qmath Quick Card

Marco Abrate
abrate.m@gmail.com

Contents

1 Classes

`class quaternion`(*attitude*)
Quaternion class.
attitude can be:
- Number (of any type, complex are included);
- list or numpy array of the components with respect to `1,i,j` and `k`;
- string of the form `'a+bi+cj+dk'`;
- a rotation about an axis using pairs (rotation angle, axis of rotation);
- list whose components are Euler angles;
- a 3X3 rotation matrix.

2 Methods

`__repr__`(*self*)
Quaternions are represented in the algebraic way: $q = a + bi + cj + dk$, and a, b, c, d are floats.

`__getitem__`(*self*, *key*)
Returns one of the four float components of the quaternion. This method allows to get the components by `quaternion[key]`.

`__setitem__`(*self*, *key*, *number*)
Set one of the four float components of the quaternion. This method allows to set the components by `quaternion[key] = number`.

`__delitem__`(*self*, *key*)
Delete (set to zero) one of the four float components of the quaternion. This method allows to write `del quaternion[key]`.

__delslice__($self, key_1, key_2$)
Delete (set to zero) the components of the quaternion from the $key_1 - th$ to the $key_2 - th$. This method allows to write `del quaternion[1:3]`.

__contains__($self, key$)
Returns 0 if the component of the quaternion with respect to key is zero, 1 otherwise. This method allows to write `del 'k' in quaternion`.

__eq__($self, other$)
Returns `True` if two quaternion are equal, `False` otherwise. This method allows to write `quaternion1 == quaternion2`.

__ne__($self, other$)
The opposite of __eq__. This method allows to write `quaternion1 != quaternion2`.

__iadd__($self, other$)
__isub__($self, other$)
__imul__($self, other$)
__idiv__($self, other$)
These methods are called to implement the augmented arithmetic assignments. These methods should attempt to do the operation in-place (modifying self) and return the result.

__add__($self, other$)
__sub__($self, other$)
__mul__($self, other$)
__div__($self, other$)
These methods are called to implement the binary arithmetic operations. For instance, to evaluate the expression `x + y`, where `x` is a quaternion, `x.__add__(y)` is called. `y` can either be a quaternion or something that can be converted to quaternion.

__rmul__($self, other$)
__rdiv__($self, other$)
These methods are called to implement the binary arithmetic operations with reflected operands.

__neg__($self$)
Return the opposite of a quaternion. You can write `- quaternion`.

__pow__($self, exponent$)
Implements the operator `**`. The power of a quaternion can be computed for integer power (both positive or negative) and also if the exponent is half or third a number: for example square or cube roots are evaluated.

__abs__($self$)
Returns the modulus of the quaternion.

__equal__(*self*, *other*[, *tolerance*])
Returns quaternion equality with arbitrary tolerance. If no tolerance is admitted it is the same as __eq__(*self*, *other*). If `tol` is the tolerance, this method lets you to write `quaternion1 == quaternion2|tol`.

real(*self*)
Returns the real part of the quaternion.

imag(*self*)
Returns the imaginary part of the quaternion.

trace(*self*)
Returns the trace of the quaternion (the double of its real part).

conj(*self*)
Returns the conjugate of the quaternion.

norm(*self*)
Returns the norm of the quaternion (the square of the modulus).

delta(*self*)
Returns the δ of the quaternion, that is the opposite of the norm of the imaginary part of the quaternion.

inverse(*self*)
Quaternionic inverse, if it exists. It is the same as `quaternion ** (-1)`.

unitary(*self*)
Returns the normalized quaternion, if different from zero.

sqrt(*self*)
Computes the square root of the quaternion. If the quaternion has only two roots, the one with positive trace is given: if this method returns r, also -r is a root.

croot(*self*)
Computes the cube root (unique) of a quaternion.

QuaternonToRotation(*self*)
Converts the quaternion, if unitary, into a rotation matrix.

3 Additional Functions

real(*object*)
The same as `object.real`().

imag(*object*)
The same as `object.imag`().

trace(*object*)
The same as `object.trace`().

`conj`(*object*)
The same as `object.conj`().

`norm`(*object*)
The same as `object.norm`().

`delta`(*object*)
The same as `object.delta`().

`inverse`(*object*)
The same as `object.inverse`().

`unitary`(*object*)
The same as `object.unitary`().

`sqrt`(*object*)
The same as `object.sqrt`().

`croot`(*object*)
The same as `object.croot`().

`QuaternonToRotation`(*object*)
The same as `object.QuaternioinToRotation`().

`RotationToQuaternion`(*angle*, *vector*)
Converts a pair angle-vector into a quaternion.

`StringToQuaternion`(*string*)
Converts a string into a quaternion.

`MatrixToEuler`(*matrix*)
Converts a 3X3 matrix into a vector having Euler angles as components.

`EulerToQuaternion`(*list*)
Converts a vector whose components are Euler angles into a quaternion.

`identity`()
Returns 1 as a quaternion.

`identity`()
Returns 0 as a quaternion.

`dot`($object_1, object_2$)
Returns the dot product of two quaternions.

`CrossRatio`(a, b, c, d)
Returns the cross ratio of four quaternions defined by:

$$\texttt{CrossRatio}(a,b,c,d) = (a-c)\cdot(a-d)^{-1}\cdot(b-d)\cdot(b-c)^{-1}.$$

If $a = d$ or $b = c$ returns the string `'Infinity'`.

`Moebius`(z, a, b, c, d)
Returns the Moebius transformation with parameters a,b,c and d:

$$f(z) = (a \cdot z + b) \cdot (c \cdot z + d)^{-1}.$$

If $c \cdot z + d = 0$ returns the string `'Infinity'`.