Package: insitu (via r-universe)

December 24, 2024

Type Package

Title By-Reference Routines for Numeric Vectors

Version 0.1.3.9019

Maintainer Mike Cheng <mikefc@coolbutuseless.com>

URL https://github.com/coolbutuseless/insitu

BugReports https://github.com/coolbutuseless/insitu/issues

Description Using by-reference semantics, functions in this package are crafted to modify the input objects in-place and avoid allocating new memory. By avoiding memory allocations (and the associated garbage colection these require), operations performed by-reference can be faster than those performed with R's default copy-on-modify semantics.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, rlang, bench, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://coolbutuseless.r-universe.dev

RemoteUrl https://github.com/coolbutuseless/insitu

RemoteRef HEAD

RemoteSha c978c15cc170dc68d98dd5bd6fa3df9d9d72eab3

Contents

`alloc_matrix` *Allocate empty matrix of the given dimensions.*

Description

The values are in the matrix are *not* initialised to zero.

Usage

```
alloc_matrix(nrow, ncol)
```

Arguments

nrow, ncol	dimensions

Value

Matrix of the requested dimensions. Values are *not* initialised.

Examples

```
m <- alloc_matrix(nrow = 3, ncol = 2)
is.matrix(m)
dim(m)
```

---

alloc_mat_mat_mul *Allocate empty matrix for the result of the matrix multiplication* A * B

Description

Allocate empty matrix for the result of the matrix multiplication A * B

Usage

```
alloc_mat_mat_mul(A, B, ta = FALSE, tb = FALSE)
```

Arguments

A, B	Primary multiplication matrices
ta	Should matrix 'A' be transposed? Default: FALSE
tb	Should matrix 'B' be transposed? Default: FALSE

Value

Uninitialized matrix of the correct size to hold the result

Examples

```
A <- matrix(1, 2, 4)
B <- matrix(1, 4, 7)
C <- alloc_mat_mat_mul(A, B)
is.matrix(C)
dim(C)
```

alloc_mat_vec_mul *Allocate empty vector for the result of the matrix-vector multiplication* `A * x`

Description

Allocate empty vector for the result of the matrix-vector multiplication `A * x`

Usage

```
alloc_mat_vec_mul(A, x)
```

Arguments

`A`	Matrix
`x`	vector

Value

Uninitialized vector of the correct size to hold the result of `A * x`

Examples

```
A <- matrix(1, 2, 4)
x <- matrix(1, 4, 1)
y <- alloc_mat_vec_mul(A, x)
dim(y)
br_mat_vec_mul(y, A, x)
y
# Compare to base R
A %*% x
```

alloc_n *Allocate a new numeric vector*

Description

These functions are like `numeric()` except the contents of the vector are not initialized.

Usage

```
alloc_n(n)

alloc_along(x)
```

Arguments

n	length of new numeric vector
x	vector used as template. New vector will be the same length as this vector

Details

`alloc_n` Allocates a numeric vector which can hold 'n' values

`alloc_along` Allocates a numeric vector which can hold the same number of values as the given vector

Value

new numeric vector of the required length. Note: This vector is *not* initialized to zero.

