Mathematics with Python and Ruby/Quaternions in Ruby

As was seen in the preceding chapter, a complex number is an object comprising 2 real numbers (called *real* and *imag* by *Ruby*). This is the Cayley-Dickson construction of the complex numbers. In a very similar manner, a quaternion can be considered as made of 2 complex numbers.

In all the following, *cmath* will be used as it handles fractions automatically. This chapter is in some way different from the preceding ones, as it shows how to create brand new objects in *Ruby*, and not how to use already available objects.

1 Definition and display

1.1 Definition

The definition of a quaternion finds its shelter in a class which is called *Quaternion*:

```
class Quaternion end
```

The first method of a quaternion will be its instantiation:

1.1.1 Instantiation

```
def initialize(a,b) @a,@b = a,b end
```

From now on, *a* and *b* (which will be complex numbers) will be the 2 quaternion's attributes

1.1.2 Attributes a and b

As the two numbers which define a quaternion are complex, it is not appropriate to call them the *real* and *imaginary* parts. Besides, an other stage will be necessary with the octonions later on. So the shortest names have been chosen, and they will be called the *a* of the quaternion, and its *b*.

```
def a @a end def b @b end
```

From now on it is possible to access to the *a* and *b* part of a quaternion *q* with *q.a* and *q.b*.

1.2 Display

In order that it be easy to display a quaternion *q* with *puts(q)* it is necessary to redefine a method *to_s* for it (a case of polymorphism). There are several choices but this one works OK:

```
def to_s '('+a.real.to_s+')+('+a.imag.to_s+')i+('+b.real.to_s+')j+('+b.imag.t
end
```

To read it loud it is better to read from right to left. For example, *a.real* denotes *the real part of a* and *q.a.real* denotes *the real part of the a part of q*.

2 Functions

2.1 Modulus

The absolute value of a quaternion is a (positive) real number.

```
def abs Math.hypot(@a.abs,@b.abs) end
```

2.2 Conjugate

The conjugate of a quaternion is another quaternion, having the same modulus.

```
def conj Quaternion.new(@a.conj,-@b) end
```

3 Operations

3.1 Addition

To add two quaternions, just add their *a*s together, and their *b*s together:

```
def +(q) Quaternion.new(@a+q.a,@b+q.b) end
```

3.2 Subtraction

The use of the - symbol is an other case of polymorphism, which allows to write rather simply the subtraction.

def -(q) Quaternion.new(@a-q.a,@b-q.b) end

3.3 Multiplication

Multiplication of the quaternions is more complex (!):

def *(q) Quaternion.new(@a*q.a-@b*q.b.conj,@a*q.b+@b*q.a.conj) end

This multiplication is not commutative, as can be checked by the following examples:

p=Quaternion.new(Complex(2,1),Complex(3,4)) q=Quaternion.new(Complex(2,5),Complex(-3,-5)) puts(p*q) puts(q*p)

3.4 Division

The division can be defined as this:

def /(q) d=q.abs**2 Quaternion.new((@a*q.a.conj+@b*q.b.conj)/d,(-@a*q.b+@b*q.a)/d) end

As they have the same modulus, the quotient of a quaternion by its conjugate has modulus one:

p=Quaternion.new(Complex(2,1),Complex(3,4)) puts((p/p.conj).abs)

This last example digs that $\left(-\frac{22}{30}\right)^2+\left(\frac{4}{30}\right)^2+\left(\frac{12}{30}\right)^2+\left(\frac{16}{30}\right)^2=1$, or $22^2+4^2+12^2+16^2=484+16+144+256=900=30^2$, which is a decomposition of 30^2 as a sum of 4 squares.

4 Quaternion class in Ruby

The complete class is here:

require 'cmath' class Quaternion def initialize(a,b) @a,@b = a,b end def a @a end def b @b end def to_s '('+a.real.to_s+')+('+a.imag.to_s+')i+('+b.real.to_s+')j+('+b.imag.to_s+')k' end def +(q) Quaternion.new(@a+q.a,@b+q.b) end def -(q) Quaternion.new(@a-q.a,@b-q.b) end def *(q) Quaternion.new(@a*q.a-@b*q.b.conj,@a*q.b+@b*q.a.conj) end def abs Math.hypot(@a.abs,@b.abs) end def conj Quaternion.new(@a.conj,-@b) end def /(q) d=q.abs**2 Quaternion.new((@a*q.a.conj+@b*q.b.conj)/d,(-@a*q.b+@b*q.a.conj)/d) end end

If this content is saved in a text file called *quaternion.rb*, after *require 'quaternion'* one can make computations on quaternions.

