Package ‘lorentz’

January 23, 2025

Type Package

Title The Lorentz Transform in Relativistic Physics

Version 1.1-2

Suggests knitr,testthat,rmarkdown,covr

Imports quadform,tensor,magic,magrittr

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Depends R (>= 4.1.0)

Description The Lorentz transform in special relativity; also the gyrogroup structure of three-velocities. Performs active and passive transforms and has the ability to use units in which the speed of light is not unity. Includes some experimental functionality for celerity and rapidity. For general relativity, see the 'schwarzschild' package.

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

URL https://github.com/RobinHankin/lorentz,
https://robinhankin.github.io/lorentz/

BugReports https://github.com/RobinHankin/lorentz/issues

RoxygenNote 7.2.3

NeedsCompilation no

Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Repository CRAN

Date/Publication 2025-01-23 16:20:01 UTC

Contents

`lorentz-package` *The Lorentz Transform in Relativistic Physics*

Description

The Lorentz transform in special relativity; also the gyrogroup structure of three-velocities. Performs active and passive transforms and has the ability to use units in which the speed of light is not unity. Includes some experimental functionality for celerity and rapidity. For general relativity, see the 'schwarzschild' package.

Details

The DESCRIPTION file:

Package:	lorentz
Type:	Package
Title:	The Lorentz Transform in Relativistic Physics
Version:	1.1-2
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gmail.co
Suggests:	knitr,testthat,rmarkdown,covr
Imports:	quadform,tensor,magic,magrittr
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Depends:	R (>= 4.1.0)
Description:	The Lorentz transform in special relativity; also the gyrogroup structure of three-velocities. Performs activ

License:	GPL-3
Encoding:	UTF-8
VignetteBuilder:	knitr
URL:	https://github.com/RobinHankin/lorentz, https://robinhankin.github.io/lorentz/
BugReports:	https://github.com/RobinHankin/lorentz/issues
RoxygenNote:	7.2.3
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

```
Extract.3vel            Extract or replace parts of three-velocity
Ops.3vel                Arithmetic Ops Group Methods for 3vel objects
as.matrix.3vel          Coerce 3-vectors and 4-vectors to a matrix
boost                   Lorentz transformations
c.3vel                  Combine vectors of three-velocities and
                        four-velocities into a single vector
celerity                Celerity and rapidity
comm_fail               Failure of commutativity and associativity
                        using visual plots
coordnames              Coordinate names for relativity
cosines                 Direction cosines
fourmom                 Four momentum
fourvel                 Four velocities
galileo                 Classical mechanics; Newtonian approximation;
                        infinite speed of light
gam                     Gamma correction
gyr                     Gyr function
lorentz-package         The Lorentz Transform in Relativistic Physics
photon                  Photons
print.3vel              Print methods for three-velocities and
                        four-velocities
r3vel                   Random relativistic velocities
reflect                 Mirrors
seq.3vel                seq method for three velocities
sol                     Speed of light and Minkowski metric
threevel                Three velocities
transform               The energy-momentum tensor
```

Author(s)

Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

References

- Ungar 2006. “Thomas precession: a kinematic effect...”. *European Journal of Physics*, 27:L17-L20.

- https://www.youtube.com/watch?v=9Y9CxiukURw&index=68&list=PL9_n3Tqzq9iWtgD8POJFdnVUCZ_zw6OiB

Examples

```
u <- as.3vel(c(0.3,0.6,-0.1))  # u is a three-velocity
gam(u)                         # relativistic gamma term for u
U <- as.4vel(u)                # U is a four-velocity
B1 <- boost(u)                 # B1 is the Lorentz transform matrix for u
B1 %*% c(1,0,0,0)              # Lorentz transform of zero 4-velocity (=-u)

B2 <- boost(as.3vel(c(-0.1,0.8,0.3)))
B3 <- boost(as.3vel(c(-0.1,0.1,0.9)))  # more boosts

Bi <- B1 %*% B2  # Bi is the boost for successive Lorentz transforms


pureboost(Bi)      # Decompose Bi into a pure boost...
orthog(Bi)         # and an orthogonal matrix

Bj <- B2 %*% B1    # B1 and B2 do not commute...

(B1 %*% B2) %*% B3
B1 %*% (B2 %*% B3)    # ...but composition *is* associative



## Three velocities and the gyrogroup

## Create some random three-velocities:

u <- r3vel(10)
v <- r3vel(10)
w <- r3vel(10)

u+v
v+u        # Three-velocity addition is not commutative...

u+(v+w)   # ... nor associative
(u+v)+w
```

`as.matrix.3vel` *Coerce 3-vectors and 4-vectors to a matrix*

Description

Coerce 3-vectors and 4-vectors to a matrix. A convenience wrapper for `unclass()`

Usage

```
## S3 method for class '3vel'
as.matrix(x, ...)
## S3 method for class '4vel'
as.matrix(x, ...)
```

Arguments

x	Object of class 3vel or 4vel
...	Further arguments (currently ignored)

Author(s)

Robin K. S. Hankin

Examples

```
as.matrix(r3vel(5))
as.matrix(r4vel(5))
```

---

boost *Lorentz transformations*

Description

Lorentz transformations: boosts and rotations

Usage

```
boost(u=0)
rot(u,v,space=TRUE)
is.consistent.boost(L, give=FALSE, TOL=1e-10)
is.consistent.boost.galilean(L, give=FALSE, TOL=1e-10)
pureboost(L,include_sol=TRUE)
orthog(L)
pureboost.galilean(L, tidy=TRUE)
orthog.galilean(L)
```

