rowan Documentation

Release 0.5.0

Vyas Ramasubramani

Feb 12, 2019

Contents:

Welcome to the documentation for rowan, a package for working with quaternions! Quaternions form a number system with various interesting properties, and they have a number of uses. This package provides tools for standard algebraic operations on quaternions as well as a number of additional tools for *e.g.* measuring distances between quaternions, interpolating between them, and performing basic point-cloud mapping. A particular focus of the rowan package is working with unit quaternions, which are a popular means of representing rotations in 3D. In order to provide a unified framework for working with the various rotation formalisms in 3D, rowan allows easy interconversion between these formalisms.

To install rowan, first clone the repository from source. Once installed, the package can be installed using setuptools:

```
$ python setup.py install --user
```

CHAPTER 1

rowan

Overview

rowan.conjugate	Conjugates an array of quaternions
rowan.inverse	Computes the inverse of an array of quaternions
rowan.exp	Computes the natural exponential function e^q.
rowan.expb	Computes the exponential function b^q.
rowan.exp10	Computes the exponential function 10^q.
rowan.log	Computes the quaternion natural logarithm.
rowan.logb	Computes the quaternion logarithm to some base b.
rowan.log10	Computes the quaternion logarithm base 10.
rowan.multiply	Multiplies two arrays of quaternions
rowan.divide	Divides two arrays of quaternions
rowan.norm	Compute the quaternion norm
rowan.normalize	Normalize quaternions
rowan.rotate	Rotate a list of vectors by a corresponding set of quaternions
rowan.vector_vector_rotation	Find the quaternion to rotate one vector onto another
rowan.from_euler	Convert Euler angles to quaternions
rowan.to_euler	Convert quaternions to Euler angles
rowan.from_matrix	Convert the rotation matrices mat to quaternions
rowan.to_matrix	Convert quaternions into rotation matrices.
rowan.from_axis_angle	Find quaternions to rotate a specified angle about a specified axis
rowan.to_axis_angle	Convert the quaternions in q to axis angle representations
rowan.from_mirror_plane	Generate quaternions from mirror plane equations.
rowan.reflect	Reflect a list of vectors by a corresponding set of quaternions
rowan.equal	Check whether two sets of quaternions are equal.

Continued on next page

Table 1 – continued from previous page

rowan.not_equal	Check whether two sets of quaternions are not equal.
rowan.isfinite	Test element-wise for finite quaternions.
rowan.isinf	Test element-wise for infinite quaternions.
rowan.isnan	Test element-wise for NaN quaternions.

Details

The core *rowan* package contains functions for operating on quaternions. The core package is focused on robust implementations of key functions like multiplication, exponentiation, norms, and others. Simple functionality such as addition is inherited directly from numpy due to the representation of quaternions as numpy arrays. Many core numpy functions implemented for normal arrays are reimplemented to work on quaternions (such as *allclose()* and *isfinite()*). Additionally, numpy broadcasting is enabled throughout rowan unless otherwise specified. This means that any function of 2 (or more) quaternions can take arrays of shapes that do not match and return results according to numpy's broadcasting rules.

rowan.**conjugate**(*q*)

Conjugates an array of quaternions

Parameters **q** (*(..,4) np.array*) – Array of quaternions

Returns An array containing the conjugates of q

Example:

```
q_star = conjugate(q)
```

rowan.**exp**(*q*)

Computes the natural exponential function e^q.

The exponential of a quaternion in terms of its scalar and vector parts $q = a + \boldsymbol{v}$ is defined by exponential power series: formula $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ as follows:

$$e^q = e^{a+v} \tag{1.1}$$

$$= e^a \left(\sum_{k=0}^{\infty} \frac{v^k}{k!} \right) \tag{1.2}$$

$$= e^a \left(\cos||\boldsymbol{v}|| + \frac{\boldsymbol{v}}{||\boldsymbol{v}||} \sin||\boldsymbol{v}|| \right) \tag{1.3}$$

Parameters **q** (*(..,4) np.array*) – Quaternions

Returns Array of shape (...) containing exponentials of q

Example:

```
q_exp = exp(q)
```

rowan.**expb**(*q*, *b*)

Computes the exponential function b^q.