One interesting fact about the Cayley-Dickson which has been used for the quaternions above, is that it can be generalized, for example for the octonions.

5 Definition and display

5.1 Definition

All the following methods will be enclosed in a class called *Octonion*:

class Octonion def initialize(a,b) @a,@b = a,b end def a @a end def b @b end

At this point, there is not much difference from the quaternion object. Only, for an octonion, *a* and *b* will be quaternions, not complex numbers. *Ruby* will know it when *a* and *b* will be instantiated.

5.2 Display

The *to_s* method of an octonion (converting it to a string object so that it can be displayed) is very similar to the quaternion equivalent, only there are 8 real numbers to display now:

def to_s '('+a.a.real.to_s+')+('+a.a.imag.to_s+')i+('+a.b.real.to_s+')j+('+a.b end

The first of these numbers is the real part of the *a* part of the first quaternion, which is the octonions's *a*! Accessing to this *real part of the a part of the octonion's a part*, requires to go through a binary tree which depth is 3.

6 Functions

Thanks to Cayley and Dickson, the methods needed for octonions computing are similar to the quaternion's.

6.1 Modulus

Same than for the quaternions:

def abs Math.hypot(@a.abs,@b.abs) end

6.2 Conjugate

def conj Octonion.new(@a.conj,Quaternion.new(0,0)-@b) end

7 Operations

7.1 Addition

Like for the quaternions, one has just to add the *a*s and the *b*s separately (only now the *a* and *b* part are quaternions):

```
def +(o) Octonion.new(@a+o.a,@b+o.b) end
```

7.2 Subtraction

```
def -(o) Octonion.new(@a-o.a,@b-o.b) end
```

7.3 Multiplication

```
def *(o) Octonion.new(@a*o.a-o.b*@b.conj,@a.conj*o.b+o.a*@b) end
```

This multiplication is still not commutative, but it is even not associative either!

```
m=Octonion.new(p,q) n=Octonion.new(q,p) o=Octonion.new(p,p) puts((m*n)*o) puts(m*(n*o))
```

7.4 Division

```
def /(o) d=1/o.abs**2 Octonion.new((@a*o.a.conj+o.b*@b.conj)*Quaternion.new(d,0),(Quaternion.new(0,0)-@a.conj*o.b+o.a.conj*@b)*Quaternion.new(d,0)) end
```

Here again, the division of an octonion by its conjugate has modulus 1:

```
puts(m/m.conj) puts((m/m.conj).abs)
```

8 The octonion class in Ruby

The file is not much heavier than the quaternion's one:

```
class Octonion def initialize(a,b) @a,@b = a,b end def a @a end def b @b end def to_s '('+a.a.real.to_s+')+('+a.a.imag.to_s+')i+('+a.b.real.to_s+')j+('+a.b.imag.to_s+')k+('+b.a.real.to_s+')l+('+b.a.imag.to_s+')li+('+b.b.rea
end def +(o) Octonion.new(@a+o.a,@b+o.b) end def -(o) Octonion.new(@a-o.a,@b-o.b) end def *(o) Octonion.new(@a*o.a-o.b*@b.conj,@a.conj*o.b+o.a*@b) end def abs Math.hypot(@a.abs,@b.abs) end def conj Octonion.new(@a.conj,Quaternion.new(0,0)-@b) end def /(o) d=1/o.abs**2 Octonion.new((@a*o.a.conj+o.b*@b.conj)*Quaternion.new(d,0),(Quatern
@a.conj*o.b+o.a.conj*@b)*Quaternion.new(d,0)) end end
```

Saving it as *octonions.rb*, any script beginning by

```
require 'octonions'
```

allows computing on octonions.

- Actually, there is already a quaternion support for Ruby, but it is not (yet) native: ; on the same site, there is also a file for the octonions, which is interesting to compare to the above one.
- A "best-downloader" book on octonions is John Baez's one:

9 Text and image sources, contributors, and licenses

9.1 Text

- **Mathematics with Python and Ruby/Quaternions in Ruby** *Source:* https://en.wikibooks.org/wiki/Mathematics_with_Python_and_Ruby/Quaternions_in_Ruby?oldid=2255489 *Contributors:* QuiteUnusual and Alain Busser

9.2 Images

9.3 Content license