

Package ‘zufft’

October 22, 2026

Title Fast Fourier, Cosine and Sine Transforms

Version 0.1.0

Description Computes discrete Fourier transforms (complex, real-to-complex and complex-to-real) and discrete cosine and sine transforms of types one to four over any axes of an array, with the fast Fourier transform (FFT) code of 'ducc0' ([\(<https://github.com/mreineck/ducc>\)](https://github.com/mreineck/ducc)) bundled in the BSD-licensed subset that 'SciPy' uses. Transforms of any length run in $O(n \log n)$ time, real input uses true real transforms, and several threads can share one transform.

License MIT + file LICENSE

URL <https://github.com/pedrobtz/zufft>,
<https://pedrobtz.github.io/zufft/>

BugReports <https://github.com/pedrobtz/zufft/issues>

Depends R (≥ 4.1)

Suggests fftwtools, knitr, parallel, rmarkdown, testthat ($\geq 3.0.0$),
withr

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Copyright See inst/COPYRIGHTS.

Encoding UTF-8

Language en-GB

SystemRequirements C++17

NeedsCompilation yes

Author Pedro Baltazar [aut, cre, cph],
Martin Reinecke [ctb] (ducc0, bundled in src/ducc0),
Peter Bell [ctb, cph] (ducc0, bundled in src/ducc0),
Max-Planck-Society [cph] (ducc0, bundled in src/ducc0),
Matteo Frigo [cph] (odd-sized DCT-IV in src/ducc0/fft/fft1d_impl.h),

Massachusetts Institute of Technology [cph] (odd-sized DCT-IV in
src/ducc0/fft/fft1d_impl.h),
The TensorFlow Authors [cph] (helpers in src/ducc0/infra/threading.cc),
SciPy Developers [cph] (fast-length search in src/zff_lengths.cpp, from
SciPy)

Maintainer Pedro Baltazar <pedrobtz@gmail.com>

Repository CRAN

Date/Publication 2026-10-22 11:10:02 UTC

Contents

fft_c2c	2
fft_dct	4
fft_freq	7
fft_next_len	7
fft_shift	8
fft_stats	9
fft_workers	10
zufft_info	11

Index	13
--------------	-----------

fft_c2c	<i>Discrete Fourier transforms</i>
---------	------------------------------------

Description

fft_c2c() computes the complex discrete Fourier transform of x or its inverse, fft_r2c() the transform of real input keeping only the non-redundant half of the last transformed axis, and fft_c2r() the inverse of fft_r2c(), returning real values. All three work over any subset of the dimensions of an array, in O(n log n) time for every length.

Usage

```
fft_c2c(  
  x,  
  inverse = FALSE,  
  n = NULL,  
  axes = NULL,  
  norm = c("backward", "ortho", "forward", "none"),  
  workers = NULL  
)  
  
fft_r2c(  
  x,  
  n = NULL,
```

```

    axes = NULL,
    norm = c("backward", "ortho", "forward", "none"),
    workers = NULL
)

fft_c2r(
  x,
  n = NULL,
  axes = NULL,
  norm = c("backward", "ortho", "forward", "none"),
  workers = NULL
)

