Lorentz_boosts_w_just_h

July 22, 2019

1 Rotations and Boosts with Quaternions

1.1 A Brief History of Quaternions and Lorentz Transformations

Quaternions ($C\ell_{0,2}(\mathcal{R})$) were known to have four degrees of freedom and do rotations in three spatial dimensions in the eighteen forties. Special relativity was created by Einstein in 1905. His math professor Minkowski recognized the Lorentz transformation as a rotation in the four dimensional vector space in 1908. It would appear that quaternions were tailor-made to do the work of Lorentz transformations. Early on, two people did give it a try: Conway (1911) and Silberstein (1912). What they both realized was that the triple product used for rotations could only work for complex-valued quaternions $C\ell_{1,2}(\mathcal{R})$:

$$B \rightarrow R' = uRu^*, \quad |u| = 1$$

This same technique is featured in "Gravitation" by Misner, Thorne, and Wheeler, Chapter 41 "Spinors". It is important to note that complex-valued quaternions are not a division algebra.

I suspect many physists do not care about the details of the math being used - that is for mathematicians to understand. That flies in the face of how central math is to physics. To my eye, when physics is done right, there is no difference between math and physics. This is why for me deciding what math to use is of ultimate importance. Physical laws have to be reversable. For that reason, the most basic math operations of all - addition and multiplication - must be reversable. If that is not the case, then there will be math operations that are not consistent with the reversability seen in physics. This is not a guiding principle in mainstream physics, but it is for me.

The fans of quantum mechanics will point out that quaternions in quantum mechanics have been a failure. For a reacent discussion on this subject by professionals, see the blog by Scott Aaronson, Why are amplitudes complex? where he wrote that quaternion quantum mechanics was a **flaming garbage fire**. If you want to know why his critique fails, just notice the trace of the matrices U and V used to claim quaternion quantum mechanics allows for superluminal signaling are *complex-valued*, so they are not observerable operators as presumed in the text. Since the operators are not observable, the analysis is not relevant to people making measurements.

I agree that the way Stephen Adler does quaternion quantum mechanics is technically flawed. I have made progress with something different, quaternion series quantum mechanics, where a quaternion series is a semi-group (has more than one idenity) with inverses. It remains an open question if I can fill in each essential aspect of quantum mechanics using quaternion series.

1.2 A Real-Valued Quaternion Lorentz Boost

It may have been some time in 2010 that I came up with a way to do only Lorentz boosts using a function that was a triple triple quaternions like so:

$$B \to B' = hBh^* + \frac{1}{2}((hhB)^* - (h^*h^*B)^*)$$

The triple triple quaternion function was found through a simple trial and error process to generate the expected boost by hyperbolic sines and cosines ($h = (\cosh(\alpha), I\sinh(\alpha))$. This worked to generate both pure boosts and boosts with rotations. I recently was informed that Dr. M. Kharinov has independently found this very function for doing boosts which he presented at the Polynomial Computer Algebra 2019, St. Petersburg, Russia.

It was another three years before I noticed that there was another kind of quaternion parameter h that could be used, namely those where the first term was not $\cosh(\alpha) \geq 1$, but was equal to zero. For these quaternions, the second and third terms of the triple triple cancel leaving only this expression:

$$B \to B' = hBh^*$$

This is *precisely* the same form Rodrigues found for doing 3D spatial rotations so many years ago. It is not however the same domain: the first term can be non-zero so long as the norm is equal to one. I did not feel a need to do a detailed analysis of this case since an infinite number of values could be used for $h = (0, a, b, c)$, so my gut feeling was all possible rotations had to be possible.

If one can do all possible boosts and rotations using the triple triple quaternion function, then it would appear that all Lorentz transformations can be acheived with this function.

1.3 The Problem with Rotations and Boosts

Two people with Ph.D.'s in physics (Dr. M. Kharinov and Purple Penguin, a commentor on YouTube) have looked at this function and strongly disagree with my assessment. Their logic is both simple and pursuasive. There are three degrees of freedom for rotations and three degrees of freedom for boosts. Since rotations are not boosts, that means one must have at a minimum parameter with six degrees of freedom. Since the triple triple quaternion function only has one parameter, at most there are four degrees of freedom. That is not enough for the task at hand. Nothing more needs to be said.

Purple Penguin cited a transformation that cannot be acheived using a rotation around the z axis using the triple triple quaternion function

$$B = (t, x, y, z) \to B' = (t, -y, x, z)$$

He was even able to write out a proof in the comments section of YouTube. That proof is not reproduced here, but it is solid: there is no value of the quaternion parameter h that will do a rotation around z axis like that shown above.

1.4 The 1-Way Wager

I have worked with quaternions as a number since 1997. Numbers in and of themselves are not viewed as a form of higher math. Numbers are grains of sand with little structure while the deep actions happen in the ocean of abstract mathematics. Little is not however none. Each time I have tried to apply quaternions to a problem in physics, eventually a solution was found. With the entire number system open, I was confident that a transformation to $B' = (t, -y, x, z)$ was necessarily possible even if I didn't know any details at the time. I therefore made a 1-way wager, that I could find a parameter h for the triple triple quaternion function that could do that specific transformation. If I failed at such a task, I would donate $100 to a charity of Purple Penguin's choice (I suggested the American Diabetes Association which he agreed to). Money can be a motivator, and I would have to create this notebook and video any answer to address the wager.

1.5 Quickly Try and Fail in an Interesting Way

As soon as I wrote down 1-way wager, I knew what I would try first: every possible rotation around the z axis. There are only 4 of these, $h = (0, \pm\frac{1}{\sqrt{2}}, \pm\frac{1}{\sqrt{2}}, 0)$. My practice is to do a simple calculation first, see what one learns later.

