Package ‘fort’

September 11, 2023

Type Package

Title Fast Orthogonal Random Transforms

Version 0.0.1

Description Provides convenient access to fast, structured, random linear transforms implemented in C++ (via 'Rcpp') that are (at least approximately) orthogonal or semi-orthogonal, and are often much faster than matrix multiplication. Useful for algorithms that require or benefit from uncorrelated random projections, such as fast dimensionality reduction (e.g., Johnson-Lindenstrauss transform) or kernel approximation (e.g., random kitchen sinks) methods.

License MIT + file LICENSE

Imports MASS, R6, Rcpp (>= 1.0.10)

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Author Tomé Silva [cre, aut] (<`https://orcid.org/0000-0002-9434-8686`>)

Maintainer Tomé Silva <`tome@tomesilva.com`>

URL `https://github.com/tomessilva/fort`, `https://tomessilva.github.io/fort/`

BugReports `https://github.com/tomessilva/fort/issues`

Suggests knitr,
rmarkdown,
testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

R topics documented:

`as.matrix.FastTransform` *Convert fast transform to matrix*

Description

Converts a fast transform created by `fort()` to the equivalent matrix form.

Usage

```
## S3 method for class 'FastTransform'
as.matrix(x, ...)
```

Arguments

`x`	An object of class `FastTransform`, created using `fort()`.
`...`	Extra parameters (ignored).

Value

A `matrix` object equivalent to x.

See Also

`fort()`

Examples

```
fast_transform <- fort(4, 15)
slow_transform <- as.matrix(fast_transform)
fast_result <- fast_transform %*% diag(4)
slow_result <- slow_transform %*% diag(4)
norm(fast_result - slow_result) # should be small
```

determinant.FastTransform *Calculate the Determinant of a Transform*

Description

det calculates the determinant of a FastTransform object. determinant returns separately the modulus of the determinant, optionally (by default) on the logarithm scale, and the sign of the determinant. If the input transform (x) is not square, the function will fail with an error.

Usage

```
## S3 method for class 'FastTransform'
determinant(x, logarithm = TRUE, ...)
```

Arguments

x	Object of FastTransform type with dim_in == dim_out.
logarithm	Logical. if TRUE (default) return the logarithm of the modulus of the determinant.
...	Extra parameters (ignored).

Value

For det, the determinant of x. For determinant, the same output format as determinant.matrix().

See Also

fort()

Examples

```
det(fort(16)) # either 1 or -1
determinant(fort(16))
```

dim.FastTransform *Dimensions of fast transform*

Description

Retrieves the dimensions of a fast transform created by fort() (i.e., the number of rows and columns of an equivalent matrix). It returns the same value that one would get from dim(as.matrix()), but much more efficiently.

Usage

```
## S3 method for class 'FastTransform'
dim(x)
```

Arguments

x An object of class FastTransform, created using fort().

Value

A vector of length 2 containing the dimensions of the fast transform (i.e., number of rows and number of columns, in this order).

See Also

fort()

Examples

```
dim(fort(3, 17)) # should return c(17,3)
dim(t(fort(3, 17))) # should return c(3,17)
```

FastTransform FastTransform *class*

Description

General specification of the type of objects generated by fort(), which correspond to structured linear transforms. Useful objects of this class must be also part of a subclass which extends this one with a specific implementation of a structured linear transform (e.g., FastTransformFFT1 or FastTransformFFT2).

Details

It is generally not recommended that the fields and methods described here are used directly, unless you have some specific reason (e.g., require low-level access to objects or want to use pipe operators). Instead, you should use fort() and the typical S3 methods for matrices, such as %*%.FastTransform and solve.FastTransform.

Public fields

inverse Logical. Indicates whether the object currently represents a forward or inverse transform.

invertible Logical. Indicates whether the inverse transform can also be expressed as a FastTransform object.

dim_in Dimensionality of the input for the forward transform.

dim_out Dimensionality of the output for the forward transform.

blocksize Dimensionality of the internal transformation (always a power of 2).

fwd_par List of parameters used in the forward transform.

fwd_mtrx Cached matrix representation of the forward transform.

rev_par List of parameters used in the inverse transform.

rev_mtrx Cached matrix representation of the inverse transform.

fort_type String indicating the type of structured transform being used.

cache_matrix Logical. Indicates whether to cache calculated matrices or not (default is TRUE).

