Quaternions

Release 2.0.0

Zach Chartrand

Apr 18, 2023

CONTENTS:

CHAPTER

ONE

INSTALLATION

Install with pip:

```
pip install quaternions-for-python
```

If you want to build from source, clone the GitHub repository:

```
git clone https://github.com/zachartrand/Quaternions.git
```

CHAPTER

TWO

SUPPORT

The best way to help this project is to open an issue on Github.

Github: https://github.com/zachartrand/Quaternions/issues

CHAPTER

THREE

OVERVIEW

3.1 The Quaternion Class

The main aspect of Quaternions for Python is the Quaternion class. The best way to use it is to import it like so:

```
>>> from quaternions import Quaternion
```

To create a quaternion, simply type

```
>>> Quaternion(a, b, c, d)
```

where a, b, c, and d correspond to a quaternion of the form $a + bi + cj + dk$. For example, creating the quaternion $1 - 2i - 3j + 4k$ looks like this in the Python interpreter:

```
>>> q1 = Quaternion(1, -2, -3, 4)
>>> q1
Quaternion(1.0, -2.0, -3.0, 4.0)
>>> print(q1)
(1 - 2i - 3j + 4k)
```

Quaternions have mathematical functionality built in. Adding or multipling two quaternions together uses the same syntax as ints and floats:

```
>>> q1, q2 = Quaternion(1, -2, -3, 4), Quaternion(1, 4, -3, -2)
>>> print(q1)
(1 - 2i - 3j + 4k)
>>> print(q2)
(1 + 4i - 3j - 2k)
>>> print(q1 + q2)
(2 + 2i - 6j + 2k)
>>> print(q1 - q2)
(-6i + 0j + 6k)
>>> print(q2 - q1)
(6i + 0j - 6k)
>>> print(q1 * q2)
(8 + 20i + 6j + 20k)
>>> print(q2 * q1)
(8 - 16i - 18j - 16k)
>>> print(q1/q2)
(-0.19999999999999996 - 0.8i - 0.4j - 0.4k)
>>> print(1/q2 * q1)
```

(continues on next page)

(continued from previous page)

```
(-0.19999999999999996 + 0.4i + 0.4j + 0.8k)
>>> print(q2/q1)
(-0.19999999999999996 + 0.8i + 0.4j + 0.4k)
```

3.2 The qmath module

The qmath module functions similarly to Python's built-in `cmath` module for complex numbers, allowing mathematical functions to be compatible with quaternions. Here are a few examples:

```
>>> from quaternions import Quaternion, qmath
>>>
>>> q = Quaternion(1, -2, -3, 4)
>>> print(q)
(1 - 2i - 3j + 4k)
>>>
>>> print(qmath.exp(q))
(1.6939227236832994 + 0.7895596245415588i + 1.1843394368123383j - 1.5791192490831176k)
>>>
>>> print(qmath.log(q))
(1.7005986908310777 - 0.5151902926640851i - 0.7727854389961277j + 1.0303805853281702k)
>>>
>>> print(qmath.sqrt(q))
(1.7996146219471076 - 0.5556745248702426i - 0.833511787305364j + 1.1113490497404852k)
```

CHAPTER

FOUR

REFERENCE

Main reference for the Quaternions API.

4.1 Quaternions

4.1.1 The Quaternion class

class quaternions.**Quaternion**(*real_component: float = 0.0*, *i_component: float = 0.0*, *j_component: float = 0.0*, *k_component: float = 0.0*)

Quaternions are an expansion of the complex numbers, where there are four (4) components–the real component, also known as the scalar part, and the imaginary components, which together are known as the vector part. The vector part is made up of three (3) components whose unit values are *i*, *j*, and *k*. The rules for these values are as follows:

$$i^2 = j^2 = k^2 = -1$$

$$jk = -kj = i$$

$$ki = -ik = j$$

$$ij = -ji = k,$$

which leads to the following statement:

$$ijk = -1.$$

The descriptions will reference a quaternion of the form $a + bi + cj + dk$, where a, b, c, and d are real numbers.