```

Arguments

<code>x</code>	A double, integer, logical or complex vector or array. Integer and logical values are converted to double. <code>fft_r2c()</code> needs real input; <code>fft_c2r()</code> treats real input as complex.
<code>inverse</code>	TRUE for the inverse transform. 0 and 1 are accepted too, as in <code>fftwtools</code> .
<code>n</code>	Transform lengths, one per transformed axis in the order of <code>axes</code> (no recycling); with <code>axes = NULL</code> , one per dimension, in dimension order. Each axis of <code>x</code> is cropped or zero-padded to its length first. <code>NULL</code> keeps the lengths of <code>x</code> . For <code>fft_c2r()</code> , <code>n</code> gives the lengths of the real output; the halved axis of <code>x</code> must then hold $n \% 2 + 1$ values and is cropped or padded to that. Its default is $2 * (m - 1)$ for an input extent <code>m</code> , which is always even: give <code>n</code> to recover an odd length.
<code>axes</code>	The dimensions to transform, 1-based, without repeats. <code>NULL</code> means all of them, in order; for <code>fft_r2c()</code> and <code>fft_c2r()</code> it means all of them in reverse order, so the first dimension is halved. With explicit <code>axes</code> , the real transforms halve the last axis listed, as SciPy does: <code>axes = seq_along(dim(x))</code> halves the last dimension. Use <code>axes = 1</code> for a column-wise transform of a matrix, as <code>stats::mvfft()</code> does.
<code>norm</code>	Where the scaling goes. "backward" (the default) leaves the forward transform unscaled and divides the inverse by <code>N</code> , the product of the transformed lengths; "forward" does the opposite; "ortho" divides both by $\sqrt{N}$; "none" scales neither direction, as FFTW, <code>fftwtools</code> and <code>stats::fft()</code> do.
<code>workers</code>	Threads to use. <code>NULL</code> uses the session default (1). Negative values count back from the number of available cores: -1 means all of them. Requests are capped at the available cores, and at 2 when <code>_R_CHECK_LIMIT_CORES_</code> is set. Threads split the independent 1-D transforms of a multi-dimensional array, so they help only arrays with at least 32 768 elements and more than one line. Results agree to rounding whatever the number of threads; they are bit-identical on builds without fused multiply-add (CRAN's x86-64 builds).

Details

The conventions are those of `scipy.fft` (`fft`, `fftn`, `rfftn`, `irfftn`): `n` crops or zero-pads each transformed axis before the transform, and `norm` chooses where the scaling goes. They differ from

SciPy in three ways: axes are 1-based; axes = NULL means every dimension rather than the last one; and with axes = NULL the real transforms halve the *first* dimension, not the last. R stores arrays column by column, so the first dimension is the contiguous one, and halving it is the fast choice (about 1.8 times faster than halving the last for a 2048 x 2048 matrix). It is also what fftwtools' 2-D and 3-D real transforms do.

Coming from fftwtools: `fft_c2c(x)` is `fftw(x)` and `fftw_r2c(x)`, `fft_r2c(x)` is `fftw_r2c(x, HermConj = 0)`, and an fftwtools inverse is the `zufft` inverse with `norm = "none"`, because fftwtools, like `stats::fft()`, never scales.

Value

For `fft_c2c()` and `fft_r2c()` a complex vector or array, for `fft_c2r()` a double one. Arrays keep their dim, with the transformed extents, and the dimnames of axes that were not transformed; names are dropped.

See Also

`stats::fft()` for R's own complex transform.

Other transforms: `fft_dct()`

Examples

```
x <- c(1, 2, 3, 4, 3, 2)
fft_c2c(x)
fft_r2c(x)
fft_c2r(fft_r2c(x), n = length(x))

# an inverse that matches fftwtools and stats::fft(inverse = TRUE)
fft_c2c(fft_c2c(x), inverse = TRUE, norm = "none") / length(x)

# columns of a matrix, and both axes
m <- matrix(rnorm(12), 4, 3)
fft_c2c(m, axes = 1)
dim(fft_r2c(m))           # halves the first dimension: 3 x 3
dim(fft_r2c(m, axes = 1:2)) # SciPy's choice, the last: 4 x 2
fft_c2r(fft_r2c(m), n = dim(m)) # back to m
```

fft_dct

Discrete cosine and sine transforms

Description

`fft_dct()` and `fft_dst()` compute discrete cosine and sine transforms of types 1 to 4, or their inverses, over any subset of the dimensions of an array. The definitions, scaling and orthogonalize behaviour are those of `scipy.fft.dct()`, `dctn()`, `dst()` and `dstn()`.

Usage

```

fft_dct(
    x,
    type = 2L,
    inverse = FALSE,
    n = NULL,
    axes = NULL,
    norm = c("backward", "ortho", "forward", "none"),
    orthogonalize = NULL,
    workers = NULL
)

fft_dst(
    x,
    type = 2L,
    inverse = FALSE,
    n = NULL,
    axes = NULL,
    norm = c("backward", "ortho", "forward", "none"),
    orthogonalize = NULL,
    workers = NULL
)

