hypothesis-gufunc

Hypothesis GU Functions Team

Sep 24, 2019

CONTENTS

Contents:

CHAPTER

ONE

HYPOTHESIS GU FUNCS

This project is experimental and the APIs are not considered stable.

Only *Python>=3.6* is officially supported, but older versions of Python likely work as well.

This package includes support for strategies which generate arguments to functions that follow the numpy general universal function API. So, it can automatically generate the matrices with shapes that follow the shape constraints. For example, to generate test inputs for *np.dot*, one can use,

```
import numpy as np
from hypothesis import given
from hypothesis.strategies import floats
from hypothesis_gufunc.gufunc import gufunc_args

easy_floats = floats(min_value=-10, max_value=10)

@given(gufunc_args('(m,n),(n,p)->(m,p)', dtype=np.float_, elements=easy_floats))
def test_np_dot(args):
    x, y = args
    assert np.allclose(np.dot(x, y), np.dot(y.T, x.T).T)

test_np_dot()  # Run the test
```

We also allow for adding extra dimensions that follow the numpy broadcasting conventions. This allows one to test that the broadcasting follows the vectorization conventions:

```
@given(gufunc_args('(m,n),(n,p)->(m,p)', dtype=np.float_, elements=easy_floats, max_
↪dims_extra=3))
def test_np_matmul(args):
    x, y = args
    f_vec = np.vectorize(np.matmul, signature='(m,n),(n,p)->(m,p)', otypes=[np.float_
↪])
    assert np.allclose(np.matmul(x, y), f_vec(x, y))

test_np_matmul()  # Run the test
```

Providing *max_dims_extra=3* gives up to 3 broadcast compatible dimensions on each of the arguments.

1.1 Quick Start/Installation

Simply install with pip:

```
pip install hypothesis-gufunc
```

If one would like the same pinned requirements we use during testing, then one can install from the repo with:

```
git clone git@github.com:uber/hypothesis-gufunc.git
cd hypothesis-gufunc
pip install -r requirements/base.txt
pip install -e .
```

1.2 Running the Tests

The tests for this package can be run by first doing a *cd* to its root directory and then

```
./test.sh
```

The script creates a conda environment using the requirements found in *requirements/test.txt*.

1.3 Hypothesis for Xarray

This package also contains an extension to hypothesis to generate xarray data structures.

To install the package with the xarray dependencies install it with pip as

```
pip install hypothesis-gufunc[xarray]
```

Once installed, one can generate a data array as follows:

```
from hypothesis.strategies import integers, lists
from hypothesis_gufunc.extra.xr import fixed_dataarrays

S = fixed_dataarrays(("a", "b"), coords_st={"a": lists(integers(0, 3))})
S.example()
```

Here, *coords_st* allows one to specify a custom strategy for the coordinates on a per-dimension basis. Likewise, if one has known coordinates one can call *fixed_coords_dataarrays*; or *dataarrays* if one wants both the dimensions and coordinates determined by the strategy.

The package also has the ability to generate a dataset:

```
from hypothesis_gufunc.extra.xr import fixed_datasets

S = fixed_datasets({5: ("a", "b"), "bar": ("b"), "baz": ()}, coords_st={"a":␣
↪lists(integers(0, 3))})
S.example()
```

One can use *fixed_coords_datasets* when the coordinates are determined; or simply *datasets* to have both the dimensions and coordinates generated.

1.4 Links

The source is hosted on GitHub.

The documentation is hosted at Read the Docs.

The main hypothesis project.

A description of the numpy Generalized Universal Function API.

Likewise, the numpy broadcasting rules are described here.

The xarray project describes data arrays and datasets.

1.5 License

CHAPTER

TWO

API OVERVIEW

2.1 Generalized Universal Function API

This module implements strategies for creating arguments for functions that follow numpy's Generalized Universal Function API.

hypothesis_gufunc.gufunc.**gufunc_args**(*signature*, *dtype*, *elements*, *unique=False*, *excluded=()*, *min_side=0*, *max_side=5*, *max_dims_extra=0*)

Strategy to generate a tuple of ndarrays for arguments to a function consistent with its signature with extra dimensions to test broadcasting.