Examples

```
x <- alloc_n(10)
br_seq(x)
x
```

---

br_abs *Math operations taking only a single input vector*

Description

Math operations taking only a single input vector

Usage

```
br_abs(x, idx = NULL, where = NULL, cols = NULL)

br_sqrt(x, idx = NULL, where = NULL, cols = NULL)

br_floor(x, idx = NULL, where = NULL, cols = NULL)

br_ceil(x, idx = NULL, where = NULL, cols = NULL)

br_trunc(x, idx = NULL, where = NULL, cols = NULL)

br_exp(x, idx = NULL, where = NULL, cols = NULL)

br_log(x, idx = NULL, where = NULL, cols = NULL)

br_log2(x, idx = NULL, where = NULL, cols = NULL)

br_log10(x, idx = NULL, where = NULL, cols = NULL)
```

```
br_cos(x, idx = NULL, where = NULL, cols = NULL)

br_sin(x, idx = NULL, where = NULL, cols = NULL)

br_tan(x, idx = NULL, where = NULL, cols = NULL)

br_not(x, idx = NULL, where = NULL, cols = NULL)

br_expm1(x, idx = NULL, where = NULL, cols = NULL)

br_log1p(x, idx = NULL, where = NULL, cols = NULL)

br_acos(x, idx = NULL, where = NULL, cols = NULL)

br_asin(x, idx = NULL, where = NULL, cols = NULL)

br_atan(x, idx = NULL, where = NULL, cols = NULL)

br_acosh(x, idx = NULL, where = NULL, cols = NULL)

br_asinh(x, idx = NULL, where = NULL, cols = NULL)

br_atanh(x, idx = NULL, where = NULL, cols = NULL)

br_cosh(x, idx = NULL, where = NULL, cols = NULL)

br_sinh(x, idx = NULL, where = NULL, cols = NULL)

br_tanh(x, idx = NULL, where = NULL, cols = NULL)

br_cospi(x, idx = NULL, where = NULL, cols = NULL)

br_sinpi(x, idx = NULL, where = NULL, cols = NULL)

br_tanpi(x, idx = NULL, where = NULL, cols = NULL)

br_sign(x, idx = NULL, where = NULL, cols = NULL)

br_is_na(x, idx = NULL, where = NULL, cols = NULL)

br_zero(x, idx = NULL, where = NULL, cols = NULL)

br_pow2(x, idx = NULL, where = NULL, cols = NULL)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation

idx	vector of indices
where	logical vector stored as floating point values. 0 = FALSE, all non-zero values treated as TRUE. Default: NULL. This value indicates if the operation should be performed for the corresponding element in x.
cols	which matrix columns should this apply to? Argument only valid if x is a matrix

Value

x argument is modified by-reference and returned invisibly

Examples

```
# By-reference \code{abs()}
x <- c(-1, 0, 1)
br_abs(x)
x

# Conditional application of sqrt()
x <- as.numeric(-5:5)
t <- duplicate(x)     # will use this to hold a numeric logical vector
br_gt(t, 0)           # where is the value > 0
br_sqrt(x, where = t) # only sqrt(x) where x > 0
x
```

---

br_add *Math operations taking two arguments*

Description

Math operations taking two arguments

Usage

```
br_add(x, y, idx = NULL, where = NULL, cols = NULL)

br_sub(x, y, idx = NULL, where = NULL, cols = NULL)

br_mul(x, y, idx = NULL, where = NULL, cols = NULL)

br_div(x, y, idx = NULL, where = NULL, cols = NULL)

br_pow(x, y, idx = NULL, where = NULL, cols = NULL)

br_eq(x, y, idx = NULL, where = NULL, cols = NULL)

br_ne(x, y, idx = NULL, where = NULL, cols = NULL)

br_lt(x, y, idx = NULL, where = NULL, cols = NULL)
```

```
br_le(x, y, idx = NULL, where = NULL, cols = NULL)

br_gt(x, y, idx = NULL, where = NULL, cols = NULL)

br_ge(x, y, idx = NULL, where = NULL, cols = NULL)

br_and(x, y, idx = NULL, where = NULL, cols = NULL)

br_or(x, y, idx = NULL, where = NULL, cols = NULL)

br_rem(x, y, idx = NULL, where = NULL, cols = NULL)

br_idiv(x, y, idx = NULL, where = NULL, cols = NULL)

br_max(x, y, idx = NULL, where = NULL, cols = NULL)

br_min(x, y, idx = NULL, where = NULL, cols = NULL)

br_assign(x, y, idx = NULL, where = NULL, cols = NULL)

br_sumsq(x, y, idx = NULL, where = NULL, cols = NULL)

br_diffsq(x, y, idx = NULL, where = NULL, cols = NULL)

br_mod(x, y, idx = NULL, where = NULL, cols = NULL)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
y	Either scalar numeric value or numeric vector of the same length as x.
idx	idx
where	logical vector stored as floating point values. 0 = FALSE, all non-zero values treated as TRUE. Default: NULL. This value indicates if the operation should be performed for the corresponding element in x.
cols	columns