Arguments

u, v	Three-velocities, coerced to class 3vel. In function boost(), if u takes the special default value 0, this is interpreted as zero three velocity
L	Lorentz transform expressed as a 4×4 matrix
TOL	Numerical tolerance
give	Boolean with TRUE meaning to return the transformed metric tensor (which should be the flat-space eta(); qv) and default FALSE meaning to return whether the matrix is a consistent boost or not

`space`	Boolean, with default `TRUE` meaning to return just the spatial component of the rotation matrix and `FALSE` meaning to return the full 4×4 matrix transformation
`tidy`	In `pureboost.galilean()`, Boolean with default `TRUE` meaning to return a "tidy" boost matrix with spatial components forced to be a 3×3 identity matrix
`include_sol`	In function `pureboost()`, Boolean with default `TRUE` meaning to correctly account for the speed of light, and `FALSE` meaning to assume $c = 1$. See details

Details

Arguments `u,v` are coerced to three-velocities.

A rotation-free Lorentz transformation is known as a *boost* (sometimes a *pure boost*), here expressed in matrix form. Pure boost matrices are symmetric if $c = 1$. Function `boost(u)` returns a 4×4 matrix giving the Lorentz transform of an arbitrary three-velocity `u`.

Boosts can be successively applied with regular matrix multiplication. However, composing two successive pure boosts does not in general return a pure boost matrix: the product is not symmetric in general. Also note that boost matrices do not commute. The resulting matrix product represents a *Lorentz transform*.

It is possible to decompose a Lorentz transform L into a pure boost and a spatial rotation. Thus $L = OP$ where O is an orthogonal matrix and P a pure boost matrix; these are returned by functions `orthog()` and `pureboost()` respectively. If the speed of light is not equal to 1, the functions still work but can be confusing.

Functions `pureboost.galilean()` and `orthog.galilean()` are the Newtonian equivalents of `pureboost()` and `orthog()`, intended to be used when the speed of light is infinite (which causes problems for the relativistic functions).

As noted above, the composition of two pure Lorentz boosts is not necessarily pure. If we have two successive boosts corresponding to u and v, then the composed boost may be decomposed into a pure boost of `boost(u+v)` and a rotation of `rot(u,v)`.

The reason argument `include_sol` exists is that function `orthog()` needs to call `pureboost()` in an environment where we pretend that $c = 1$.

Value

Function `boost()` returns a 4×4 matrix; function `rot()` returns an orthogonal matrix.

Note

Function `rot()` uses `crossprod()` for efficiency reasons but is algebraically equivalent to

`boost(-u-v) %*% boost(u) %*% boost(v)`.

Author(s)

Robin K. S. Hankin

References

- Ungar 2006. "Thomas precession: a kinematic effect...". European Journal of Physics, 27:L17-L20
- Sbitneva 2001. "Nonassociative geometry of special relativity". International Journal of Theoretical Physics, volume 40, number 1, pages 359–362
- Wikipedia contributors 2018. "Wigner rotation", Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Wigner_rotation&oldid=838661305. Online; accessed 23 August 2018

Examples

```
boost(as.3vel(c(0.4,-0.2,0.1)))

u <- r3vel(1)
v <- r3vel(1)
w <- r3vel(1)

boost(u) - solve(boost(-u))  # should be zero

boost(u) %*% boost(v)   # not a pure boost (not symmetrical)
boost(u+v)  # not the same!
boost(v+u)  # also not the same!

u+v  # returns a three-velocity


boost(u) %*% boost(v) %*% boost(w)  # associative, no brackets needed
boost(u+(v+w))  # not the same!
boost((u+v)+w)  # also not the same!


rot(u,v)
rot(v,u)    # transpose (=inverse) of rot(u,v)


rot(u,v,FALSE) %*% boost(v) %*% boost(u)
boost(u+v)     # should be the same.


orthog(boost(u) %*% boost(v)) - rot(u,v,FALSE)  # zero to numerical precision
pureboost(boost(v) %*% boost(u)) - boost(u+v)   # ditto


## Define a random-ish Lorentz transform
L <- boost(r3vel(1)) %*% boost(r3vel(1)) %*% boost(r3vel(1))

## check it:


## Not run:    # needs emulator package
quad.form(eta(),L)  # should be eta()
```

```
## End(Not run)

## More concisely:
is.consistent.boost(L)     # should be TRUE

## Decompose L into a rotation and a pure boost:
U <- orthog(L)
P <- pureboost(L)

L - U %*% P               # should be zero (L = UP)
crossprod(U)                # should be identity (U is orthogonal)
P - t(P)                    # should be zero (P is symmetric)

## First row of P should be a consistent 4-velocity:
is.consistent.4vel(P[1,,drop=FALSE],give=TRUE)
```

`c.3vel` *Combine vectors of three-velocities and four-velocities into a single vector*

Description

Combines its arguments recursively to form a vector of three velocities or four velocities

Usage

```
## S3 method for class '3vel'
c(...)
## S3 method for class '3cel'
c(...)
## S3 method for class '4vel'
c(...)
```

Arguments

`...`	Vectors of three-velocities

Details

Returns a vector of three-velocities or four-velocities. These are stored as three- or four- column matrices; each row is a velocity.

Names are inherited from the behaviour of `cbind()`, not `c()`.

Note

This function is used extensively in `inst/distributive_search.R`.