```

Arguments

<code>x</code>	A double, integer, logical or complex vector or array. Complex input is transformed as its real and imaginary parts separately.
<code>type</code>	The transform type: 1, 2, 3 or 4.
<code>inverse</code>	TRUE for the inverse transform.
<code>n</code>	Transform lengths, one per transformed axis in the order of <code>axes</code> (no recycling); with <code>axes = NULL</code> , one per dimension, in dimension order. Each axis of <code>x</code> is cropped or zero-padded to its length first. <code>NULL</code> keeps the lengths of <code>x</code> . For <code>fft_c2r()</code> , <code>n</code> gives the lengths of the real output; the halved axis of <code>x</code> must then hold $n \% 2 + 1$ values and is cropped or padded to that. Its default is $2 * (m - 1)$ for an input extent <code>m</code> , which is always even: give <code>n</code> to recover an odd length.
<code>axes</code>	The dimensions to transform, 1-based, without repeats. <code>NULL</code> means all of them, in order; for <code>fft_r2c()</code> and <code>fft_c2r()</code> it means all of them in reverse order, so the first dimension is halved. With explicit <code>axes</code> , the real transforms halve the last axis listed, as SciPy does: <code>axes = seq_along(dim(x))</code> halves the last dimension. Use <code>axes = 1</code> for a column-wise transform of a matrix, as <code>stats::mvfft()</code> does.
<code>norm</code>	Where the scaling goes, as in <code>fft_c2c()</code> . "ortho" makes the transform orthonormal when <code>orthogonalize</code> is TRUE.
<code>orthogonalize</code>	Whether to scale the first and last terms so that the "ortho" transform matrix is orthogonal. <code>NULL</code> (the default) means <code>norm == "ortho"</code> , as in SciPy. For complex input an explicit value is ignored and <code>norm == "ortho"</code> is used, as SciPy does.

workers Threads to use. NULL uses the session default (1). Negative values count back from the number of available cores: -1 means all of them. Requests are capped at the available cores, and at 2 when `_R_CHECK_LIMIT_CORES_` is set. Threads split the independent 1-D transforms of a multi-dimensional array, so they help only arrays with at least 32 768 elements and more than one line. Results agree to rounding whatever the number of threads; they are bit-identical on builds without fused multiply-add (CRAN's x86-64 builds).

Details

Unnormalised (`norm = "backward"` or `"none"` in the forward direction), for a length N , `fft_dct()` computes

- type 1: $y[k] = x[0] + (-1)^k x[N-1] + 2 \sum_{n=1}^{N-2} x[n] \cos(\pi k n / (N-1))$
- type 2: $y[k] = 2 \sum_{n=0}^{N-1} x[n] \cos(\pi k (2n+1) / (2N))$
- type 3: $y[k] = x[0] + 2 \sum_{n=1}^{N-1} x[n] \cos(\pi (2k+1) n / (2N))$
- type 4: $y[k] = 2 \sum_{n=0}^{N-1} x[n] \cos(\pi (2k+1) (2n+1) / (4N))$

and `fft_dst()` the matching sine sums (type 1 over $N+1$, types 2 to 4 as above with sines and indices shifted by one). Type 1 needs lengths of at least 2 for the cosine transform.

`inverse = TRUE` computes the inverse: types 2 and 3 swap, and the scaling moves to the other direction. With `norm = "backward"` the inverse is scaled by $1/N$, where N is $2(n-1)$ for DCT-I, $2(n+1)$ for DST-I and $2n$ otherwise, per transformed axis, so that `fft_dct(fft_dct(x), inverse = TRUE)` returns x .

Value

A double vector or array (complex for complex input) of the same shape as x after cropping or padding to n .

See Also

Other transforms: [fft_c2c\(\)](#)

Examples

```
x <- c(1, 2, 3, 4, 3, 2)
fft_dct(x)
fft_dct(fft_dct(x), inverse = TRUE)
fft_dst(x, type = 4, norm = "ortho")

# an orthonormal DCT-II along the columns of a matrix
m <- matrix(rnorm(12), 4, 3)
fft_dct(m, axes = 1, norm = "ortho")
```

fft_freq

*Sample frequencies***Description**

`fft_freq()` returns the frequency of each entry of a length- n transform from `fft_c2c()`, in cycles per unit of the sample spacing d : $0, 1, \dots, \text{ceiling}(n / 2) - 1, -\text{floor}(n / 2), \dots, -1$, divided by $d * n$. `fft_rfreq()` returns the frequencies of the $n \% 2 + 1$ entries of `fft_r2c()`: $0, 1, \dots, n \% 2$, divided by $d * n$. They are SciPy's `fftfreq()` and `rfftfreq()`.