Value

x argument is modified by-reference and returned invisibly

Examples

```
# x <- x + y
x <- as.numeric(1:10)
y <- as.numeric(1:10)
br_add(x, y)
```

```
x
# x <- x + 1
br_add(x, 1)
x
```

---

br_copy *Copy all or part of one vector into another*

Description

Copy all or part of one vector into another

Usage

```
br_copy(x, y, n = NULL, xi = 1L, yi = 1L)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
y	Either scalar numeric value or numeric vector of the same length as x.
n	number of elements to copy. Default: NULL. This default requires x and y to be the same length, and simply copies the full vector y into x.
xi, yi	starting indices for the copy operation within the two vectors. Default: 1 i.e. the first element

Value

x argument is modified by-reference and returned invisibly

Examples

```
# x, y are the same length. copy all of 'y' into 'x'
x <- as.numeric(1:10)
y <- as.numeric(10:1)
br_copy(x, y)
x

# copy the last 3 elements of 'y' into the first 3 elements of x
x <- as.numeric(1:10)
y <- as.numeric(10:1)
br_copy(x, y, n = 3, xi = 1, yi = 8)
x

# copy a scalar value into the last 3 elements of x
x <- as.numeric(1:10)
br_copy(x, y = 99, n = 3, xi = -3)
x
```

br_cumsum *Math operations taking only a single input vector*

Description

Math operations taking only a single input vector

Usage

```
br_cumsum(x)

br_cumprod(x)

br_cummax(x)

br_cummin(x)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation

Value

x argument is modified by-reference and returned invisibly

Examples

```
# By-reference \code{abs()}
x <- c(-1, 0, 1)
br_abs(x)
x

# Conditional application of sqrt()
x <- as.numeric(-5:5)
t <- duplicate(x)     # will use this to hold a numeric logical vector
br_gt(t, 0)           # where is the value > 0
br_sqrt(x, where = t) # only sqrt(x) where x > 0
x
```

br_fmadd *Fused multiply add*

Description

Performs a compound calculation equivalent to `x <- x * a + b` using the best precision from the math library.

Usage

```
br_fmadd(x, a, b)

br_fmsub(x, a, b)

br_fnmadd(x, a, b)

br_fnmsub(x, a, b)
```

Arguments

`x`	numeric vector. This vector will be modified by-reference to contain the result of the calculation
`a, b`	Either scalar numeric values or numeric vectors of the same length as `x`.

Details

`br_fmadd()` Equivalent to: `x <- x * a + b`

`br_fmsub()` Equivalent to: `x <- x * a - b`

`br_fnmadd()` Equivalent to: `x <- -x * a + b`

`br_fnmsub()` Equivalent to: `x <- -x * a - b`

Value

`x` argument is modified by-reference and returned invisibly

Examples

```
x <- c(1, 2, 3)
a <- c(2, 3, 4)
b <- c(10, 10, 10)
br_fmadd(x, a, b)
x
```

br_mat_col_get *Get/set a column or row of a matrix*

Description

Get/set a column or row of a matrix

Usage

```
br_mat_col_get(mat, i, vec)

br_mat_col_set(mat, i, vec)

br_mat_row_get(mat, i, vec)

br_mat_row_set(mat, i, vec)
```

Arguments

mat	matrix
i	index of row or column
vec	vector space. For *get* operations, vector length must match the row or column length of the matirx. When doing a `set` operation, `vec` may also be a single value (which will be repeated to fill the row/column)

Value

None. For `get` operations, `vec` is modified by-reference and returned invisibly. For `set` operations, `mat` is modified by reference and returned invisibly.

