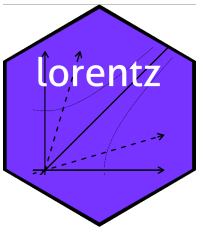
Special relativity in R: introducing the lorentz package

Robin K. S. Hankin

Abstract

Here I present the **lorentz** package for working with relativistic physics. The package includes functionality for Lorentz transformations and Einsteinian three-velocity addition, which is noncommutative and nonassociative. It is available on CRAN at `https://CRAN.R-project.org/package=lorentz`. To cite the **lorentz** package, use Hankin (2025).

Keywords: Lorentz transformation, Lorentz group, Lorentz law, Lorentz velocity addition, special relativity, relativistic physics, Einstein velocity addition, Wigner rotation, gyrogroup, gyromorphism, gyrocommutative, gyroassociative, four velocity, three-velocity, nonassociative, noncommutative.



1. Introduction

In special relativity, the Lorentz transformations supersede their classical equivalent, the Galilean transformations (Goldstein 1980). Lorentz transformations operate on four-vectors such as the four-velocity or four-potential and are usually operationalised as multiplication by a 4×4 matrix. A Lorentz transformation takes the components of an arbitrary four-vector as observed in one coordinate system and returns the components observed in another system which is moving at constant velocity with respect to the first.

There are a few existing software tools for working with Lorentz transformations, mostly developed in an educational context. Early work would include that of Horwitz, Taylor, and Barowy (1994) who describe **relLab**, a system for building a range of *gendanken* experiments in an interactive graphical environment. The author asserts that it runs on "any Macintosh computer with one megabyte of RAM or more" but it is not clear whether the software is still available. More modern contributions would include the OpenRelativity toolkit (Sherin, Cheu, Tan, and Kortemeyer 2016) which simulates the effects of special relativity in the Unity game engine.

The **lorentz** package provides R-centric functionality for Lorentz transformations. It deals with formal Lorentz boosts, converts between three-velocities and four-velocities, and provides

computational support for the gyrogroup structure of relativistic three-velocity addition.

2. Lorentz transformations: active and passive

Passive transformations are the usual type of transformations taught and used in relativity. However, sometimes active transformations are needed and it is easy to confuse the two. Here I will discuss passive and then active transformations, and illustrate both in a computational context.

Passive transformations

Consider the following canonical Lorentz transformation in which we have motion in the x-direction at speed $v > 0$; the motion is from left to right. We consider only the first two components of four-vectors, the y- and z- components being trivial. A typical physical interpretation is that I am at rest, and my friend is in his spaceship moving at speed v past me; and we are wondering what vectors which I measure in my own rest frame look like to him. The (passive) Lorentz transformation is:

$$\begin{pmatrix} \gamma & -\gamma v \\ -\gamma v & \gamma \end{pmatrix}$$

And the canonical example of that would be:

$$\begin{pmatrix} \gamma & -\gamma v \\ -\gamma v & \gamma \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \gamma \\ -\gamma v \end{pmatrix}$$

where the vectors are four velocities (recall that $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is the four-velocity of an object at rest). Operationally, I measure the four-velocity of an object to be $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$, and he measures the same object as having a four-velocity of $\begin{pmatrix} \gamma \\ -\gamma v \end{pmatrix}$. So I see the object at rest, and he sees it as moving at speed $-v$; that is, he sees it moving to the left (it moves to the left because he is moving to the right relative to me). The package makes computations easy. Suppose $v = 0.6c$ in the x-direction.

```
>                            # NB: speed of light = 1 by default
> u <- as.3vel(c(0.6,0,0))  # coerce to a three-velocity
> u

A vector of three-velocities (speed of light = 1)
       x y z
[1,] 0.6 0 0

> as.4vel(u)                 # four-velocity is better for calculations
```

```
A vector of four-velocities (speed of light = 1)
        t    x y z
[1,] 1.25 0.75 0 0
```

```
> (B <- boost(u))           # transformation matrix
```

```
      t     x y z
t  1.25 -0.75 0 0
x -0.75  1.25 0 0
y  0.00  0.00 1 0
z  0.00  0.00 0 1
```

(note that element $[1,2]$ of the boost matrix B is negative as we have a passive transformation). Then a four-velocity of $(1,0,0,0)^T$ would appear in the moving frame as

```
> B %*% c(1,0,0,0)
```

```
   [,1]
t  1.25
x -0.75
y  0.00
z  0.00
```

This corresponds to a speed of $-0.75/1.25 = -0.6$. Observe that it is possible to transform an arbitrary four-vector:

```
> B %*% c(4,6,-8,9)
```

```
  [,1]
t  0.5
x  4.5
y -8.0
z  9.0
```

Null vectors: light

Let's try it with light (see section 9 for more details on photons). Recall that we describe a photon in terms of its four momentum, not four-velocity, which is undefined for a photon. Specifically, we *define* the four-momentum of a photon to be

$$\begin{pmatrix} E/c \\ Ev_x/c^2 \\ Ev_y/c^2 \\ Ev_z/c^2 \end{pmatrix}$$

So if we consider unit energy and keep $c = 1$ we get $p = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ in our one-dimensional world (for a rightward-moving photon) and the Lorentz transformation is then

$$\begin{pmatrix} \gamma & -\gamma v \\ -\gamma v & \gamma \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \gamma - \gamma v \\ \gamma - \gamma v \end{pmatrix}$$

So, in the language used above, I see a photon with unit energy, and my friend sees the photon with energy $\gamma(1-v) = \sqrt{\frac{1-v}{1+v}} < 1$, provided that $v > 0$: the photon has less energy in his frame than mine because of Doppler redshifting. It's worth doing the same analysis with a leftward-moving photon:

$$\begin{pmatrix} \gamma & -\gamma v \\ -\gamma v & \gamma \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \gamma(1+v) \\ -\gamma(1+v) \end{pmatrix}$$

Here the photon has more energy for him than me because of blue shifting: he is moving to the right and encounters a photon moving to the left. The R idiom would be

```
> B %*% c(1,1,0,0)

  [,1]
t  0.5
x  0.5
y  0.0
z  0.0

> B %*% c(1,-1,0,0)

  [,1]
t    2
x   -2
y    0
z    0
```

for the left- and right- moving photons respectively.

The above analysis uses *passive* transformations: there is a single physical reality, and we describe that one physical reality using two different coordinate systems. One of the coordinate systems uses a set of axes that are *boosted* relative to the axes of the other.

This is why it makes sense to use prime notation as in $x \longrightarrow x'$ and $t \longrightarrow t'$ for a passive Lorentz transformation: the prime denotes measurements made using coordinates that are defined with respect to the boosted system, and we see notation like

$$\begin{pmatrix} t' \\ x' \end{pmatrix} = \begin{pmatrix} \gamma & -\gamma v \\ -\gamma v & \gamma \end{pmatrix} \begin{pmatrix} t \\ x \end{pmatrix}$$

These are the first two elements of a displacement four-vector. It is the same four-vector but viewed in two different reference frames.

Active transformations

In the passive view, there is a single physical reality, and we are just describing that one physical reality using two different coordinate systems. Now we will consider active transformations: there are two physical realities, but one is boosted with respect to another.

Suppose me and my friend have zero relative velocity, but my friend is in a spaceship and I am outside it, in free space, with both of us at rest (with respect to the distant stars, say). He constructs a four-vector in his spaceship; for example, he could fire bullets out of a gun which is fixed in the spaceship, and then calculate their four-velocity as it appears to him in his spaceship-centric coordinate system. We both agree on this four-velocity as our reference frames are identical: we have no relative velocity.

Now his spaceship acquires a constant velocity, leaving me stationary. My friend continues to fire bullets out of his gun and sees that their four-velocity, as viewed in his spaceship coordinates, is the same as when we were together.

Now he wonders what the four-velocity of the bullets is in my reference frame. This is an *active* transformation: we have two distinct physical realities, one in the spaceship when it was at rest with respect to me, and one in the spaceship when moving. And both these realities, by construction, look the same to my friend in the spaceship.