Usage

```
fft_freq(n, d = 1)
```

```
fft_rfreq(n, d = 1)
```

Arguments

n The transform length, a positive whole number.

d The sample spacing, a positive number.

Value

A double vector.

See Also

Other helpers: `fft_next_len()`, `fft_shift()`

Examples

```
fft_freq(8)
fft_freq(7, d = 0.1)
fft_rfreq(8)
```

fft_next_len

*Fast transform lengths***Description**

`fft_next_len()` returns the smallest length at least n that `zufft` transforms fastest, and `fft_prev_len()` the largest at most n . Padding or cropping an input to such a length with the n argument of the transforms can speed them up considerably. They are SciPy's `next_fast_len()` and `prev_fast_len()`.

Usage

```
fft_next_len(n, real = FALSE)
```

```
fft_prev_len(n, real = FALSE)
```

Arguments

n Positive whole numbers, at most 2^{52} .

real TRUE for the lengths of real transforms (`fft_r2c()`, `fft_c2r()`, `fft_dct()`, `fft_dst()`).

Details

Fast lengths have no prime factor above 11 for complex transforms, or above 5 for real ones (`real = TRUE`), as in `ducc0`'s `good_size`.

Value

A double vector of the same length as `n`.

See Also

`stats::nextn()`, R's own version for `stats::fft()`.

Other helpers: `fft_freq()`, `fft_shift()`

Examples

```
fft_next_len(c(1009, 999983))
fft_next_len(1009, real = TRUE)
fft_prev_len(1009)
```

fft_shift
Shift the zero frequency to the centre

Description

`fft_shift()` rotates each chosen dimension of `x` by $\text{floor}(n / 2)$, so that the zero-frequency entry of a transform sits in the middle; `fft_ishift()` undoes it. They are SciPy's `fftshift()` and `ifftshift()`. `dim`, `dimnames` and `names` move with the values.

Usage

```
fft_shift(x, axes = NULL)
```

```
fft_ishift(x, axes = NULL)
```

Arguments

`x` A vector or array.

`axes` The dimensions to shift, 1-based. NULL means all of them.

Value

`x`, rotated.

See Also

Other helpers: [fft_freq\(\)](#), [fft_next_len\(\)](#)

Examples

```
fft_shift(fft_freq(8))
fft_ishift(fft_shift(1:7))
```

fft_stats	<i>Drop-in replacements for stats::fft() and stats::mvfft()</i>
-----------	---

Description

`fft_stats()` and `mvfft_stats()` take the same arguments, check them the same way and return the same structure as [stats::fft\(\)](#) and [stats::mvfft\(\)](#): the attributes of `z` are kept, the inverse is not scaled, and inputs of length 0 or 1 come back unchanged (as complex). The values agree to rounding. The differences: every length runs in $O(n \log n)$ (R's code is quadratic in a large prime factor), real input uses a real transform, and threads follow [fft_workers\(\)](#).

Usage

```
fft_stats(z, inverse = FALSE)

mvfft_stats(z, inverse = FALSE)
```

Arguments

`z` A real or complex vector or array (`fft_stats()`), or matrix (`mvfft_stats()`).

`inverse` TRUE for the unnormalised inverse transform. NA means forward, as in [stats::fft\(\)](#).

Details

A package moves by replacing `fft(` with `zufft::fft_stats(` and adding `zufft` to Imports.

Value

A complex vector, array or matrix with the attributes of `z`.

Examples

```
x <- 1:4
fft_stats(x)
fft_stats(fft_stats(x), inverse = TRUE) / length(x)
all.equal(fft_stats(x), stats::fft(x))

m <- matrix(rnorm(12), 4)
all.equal(mvfft_stats(m), stats::mvfft(m))
```

fft_workers	<i>Threads per transform</i>
-------------	------------------------------

Description

fft_workers() gets or sets the number of threads a transform uses when its workers argument is NULL. with_fft_workers() sets it while code runs and local_fft_workers() until the calling function returns, in the style of the withr package; both restore the old value. Together they are SciPy's get_workers() and set_workers().