logdet List with cached determinants of the forward and inverse transforms

Methods

Public methods:

- `FastTransform$fwd_eval()`
- `FastTransform$rev_eval()`
- `FastTransform$calculate_rev_par()`
- `FastTransform$new()`
- `FastTransform$evaluate()`
- `FastTransform$get_ncol()`
- `FastTransform$get_nrow()`
- `FastTransform$get_dim()`
- `FastTransform$get_n_par()`
- `FastTransform$get_inverse()`
- `FastTransform$get_transpose()`
- `FastTransform$get_logdet()`
- `FastTransform$get_norm()`
- `FastTransform$get_norm_margin()`
- `FastTransform$as_matrix()`
- `FastTransform$print()`
- `FastTransform$summary()`
- `FastTransform$clone()`

Method `fwd_eval()`: Function that performs the forward transform. Do *not* call this directly unless you know what you are doing: use the `FastTransform$evaluate()` method instead.

Usage:

```
FastTransform$fwd_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `rev_eval()`: Function that performs the inverse transform. Do *not* call this directly unless you know what you are doing: use the `FastTransform$evaluate()` method instead.

Usage:

```
FastTransform$rev_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `calculate_rev_par()`: Function that calculates and caches the parameters for the inverse transform. Do not call this directly unless you know what you are doing. If you need the inverse transform, use the `FastTransform$get_inverse()` method instead.

Usage:

```
FastTransform$calculate_rev_par()
```

Method `new()`: Raw object creation function. Note that calling this function does *not* result in a useful object. Instead, you should call the `fort()` function.

Usage:

```
FastTransform$new(dim_in, dim_out, blocksize)
```

Arguments:

`dim_in` Dimensionality of the input for the forward transform.

`dim_out` Dimensionality of the output for the forward transform.

`blocksize` Dimensionality of the internal transformation (*must* be a power of 2).

Returns: A matrix with the same number of columns as `x`.

Method `evaluate()`**:** Evaluates the result of applying the transform represented by this object on an input matrix `x`. It is important that the provided matrix has compatible dimensionality since *no input validation is performed*. This method is compatible with the use of pipe operators (e.g., `|>` or **magrittr**'s `%>%` and `%<>%` pipe operators).

Usage:

```
FastTransform$evaluate(x)
```

Arguments:

`x` Input matrix with correct dimensionality.

Returns: A matrix with the same number of columns as `x`.

Examples:

```
x <- fort(4) # random transform
y <- diag(4) # data to transform
x %*% y # y transformed by x
y |> x$evaluate() # same as previous line
```

Method `get_ncol()`**:** Returns the number of columns of the linear transform represented by this object.

Usage:

```
FastTransform$get_ncol()
```

Returns: A numeric value.

Method `get_nrow()`**:** Returns the number of rows of the linear transform represented by this object.

Usage:

```
FastTransform$get_nrow()
```

Returns: A numeric value.

Method `get_dim()`**:** Returns the dimensions of the linear transform represented by this object.

Usage:

```
FastTransform$get_dim()
```

Returns: A numeric vector with length 2.

Method `get_n_par()`**:** Returns the number of parameters required to represent the linear transform represented by this object.

Usage:

```
FastTransform$get_n_par()
```

Returns: A numeric value.

Method `get_inverse()`**:** Returns a new `FastTransform` object that represents the inverse transform of the transform represented by this object.

Usage:

```
FastTransform$get_inverse()
```

Returns: A new object of type `FastTransform`.

Method `get_transpose()`**:** Returns either a `FastTransform` object that represents the transpose of the transform represented by this object (if the value of the `invertible` field is `TRUE`), or an equivalent matrix.

Usage:

```
FastTransform$get_transpose()
```

Returns: Either an new object of type `FastTransform` or a matrix.

Method `get_logdet()`**:** Returns information on the determinant of the transform represented by this object. Fails if `dim_in != dim_out`. The modulus of the determinant is provided in log scale.

Usage:

```
FastTransform$get_logdet()
```

Returns: The same type of object returned by `determinant.matrix`.

Method `get_norm()`**:** Returns norm of the matrix equivalent to the linear transform represented by this object.

Usage:

```
FastTransform$get_norm(type = "o")
```

Arguments:

`type` String indicating the type of matrix norm to calculate, using the same convention as `base::norm` (default is "o", which corresponds to the maximum absolute column sum).