Suppose, for example, he sees the bullets at rest in his spaceship; they have a four-velocity of $\begin{pmatrix}1\\0\end{pmatrix}$, and my friend says to himself: "I see bullets with a four velocity of $\begin{pmatrix}1\\0\end{pmatrix}$, and I know what that means. The bullets are at rest. What are the bullets' four velocities in Robin's reference frame?". This is an *active* transformation:

$$\begin{pmatrix}\gamma & \gamma v\\ \gamma v & \gamma\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}=\begin{pmatrix}\gamma\\ \gamma v\end{pmatrix}$$

(we again suppose that the spaceship moves at speed $v>0$ from left to right). So he sees a four velocity of $\begin{pmatrix}1\\0\end{pmatrix}$ and I see $\begin{pmatrix}\gamma\\ \gamma v\end{pmatrix}$, that is, with a positive speed: the bullets move from left to right (with the spaceship). The R idiom would be:

```
> (B <- boost(as.3vel(c(0.8,0,0))))  # 0.8c left to right

          t         x y z
t  1.666667 -1.333333 0 0
x -1.333333  1.666667 0 0
y  0.000000  0.000000 1 0
z  0.000000  0.000000 0 1

> solve(B) %*% c(1,0,0,0)            # active transformation

      [,1]
t 1.666667
x 1.333333
y 0.000000
z 0.000000
```

3. Successive Lorentz transformation

Coordinate transformation is effected by standard matrix multiplication; thus composition of two Lorentz transformations is also ordinary matrix multiplication:

```
> u <- as.3vel(c(0.3,-0.4,+0.8))
> v <- as.3vel(c(0.4,+0.2,-0.1))
> L <- boost(u) %*% boost(v)
> L
```

```
          t          x          y          z
t  3.256577 -2.2327055  0.5419596 -2.0800479
x -1.437147  1.6996791 -0.0237489  0.4194255
y  1.091131 -0.7581795  1.1190282 -0.6029155
z -2.519789  1.5878378 -0.2023170  2.1879612
```

But observe that the resulting transformation is not a pure boost, as the spatial components are not symmetrical. We may decompose the matrix product L into a pure translation composed with an orthogonal matrix, which represents a coordinate rotation. The R idiom is `pureboost()` for the pure boost component, and `orthog()` for the rotation:

```
> (P <- pureboost(L))  # pure boost
```

```
           t          x          y          z
t  3.2565770 -2.2327055  0.5419596 -2.0800479
x -2.2327055  2.1711227 -0.2842745  1.0910491
y  0.5419596 -0.2842745  1.0690039 -0.2648377
z -2.0800479  1.0910491 -0.2648377  2.0164504
```

```
> P - t(P)   # check for symmetry
```

```
  t             x            y             z
t 0  0.000000e+00 0.000000e+00  0.000000e+00
x 0  0.000000e+00 5.551115e-17 -2.220446e-16
y 0 -5.551115e-17 0.000000e+00  0.000000e+00
z 0  2.220446e-16 0.000000e+00  0.000000e+00
```

Now we compute the rotation:

```
> (U <- orthog(L))                  # rotation matrix
```

```
              t             x             y             z
t  1.000000e+00  1.018630e-14 -2.609908e-15  1.010454e-14
x  4.628599e-15  9.458514e-01  1.592328e-01 -2.828604e-01
y -4.401797e-15 -1.858476e-01  9.801022e-01 -6.971587e-02
z  9.079414e-15  2.661311e-01  1.185098e-01  9.566241e-01
```

```
> U[2:4,2:4]                        # inspect the spatial components

           x         y           z
x  0.9458514 0.1592328 -0.28286043
y -0.1858476 0.9801022 -0.06971587
z  0.2661311 0.1185098  0.95662410

> round(crossprod(U) - diag(4),10)  # check for orthogonality

  t x y z
t 0 0 0 0
x 0 0 0 0
y 0 0 0 0
z 0 0 0 0

> ## zero to within numerical uncertainty
```

4. Units in which $\mathbf{c} \neq 1$

The preceding material used units in which $c = 1$. Here I show how the package deals with units such as SI in which $c = 299792458 \neq 1$. For obvious reasons we cannot have a function called `c()` so the package gets and sets the speed of light with function `sol()`:

```
> sol(299792458)

[1] 299792458

> sol()

[1] 299792458
```

The speed of light is now 299792458 until re-set by `sol()` (an empty argument queries the speed of light). We now consider speeds which are fast by terrestrial standards but involve only a small relativistic correction to the Galilean result:

```
> u <- as.3vel(c(100,200,300))
> as.4vel(u)

A vector of four-velocities (speed of light = 299792458)
     t   x   y   z
[1,] 1 100 200 300
```

The gamma correction term γ is only very slightly larger than 1 and indeed R's default print method suppresses the difference:

```
> gam(u)

[1] 1
```

However, we can display more significant figures by subtracting one:

```
> gam(u)-1

[1] 7.789325e-13
```

or alternatively we can use the `gamm1()` function which calculates $\gamma - 1$ more accurately for speeds $\ll c$:

```
> gamm1(u)

[1] 7.78855e-13
```

The Lorentz boost is again calculated by the `boost()` function:

```
> boost(u)

     t             x            y             z
t    1 -1.112650e-15 -2.22530e-15 -3.337950e-15
x -100  1.000000e+00  1.11265e-13  1.668975e-13
y -200  1.112650e-13  1.00000e+00  3.337950e-13
z -300  1.668975e-13  3.33795e-13  1.000000e+00
```

The boost matrix is not symmetrical, even though it is a pure boost, because $c \neq 1$.
Note how the transformation is essentially the Galilean result, which is discussed below.

4.1. Changing units

Often we have a four-vector in SI units and wish to express this in natural units.

```
> sol(299792458)

[1] 299792458

> disp  <- c(1,1,0,0)
```

If we interpret `disp` as a four-displacement, it corresponds to moving 1 metre along the x-axis and waiting for one second. To convert this to natural units we multiply by the passive transformation matrix given by `ptm()`:

```
> ptm(to_natural=TRUE) %*% disp
```

```
       [,1]
t 299792458
x         1
y         0
z         0
```

In the above, see how the same vector is expressed in natural units in which the speed of light is equal to 1: the unit of time is about 3×10^{-9} seconds and the unit of distance remains the metre. Alternatively, we might decide to keep the unit of time equal to one second, and use a unit of distance equal to 299792458 metres which again ensures that $c = 1$:

```
> ptm(to_natural=TRUE,change_time=FALSE) %*% disp
```

```
          [,1]
t 1.000000e+00
x 3.335641e-09
y 0.000000e+00
z 0.000000e+00
```

As a further check, we can take two boost matrices corresponding to the same coordinate transformation but expressed using different units of length and verify that their orthogonal component agrees:

```
> sol(1)
```

```
[1] 1
```

```
> B1 <- boost((2:4)/10) %*% boost(c(-5,1,3)/10)
> orthog(B1)[2:4,2:4]
```

```
           x           y          z
x  0.9832336  0.09752166 0.15408208
y -0.1020390  0.99454439 0.02166761
z -0.1511284 -0.03702671 0.98782044
```

Now we create `B2` which is the same physical object but using a length scale of one-tenth of `B2` (which requires that we multiply the speed of light by a factor of 10):

```
> sol(10)
```

```
[1] 10
```

```
> B2 <- boost(2:4) %*% boost(c(-5,1,3))  # exactly the same as B1 above
> orthog(B2)[2:4,2:4]
```

```
           x           y          z
x  0.9832336  0.09752166 0.15408208
y -0.1020390  0.99454439 0.02166761
z -0.1511284 -0.03702671 0.98782044
```

so the two matrices agree, as expected.