Parameters

- **signature** (*str*) – Signature for shapes to be compatible with. Expects string in format of numpy generalized universal function signature, e.g., *'(m,n),(n)->(m)'* for vectorized matrix-vector multiplication.
- **dtype** (list(numpy.dtype)) – List of numpy *dtype* for each argument. These can be either strings (`'int64'`), type (`np.int64`), or numpy *dtype* (`np.dtype('int64')`). Built in Python types (*int*, *float*, etc) also work. A single *dtype* can be supplied for all arguments.
- **elements** (*list*) – List of strategies to fill in array elements on a per argument basis. One can also specify a single strategy (e.g., `floats()`) and have it applied to all arguments.
- **unique** (*list(bool)*) – Boolean flag to specify if all elements in an array must be unique. One can also specify a single boolean to apply it to all arguments.
- **excluded** (*set(int)*) – Set of integers representing the positional for which the function will not be vectorized. Uses same format as `numpy.vectorize`.
- **min_side** (*int or dict*) – Minimum size of any side of the arrays. It is good to test the corner cases of 0 or 1 sized dimensions when applicable, but if not, a min size can be supplied here. Minimums can be provided on a per-dimension basis using a dict, e.g. `min_side={'n': 2}`. One can use, e.g., `min_side={hypothesis_gufunc.gufunc.BCAST_DIM: 2}` to limit the size of the broadcasted dimensions.
- **max_side** (*int or dict*) – Maximum size of any side of the arrays. This can usually be kept small and still find most corner cases in testing. Dictionaries can be supplied as with *min_side*.
- **max_dims_extra** (*int*) – Maximum number of extra dimensions that can be appended on left of arrays for broadcasting. This should be kept small as the memory used grows exponentially with extra dimensions. By default, no extra dimensions are added.

Returns **res** – Resulting ndarrays with shapes consistent with *signature* and elements from *elements*. Extra dimensions for broadcasting will be present.

Return type tuple(numpy.ndarray)

Examples

```
>>> from hypothesis_gufunc.gufunc import BCAST_DIM
>>> from hypothesis.strategies import integers, booleans
>>> gufunc_args('(m,n),(n)->(m)',
                dtype=np.int_, elements=integers(0, 9), max_side=3,
                min_side={'m': 1, 'n': 2, BCAST_DIM: 3}).example()
(array([[9, 8, 1],
        [1, 7, 1]]), array([5, 6, 5]))
>>> gufunc_args('(m,n),(n)->(m)', dtype=['bool', 'int32'],
                    elements=[booleans(), integers(0, 100)],
                    unique=[False, True], max_dims_extra=3).example()
(array([[[[[ True,  True,  True,  True,  True],
           [False,  True,  True,  True, False]]]]], dtype=bool),
 array([67, 43,  0, 34, 66], dtype=int32))
```

hypothesis_gufunc.gufunc.**gufunc_arg_shapes**(*signature*, *excluded=()*, *min_side=0*, *max_side=5*, *max_dims_extra=0*)

Strategy to generate the shape of ndarrays for arguments to a function consistent with its signature with extra dimensions to test broadcasting.

Parameters

- **signature** (*str*) – Signature for shapes to be compatible with. Expects string in format of numpy generalized universal function signature, e.g., *'(m,n),(n)->(m)'* for vectorized matrix-vector multiplication.
- **excluded** (*set(int)*) – Set-like of integers representing the positional for which the function will not be vectorized. Uses same format as numpy.vectorize.
- **min_side** (*int or dict*) – Minimum size of any side of the arrays. It is good to test the corner cases of 0 or 1 sized dimensions when applicable, but if not, a min size can be supplied here. Minimums can be provided on a per-dimension basis using a dict, e.g. `min_side={'n': 2}`. One can use, e.g., `min_side={hypothesis_gufunc.gufunc.BCAST_DIM: 2}` to limit the size of the broadcasted dimensions.
- **max_side** (*int or dict*) – Maximum size of any side of the arrays. This can usually be kept small and still find most corner cases in testing. Dictionaries can be supplied as with *min_side*.
- **max_dims_extra** (*int*) – Maximum number of extra dimensions that can be appended on left of arrays for broadcasting. This should be kept small as the memory used grows exponentially with extra dimensions. By default, no extra dimensions are added.

Returns **shapes** – list of tuples where each tuple is the shape of an argument. Extra dimensions for broadcasting will be present in the shapes.

