Module 29.3: nag_tsa_spectral
Time Series Spectral Analysis

`nag_tsa_spectral` calculates the smoothed sample spectrum of a univariate and bivariate time series.

Contents

Introduction

This module is concerned with the calculation of the time series spectral analysis. It contains procedures that calculate the smoothed sample spectrum of a univariate and bivariate time series.

1 Univariate time series

A standard tool for examining the structure of a time series, $x_1, x_2, \ldots, x_n$, is the auto-covariance function, $\mathrm{Cov}(X_t, X_{t+k}) = \mathrm{Cov}(X_{t+k}, X_t)$ estimated by the sample auto-covariance function, $C_x(k)$, with $C_x(k) = C_x(0)\rho_k$, where ρ_k is the sample autocorrelation function (acf) and $C_k(0)$ is the sample variance. These functions examine the series in terms of the relationship between x_t and x_{x+t}, in the time domain. An alternative approach is in the frequency domain in which the time series is considered as a the sum of components of different frequencies. The spectral density function or spectrum, $f(\omega)$ is such that $f(\omega)\delta\omega$ represents the contribution to the variance of the series of components of frequencies in the range $(\omega, \omega + \delta\omega)$.

The sample spectrum is defined as

$$f^*(\omega) = \frac{1}{2\pi}\left|\sum_{t=1}^{n} x_t e^{i\omega t}\right|^2$$

for frequency values $\omega_i = \dfrac{2\pi j}{n}, \quad j = 0, 1, \ldots, [n/2]$. Note that different authors may use different scaling factors. This may also be defined in terms of the sample auto-covariance function given by

$$f^*(\omega) = \frac{1}{2\pi}\left(C_x(0) + 2\sum_{k=1}^{n-1} w_k C_x(k)\cos(\omega k)\right).$$

The sample spectrum is a poor estimator of the theoretical spectrum $f(\omega)$ since $f^*(\omega_i)$ does not have variance of order $1/n$ but of order $f^2(\omega)$, and both $f^*(\omega_i)$ and $f^*(\omega_j)$ are independent if ω_i and ω_j are a multiple of $2\pi/n$ apart. Thus $f^*(\omega)$ can fluctuate violently. In order to produce a reasonable estimate, $\hat{f}(\omega)$, the sample spectrum, has to be smoothed. This can be achieved in two ways.

1. Direct smoothing of the sample spectrum.
2. Weighting the auto-covariance function.

When directly smoothing the sample spectrum the smoothed sample spectrum at ω_j is given by

$$\hat{f}(\omega_j) = \frac{\sum_k w_{kj} f^*(\omega_k)}{\sum_k w_{kj}}$$

for a subset of $\omega_j = \dfrac{\pi j}{l}, \quad j = 0, 1, \ldots, l \leq [n/2]$
The weights are given by the trapezium window which is nonzero on $(-1/m, 1/m)$

$$\begin{array}{ll} w_{kj} = 1, & |p| \leq \tau/m \\ w_{kj} = \frac{1-|p|m}{1-\tau}, & \tau/m < |p| \leq 1/m \\ w_{kj} = 0, & \text{otherwise} \end{array}$$

with

$$p = \frac{k-j}{n/2}$$

where the parameters m and τ define the window width and shape. The value $\tau = 1$ gives a rectangular window and $\tau = 0$ gives a triangular window. The smaller the value of m the greater the smoothing. The bandwidth of the window, b depends on p and gives a method of comparing different windows with the rectangular window for which $b = 2\pi/m$. Also, frequencies much greater than b may be considered

independent. This approach was originally proposed by Daniell and is sometimes known as a Daniell window.

The second approach involves applying a weights to the auto-covariances to give

$$\hat{f}(\omega) = \frac{1}{2\pi}\left(C_x(k) + 2\sum_{k=1}^{m-1} w_k C_x(k)\cos(\omega k)\right).$$

Again m represents the width of the smoothing window and there are four common windows and associated weights:

rectangular:

$$w_k = 1, \quad k = 0, 1, \ldots, m$$

Bartlett:

$$w_k = 1 - \left(\frac{k}{m}\right), \quad k = 0, 1, \ldots, m$$

Tukey:

$$w_k = \frac{1}{2}\left(1 + \cos\left(\frac{k}{m}\right)\right), \quad k = 0, 1, \ldots, m$$

Parzen:

$$w_k = 1 - 6\left(\frac{k}{m}\right)^2 + 6\left(\frac{k}{m}\right)^3, \quad k = 0, 1, \ldots, m/2$$

$$w_k = 2\left(1 - \left(\frac{k}{m}\right)\right)^3, \qquad k = m/2, m/2+1, \ldots, m.$$

The bandwidth, b, for the above windows are the bandwidth of the corresponding rectangular window in the direct smoothing approach and is approximately inversely proportional to m.

Using either approach the sampling distribution of $\hat{f}(\omega)$ is approximately that of a scaled χ^2_d variate, where the degrees of freedom d depend on the number of observations and the window used. The χ^2_d can be used to compute with multiplying limits γ_u, γ_l from which approximate 95% confidence intervals for the true spectrum $f(\omega)$ may be constructed as $[\gamma_l \hat{f}(\omega), \gamma_u \hat{f}(\omega)]$. Alternatively, $\log \hat{f}(\omega)$ may be returned, with additive limits.

The choice of m, or equivalently bandwidth, is a key aspect of spectral analysis. As m gets smaller the spectral density estimate gets smoother and the variance decreases, but the bias gets larger.

2 Bivariate time series

The univariate correlation and covariance functions can be extended to the cross-correlations and cross-covariances between two series, x and y with sample cross-covariance function $C_{xy}(k)$. However, for the cross-covariance function $\mathrm{Cov}(X_t, Y_{t+k}) \neq \mathrm{Cov}(X_{t+k}, Y_t)$ so both $C_{xy}(k)$, in which series y leads series x, and $C_{yx}(k)$, in which series x leads series y, are needed for a full description of the relationship between the two series. Similarly the univariate spectrum can also be extended to a cross-spectrum $f_{xy}(\omega)$, but this is now a complex function with real and imaginary parts.

The sample cross spectrum is a complex valued function of frequency ω,

$$f^*_{xy}(\omega) = c(\omega) + iq(\omega) = \frac{1}{2\pi n}\left\{\sum_{t=1}^{n} y_t \exp(i\omega t)\right\} \times \left\{\sum_{t=1}^{n} x_t \exp(-i\omega t)\right\}$$

for frequency values $\omega_j = \frac{2\pi j}{n}, j = 0, 1, \ldots, [n/2]$ with the real part, $c(\omega)$ known as the co-spectrum and the imaginary part, $q(\omega)$, known as the quadrature spectrum. The co-spectrum and quadrature spectrum may also be defined in terms of the cross-covariance function,

$$c(\omega) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} C_{xy}(k) \cos(\omega k)$$

and

$$q(\omega) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} C_{xy}(k) \sin(\omega k).$$

As in the univariate case the sample spectrum needs to be smoothed to provide a consistent estimator of $f_{xy}(\omega)$ and either direct smoothing or cross-covariance weighting can be used.

There are other representations of $f_{xy}(\omega)$ which can be useful in examining the series.

1. Amplitude and phase

   Using the polar representation of a complex number gives,

   $$f_{xy}(\omega) = \alpha_{xy} e^{i\phi_{xy}\omega},$$

   where $\alpha_{xy}(\omega) = \sqrt{(c^2(\omega) + q^2(\omega))}$ is the cross-amplitude spectrum and $\phi_{xy}(\omega) = \tan^{-1}(-q(\omega)/c(\omega))$ is the phase spectrum.

2. Squared coherency

   $$W(\omega) = \frac{c^2(\omega) + q^2(\omega)}{f_x(\omega) f_y(\omega)} = \frac{\alpha_{xy}^2}{f_x(\omega) f_y(\omega)}.$$

   This gives the linear correlation between the two series and is analogous to the square of the correlation coefficient between two variables.

