egcd

Release 1.0.0

Andrei Lapets

Mar 11, 2024

CONTENTS

Pure-Python extended Euclidean algorithm implementation that accepts any number of integer arguments.

CHAPTER

ONE

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install egcd
```

The library can be imported in the usual way:

```
from egcd import egcd
```

1.1 Examples

The function `egcd` is a pure-Python implementation of the extended Euclidean algorithm that can be viewed as an expansion of the functionality and interface of the built-in `math.gcd` function. When it is supplied two integer arguments `a` and `b`, it returns a tuple of the form `(g, s, t)` where the three integers in the tuple satisfy the identity `(a * s) + (b * t) == g`:

```
>>> egcd(1, 1)
(1, 0, 1)
>>> egcd(12, 8)
(4, 1, -1)
>>> egcd(23894798501898, 23948178468116)
(2, 2437250447493, -2431817869532)
>>> egcd(pow(2, 50), pow(3, 50))
(1, -260414429242905345185687, 408415383037561)
```

However, any number of integer arguments can be supplied. When no arguments are supplied, the result is `(0,)` (just as the expression `math.gcd()` evaluates to `0` in Python 3.9 and higher). In all other cases, the result contains the greatest common divisor of all the supplied integers and the coefficients of the generalized form of the associated identity:

```
>>> egcd(2, 4, 3, 9)
(1, -1, 0, 1, 0)
>>> 1 == ((-1) * 2) + (0 * 4) + (1 * 3) + (0 * 9)
1
```

A succinct way to extract the greatest common divisor and the coefficients is to take advantage of Python's support for iterable unpacking via the asterisk notation:

```
>>> bs = (26, 16, 34)
>>> (g, *cs) = egcd(*bs)
>>> (g, cs)
```

(continues on next page)

(continued from previous page)

```
(2, [-3, 5, 0])
>>> g == sum(c * b for (c, b) in zip(cs, bs))
True
```

CHAPTER

TWO

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]
```

2.1 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
```

2.2 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/egcd/egcd.py -v
```

Style conventions are enforced using Pylint:

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

2.3 Contributions

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

2.4 Versioning

Beginning with version 0.1.0, 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.

2.5 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/*
```

2.5.1 egcd module

Pure-Python extended Euclidean algorithm implementation that accepts any number of integer arguments.

egcd.egcd.**egcd**(**integers: int*) → Tuple[int, ...]

To support the most typical use case, this function accepts two integers `a` and `b` and returns a tuple of the form `(g, s, t)` such that `g` is the greatest common divisor of `a` and `b` and such that the identity `g == (a * s) + (b * t)` is satisfied.

```
>>> egcd(1, 1)
(1, 0, 1)
>>> egcd(12, 8)
(4, 1, -1)
>>> 4 == (1 * 12) + ((-1) * 8)
True
>>> egcd(23894798501898, 23948178468116)
```

(continues on next page)

(continued from previous page)

```
(2, 2437250447493, -2431817869532)
>>> egcd(pow(2, 50), pow(3, 50))
(1, -260414429242905345185687, 408415383037561)
```

This implementation has been adapted from the algorithms listed below (and subsequently expanded to support any number of integer inputs):

- Extended Euclidean Algorithm on Brilliant.org,
- Modular inverse on Rosetta Code,
- Algorithm Implementation/Mathematics/Extended Euclidean algorithm on Wikibooks, and
- Extended Euclidean algorithm on Wikipedia.

This function can accept any number of integer arguments (as the built-in function `math.gcd` does in Python 3.9 and higher). In general, the greatest common divisor of *all* the arguments is returned (along with coefficients that satisfy the generalized form of the associated identity).

```
>>> egcd(2, 2, 3)
(1, 0, -1, 1)
>>> egcd(13, 16, 17)
(1, 5, -4, 0)
>>> egcd(2, 4, 3, 9)
(1, -1, 0, 1, 0)
>>> 1 == ((-1) * 2) + (0 * 4) + (1 * 3) + (0 * 9)
True
```

This function conforms to the behavior of `math.gcd` when all arguments are `0` (returning `0` as the greatest common divisor) or when any of the arguments are negative (returning the largest *positive* integer that is a divisor of all the arguments).

```
>>> egcd(0, 0)
(0, 1, 0)
>>> egcd(-25, -15)
(5, 1, -2)
>>> egcd(-9, 6, -33, -3)
(3, 0, 0, 0, -1)
```

To conform to the behavior of `math.gcd` in its base cases (in Python 3.9 and higher), this function returns `(0,)` when there are no arguments and the sole argument paired with the coefficient `1` when only one argument is supplied.

```
>>> egcd()
(0,)
>>> egcd(5)
(5, 1)
```

A succinct way to extract the greatest common divisor and the coefficients is to take advantage of Python's support for iterable unpacking via the asterisk notation.

```
>>> bs = (26, 16, 34)
>>> (g, *cs) = egcd(*bs)
>>> (g, cs)
(2, [-3, 5, 0])
```

(continues on next page)

(continued from previous page)

```
>>> g == sum(c * b for (c, b) in zip(cs, bs))
True
```

If an argument is not an integer, an exception is raised.

```
>>> egcd(1.2, 3, 4)
Traceback (most recent call last):
  ...
TypeError: 'float' object cannot be interpreted as an integer
```

```
>>> egcd(1, 2.3)
Traceback (most recent call last):
  ...
TypeError: 'float' object cannot be interpreted as an integer
```

The example below tests the behavior of this function over a range of input pairs using the built-in `math.gcd` function.

```
>>> from math import gcd
>>> from itertools import product
>>> checks = []
>>> for (a, b) in product(range(-1000, 1000), range(-1000, 1000)):
...     (g, s, t) = egcd(a, b)
...     assert(g == gcd(a, b))
...     assert(g == (a * s) + (b * t))
```

The more complex example below tests the behavior of this function over a range of input sequences. The function `gcd_` below is introduced to ensure that the example is compatible with Python 3.7 and 3.8.

```
>>> from functools import reduce
>>> gcd_ = lambda *bs: reduce(gcd, bs, bs[0]) # Backwards-compatible.
>>> checks = []
>>> for k in range(1, 5):
...     for bs in product(*([range(-50 // k, 50 // k)] * k)):
...         (g, *cs) = egcd(*bs)
...         assert(g == gcd_(*bs))
...         assert(g == sum(c * b for (c, b) in zip(cs, bs)))
```

PYTHON MODULE INDEX

e

INDEX