Parameters

- **real_component** – The real component (a) of the quaternion.
- **i_component** – The i component (b) of the quaternion.
- **j_component** – The j component (c) of the quaternion.
- **k_component** – The k component (d) of the quaternion.

Each component can be returned by calling the attribute of the same name.

Example

```
>>> q = Quaternion(1, -2, -3, 4)
>>> print(q)
(1 - 2i - 3j + 4k)
>>> q.real
1.0
>>> q.i
-2.0
>>> q.j
-3.0
>>> q.k
4.0
```

4.1.2 Methods

Object-based methods

Quaternion.**abs_components**()

Return a list of the absolute values of the components of self.

New in version 2.0.0.

Quaternion.**from_complex**(*z: complex*) → *Quaternion*

Return a Quaternion from a complex number and the vector of self.

If `u == self.unit_vector()`, this is equivalent to

```
Quaternion(z.real, z.imag*u.i, z.imag*u.j, z.imag*u.k)
```

New in version 2.0.0.

Quaternion.**get_imag**() → float

Return the imaginary component of the quaternion if only one of the imaginary components is nonzero. If the quaternion is scalar, return `0.0`. Otherwise, return `None`.

Quaternion.**get_vector_components**() → Tuple[float]

Return the vector components of the Quaternion as a tuple formatted as `(i, j, k)`.

Mathematical methods

Quaternion.**conjugate**() → *Quaternion*

Return the conjugate of self. This is analogous to the complex conjugate, reversing the signs of the vector components.

Quaternion.**inverse**() → *Quaternion*

Return 1/self.

Return the inverse of the quaternion. The inverse of a quaternion is defined as the conjugate divided by the norm squared:

```
q.inverse() = q.conjugate()/(q.norm)**2
```

Quaternion.**squared**() → *Quaternion*

Return self**2.

New in version 2.0.0.

Quaternion.**cos_norm**() → float

Return the cosine of the norm of self.

New in version 2.0.0.

Quaternion.**sin_norm**() → float

Return the sine of the norm of self.

New in version 2.0.0.

Quaternion.**log_norm**() → float

Return the natural logarithm of the norm of self.

This tends to be more accurate than

```
>>> math.log(self.norm)
```

New in version 2.0.0.

Quaternion.**unit_quaternion**() → *Quaternion*

Return the quaternion normalized to magnitude one (1).

If the quaternion is a zero (0) quaternion, return the zero quaternion.

Quaternion.**unit_vector**() → *Quaternion*

Return the vector part of the quaternion normalized to a magnitude of one (1.0). Return the zero quaternion if the magnitude of the quaternion is zero (0.0).

Boolean methods

Quaternion.**is_complex**() → bool

Return `True` if only one of the *i*, *j*, and *k* components is nonzero. Otherwise, return `False`.

Quaternion.**is_scalar**() → bool

Return `True` if the vector components all equal zero. Otherwise, return `False`.

Quaternion.**is_vector**() → bool

Return `True` if the scalar part is zero and at least one of the vector components is nonzero. Otherwise, return `False`.

Quaternion.**is_zero**() → bool

Return `True` if self is the zero quaternion. Otherwise, return `False`.

New in version 2.0.0.

Quaternion.**is_not_zero**() → bool

Return `False` if self is the zero quaternion. Otherwise, return `True`.

New in version 2.0.0.

Class methods

classmethod Quaternion.**from_angle**(*angle: float*, *vector: Iterable[float]*, *norm: Optional[float] = None*, *degrees: bool = True*) → *Quaternion*

Return a quaternion from an angle and vector.

Quaternions can be expressed as `norm*(cos(theta) + u*sin(theta))`, where `u` is a 3D unit vector. This function takes an angle and a vector to create a quaternion. If you want a quaternion with a specific magnitude, you can change the `norm` argument. If no argument is given for *norm*, the resulting quaternion will have a norm equal to the magnitude of *vector*. By default, angles are entered in degrees. If you want to enter an angle in radians, set `degrees` to False.