3. Gain

   The gain spectra given by

   $$G_{xy} = \sqrt{(f_y(\omega) C(\omega)/f_x(\omega))} = \alpha_{xy}(\omega)/f_x(\omega).$$

   This is basically the regression coefficients of y_t on x_t.

To obtain a full picture of the relationship between the two series usually three of the above need to be examined. The most suitable three will depend on the situation but will generally consist of the squared coherency along with either the co-spectrum and quadrature spectrum, the cross-amplitude and phase spectra or the gain and phase spectra. The squared coherency spectrum is most useful when the two series are considered to be on an equal footing, as in correlation analysis, but it also gives an indication of the strength of the relationship. The gain and phase spectra are most useful when one series is considered to be dependent on the other as in linear regression.

The above derived quantities can be computes from the cross-spectrum and the individual spectra in the obvious way. However, there is a problem with bias in the case of the squared coherency. To reduce the bias an alignment shift, s, between the two series can be used. This is usually chosen so that the cross-correlation function has its largest value at lag s. The cross-spectrum is then computed by shifting either the cross-covariances by an amount s in computing the smoothed cross-spectrum

$$c(\omega) = \frac{1}{2\pi} \sum_{k=-M+1}^{M-1} w_k C_{xy}(k + S) \cos(\omega k)$$

and

$$q(\omega) = \frac{1}{2\pi} \sum_{k=-M+1}^{M-1} w_k C_{xy}(k+S) \sin(\omega k)$$

or by scaling the weights when directly smoothing the spectrum.

The significance of the squared coherency can be tested by comparing $W(\omega)$ with the quantity T derived form the upper percentage point of the F-distribution on $(2, d-2)$ degrees of freedom:

$$T = \frac{2F}{d-2+2F}$$

where d is the degrees of freedom associated with the univariate spectrum estimates. Tests at two frequencies separated by more than the bandwidth may be taken to be independent. Since $\sqrt{W(\omega)}$ can be considered as a correlation coefficient, Fisher's transformation can be used to compute confidence intervals for $W(\omega)$ using the approximation that $\tanh^{-1}\left(\sqrt{W(\omega)}\right)$ is Normal with variance $1/d$. These confidence intervals and the confidence intervals given below are only appropriate at values of ω at which $W(\omega)$ is significant.

Confidence limits for the cross amplitude estimate, $\hat{\alpha}_{xy}(\omega)$, are computed using the approximately that $\log(\hat{\alpha}_{xy}(\omega))$ has a normal distribution with variance $(W(\omega)^{-1}+1)/d$. except that a negative lower limit is reset to 0.0, in which case the approximation is rather poor.

Confidence for both gain and phase are based on the assumption that both $\log(\hat{G}(\omega))$ and $\hat{\phi}(\omega)$ are Normal with variance

$$\frac{1}{d}\left(\frac{1}{W(\omega)} - 1\right).$$

Although the estimate of $\phi(\omega)$ is always given in the range $[0, 2\pi]$, no attempt is made to restrict its confidence limits to this range.

3 Linear Systems

A linear system in continuous time can be written as

$$y(t) = \int_{-\infty}^{\infty} h(u)x(t-u)du,$$

where $h(u)$ is known as the impulse response function. In discrete time the model can be written as

$$y_t = \sum_{k=-\infty}^{\infty} h_k x_{t-k}.$$

Information on the form of the impulse response function can be obtained from the phase and gain spectra, see Jenkins and Watts [1].

In practice there may be additional noise in the system, so that, in the discrete case, the dependence of y_t on x_t can be assumed to be represented by

$$y_t = h_0 x_t + h_1 x_{t-1} + \ldots + n_t$$

where the noise, n_t, is independent of x_t, The spectrum of this noise can be estimated by

$$f_{y|x}(\omega) = f_{yy}(\omega)(1 - W(\omega))$$

Confidence intervals for $f_{y|x}(\omega)$ can be compute as for the univariate spectrum except that there are $d-2$ degrees of freedom.

The impulse response function $\ldots h_{-1}, h_0, h_1, h_2, \ldots$ may be estimated by

$$h_k = \frac{1}{\pi}\int_0^{\pi} \mathrm{Re}\left(\frac{\exp(ik\omega)f_{xy}(\omega)}{f_{xx}(\omega)}\right) d\omega$$

where Re indicates the real part of the expression.

Clearly since y_t cannot depend on future events so the coefficients $h_{-1}, h_{-2}, \ldots$, known as the anticipatory responses, should be zero. Non-zero values indicates that there is feedback from y_t to x_t. This will bias the estimates of $h_0, h_1, \ldots$.

Procedure: nag_spectral_data

1 Description

nag_spectral_data calculates the smoothed sample spectrum of a univariate time series using one of six lag windows – rectangular, Bartlett, Tukey, Parzen, Daniell (trapezium frequency) or no window. The user must supply the time series data points x_i, for $i = 1, 2, \ldots, n$.

