# lagrange

***Release 2.0.2***

**Andrei Lapets**

**Jul 31, 2022**

# CONTENTS

Pure-Python implementation of Lagrange interpolation over finite fields.

CHAPTER

# ONE

# PURPOSE

This library provides a pure-Python implementation of the Lagrange interpolation algorithm over finite fields.

CHAPTER

TWO

# INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install lagrange
```

The library can be imported in the usual way:

```
from lagrange import lagrange
```

## 2.1 Examples

Interpolation can be performed on collections of points represented in a variety of ways:

```
>>> lagrange({1: 15, 2: 9, 3: 3}, 17)
4
>>> lagrange([(1, 15), (2, 9), (3, 3)], 17)
4
>>> lagrange([15, 9, 3], 17)
4
>>> lagrange(\
    {1: 119182, 2: 11988467, 3: 6052427, 4: 8694701,\
     5: 9050123, 6: 3676518, 7: 558333, 8: 12198248,\
     9: 7344866, 10: 10114014, 11: 2239291, 12: 2515398},\
    15485867)
123
>>> lagrange(\
    [119182, 11988467, 6052427, 8694701, 9050123, 3676518,\
     558333, 12198248, 7344866, 10114014, 2239291, 2515398],\
    15485867)
123
```

CHAPTER

THREE

# DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

## 3.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

## 3.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Alternatively, all unit tests are included in the module itself and can be executed using doctest:

```
python src/lagrange/lagrange.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/lagrange
```

## 3.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

## 3.4 Versioning

Beginning with version 0.2.0, the version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

## 3.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

### 3.5.1 lagrange module

Pure-Python implementation of Lagrange interpolation over finite fields.

lagrange.lagrange.**interpolate**(*points*, *modulus*, *degree=None*)

Determine the value at the origin of the domain (*e.g.*, where $x = 0$) given a collection of points. The point information can be represented as a collection of two-component coordinates, as a dictionary, or as a sequence of values.

**Parameters**

- **points** (Union[dict, Sequence[int], Iterable[Sequence[int]]]) – Collection of points to interpolate.
- **modulus** (int) – Modulus representing the finite field within which to interpolate.
- **degree** (Optional[int]) – Degree of the target polynomial.

```
>>> interpolate([(1, 15), (2, 9), (3, 3)], 17)
4
>>> interpolate({1: 15, 2: 9, 3: 3}, 17)
4
>>> interpolate(
...     {
...         1: 119182, 2: 11988467, 3: 6052427, 4: 8694701,
...         5: 9050123, 6: 3676518, 7: 558333, 8: 12198248,
...         9: 7344866, 10: 10114014, 11: 2239291, 12: 2515398
...     },
...     15485867
... )
123
```

If a list of integers is supplied, it is assumed that they are the image of the sequence `[1, 2, 3, ...]`.

```
>>> interpolate([15, 9, 3], 17)
4
>>> interpolate(
...     [
...         119182, 11988467, 6052427, 8694701, 9050123, 3676518,
...         558333, 12198248, 7344866, 10114014, 2239291, 2515398
...     ],
...     15485867
... )
123
```

If the point information is supplied as an `Iterable` of integers, that iterable object must be a `Sequence`.

```
>>> interpolate({15, 9, 3}, 17)
Traceback (most recent call last):
  ...
TypeError: iterable of integers that represents points must be a sequence
```

Alternatively, a two-coordinate `Sequence` can be used to represent each individual point. In that case, any `Iterable` of such individual points is supported.

```
>>> interpolate({(1, 15), (2, 9), (3, 3)}, 17)
4
```

This function is able to interpolate when supplied more points than necessary (*i.e.*, given the degree).

```
>>> lagrange({1: 4, 2: 6, 3: 8, 4: 10, 5: 12}, modulus=65537)
2
>>> lagrange({1: 4, 2: 6, 3: 8, 4: 10, 5: 12}, degree=4, modulus=65537)
2
>>> lagrange({1: 4, 2: 6, 3: 8, 4: 10, 5: 12}, degree=5, modulus=65537)
Traceback (most recent call last):
  ...
ValueError: not enough points for a unique interpolation
>>> lagrange({1: 4, 2: 6, 3: 8, 4: 10, 5: 12}, degree=1, modulus=65537)
2
>>> lagrange({49: 200, 5: 24, 3: 16}, degree=2, modulus=65537)
```

(continues on next page)

(continued from previous page)

```
4
>>> lagrange({49: 200, 5: 24, 3: 16}, degree=1, modulus=65537)
4
>>> lagrange({1: 16, 2: 25, 3: 36}, degree=1, modulus=65537)
7
>>> lagrange({3: 36, 1: 16, 2: 25}, degree=1, modulus=65537)
6
>>> lagrange({1: 16, 2: 25, 3: 36}, degree=2, modulus=65537)
9
>>> lagrange({3: 36, 1: 16, 2: 25}, degree=2, modulus=65537)
9
>>> lagrange({5: 64, 2: 25, 3: 36}, degree=2, modulus=65537)
9
```

Interpolation in trivial scenarios is supported, as well.

```
>>> lagrange([12345], degree=0, modulus=65537)
12345
```

At least one point must be supplied.

```
>>> interpolate([], 17)
Traceback (most recent call last):
  ...
ValueError: at least one point is required
```

An exception is raised if a supplied argument (or a component thereof) does not have the expected structure or is not of the expected type.

```
>>> interpolate({1: 15.0, 'a': 9, 'b': 3}, 17)
Traceback (most recent call last):
  ...
TypeError: dictionary that represents points must have integer keys and values
>>> interpolate({(1, 15, 0), (2, 9, 0), (3, 3, 0)}, 17)
Traceback (most recent call last):
  ...
TypeError: iterable that represents points must contain integers or two-element␣
↪sequences of integers
>>> interpolate('abc', 123)
Traceback (most recent call last):
  ...
TypeError: iterable that represents points must contain integers or two-element␣
↪sequences of integers
>>> interpolate(1.23, 123)
Traceback (most recent call last):
  ...
TypeError: expecting dictionary or iterable that represents points
>>> interpolate([15, 9, 3], 'abc')
Traceback (most recent call last):
  ...
TypeError: expecting an integer prime modulus
>>> interpolate([15, 9, 3], -17)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
  ...
ValueError: expecting a positive integer prime modulus
>>> interpolate([15, 9, 3], 17, 'abc')
Traceback (most recent call last):
  ...
TypeError: expecting an integer degree
>>> interpolate([15, 9, 3], 17, -3)
Traceback (most recent call last):
  ...
ValueError: expecting a nonnegative integer degree
```

**Return type** int

lagrange.lagrange.**lagrange**(*points*, *modulus*, *degree=None*)

Alias for *interpolate*.

**Return type** int

# PYTHON MODULE INDEX

# INDEX