5. Infinite speed of light

In the previous section considered speeds that were small compared with the speed of light and here we will consider the classical limit of infinite c:

```
> sol(Inf)
```

```
[1] Inf
```

Then the familiar parallelogram law operates:

```
> u <- as.3vel(1:3)
> v <- as.3vel(c(-6,8,3))
> u+v
```

```
A vector of three-velocities (speed of light = Inf)
      x  y z
[1,] -5 10 6
```

```
> v+u
```

```
A vector of three-velocities (speed of light = Inf)
      x  y z
[1,] -5 10 6
```

Above we see that composition of velocities is commutative, unlike the relativistic case. The boost matrix is instructive:

```
> boost(u)
```

```
   t x y z
t  1 0 0 0
x -1 1 0 0
y -2 0 1 0
z -3 0 0 1
```

```
> boost(u+v)
```

```
    t x y z
t   1 0 0 0
x   5 1 0 0
y -10 0 1 0
z  -6 0 0 1
```

```
> boost(u) %*% boost(v)
```

```
    t x y z
t   1 0 0 0
x   5 1 0 0
y -10 0 1 0
z  -6 0 0 1
```

Above, see how the boost matrix for the composed velocity of $u+v$ does not have any rotational component, unlike the relativistic case [recall that `boost()` gives a *passive* transformation, which is why the sign of the numbers in the first column is changed]. With an infinite speed of light, even "large" speeds have zero relativistic correction:

```
> gamm1(1e100)
```

```
[1] 0
```

Function `rboost()` returns a random Lorentz transformation matrix, which is in general a combination of a pure Lorentz boost and an orthogonal rotation. With an infinite speed of light, it requires a speed:

```
> set.seed(0)
> options(digits=3)
> (B <- rboost(1))  # random boost, speed 1
```

```
          t      x       y      z
[1,]  1.000  0.000  0.0000  0.000
[2,] -0.411  0.213 -0.9402  0.266
[3,] -0.279 -0.917 -0.0989  0.385
[4,]  0.868 -0.336 -0.3260 -0.884
```

We can decompose `B` into a pure boost and an orthogonal transformation:

```
> orthog(B)
```

```
     [,1]   [,2]    [,3]   [,4]
[1,]    1  0.000  0.0000  0.000
[2,]    0  0.213 -0.9402  0.266
[3,]    0 -0.917 -0.0989  0.385
[4,]    0 -0.336 -0.3260 -0.884
```

```
> pureboost(B)
```

```
          t x y z
[1,]  1.000 0 0 0
[2,] -0.123 1 0 0
[3,]  0.131 0 1 0
[4,] -0.984 0 0 1
```

Boost matrices can be applied to any four-vector. Here I show how pure spatial displacements transform with an infinite light speed.

```
> (u <- as.3vel(c(10,0,0))) # velocity of 10, parallel to x axis
```

```
A vector of three-velocities (speed of light = Inf)
      x y z
[1,] 10 0 0
```

```
> (B <- boost(u))
```

```
    t x y z
t   1 0 0 0
x -10 1 0 0
y   0 0 1 0
z   0 0 0 1
```

```
> d <- c(0,1,0,0) # displacement of distance one, parallel to the x-axis
> B %*% d
```

```
  [,1]
t    0
x    1
y    0
z    0
```

Above we see that a spatial displacement is the same for both observers. We can similarly apply a boost to a temporal displacement:

```
> d <- c(1,0,0,0) # displacement of one unit of time, no spatial component
> B %*% d
```

```
  [,1]
t    1
x  -10
y    0
z    0
```

Above we see the result expected from classical mechanics.

6. Vectorization

Here I discuss vectorized operations (to avoid confusion between boost matrices and their transposes we will use $c = 10$). The issue is difficult because a Lorentz boost is conceptually a matrix product of a 4×4 matrix with vector with four elements:

```
> sol(10)

[1] 10

> u <- as.3vel(c(5,-6,4))
> (U <- as.4vel(u))

A vector of four-velocities (speed of light = 10)
        t    x     y    z
[1,] 2.09 10.4 -12.5 8.34

> B <- boost(U)
> B %*% as.vector(U)

       [,1]
t  1.00e+00
x  0.00e+00
y -1.78e-15
z  1.78e-15
```

(note that the result is the four-velocity of an object at rest, as expected, for we use passive transformations by default). However, things are different if we wish to consider many four-vectors in one R object. A vector $\mathbf{V}$ of four-velocities is a matrix: each *row* of $\mathbf{V}$ is a four-velocity. In the package we represent this with objects of class `4vel`. Because a vector is treated (almost) as a one-column matrix in R, and the four-velocities are rows, we need to take a transpose in some sense.

```
> u <- 1:7  # speed in the x-direction [c=10]
> jj <- cbind(gam(u),gam(u)*u,0,0)
> (U <- as.4vel(jj))

A vector of four-velocities (speed of light = 10)
        t    x y z
[1,] 1.01 1.01 0 0
[2,] 1.02 2.04 0 0
[3,] 1.05 3.14 0 0
[4,] 1.09 4.36 0 0
[5,] 1.15 5.77 0 0
[6,] 1.25 7.50 0 0
[7,] 1.40 9.80 0 0
```

Now a boost, also in the x-direction:

```
> (B <- boost(as.3vel(c(6,0,0))))  # 60% speed of light
```

```
      t      x y z
t  1.25 -0.075 0 0
x -7.50  1.250 0 0
y  0.00  0.000 1 0
z  0.00  0.000 0 1
```

Note the asymmetry of B, in this case reflecting the speed of light being 10 (but note that boost matrices are not always symmetrical, even if $c = 1$).

To effect a *passive* boost we need to multiply each row of U by the transpose of the boost matrix B:

```
> U %*% t(B)
```

```
        t     x y z
[1,] 1.18 -6.28 0 0
[2,] 1.12 -5.10 0 0
[3,] 1.07 -3.93 0 0
[4,] 1.04 -2.73 0 0
[5,] 1.01 -1.44 0 0
[6,] 1.00  0.00 0 0
[7,] 1.02  1.75 0 0
```

we can verify that the above is at least plausible:

```
> is.consistent.4vel(U %*% t(B))
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

the above shows that the four velocities U, as observed by an observer corresponding to boost B, satisfies $U^i U_i = -c^2$. Anyway, in this context we really ought to use `tcrossprod()`:

```
> tcrossprod(U,B)
```

```
        t     x y z
[1,] 1.18 -6.28 0 0
[2,] 1.12 -5.10 0 0
[3,] 1.07 -3.93 0 0
[4,] 1.04 -2.73 0 0
[5,] 1.01 -1.44 0 0
[6,] 1.00  0.00 0 0
[7,] 1.02  1.75 0 0
```

which would be preferable (because this idiom does not require one to take a transpose) although the speed increase is unlikely to matter much because B is only 4×4.

The above transformations were passive: we have some four-vectors measured in my rest frame, and we want to see what these are four-vectors as measured by my friend, who is moving in the positive x direction at 60% of the speed of light (remember that $c = 10$). See how the x-component of the transformed four-velocity is negative, because in my friend's rest frame, the four velocities are pointing backwards.

To effect an *active* transformation we need to take the matrix inverse of B:

```
> solve(B)
```

```
     t     x y z
t 1.25 0.075 0 0
x 7.50 1.250 0 0
y 0.00 0.000 1 0
z 0.00 0.000 0 1
```

and then

```
> tcrossprod(U,solve(B))
```

```
        t     x y z
[1,] 1.33  8.79 0 0
[2,] 1.43 10.21 0 0
[3,] 1.55 11.79 0 0
[4,] 1.69 13.64 0 0
[5,] 1.88 15.88 0 0
[6,] 2.13 18.75 0 0
[7,] 2.49 22.75 0 0
```

In the above, note how the positive x-component of the four-velocity is increased because we have actively boosted it. We had better check the result for consistency:

```
> is.consistent.4vel(tcrossprod(U,solve(B)))
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