2 Usage

```
USE nag_tsa_spectral
CALL nag_spectral_data(method, ts, m, f  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array **x** must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 1$ — the number of values in the time series

$1 < l \leq n$ — the number of spectral estimates

$1 \leq m \leq n$ — the width of the smoothing window

$1 \leq k \leq n$ — the number of output covariance elements

3.1 Mandatory Arguments

method — character(len=1), intent(in)

Input: indicates the choice of the lag window.

If method = 'r' or 'R', a rectangular lag window is used;

if method = 'b' or 'B', a Bartlett lag window is used;

if method = 't' or 'T', a Tukey lag window is used;

if method = 'p' or 'P', a Parzen lag window is used;

if method = 'd' or 'D', a Daniell lag window is used;

if method = 'n' or 'N', no lag window is used and the unsmoothed sample spectrum is returned.

ts(n) — real(kind=*wp*), intent(in)

Input: ts(i) must contain the ith data point of the time series.

m — integer, intent(in)

Input: specifies the width of the smoothing window, m, relative to any alignment shift that has been applied.

Constraints:

if method = 'r', 'R', 'b', 'B', 't', 'T', 'p' or 'P', then $1 \leq$ m $< l$;

if method = 'd' or 'D', then $1 \leq$ m $\leq n$;

if method = 'n' or 'N', m is not used and will be ignored.

f(*l*) — real(kind=*wp*), intent(out)

Output: the spectral estimates, $\hat{f}(\omega_i)$, for $i = 1, 2, \ldots, l$ (logged or unlogged, according to the optional argument `logged`).

Note: the number of spectral estimates computed, l, gives the frequency division of the spectral estimates as $\pi/(l-1)$. For small and medium sized data sets l will often be chosen to be close to n. However, for the covariance based methods (`method` = `'r'`, `'R'`, `'b'`, `'B'`, `'t'`, `'T'` ,`'p'` or `'P'`) l also determines the order of the FFT used to compute the sample spectrum from the covariances. So for efficiency l should be chosen such that $l-1$ is the product of small primes. For the direct methods (`method` = `'d'`, `'D'`, `'r'`, `'R'`, `'n'` or `'N'`) the size of l constrains the order of the FFT applied to the data, see the optional argument `ne`.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

shape — real(kind=*wp*), intent(in), optional

Input: the shape parameter, p, of the trapezium frequency Daniell window. A value of 0.0 gives a triangular window, and a value of 1.0 a rectangular window.

Constraints: $0.0 \leq$ `shape` ≤ 1.0 and `shape` can only be supplied if `method` = `'d'` or `'D'`.

Default: `shape` = 1.0.

correc — character(len=1), intent(in), optional

Input: specifies the choice of the data correction:

if `correc` = `'n'` or `'N'`, no correction is used;

if `correc` = `'m'` or `'M'`, a mean correction is used;

if `correc` = `'t'` or `'T'`, a trend correction is used.

Default: `correc` = `'n'` or `'N'`.

tap — real(kind=*wp*), intent(in), optional

Input: specifies the proportion of the data, h, (totalled over both ends) to be initially tapered by the split cosine bell taper. A value of 0.0 implies no tapering.

Constraints: $0.0 \leq$ `tap` ≤ 1.0.

Default: `tap` = 0.0.

ne — integer, intent(in), optional

Input: the length, n_e, of the extended series (data plus zeros) and hence the order of the initial FFT. For efficiency `ne` should be a product of small primes.

Constraints:

If `cov` is present, `ne` $\geq n + \max(k, m)$;

if `method` = `'d'`, `'D'`, `'n'` or `'N'`, `ne` $\geq n$ and `ne` must be a multiple of $(l-1)$;

otherwise, `ne` $\geq n + m$.

Default:

If `cov` is present, `ne` $= n + \max(m, k)$;

if `method` = `'d'`, `'D'`, `'n'` or `'N'`, `ne` $= 2\kappa(l-1)$, where $2(\kappa-1)(l-1) < n \leq 2\kappa(l-1)$;

otherwise, `ne` $= n + m$.

logged — logical, intent(in), optional

Input: indicates whether logged or unlogged spectral estimates and confidence limits are required.

Default: `logged` = `.false.`.

stats(4) — real(kind=*wp*), intent(out), optional

Output: the four associated statistics. These are the degrees of freedom in stats(1), the lower and upper 95% confidence limit factors in stats(2) and stats(3) respectively (logged if logged = .true.), and the bandwidth in stats(4).

cov(*k*) — real(kind=*wp*), intent(out), optional

Output: the calculated autocovariances for lags $0, 1, \ldots, k-1$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The calculation of confidence limit factors has failed. Spectral estimates (logged if requested) are returned in f. The degrees of freedom and bandwidth are returned in stats (if present).

Warnings (error%level = 1):

error%code	Description
101	m will be ignored. The width of the smoothing window, m, is not required for method for method = 'n' or 'N'.
102	One or more spectral estimates are negative. Unlogged spectral estimates are returned in f. The degrees of freedom, unlogged confidence limit factors and bandwidth are returned in stats (if present).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The computation of the spectral estimates makes use of a discrete Fourier transform or FFT which is computed using the procedure `nag_fft_1d_real`.

For a series $z_j, j = 0, 1, \ldots, n_t$ the FFT computes

$$a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_i e^{-2\pi j(k/n)i}$$

for $k = 0, 1, \ldots, n_t/2$ where $i = \sqrt{-1}$. The computation of an FFT is fastest if the length of the series only has prime factors 2, 3 or 5 and is slowest if the length of the series has large prime factors. In order to make the computation of the FFT more efficient the series can be extended by adding a number of zero points to the series to give an extended series the length of which only has low prime factors.

The FTT is applied to the time series which has been mean or trend corrected as appropriate and has had zeros added to give a series of length n_e For efficiency n_e should only have low prime factors. In addition to mean or trend correcting the data it can be advisable to taper the series to avoid discontinuity at the end of the actual series. The split cosine bell is used for tapering:

$$\begin{array}{ll} \frac{1}{2}(1 - \cos(\pi(t - \frac{1}{2})/T)), & 1 \le t \le T \\ \frac{1}{2}(1 - \cos(\pi(n - t + \frac{1}{2})/T)), & n + 1 - T \le t \le n \\ 1, & \text{otherwise} \end{array}$$

where $T = \left[\frac{nh}{2}\right]$ and h is the tapering proportion.

Direct smoothing of the sample spectrum

From the coefficients computed by the FFT, (a_k, b_k) for $k = 1, 2, \ldots, [n_e/2]$, the sample spectrum is computed as $f_i^* = a_k^2 + b_k^2$ and then sampled at points $j = 1, 2, \ldots, l$ where $j = (l/n_e)i$. If smoothing is used this is then computed from the sample spectrum using the weights as given above. These are applied to the n_e values of f_i^* to give l equidistant smoothed values. In either case l has to be a factor of n_e so by default n_e is chosen such that n_e is the lowest multiple of $l \ge n$.

Computing the autocovariance function

The autocovariance function is computed using a FFT. First a FFT is applied to the spectral mean or trend corrected time series with at least n_c added zeros, where n_c is the number of autocovariances to be computed. This stops circular autocorrelation coefficients being computed rather than non-circular as are normally used. The covariances are then computed by applying a second FFT to the series $f_i^*, i = 1, 2, \ldots, n_e$, where $f_{n_e - i}^* = f_i^*$, and scaling the resulting real coefficients.

Transforming the autocovariances

Given the autocovariances, computed internally as described above, the smoothed spectrum is computed by applying a FFT of length $2(l - 1)$ to a series formed from the autocovariances

$$w_0 C_1, w_1 C_2, \ldots, w_{m-1} C_{m-1}, 0, \ldots, 0, w_{m-1} C_{m-1}, w_{m-2} C_{m-2}, \ldots, w_1 C_1$$

and scaling the resulting $l > m$ real coefficients. For efficiency l should be such that $(l - 1)$ has small prime factors.

6.2 Accuracy

The FFT is a numerically stable process, and any errors introduced during the computation will normally be insignificant compared with uncertainty in the data.

Procedure: nag_spectral_cov

1 Description

nag_spectral_cov calculates the smoothed sample spectrum of a univariate time series using one of four lag windows – rectangular, Bartlett, Tukey or Parzen. The user must supply C, the autocovariances of the data.

2 Usage

```
USE nag_tsa_spectral
CALL nag_spectral_cov(method, cov, f  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array **x** must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$l > 1$ — the number of spectral estimates

$1 \leq m < l$ — the width of the smoothing window

3.1 Mandatory Arguments

method — character(len=1), intent(in)

Input: indicates the choice of the lag window.

If method = 'r' or 'R', a rectangular lag window is used;

if method = 'b' or 'B', a Bartlett lag window is used;

if method = 't' or 'T', a Tukey lag window is used;

if method = 'p' or 'P', a Parzen lag window is used.

cov(m) — real(kind=*wp*), intent(in)

Input: indicates the autocovariances for lags, C.

f(l) — real(kind=*wp*), intent(out)

Output: the spectral estimates, $\hat{f}(\omega_i)$, for $i = 1, 2, \ldots, l$ (logged or unlogged, according to the optional argument logged).

Note: the number of spectral estimates computed, l, gives the frequency division of the spectral estimates as $\pi/(l-1)$. For small and medium sized data sets l will often be chosen to be close to n (the number of values in the original time series, see the optional argument n). However, l also determines the order of the FFT used to compute the sample spectrum from the covariances. So for efficiency l should be chosen such that $l-1$ is the product of small primes.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

tap — real(kind=*wp*), intent(in), optional

Input: if the supplied covariances were calculated using `nag_spectral_data`, `tap` specifies the proportion of the data tapered. A value of 0.0 implies no tapering.

Constraints: $0.0 \leq$ `tap` ≤ 1.0.

Default: `tap` = 0.0.

logged — logical, intent(in), optional

Input: indicates whether logged or unlogged spectral estimates and confidence limits are required.

Default: `logged` = `.false.`.

n — integer, intent(in), optional

Input: the number of values in the original time series.

Constraints: `n` $\geq m$.

stats(4) — real(kind=*wp*), intent(out), optional

Output: the four associated statistics. These are the degrees of freedom in `stats`(1), the lower and upper 95% confidence limit factors in `stats`(2) and `stats`(3) respectively (logged if `logged` = `.true.`), and the bandwidth in `stats`(4).

Constraints: `n` must be present if `stats` is present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The calculation of confidence limit factors has failed. Spectral estimates (logged if requested) are returned in `f`. The degrees of freedom and bandwidth are returned in `stats` (if present).

Warnings (error%level = 1):

error%code	Description
101	`n` will be ignored. `n` is not required if `stats` is not present.
102	One or more spectral estimates are negative. Unlogged spectral estimates are returned in `f`. The degrees of freedom, unlogged confidence limit factors and bandwidth are returned in `stats` (if present).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

Transforming the autocovariances

Given the autocovariances, the smoothed spectrum is computed by applying a FFT of length $2(l-1)$ to a series formed from the autocovariances

$$w_0C_1, w_1C_2, \ldots, w_{m-1}C_{m-1}, 0, \ldots, 0, w_{m-1}C_{m-1}, w_{m-2}C_{m-2}, \ldots, w_1C_1$$

and scaling the resulting $l > m$ real coefficients. For efficiency l should be such that $(l-1)$ has small prime factors.

