, 3). Thus the three coordinates of the Points stay close.

Parameters

- **constructor** (*points3DInstance -> copy*) –
- **[points3DInstance1** –
- **points3DInstance2** –
- **treated** (*...] -> coord_sys are correctly*) –
- **coordinates** (*list of*) –

Kwargs:

coord_sys (str):
Use tfields.bases.CARTESIAN -> x, y, z Use tfields.bases.CYLINDER -> r, phi, z Use tfields.bases.SPHERICAL -> r, phi, theta

Examples

Initializing with 3 vectors >>> import tfields >>> import numpy as np >>> p1 = tfields.Points3D([[1., 2., 3.], [4., 5., 6.], [1, 2, -6]]) >>> assert p1.equal([[1., 2., 3.], … [4., 5., 6.], … [1., 2., -6.]])

Initializing with list of coordinates >>> p2 = tfields.Points3D(np.array([[1., 2., 3., 4, 5,], … [4., 5., 6., 7, 8], … [1, 2, -6, -1, 0]]).T) >>> assert p2.equal(… [[1., 4., 1.], … [2., 5., 2.], … [3., 6., -6.], … [4., 7., -1.], … [5., 8., 0.]], atol=1e-8) >>> p2.transform(tfields.bases.CYLINDER) >>> assert p2.equal(… [[4.12310563, 1.32581766, 1.], … [5.38516481, 1.19028995, 2.], … [6.70820393, 1.10714872, -6.], … [8.06225775, 1.05165021, -1.], … [9.43398113, 1.01219701, 0.]], atol=1e-8)

Copy constructor with one instance preserves coord_sys of instance >>> assert tfields.Points3D(p2).coord_sys == p2.coord_sys

Unless you specify other: >>> assert tfields.Points3D(p2, … coord_sys=tfields.bases.CARTESIAN).equal(… [[1., 4., 1.], … [2., 5., 2.], … [3., 6., -6.], … [4., 7., -1.], … [5., 8., 0.]], atol=1e-8)

Copy constructor with many instances chooses majority of coordinates systems to avoid much transformation >>> assert tfields.Points3D.merged(p1, p2, p1).equal(… [[1., 2., 3.], … [4., 5., 6.], … [1., 2., -6.], … [1., 4., 1.], … [2., 5., 2.], … [3., 6., -6.], … [4., 7., -1.], … [5., 8., 0.], … [1., 2., 3.], … [4., 5., 6.], … [1., 2., -6.]], atol=1e-8) >>> p1.transform(tfields.bases.CYLINDER)

… unless specified other. Here it is specified >>> assert tfields.Points3D.merged(… p1, p2, coord_sys=tfields.bases.CYLINDER).equal(… [[2.23606798, 1.10714872, 3.], … [6.40312424, 0.89605538, 6.], … [2.23606798, 1.10714872, -6.], … [4.12310563, 1.32581766, 1.], … [5.38516481, 1.19028995, 2.], … [6.70820393, 1.10714872, -6.], … [8.06225775, 1.05165021, -1.], … [9.43398113, 1.01219701, 0.]], atol=1e-8)

Shape is always (…, 3) >>> p = tfields.Points3D([[1., 2., 3.], [4., 5., 6.], … [1, 2, -6], [-5, -5, -5], [1,0,-1], [0,1,-1]]) >>> p.shape (6, 3)

Empty array will create an ndarray of the form (0, 3) >>> tfields.Points3D([]) Points3D([], shape=(0, 3), dtype=float64)

Use it as np.ndarrays -> masking etc. is inherited >>> mask = np.array([True, False, True, False, False, True]) >>> mp = p[mask].copy()

Copy constructor >>> assert mp.equal(… [[1., 2., 3.], … [1., 2., -6.], … [0., 1., -1.]]) >>> assert tfields.Points3D(mp).equal(… [[1., 2., 3.], … [1., 2., -6.], … [0., 1., -1.]])