Returns: A numeric value.

Method `get_norm_margin()`**:** Returns norms of the rows (or columns) or the matrix equivalent to the linear transform represented by this object.

Usage:

```
FastTransform$get_norm_margin(type = "2", by = 1)
```

Arguments:

`type` String indicating the type of matrix norm to calculate, using a convention compatible with `base::norm` (default is "2", which corresponds to the Euclidian norm; use "o" for L1 norm, "m" for Inf norm).

`by` The norms of the rows are calculated by default (`by = 1`). To calculate the norms of columns instead, use `by = 2`.

Returns: A vector of numeric values.

Method `as_matrix()`**:** Returns the matrix equivalent to the transform represented by this object.

Usage:

```
FastTransform$as_matrix()
```

Returns: A matrix.

Method `print()`**:** Prints terse information about the object.

Usage:

```
FastTransform$print()
```

Returns: The object itself (invisibly).

Method `summary()`**:** Prints verbose information about the object.

Usage:

```
FastTransform$summary()
```

Returns: The object itself (invisibly).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FastTransform$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- `fort()` and fort-package, for more detailed information
- FastTransformFFT1 and FastTransformFFT2, for specific `FastTransform` subclasses

Examples

```
## ------------------------------------------------
## Method `FastTransform$evaluate`
## ------------------------------------------------

x <- fort(4) # random transform
y <- diag(4) # data to transform
x %*% y # y transformed by x
y |> x$evaluate() # same as previous line
```

`FastTransformFFT1` `FastTransformFFT1` *subclass*

Description

`FastTransformFFT1` subclass

`FastTransformFFT1` subclass

Details

A specific implementation of a structured fast transform. Inherits from FastTransform.

In particular, the `fft1` type applies the following set of operations to each input (column) vector:

1. Permute/expand (P_1) rows and pack them into a complex vector x;
2. Apply a $y = D_2 F D_1 x$ linear transform, where F represents a complex FFT, and D_i represent diagonal matrices of random unitary complex values;
3. Unpack complex vector y to real vector and permute/contract (P_2) rows.

Note that this transform will be orthonormal only when $dim\_in = dim\_out = blocksize$ (in which case, both P_1 and P_2 are permutations).

Otherwise, when $dim\_in < blocksize$, P_1 represents an expansion (rather than a permutation), and when $dim\_out < blocksize$, P_2 represents a contraction/decimation (rather than a permutation). When both of these conditions are true, the resulting transform will *not* be exactly orthogonal or semi-orthogonal, but the rows and columns of the transform are still going to be generally uncorrelated.

It is **not** recommended that the methods described below are called directly. Instead, use the methods described in the `fort()` documentation, if possible, unless you positively need low-level access (e.g., to speed up computation on *pre-validated* inputs).

Super class

`fort::FastTransform` -> `FastTransformFFT1`

Methods

Public methods:

- `FastTransformFFT1$new()`
- `FastTransformFFT1$fwd_eval()`
- `FastTransformFFT1$rev_eval()`
- `FastTransformFFT1$calculate_rev_par()`
- `FastTransformFFT1$clone()`

Method `new()`: Object creation function. It is recommended to call the `fort()` function with `type = "FastTransformFFT1"`, instead of this method, since *no input validation is performed by this method.*

Usage:

```
FastTransformFFT1$new(dim_in, dim_out, blocksize)
```

Arguments:

`dim_in` Dimensionality of the input for the forward transform.

`dim_out` Dimensionality of the output for the forward transform.

`blocksize` Dimensionality of the internal transformation (*must* be a power of 2).

Returns: A matrix with the same number of columns as `x`.

Method `fwd_eval()`: Function that performs the forward transform. Do *not* call this directly unless you know what you are doing: use the `%*%.FastTransform` or `FastTransform$evaluate()` methods instead.

Usage:

```
FastTransformFFT1$fwd_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `rev_eval()`: Function that performs the inverse transform. Do *not* call this directly unless you know what you are doing: use the `%*%.FastTransform` or `FastTransform$evaluate()` methods instead.

Usage:

```
FastTransformFFT1$rev_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `calculate_rev_par()`**:** Function that calculates and caches the parameters for the inverse transform. Do not call this directly unless you know what you are doing. If you need the inverse transform, use the `solve.FastTransform` or `FastTransform$get_inverse()` methods instead.