For "c" as in celerity or speed of light, see `sol()`.

Author(s)

Robin K. S. Hankin

See Also

sol

Examples

```
c(r3vel(3),r3vel(6,0.99))
```

---

celerity *Celerity and rapidity*

Description

The celerity and rapidity of an object (experimental functionality)

Usage

```
## S3 method for class '3vel'
celerity(u)
## S3 method for class '4vel'
celerity(u)
celerity_ur(d)
## S3 method for class '3vel'
rapidity(u)
## S3 method for class '4vel'
rapidity(u)
rapidity_ur(d)
as.3cel(x)
cel_to_vel(x)
vel_to_cel(x)
```

Arguments

u, x	Speed: either a vector of speeds or a vector of three-velocities or four-velocities
d	In functions `celerity_ur()` and `rapidity_ur()`, deficit of speed; speed of light minus speed of object

Details

The *celerity* corresponding to speed u is defined as $u\gamma$ and the *rapidity* is $c \cdot \operatorname{atanh}(u/c)$.

Functions `celerity_ur()` and `rapidity_ur()` are used for the ultrarelativistic case where speeds are very close to the speed of light. Its argument `d` is the deficit, that is, $d = c - v$ where v is the speed of the transformation. Algebraically, `celerity_ur(c-v) == celerity(v)`, but if $d = 1-v/c$ is small the result of `celerity_ur()` is more accurate than that of `celerity()`.

Things get a bit sticky for celerity and rapidity if $c \neq 1$. The guiding principle in the package is to give the celerity and rapidity the same units as c, so if $u \ll c$ we have that all three of `celerity(u)`, `rapidity(u)` and `u` are approximately equal. Note carefully that, in contrast, γ is dimensionless. Also observe that `d` in functions `celerity_ur()` and `rapidity_ur()` has the same units as c.

Author(s)

Robin K. S. Hankin

See Also

`gam`

Examples

```
u <- 0.1  # c=1
c(u,celerity(u),rapidity(u))

omgp <- 4.9e-24  # speed deficit of the Oh-My-God particle
c(celerity_ur(omgp),rapidity_ur(omgp))


sol(299792458)                 # use SI units
u <- 3e7  # ~0.1c
c(u,celerity(u),rapidity(u))


snail <- 0.00275
c(snail,celerity(snail),rapidity(snail))


omgp <- omgp*sol()
c(celerity_ur(omgp),rapidity_ur(omgp))


sol(1)
```

comm_fail *Failure of commutativity and associativity using visual plots*

Description

Relativistic addition of three-velocities is neither commutative nor associative, and the functions documented here show this visually.

Usage

```
comm_fail1(u, v, bold=5, r=1)
comm_fail2(u, v, bold=5, r=1)
ass_fail(u, v, w, bold=5,r=1)
my_seg(u,start=as.3vel(0), bold=5, ...)
```

Arguments

u, v, w, start	Three velocities. Arguments u and w are single-element three velocities, argument v is a vector. See the examples
bold	Integer specifying which vector element to be drawn in bold
r	Radius of dotted green circle, defaulting to 1 (corresponding to $c = 1$). Use NA to suppress plotting of circle
...	Further arguments, passed to arrows()

Value

These functions are called for their side-effect of plotting a diagram.

Note

The vignette lorentz gives more details and interpretation of the diagrams.

Function my_seg() is an internal helper function.

Author(s)

Robin K. S. Hankin

Examples

```
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))

comm_fail1(u=u, v=v)
comm_fail2(u=u, v=v)
  ass_fail(u=u, v=v, w=w, bold=10)
```

`coordnames` *Coordinate names for relativity*

Description

Trivial function to set coordinate names to "`t`", "`x`", "`y`", "`z`".

Usage

```
coordnames(...)
flob(x)
```

Arguments

`...`	Further arguments, currently ignored
`x`	A matrix

Details

Function `coordnames()` simply returns the character string `c("t","x","y","z")`. It may be overwritten. Function `flob()` sets the row and columnnames of a 4×4 matrix to `coordnames()`.

Note

If anyone can think of a better name than `flob()` let me know.

Author(s)

Robin K. S. Hankin

Examples

```
coordnames()

flob(diag(3))
flob(matrix(1,4,4))

## You can change the names if you wish:
coordnames <- function(x){letters[1:4]}
flob(outer(1:4,1:4))
```

`cosines` *Direction cosines*

Description

Given a vector of three-velocities, returns their direction cosines

Usage

```
cosines(u, drop = TRUE)
```

Arguments

`u`	A vector of three-velocities
`drop`	Boolean, with default `TRUE` meaning to coerce return value from a one-row matrix to a vector, and `FALSE` meaning to consistently return a matrix

Author(s)

Robin K. S. Hankin

Examples

```
cosines(r3vel(7))


cosines(r3vel(1),drop=TRUE)
cosines(r3vel(1),drop=FALSE)
```

`Extract.3vel` *Extract or replace parts of three-velocity*

Description

Extract or replace subsets of three-velocities

Arguments

`x`	A three-vector
`index`	elements to extract or replace
`value`	replacement value

Details

These methods (should) work as expected: an object of class `3vel` is a three-column matrix with rows corresponding to three-velocities; a single argument is interpreted as a row number. Salient use-cases are `u[1:5] <- u[1]` and `u[1] <- 0`.

To extract a single component, pass a second index: `u[,1]` returns the x- component of the three-velocity.

Extraction functions take a `drop` argument, except for `x[i]` which returns a `vec` object.

