Title

stata.com

fft( ) — Fourier transform

Syntax Description Remarks and examples Conformability
Diagnostics Also see

Syntax

complex vector	`fft(`*numeric vector h*`)`
numeric vector	`invfft(`*numeric vector H*`)`
void	`_fft(`*complex vector h*`)`
void	`_invfft(`*complex vector H*`)`
numeric vector	`convolve(`*numeric vector r*`,` *numeric vector s*`)`
numeric vector	`deconvolve(`*numeric vector r*`,` *numeric vector sm*`)`
numeric vector	`Corr(`*numeric vector g*`,` *numeric vector h*`,` *real scalar k*`)`
real vector	`ftperiodogram(`*numeric vector H*`)`
numeric vector	`ftpad(`*numeric vector h*`)`
numeric vector	`ftwrap(`*numeric vector r*`,` *real scalar n*`)`
numeric vector	`ftunwrap(`*numeric vector H*`)`
numeric vector	`ftretime(`*numeric vector r*`,` *numeric vector s*`)`
real vector	`ftfreqs(`*numeric vector H*`,` *real scalar delta*`)`

Description

H=`fft(`*h*`)` and *h*=`invfft(`*H*`)` calculate the Fourier transform and inverse Fourier transform. The length of *h* (*H*) must be a power of 2.

`_fft(`*h*`)` and `_invfft(`*H*`)` do the same thing, but they perform the calculation in place, replacing the contents of *h* and *H*.

`convolve(`*r*`,` *s*`)` returns the convolution of the signal *s* with the response function *r*. `deconvolve(`*r*`,` *sm*`)` deconvolves the smeared signal *sm* with the response function *r* and is thus the inverse of `convolve()`.

`Corr(`*g*`,` *h*`,` *k*`)` returns a $2k + 1$ element vector containing the correlations of *g* and *h* for lags and leads as large as *k*.

`ftperiodogram(`*H*`)` returns a real vector containing the one-sided periodogram of *H*.

`ftpad(`*h*`)` returns *h* padded with 0s to have a length that is a power of 2.

`ftwrap(`*r*`, `*n*`)` converts the symmetrically stored response function *r* into wraparound format of length *n*, $n \geq$ `rows(`*r*`)*cols(`*r*`)` and `rows(`*r*`)*cols(`*r*`)` odd.

`ftunwrap(`*H*`)` unwraps frequency-wraparound order such as returned by `fft()`. You may find this useful when graphing or listing results, but it is otherwise unnecessary.

`ftretime(`*r*`, `*s*`)` retimes the signal *s* to be on the same time scale as `convolve(`*r*`, `*s*`)`. This is useful in graphing data and listing results but is otherwise not required.

`ftfreqs(`*H*`, `*delta*`)` returns a vector containing the frequencies associated with the elements of *H*; *delta* is the sampling interval and is often specified as 1.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Definitions, notation, and conventions
Fourier transform
Convolution and deconvolution
Correlation
Utility routines
Warnings

Definitions, notation, and conventions

A signal *h* is a row or column vector containing real or complex elements. The length of the signal is defined as the number of elements of the vector. It is occasionally necessary to pad a signal to a given length. This is done by forming a new vector equal to the original and with zeros added to the end.

The Fourier transform of a signal *h*, typically denoted by capital letter *H* of *h*, is stored in frequency-wraparound order. That is, if there are *n* elements in *H*:

$H[1]$	frequency 0
$H[2]$	frequency 1
$H[3]$	frequency 2
⋮	
$H[n/2]$	frequency $n/2-1$
$H[n/2+1]$	frequency $n/2$ ($-n/2$, aliased)
$H[n/2+2]$	frequency $-(n/2-1)$
⋮	
$H[n-1]$	frequency -2
$H[n]$	frequency -1

All routines expect and use this order, but see `ftunwrap()` below.