Coordinate system is implemented. Default is cartesian >>> p_cart = p.copy() >>> p.transform(tfields.bases.CYLINDER) >>> assert p.equal(… tfields.Points3D([[2.236, 1.107, 3.], … [6.403, 0.896, 6.], … [2.236, 1.107, -6.], … [7.071, -2.356, -5.], … [1. , 0. , -1.], … [1. , 1.571, -1.]], … coord_sys=tfields.bases.CYLINDER), … atol=1e-3) >>> p.transform(tfields.bases.CARTESIAN) >>> assert p.equal(p_cart, atol=1e-15)

balls(*radius*, *spacing=(5, 3)*)

Parameters

- **radius** (*float*) – radius of spheres
- **spacing** (*tuple of int*) – n_phi, n_theta

Returns

Builds a sphere around each point with a resolution
defined by spacing and given radius

Return type
tfields.Mesh3D

coord_sys

name

4.1.9 tfields.tensor_grid module

Implementaiton of TensorGrid class

class tfields.tensor_grid.TensorGrid(*tensors*, **fields*, ***kwargs*)

Bases: *TensorFields*

A Tensor Grid is a TensorField which is aware of it’s grid nature, which is order of iteration (iter-order) over the base vectors (base_vectors).

Parameters

- ***base_vectors** (*tuple*) – indices of the axes which should be iterated
- ****kwargs** – num (np.array): same as np.linspace ‘num’ iter_order (np.array): index order of building the grid. further: see TensorFields class

base_num_tuples()

Returns the grid style base_vectors + num tuples

base_vectors

change_iter_order(*iter_order*)

Transform the iter order

coord_sys

classmethod **empty**(**base_vectors*, ***kwargs*)

Build the grid (implicitly) from base vectors

explicit()

Build the grid explicitly (e.g. after changing base_vector, iter_order or init with empty)

fields

classmethod **grid**(**base_vectors*, *tensors=None*, *fields=None*, ***kwargs*)

Build the grid (explicitly) from base vectors

Parameters

- **args** (*explicit*) – see __new__
- ****kwargs** – see TensorFields

is_empty()

Check if the object is an implicit grid (base points are empty but base_vectors and iter order can be used to build the explicit grid's base points).

iter_order

classmethod **merged**(**objects*, ***kwargs*)

Factory method Merges all input arguments to one object

Parameters

- **return_templates** (*bool*) – return the templates which can be used together with cut to retrieve the original objects
- **dim** (*int*) –
- ****kwargs** – passed to cls

Examples

```
>>> import numpy as np
>>> import tfields
>>> import tfields.bases
```

The new object with turn out in the most frequent coordinate system if not specified explicitly

```
>>> vec_a = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vec_b = tfields.Tensors([[5, 4, 1]],
...     coord_sys=tfields.bases.cylinder)
>>> vec_c = tfields.Tensors([[4, 2, 3]],
...     coord_sys=tfields.bases.cylinder)
>>> merge = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, [[2, 0, 1]])
>>> assert merge.coord_sys == 'cylinder'
>>> assert merge.equal([[0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
...                     [0, 0, 1],
...                     [1, -np.pi / 2, 0],
...                     [5, 4, 1],
...                     [4, 2, 3],
...                     [2, 0, 1]])
```

Merge also shifts the maps to still refer to the same tensors

```
>>> tm_a = tfields.TensorMaps(merge, maps=[[[0, 1, 2]]])
>>> tm_b = tm_a.copy()
>>> assert tm_a.coord_sys == 'cylinder'
>>> tm_merge = tfields.TensorMaps.merged(tm_a, tm_b)
>>> assert tm_merge.coord_sys == 'cylinder'
>>> assert tm_merge.maps[3].equal([[0, 1, 2],
...                               list(range(len(merge),
...                                          len(merge) + 3,
...                                          1))])
```

```
>>> obj_list = [tfields.Tensors([[1, 2, 3]],
...             coord_sys=tfields.bases.CYLINDER),
...             tfields.Tensors([[3] * 3]),
...             tfields.Tensors([[5, 1, 3]])]
>>> merge2 = tfields.Tensors.merged(
...     *obj_list, coord_sys=tfields.bases.CARTESIAN)
>>> assert merge2.equal([[-0.41614684, 0.90929743, 3.],
...                      [3, 3, 3], [5, 1, 3]], atol=1e-8)
```