Currently, `u[]` returns `u` but I am not sure this is desirable. Maybe it should return `unclass(u)` or perhaps `c(unclass(u))`.

Use idiom `u[] <- x` to replace entries of `u` elementwise.

Examples

```
u <- r3vel(10)
u[1:4]
u[5:6] <- 0

u[7:8] <- u[1]

u[,1] <- 0.1
```

`fourmom` *Four momentum*

Description

Create and test for four-momentum

Usage

```
## S3 method for class '4mom'
Ops(e1, e2)
## S3 method for class '4mom'
sum(..., na.rm=FALSE)
vel_to_4mom(U,m=1)
p_to_4mom(p,E=1)
as.4mom(x)
is.4mom(x)
fourmom_mult(P,n)
fourmom_add(e1,e2)
```

Arguments

`x, P, e1, e2`	Four-momentum
`p`	Three-momentum
`E`	Scalar; energy
`U`	Object coerced to four-velocity
`m`	Scalar; rest mass
`n`	Multiplying factor
`..., na.rm`	Arguments sent to `sum()`

Details

Four-momentum is a relativistic generalization of three-momentum, with the object's energy as the first element. It can be defined as mU, where m is the rest mass and U the four-velocity. Equivalently, one can define four-momentum as $(E/c, p_x, p_y, p_z)$ where E is the energy and (p_x, p_y, p_z) the three-momentum.

Function `vel_to_4mom()` converts three-velocity to four-momentum, and function `p_to_4mom()`) converts a three-momentum to a four-momentum.

The function `Ops.4mom()` passes unary and binary arithmetic operators "+", "-" and "*" to the appropriate specialist function.

The package is designed so that natural R idiom may be used for physically meaningful operations such as combining momenta of different objects, using the conservation of four-momentum.

For the four-momentum of a photon, use `as.photon()`.

Author(s)

Robin K. S. Hankin

See Also

`boost`,`as.photon`

Examples

```
# Define 5 random three velocities:
v <- r3vel(5)

# convert to four-velocity:
as.4vel(v)

# Now convert 'v' to four-momentum, specifying rest mass:
vel_to_4mom(v)         # 4mom of five objects with 3vel v, all unit mass
vel_to_4mom(v,   1:5)  # 4mom of five objects with 3vel v, masses 1-5
vel_to_4mom(v[1],1:5)  # 4mom of five objects with same 3vel, masses 1..5

# Now convert 'v' to four-momentum, specifying energy E:
p_to_4mom(v,E=1)
p_to_4mom(v,E=10)   # slower
```

```
p_to_4mom(v,E=100)  # even slower

# Four-momentum of objects moving closely parallel to the x-axis:
P <- vel_to_4mom(as.3vel(c(0.8,0,0)) + r3vel(7,0.01))

reflect(P)
reflect(P,c(1,1,1))

sum(P)
```

`fourvel` *Four velocities*

Description

Create and test for four-velocities.

Usage

```
as.4vel(u)
is.consistent.4vel(U, give=FALSE, TOL=1e-10)
inner4(U,V=U)
to3(U)
```

Arguments

`u`	A vector of three-velocities
`U, V`	A vector of four-velocities
`give`	In function `is.consistent.4vel()`, Boolean with `TRUE` meaning to return $U \cdot U + c^2$, which is zero for a four-velocity, and default `FALSE` meaning to return whether the four-velocity is consistent to numerical precision
`TOL`	Small positive value used for tolerance

Details

Function `as.4vel()` takes a three-velocity and returns a four-velocity.

Given a four-vector V, function `inner4()` returns the Lorentz invariant $V^iV_i = \eta_{ij}V^iV^j$. This quantity is unchanged under Lorentz transforms. Note that function `inner4()` works for any four-vector, not just four-velocities. It will work for (eg) a four-displacement, a four-momentum vector or a four-frequency. In electromagnetism, we could have a four-current or a four-potential. If U is a four-velocity, then $U^iU_i = -c^2$; if U is a 4-displacement, then U^iU_i is the squared interval. If P is the four-momentum of a photon then $P^iP_i = 0$.

Function `to3()` is a low-level helper function used when `as.3vel()` is given a four-velocity.

Function `is.consistent.4vel()` checks for four-velocities being consistent in the sense that $U^iU_i = -c^2$. Giving this function a vector, for example, `is.consistent.4vel(1:5)`, will return an error.

Compare the functions documented here with `boost()`, which returns a 4×4 transformation matrix (which also includes rotation information).

Author(s)

Robin K. S. Hankin

See Also

boost

Examples

```
a <- r3vel(10)
as.4vel(a)     # a four-velocity

as.3vel(as.4vel(a))-a   # zero to numerical precision

inner4(as.4vel(a))   #  -1 to numerical precision

stopifnot(all(is.consistent.4vel(as.4vel(a))))


## check Lorentz invariance of dot product:
U <- as.4vel(r3vel(10))
V <- as.4vel(r3vel(10))
B <- boost(as.3vel(1:3/10))

frame1dotprod <- inner4(U, V)
frame2dotprod <- inner4(U %*% B, V %*% B)
max(abs(frame1dotprod-frame2dotprod))  # zero to numerical precision
```

---

galileo *Classical mechanics; Newtonian approximation; infinite speed of light*

Description

The Lorentz transforms reduce to their classical limit, the Galilean transforms, if speeds are low compared with c. Package idiom for working in a classical framework is to use an infinite speed of light: `sol(Inf)`. Here I show examples of this.