We define the exponential of a quaternion to an arbitrary base relative to the exponential function e^q using the change of base formula as follows:

$$b^q = y \tag{1.4}$$

$$q = \log_b y = \frac{\ln y}{\ln b} \tag{1.5}$$

$$y = e^{q \ln b} \tag{1.6}$$

Parameters **q** (*(.., 4) np.array*) – Quaternions

Returns Array of shape (...) containing exponentials of q

Example:

```
q_exp = exp(q, 2)
```

rowan.**exp10**(*q*)

Computes the exponential function 10^q.

Wrapper around *expb()*.

Parameters **q** (*(.., 4) np.array*) – Quaternions

Returns Array of shape (...) containing exponentials of q

Example:

```
q_exp = exp(q, 2)
```

rowan.**log**(*q*)

Computes the quaternion natural logarithm.

The natural of a quaternion in terms of its scalar and vector parts $q = a + \boldsymbol{v}$ is defined by inverting the exponential formula (see *exp()*), and is defined by the formula :math:‘ frac{x^k}{k!}‘ as follows:

$$\ln(q) = \ln||q|| + \frac{\boldsymbol{v}}{||\boldsymbol{v}||} \arccos\left(\frac{a}{q}\right) \quad (1.7)$$

Parameters **q** (*(.., 4) np.array*) – Quaternions

Returns Array of shape (...) containing logarithms of q

Example:

```
ln_q = log(q)
```

rowan.**logb**(*q*, *b*)

Computes the quaternion logarithm to some base b.

The quaternion logarithm for arbitrary bases is defined using the standard change of basis formula relative to the natural logarithm.

$$\log_b q = y \quad (1.8)$$
$$q = b^y \quad (1.9)$$
$$\ln q = y \ln b \quad (1.10)$$
$$y = \log_b q = \frac{\ln q}{\ln b} \quad (1.11)$$

Parameters

- **q** (*(.., 4) np.array*) – Quaternions
- **n** (*(..) np.array*) – Scalars to use as log bases

Returns Array of shape (...) containing logarithms of q

Example:

```
log_q = log(q, 2)
```

rowan.**log10**(*q*)

Computes the quaternion logarithm base 10.

Wrapper around *logb()*.

Parameters **q** (*(..,4) np.array*) – Quaternions

Returns Array of shape (...) containing logarithms of q

Example:

```
log_q = log(q, 2)
```

rowan.**power**(*q*, *n*)

Computes the power of a quaternion q^n.

Quaternions raised to a scalar power are defined according to the polar decomposition angle θ and vector $\hat{u}$: $q^n = ||q||^n \left(\cos(n\theta) + \hat{u}\sin(n\theta)\right)$. However, this can be computed more efficiently by noting that $q^n = \exp(n\ln(q))$.

Parameters

- **q** (*(..,4) np.array*) – Quaternions.
- **n** (*(..) np.arrray*) – Scalars to exponentiate quaternions with.

Returns Array of shape (...) containing of q

Example:

```
q_n = pow(q^n)
```

rowan.**multiply**(*qi*, *qj*)

Multiplies two arrays of quaternions

Note that quaternion multiplication is generally non-commutative.

Parameters

- **qi** (*(..,4) np.array*) – First set of quaternions
- **qj** (*(..,4) np.array*) – Second set of quaternions

Returns An array containing the products of row i of qi with column j of qj

Example:

```
qi = np.array([[1, 0, 0, 0]])
qj = np.array([[1, 0, 0, 0]])
prod = multiply(qi, qj)
```

rowan.**norm**(*q*)

Compute the quaternion norm

Parameters **q** (*(..,4) np.array*) – Quaternions to find norms for

Returns An array containing the norms for qi in q

Example:

```
q = np.random.rand(10, 4)
norms = norm(q)
```

rowan.**normalize**(*q*)

Normalize quaternions

Parameters **q** (*(..,4) np.array*) – Array of quaternions to normalize

Returns An array containing the unit quaternions q/norm(q)

Example:

```
q = np.random.rand(10, 4)
u = normalize(q)
```

rowan.**from_mirror_plane**(*x*, *y*, *z*)

Generate quaternions from mirror plane equations.

Reflection quaternions can be constructed of the from $(0, x, y, z)$, *i.e.* with zero real component. The vector (x, y, z) is the normal to the mirror plane.