6.2 Accuracy

The FFT is a numerically stable process, and any errors introduced during the computation will normally be insignificant compared with uncertainty in the data.

Procedure: nag_bivar_spectral_data

1 Description

nag_bivar_spectral_data calculates the smoothed sample cross spectrum of a bivariate time series using one of six lag windows – rectangular, Bartlett, Tukey, Parzen, Daniell (trapezium frequency) or none window. The user must supply the time series data points x_i and y_i, for $i = 1, 2, \ldots, n$.

2 Usage

```
USE nag_tsa_spectral
CALL nag_bivar_spectral_data(method, x_ts, y_ts, m, f_xy  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array **x** must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 1$ — the number of values in the time series x and y

$1 < l \leq n$ — the number of complex spectral estimates

$1 \leq m \leq n$ — the width of the smoothing window

$m + \alpha \leq k \leq n$ — the number of output cross covariance elements, where $\alpha = 0$, if optional argument **s** is not present and $\alpha =$ ABS(**s**), if **s** is present

3.1 Mandatory Arguments

method — character(len=1), intent(in)

Input: indicates the choice of the lag window.

If method = 'r' or 'R', a rectangular lag window is used;

if method = 'b' or 'B', a Bartlett lag window is used;

if method = 't' or 'T', a Tukey lag window is used;

if method = 'p' or 'P', a Parzen lag window is used;

if method = 'd' or 'D', a Daniell lag window is used;

if method = 'n' or 'N', no lag window is used, the unsmoothed sample spectrum is returned.

x_ts(n) — real(kind=*wp*), intent(in)

Input: x_ts(i) contains the i data point of the x time series.

y_ts(n) — real(kind=*wp*), intent(in)

Input: y_ts(i) contains the i data point of the y time series.

m — integer, intent(in)

Input: specifies the width of the smoothing window, m, relative to any alignment shift that has been applied.

Constraints:

if method = 'r', 'R', 'b', 'B', 't', 'T', 'p' or 'P', then $1 \leq$ m $<$ l;

if method = 'd' or 'D', then $1 \leq$ m $\leq n$;

if method = 'n' or 'N', m is not used and will be ignored.

f_xy(*l*) — complex(kind=*wp*), intent(out)

Output: the complex spectral estimates, $f_{xy}(w)$.

Note: the number of spectral estimates computed, l, gives the frequency division of the spectral estimates as $\pi/(l-1)$. For small and medium sized data sets l will often be chosen to be close to n. However, for the covariance based methods (`method` = `'r'`, `'R'`, `'b'`, `'B'`, `'t'`, `'T'` ,`'p'` or `'P'`) l also determines the order of the FFT used to compute the sample spectrum from the covariances. So for efficiency l should be chosen such that $l-1$ is the product of small primes. For the direct methods (`method` = `'d'`, `'D'`, `'r'`, `'R'`, `'n'` or `'N'`) the size of l constrains the order of the FFT applied to the data, see the optional argument `ne`.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

s — integer, intent(in), optional

Input: specifies the alignment shift, S, between the x and y series. If x leads y, the shift is positive.

Constraints:

if `method` = `'n'` or `'N'`, `s` is not used;

if `method` = `'d'` or `'D'` and `m` = n=(size(`x_ts`)), `s` is not used;

if `method` = `'d'` or `'D'` and `m` $\neq n$, then $-2(l-1) <$ `s` $< 2(l-1)$;

otherwise,$-$`m` $<$ `s` $<$ `m`.

Default: `s` = 0.

shape — real(kind=*wp*), intent(in), optional

Input: the shape parameter, p, of the trapezium frequency Daniell window. A value of 0.0 gives a triangular window, and a value of 1.0 a rectangular window.

Constraints: $0.0 \leq$ `shape` ≤ 1.0 and `shape` can only be supplied if `method` = `'d'` or `'D'`.

Default: `shape` = 1.0.

correc — character(len=1), intent(in), optional

Input: specifies the choice of the data correction:

if `correc` = `'n'` or `'N'`, no correction is used;

if `correc` = `'m'` or `'M'`, a mean correction is used;

if `correc` = `'t'` or `'T'`, a trend correction is used.

Default: `correc` = `'n'` or `'N'`.

tap — real(kind=*wp*), intent(in), optional

Input: specifies the proportion of the data, h, (totalled over both ends) to be initially tapered by the split cosine bell taper. A value of 0.0 implies no tapering.

Constraints: $0.0 \leq$ `tap` ≤ 1.0.

Default: `tap` = 0.0.

ne — integer, intent(in), optional

Input: the length, n_e, of the extended series (data plus zeros) and hence the order of the initial FFT. For efficiency `ne` should be a product of small primes.

Constraints:

if `xy_cov` or `yx_cov` is present, `ne` $\geq n + \max(k, m)$;

if `method` = `'d'`, `'D'`, `'n'` or `'N'`, `ne` $\geq n$ and `ne` must be a multiple of $(l-1)$ or $(2l-1)$;

otherwise, `ne` $\geq n + m$.

Default:

if xy_cov or yx_cov is present, ne $= n + \max(m, k)$;
if method = 'd', 'D', 'n' or 'N', ne $= 2\kappa(l-1)$, where $2(\kappa-1)(l-1) < n \leq 2\kappa(l-1)$;
otherwise, ne $= n + m$.

xy_cov(k) — real(kind=*wp*), intent(out), optional

Output: the calculated cross autocovariances between values in the y series and earlier values in time in the x series for lags $0, 1, \ldots, k-1$.

yx_cov(k) — real(kind=*wp*), intent(out), optional

Output: the calculated cross autocovariances between values in the y series and later values in time in the x series for lags $0, 1, \ldots, k-1$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	m will be ignored The width of the smoothing window, m, is not required for method for method = 'n' or 'N'.
102	The alignment shift between the x and y series is ignored. s is not required, if either (method = 'n' or 'N') or (method = 'd' or 'D' and m = SIZE(x_ts)).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The computation of the cross spectral estimates makes use of a discrete Fourier transform or FFT which is computed using the procedure `nag_fft_1d_real`.

For a series $z_j, j = 0, 1, \ldots, n_t$ the FFT computes

$$a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_i e^{-2\pi j(k/n)i}$$

for $k = 0, 1, \ldots, n_t/2$ where $i = \sqrt{-1}$. The computation of an FFT is fastest if the length of the series only has prime factors 2, 3 or 5 and is slowest if the length of the series has large prime factors. In order to make the computation of the FFT more efficient the series can be extended by adding a number of zero points to the series to give an extended series the length of which only has low prime factors.

The FTT is applied to the bivariate time series which has been mean or trend corrected as appropriate and has had zeros added to give a series of length n_e For efficiency n_e should only have low prime factors. In addition to mean or trend correcting the data it can be advisable to taper the series to avoid discontinuity at the end of the actual series. The split cosine bell is used for tapering:

$$\begin{array}{ll} \frac{1}{2}(1 - \cos(\pi(t - \frac{1}{2})/T)), & 1 \le t \le T \\ \frac{1}{2}(1 - \cos(\pi(n - t + \frac{1}{2})/T)), & n + 1 - T \le t \le n \\ 1, & \text{otherwise} \end{array}$$

where $T = \left[\frac{nh}{2}\right]$ and h is the tapering proportion.

Direct smoothing of the sample spectrum

From the coefficients computed by the FFT from the x and y series, $(a_{xk}, b_{xk}, a_{yk}, b_{yk})$ for $k = 1, 2, \ldots, [n_e/2]$, the sample spectrum is computed as

$$f^*_{xy}(\omega_k) = c(\omega_i) + q(\omega_i) \propto (a_{xk}a_{yk} + b_{xk}b_{yk}) + i(a_{yk}b_{xk} - a_{xk}b_{yk})$$

and then sampled at points $j = 1, 2, \ldots, l$ where $j = (l/n_e)i$ If direct smoothing is used this is then computed from the sample spectrum using the weights as given in the introduction. These are applied to the n_e values of $f^*_{xy}(\omega_i)$ to give l equidistant smoothed values. In either case l has to be a factor of n_e so by default n_e is chosen such that n_e is the lowest multiple of $l \ge n$.

Computing the cross-covariance function

The cross-covariance function can be computed using FFTs. First FFTs are applied to the mean or trend corrected time series with at least n_c added zeros, where n_c is the number of cross-covariances to be computed. This stops circular cross-covariances being computed rather than non-circular as are normally used. The covariances are then computed by applying a second FFT to the series stored as a Hermitian sequence, i.e.,

$$c(\omega_1), c(\omega_2), \ldots c(\omega_{n_e/2}), q(\omega_{n_e/2}), c(\omega_{n_e/2-1}), \ldots q(\omega_1)$$

and scaling the resulting real coefficients.

Transforming the cross-covariances

Given the cross-covariances, the smoothed spectrum is computed by applying a FFT of length $2(l-1)$ to a series formed from the autocovariances

$$w_0C_{xy}(0), w_1C_{xy}(1), \ldots, w_{m-1}C_{xy}(m-1), 0, \ldots, 0, w_{m-1}C_{yx}(m-1), w_{m-2}C_{yx}(m-2), \ldots, w_1C_{yx}(1)$$

and scaling the resulting $l > m$ real and imaginary coefficients. For efficiency l should be such that $(l-1)$ has small prime factors.

6.2 Accuracy

The FFT is a numerically stable process, and any errors introduced during the computation will normally be insignificant compared with uncertainty in the data.

Procedure: nag_bivar_spectral_cov

1 Description

nag_bivar_spectral_cov calculates the smoothed sample cross spectrum of a bivariate time series using one of four lag windows – rectangular, Bartlett, Tukey or Parzen. The user must supply C, the cross autocovariances of the data.

2 Usage