Usage:

```
FastTransformFFT1$calculate_rev_par()
```

Returns: The object itself (invisibly).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FastTransformFFT1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`fort()`, FastTransform

`FastTransformFFT2` `FastTransformFFT2` *subclass*

Description

`FastTransformFFT2` subclass

`FastTransformFFT2` subclass

Details

A specific implementation of a structured fast transform. Inherits from FastTransform.

In particular, the `fft2` type applies the following set of operations to each input (column) vector:

1. Permute/expand (P_1) rows and pack them into a complex vector x;
2. Apply a $y = D_3 F D_2 F D_1 x$ linear transform, where F represents a complex FFT, and D_i represent diagonal matrices of random unitary complex values;
3. Unpack complex vector y to real vector and permute/contract (P_2) rows.

Note that this transform will be orthonormal only when $dim\_in = dim\_out = blocksize$ (in which case, both P_1 and P_2 are permutations).

Otherwise, when $dim\_in < blocksize$, P_1 represents an expansion (rather than a permutation), and when $dim\_out < blocksize$, P_2 represents a contraction/decimation (rather than a permutation). When both of these conditions are true, the resulting transform will *not* be exactly orthogonal or semi-orthogonal, but the rows and columns of the transform are still going to be generally uncorrelated.

It is **not** recommended that the methods described below are called directly. Instead, use the methods described in the `fort()` documentation, if possible, unless you positively need low-level access (e.g., to speed up computation on *pre-validated* inputs).

Super class

`fort::FastTransform` -> `FastTransformFFT2`

Methods

Public methods:

- `FastTransformFFT2$new()`
- `FastTransformFFT2$fwd_eval()`
- `FastTransformFFT2$rev_eval()`
- `FastTransformFFT2$calculate_rev_par()`
- `FastTransformFFT2$clone()`

Method `new()`: Object creation function. It is recommended to call the `fort()` function with `type = "FastTransformFFT2"`, instead of this method, since *no input validation is performed by this method.*

Usage:

```
FastTransformFFT2$new(dim_in, dim_out, blocksize)
```

Arguments:

`dim_in` Dimensionality of the input for the forward transform.

`dim_out` Dimensionality of the output for the forward transform.

`blocksize` Dimensionality of the internal transformation (*must* be a power of 2).

Returns: A matrix with the same number of columns as `x`.

Method `fwd_eval()`: Function that performs the forward transform. Do *not* call this directly unless you know what you are doing: use the `%*%.FastTransform` or `FastTransform$evaluate()` methods instead.

Usage:

```
FastTransformFFT2$fwd_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `rev_eval()`: Function that performs the inverse transform. Do *not* call this directly unless you know what you are doing: use the `%*%.FastTransform` or `FastTransform$evaluate()` methods instead.

Usage:

```
FastTransformFFT2$rev_eval(x)
```

Arguments:

`x` Input matrix of the *correct* dimensionality

Returns: A matrix with the same number of columns as `x`.

Method `calculate_rev_par()`: Function that calculates and caches the parameters for the inverse transform. Do not call this directly unless you know what you are doing. If you need the inverse transform, use the `solve.FastTransform` or `FastTransform$get_inverse()` methods instead.

Usage:

```
FastTransformFFT2$calculate_rev_par()
```

Returns: The object itself (invisibly).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FastTransformFFT2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`fort()`, FastTransform

`fort` *Create a Fast Orthogonal Random Transform*

Description

`fort()` creates an object (that inherits from class FastTransform) which represents a fast random $\mathbb{R}^{dim\_in} \to \mathbb{R}^{dim\_out}$ linear transform. This transform will be orthonormal when $dim\_in = dim\_out$ *and* they are a power of 2, and approximately orthogonal or semi-orthogonal (in the sense that either $W^T W \approx I_{dim\_in}$ or $WW^T \approx I_{dim\_out}$, if W represents the transform and I_n represents an N-dimensional identity matrix) otherwise.

Usage