Parameters

- **x** (*(..) np.array*) – First planar component
- **y** (*(..) np.array*) – Second planar component
- **z** (*(..) np.array*) – Third planar component

Returns An array of quaternions corresponding to the provided reflections.

Example:

```
plane = (1, 2, 3)
quat_ref = from_mirror_plane(*plane)
```

rowan.**reflect**(*q*, *v*)

Reflect a list of vectors by a corresponding set of quaternions

For help constructing a mirror plane, see *from_mirror_plane()*.

Parameters

- **q** (*(..,4) np.array*) – Quaternions to use for reflection
- **v** (*(..,3) np.array*) – Vectors to reflect.

Returns An array of the vectors in v reflected by q

Example:

```
q = np.random.rand(1, 4)
v = np.random.rand(1, 3)
v_rot = rotate(q, v)
```

rowan.**rotate**(*q*, *v*)

Rotate a list of vectors by a corresponding set of quaternions

Parameters

- **q** (*(..,4) np.array*) – Quaternions to rotate by.
- **v** (*(..,3) np.array*) – Vectors to rotate.

Returns An array of the vectors in v rotated by q

Example:

```
q = np.random.rand(1, 4)
v = np.random.rand(1, 3)
v_rot = rotate(q, v)
```

rowan.**vector_vector_rotation**(*v1*, *v2*)

Find the quaternion to rotate one vector onto another

Parameters

- **v1** (*(..,3) np.array*) – Vector to rotate
- **v2** (*(..,3) np.array*) – Desired vector

Returns Array (..., 4) of quaternions that rotate v1 onto v2.

rowan.**from_euler**(*alpha*, *beta*, *gamma*, *convention='zyx'*, *axis_type='intrinsic'*)

Convert Euler angles to quaternions

For generality, the rotations are computed by composing a sequence of quaternions corresponding to axis-angle rotations. While more efficient implementations are possible, this method was chosen to prioritize flexibility since it works for essentially arbitrary Euler angles as long as intrinsic and extrinsic rotations are not intermixed.

Parameters

- **alpha** (*(..) np.array*) – Array of α values in radians.
- **beta** (*(..) np.array*) – Array of β values in radians.
- **gamma** (*(..) np.array*) – Array of γ values in radians.
- **convention** (*str*) – One of the 12 valid conventions xzx, xyx, yxy, yzy, zyz, zxz, xzy, xyz, yxz, yzx, zyx, zxy
- **axes** (*str*) – Whether to use extrinsic or intrinsic rotations

Returns An array containing the converted quaternions

Example:

```
rands = np.random.rand(100, 3)
alpha, beta, gamma = rands.T
ql.from_euler(alpha, beta, gamma)
```

rowan.**to_euler**(*q*, *convention='zyx'*, *axis_type='intrinsic'*)

Convert quaternions to Euler angles

Euler angles are returned in the sequence provided, so in, *e.g.*, the default case ('zyx'), the angles returned are for a rotation $Z(\alpha)Y(\beta)X(\gamma)$.

Note: In all cases, the α and γ angles are between $\pm\pi$. For proper Euler angles, β is between 0 and pi degrees. For Tait-Bryan angles, β lies between $\pm\pi/2$.

For simplicity, quaternions are converted to matrices, which are then converted to their Euler angle representations. All equations for rotations are derived by considering compositions of the three elemental rotations about

the three Cartesian axes:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$
$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 1 & \cos\theta \end{pmatrix}$$
$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Extrinsic rotations are represented by matrix multiplications in the proper order, so $z - y - x$ is represented by the multiplication XYZ so that the system is rotated first about Z, then about y, then finally X. For intrinsic rotations, the order of rotations is reversed, meaning that it matches the order in which the matrices actually appear *i.e.* the $z - y' - x''$ convention (yaw, pitch, roll) corresponds to the multiplication of matrices ZYX. For proof of the relationship between intrinsic and extrinsic rotations, see the Wikipedia page on Davenport chained rotations.

For more information, see the Wikipedia page for Euler angles (specifically the section on converting between representations).