Examples

```
m <- matrix(as.numeric(1:6), 2, 3)
m
v <- alloc_n(nrow(m))
br_mat_col_get(m, 2, v)
v
```

br_mat_dist2 *Distance calculation*

Description

Distance calculation

Usage

```
br_mat_dist2(d, mat1, mat2)

br_mat_dist3(d, mat1, mat2)
```

Arguments

d	pre-allocated vector to hold the result
mat1, mat2	matrices holding (x, y) or (x, y, z) coordinates

Value

None. Argument d modified in-place and returned invisibly

Examples

```
N <- 10
x1 <- rep(1, N)
y1 <- runif(N)
z1 <- runif(N)
x2 <- runif(N)
y2 <- runif(N)
z2 <- runif(N)
mat1 <- cbind(x1, y1, z1)
mat2 <- cbind(x2, y2, z2)
d <- alloc_n(N)
br_mat_dist2(d, mat1, mat2)
d
```

br_mat_hypot2 *Hypotenuse length calculation (distance from origin)*

Description

Hypotenuse length calculation (distance from origin)

Usage

```
br_mat_hypot2(d, mat)

br_mat_hypot3(d, mat)
```

Arguments

d	pre-allocated numeric vector with length matching nrow(mat)
mat	numeric matrix

Value

None. Argument d modified in-place and returned invisibly.

Examples

```
# Distance from origin in 3D
N <- 10
x <- rep(1, N)
y <- runif(N)
z <- runif(N)
mat <- cbind(x, y, z)
d1 <- alloc_n(N)
br_mat_hypot3(d1, mat)
d1

# Compare to base R
d2 <- sqrt(x * x + y * y + z * z)
all.equal(d1, d2)
```

---

br_mat_mat_mul *Matrix-matrix multiplication*

Description

This function exposes the general matrix multiplication operation from R's included BLAS. C = alpha * A * B + beta * C

Usage

```
br_mat_mat_mul(C, A, B, alpha = 1, beta = 0, ta = FALSE, tb = FALSE)
```

Arguments

C	Matrix which will be modified by-reference
A, B	Primary multiplication matrices
alpha	Scaling factor for A * B. Default: 1.0

beta	Scaling factor for C. Default: 0.0
ta	Should matrix 'A' be transposed? Default: FALSE
tb	Should matrix 'B' be transposed? Default: FALSE

Value

C is modified by-reference and returned invisibly

Examples

```
A <- matrix(as.numeric(1:32), 8, 4)
B <- matrix(as.numeric(1:24), 4, 6)
C <- alloc_mat_mat_mul(A, B)

br_mat_mat_mul(C, A, B) # By reference. Overwriting 'C'
C

A %*% B  # Compare to base R
```

---

br_mat_mat_mul_bsq *Matrix-matrix multiplication when B is a square matrix*

Description

This function exposes the general matrix multiplication operation from R's included BLAS. A = alpha * A * B

Usage

```
br_mat_mat_mul_bsq(A, B, alpha = 1, tb = FALSE)
```

Arguments

A, B	Primary multiplication matrices
alpha	Scaling factor for A * B. Default: 1.0
tb	Should matrix 'B' be transposed? Default: FALSE

Value

C is modified by-reference and returned invisibly

Examples

```
A <- matrix(as.numeric(1:32), 8, 4)
B <- matrix(as.numeric(1:16), 4, 4)
A %*% B  # Compare to base R

br_mat_mat_mul_bsq(A, B) # By reference. Overwriting 'A'
A
```

br_mat_normalise2 *Normalise matrix of (x,y) coordinates to length 1*

Description

These functions will normalise matrices of (x, y) or (x, y, z) coordinates so that the represented vector quantity have a length of 1.

Usage

```
br_mat_normalise2(mat)

br_mat_normalise3(mat)
```

Arguments

mat	numeric matrix

Value

Input matrix modified in-place and returned invisibly.