```
USE nag_tsa_spectral
CALL nag_bivar_spectral_cov(method, xy_cov, yx_cov, f_xy  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array **x** must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$l > 1$ — the number of cross spectral estimates
$1 \leq m < l$ — the width of the smoothing window

3.1 Mandatory Arguments

method — character(len=1), intent(in)

Input: indicates the choice of the lag window.

If method = 'r' or 'R', a rectangular lag window is used;
if method = 'b' or 'B', a Bartlett lag window is used;
if method = 't' or 'T', a Tukey lag window is used;
if method = 'p' or 'P', a Parzen lag window is used.

xy_cov(m) — real(kind=*wp*), intent(in)

Input: indicates the cross autocovariances between values in the y series and earlier values in time in the x series for lags, C.

yx_cov(m) — real(kind=*wp*), intent(in)

Input: indicates the cross autocovariances between values in the y series and later values in time in the x series for lags, C.

f_xy(l) — complex(kind=*wp*), intent(out)

Output: the complex spectral estimates, $f_{xy}(w)$.

Note: the number of spectral estimates computed, l, gives the frequency division of the spectral estimates as $\pi/(l-1)$. For small and medium sized data sets l will often be chosen to be close to n (the number of values in the original time series, see the optional argument n). However, l also determines the order of the FFT used to compute the sample spectrum from the covariances. So for efficiency l should be chosen such that $l-1$ is the product of small primes.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

s — integer, intent(in), optional

Input: specifies the alignment shift, S, between the x and y series. If x leads y, the shift is positive.

Constraints: $-m <$ `s` $< m$.

Default: `s` $= 0$.

tap — real(kind=*wp*), intent(in), optional

Input: if the supplied cross autocovariance were computed using `nag_bivar_spectral_data`, `tap` specifies the proportion of the data tapered. A value of 0.0 implies no tapering.

Constraints: $0.0 \leq$ `tap` ≤ 1.0.

Default: `tap` $= 0.0$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The computation of the cross spectral estimates makes use of a discrete Fourier transform or FFT which is computed using the procedure `nag_fft_1d_real`.

Transforming the cross-covariances

Given the cross-covariances, the smoothed spectrum is computed by applying a FFT of length $2(l-1)$ to a series formed from the autocovariances

$$w_0 C_{xy}(0), w_1 C_{xy}(1), \ldots, w_{m-1} C_{xy}(m-1), 0, \ldots, 0, w_{m-1} C_{yx}(m-1), w_{m-2} C_{yx}(m-2), \ldots, w_1 C_{yx}(1)$$

and scaling the resulting $l > m$ real and imaginary coefficients. For efficiency l should be such that $(l-1)$ has small prime factors.

6.2 Accuracy

The FFT is a numerically stable process, and any errors introduced during the computation will normally be insignificant compared with uncertainty in the data.

Procedure: nag_bivar_spectral_coh

1 Description

nag_bivar_spectral_coh calculates the squared coherency, the cross amplitude spectrum, the gain and the phase, together with lower and upper bounds from the univariate and bivariate spectra.

2 Usage

```
USE nag_tsa_spectral
CALL nag_bivar_spectral_coh(f_x, f_y, f_xy, stats, sqrd_coh  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$l \geq 1$ — the number of spectral, cross amplitude spectra, squared coherency, gain and phase estimates

3.1 Mandatory Arguments

f_x(l) — real(kind=*wp*), intent(in)

Input: indicates the univariate spectral estimates, $f_{xx}(w)$, for the x time series.

f_y(l) — real(kind=*wp*), intent(in)

Input: indicates the univariate spectral estimates, $f_{yy}(w)$, for the y time series.

f_xy(l) — complex(kind=*wp*), intent(in)

Input: the complex bivariate spectral estimates, $f_{xy}(w)$, for the x and y series.

Note: the two univariate and the bivariate spectra must been calculated using the same method of smoothing. For rectangular, Bartlett, Tukey or Parzen smoothing windows, the same cut off points of the lag window and the same frequency division of the spectral estimates must be used. For the trapezium frequency width and the shape of the window and the frequency division of the spectral estimates must be the same. The spectral estimates and statistics must also be unlogged.

stats(4) — real(kind=*wp*), intent(in)

Input: the four associated statistics for the univariate spectral estimates for the x and y series. These are the degrees of freedom in stats(1), the lower and upper bound multiplying factors in stats(2) and stats(3) respectively, and the bandwidth in stats(4).

Constraints:

stats(1) ≥ 3.0,

$0.0 <$ stats(2) ≤ 1.0,

stats(3) ≥ 1.0.

sqrd_coh(*l*) — real(kind=*wp*), intent(out)

Output: indicates the squared coherency estimates, $\hat{W}(w)$, at each frequency w.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

sqrd_coh_bounds(*l*, 2) — real(kind=*wp*), intent(out), optional

Output: `sqrd_coh_bounds`(:,1) contains the lower bounds for the squared coherency estimates and `sqrd_coh_bounds`(:,2) contains the upper bounds for the squared coherency estimates.

t — real(kind=*wp*), intent(out), optional

Output: indicates the critical value for the significance of the squared coherency, T.

cross_ampl(*l*) — real(kind=*wp*), intent(out), optional

Output: indicates the cross amplitude spectral estimates, $\hat{A}(w)$, at each frequency of w.

cross_ampl_bounds(*l*, 2) — real(kind=*wp*), intent(out), optional

Output: `cross_ampl_bounds`(:,1) contains the lower bounds for the cross amplitude spectral estimates and `cross_ampl_bounds`(:,2) contains the upper bounds for the cross amplitude spectral estimates.

gain(*l*) — real(kind=*wp*), intent(out), optional

Output: indicates the gain estimates, $\hat{G}(w)$, at each frequency w.

gain_bounds(*l*, 2) — real(kind=*wp*), intent(out), optional

Output: `gain_bounds`(:,1) contains the lower bounds for the gain estimates and `gain_bounds`(:,2) contains the upper bounds for the gain estimates.

phase(*l*) — real(kind=*wp*), intent(out), optional

Output: indicates the phase estimates, $\hat{\phi}(w)$, at each frequency w.

phase_bounds(*l*, 2) — real(kind=*wp*), intent(out), optional

Output: `phase_bounds`(:,1) contains the lower bounds for the phase estimates and `phase_bounds`(:,2) contains the upper bounds for the phase estimates.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.

Failures (error%level = 2):

error%code	Description
201	At least one of the x univariate spectral estimates f_{xx} is negative. For this frequency the cross amplitude spectrum and squared coherency, the gain and the phase and their bounds are set to zero.
202	At least one of the y univariate spectral estimates f_{yy} is negative. For this frequency the cross amplitude spectrum and squared coherency, the gain and the phase and their bounds are set to zero.
203	At least one univariate spectral estimate f_{xx} or f_{yy} is zero. For this frequency the cross amplitude spectrum and squared coherency, the gain and the phase and their bounds are set to zero.
204	At least one bivariate spectral estimate f_{xy} is zero. For this frequency the cross amplitude spectrum and squared coherency, the gain and the phase and their bounds are set to zero.
205	A calculated value of the squared coherency exceeds 1.0. For this frequency the squared coherency is reset to 1.0 and this value for the squared coherency is used in the formulae for the calculation of bounds for the cross amplitude spectrum, the squared coherency, the gain and phase. This has the consequence that both squared coherency bounds are 1.0.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

6 Further Comments

6.1 Accuracy

All computations are very stable and yield good accuracy.

Procedure: nag_bivar_spectral_lin_sys

1 Description

nag_bivar_spectral_lin_sys calculates the noise spectrum together with multiplying factors for the bounds and the impulse response function and its standard error from a linear system.

Note: all the output arguments of this procedure are optional. However, at least one output argument must be present in every call statement.

2 Usage