7. Multiple boosts

If we are considering multiple boosts, it is important to put them in the correct order. First we will do some passive boosts.

```
> sol(100)
```

```
[1] 100
```

```
> B1 <- boost(r3vel(1)) %*% boost(r3vel(1))
> B2 <- boost(r3vel(1)) %*% boost(r3vel(1))
> (U <- r4vel(5))

A vector of four-velocities (speed of light = 100)
        t      x     y      z
[1,] 1.99 -162.4  46.3  34.32
[2,] 2.58  149.0 185.0  -3.07
[3,] 1.75 -110.4 -18.4 -90.19
[4,] 1.70   55.1 -88.3  90.15
[5,] 2.62 -205.6  98.2  81.53
```

Successive boosts are effected by matrix multiplication; there are at least four equivalent R constructions:

```
> U %*% t(B1) %*% t(B2)

         t    x     y    z
[1,] 11.09 -601  -844 -383
[2,]  3.00  -70  -166 -218
[3,] 13.98 -639 -1087 -594
[4,]  7.75 -239  -697 -220
[5,] 12.23 -706  -911 -394

> U %*% t(B2 %*% B1)    # note order of operations

         t    x     y    z
[1,] 11.09 -601  -844 -383
[2,]  3.00  -70  -166 -218
[3,] 13.98 -639 -1087 -594
[4,]  7.75 -239  -697 -220
[5,] 12.23 -706  -911 -394

> tcrossprod(U, B2 %*% B1)

         t    x     y    z
[1,] 11.09 -601  -844 -383
[2,]  3.00  -70  -166 -218
[3,] 13.98 -639 -1087 -594
[4,]  7.75 -239  -697 -220
[5,] 12.23 -706  -911 -394

> U %>% tcrossprod(B2 %*% B1)

         t    x     y    z
[1,] 11.09 -601  -844 -383
```

```
[2,]  3.00  -70  -166 -218
[3,] 13.98 -639 -1087 -594
[4,]  7.75 -239  -697 -220
[5,] 12.23 -706  -911 -394
```

(in the above, note that the result is the same in each case).

A warning

It is easy to misapply matrix multiplication in this context. Note carefully that the following natural idiom is **incorrect**:

```
> U %*% B  # Young Frankenstein: Do Not Use This Brain!

         t      x     y      z
[1,]  1220 -203.2  46.3  34.32
[2,] -1115  186.1 185.0  -3.07
[3,]   830 -138.1 -18.4 -90.19
[4,]  -411   68.7 -88.3  90.15
[5,]  1545 -257.1  98.2  81.53

> ## The above idiom is incorrect.  See
> ## https://www.youtube.com/watch?v=m7-bMBuVmHo&t=1s
> ## (in particular @1:08) for a technical explanation of why
> ## this is a Very Bad Idea (tm).
```

It is not clear to me that the idiom above has any meaning at all.

8. The stress-energy tensor

The stress-energy tensor (sometimes the energy-momentum tensor) is a generalization and combination of the classical concepts of density, energy flux, and the classical stress tensor (Schutz 1985). It is a contravariant tensor of rank two, usually represented as a symmetric 4×4 matrix. The **lorentz** package includes functionality for applying Lorentz transformations to the stress energy tensor.

```
> sol(1)        # revert to natural units

[1] 1

> D <- dust(1)  # Dust is the simplest nontrivial SET, with
> D             # only one nonzero component

  t x y z
t 1 0 0 0
x 0 0 0 0
y 0 0 0 0
z 0 0 0 0
```

The stress-energy tensor is usually written with two upstairs (contravariant) indices, as in $T^{\alpha\beta}$; it may be transformed using the `transform_uu()` function:

```
> B <- boost(as.3vel(c(0.0,0.8,0.0)))
> transform_uu(D,B)

      t x     y z
t  2.78 0 -2.22 0
x  0.00 0  0.00 0
y -2.22 0  1.78 0
z  0.00 0  0.00 0
```

In this reference frame, the dust is not at rest: the stress-energy tensor has components corresponding to nonzero pressure and momentum transfer, and the $[t,t]$ component is greater, at 2.78, than its rest value of 1. Note that the $[t,y]$ component is negative as we use passive transformations. If one wants to consider the stress-energy tensor with downstairs indices (here we will use a photon gas), we need to use `transform_dd()`:

```
> pg <- photongas(3)
> pg

  t x y z
t 3 0 0 0
x 0 1 0 0
y 0 0 1 0
z 0 0 0 1

> transform_uu(pg,B)

      t x     y z
t 10.11 0 -8.89 0
x  0.00 1  0.00 0
y -8.89 0  8.11 0
z  0.00 0  0.00 1
```

again we see that the $[0,0]$ component is larger than its rest value, and we see nonzero off-diagonal components which correspond to the dynamical behaviour. As a consistency check we can verify that this is the same as transforming the SET with upstairs indices, using the `lower()` and `raise()` functions:

```
> raise(transform_dd(lower(pg),lower(B)))

      t x     y z
t 10.11 0 -8.89 0
x  0.00 1  0.00 0
y -8.89 0  8.11 0
z  0.00 0  0.00 1
```

```
> raise(transform_dd(lower(pg),lower(B))) - transform_uu(pg,B) #zero to numerical precisio

  t x y z
t 0 0 0 0
x 0 0 0 0
y 0 0 0 0
z 0 0 0 0
```

One of the calls to `lower()` is redundant; for a photon gas, raising or lowering both indices does not change the components as the Minkowski metric is symmetric and orthogonal.

8.1. Successive boosts

Successive boots are represented as ordinary matrix multiplication. Again the **magrittr** package can be used for more readable idiom.

```
> B1 <- boost(as.3vel(c(0.5,-0.4,0.6)))
> B2 <- boost(as.3vel(c(0.1,-0.1,0.3)))
> pf <- perfectfluid(4,1)
> pf

  t x y z
t 4 0 0 0
x 0 1 0 0
y 0 0 1 0
z 0 0 0 1

> pf %>% transform_uu(B1) %>% transform_uu(B2)

      t      x      y     z
t  38.4 -18.17  15.24 -28.2
x -18.2   9.38  -7.03  13.0
y  15.2  -7.03   6.89 -10.9
z -28.2  12.98 -10.89  21.1

> pf %>% transform_uu(B2 %*% B1)  # should match

      t      x      y     z
t  38.4 -18.17  15.24 -28.2
x -18.2   9.38  -7.03  13.0
y  15.2  -7.03   6.89 -10.9
z -28.2  12.98 -10.89  21.1
```

Again as a consistency check, we may verify that transforming downstairs indices gives the same result:

```
> lower(pf) %>% transform_dd(lower(B1) %*% lower(B2)) %>% raise()
```

```
      t      x      y     z
t  38.4 -18.17  15.24 -28.2
x -18.2   9.38  -7.03  13.0
y  15.2  -7.03   6.89 -10.9
z -28.2  12.98 -10.89  21.1
```

(note that the matrix representation of the Lorentz transformations requires that the order of multiplication be reversed for successive covariant transformations, so `B1` and `B2` must be swapped).

8.2. Speed of light and the stress-energy tensor

Here I will perform another consistency check, this time with non-unit speed of light, for a perfect fluid:

```
> sol(10)

[1] 10

> pf_rest <- perfectfluid(1,4)
> pf_rest

     t    x    y    z
t 1.04 0.00 0.00 0.00
x 0.00 0.04 0.00 0.00
y 0.00 0.00 0.04 0.00
z 0.00 0.00 0.00 0.04
```

Thus `pf_rest` is the stress energy for a perfect fluid at rest in a particular frame F. We may now consider the same perfect fluid, but moving with a three velocity of $(3,4,5)'$: with respect to F:

```
> u <- as.3vel(3:5)
> pf_moving <- perfectfluid(1,4,u)
> pf_moving

      t     x     y    z
t  2.08  6.24  8.32 10.4
x  6.24 18.76 24.96 31.2
y  8.32 24.96 33.32 41.6
z 10.40 31.20 41.60 52.0
```

The consistency check is to verify that transforming to a frame in which the fluid is at rest will result in a stress-energy tensor that matches `pf_rest`:

```
> transform_uu(perfectfluid(1,4,u),boost(u))
```

```
          t         x         y         z
t  1.04e+00 -1.22e-16 -6.41e-16  1.64e-15
x -1.16e-15  4.00e-02 -3.88e-15 -7.50e-15
y -2.88e-15  6.28e-15  4.00e-02 -2.90e-15
z -3.91e-15  1.60e-15  7.69e-15  4.00e-02
```

thus showing agreement to within numerical precision.