Examples

```
N <- 10
x <- runif(N)
y <- runif(N)
mat <- cbind(x, y)
br_mat_normalise2(mat)
# Lengths should now be 1.
sqrt(mat[,1]^2 + mat[,2]^2)
```

br_mat_roll *Roll elements of a matrix*

Description

Roll elements of a matrix

Usage

```
br_mat_roll(mat, rows = 0, cols = 0)
```

Arguments

mat	matrix
rows, cols	Number of rows and columns to roll

Value

None. Matrix is modified by-reference and returned invisibly

Examples

```
m <- matrix(as.numeric(1:6), 2, 3)
m
br_mat_roll(m, rows = 3)
m
```

`br_mat_transpose` *Transpose matrix*

Description

Transpose matrix

Usage

```
br_mat_transpose(mat)
```

Arguments

`mat`	matrix

Value

None. Matrix is modified by-reference and returned invisibly

Examples

```
m <- matrix(as.numeric(1:6), 2, 3)
m
br_mat_transpose(m)
m
```

br_mat_vec_mul *Matrix-vector multiplication*

Description

This function exposes the general matrix-vector multiplication operation from R's included BLAS. y = alpha * A * x + beta * y

Usage

```
br_mat_vec_mul(y, A, x, alpha = 1, beta = 0, ta = FALSE)
```

Arguments

y	Vector which will be modified by-reference
A	Matrix
x	vector
alpha	Scaling factor for A * x. Default: 1.0
beta	Scaling factor for y. Default: 0.0
ta	Should matrix 'A' be transposed? Default: FALSE

Value

y is modified by-reference and returned invisibly

Examples

```
A <- matrix(1, 3, 5)
x <- rep(1, 5)
y <- rep(0, 3)

# Calculate. By-reference. Overwriting 'y'
br_mat_vec_mul(y, A, x)
y

# Compare to R's method
A %*% x
```

br_mat_vec_mul_asq *Matrix-vector multiplication when A is square*

Description

This function exposes the general matrix-vector multiplication operation from R's included BLAS. x = alpha * A * x

Usage

```
br_mat_vec_mul_asq(A, x, alpha = 1, ta = FALSE)
```

Arguments

A	Matrix
x	vector
alpha	Scaling factor for A * x. Default: 1.0
ta	Should matrix 'A' be transposed? Default: FALSE

Value

None. x is modified by-reference and returned invisibly

Examples

```
A <- matrix(1, 3, 3)
x <- rep(1, 3)

# Compare to R's method
A %*% x

# Calculate. By-reference. Overwriting 'x'
br_mat_vec_mul_asq(A, x)
x
```

br_rev *Reverse a vector in place*

Description

Reverse a vector in place

Usage

```
br_rev(x)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation

Value

x argument is modified by-reference and returned invisibly

Examples

```
x <- as.numeric(1:10)
br_rev(x)
x
```

---

br_roll *Roll a vector*

Description

Roll a vector

Usage

```
br_roll(x, dist)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
dist	Distance to roll. Can be postive or negative

Value

None. x modified in-place and returned invisibly

Examples

```
x <- c(1, 2, 3, 4, 5)
br_roll(x, 2)
x
br_roll(x, -2)
x
br_roll(x, -3)
```

br_round *Round*

Description

Round

Usage

```
br_round(x, digits)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
digits	number of decimal places

Value

x modified by reference and returned invisibly

Examples

```
x <- c(1.234, 5.6789)
br_round(x, 2)
x
```

br_runif *Fill a vector with uniform random values*

Description

Random numbers are generated using a Lehmer RNG (also known as Park-Miller RNG). Note: The standard R RNG is only used to seed the Lehmer RNG - which means that `set.seed()` can be used for generating repeatable streams.

Usage

```
br_runif(x, lower = 0, upper = 1)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
lower, upper	limits of random numbers generated. Default: 0, 1

Value

x argument is modified by-reference and returned invisibly

Examples

```
set_seed_lehmer(1)
x <- alloc_n(10)
br_runif(x)
x
```

---

br_seq *Fill vector with a numeric sequence*

Description

Fill vector with a numeric sequence

Usage

```
br_seq(x, from = 1, to = NULL, step = NULL)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
from	starting value in sequence. Default: 1
to	ending value in sequence. Default: NULL. If NULL, then to = length(x) + from - 1
step	step size. This value is ignored if to is set.

Value

x argument is modified by-reference and returned invisibly

Examples