Properties

property Quaternion.**angle**: float

The angle of the quaternion in radians.

property Quaternion.**angle_in_radians**

Same as *Quaternion.angle*

property Quaternion.**angle_in_degrees**: float

The angle of the quaternion in degrees.

property Quaternion.**components**: Tuple[float]

The components of the quaternion as a tuple in the order `(real, i, j, k)`.

property Quaternion.**norm**: float

The norm (magnitude) of the quaternion.

property Quaternion.**scalar**: *Quaternion*

The real part of the quaternion.

property Quaternion.**vector**: *Quaternion*

The vector part of the quaternion.

property Quaternion.**vector_norm**: float

The norm of the vector part of the quaternion.

property Quaternion.**versor**: *Quaternion*

The quaternion normalized to a magnitude of one (1).

4.2 The qmath module

4.2.1 qmath

Similar to the built-in module `cmath`, this module has definitions of mathematical functions expanded to work with quaternions.

4.2.2 Quaternion Boolean functions

quaternions.qmath.**isclose**(*a*, *b*, ***, *rel_tol=1e-09*, *abs_tol=1e-09*) → bool

Determine whether two Quaternions are close in value.

For the values to be considered close, the difference between them must be smaller than at least one of the tolerances.

-inf, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and -inf are only close to themselves.

Parameters

- **a** (Quaternion) – The first Quaternion.
- **b** (Quaternion) – The second Quaternion.
- **rel_tol** (*float*) – maximum difference for being considered "close", relative to the magnitude of the input values
- **abs_tol** (*float*) – maximum difference for being considered "close", regardless of the magnitude of the input values

Returns

True if a is close in value to b,
and False otherwise.

Return type
bool

New in version 2.0.0.

quaternions.qmath.**isfinite**(*q:* Quaternion) → bool

Return True if all components of q are finite, and False otherwise.

New in version 2.0.0.

quaternions.qmath.**isinf**(*q:* Quaternion) → bool

Return True if any component of q is an infinity, and False otherwise.

New in version 2.0.0.

quaternions.qmath.**isnan**(*q:* Quaternion) → bool

Return True if any component of q is a NaN, and False otherwise.

New in version 2.0.0.

4.2.3 Quaternion mathematical functions

quaternions.qmath.**exp**(*q:* Quaternion) → *Quaternion*

Return the exponential of a quaternion.

quaternions.qmath.**log**(*q:* Quaternion, *base:* Quaternion = *2.718281828459045*) → *Quaternion*

Return the logarithm of a quaternion to the given base.

If the base is not specified, returns the natural logarithm (base *e*) of the quaternion.

quaternions.qmath.**log10**(*q:* Quaternion) → *Quaternion*

Return the base-10 logarithm of the quaternion.

quaternions.qmath.**sqrt**(*q:* Quaternion) → *Quaternion*

Return the square root of the quaternion.

quaternions.qmath.**cos**(*q:* Quaternion) → *Quaternion*

Return the cosine of the quaternion.

New in version 2.0.0.

quaternions.qmath.**cosh**(*q:* Quaternion) → *Quaternion*

Return the hyperbolic cosine of q.

New in version 2.0.0.

quaternions.qmath.**sin**(*q:* Quaternion) → *Quaternion*

Return the sine of the quaternion.

New in version 2.0.0.

quaternions.qmath.**sinh**(*q:* Quaternion) → *Quaternion*

Return the hyperbolic sine of q.

New in version 2.0.0.

quaternions.qmath.**tan**(*q:* Quaternion) → *Quaternion*

Return the tangent of the quaternion.

New in version 2.0.0.

quaternions.qmath.**tanh**(*q:* Quaternion) → *Quaternion*

Return the hyperbolic tangent of q.

New in version 2.0.0.

4.2.4 Rotation functions

quaternions.qmath.**rotate3d**(*point: Iterable[float]*, *angle: float*, *axis: Iterable[float] = (0.0, 0.0, 1.0)*, *rounding: int = -1*, *degrees: bool = True*) → Tuple[float]

Rotate a point around an axis.

Take a point in 3d space represented as a tuple or list of three (3) values and rotate it by an angle around a given axis vector.