9. Photons

It is possible to define the four-momentum of photons by specifying their three-velocity and energy, and using `as.photon()`:

```
> sol(1)

[1] 1

> (A <- as.photon(as.3vel(cbind(0.9,1:5/40,5:1/40))))

     E   p_x    p_y    p_z
[1,] 1 0.990 0.0275 0.1375
[2,] 1 0.992 0.0551 0.1103
[3,] 1 0.993 0.0828 0.0828
[4,] 1 0.992 0.1103 0.0551
[5,] 1 0.990 0.1375 0.0275
```

above, A is a vector of four-momentum of five photons, all of unit energy, each with a null world line. They are all moving approximately parallel to the x-axis. We can check that this is indeed a null vector:

```
> inner4(A)

[1]  1.45e-16  2.56e-17 -5.55e-17  2.56e-17  1.45e-16
```

showing that the vectors are indeed null to numerical precision. What do these photons look like in a frame moving along the x-axis at $0.7c$?

```
> tcrossprod(A,boost(as.3vel(c(0.7,0,0))))

         t     x      y      z
[1,] 0.430 0.406 0.0275 0.1375
[2,] 0.428 0.409 0.0551 0.1103
[3,] 0.427 0.410 0.0828 0.0828
[4,] 0.428 0.409 0.1103 0.0551
[5,] 0.430 0.406 0.1375 0.0275
```

Above, see how the photons have lost the majority of their energy due to redshifting. Blue shifting is easy to implement as either a passive transformation:

```
> tcrossprod(A,boost(as.3vel(c(-0.7,0,0))))
```

```
        t    x      y      z
[1,] 2.37 2.37 0.0275 0.1375
[2,] 2.37 2.37 0.0551 0.1103
[3,] 2.37 2.37 0.0828 0.0828
[4,] 2.37 2.37 0.1103 0.0551
[5,] 2.37 2.37 0.1375 0.0275
```

or an active transformation:

```
> tcrossprod(A,solve(boost(as.3vel(c(0.7,0,0)))))
```

```
        t    x      y      z
[1,] 2.37 2.37 0.0275 0.1375
[2,] 2.37 2.37 0.0551 0.1103
[3,] 2.37 2.37 0.0828 0.0828
[4,] 2.37 2.37 0.1103 0.0551
[5,] 2.37 2.37 0.1375 0.0275
```

giving the same result.

9.1. Reflection in mirrors

Gjurchinovski (2004) discusses reflection of light from a uniformly moving mirror and here I show how the **lorentz** package can illustrate some of his insights. We are going to take the five photons defined above and reflect them in an oblique mirror which is itself moving at half the speed of light along the x-axis. The first step is to define the mirror `m`, and the boost `B` corresponding to its velocity:

```
> m <- c(1,1,1)
> B <- boost(as.3vel(c(0.5,0,0)))
```

Above, the three-vector m is parallel to the normal vector of the mirror and B shows the Lorentz boost needed to bring it to rest. We are going to reflect these photons in this mirror. The R idiom for the reflection is performed using a sequence of transformations. First, transform the photons' four-momentum to a frame in which the mirror is at rest:

```
> A
```

```
     E   p_x    p_y    p_z
[1,] 1 0.990 0.0275 0.1375
[2,] 1 0.992 0.0551 0.1103
[3,] 1 0.993 0.0828 0.0828
[4,] 1 0.992 0.1103 0.0551
[5,] 1 0.990 0.1375 0.0275
```

```
> (A <- as.4mom(A %*% t(B)))
```

```
         E   p_x    p_y    p_z
[1,] 0.583 0.566 0.0275 0.1375
[2,] 0.582 0.569 0.0551 0.1103
[3,] 0.581 0.569 0.0828 0.0828
[4,] 0.582 0.569 0.1103 0.0551
[5,] 0.583 0.566 0.1375 0.0275
```

Above, see how the photons have lost energy because of a redshift (the `as.4mom()` function has no effect other than changing the column names). Next, reflect the photons in the mirror (which is at rest):

```
> (A <- reflect(A,m))
```

```
         E    p_x    p_y    p_z
[1,] 0.583 0.0786 -0.460 -0.350
[2,] 0.582 0.0793 -0.434 -0.379
[3,] 0.581 0.0795 -0.407 -0.407
[4,] 0.582 0.0793 -0.379 -0.434
[5,] 0.583 0.0786 -0.350 -0.460
```

Above, see how the reflected photons have a reduced the x-component of momentum; but have acquired a substantial y- and z- component. Finally, we transform back to the original reference frame. Observe that this requires an *active* transformation which means that we need to use the matrix inverse of B:

```
> (A <- as.4mom(A %*% solve(t(B))))
```

```
         E   p_x    p_y    p_z
[1,] 0.719 0.427 -0.460 -0.350
[2,] 0.718 0.427 -0.434 -0.379
[3,] 0.717 0.427 -0.407 -0.407
[4,] 0.718 0.427 -0.379 -0.434
[5,] 0.719 0.427 -0.350 -0.460
```

Thus in the original frame, the photons have lost about a quarter of their energy as a result of a Doppler effect: the mirror was receding from the source. The photons have imparted energy to the mirror as a result of mechanical work. It is possible to carry out the same operations in one line:

```
> A <- as.photon(as.3vel(cbind(0.9,1:5/40,5:1/40)))
> A %>% tcrossprod(B) %>% reflect(m) %>% tcrossprod(solve(B)) %>% as.4mom
```

```
         E   p_x    p_y    p_z
[1,] 0.719 0.427 -0.460 -0.350
```

```
[2,] 0.718 0.427 -0.434 -0.379
[3,] 0.717 0.427 -0.407 -0.407
[4,] 0.718 0.427 -0.379 -0.434
[5,] 0.719 0.427 -0.350 -0.460
```

9.2. Disco ball

It is easy to define a disco ball, which is a sphere covered in mirrors. For the purposes of exposition, we will use a rather shabby ball with only 7 mirrors:

```
> dfun <- function(n){matrix(rnorm(n*3),ncol=3) %>% sweep(1, sqrt(rowSums(.^2)),`/`)}
> (disco <- dfun(7))
```

```
        [,1]    [,2]    [,3]
[1,] -0.8255  0.5638 -0.0246
[2,]  0.4893 -0.0811  0.8683
[3,] -0.0654  0.4519  0.8897
[4,] -0.6690 -0.2310 -0.7065
[5,] -0.7243 -0.6781 -0.1248
[6,] -0.5078 -0.4407  0.7402
[7,]  0.5224 -0.4929  0.6958
```

Then we define a unit-energy photon moving parallel to the x-axis, and reflect it in the disco ball:

```
> p <- as.photon(c(1,0,0))
> reflect(p,disco)
```

```
  E     p_x     p_y     p_z
x 1 -0.3630  0.9309 -0.0406
x 1  0.5212  0.0794 -0.8497
x 1  0.9914  0.0591  0.1164
x 1  0.1049 -0.3091 -0.9452
x 1 -0.0491 -0.9823 -0.1807
x 1  0.4843 -0.4476  0.7518
x 1  0.4542  0.5150 -0.7270
```

(above, `p` is a photon moving along the x-axis; standard R recycling rules imply that we effectively have one photon per mirror in `disco`. See how the photons' energy is unchanged by the reflection). We might ask what percentage of photons are reflected towards the source; but to do this we would need a somewhat more stylish disco ball, here with 1000 mirrors:

```
> table(reflect(p,dfun(1000))[,2]>0) # should be TRUE with probability sqrt(0.5)
```

```
FALSE  TRUE
  294   706
```

(compare the expected value of $1000/\sqrt{2} \simeq 707$). But it is perhaps more fun to consider a relativistic disco in which the mirror ball moves at 0.5c:

```
> B <- boost(as.3vel(c(0.5,0,0)))
> p %>% tcrossprod(B) %>% reflect(disco) %>% tcrossprod(solve(B))

      t      x       y       z
x 0.546 0.0913  0.5375 -0.0235
x 0.840 0.6808  0.0458 -0.4906
x 0.997 0.9943  0.0341  0.0672
x 0.702 0.4033 -0.1785 -0.5457
x 0.650 0.3006 -0.5671 -0.1043
x 0.828 0.6562 -0.2584  0.4340
x 0.818 0.6361  0.2973 -0.4197
```

Above, note the high energy of the photon in the third row. This is because the third mirror of `disco` is such that the photon hits it with grazing incidence; this means that the receding of the mirror is almost immaterial. Note further that a spinning disco ball would give the same (instantaneous) results.