A response function *r* is a row or column vector containing $m = 2k + 1$ real or complex elements. *m* is called the duration of the response function. Response functions are generally stored symmetrically, although the response function itself need not be symmetric. The response vector contains

$r[1]$	response at lag $-k$
$r[2]$	response at lag $-k+1$
⋮	
$r[k]$	response at lag -1
$r[k+1]$	contemporaneous response
$r[k+2]$	response at lead 1
$r[k+3]$	response at lead 2
⋮	
$r[2k+1]$	response at lead k

Response functions always have odd lengths. Response vectors are never padded.

You may occasionally find it convenient to store a response vector in "wraparound" order (similar to frequency-wraparound order), although none of the routines here require this. In wraparound order:

wrap$[1]$	contemporaneous response
wrap$[2]$	response at lead 1
wrap$[3]$	response at lead 2
⋮	
wrap$[k+1]$	response at lead k
wrap$[k+2]$	response at lag $-k$
wrap$[k+3]$	response at lag $-k+1$
⋮	
wrap$[2k+1]$	response at lag -1

Response vectors stored in wraparound order may be internally padded (as opposed to merely padded) to a given length by the insertion of zeros between wrap$[k+1]$ and wrap$[k+2]$.

Fourier transform

`fft(`*h*`)` returns the discrete Fourier transform of *h*. *h* may be either real or complex, but its length must be a power of 2, so one typically codes `fft(ftpad(`*h*`))`; see `ftpad()`, below. The returned result is p-conformable with *h*. The calculation is performed by `_fft()`.

`invfft(`*H*`)` returns the discrete inverse Fourier transform of *H*. *H* may be either real or complex, but its length must be a power of 2. The returned result is p-conformable with *H*. The calculation is performed by `_invfft()`.

`invfft(`*H*`)` may return a real or complex. This should be viewed as a feature, but if you wish to ensure the complex interpretation, code `C(invfft(`*H*`))`.

`_fft(`*h*`)` is the built-in procedure that performs the fast Fourier transform in place. *h* must be complex, and its length must be a power of 2.

`_invfft(`*H*`)` is the built-in procedure that performs the inverse fast Fourier transform in place. *H* must be complex, and its length must be a power of 2.

Convolution and deconvolution

`convolve(r, s)` returns the convolution of the signal *s* with the response function *r*. Calculation is performed by taking the `fft()` of the elements, multiplying, and using `invfft()` to transform the results back. Nevertheless, it is not necessary that the length of *s* be a power of 2. `convolve()` handles all paddings necessary, including paddings to *s* required to prevent the result from being contaminated by erroneous wrapping around of *s*. Although one thinks of the convolution operator as being commutative, `convolve()` is not commutative since required zero-padding of the response and signal differ.

If n is the length of the signal and $2k + 1$ is the length of the response function, the returned result has length $n + 2k$. The first k elements are the convoluted signal before the true signal begins, and the last k elements are the convoluted signal after the true signal ends. See `ftretime()`, below. In any case, you may be interested only in the elements `convolve()[|`*k*`+1\`*n*`-`*k*`|]`, the part contemporaneous with *s*.

The returned vector is a row vector if *s* is a row vector and a column vector otherwise. The result is guaranteed to be real if both *r* and *s* are real; the result may be complex or real, otherwise.

It is not required that the response function be shorter than the signal, although this will typically be the case.

`deconvolve(r, sm)` deconvolves the smeared signal *sm* with the response function *r* and is thus the inverse of `convolve()`. In particular,

`deconvolve(`*r*`, convolve(`*r*`,`*s*`))` $= s$ (up to roundoff error)

Everything said about `convolve()` applies equally to `deconvolve()`.

Correlation

Here we refer to correlation in the signal-processing sense, not the statistical sense.