Return type list(tuple(int))

Examples

```
>>> from hypothesis_gufunc.gufunc import BCAST_DIM
>>> gufunc_arg_shapes('(m,n),(n)->(m)',
                      min_side={'m': 1, 'n': 2}, max_side=3).example()
[(2, 3), (3,)]
```

(continues on next page)

(continued from previous page)

```
>>> gufunc_arg_shapes('(m,n),(n)->(m)', max_side=9,
                      min_side={'m': 1, 'n': 2, BCAST_DIM: 5},
                      max_dims_extra=3).example()
[(6, 6, 7), (6, 7)]
>>> gufunc_arg_shapes('(m,n),(n)->(m)', excluded=(0,),
                      max_side=20, max_dims_extra=3).example()
[(11, 13), (1, 1, 1, 13)]
```

2.2 Hypothesis for Xarray

This module implements strategies for creating `xarray.DataArray` and `xarray.Dataset` objects.

hypothesis_gufunc.extra.xr.**dataarrays**(*dtype=<class 'int'>*, *elements=None*, *coords_elements=None*, *min_side=0*, *max_side=5*, *min_dims=0*, *max_dims=5*)

Generate a `xarray.DataArray` with no dimensions or coordinates fixed a-priori.

Parameters

- **dtype** (*type*) – Data type for values in the `xarray.DataArray`. This can be anything understood by `hypothesis.extra.numpy.arrays()`.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the `xarray.DataArray`. If *None*, a default is selected based on *dtype*.
- **coords_elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates.
- **min_side** (*int*) – Minimum side length of the `xarray.DataArray`.
- **max_side** (*int or None*) – Maximum side length of the `xarray.DataArray`.
- **min_dims** (*int*) – Minimum number of dimensions.
- **max_dims** (*int or None*) – Maximum number of dimensions.

Returns **da** – `xarray.DataArray` generated with the dimensions, simple coordinates, and elements from the specified strategies.

Return type `xarray.DataArray`

hypothesis_gufunc.extra.xr.**datasets**(*dtype=<class 'int'>*, *elements=None*, *coords_elements=None*, *min_side=0*, *max_side=5*, *min_vars=0*, *max_vars=5*, *min_dims=0*, *max_dims=5*)

Generate a `xarray.Dataset` with no variables, dimensions, or coordinates fixed a-priori.

We could also allow a strategy with a different data type per variable, but until there is a use case for that, we will leave *dtype* as a scalar input.

Parameters

- **dtype** (*type*) – Data type used to fill the elements of the `xarray.Dataset`.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the `xarray.Dataset`. If *None*, a default is selected based on *dtype*.
- **coords_elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates.
- **min_side** (*int*) – Minimum side length of the `xarray.Dataset`.

- **max_side** (*int or None*) – Maximum side length of the xarray.Dataset.
- **min_vars** (*int*) – Minimum number of variables.
- **max_vars** (*int or None*) – Maximum number of variables.
- **min_dims** (*int*) – Minimum number of dimensions.
- **max_dims** (*int or None*) – Maximum number of dimensions.

Returns **ds** – xarray.Dataset generated with the variables, dimensions, coordinates, and elements from the specified strategies.

Return type xarray.Dataset

hypothesis_gufunc.extra.xr.**fixed_coords_dataarrays**(*dims*, *coords*, *dtype=<class 'int'>*, *elements=None*)

Generate a xarray.DataArray with coordinates that are fixed a-priori.

Parameters

- **dims** (*list(str)*) – Dimensions we need to generate coordinates for.
- **coords** (*dict(str, list)*) – Dictionary mapping dimension name to its coordinate values.
- **dtype** (*type*) – Data type for values in the xarray.DataArray. This can be anything understood by hypothesis.extra.numpy.arrays().
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the xarray.DataArray. If *None*, a default is selected based on *dtype*.

Returns **da** – xarray.DataArray generated with the specified coordinates and elements from the specified strategy.

Return type xarray.DataArray

hypothesis_gufunc.extra.xr.**fixed_coords_datasets**(*vars_to_dims*, *coords*, *dtype=None*, *elements=None*)

Generate a xarray.Dataset where the variables, dimensions, and coordinates are specified a-priori.

Parameters

- **vars_to_dims** (*dict(typing.Hashable, list(str))*) – Mapping of variable names to list of dimensions, which can be fed to constructor for a xarray.Dataset.
- **coords** (*dict(str, list)*) – Dictionary mapping dimension name to its coordinate values.
- **dtype** (*dict(typing.Hashable, type) or None*) – Dictionary mapping variables names to the data type for that variable's elements.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the xarray.Dataset. If *None*, a default is selected based on *dtype*.