9.3. Mirrors and rotation-boost coupling

Consider the following situation: we take a bunch of photons which in a certain reference frame are all moving (almost) parallel to the x-axis. Then we reflect the photons from a mirror which is moving with a composition of pure boosts, and examine the reflected light in their original reference frame. The R idiom for this would be:

```
> sol(1)

[1] 1

> light_start <- as.photon(as.3vel(cbind(0.9,1:5/40,5:1/40)))
> m <- c(1,0,0)     # mirror normal to x-axis
> B1 <- boost(as.3vel(c(-0.5, 0.1, 0.0)))
> B2 <- boost(as.3vel(c( 0.2, 0.0, 0.0)))
> B3 <- boost(as.3vel(c( 0.0, 0.0, 0.6)))
> B <- B1 %*% B2 %*% B3   # matrix multiplication is associative!
> light <- light_start %*% t(B)
> light <- reflect(light,m)
> light <- as.4mom(light %*% solve(t(B)))
> light

        E   p_x   p_y   p_z
[1,] 2.30 -2.11 0.119 0.920
[2,] 2.31 -2.13 0.147 0.897
[3,] 2.32 -2.14 0.175 0.874
[4,] 2.32 -2.15 0.203 0.849
[5,] 2.33 -2.16 0.230 0.824
```

See how the photons have picked up momentum in the y- and z- direction, even though the mirror is oriented perpendicular to the x-axis (in its own frame). Again it is arguably preferable to use pipes:

```
> light_start %>% tcrossprod(B) %>% reflect(m) %>% tcrossprod(solve(B)) %>% as.4mom
```

```
        E   p_x   p_y   p_z
[1,] 2.30 -2.11 0.119 0.920
[2,] 2.31 -2.13 0.147 0.897
[3,] 2.32 -2.14 0.175 0.874
[4,] 2.32 -2.15 0.203 0.849
[5,] 2.33 -2.16 0.230 0.824
```

Compare when the speed of light is infinite:

```
> sol(Inf)
```

```
[1] Inf
```

```
> light_start <- as.photon(as.3vel(cbind(0.9,1:5/40,5:1/40)))
> B1 <- boost(as.3vel(c(-0.5, 0.1, 0.0)))
> B2 <- boost(as.3vel(c( 0.2, 0.0, 0.0)))
> B3 <- boost(as.3vel(c( 0.0, 0.0, 0.6)))
> B <- B1 %*% B2 %*% B3
> light_start
```

```
     E   p_x    p_y    p_z
[1,] 0 0.990 0.0275 0.1375
[2,] 0 0.992 0.0551 0.1103
[3,] 0 0.993 0.0828 0.0828
[4,] 0 0.992 0.1103 0.0551
[5,] 0 0.990 0.1375 0.0275
```

```
> light_start %>% tcrossprod(B) %>% reflect(m) %>% tcrossprod(solve(B)) %>% as.4mom
```

```
     E    p_x    p_y    p_z
[1,] 0 -0.990 0.0275 0.1375
[2,] 0 -0.992 0.0551 0.1103
[3,] 0 -0.993 0.0828 0.0828
[4,] 0 -0.992 0.1103 0.0551
[5,] 0 -0.990 0.1375 0.0275
```

Note that, in the infinite light speed case, the energy of the photons is zero (photons have zero rest mass); further observe that in this classical case, the effect of the mirror is to multiply the x-momentum by -1 and leave the other components unchanged, as one might expect from a mirror perpendicular to $(1, 0, 0)$.

10. Three-velocities

In contrast to four-velocities, three-velocities do not form a group under composition as the velocity addition law is not associative (Ungar 2006). Instead, three-velocity composition has an algebraic structure known as a *gyrogroup* (this observation was the original motivation for the package). Ungar shows that the velocity addition law for three-velocities is

$$\mathbf{u}\oplus\mathbf{v}=\frac{1}{1+\mathbf{u}\cdot\mathbf{v}}\left\{\mathbf{u}+\frac{\mathbf{v}}{\gamma_{\mathbf{u}}}+\frac{\gamma_{\mathbf{u}}\left(\mathbf{u}\cdot\mathbf{v}\right)\mathbf{u}}{1+\gamma_{\mathbf{u}}}\right\} \tag{1}$$

where $\gamma_{\mathbf{u}}=\left(1-\mathbf{u}\cdot\mathbf{u}\right)^{-1/2}$ and we are assuming $c=1$. Ungar goes on to show that, in general, $\mathbf{u}\oplus\mathbf{v}\neq\mathbf{v}\oplus\mathbf{u}$ and $\left(\mathbf{u}\oplus\mathbf{v}\right)\oplus\mathbf{w}\neq\mathbf{u}\oplus\left(\mathbf{v}\oplus\mathbf{w}\right)$. He also defines the binary operator $\ominus$ as $\mathbf{u}\ominus\mathbf{v}=\mathbf{u}\oplus\left(-\mathbf{v}\right)$, and implicitly defines $\ominus\mathbf{u}\oplus\mathbf{v}$ to be $\left(-\mathbf{u}\right)\oplus\mathbf{v}$. If we have

$$\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{x}=-\left(\mathbf{u}\oplus\mathbf{v}\right)\oplus\left(\mathbf{u}\oplus\left(\mathbf{v}\oplus\mathbf{x}\right)\right) \tag{2}$$

then

$$\begin{aligned}
\mathbf{u}\oplus\mathbf{v} &= \operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\left(\mathbf{v}\oplus\mathbf{u}\right) && (3)\\
\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{x}\cdot\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{y} &= \mathbf{x}\cdot\mathbf{y} && (4)\\
\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\left(\mathbf{x}\oplus\mathbf{y}\right) &= \operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{x}\oplus\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{y} && (5)\\
\left(\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\right)^{-1} &= \left(\operatorname{gyr}\left[\mathbf{v},\mathbf{u}\right]\right) && (6)\\
\mathbf{u}\oplus\left(\mathbf{v}\oplus\mathbf{w}\right) &= \left(\mathbf{u}\oplus\mathbf{v}\right)\oplus\operatorname{gyr}\left[\mathbf{u},\mathbf{v}\right]\mathbf{w} && (7)\\
\left(\mathbf{u}\oplus\mathbf{v}\right)\oplus\mathbf{w} &= \mathbf{u}\oplus\left(\mathbf{v}\oplus\operatorname{gyr}\left[\mathbf{v},\mathbf{u}\right]\mathbf{w}\right) && (8)
\end{aligned}$$

Consider the following R session:

```
> sol(1)

[1] 1

> u <- as.3vel(c(-0.7,+0.2,-0.3))
> v <- as.3vel(c(+0.3,+0.3,+0.4))
> w <- as.3vel(c(+0.1,+0.3,+0.8))
> x <- as.3vel(c(-0.2,-0.1,-0.9))
> u

A vector of three-velocities (speed of light = 1)
        x   y    z
[1,] -0.7 0.2 -0.3
```