```
fort(
  dim_in,
  dim_out = NULL,
  type = "default",
  cache_matrix = TRUE,
  seed = NULL
)
```

Arguments

`dim_in`	Either a scalar indicating the input dimensionality, or a vector of length 2 indicating the input and output dimensionality of the transform (if dim_out is not specified).
`dim_out`	A scalar indicating the output dimensionality of the transform (not required if the first parameter is a vector of length 2).
`type`	A string indicating the type of transform to use (optional); current valid options are: `fft2` (i.e. `default`).
`cache_matrix`	Logical that controls whether matrices are cached when as.matrix() is called; should be set to FALSE if saving memory is important (optional, default = TRUE).
`seed`	If set, defines the seed used to generate the random transform (optional, default = NULL).

Details

The goal of `fort()` is to provide an easy and efficient way of calculating fast orthogonal random transforms (when `dim_in` is the same as `dim_out`) or semi-orthogonal transforms (when `dim_in` is different from `dim_out`) within R, by using fast structured transforms (like the Fast Fourier Transform or the Fast Walsh-Hadamard Transform) to avoid matrix multiplications, in the same spirit as the Fastfood (Rahimi et al. (2007)), ACDC (Moczulski et al. (2015)), HD (Yu et al. (2016)) and SD (Choromanski et al. (2017)) families of random structured transforms.

Internally, all `fort` transforms assume a blocksize which *must* be a power of 2 and no smaller than `max(dim_in, dim_out)`. The resulting transform will be practically orthonormal when $dim\_in = dim\_out$ and they match the blocksize of the transform, and practically semi-orthogonal when $dim\_in \neq dim\_out$ and `max(dim_in, dim_out)` matches the blocksize. Otherwise, these properties will *only approximately* hold, since the output will result from a decimated transform (i.e., the rows and columns of the transform should be decorrelated, but not necessarily orthogonal).

Value

An object of a class that inherits from class FastTransform and which represents a fast linear transform.

`fort` transform types

The specific type of transform returned depends on the value passed in the `type` parameter, but all methods rely on alternating between applying permutations (complexity $O(N)$), diagonal scaling matrices (complexity $O(N)$) and structured fast linear transforms (such as the Fast Fourier Transform or the Fast Walsh-Hadamard Transform, which can be implemented with complexity $O(N \log N)$). Thus, it becomes possible to reduce the complexity of transforming an $\mathbb{R}^N$ vector from $O(N^2)$ (using matrix multiplication) to $O(N \log N)$.

Currently, the available options for the `type` parameter are:

- `default`: this is the default option, if no `type` is specified; currently, it assumes the `fft2` type, but this is subject to change (so avoid this option in non-interactive usage);
- `fft1`: this type of `fort` transform uses the Fast Fourier Transform as base transform (which is used once); for more technical details, see FastTransformFFT1.
- `fft2`: this type of `fort` transform uses the Fast Fourier Transform as base transform (which is used twice); for more technical details, see FastTransformFFT2.

Using `fort` transforms

In practice, to apply the fast transform to the columns of a matrix, you should use the `%*%` operator as if the output of `fort()` was a matrix (e.g., `fort(4,6) %*% matrix(1:12,4,3)` will output a 6 by 3 matrix that results from applying the transform on the left to the matrix on the right of the `%*%` operator).

Objects generated by `fort()` are also compatible with other methods applicable to `matrix` objects, such as `dim()`, `ncol()`, `nrow()`, `solve()`, `t()` and `det()`. Furthermore, these object can also be easily converted to matrices (using `as.matrix()`), if required.

References

Krzysztof M. Choromanski, Mark Rowland, and Adrian Weller. (2017). The unreasonable effectiveness of structured random orthogonal embeddings. *Conference and Workshop on Neural Information Processing Systems*. `http://papers.neurips.cc/paper/6626-the-unreasonable-effectiveness-of-str`

Felix Xinnan X. Yu, Ananda Theertha Suresh, Krzysztof M. Choromanski, Daniel N. Holtmann-Rice, and Sanjiv Kumar. (2016). Orthogonal random features. *Conference and Workshop on Neural Information Processing Systems*. http://papers.neurips.cc/paper/6246-orthogonal-random-features

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. (2015). ACDC: A structured efficient linear layer. https://arxiv.org/abs/1511.05946

Quoc Le, Tamás Sarlós and Alex Smola. (2013). Fastfood - approximating kernel expansions in loglinear time. *International Conference on Machine Learning*. https://proceedings.mlr.press/v28/le13-supp.pdf

See Also

- How to apply `fort` transforms: `%*%.FastTransform`
- How to obtain a `fort` transform in matrix form: `as.matrix.FastTransform()`
- How to invert `fort` transforms: `solve.FastTransform()`
- How to access low-level functionality of `fort` transforms: `FastTransform`

Examples