The return_templates argument allows to retrieve a template which can be used with the cut method.

```
>>> merge, templates = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, return_templates=True)
>>> assert merge.cut(templates[0]).equal(vec_a)
>>> assert merge.cut(templates[1]).equal(vec_b)
>>> assert merge.cut(templates[2]).equal(vec_c)
```

name

num

property rank

Tensor rank

4.1.10 tfields.triangles_3d module

Author: Daniel Boeckenhoff Mail: daniel.boeckenhoff@ipp.mpg.de

part of tfields library

class tfields.triangles_3d.**Triangles3D**(*tensors*, **fields*, ***kwargs*)

Bases: *TensorFields*

Points3D child restricted to n * 3 Points. Three Points always group together to one triangle.

Parameters

- **tensors** (*Iterable | tfields.TensorFields*) –
- ***fields** (*Iterable | tfields.Tensors*) – Fields with the same length as tensors
- ****kwargs** – passed to base class

see :class:`~tfields.TensorFields`

Examples

```
>>> import tfields
>>> t = tfields.Triangles3D([[1,2,3], [3,3,3], [0,0,0]])
```

You can add fields to each triangle

```
>>> t = tfields.Triangles3D(t, tfields.Tensors([42]))
>>> assert t.fields[0].equal([42])
```

areas(*transform=None*)

Calculate area with "heron's formula"

Parameters

transform (*np.ndarray*) – optional transformation matrix The triangle points are transformed with transform if given before calclulating the area

Examples

```
>>> import numpy as np
>>> import tfields
>>> m = tfields.Mesh3D([[1,0,0], [0,0,1], [0,0,0]],
...                    faces=[[0, 1, 2]])
>>> assert np.allclose(m.triangles().areas(), np.array([0.5]))
```

```
>>> m = tfields.Mesh3D([[1,0,0], [0,1,0], [0,0,0], [0,0,1]],
...                    faces=[[0, 1, 2], [1, 2, 3]])
>>> assert np.allclose(m.triangles().areas(), np.array([0.5, 0.5]))
```

```
>>> m = tfields.Mesh3D([[1,0,0], [0,1,0], [1,1,0], [0,0,1], [1,0,1]],
...                    faces=[[0, 1, 2], [0, 3, 4]])
>>> assert np.allclose(m.triangles().areas(), np.array([0.5, 0.5]))
```

centroids()

Returns

_centroids()

Examples

```
>>> import tfields
>>> m = tfields.Mesh3D([[0,0,0], [1,0,0], [-1,0,0], [0,1,0], [0,0,1]],
...                    faces=[[0, 1, 3],[0, 2, 3],[1,2,4], [1, 3, 4]]);
>>> assert m.triangles().centroids().equal(
...     [[1./3, 1./3, 0.],
...      [-1./3, 1./3, 0.],
...      [0., 0., 1./3],
...      [1./3, 1./3, 1./3]])
```

circumcenters()

Semi baricentric method to calculate circumcenter points of the triangles

Examples

```
>>> import numpy as np
>>> import tfields
>>> m = tfields.Mesh3D([[0,0,0], [1,0,0], [-1,0,0], [0,1,0], [0,0,1]],
...                    faces=[[0, 1, 3],[0, 2, 3],[1,2,4], [1, 3, 4]]);
>>> assert np.allclose(
...     m.triangles().circumcenters(),
...     [[0.5, 0.5, 0.0],
...      [-0.5, 0.5, 0.0],
...      [0.0, 0.0, 0.0],
...      [1.0 / 3, 1.0 / 3, 1.0 / 3]])
```

coord_sys

corners()

Returns
three np.arrays with corner points of triangles

cut(*expression*, *coord_sys=None*)