Here we have three-vectors `u` etc. We can see that `u` and `v` do not commute:

```
> u+v
```

```
A vector of three-velocities (speed of light = 1)
          x     y        z
[1,] -0.545 0.482 -0.00454
```

```
> v+u
```

```
A vector of three-velocities (speed of light = 1)
          x     y     z
[1,] -0.429 0.572 0.132
```

(the results differ). We can use equation 3

```
> (u+v)-gyr(u,v,v+u)
```

```
A vector of three-velocities (speed of light = 1)
            x         y        z
[1,] 1.77e-16 -7.08e-16 1.23e-16
```

showing agreement to within numerical error. It is also possible to use the functional idiom in which we define `f()` to be the map $\mathbf{x} \mapsto \mathrm{gyr}\left[\mathbf{u}, \mathbf{v}\right]\mathbf{x}$. In R:

```
> f <- gyrfun(u,v)
> (u+v)-f(v+u)    # should be zero
```

```
A vector of three-velocities (speed of light = 1)
            x         y        z
[1,] 1.77e-16 -7.08e-16 1.23e-16
```

Function `gyrfun()` is vectorized, which means that it plays nicely with (R) vectors. Consider

```
> u9 <- r3vel(9)
> u9
```

```
A vector of three-velocities (speed of light = 1)
            x       y       z
 [1,] -0.6635 -0.1850 -0.0324
 [2,]  0.7232  0.3007  0.0448
 [3,] -0.4816 -0.2698  0.0903
 [4,]  0.7186  0.5215  0.1149
 [5,] -0.4988 -0.5469  0.4781
 [6,] -0.0446  0.7934 -0.4839
 [7,]  0.4225  0.6831 -0.2474
 [8,] -0.5546 -0.1326  0.4232
 [9,]  0.0366  0.0748  0.4407
```

Then we can create a vectorized gyrofunction:

```
> f <- gyrfun(u9,v)
> f(x)

A vector of three-velocities (speed of light = 1)
            x         y      z
 [1,] -0.0282 -7.04e-02 -0.924
 [2,] -0.3568 -1.36e-01 -0.845
 [3,] -0.0692 -2.67e-02 -0.924
 [4,] -0.3503 -1.89e-01 -0.838
 [5,]  0.0444  1.59e-01 -0.913
 [6,] -0.2297 -4.52e-01 -0.776
 [7,] -0.3268 -3.10e-01 -0.811
 [8,]  0.0130  3.85e-06 -0.927
 [9,] -0.1409 -5.01e-02 -0.915
```

Note that the package vectorization is transparent when using syntactic sugar:

```
> u9+x

A vector of three-velocities (speed of light = 1)
            x       y       z
 [1,] -0.7436 -0.2345 -0.5826
 [2,]  0.6411  0.2532 -0.6615
 [3,] -0.6319 -0.3444 -0.6273
 [4,]  0.6860  0.5266 -0.4421
 [5,] -0.6903 -0.6791 -0.0511
 [6,] -0.0952  0.7096 -0.6909
 [7,]  0.3115  0.6168 -0.6976
 [8,] -0.8231 -0.2462 -0.3690
 [9,] -0.2550 -0.0524 -0.7806
```

(here, the addition operates using R's standard recycling rules).

10.1. Associativity

Three velocity addition is not associative:

```
> (u+v)+w

A vector of three-velocities (speed of light = 1)
           x     y     z
[1,] -0.465 0.655 0.501

> u+(v+w)

A vector of three-velocities (speed of light = 1)
           x     y     z
[1,] -0.549 0.667 0.416
```

But we can use equations 7 and 8:

```
> (u+(v+w)) - ((u+v)+gyr(u,v,w))

A vector of three-velocities (speed of light = 1)
             x         y         z
[1,] 6.92e-16 -1.38e-15 -6.92e-16

> ((u+v)+w) - (u+(v+gyr(v,u,w)))

A vector of three-velocities (speed of light = 1)
     x y        z
[1,] 0 0 5.35e-16
```

10.2. Visualization of noncommutativity and nonassociativity of three-velocities

Consider the following three-velocities:

```
> u <- as.3vel(c(0.4,0,0))
> v <- seq(as.3vel(c(0.4,-0.2,0)), as.3vel(c(-0.3,0.9,0)),len=20)
> w <- as.3vel(c(0.8,-0.4,0))
```

Objects **v** and **w** are single three-velocities, and object **v** is a vector of three velocities. We can see the noncommutativity of three velocity addition in figures 1 and 2, and the nonassociativity in figure 3.

10.3. The magrittr package: pipes

Three velocities in the **lorentz** package work nicely with **magrittr**. If we define

```
>  u <- as.3vel(c(+0.5,0.1,-0.2))
>  v <- as.3vel(c(+0.4,0.3,-0.2))
>  w <- as.3vel(c(-0.3,0.2,+0.2))
```

Then pipe notation operates as expected:

```
> jj1 <- u %>% add(v)
> jj2 <- u+v
> speed(jj1-jj2)

[1] 2.21e-16
```

The pipe operator is left associative:

```
> comm_fail1(u=u, v=v)
```

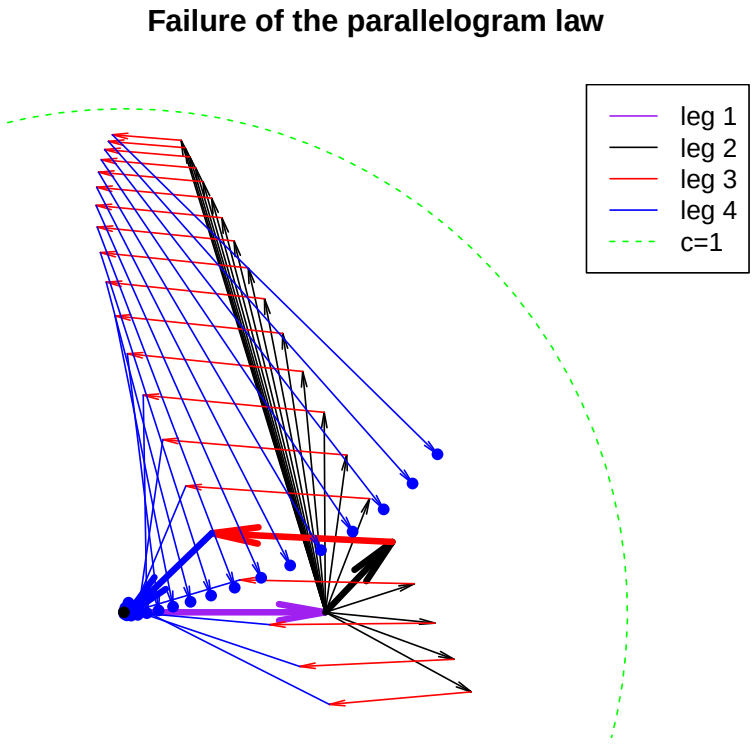


Figure 1: Failure of the commutative law for velocity composition in special relativity. The arrows show successive velocity boosts of $+\mathbf{u}$ (purple), $+\mathbf{v}$ (black), $-\mathbf{u}$ (red), and $-\mathbf{v}$ (blue) for $\mathbf{u}, \mathbf{v}$ as defined above. Velocity $\mathbf{u}$ is constant, while $\mathbf{v}$ takes a sequence of values. If velocity addition is commutative, the four boosts form a closed quadrilateral; the thick arrows show a case where the boosts almost close and the boosts nearly form a parallelogram. The blue dots show the final velocity after four successive boosts; the distance of the blue dot from the origin measures the combined velocity, equal to zero in the classical limit of low speeds. The discrepancy becomes larger and larger for the faster elements of the sequence $\mathbf{v}$

```
> comm_fail2(u=u, v=v)
```