Parameters

- **point** – The point to rotate. The format for the coordinates is `(x, y, z)`.
- **angle** – The angle of rotation. By default, angle is set to be input in degrees. See the **degrees** parameter if you want to use radians instead.
- **axis** – The axis to rotate the point around. By default, this is the z-axis `(0, 0, 1)`.
- **rounding** – The number of decimal points the result will be rounded to. Default value is -1, which does not round the end result.
- **degrees** – When set to `True`, this function interprets the parameter **angle** as degrees. Set this parameter to `False` to use angles in radians. Default is `True`.

For the point and axis parameters, if only one value is given, the value will be assumed to be an x-coordinate with the y- and z-coordinates equal to zero (0). If two values are given, they will be assumed to be x- and y-coordinates with the z-coordinate equal to zero (0).

quaternions.qmath.**rotate_Euler**(*point: Iterable[float]*, *yaw: float*, *pitch: float*, *roll: float*, *x_axis: Iterable[float] = (1.0, 0.0, 0.0)*, *z_axis: Iterable[float] = (0.0, 0.0, 1.0)*, *degrees: bool = True*) → Tuple[float]

Rotate a given point using Euler angles.

This function uses the rotation convention of z-y'-x", rotating yaw, then pitch, then roll.

Parameters

- **point** – The point to rotate. The format for the coordinates is `(x, y, z)`.
- **yaw** – The angle of rotation around the z-axis.
- **pitch** – The angle of rotation around the y'-axis. The y'-axis is the y-axis after the yaw rotation has been applied.
- **roll** – The angle of rotation around the x"-axis. The x"-axis is the x-axis after both the yaw and pitch rotations.
- **x_axis** – The initial x-axis of the coordinate system that **point** belongs to. Default value is `(1, 0, 0)`.
- **z_axis** – The initial z-axis of the coordinate system that **point** belongs to. Default value is `(0, 0, 1)`.
- **degrees** – When set to `True`, this function interprets the angle parameters as degrees. Set this parameter to `False` to use angles in radians. Default is `True`.

New in version 1.1.0.

4.2.5 Vector functions

quaternions.qmath.**cross_product**(*vector1: Iterable[float]*, *vector2: Iterable[float]*) → Tuple[float]

Return the cross product of two vectors.

Because this uses quaternions to calculate, this only works for vectors up to three (3) dimensions.

New in version 1.1.0.

quaternions.qmath.**dot_product**(*vector1: Iterable[float]*, *vector2: Iterable[float]*) → float

Return the dot product of two vectors.

Because this uses quaternions to calculate, this only works for vectors up to three (3) dimensions.

New in version 1.1.0.

4.2.6 Constants

quaternions.qmath.**pi**

The mathematical constant , as a float.

quaternions.qmath.**tau**

The mathematical constant , as a float.

quaternions.qmath.**e**

The mathematical constant *e*, as a float.

quaternions.qmath.**inf**

Floating-point positive infinity. Equivalent to `float('inf')`.

quaternions.qmath.**infi**

Quaternion with positive infinity i part and zero for all the other parts. Equivalent to `Quaternion(0, float('inf'), 0, 0)`.

quaternions.qmath.**infj**

Quaternion with positive infinity j part and zero for all the other parts. Equivalent to `Quaternion(0, 0, float('inf'), 0)`.

quaternions.qmath.**infk**

Quaternion with positive infinity k part and zero for all the other parts. Equivalent to `Quaternion(0, 0, 0, float('inf'))`.

quaternions.qmath.**nan**

A floating-point "not a number" (NaN) value. Equivalent to `float('nan')`.

quaternions.qmath.**nani**

Quaternion with NaN i part and zero for all the other parts. Equivalent to `Quaternion(0, float('nan'), 0, 0)`.

quaternions.qmath.**nanj**

Quaternion with NaN j part and zero for all the other parts. Equivalent to `Quaternion(0, 0, float('nan'), 0)`.

quaternions.qmath.**nank**

Quaternion with NaN k part and zero for all the other parts. Equivalent to `Quaternion(0, 0, 0, float('nan'))`.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

T

U

V