Default cut method for Triangles3D

Examples

```
>>> import sympy
>>> import numpy as np
>>> import tfields
>>> x, y, z = sympy.symbols('x y z')
>>> t = tfields.Triangles3D([[1., 2., 3.], [-4., 5., 6.], [1, 2, -6],
...                          [5, -5, -5], [1, 0, -1], [0, 1, -1],
...                          [-5, -5, -5], [1, 0, -1], [0, 1, -1]])
>>> tc = t.cut(x >= 0)
>>> assert tc.equal(tfields.Triangles3D([[ 5., -5., -5.],
...                                      [ 1.,  0., -1.],
...                                      [ 0.,  1., -1.]]))
>>> t.fields.append(tfields.Tensors([1,2,3]))
>>> tc2 = t.cut(x >= 0)
>>> assert np.array_equal(tc2.fields[-1], np.array([2.]))
```

edges()

Retrieve two of the three edge vectors

Returns

vectors ab and ac, where a, b, c are corners (see
self.corners)

Return type
two np.ndarrays

evalf(*expression=None*, *coord_sys=None*)

Triangle3D implementation

Examples

```
>>> from sympy.abc import x
>>> import numpy as np
>>> import tfields
>>> t = tfields.Triangles3D([[1., 2., 3.], [-4., 5., 6.], [1, 2, -6],
...                          [5, -5, -5], [1,0,-1], [0,1,-1],
...                          [-5, -5, -5], [1,0,-1], [0,1,-1]])
>>> mask = t.evalf(x >= 0)
>>> assert np.array_equal(t[mask],
...                       tfields.Triangles3D([[ 5., -5., -5.],
...                                            [ 1.,  0., -1.],
...                                            [ 0.,  1., -1.]]))
```

Returns
mask which is True, where expression evaluates True

Return type
np.array

fields

in_triangles(*tensors*, *delta: Optional[float] = 0.0*, *assign_multiple: bool = False*) → Union[List[List[int]], array]

Barycentric method to obtain, which tensors are containes in any of the triangles

Parameters

- **tensors** (*Points3D instance*) –
- **optional** –
- **delta** –

  float: Normal distance to a triangle, that the points
  are concidered to be contained in the triangle.

  None: Find the minimum distance. Default is 0.
- **assign_multiple** – If True, one point may belong to multiple triangles at the same time. In the other case the first occurence will be True the other False

Returns
[index(or indices if assign_multiple) of triangle for point in tensors]

Return type
list

classmethod merged(**objects*, ***kwargs*)

Factory method Merges all input arguments to one object

Parameters

- **return_templates** (*bool*) – return the templates which can be used together with cut to retrieve the original objects
- **dim** (*int*) –
- ****kwargs** – passed to cls

Examples

```
>>> import numpy as np
>>> import tfields
>>> import tfields.bases
```

The new object with turn out in the most frequent coordinate system if not specified explicitly

```
>>> vec_a = tfields.Tensors([[0, 0, 0], [0, 0, 1], [0, -1, 0]])
>>> vec_b = tfields.Tensors([[5, 4, 1]],
...     coord_sys=tfields.bases.cylinder)
>>> vec_c = tfields.Tensors([[4, 2, 3]],
...     coord_sys=tfields.bases.cylinder)
>>> merge = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, [[2, 0, 1]])
>>> assert merge.coord_sys == 'cylinder'
>>> assert merge.equal([[0, 0, 0],
...                     [0, 0, 1],
...                     [1, -np.pi / 2, 0],
...                     [5, 4, 1],
...                     [4, 2, 3],
...                     [2, 0, 1]])
```