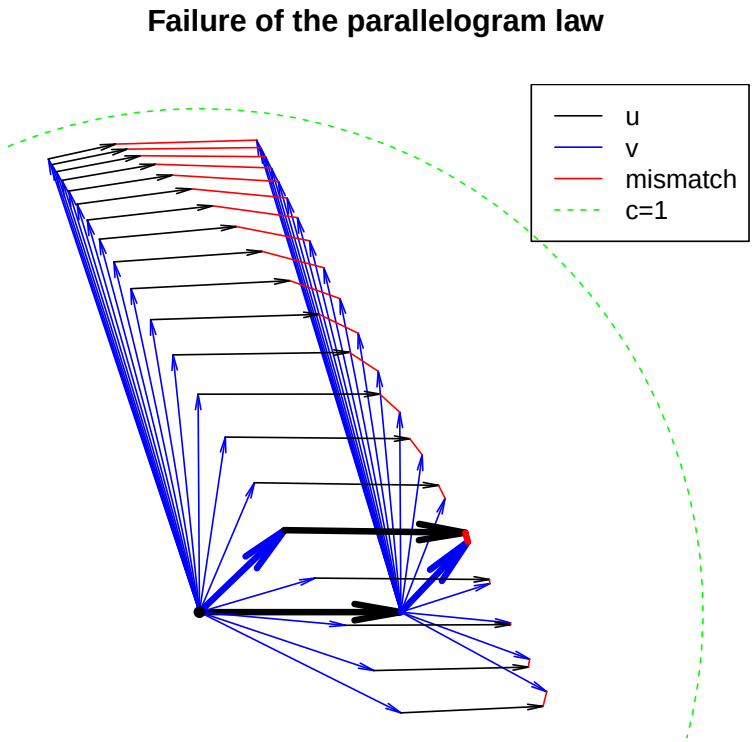


Figure 2: Another view of the failure of the commutative law in special relativity. The black arrows show velocity boosts of $\mathbf{u}$ and the blue arrows show velocity boosts of $\mathbf{v}$, with $\mathbf{u}, \mathbf{v}$ as defined above; $\mathbf{u}$ is constant while $\mathbf{v}$ takes a sequence of values. If velocity addition is commutative, then $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$ and the two paths end at the same point: the parallelogram is closed. The red lines show the difference between $\mathbf{u} + \mathbf{v}$ and $\mathbf{v} + \mathbf{u}$

```
> ass_fail(u=u, v=v, w=w, bold=10)
```

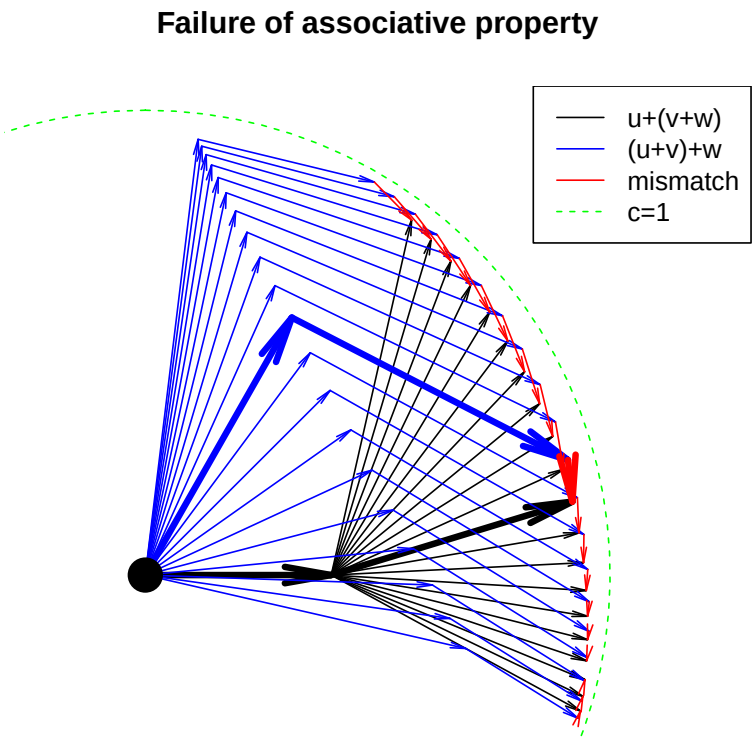


Figure 3: Failure of the associative law for velocity composition in special relativity. The arrows show successive boosts of $\mathbf{u}$ followed by $\mathbf{v}+\mathbf{w}$ (black lines), and $\mathbf{u}+\mathbf{v}$ followed by $\mathbf{w}$ (blue lines), for $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ as defined above; $\mathbf{u}$ and $\mathbf{w}$ are constant while $\mathbf{v}$ takes a sequence of values. The mismatch between $\mathbf{u}+(\mathbf{v}+\mathbf{w})$ and $(\mathbf{u}+\mathbf{v})+\mathbf{w}$ is shown in red

```
> jj1 <- u %>% add(v) %>% add(w)
> jj2 <- (u+v)+w
> speed(jj1-jj2)

[1] 7.39e-17
```

If we want right associative addition, the pipe operator needs brackets:

```
> jj1 <- u %>% add(v %>% add(w))
> jj2 <- u+(v+w)
> speed(jj1-jj2)

[1] 3.45e-17
```

10.4. Numerical verification

Here I provide numerical verification of equations 3 to 8. If we have

```
> x <- as.3vel(c(0.7, 0.0, -0.7))
> y <- as.3vel(c(0.1, 0.3, -0.6))
> u <- as.3vel(c(0.0, 0.8, +0.1))   # x,y,u: single three-velocities
> v <- r3vel(5,0.9)
> w <- r3vel(5,0.8)                 # v,w: vector of three-velocities
> f <- gyrfun(u,v)
> g <- gyrfun(v,u)
```

Then we can calculate the difference between the left hand side and right hand side numerically:

```
> max(speed((u+v) - f(v+u)))              # equation 3

[1] 3.49e-13

> max(abs(prod3(f(x),f(y)) - prod3(x,y))) # equation 4

[1] 5.94e-15

> max(speed(f(x+y) - (f(x)+f(y))))        # equation 5

[1] 3.81e-12

> max(speed(f(g(x)) - g(f(x))))           # equation 6

[1] 7.3e-13
```

```
> max(speed((u+(v+w)) - ((u+v)+f(w))))    # equation 7

[1] 6.25e-14

> max(speed(((u+v)+w) - (u+(v+g(w)))))    # equation 8

[1] 2.78e-14
```

(all zero to numerical precision).

11. Conclusions

The **lorentz** package furnishes some functionality for manipulating four-vectors and three-velocities in the context of special relativity. The R idiom is relatively natural and the package has been used to illustrate different features of relativistic kinematics.

References

Gjurchinovski A (2004). "Reflection of light from a uniformly moving mirror." *American Journal of Physics*, **72**(10), 1316–1324.

Goldstein H (1980). *Classical mechanics*. Second edition. Addison-Wesley.

Hankin R (2025). "Special relativity in R: the lorentz package." *Journal of Open Source Education*, **8**(88). doi:10.21105/jose.00196. URL https://doi.org/10.21105/jose.00196.

Horwitz P, Taylor EF, Barowy W (1994). "Teaching special relativity with a computer." *Computers in Physics*, **8**(1), 92–97.

Schutz B (1985). *A first course in general relativity*. Cambridge University Press.

Sherin ZW, Cheu R, Tan P, Kortemeyer G (2016). "Visualizing relativity: the OpenRelativity project." *American Journal of Physics*, **84**(5), 369–374. URL https://github.com/MITGameLab/OpenRelativity/.

Ungar AA (2006). "Thomas precession: a kinematic effect of the algebra of Einstein's velocity addition law. Comments on 'Deriving relativistic momentum and energy: II. Three-dimensional case'." *European Journal of Physics*, **27**, L17–L20.

Affiliation:

Robin K. S. Hankin
University of Stirling
Scotland
email: hankin.robin@gmail.com