```
USE nag_tsa_spectral
CALL nag_bivar_spectral_lin_sys(f_x, f_y, f_xy, stats, n  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$l \geq 1$ — the number of spectral and noise spectral estimates

3.1 Mandatory Arguments

f_x(l) — real(kind=*wp*), intent(in)

Input: indicates the univariate spectral estimates, $f_{xx}(w)$, for the x time series.

f_y(l) — real(kind=*wp*), intent(in)

Input: indicates the univariate spectral estimates, $f_{yy}(w)$, for the y time series.

f_xy(l) — complex(kind=*wp*), intent(in)

Input: the complex bivariate spectral estimates, $f_{xy}(w)$, for the x and y series.

Note: the two univariate and the bivariate spectra must been calculated using the same method of smoothing. For rectangular, Bartlett, Tukey or Parzen smoothing windows, the same cut off points of the lag window and the same frequency division of the spectral estimates must be used. For the trapezium frequency width and the shape of the window and the frequency division of the spectral estimates must be the same. The spectral estimates and statistics must also be unlogged.

stats(4) — real(kind=*wp*), intent(in)

Input: the four associated statistics for the univariate spectral estimates for the x and y series. These are the degrees of freedom in stats(1), the lower and upper bound multiplying factors in stats(2) and stats(3) respectively, and the bandwidth in stats(4).

Constraints:

stats(1) ≥ 3.0,

$0.0 <$ stats(2) ≤ 1.0,

stats(3) ≥ 1.0.

n — integer, intent(in)

Input: the number of points in each of the time series x and y. `n` should have the same values as l in the call of `nag_bivar_spectral_data` or `nag_bivar_spectral_cov` which calculated the smoothed sample cross spectrum. `n` is used in calculating the impulse response function standard error (`rfse`).

Constraints: `n` ≥ 1.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

ns(l) — real(kind=*wp*), intent(out), optional

Output: indicates the noise spectrum, $\hat{f}_{y/x}(w)$, at each frequency.

ns_bounds($l, 2$) — real(kind=*wp*), intent(out), optional

Output: `ns_bounds`(:,1) contains the noise spectrum lower bounds and `ns_bounds`(:,2) contains the noise spectrum upper bounds.

crf(l) — real(kind=*wp*), intent(out), optional

Output: indicates the causal responses for the impulse response function.

arf($l-1$) — real(kind=*wp*), intent(out), optional

Output: indicates the anticipatory responses for the impulse response function.

rfse — real(kind=*wp*), intent(out), optional

Output: indicates the impulse response function standard error.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.

Failures (error%level = 2):

error%code	Description
201	At least one of the x univariate spectral estimates f_{xx} is negative. For this frequency the noise spectrum is set to zero and the contributions to the impulse response function and its standard error are set to zero.
202	At least one of the y univariate spectral estimates f_{yy} is negative.

For this frequency the noise spectrum is set to zero and the contributions to the impulse response function and its standard error are set to zero.

203 At least one univariate spectral estimate $f_{xx} or f_{yy}$ is zero.

For this frequency the noise spectrum is set to zero and the contributions to the impulse response function and its standard error are set to zero.

204 At least one bivariate spectral estimate f_{xy} is zero.

For this frequency the noise spectrum is set to zero and the contribution to the impulse response function and its standard error is set to zero.

205 A calculated value of the squared coherency exceeds 1.0.

For this frequency the squared coherency is reset to 1.0 and the noise spectrum is zero and the contribution to the impulse response function at this frequency is zero.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

6 Further Comments

6.1 Accuracy

The computation of the noise is stable and yields good accuracy. The FFT is a numerically stable process, and any errors introduced during the computation will normally be insignificant compared with uncertainty in the data.

Example 1: Calculation of the Smoothed Sample Spectrum of a Univariate Time Series

This example program shows how nag_spectral_data is used to calculate the univariate spectrum. The optional argument cov (returned by nag_spectral_data) is then used to show how nag_spectral_cov may be used to calculate the same sample spectrum.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_tsa_spectral_ex01

  ! Example Program Text for nag_tsa_spectral
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_tsa_spectral, ONLY : nag_spectral_data, nag_spectral_cov
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, l, m, n
  CHARACTER (1) :: method
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: cov(:), f(:), ts(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) &
   'Example Program Results for nag_tsa_spectral_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, m, l

  ALLOCATE (ts(n),f(l),cov(m)) ! Allocate storage

  READ (nag_std_in,*) ts
  method = 'p'

  CALL nag_spectral_data(method,ts,m,f,cov=cov)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) ' Results using the time series data'
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,A,I3)') &
   'Frequency width of smoothing window = 1/', m

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) &
   '       Spectrum       Spectrum       Spectrum       Spectrum'
  WRITE (nag_std_out,*) &
   '       estimate       estimate       estimate       estimate'
  WRITE (nag_std_out,'((1X,4(I5,F10.3)))') (i,f(i),i=1,l)

  CALL nag_spectral_cov(method,cov,f)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) ' Results using the time series auto-covariance'
```

```
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) &
       '       Spectrum       Spectrum       Spectrum       Spectrum'
      WRITE (nag_std_out,*) &
       '       estimate       estimate       estimate       estimate'
      WRITE (nag_std_out,'((1X,4(I5,F10.3)))') (i,f(i),i=1,l)

      DEALLOCATE (ts,f,cov)        ! Deallocate storage

    END PROGRAM nag_tsa_spectral_ex01
```

2 Program Data

```
Example Program Data for nag_tsa_spectral_ex01
 131 31 51                                                  : n,m,l
 11.500  9.890  8.728  8.400  8.230  8.365  8.383  8.243
  8.080  8.244  8.490  8.867  9.469  9.786 10.100 10.714
 11.320 11.900 12.390 12.095 11.800 12.400 11.833 12.200
 12.242 11.687 10.883 10.138  8.952  8.443  8.231  8.067
  7.871  7.962  8.217  8.689  8.989  9.450  9.883 10.150
 10.787 11.000 11.133 11.100 11.800 12.250 11.350 11.575
 11.800 11.100 10.300  9.725  9.025  8.048  7.294  7.070
  6.933  7.208  7.617  7.867  8.309  8.640  9.179  9.570
 10.063 10.803 11.547 11.550 11.800 12.200 12.400 12.367
 12.350 12.400 12.270 12.300 11.800 10.794  9.675  8.900
  8.208  8.087  7.763  7.917  8.030  8.212  8.669  9.175
  9.683 10.290 10.400 10.850 11.700 11.900 12.500 12.500
 12.800 12.950 13.050 12.800 12.800 12.800 12.600 11.917
 10.805  9.240  8.777  8.683  8.649  8.547  8.625  8.750
  9.110  9.392  9.787 10.340 10.500 11.233 12.033 12.200
 12.300 12.600 12.800 12.650 12.733 12.700 12.259 11.817
 10.767  9.825  9.150                                      : ts(1:n)
```

3 Program Results