Author(s)

Robin K. S. Hankin

See Also

boost

Examples

```
sol(Inf)
boost(as.3vel(1:3))
as.3vel(1:3) + as.3vel(c(-1,4,5))     # classical velocity addition
rot(as.3vel(1:3),as.3vel(c(-4,5,2)))  # identity matrix


B <- boost(as.3vel(1:3))
orthog(B) %*% pureboost(B)  # should be B

sol(1)
```

---

gam *Gamma correction*

Description

Lorentz gamma correction term in special relativity

Usage

```
## S3 method for class '3vel'
speed(u)
## S3 method for class '4vel'
speed(u)
speedsquared(u)
gam(u)
gamm1(u)
## S3 method for class '3vel'
gam(u)
## S3 method for class '3cel'
gam(u)
## S3 method for class '4vel'
gam(u)
## S3 method for class '3vel'
gamm1(u)
## S3 method for class '4vel'
gamm1(u)
gam_ur(d)
```

Arguments

u	Speed: either a vector of speeds or a vector of three-velocities or four-velocities
d	In function `gam_ur()`, deficit of speed; speed of light minus speed of object

Details

Function speed(u) returns the speed of a 3vel object or 4vel object.

Function gam(u) returns the Lorentz factor

$$\frac{1}{\sqrt{1-\mathbf{u}\cdot\mathbf{u}/c^2}}$$

Function gamm1(u) returns the Lorentz factor minus 1, useful for slow speeds when larger accuracy is needed (much like expm1()); to see the R idiom, type "gamm1.3vel" at the commandline. Function gamm1() is intended to work with 3vel objects or speeds. The function will take a 4-velocity, but this is not recommended as accuracy is lost (all it does is return the time component of the 4-velocity minus 1).

Function gam_ur() is used for the ultrarelativistic case where speeds are very close to the speed of light (the function is named for "gamma, ultrarelativistic"). Its argument d is the deficit, that is, $c-v$ where v is the speed of the transformation. Algebraically, gam_ur(c-v) == gam(v), but if d is small compared to c the result is more accurate.

Function speedsquared(u) returns the square of the speed of a 3vel object. Use this to avoid taking a needless square root.

Author(s)

Robin K. S. Hankin

Examples

```
gam(seq(from=0,by=0.1,len=10))
gam(r3vel(6,0.7))


x <- as.3vel(c(0.1,0.4,0.5))
speed(x)

gam(speed(x))  # works, but slow and inaccurate
gam(x)         # recommended: avoids needless coercion



## Use SI units and deal with terrestrial speeds.  Use gamm1() for this.
sol(299792458)
sound <- 343 # speed of sound in SI
gam(sound)
gam(sound)-1
gamm1(sound)   # gamm1() gives much higher precision

snail <- as.3vel(c(0.00275,0,0)) # even the world's fastest snail...
gamm1(snail)                     # ...has only a small relativistic correction


## For the ultrarelativistic case of speeds very close to the speed of
## light, use gam_ur():
```

```
sol(1)          # revert to relativistic units

gam(0.99) - gam_ur(0.01) # zero to numerical accuracy

omgp <- 4.9e-24  # speed deficit of the Oh-My-God particle
gam(1-omgp)      # numeric overflow
gam_ur(omgp)     # large but finite
```

`gyr` *Gyr function*

Description

Relativistic addition of three velocities

Usage

```
gyr(u, v, x)
gyr.a(u, v, x)
gyrfun(u, v)
```

Arguments

u, v, x	Three-velocities, objects of class `3vel`

Details

Function `gyr(u,v,x)` returns the three-vector $\mathrm{gyr}[u,v]x$.

Function `gyrfun(u,v)` returns a function that returns a three-vector; see examples.

The speed of light (1 by default) is not used directly by these functions; set it with `sol()`.

Note

Function `gyr()` is slightly faster than `gyr.a()`, which is included for pedagogical reasons.

Function `gyr()` is simply

`add3(neg3(add3(u,v)),add3(u,add3(v,x)))`

while function `gyr.a()` uses the slower but more transparent idiom

`-(u+v) + (u+(v+x))`

Author(s)

Robin K. S. Hankin

References

- Ungar 2006. "Thomas precession: a kinematic effect of the algebra of Einstein's velocity addition law. Comments on 'Deriving relativistic momentum and energy: I. Three-dimensional case"'. European Journal of Physics, 27:L17-L20.
- Sbitneva 2001. "Nonassociative geometry of special relativity". International Journal of Theoretical Physics, volume 40, number 1, pages 359–362

Examples

```
u <- r3vel(10)
v <- r3vel(10)
w <- r3vel(10)

x <- as.3vel(c(0.4,0.1,-0.5))
y <- as.3vel(c(0.1,0.2,-0.7))
z <- as.3vel(c(0.2,0.3,-0.1))


gyr(u,v,x)  # gyr[u,v]x

f <- gyrfun(u,v)
g <- gyrfun(v,u)

f(x)
f(r3vel(10))

f(g(x)) - x              # zero, by eqn 9
g(f(x)) - x              # zero, by eqn 9
(x+y) - f(y+x)           # zero by eqn 10
(u+(v+w)) - ((u+v)+f(w)) # zero by eqn 11


# Following taken from Sbitneva 2001:

rbind(x+(y+(x+z))  ,   (x+(y+x))+z)   # left Bol property
rbind((x+y)+(x+y)  ,   x+(y+(y+x)))   # left Bruck property


sol(299792458)   # speed of light in SI
as.3vel(c(1000,3000,1000)) + as.3vel(c(1000,3000,1000))
## should be close to Galilean result

sol(1)   # revert to default c=1
```