Returns **ds** – xarray.Dataset with the specified variables, dimensions, and coordinates.

Return type xarray.Dataset

hypothesis_gufunc.extra.xr.**fixed_dataarrays**(*dims*, *dtype=<class 'int'>*, *elements=None*, *coords_elements=None*, *min_side=0*, *max_side=5*, *coords_st={}*)

Generate xarray.DataArray with dimensions (but not coordinates) that are fixed a-priori.

Parameters

- **dims** (*list(str)*) – Dimensions we need to generate coordinates for.

- **dtype** (*type*) – Data type for values in the `xarray.DataArray`. This can be anything understood by `hypothesis.extra.numpy.arrays()`.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the `xarray.DataArray`. If *None*, a default is selected based on *dtype*.
- **coords_elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates.
- **min_side** (*int*) – Minimum side length of the `xarray.DataArray`.
- **max_side** (*int or None*) – Maximum side length of the `xarray.DataArray`.
- **coords_st** (*dict(str, SearchStrategy)*) – Special strategies for filling specific dimensions. Use the dimension name as the key and the strategy for generating the coordinate as the value.

Returns **da** – `xarray.DataArray` generated with the dimensions and elements from the specified strategy.

Return type `xarray.DataArray`

hypothesis_gufunc.extra.xr.**fixed_datasets**(*vars_to_dims*, *dtype=None*, *elements=None*, *coords_elements=None*, *min_side=0*, *max_side=5*, *coords_st={}*)

Generate `xarray.Dataset` where the variables and dimensions (but not coordinates) are specified a-priori.

Parameters

- **vars_to_dims** (*dict(typing.Hashable, list(str))*) – Mapping of variable names to list of dimensions, which can be fed to constructor for a `xarray.Dataset`.
- **dtype** (*dict(typing.Hashable, type) or None*) – Dictionary mapping variables names to the data type for that variable's elements.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the `xarray.Dataset`. If *None*, a default is selected based on *dtype*.
- **coords_elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates.
- **min_side** (*int*) – Minimum side length of the `xarray.Dataset`.
- **max_side** (*int or None*) – Maximum side length of the `xarray.Dataset`.
- **coords_st** (*dict(str, SearchStrategy)*) – Special strategies for filling specific dimensions. Use the dimension name as the key and the strategy for generating the coordinate as the value.

Returns **ds** – `xarray.Dataset` with the specified variables and dimensions.

Return type `xarray.Dataset`

hypothesis_gufunc.extra.xr.**simple_coords**(*min_side=0*, *max_side=5*)

Generate a simple coordinate for a `xarray.DataArray`.

A simple coordinate is one in which the values go: 0, 1, . . . , n.

Parameters

- **min_side** (*int*) – Minimum length of coordinates array.
- **max_side** (*int or None*) – Maximum length of coordinates array.

Returns **L** – The coordinates filled with values of: `list(range(len(L)))`.

Return type list(int)

hypothesis_gufunc.extra.xr.**simple_dataarrays**(*dims*, *dtype=<class 'int'>*, *elements=None*, *min_side=0*, *max_side=5*)

Generate a xarray.DataArray with dimensions that are fixed a-priori and simple coordinates.

Parameters

- **dims** (*list(str)*) – Dimensions we need to generate coordinates for.
- **dtype** (*type*) – Data type for values in the xarray.DataArray. This can be anything understood by hypothesis.extra.numpy.arrays().
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the xarray.DataArray. If *None*, a default is selected based on *dtype*.
- **min_side** (*int*) – Minimum side length of the xarray.DataArray.
- **max_side** (*int or None*) – Maximum side length of the xarray.DataArray.

Returns **da** – xarray.DataArray generated with the dimensions, simple coordinates, and elements from the specified strategy.

Return type xarray.DataArray

hypothesis_gufunc.extra.xr.**simple_datasets**(*vars_to_dims*, *dtype=None*, *elements=None*, *min_side=0*, *max_side=5*)

Generate xarray.Dataset with variables and dimensions that are fixed a-priori and simple coordinates.

Parameters

- **vars_to_dims** (*dict(typing.Hashable, list(str))*) – Mapping of variable names to list of dimensions, which can be fed to constructor for a xarray.Dataset.
- **dtype** (*dict(typing.Hashable, type) or None*) – Dictionary mapping variables names to the data type for that variable's elements.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of the xarray.Dataset. If *None*, a default is selected based on *dtype*.
- **min_side** (*int*) – Minimum side length of the xarray.Dataset.
- **max_side** (*int or None*) – Maximum side length of the xarray.Dataset.