```
 Example Program Results for nag_tsa_spectral_ex01

  Results using the time series data

 Frequency width of smoothing window = 1/ 31

        Spectrum        Spectrum        Spectrum        Spectrum
        estimate        estimate        estimate        estimate
    1   372.438     2   321.103     3   204.187     4    94.109
    5    32.066     6    10.384     7     4.839     8     2.657
    9     1.846    10     1.748    11     1.359    12     0.807
   13     0.530    14     0.435    15     0.380    16     0.380
   17     0.381    18     0.328    19     0.272    20     0.241
   21     0.217    22     0.202    23     0.198    24     0.186
   25     0.165    26     0.152    27     0.143    28     0.137
   29     0.133    30     0.127    31     0.115    32     0.107
   33     0.106    34     0.109    35     0.109    36     0.105
   37     0.098    38     0.091    39     0.089    40     0.088
   41     0.085    42     0.083    43     0.081    44     0.077
   45     0.074    46     0.072    47     0.071    48     0.071
   49     0.071    50     0.070    51     0.069

  Results using the time series auto-covariance

        Spectrum        Spectrum        Spectrum        Spectrum
        estimate        estimate        estimate        estimate
```

```
  1   372.438     2   321.103     3   204.187     4    94.109
  5    32.066     6    10.384     7     4.839     8     2.657
  9     1.846    10     1.748    11     1.359    12     0.807
 13     0.530    14     0.435    15     0.380    16     0.380
 17     0.381    18     0.328    19     0.272    20     0.241
 21     0.217    22     0.202    23     0.198    24     0.186
 25     0.165    26     0.152    27     0.143    28     0.137
 29     0.133    30     0.127    31     0.115    32     0.107
 33     0.106    34     0.109    35     0.109    36     0.105
 37     0.098    38     0.091    39     0.089    40     0.088
 41     0.085    42     0.083    43     0.081    44     0.077
 45     0.074    46     0.072    47     0.071    48     0.071
 49     0.071    50     0.070    51     0.069
```

Example 2: Calculation of the Smoothed Sample Cross Spectrum of a Bivariate Time Series

This example program shows how nag_bivar_spectral_data is used to calculate the bivariate cross spectrum. The optional arguments xy_cov and yx_cov (returned by nag_bivar_spectral_data) is then used to show how nag_bivar_spectral_cov may be used to calculate the same sample cross spectrum.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_tsa_spectral_ex02

  ! Example Program Text for nag_tsa_spectral
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_tsa_spectral, ONLY : nag_bivar_spectral_data, &
   nag_bivar_spectral_cov
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC AIMAG, KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, l, m, n
  CHARACTER (1) :: method
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: xy_cov(:), x_ts(:), yx_cov(:), y_ts(:)
  COMPLEX (wp), ALLOCATABLE :: f_xy(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) &
   'Example Program Results for nag_tsa_spectral_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, m, l

  method = 'R'

  ALLOCATE (x_ts(n),y_ts(n),xy_cov(m),yx_cov(m), &
   f_xy(l))                    ! Allocate storage

  READ (nag_std_in,*) x_ts
  READ (nag_std_in,*) y_ts

  CALL nag_bivar_spectral_data(method,x_ts,y_ts,m,f_xy,xy_cov=xy_cov, &
   yx_cov=yx_cov)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) &
   '    Returned sample spectrum - using nag_bivar_spectral_data'
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) &
   '       Real  Imaginary       Real  Imaginary       Real  Imaginary'
  WRITE (nag_std_out,*) &
   ' Lag   part     part   Lag   part     part   Lag   part     part'
  WRITE (nag_std_out,'((3(I4,2F9.3)))') (i-1,REAL(f_xy(i)),AIMAG(f_xy(i)), &
   i=1,l)
```

```
      CALL nag_bivar_spectral_cov(method,xy_cov,yx_cov,f_xy)

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) &
       '    Returned sample spectrum - using nag_bivar_spectral_cov'
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) &
       '       Real  Imaginary       Real  Imaginary       Real  Imaginary'
      WRITE (nag_std_out,*) &
       ' Lag   part     part   Lag   part     part   Lag   part     part'
      WRITE (nag_std_out,'((3(I4,2F9.4)))') (i-1,REAL(f_xy(i)),AIMAG(f_xy(i)), &
       i=1,l)

      DEALLOCATE (x_ts,y_ts,xy_cov,yx_cov,f_xy) ! Deallocate storage

    END PROGRAM nag_tsa_spectral_ex02
```

2 Program Data

```
Example Program Data for nag_tsa_spectral_ex02
   150 41 50                                          : n,m,l
 -0.109  0.000  0.178  0.339  0.373  0.441  0.461  0.348
  0.127 -0.180 -0.588 -1.055 -1.421 -1.520 -1.302 -0.814
 -0.475 -0.193  0.088  0.435  0.771  0.866  0.875  0.891
  0.987  1.263  1.775  1.976  1.934  1.866  1.832  1.767
  1.608  1.265  0.790  0.360  0.115  0.088  0.331  0.645
  0.960  1.409  2.670  2.834  2.812  2.483  1.929  1.485
  1.214  1.239  1.608  1.905  2.023  1.815  0.535  0.122
  0.009  0.164  0.671  1.019  1.146  1.155  1.112  1.121
  1.223  1.257  1.157  0.913  0.620  0.255 -0.280 -1.080
 -1.551 -1.799 -1.825 -1.456 -0.944 -0.570 -0.431 -0.577
 -0.960 -1.616 -1.875 -1.891 -1.746 -1.474 -1.201 -0.927
 -0.524  0.040  0.788  0.943  0.930  1.006  1.137  1.198
  1.054  0.595 -0.080 -0.314 -0.288 -0.153 -0.109 -0.187
 -0.255 -0.299 -0.007  0.254  0.330  0.102 -0.423 -1.139
 -2.275 -2.594 -2.716 -2.510 -1.790 -1.346 -1.081 -0.910
 -0.876 -0.885 -0.800 -0.544 -0.416 -0.271  0.000  0.403
  0.841  1.285  1.607  1.746  1.683  1.485  0.993  0.648
  0.577  0.577  0.632  0.747  0.999  0.993  0.968  0.790
  0.399 -0.161 -0.553 -0.603 -0.424 -0.194          : x_ts(1:n)
 53.8 53.6 53.5 53.5 53.4 53.1 52.7 52.4 52.2 52.0 52.0
 52.4 53.0 54.0 54.9 56.0 56.8 56.8 56.4 55.7 55.0 54.3
 53.2 52.3 51.6 51.2 50.8 50.5 50.0 49.2 48.4 47.9 47.6
 47.5 47.5 47.6 48.1 49.0 50.0 51.1 51.8 51.9 51.7 51.2
 50.0 48.3 47.0 45.8 45.6 46.0 46.9 47.8 48.2 48.3 47.9
 47.2 47.2 48.1 49.4 50.6 51.5 51.6 51.2 50.5 50.1 49.8
 49.6 49.4 49.3 49.2 49.3 49.7 50.3 51.3 52.8 54.4 56.0
 56.9 57.5 57.3 56.6 56.0 55.4 55.4 56.4 57.2 58.0 58.4
 58.4 58.1 57.7 57.0 56.0 54.7 53.2 52.1 51.6 51.0 50.5
 50.4 51.0 51.8 52.4 53.0 53.4 53.6 53.7 53.8 53.8 53.8
 53.3 53.0 52.9 53.4 54.6 56.4 58.0 59.4 60.2 60.0 59.4
 58.4 57.6 56.9 56.4 56.0 55.7 55.3 55.0 54.4 53.7 52.8
 51.6 50.6 49.4 48.8 48.5 48.7 49.2 49.8 50.4 50.7 50.9
 50.7 50.5 50.4 50.2 50.4 51.2 52.3                 : y_ts(1:n)
```