`Corr(g, h, k)` returns a $2k + 1$ element vector containing the correlations of *g* and *h* for lags and leads as large as *k*. For instance, `Corr(g, h, 2)` returns a five-element vector, the first element of which contains the correlation for lag 2, the second element lag 1, the third (middle) element the contemporaneous correlation, the fourth element lead 1, and the fifth element lead 2. *k* must be greater than or equal to 1. The returned vector is a row or column vector depending on whether *g* is a row or column vector. *g* and *h* must have the same number of elements but need not be p-conformable.

The result is obtained by padding with zeros to avoid contamination, taking the Fourier transform, multiplying $G \times \text{conj}(H)$, and rearranging the inverse transformed result. Nevertheless, it is not required that the number of elements of *g* and *h* be powers of 2 because the program pads internally.

Utility routines

`ftpad(h)` returns *h* padded with 0s to have a length that is a power of 2. For instance,

```
: h = (1,2,3,4,5)
: ftpad(h)
        1   2   3   4   5   6   7   8
    +-------------------------------+
  1 |  1   2   3   4   5   0   0   0 |
    +-------------------------------+
```

If h is a row vector, a row vector is returned. If h is a column vector, a column vector is returned.

`ftwrap(`r`, `n`)` converts the symmetrically stored response function r into wraparound format of length n, $n \geq$ `rows(`r`)*cols(`r`)` and `rows(`r`)*cols(`r`)` odd. A symmetrically stored response function is a vector of odd length, for example:

```
(.1, .5, 1, .2, .4)
```

The middle element of the vector represents the response function at lag 0. Elements before the middle represent lags while elements after the middle represent leads. Here .1 is the response for lag 2 and .5 for lag 1, 1 the contemporaneous response, .2 the response for lead 1, and .4 the response for lead 2. The wraparound format of a response function records the response at times 0, 1, and so on in the first positions, has some number of zeros, and then records the most negative time value of the response function, and so on.

For instance,

```
: r
                1    2    3    4    5
            1 |  .1   .5   1   .2   .4 |

: ftwrap(r, 5)
                1    2    3    4    5
            1 |  1   .2   .4   .1   .5 |

: ftwrap(r, 6)
                1    2    3    4    5    6
            1 |  1   .2   .4   0   .1   .5 |

: ftwrap(r, 8)
                1    2    3    4    5    6    7    8
            1 |  1   .2   .4   0    0    0   .1   .5 |
```

`ftunwrap(`H`)` unwraps frequency-wraparound order such as returned by `fft()`. You may find this useful when graphing or listing results, but it is otherwise unnecessary. Frequency-unwrapped order is defined as

unwrap$[1]$	frequency $-(n/2) + 1$
unwrap$[2]$	frequency $-(n/2) + 2$
$\vdots$	
unwrap$[n/2 - 1]$	frequency -1
unwrap$[n/2]$	frequency 0
unwrap$[n/2 + 1]$	frequency 1
$\vdots$	
unwrap$[n - 1]$	frequency $n/2 - 1$
unwrap$[n]$	frequency $n/2$

Here we assume that n is even, as will usually be true. The aliased (highest) frequency is assigned the positive sign.

Also see ftperiodogram(), below.

ftretime(*r*, *s*) retimes the signal *s* to be on the same time scale as convolve(*r*, *s*). This is useful in graphing and listing results but is otherwise not required. ftretime() uses only the length of *r*, and not its contents, to perform the retiming. If the response vector is of length $2k + 1$, a vector containing *k* zeros, *s*, and *k* more zeros is returned. Thus the result of ftretime(*r*, *s*) is p-conformable with convolve(*r*, *s*).

ftfreqs(*H*, *delta*) returns a p-conformable-with-*H* vector containing the frequencies associated with the elements of *H*. *delta* is the sampling interval and is often specified as 1.

ftperiodogram(*H*) returns a real vector of length *n*/2 containing the one-sided periodogram of *H* (length *n*), calculated as

$$|H(f)|^2 + |H(-f)|^2$$

excluding frequency 0. Thus ftperiodogram(*H*)[1] corresponds to frequency 1 (-1), ftperiodogram(*H*)[2] to frequency 2 (-2), and so on.