Usage

```
fft_workers(n = NULL)

with_fft_workers(n, code)

local_fft_workers(n, .local_envir = parent.frame())
```

Arguments

n	A non-zero whole number. NULL (in fft_workers()) only reads the current value.
code	Code to run with n threads.
.local_envir	The environment whose exit restores the old value.

Details

A positive value asks for that many threads; a negative one counts back from the number of available cores, so -1 means all of them. Requests are capped at the available cores, and at 2 when _R_CHECK_LIMIT_CORES_ is set (as during R CMD check). The initial value is 1, as in SciPy.

Threads help when a transform has many independent lines (the columns of a matrix, a multi-dimensional array) and at least 32 768 elements; a single 1-D real transform always runs on one thread. Results agree to rounding whatever the number of threads, and are bit-identical on builds without fused multiply-add, such as CRAN's x86-64 binaries.

Value

`fft_workers()` returns the current value when `n` is `NULL`, and otherwise the previous value, invisibly. `with_fft_workers()` returns the value of code; `local_fft_workers()` returns the previous value, invisibly.

Examples

```
fft_workers()
with_fft_workers(2, fft_c2c(matrix(rnorm(1e4), 100), axes = 1))
f <- function() {
  local_fft_workers(2)
  fft_workers()
}
f()
fft_workers()
```

zufft_info

Information about the zufft build

Description

Reports which ducc0 source zufft was built from and how: the vendored commit and the SciPy commit that pins it, the SIMD backend and vector width the compiler selected, whether the target has fused multiply-add instructions, the threading mode, and the thread pool.

Usage

```
zufft_info()
```

Value

A named list:

`ducc0_commit` ducc0 commit of the vendored FFT subset.

`scipy_commit` SciPy commit whose vendoring script pins it.

`simd_backend` "ducc0", "std::experimental::simd" or "none".

`simd_width` Doubles per SIMD vector, 1 without SIMD.

`fma` TRUE if the target has fused multiply-add.

`fp_contract` "off" when the transforms never fuse multiply-adds, so results are bit-identical whatever workers is; "fast" for a GCC build targeting FMA hardware, where they can differ in the last bit. CRAN's builds are "off".

`threading` "zufft" when zufft supplies the thread pool, "none" when threads are compiled out.

`pool_size` Threads the pool can use now, the caller included.

`hardware_threads` Threads the hardware offers.

`complex_path` "zero-copy" when complex vectors are viewed in place, "marshal" when they are copied (design §12.2).

Examples

```
zufft_info()$simd_width
```

Index

- * **compatibility**
 - fft_stats, [9](#)
- * **helpers**
 - fft_freq, [7](#)
 - fft_next_len, [7](#)
 - fft_shift, [8](#)
- * **threads**
 - fft_workers, [10](#)
- * **transforms**
 - fft_c2c, [2](#)
 - fft_dct, [4](#)

fft_c2c, [2](#)
fft_c2c(), [5–7](#)
fft_c2r (fft_c2c), [2](#)
fft_c2r(), [8](#)
fft_dct, [4](#)
fft_dct(), [4, 8](#)
fft_dst (fft_dct), [4](#)
fft_dst(), [8](#)
fft_freq, [7](#)
fft_freq(), [8, 9](#)
fft_ishift (fft_shift), [8](#)
fft_next_len, [7](#)
fft_next_len(), [7, 9](#)
fft_prev_len (fft_next_len), [7](#)
fft_r2c (fft_c2c), [2](#)
fft_r2c(), [7, 8](#)
fft_rfreq (fft_freq), [7](#)
fft_shift, [8](#)
fft_shift(), [7, 8](#)
fft_stats, [9](#)
fft_workers, [10](#)
fft_workers(), [9](#)

local_fft_workers (fft_workers), [10](#)

mvfft_stats (fft_stats), [9](#)

stats::fft(), [3, 4, 8, 9](#)

stats::mvfft(), [3, 5, 9](#)
stats::nextn(), [8](#)

with_fft_workers (fft_workers), [10](#)

zufft_info, [11](#)