```
# fill 1:10
x <- alloc_n(10)
br_seq(x)
x


# fill from 1 to 100
x <- alloc_n(10)
br_seq(x, from = 1, to = 100)
x
```

```
# fill from 0 in steps of 2.5
x <- alloc_n(10)
br_seq(x, from = 0, step = 0.25)
x
```

`br_shuffle` *Shuffle a vector in place*

Description

Uses fisher-yates.

Usage

```
br_shuffle(x)
```

Arguments

`x`	numeric vector. This vector will be modified by-reference to contain the result of the calculation

Details

The *fast* variant of this function uses a xoshiro256++ PRNG which is much faster than sourcing randomness from R's random number generator.

Value

`x` argument is modified by-reference and returned invisibly

Examples

```
set.seed(1)
set_seed_lehmer()
x <- as.numeric(1:10)
br_shuffle(x)
x
```

br_sort *Sort a numeric vector by-reference*

Description

Sort a numeric vector by-reference

Usage

```
br_sort(x, decreasing = FALSE)
```

Arguments

x	numeric vector. This vector will be modified by-reference to contain the result of the calculation
decreasing	Logical. Sort values in decreasing order? default FALSE

Value

x argument is modified by-reference and returned invisibly

Examples

```
set.seed(1)
x <- sample(as.numeric(1:10))
br_sort(x)
x
```

by_reference *List of all functions*

Description

List of all functions

Usage

```
by_reference
```

Format

An object of class `list` of length 66.

Examples

```
x <- as.numeric(1:10)
with(by_reference, {x + 1; x * 2})
x
```

duplicate *Duplicate an R object*

Description

Duplicate an R object

Usage

```
duplicate(x)
```

Arguments

x	Any R object

Value

duplicate of the given object

Examples

```
x <- c(1, 2, 3)
y <- duplicate(x)
y
```

set_seed_lehmer *Initialise the state of this package's Lehmer RNG*

Description

Initialise the state of this package's Lehmer RNG

Usage

```
set_seed_lehmer(seed)
```

Arguments

seed	initeger seed value

Value

None.

Examples

```
set_seed_lehmer(1)
x <- alloc_n(10)
br_runif(x)
x
```

`tf2_add_rotate` *Add rotation to a transformation matrix*

Description

Add rotation to a transformation matrix

Usage

```
tf2_add_rotate(mat, theta)
```

Arguments

`mat`	3x3 transformation matrix
`theta`	rotation angle (radians)

Value

None. `mat` modified by reference and returned invisibly

Examples

```
tf <- tf2_new()
tf2_add_rotate(tf, pi/6)
tf
```

`tf2_add_scale` *Add scaling to a transformation matrix*

Description

Add scaling to a transformation matrix

Usage

```
tf2_add_scale(mat, x = 1, y = x)
```

Arguments

`mat`	3x3 transformation matrix
`x, y`	scaling

Value

None. `mat` modified by reference and returned invisibly

Examples

```
tf <- tf2_new()
tf2_add_scale(tf, 1, 2)
tf
```

`tf2_add_translate` *Add translation to a transformation matrix*

Description

Add translation to a transformation matrix

Usage

```
tf2_add_translate(mat, x = 0, y = 0)
```

Arguments

`mat`	3x3 transformation matrix
`x, y`	translation

Value

None. `mat` modified by reference and returned invisibly

Examples

```
tf <- tf2_new()
tf2_add_translate(tf, 1, 2)
tf
```

tf2_apply *Apply a 2-D affine transform to a matrix or data.frame of coordinates*

Description

Apply a 2-D affine transform to a matrix or data.frame of coordinates

Usage

```
tf2_apply(x, tf)
```

Arguments

x	matrix or data.frame. If a matrix, the first two columns must be x,y coordinates. If a data.frame, must contain columns 'x' and 'y'. Other columns will not be affected.
tf	3x3 transformation matrix

Value

None. `mat` is modified by reference and returned invisibly

Examples