Warnings

invfft(*H*) will cast the result down to real if possible. Code C(invfft(*H*)) if you want to be assured of the result being stored as complex.

convolve(*r*, *s*) is not the same as convolve(*s*, *r*).

convolve(*r*, *s*) will cast the result down to real if possible. Code C(convolve(*r*, *s*)) if you want to be assured of the result being stored as complex.

For convolve(*r*, *s*), the response function *r* must have odd length.

Conformability

fft(*h*):

h:	$1 \times n$	or	$n \times 1$,	*n* a power of 2
result:	$1 \times n$	or	$n \times 1$	

invfft(*H*):

H:	$1 \times n$	or	$n \times 1$,	*n* a power of 2
result:	$1 \times n$	or	$n \times 1$	

_fft(*h*):

h:	$1 \times n$	or	$n \times 1$,	*n* a power of 2
result:	*void*			

_invfft(*H*):

H:	$1 \times n$	or	$n \times 1$,	*n* a power of 2
result:	*void*			

convolve(*r*, *s*):

r:	$1 \times n$	or	$n \times 1$,	$n > 0$, *n* odd
s:	$1 \times 2k + 1$	or	$2k + 1 \times 1$,	i.e., *s* of odd length
result:	$1 \times 2k + n$	or	$2k + n \times 1$	

`deconvolve(`*r*`,` *sm*`)`:

r:	$1 \times n$	or	$n \times 1$,	$n > 0$, n odd
sm:	$1 \times 2k + n$	or	$2k + n \times 1$	
result:	$1 \times 2k + 1$	or	$2k + 1 \times 1$	

`Corr(`*g*`,` *h*`,` *k*`)`:

g:	$1 \times n$	or	$n \times 1$,	$n > 0$
h:	$1 \times n$	or	$n \times 1$	
k:	1×1	or	1×1,	$k > 0$
result:	$1 \times 2k + 1$	or	$2k + 1 \times 1$	

`ftperiodogram(`*H*`)`:

H:	$1 \times n$	or	$n \times 1$,	n even
result:	$n/2 \times 1$	or	$1 \times n/2$	

`ftpad(`*h*`)`:

h:	$1 \times n$	or	$n \times 1$	
result:	$1 \times N$	or	$N \times 1$,	$N = n$ rounded up to power of 2

`ftwrap(`*r*`,` *n*`)`:

r:	$1 \times m$	or	$m \times 1$,	$m > 0$, m odd
n:	1×1	or	1×1,	$n \geq m$
result:	$1 \times n$	or	$n \times 1$	

`ftunwrap(`*H*`)`:

H:	$1 \times n$	or	$n \times 1$	
result:	$1 \times n$	or	$n \times 1$	

`ftretime(`*r*`,` *s*`)`:

r:	$1 \times n$	or	$n \times 1$,	$n > 0$, n odd
s:	$1 \times 2k + 1$	or	$2k + 1 \times 1$,	i.e., *s* of odd length
result:	$1 \times 2k + n$	or	$2k + n \times 1$	

`ftfreqs(`*H*`,` *delta*`)`:

H:	$1 \times n$	or	$n \times 1$,	n even
delta:	1×1			
result:	$1 \times n$	or	$n \times 1$	

Diagnostics

All functions abort with error if the conformability requirements are not met. This is always true, of course, but pay particular attention to the requirements outlined under *Conformability* directly above.

`fft(`*h*`)`, `_fft(`*h*`)`, `invfft(`*H*`)`, `_invfft(`*H*`)`, `convolve(`*r*`,` *s*`)`, `deconvolve(`*r*`,` *sm*`)`, and `Corr(`*g*`,` *h*`,` *k*`)` return missing results if any argument contains missing values.

`ftwrap(`*r*`,` *n*`)` aborts with error if *n* contains missing value.

Also see

[M-4] **mathematical** — Important mathematical functions