Ops.3vel *Arithmetic Ops Group Methods for 3vel objects*

Description

Arithmetic operations for three-velocities

Usage

```
## S3 method for class '3vel'
Ops(e1, e2)
## S3 method for class '4vel'
Ops(e1, e2)
massage3(u,v)
neg3(u)
prod3(u,v=u)
add3(u,v)
dot3(v,r)
```

Arguments

e1, e2, u, v	Objects of class "3vel", three-velocities
r	Scalar value for circle-dot multiplication

Details

The function `Ops.3vel()` passes unary and binary arithmetic operators "+", "-" and "*" to the appropriate specialist function.

The most interesting operators are "+" and "*", which are passed to `add3()` and `dot3()` respectively. These are defined, following Ungar, as:

$$\mathbf{u}+\mathbf{v}=\frac{1}{1+\mathbf{u}\cdot\mathbf{b}/c^2}\left\{\mathbf{u}+\frac{1}{\gamma_\mathbf{u}}\mathbf{v}+\frac{1}{c^2}\frac{\gamma_\mathbf{u}}{1+\gamma_\mathbf{u}}\left(\mathbf{u}\cdot\mathbf{v}\right)\mathbf{u}\right\}$$

and

$$r\odot\mathbf{v}=c\tanh\left(r\tanh^{-1}\frac{||\mathbf{v}||}{c}\right)\frac{\mathbf{v}}{||\mathbf{v}||}$$

where $\mathbf{u}$ and $\mathbf{v}$ are three-vectors and r a scalar. Function `dot3()` has special dispensation for zero velocity and does not treat `NA` entries entirely consistently.

Arithmetic operations, executed via `Ops.4vel()`, are not defined on four-velocities.

The package is designed so that natural R idiom may be used for three velocity addition, see the examples section.

Value

Returns an object of class `3vel`, except for `prod3()` which returns a numeric vector.

Examples

```
u <- as.3vel(c(-0.7, 0.1,-0.1))
v <- as.3vel(c( 0.1, 0.2, 0.3))
w <- as.3vel(c( 0.5, 0.2,-0.3))

x <- r3vel(10)   # random three velocities
y <- r3vel(10)   # random three velocities


u+v   # add3(u,v)
u-v   # add3(u,neg3(v))

-v    # neg3(v)

gyr(u,v,w)

## package is vectorized:


u+x
x+y

f <- gyrfun(u,v)
g <- gyrfun(v,u)

f(g(x)) - x    # should be zero by eqn10
g(f(x)) - x


(u+v) - f(v+u)                     # zero by eqn 10
(u+(v+w)) - ((u+v)+f(w))           # zero by eqn 11
((u+v)+w) - (u+(v+g(w)))           # zero by eqn 11


## NB, R idiom is unambiguous.  But always always ALWAYS use brackets.

## Ice report in lat 42.n to 41.25n Long 49w to long 50.30w saw much
## heavy pack ice and great number large icebergs also field
## ice.  Weather good clear

## -u+v == (-u) + v == neg3(u) + v == add3(neg3(u),v)

## u+v+w == (u+v)+w == add3(add3(u,v),w)
```

---

photon *Photons*

Description

Various functionality to deal with the 4-momentum of a photon

Usage

```
is.consistent.nullvec(N,TOL=1e-10)
as.photon(x,E=1)
```

Arguments

N	Four-momentum to be tested for nullness
TOL	tolerance
x	Vector of three-velocities
E	Energy, a scalar

Details

Returns the four-momentum of a photon.

Author(s)

Robin K. S. Hankin

See Also

4mom,reflect

Examples

```
## A bunch of photons all approximately parallel to the x-axis:
as.photon(as.3vel(cbind(0.9,runif(10)/1000,runif(10)/1000)))


## mirror ball:
jj <- matrix(rnorm(30),10,3)
disco <- sweep(matrix(rnorm(30),10,3),1,sqrt(rowSums(jj^2)),`/`)
p <- as.photon(c(1,0,0))
reflect(p,disco)

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

## relativistic  disco; mirror ball moves at 0.5c:

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

print.3vel *Print methods for three-velocities and four-velocities*

Description

Print methods for three-velocities

Usage

```
## S3 method for class '3vel'
print(x, ...)
## S3 method for class '3cel'
print(x, ...)
## S3 method for class '4vel'
print(x, ...)
## S3 method for class '4mom'
print(x, ...)
```

Arguments

x	Vector of three-velocities
...	Further arguments, currently ignored

Value

Returns a vector of three-velocities

Author(s)

Robin K. S. Hankin

Examples

```
r3vel(10)
```

r3vel *Random relativistic velocities*

Description

Generates random three-velocities or four-velocities, optionally specifiying a magnitude

Usage

```
r3vel(n=7, r = NA)
r4vel(...)
rboost(r = NA)
```

Arguments

n	Number of three- or four- velocities to generate
r	Absolute value of the three-velocities, with default NA meaning to sample uniformly from the unit ball
...	Arguments passed to r3vel()

Details

Function r3vel() returns a random three-velocity. Function r4vel() is a convenience wrapper for as.4vel(r3vel()).

Function rboost() returns a random 4×4 Lorentz boost matrix, drawn from the connected component. If given r=0, then a transform corresponding to a random rotation will be returned.

Value

Returns a vector of three- or four- velocities.

Note

If the speed of light is infinite, these functions require a specified argument for r.

It is not entirely trivial to sample *uniformly* from the unit ball or unit sphere, but it is not hard either.

Author(s)

Robin K. S. Hankin

Examples