```
fort(16) # a random orthogonal transform from R^16 to R^16
fort(5, 33) # a random transform from R^5 to R^33
fort(c(5, 33)) # same as previous line
# apply a random orthogonal transformation to the canonical R^4 basis
fort(4) %*% diag(4)
```

`solve.FastTransform` *Solve a System of Equations*

Description

Solves an equation of the form `a %*% x = b` for `x`, where `a` is a linear operation represented by a `FastTransform` object, while `b` can be either a vector or a matrix. If `b` is missing, it returns a `FastTransform` object corresponding to the inverse (or a generalized inverse) of `a`.

Usage

```
## S3 method for class 'FastTransform'
solve(a, b, ...)
```

Arguments

`a`	An object of class `FastTransform`, created using `fort()`.
`b`	A numeric vector or matrix (to solve the equation), or nothing (to obtain a generalized inverse of `a`).
`...`	Extra parameters (ignored).

Details

Note that the inverse transform will only be fast (i.e., avoid matrix multiplication) if $dim\_in = dim\_out = blocksize$.

Value

Either a matrix (representing x), or a FastTransform object (representing a generalized inverse of a; if parameter b is missing).

See Also

fort()

Examples

```
a <- fort(4)
inv_a <- solve(a) # inverse of a
inv_a %*% diag(4) # applying the inverse of a
solve(a, diag(4)) # should give the same output
```

summary.FastTransform *Summarize fast transform*

Description

Provides a summary of a fast transform created by fort() with slightly more detail than the information provided by using print().

Usage

```
## S3 method for class 'FastTransform'
summary(object, ...)
```

Arguments

object	An object of class FastTransform, created using fort().
...	Extra parameters (ignored).

Value

The input object (invisibly).

See Also

fort()

Examples

```
summary(fort(3, 17))
```

t.FastTransform *Transform Transpose*

Description

Given a FastTransform object x, t returns the transpose of x. If x represents an orthonormal transformation (i.e., if x$invertible is TRUE), then a FastTransform object (representing the transpose of x) will be returned; otherwise, a matrix object (representing the transpose of x) will be returned, with a warning.

Usage

```
## S3 method for class 'FastTransform'
t(x)
```

Arguments

x	An object of class FastTransform, created using fort().

Value

Either an object of class FastTransform (if x$invertible is TRUE) or a matrix.

See Also

fort() and solve.FastTransform()

Examples

```
(a <- fort(4))
(b <- t(t(a))) # transpose a twice
# the result below should be close to zero
sum((a %*% diag(4) - b %*% diag(4))^2)
```

%*%.FastTransform *Apply a fast transform*

Description

Applies a fast transform created by fort() (x) to the columns of a conformable matrix (y).

Usage

```
## S3 method for class 'FastTransform'
x %*% y
```

Arguments

x	An object of class FastTransform, created using fort().
y	A numeric (real) matrix/vector with an appropriate number of rows/elements.

Value

A numeric (real) matrix with the same number of columns as y.

See Also

fort() to create FastTransform objects, and %***% for **unsafe** evaluation

Examples

```
Z <- fort(4, 1024)
Z %*% matrix(1:2, 4, 3) # output is a 1024 by 3 matrix
# the example below works: y is assumed to be a single column vector
Z %*% 1:4 # output is a 1024 by 1 matrix
```

%***% *Unsafely apply a fast transform*

Description

Applies a fast transform created by fort() (x) to the columns of a conformable matrix (y), typically equivalent to the use of the %*% operator, but using an **unsafe** method.

Usage

```
x %***% y
```

Arguments

x An object of class FastTransform, created using fort().

y A numeric (real) matrix/vector with an appropriate number of rows/elements.

Details

This operator works in a similar way to %*%, but avoids dispatching and does not perform any type of validation of its inputs, in order to reduce overhead when performing repeated operations inside a function on *pre-validated* inputs.

It is *not* recommended that this operator is used interactively and/or on non-validated inputs.

Value

A numeric (real) matrix with the same number of columns as y.

See Also

fort(), %*%.FastTransform

Examples

```
Z <- fort(4, 1024)
Z %*% matrix(1:2, 4, 3) # output is a 1024 by 3 matrix
Z %***% matrix(1:2, 4, 3) # output is also a 1024 by 3 matrix
```

Index