Parameters

- **q** (*(.., 4) np.array*) – Quaternions to transform
- **convention** (*str*) – One of the 6 valid conventions zxz, xyx, yzy, zyz, xzx, yxy
- **axes** (*str*) – Whether to use extrinsic or intrinsic

Returns An array with Euler angles (α, β, γ) as the last dimension (in radians)

Example:

```
rands = np.random.rand(100, 3)
alpha, beta, gamma = rands.T
ql.from_euler(alpha, beta, gamma)
alpha_return, beta_return, gamma_return = ql.to_euler(full)
```

rowan.**from_matrix**(*mat*, *require_orthogonal=True*)

Convert the rotation matrices mat to quaternions

Thhis method uses the algorithm described by Bar-Itzhack in *[Itzhack00]*. The idea is to construct a matrix K whose largest eigenvalue corresponds to the desired quaternion. One of the strengths of the algorithm is that for nonorthogonal matrices it gives the closest quaternion representation rather than failing outright.

Parameters **mat** (*(.., 3, 3) np.array*) – An array of rotation matrices

Returns An array containing the quaternion representations of the elements of mat (i.e. the same elements of SO(3))

rowan.**to_matrix**(*q*, *require_unit=True*)

Convert quaternions into rotation matrices.

Uses the conversion described on Wikipedia.

Parameters **q** (*(.., 4) np.array*) – An array of quaternions

Returns The array containing the matrix representations of the elements of q (i.e. the same elements of SO(3))

rowan.**from_axis_angle**(*axes*, *angles*)
Find quaternions to rotate a specified angle about a specified axis

Parameters

- **axes** (*(..,3) np.array*) – An array of vectors (the axes)
- **angles** (*float or (..,1) np.array*) – An array of angles in radians. Will be broadcast to match shape of v as needed

Returns An array of the desired rotation quaternions

Example:

```
import numpy as np
axis = np.array([[1, 0, 0]])
ang = np.pi/3
quat = about_axis(axis, ang)
```

rowan.**to_axis_angle**(*q*)
Convert the quaternions in q to axis angle representations

Parameters **q** (*(..,4) np.array*) – An array of quaternions

Returns A tuple of np.arrays (axes, angles) where axes has shape (...,3) and angles has shape (...,1). The angles are in radians

rowan.**isnan**(*q*)
Test element-wise for NaN quaternions.

A quaternion is defined as NaN if any elements are NaN.

Parameters **q** (*(..,4) np.array*) – Quaternions to check

Returns A boolean array of shape (...) indicating NaN.

rowan.**isinf**(*q*)
Test element-wise for infinite quaternions.

A quaternion is defined as infinite if any elements are infinite.

Parameters **q** (*(..,4) np.array*) – Quaternions to check

Returns A boolean array of shape (...) indicating infinite quaternions.

rowan.**isfinite**(*q*)
Test element-wise for finite quaternions.

A quaternion is defined as finite if all elements are finite.

Parameters **q** (*(..,4) np.array*) – Quaternions to check

Returns A boolean array of shape (...) indicating finite quaternions.

rowan.**equal**(*p*, *q*)
Check whether two sets of quaternions are equal.

This function is a simple wrapper that checks array equality and then aggregates along the quaternion axis.

Parameters

- **p** (*(..,4) np.array*) – First set of quaternions
- **q** (*(..,4) np.array*) – First set of quaternions

Returns A boolean array of shape (...) indicating equality.

rowan.**not_equal**(*p*, *q*)
Check whether two sets of quaternions are not equal.

This function is a simple wrapper that checks array equality and then aggregates along the quaternion axis.

Parameters

- **p** (*(..,4) np.array*) – First set of quaternions
- **q** (*(..,4) np.array*) – First set of quaternions

Returns A boolean array of shape (...) indicating inequality.

rowan.**allclose**(*p*, *q*, ***kwargs*)
Check whether two sets of quaternions are all close.

This is a direct wrapper of the corresponding numpy function.

Parameters

- **p** (*(..,4) np.array*) – First set of quaternions
- **q** (*(..,4) np.array*) – First set of quaternions
- ****kwargs** – Keyword arguments to pass to np.allclose

Returns Whether or not all quaternions are close

rowan.**isclose**(*p*, *q*, ***kwargs*)
Element-wise check of whether two sets of quaternions close.

This function is a simple wrapper that checks using the corresponding numpy function and then aggregates along the quaternion axis.