```
r3vel()

a <- r3vel(10000)
b <- r3vel(1000,0.8)
u <- as.3vel(c(0,0,0.9))

pairs(unclass(u+a),asp=1)
pairs(unclass(a+u),asp=1)

is.consistent.boost(rboost())

sol(299792458)    # switch to SI units
sound <- 343      # speed of sound in metres per second
r3vel(100,343)    # random 3-velocities with speed = 343 m/s

sol(1)   # return to default c=1
```

`reflect` *Mirrors*

Description

Plane mirrors in special relativity

Usage

```
reflect(P,m,ref=1)
```

Arguments

`P`	Vector of four-momenta
`m`	Orientation of mirror, expressed as a three-vector
`ref`	Coefficient of reflectivity of the mirror

Value

Takes a four-momentum and returns the four-momentum after reflection. Will handle objects or photons.

Note

All four-momenta are measured in the rest frame of the mirror, but it is easy to reflect from moving mirrors; see examples.

However, note that the `ref` argument is designed to work with photons only, where it is conceptually the percentage of photons reflected and not absorbed by the mirror. If `ref` is less than unity, odd results are given for four momenta of nonzero restmass objects.

Author(s)

Robin K. S. Hankin

See Also

`photon`

Examples

```
## We will reflect some photons from an oblique mirror moving at half
## the speed of light.

## First create 'A', a bunch of photons all moving roughly along the x-axis:
A <- as.photon(as.3vel(cbind(0.9,runif(10)/1000,runif(10)/1000)))

## Now create 'm', a mirror oriented perpendicular to c(1,1,1):
m <- c(1,1,1)
```

```
## Reflect the photons in the mirror:
reflect(A,m)

## Reflect the photons in a series of mirrors:
A |> reflect(m) |> reflect(1:3) |> reflect(3:1)


## To reflect from a moving mirror we need to transform to a frame in
## which the mirror is at rest, then transform back to the original
## frame.  First create B, a boost representing the mirror's movement
## along the x-axis at speed c/2:

B <- boost(as.3vel(c(0.5,0,0)))


## Transform to the mirror's rest frame:
A %*% t(B)

## NB: in the above, take a transpose because the *rows* of A are 4-vectors.

## Then reflect the photons in the mirror:
reflect(A %*% t(B),m)


## Now transform back to the original rest frame (NB: active transform):
A |> tcrossprod(B) |> reflect(m) |> tcrossprod(solve(B))
```

`seq.3vel` *seq method for three velocities*

Description

Simplified version of `seq()` for three-velocities.

Usage

```
## S3 method for class '3vel'
seq(from, to, len, ...)
```

Arguments

`from, to`	Start and end of sequence
`len`	Length of vector returned
`...`	Further arguments (currently ignored)

Details

`seq(a,b,n)` returns `a + t*(-b+a)` where `t` is numeric vector `seq(from=0,to=1,len=n)`.

This definition is one of several plausible alternatives, but has the nice property that the first and last elements are exactly equal to `a` and `b` respectively.

Author(s)

Robin K. S. Hankin

Examples

```
a <- as.3vel(c(4,5,6)/9)
b <- as.3vel(c(-5,6,8)/14)
x <- seq(a,b,len=9)

x[1]-a # should be zero
x[9]-b # should be zero


jj <- a + seq(0,1,len=9)*(b-a)

jj-x   # decidedly non-zero
```

`sol` *Speed of light and Minkowski metric*

Description

Getting and setting the speed of light

Usage

```
sol(c)
eta(downstairs=TRUE)
ptm(to_natural=TRUE, change_time=TRUE)
```

Arguments

`c`	Scalar, speed of light. If missing, return the speed of light
`downstairs`	Boolean, with default `TRUE` meaning to return the covariant metric tensor g_{ij} with two downstairs indices, and `FALSE` meaning to return the contravariant version g^{ij} with two upstairs indices
`to_natural, change_time`	Boolean, specifying the nature of the passive transform matrix

Details

In the context of an R package, the symbol "c" presents particular problems. In the **lorentz** package, the speed of light is denoted "sol", for 'speed of light'. You can set the speed of light with `sol(x)`, and query it with `sol()`; see the examples. An infinite speed of light is sometimes useful for Galilean transforms.

The speed of light is a global variable, governed by `options("c")`. If `NULL`, define `c=1`. Setting `showSOL` to `TRUE` makes `sol()` change the prompt to display the speed of light which might be useful.

Function `eta()` returns the Minkowski flat-space metric

$$\mathrm{diag}\left(-c^2, 1, 1, 1\right).$$

Note that the top-left element of `eta()` is $-c^2$, not -1.

Function `ptm()` returns a passive transformation matrix that converts displacement vectors to natural units (`to_natural=TRUE`) or from natural units (`to_natural=FALSE`). Argument `change_time` specifies whether to change the unit of time (if `TRUE`) or the unit of length (if `FALSE`).

Note

Typing "`sol(299792458)`" is a lot easier than typing "`options("c"=299792458)`", which is why the package uses the idiom that it does.

In a R-devel discussion about options for printing, Martin Maechler makes the following observation: "Good programming style for functions according to my book is to have them depend only on their arguments, and if a global option really (really? think twice!) should influence behavior, there should be arguments of the function which have a default determined by the global option"

I think he is right in general, but offer the observation that the speed of light depends on the units chosen, and typically one fixes one's units once and for all, and does not subsequently change them. This would indicate (to me at least) that a global option would be appropriate. Further, there *is* a default, $c = 1$, which is returned by `sol()` if the option is unset. This is not just a "default", though: it is used in the overwhelming majority of cases. Indeed, pedagogically speaking, one learning objective from the package is that units in which $c \neq 1$ are difficult, awkward, and unnatural. In the package R code, the *only* place the speed of light option is accessed is via `sol()`. Similar arguments are presented in the **clifford** package at `signature.Rd`.

Looking again at Martin's observation he seems to be suggesting that something along the lines of