```
# Transform a matrix
tf <- tf2_new()
tf <- tf2_add_translate(tf, 1, 10)
x <- cbind(
  x = as.numeric(1:6),
  y = as.numeric(1:6),
  other = 999
)
tf
x

tf2_apply(x, tf)
x

# Transform a data.frame
x <- data.frame(
  other = 'hello',
  x = as.numeric(1:6),
  y = as.numeric(1:6)
)
tf2_apply(x, tf)
x
```

tf2_new *Create identity transform for 2-D*

Description

Create identity transform for 2-D

Usage

```
tf2_new()
```

Value

3x3 identity matrix

Examples

```
tf2_new()
```

tf2_reset *Reset a 2-D transformation matrix back to the identity matrix*

Description

Reset a 2-D transformation matrix back to the identity matrix

Usage

```
tf2_reset(mat)
```

Arguments

mat	3x3 transformation matrix

Value

None. `mat` modified by reference and returned invisibly

Examples

```
tf <- matrix(1, 3, 3)
tf2_reset(tf)
tf
```

tf3_add_rotate_x *Add rotation to a transformation matrix*

Description

Add rotation to a transformation matrix

Usage

```
tf3_add_rotate_x(mat, theta)

tf3_add_rotate_y(mat, theta)

tf3_add_rotate_z(mat, theta)
```

Arguments

mat	4x4 transformation matrix
theta	rotation angle (radians)

Value

None. mat modified by reference and returned invisibly

Examples

```
tf <- tf3_new()
tf3_add_rotate_x(tf, pi/6)
tf
```

tf3_add_scale *Add scaling to a transformation matrix*

Description

Add scaling to a transformation matrix

Usage

```
tf3_add_scale(mat, x = 1, y = x, z = x)
```

Arguments

mat	4x4 transformation matrix
x, y, z	scaling

Value

None. mat modified by reference and returned invisibly

Examples

```
tf <- tf3_new()
tf3_add_scale(tf, 1, 2, 3)
tf
```

---

tf3_add_translate *Add translation to a transformation matrix*

Description

Add translation to a transformation matrix

Usage

```
tf3_add_translate(mat, x = 0, y = 0, z = 0)
```

Arguments

mat	4x4 transformation matrix
x, y, z	translation

Value

None. mat modified by reference and returned invisibly

Examples

```
tf <- tf3_new()
tf3_add_translate(tf, 1, 2, 3)
tf
```

tf3_apply *Apply a 3D affine transform to a matrix or data.frame of coordinates*

Description

Apply a 3D affine transform to a matrix or data.frame of coordinates

Usage

```
tf3_apply(x, tf)
```

Arguments

x	matrix or data.frame. If a matrix, the first two columns must be x,y coordinates. If a data.frame, must contain columns 'x' and 'y'. Other columns will not be affected.
tf	4x4 transformation matrix

Value

None. `mat` is modified by reference and returned invisibly

Examples

```
# Transform a matrix
tf <- tf3_new()
tf <- tf3_add_translate(tf, 1, 10, 100)
x <- cbind(
  x = as.numeric(1:6),
  y = as.numeric(1:6),
  z = as.numeric(1:6),
  other = 999
)
tf
x

tf3_apply(x, tf)
x

# Transform a data.frame
x <- data.frame(
  other = 'hello',
  x = as.numeric(1:6),
  y = as.numeric(1:6),
  z = as.numeric(1:6)
)
tf3_apply(x, tf)
x
```

tf3_new *Create identity transform*

Description

Create identity transform

Usage

```
tf3_new()
```

Value

4x4 identity matrix

Examples

```
tf3_new()
```

tf3_reset *Reset a transformation matrix back to the identity matrix*

Description

Reset a transformation matrix back to the identity matrix

Usage

```
tf3_reset(mat)
```

Arguments

mat	4x4 transformation matrix

Value

None. mat modified by reference and returned invisibly

Examples

```
tf <- matrix(1, 4, 4)
tf3_reset(tf)
tf
```

Index