Parameters

- **p** (*(..,4) np.array*) – First set of quaternions
- **q** (*(..,4) np.array*) – First set of quaternions
- ****kwargs** – Keyword arguments to pass to np.isclose

Returns A boolean array of shape (...)

rowan.**inverse**(*q*)
Computes the inverse of an array of quaternions

Parameters **q** (*(..,4) np.array*) – Array of quaternions

Returns An array containing the inverses of q

Example:

```
q_inv = inverse(q)
```

rowan.**divide**(*qi*, *qj*)
Divides two arrays of quaternions

Division is non-commutative; this function returns $q_i q_j^{-1}$.

Parameters

- **qi** (*(..,4) np.array*) – Dividend quaternion
- **qj** (*(..,4) np.array*) – Divisors quaternions

Returns An array containing the quotients of row i of qi with column j of qj

Example:

```
qi = np.array([[1, 0, 0, 0]])
qj = np.array([[1, 0, 0, 0]])
prod = divide(qi, qj)
```

CHAPTER 2

random

Overview

rowan.random.rand	Generate random rotations uniformly
rowan.random.random_sample	Generate random rotations unifo

Details

Various functions for generating random sets of rotation quaternions. Note that if you simply want random quaternions not restricted to $SO(3)$ you can just generate these directly using *numpy.random.rand(... 4)*. This subpackage is entirely focused on generating rotation quaternions.

rowan.random.**rand**(**args*)

Generate random rotations uniformly

This is a convenience function *a la* np.random.rand. If you want a function that takes a tuple as input, use *random_sample()* instead.

Parameters **shape** (*tuple*) – The shape of the array to generate.

Returns Random quaternions of the shape provided with an additional axis of length 4.

rowan.random.**random_sample**(*size=None*)

Generate random rotations unifo

In general, sampling from the space of all quaternions will not generate uniform rotations. What we want is a distribution that accounts for the density of rotations, *i.e.*, a distribution that is uniform with respect to the appropriate measure. The algorithm used here is detailed in *[Shoe92]*.

Parameters **size** (*tuple*) – The shape of the array to generate

Returns Random quaternions of the shape provided with an additional axis of length 4

CHAPTER 3

Development Guide

3.1 Philosophy

The goal of rowan is to provide a flexible, easy-to-use, and scalable approach to dealing with rotation representations. To ensure maximum flexibility, rowan operates entirely on numpy arrays, which serve as the *de facto* standard for efficient multi-dimensional arrays in Python. To be available for a wide variety of applications, rowan aims to work for arbitrarily shaped numpy arrays, mimicking numpy broadcasting to the extent possible. Functions for which this broadcasting is not available should be documented as such.

Since rowan is designed to work everywhere, all hard dependencies aside from numpy are avoided, although soft dependencies for specific functions are allowed. To avoid any dependencies on compilers or other software, all rowan code is written in **pure Python**. This means that while rowan is intended to provide good performance, it may not be the correct choice in cases where performance is critical. The package was written principally for use-cases where quaternion operations are not the primary bottleneck, so it prioritizes portability, maintainability, and flexibility over optimization.

3.1.1 PEP 20

In general, all code in rowan should follow the principles in PEP 20. In particular, prefer simple, explicit code where possible, avoiding unnecessary convolution or complicated code that could be written more simply. Avoid writing code that is not easy to parse up front.

Inline comments are **highly encouraged**; however, code should be written in a way that it could be understood without comments. Comments such as "Set x to 10" are not helpful and simply clutter code. The most useful comments in a package such as rowan are the ones that explain the underlying algorithm rather than the implementations, which should be simple. For example, the comment "compute the spectral decomposition of A" is uninformative, since the code itself should make this obvious, *e.g*, `np.linalg.eigh`. On the other hand, the comment "the eigenvector corresponding to the largest eigenvalue of the A matrix is the quaternion" is instructive.

3.2 Source Code Conventions

All code in rowan should follow PEP 8 guidelines, which are the *de facto* standard for Python code. In addition, follow the Google Python Style Guide, which is largely a superset of PEP 8. Note that Google has amended their standards to match PEP 8's 4 spaces guideline, so write code accordingly. In particular, write docstrings in the Google style.