Returns **ds** – A xarray.Dataset with the specified variables and dimensions, and simple coordinates.

Return type xarray.Dataset

hypothesis_gufunc.extra.xr.**subset_lists**(*L*, *min_size=0*, *max_size=None*)

Strategy to generate a subset of a *list*.

This should be built in to hypothesis (see hypothesis issue #1115), but was rejected.

Parameters

- **L** (*list*) – List of elements we want to get a subset of.
- **min_size** (*int*) – Minimum size of the resulting subset list.
- **max_size** (*int or None*) – Maximum size of the resulting subset list.

Returns **L** – List that is subset of *L* with all unique elements.

Return type list

hypothesis_gufunc.extra.xr.**vars_to_dims_dicts**(*min_vars=0*, *max_vars=5*, *min_dims=0*, *max_dims=5*)

Generate mapping of variable name to *list* of dimensions, which is compatible with building a xarray.Dataset.

Parameters

- **min_vars** (*int*) – Minimum size of the resulting variable list.
- **max_vars** (*int or None*) – Maximum size of the resulting variable list.
- **min_dims** (*int*) – Minimum size of the resulting dimension list.
- **max_dims** (*int or None*) – Maximum size of the resulting dimension list.

Returns **D** – Mapping of variable names to *list* of dimensions, which can be fed to constructor for a `xarray.Dataset`.

Return type dict(typing.Hashable, list(str))

hypothesis_gufunc.extra.xr.**xr_coords**(*elements=None*, *min_side=0*, *max_side=5*, *unique=True*)

Generate values for the coordinates in a `xarray.DataArray`.

Non-unique coords do not make much sense, but xarray allows it. So we should be able to generate it.

Parameters

- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates. Uses `hypothesis.strategies.integers()` by default.
- **min_side** (*int*) – Minimum length of coordinates array.
- **max_side** (*int or None*) – Maximum length of coordinates array.
- **unique** (*bool*) – If all coordinate values should be unique. *xarray* allows non-unique values, but it makes no sense.

Returns **L** – The coordinates filled with samples from *elements*.

Return type list

hypothesis_gufunc.extra.xr.**xr_coords_dicts**(*dims*, *elements=None*, *min_side=0*, *max_side=5*, *unique_coords=True*, *coords_st={}*)

Build a dictionary of coordinates for the purpose of building a `xarray.DataArray`.

xarray allows some dims to not have any specified coordinate. This strategy assigns a coord to every dimension. If we really want to test those possibilities we need to take a subset of the *dict* that is sampled from this strategy.

Parameters

- **dims** (*list(str)*) – Dimensions we need to generate coordinates for.
- **elements** (*SearchStrategy or None*) – Strategy to fill the elements of coordinates. Uses *integers* by default.
- **min_side** (*int*) – Minimum length of coordinates array.
- **max_side** (*int or None*) – Maximum length of coordinates array.
- **unique_coords** (*bool*) – If all coordinate values should be unique. *xarray* allows non-unique values, but it makes no sense.
- **coords_st** (*dict(str, SearchStrategy)*) – Special strategies for filling specific dimensions. Use the dimension name as the key and the strategy for generating the coordinate as the value.

Returns **coords** – Dictionary mapping dimension name to its coordinate values (a list with elements from the *elements* strategy).

Return type dict(str, list)

hypothesis_gufunc.extra.xr.**xr_dim_lists**(*min_dims=0*, *max_dims=5*)
Generate *list* of dimension names for a xarray.DataArray.

Parameters

- **min_dims** (*int*) – Minimum size of the resulting dimension list.
- **max_dims** (*int or None*) – Maximum size of the resulting dimension list.

Returns **L** – List of dimension names.

Return type list(str)

hypothesis_gufunc.extra.xr.**xr_var_lists**(*min_vars=0*, *max_vars=5*)
Generate *list* of variable names for a xarray.Dataset.

Parameters

- **min_vars** (*int*) – Minimum size of the resulting variable list.
- **max_vars** (*int or None*) – Maximum size of the resulting variable list.

Returns **L** – List of variable names.

Return type list(typing.Hashable)

CHAPTER

THREE

CREDITS

3.1 Development lead

Ryan Turner (rdturnermtl)

3.2 Contributors

Looking for some.

PYTHON MODULE INDEX

h

INDEX