```
gam <- function(u, c=1){1/sqrt(1-u^2/c^2)}
```

But this is asking for trouble:

```
c(gam(0.4,c=1),gam(0.4,c=10))
```

which is meaningless at best and misleading at worst.

Author(s)

Robin K. S. Hankin

Examples

```
sol()                          # returns current speed of light
sol(299792458)                 # use SI units
sol()                          # speed of light now SI value

eta()                          # note [t,t] term
u <- as.3vel(c(100,200,300))   # fast terrestrial speed, but not relativistic
boost(u)                       # boost matrix practically Galilean
is.consistent.boost(boost(u))  # should be TRUE
sol(1)                         # revert to relativistic units
```

---

threevel *Three velocities*

Description

Create and test for three-velocities, `3vel` objects.

Usage

```
`3vel`(n)
threevel(n)
as.3vel(x)
is.3vel(x)
## S3 method for class 'vec'
length(x)
## S3 method for class 'vec'
names(x)
## S3 replacement method for class 'vec'
names(x) <- value
```

Arguments

`n`	In function `3vel()`, number of three velocities to create
`x, value`	Vectors of three-velocities

Note

Class `vel` is a virtual class containing classes `3vel` and `4vel`.

Function `threevel()` is a convenience wrapper for `3vel()`.

Author(s)

Robin K. S. Hankin

Examples

```
U <- r4vel(7)
as.4vel(as.3vel(U)) # equal to U, to numerical precision

x <- as.3vel(1:3/4)
u <- as.3vel(matrix(runif(30)/10,ncol=3))

names(u) <- letters[1:10]

x+u
u+x  # not equal
```

---

transform — *The energy-momentum tensor*

Description

Various functionality to deal with the stress-energy tensor in special relativity.

Usage

```
perfectfluid(rho,p,u=0)
dust(rho,u=0)
photongas(rho,u=0)
transform_dd(TT, B)
transform_ud(TT, B)
transform_uu(TT, B)
raise(TT)
lower(TT)
```

Arguments

`TT`	A second-rank tensor with indices either downstairs-downstairs, downstairs-upstairs, or upstairs-upstairs
`B`	A boost matrix
`rho, p, u`	Density, pressure, and four-velocity of the dust

Details

Function `perfectfluid()` returns the stress-energy tensor, with two upstairs indices, for a perfect fluid with the conditions specified. No checking for physical reasonableness (eg the weak energy condition) is performed: caveat emptor!

Function `dust()` is a (trivial) function that returns the stress-energy tensor of a zero-pressure perfect fluid (that is, dust). Function `photongas()` returns the stress-energy tensor of a photon gas. They are here for discoverability reasons; both are special cases of a perfect fluid.

Functions `transform_dd()` et seq transform a second-rank tensor using the Lorentz transform. The letters "u" or "d" denote the indices of the tensor being upstairs (contravariant) or downstairs (covariant). The stress-energy tensor is usually written with two upstairs indices, so use `transform_uu()` to transform it.

Function `lower()` lowers both indices of a tensor with two upstairs indices. Function `raise()` raises two downstairs indices. These two functions have identical R idiom but do not return identical values if $c \neq 1$.

Author(s)

Robin K. S. Hankin

Examples

```
perfectfluid(10,1)

u <- as.3vel(c(0.4,0.4,0.2))

## In the following, LHS is stationary dust and RHS is dust moving at
## velocity 'u', but transformed to a frame also moving at velocity 'u':

LHS <- dust(1)
RHS <- transform_uu(dust(1,u),boost(u))
max(abs(LHS-RHS))  # should be small


## In the following, negative sign needed because active/passive
## difference:

LHS <- dust(1,u)
RHS <- transform_uu(dust(1),boost(-u))
max(abs(LHS-RHS))  # should be small

## Now test behaviour when  c!=1:


sol(299792458)
perfectfluid(1.225,101325) # air at STP

LHS <- transform_uu(perfectfluid(1.225,101325),boost(as.3vel(c(1000,0,0))))
RHS <- perfectfluid(1.225,101325)
LHS-RHS  # should be small
```

```
sol(10)
u <- as.3vel(4:6)
LHS <- photongas(1,u)
RHS <- transform_uu(photongas(1),boost(-u))
LHS-RHS # should be small



B1 <- boost(r3vel(1)) %*% boost(r3vel(1))
B2 <- boost(r3vel(1)) %*% boost(r3vel(1))
LHS <- transform_uu(transform_uu(dust(1),B1),B2)
RHS <- transform_uu(dust(1),B2 %*% B1)   # note order
LHS-RHS  # should be small


## remember to re-set c:
sol(1)
```

Index