Merge also shifts the maps to still refer to the same tensors

```
>>> tm_a = tfields.TensorMaps(merge, maps=[[[0, 1, 2]]])
>>> tm_b = tm_a.copy()
>>> assert tm_a.coord_sys == 'cylinder'
>>> tm_merge = tfields.TensorMaps.merged(tm_a, tm_b)
>>> assert tm_merge.coord_sys == 'cylinder'
>>> assert tm_merge.maps[3].equal([[0, 1, 2],
...                               list(range(len(merge),
...                                          len(merge) + 3,
...                                          1))])
```

```
>>> obj_list = [tfields.Tensors([[1, 2, 3]],
...             coord_sys=tfields.bases.CYLINDER),
...             tfields.Tensors([[3] * 3]),
...             tfields.Tensors([[5, 1, 3]])]
>>> merge2 = tfields.Tensors.merged(
...     *obj_list, coord_sys=tfields.bases.CARTESIAN)
>>> assert merge2.equal([[-0.41614684, 0.90929743, 3.],
...                      [3, 3, 3], [5, 1, 3]], atol=1e-8)
```

The return_templates argument allows to retrieve a template which can be used with the cut method.

```
>>> merge, templates = tfields.Tensors.merged(
...     vec_a, vec_b, vec_c, return_templates=True)
>>> assert merge.cut(templates[0]).equal(vec_a)
>>> assert merge.cut(templates[1]).equal(vec_b)
>>> assert merge.cut(templates[2]).equal(vec_c)
```

mesh()

Returns
tfields.Mesh3D

name

norms()

Examples

```
>>> import numpy as np
>>> import tfields
>>> m = tfields.Mesh3D([[0,0,0], [1,0,0], [-1,0,0], [0,1,0], [0,0,1]],
...                    faces=[[0, 1, 3],[0, 2, 3],[1,2,4], [1, 3, 4]]);
>>> assert np.allclose(m.triangles().norms(),
...                    [[0.0, 0.0, 1.0],
...                     [0.0, 0.0, -1.0],
...                     [0.0, 1.0, 0.0],
...                     [0.57735027] * 3],
...                    atol=1e-8)
```

ntriangles()

Returns
number of triangles

Return type
int

4.1.11 Module contents

Top-level package of tfields. TODO: proper documentation, also in dough.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://gitlab.mpcdf.mpg.de/dboe/tfields/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

If you want quick feedback, it is helpful to mention speicific developers (@devloper_name) or @all. This will trigger a mail to the corresponding developer(s).

5.1.2 Fix Bugs

Look through the repository issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the remote issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

tfields could always use more *documentation*, whether as part of the official *tfields* docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Write Unittests or Doctests

tfields profits a lot from better *testing*. We encourage you to add unittests (in the *tests* directory) or doctests (as part of docstrings or in the documentation).

5.1.6 Submit Feedback

The best way to send feedback is to file an Issue.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *tfields* for local development.

1. Fork the *tfields* repo.
2. Clone your fork locally:

```
$ git clone git@gitlab.mpcdf.mpg.de:dboe/tfields.git
```

3. Set up your fork for local development:

```
$ cd tfields/
$ pip install .[dev]
```

4. Step 3. already installed pre-commit. Initialize it by running:

```
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ make test
```

7. Commit your changes and push your branch to origin:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the repository website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://gitlab.mpcdf.mpg.de/dboe/tfields/-/merge_requests and make sure that the tests pass for all supported Python versions.

5.4 Testing

To run tests, use:

```
$ make test
```

To run a subset of tests, you have the following options:

```
$ pytest tests/test_package.py

$ pytest tests/test_package.py::Test_tfields::test_version_type

$ pytest --doctest-modules docs/usage.rst

$ pytest --doctest-modules tfields/core.py -k "MyClass.funciton_with_doctest"
```

Use the '–trace' option to directly jump into a pdb debugger on fails. Check out the coverage of your api with:

```
$ make coverage
```

5.5 Documentation

To compile the documentation (including automatically generated module api docs), run:

```
$ make doc
```

Use doctests as much as possible in order to have tested examples in your documentation.

5.6 Styleguide

Please follow the google style guide illustrated by this example.

5.7 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed. Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

or use the convenient alias for the above (patch increases only):

```
$ make publish
```

The CI will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

This package was created with Cookiecutter and the dboe/dough project template.

6.1 Development Lead

- Daniel Böckenhoff <dboe@ipp.mpg.de>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

I

K

L

M

U

V