3 Program Results

```
Example Program Results for nag_tsa_spectral_ex02

    Returned sample spectrum - using nag_bivar_spectral_data

       Real  Imaginary       Real  Imaginary       Real  Imaginary
Lag    part      part   Lag   part      part   Lag   part      part
  0 129.889     0.000    1 29.847    -9.867    2 -21.247  -14.846
  3   9.819     9.148    4 -11.436    2.775    5   6.938   -5.992
  6   0.847     2.442    7  -3.102   -3.512    8   6.386    0.235
  9  -7.236    -0.071   10   3.915   -1.597   11  -1.288    1.671
 12  -1.021    -1.764   13   3.249    0.956   14  -3.930   -0.266
 15   3.462    -0.537   16  -2.078    0.775   17   0.325   -1.178
 18   1.188     1.037   19  -2.265   -0.605   20   2.761    0.068
 21  -2.363     0.405   22   1.337   -0.776   23  -0.052    0.847
 24  -1.183    -0.738   25   1.994    0.426   26  -2.205    0.042
 27   1.780    -0.423   28  -0.911    0.687   29  -0.245   -0.718
 30   1.201     0.555   31  -1.797   -0.264   32   1.862   -0.102
 33  -1.412     0.432   34   0.579   -0.618   35   0.394    0.625
 36  -1.241    -0.465   37   1.694    0.175   38  -1.668    0.168
 39   1.159    -0.450   40  -0.340    0.601   41  -0.583   -0.587
 42   1.309     0.378   43  -1.659   -0.084   44   1.519   -0.238
 45  -0.943     0.494   46   0.103   -0.594   47   0.750    0.527
 48  -1.408    -0.308   49   1.635    0.000

    Returned sample spectrum - using nag_bivar_spectral_cov

       Real  Imaginary       Real  Imaginary       Real  Imaginary
Lag    part      part   Lag   part      part   Lag   part      part
  0 129.8894   0.0000    1 29.8473  -9.8673    2 -21.2468 -14.8464
  3   9.8193   9.1483    4 -11.4359   2.7751   5   6.9385  -5.9925
  6   0.8472   2.4417    7  -3.1024  -3.5116   8   6.3857   0.2352
  9  -7.2365  -0.0707   10   3.9147  -1.5974  11  -1.2882   1.6706
 12  -1.0208  -1.7636   13   3.2489   0.9559  14  -3.9303  -0.2662
 15   3.4621  -0.5368   16  -2.0782   0.7745  17   0.3249  -1.1777
 18   1.1883   1.0368   19  -2.2646  -0.6055  20   2.7612   0.0680
 21  -2.3628   0.4047   22   1.3370  -0.7761  23  -0.0520   0.8470
 24  -1.1835  -0.7378   25   1.9942   0.4256  26  -2.2048   0.0419
 27   1.7805  -0.4232   28  -0.9111   0.6872  29  -0.2449  -0.7180
 30   1.2008   0.5554   31  -1.7974  -0.2644  32   1.8620  -0.1016
 33  -1.4125   0.4318   34   0.5785  -0.6179  35   0.3936   0.6246
 36  -1.2406  -0.4653   37   1.6936   0.1750  38  -1.6685   0.1676
 39   1.1592  -0.4499   40  -0.3400   0.6010  41  -0.5827  -0.5868
 42   1.3093   0.3784   43  -1.6594  -0.0844  44   1.5192  -0.2383
 45  -0.9428   0.4937   46   0.1033  -0.5935  47   0.7498   0.5272
 48  -1.4084  -0.3079   49   1.6349   0.0000
```

Example 3: Calculation of the Squared Coherency and the Impulse Causal Response Function

This example program shows how `nag_bivar_spectral_coh` may be used to calculate the squared coherency. It also shows how `nag_bivar_spectral_lin_sys` may be used to calculate the impulse casual response function by using the optional argument `crf`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_tsa_spectral_ex03

  ! Example Program Text for nag_tsa_spectral
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_tsa_spectral, ONLY : nag_bivar_spectral_coh, &
   nag_bivar_spectral_lin_sys
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, l, n
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: crf(:), f_x(:), f_y(:), sqrd_coh(:)
  REAL (wp) :: stats(4)
  COMPLEX (wp), ALLOCATABLE :: f_xy(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) &
   'Example Program Results for nag_tsa_spectral_ex03'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) l, n

  ALLOCATE (f_x(l),f_y(l),f_xy(l),sqrd_coh(l),crf(l)) ! Allocate storage

  READ (nag_std_in,*) stats
  READ (nag_std_in,*) f_x
  READ (nag_std_in,*) f_y
  READ (nag_std_in,*) f_xy

  CALL nag_bivar_spectral_coh(f_x,f_y,f_xy,stats,sqrd_coh)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) &
   '  Squared coherency - using nag_bivar_spectral_coh'
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'((1X,4(I5,f10.4)))') (i-1,sqrd_coh(i),i=1,l)

  CALL nag_bivar_spectral_lin_sys(f_x,f_y,f_xy,stats,n,crf=crf)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '  Impulse causal response function &
   &- using nag_bivar_spectral_lin_sys'
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'((1X,4(I5,f10.4)))') (i-1,crf(i),i=1,l)
```

```
      DEALLOCATE (f_x,f_y,f_xy,sqrd_coh,crf) ! Deallocate storage

    END PROGRAM nag_tsa_spectral_ex03
```

2 Program Data

```
Example Program Data for nag_tsa_spectral_ex03
  9 296                                          : l, n
 30.00000  0.63858  1.78670  0.33288             : stats (1:4)
  2.03490  0.51554  0.07640  0.01068  0.00093
  0.00100  0.00076  0.00037  0.00021             : f_x(1:l)
 21.97712  3.29761  0.28782  0.02480  0.00285
  0.00203  0.00125  0.00107  0.00191             : f_y(1;l)
 (-6.54995,0.00000) (0.34107,-1.19030)
 (0.12335,0.04087)  (-0.00514,0.00842)
 (-0.00033,0.00032) (-0.00039,-0.00001)
 (-0.00026,0.00018) (0.00011,-0.00016)
 (0.00007,0.00000)                              : f_xy(1:l)
```

3 Program Results

```
 Example Program Results for nag_tsa_spectral_ex03

   Squared coherency - using nag_bivar_spectral_coh

     0    0.9593    1    0.9018    2    0.7679    3    0.3674
     4    0.0797    5    0.0750    6    0.1053    7    0.0952
     8    0.0122

   Impulse causal response function - using nag_bivar_spectral_lin_sys

     0   -0.0547    1    0.0586    2   -0.0322    3   -0.6956
     4   -0.7181    5   -0.8019    6   -0.4303    7   -0.2392
     8   -0.0766
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_tsa_spectral_ex04`

Computes the univariate spectrum together with the optional 95% confidence multiplying limits, statistics and autocovariances for a univariate time series.

`nag_tsa_spectral_ex05`

Computes the univariate logged spectrum together with the optional 95% confidence multiplying limits, statistics and frequency width of smoothing window for a univariate time series. No smoothing is carried out.

`nag_tsa_spectral_ex06`

Computes the univariate spectrum together with the optional 95% confidence multiplying limits and statistics for a univariate time series using autocovariances data.

`nag_tsa_spectral_ex07`

Computes the cross spectrum and the optional cross covariances for a bivariate time series.

`nag_tsa_spectral_ex08`

Computes the cross spectrum for a bivariate time series using autocovariances data.

`nag_tsa_spectral_ex09`

Computes the squared coherency together with the optional squared coherency lower and upper bounds, the cross amplitude spectrum and its bounds, the gain and the phase and their bounds for a univariate and bivariate spectra.

`nag_tsa_spectral_ex10`

Computes the optional noise spectrum and its lower and upper bounds, the impulse causal and anticipatory response function and its standard error for a linear system.

References

[1] Jenkins G M and Watts D G (1968) *Spectral Analysis and its Applications* Holden-Day