Technical sidebar: I should point out I am using a variety of conjugate functions: the conjugate does as expected, but the first conjugate *1 will flip the signs of the all but the first spatial term. The second conjugate does a similar thing, flipping all signs but the second spatial term. flip_signs() is no different from multiplying by negative one. Gauss was the first to use the first and second conjugate in an unpublished notebook although he did not refence them in this way.

Load the needed libraries.

```
In [2]: %%capture
        %matplotlib inline
        import numpy as np
        import sympy as sp
        import matplotlib.pyplot as plt
        import math
        import unittest

        # To get equations the look like, well, equations, use the following.
        from sympy.interactive import printing
        printing.init_printing(use_latex=True)
        from IPython.display import display

        # Tools for manipulating quaternions written by D. Sweetser.
        from QH import QH;

        from IPython.core.display import display, HTML, Math, Latex
        display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
In [3]: hpp = QH([0, 1/sp.sqrt(2), 1/sp.sqrt(2), 0])
        hnn = hpp.flip_signs()
        hpn = hpp.conj(1)
        hnp = hpp.conj(2)

        hpp.print_state("hpp")
        hnn.print_state("hnn")
        hpn.print_state("hpn")
        hnp.print_state("hnp")
```

```
hpp
(0, sqrt(2)/2, sqrt(2)/2, 0)
hnn
(0, -sqrt(2)/2, -sqrt(2)/2, 0)
hpn
(0, sqrt(2)/2, -sqrt(2)/2, 0)
hnp
(0, -sqrt(2)/2, sqrt(2)/2, 0)
```

```
In [4]: t, x, y, z = sp.symbols("t x y z")
        txyz = QH([t, x, y, z])
        txyz_hpp = txyz.boost(hpp).simple_q()
        txyz_hnn = txyz.boost(hnn).simple_q()
        txyz_hpn = txyz.boost(hpn).simple_q()
        txyz_hnp = txyz.boost(hnp).simple_q()

        txyz_hpp.print_state("txyz_hpp")
        txyz_hnn.print_state("txyz_hnn")
        txyz_hpn.print_state("txyz_hpn")
        txyz_hnp.print_state("txyz_hnp")
```

```
txyz_hpp
(t, y, x, -z)
txyz_hnn
(t, y, x, -z)
txyz_hpn
(t, -y, -x, -z)
txyz_hnp
(t, -y, -x, -z)
```

Even though 4 parameter h were used, the conjugates return identical results. Neither of these two are the rotation desired.

Generalize the problem by looking at all 8 posssible signs for y, x, and z:

#	pos. 0	pos. 1	pos. 2	pos. 3
1	t	y	x	z
2	t	y	x	-z
3	t	-y	x	z
4	t	-y	x	-z
5	t	y	-x	z
6	t	y	-x	-z
7	t	-y	-x	z
8	t	-y	-x	-z

The goal is to generate #3, but there is nothing particularly special about that particular one. A complete solution can generate all 8. The initial guess generated #2 and #8.

What sort of different function would be able to generate all 8 cases? I wrote a function conj_q() that can apply all three conjugates and multiply by minus one. The ability to use involutive automorphisms (a fancy name for conjugates) and multiplying by a minus sign means all 8 cases can be generated by also appling conj_q() in the right way for a boost.

```
In [5]: txyz_hpp.conj_q(QH([True, True, True, False])).print_state("#1 hpp * *1 *2")
        txyz_hpp.conj_q(QH([False, False, False, False])).print_state("#2 hpp")
        txyz_hpp.conj_q(QH([True, False, True, True])).print_state("#3 hpp * *2 *-1")
        txyz_hpp.conj_q(QH([False, True, False, True])).print_state("#4 hpp *1 *-1")
        txyz_hpp.conj_q(QH([True, True, False, True])).print_state("#5 hpp * *1 *-1")
        txyz_hpp.conj_q(QH([False, False, True, True])).print_state("#6 hpp *2 *-1")
        txyz_hpp.conj_q(QH([True, False, False, False])).print_state("#7 hpp *")
        txyz_hpp.conj_q(QH([False, True, True, False])).print_state("#8 hpp *1 *2")
```

```
#1 hpp * *1 *2
(t, y, x, z)
#2 hpp
(t, y, x, -z)
#3 hpp * *2 *-1
(t, -y, x, z)
#4 hpp *1 *-1
(t, -y, x, -z)
#5 hpp * *1 *-1
(t, y, -x, z)
#6 hpp *2 *-1
(t, y, -x, -z)
#7 hpp *
(t, -y, -x, z)
#8 hpp *1 *2
(t, -y, -x, -z)
```

This is **not an answer** to the 1-way wager since it changes the function used (the triple triple triple conjugate quaternion function perhaps?). Notice that only one of the 8 possible cases has no conjugates at all. All the others are variations on the simple case.

There are another 8 cases for improper rotations, ones were time flip signs. These can be found simply enough: just multiply each of these 8 cases by minus one.

```
In [6]: txyz_hpp.conj_q(QH([True, True, True, True])).print_state("improper #8 hpp * *1 *2 *-1
        print("...and the others")
```

```
improper #8 hpp * *1 *2 *-1
(-t, -y, -x, -z)
...and the others
```

This may appear to some as just a mathematical game. The conjugate operator does play a leading role in quantum mechanics, albeit in a different context (quaternion series not quaternions as a number). One use of this section is there are now 8 tests to pass for any solution to the 1-way wager.