Python example:

```
# This is the correct style
def multiply(x, y):
    """Multiply two numbers

    Args:
        x (float): The first number
        y (float): The second number

    Returns:
        The product
    """

# This is the incorrect style
def multiply(x, y):
    """Multiply two numbers

    :param x: The first number
    :type x: float
    :param y: The second number
    :type y: float
    :returns: The product
    :rtype: float
    """
```

Documentation must be included for all files, and is then generated from the docstrings using sphinx.

3.3 Unit Tests

All code should include a set of unit tests which test for correct behavior. All tests should be placed in the `tests` folder at the root of the project. These tests should be as simple as possible, testing a single function each, and they should be kept as short as possible. Tests should also be entirely deterministic: if you are using a random set of objects for testing, they should either be generated once and then stored in the `tests/files` folder, or the random number generator in use should be seeded explicitly (*e.g*, *numpy.random.seed* or *random.seed*). Tests should be written in the style of the standard Python unittest framework. At all times, tests should be executable by simply running *python -m unittest discover tests* from the root of the project.

3.4 General Notes

- For consistency, NumPy should **always** be imported as `np` in code: `import numpy as np`.
- Avoid external dependencies where possible, and avoid introducing **any** hard dependencies. Dependencies other than NumPy should always be soft, enabling the rest of the package to function as is.

CHAPTER 4

License

```
rowan Open Source Software License Copyright 2010-2018 The Regents of
the University of Michigan All rights reserved.

rowan may contain modifications ("Contributions") provided, and to which
copyright is held, by various Contributors who have granted The Regents of the
University of Michigan the right to modify and/or distribute such Contributions.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
   may be used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 5

Changelog

The format is based on Keep a Changelog. This project adheres to *Semantic Versioning* <http://semver.org/spec/v2.0.0.html>‘_.

5.1 Unreleased

5.1.1 Added

- Various distance metrics on quaternion space.
- Quaternion interpolation.

5.1.2 Fixed

- Update empty __all__ variable in geometry to export functions.

5.2 v0.4.4 - 2018-04-10

5.2.1 Added

- Rewrote internals for upload to PyPI.

5.3 v0.4.3 - 2018-04-10

5.3.1 Fixed

- Typos in documentation.

5.4 v0.4.2 - 2018-04-09

5.4.1 Added

- Support for Read The Docs and Codecov.
- Simplify CircleCI testing suite.
- Minor changes to README.
- Properly update this document.

5.5 v0.4.1 - 2018-04-08

5.5.1 Fixed

- Exponential for bases other than e are calculated correctly.

5.6 v0.4.0 - 2018-04-08

5.6.1 Added

- Add functions relating to exponentiation: exp, expb, exp10, log, logb, log10, power.
- Add core comparison functions for equality, closeness, finiteness.

5.7 v0.3.0 - 2018-03-31

5.7.1 Added

- Broadcasting works for all methods.
- Quaternion reflections.
- Random quaternion generation.

5.7.2 Changed

- Converting from Euler now takes alpha, beta, and gamma as separate args.
- Ensure more complete coverage.

5.8 v0.2.0 - 2018-03-08

5.8.1 Added

- Added documentation.

- Add tox support.
- Add support for range of python and numpy versions.
- Add coverage support.

5.8.2 Changed

- Clean up CI.
- Ensure pep8 compliance.

5.9 v0.1.0 - 2018-02-26

5.9.1 Added

- Initial implementation of all functions.

CHAPTER 6

Credits

The following people contributed to the *rowan* package.

Vyas Ramasubramani, University of Michigan - **Lead developer**.

- Initial design
- Core quaternion operations
- Sphinx docs support

CHAPTER 7

Support and Contribution

This package is hosted on Bitbucket. Please report any bugs or problems that you find on the issue tracker.

All contributions to rowan are welcomed! Please see the *development guide* for more information.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[Itzhack00] Itzhack Y. Bar-Itzhack. “New Method for Extracting the Quaternion from a Rotation Matrix”, Journal of Guidance, Control, and Dynamics, Vol. 23, No. 6 (2000), pp. 1085-1087 https://doi.org/10.2514/2.4654

[Shoe92] Shoemake, K.: Uniform random rotations. In: D. Kirk, editor, Graphics Gems III, pages 124-132. Academic, New York, 1992.

Python Module Index

r

Index