

NAG Library Function Document

nag_tsa_spectrum_univar (g13cbc)

1 Purpose

nag_tsa_spectrum_univar (g13cbc) calculates the smoothed sample spectrum of a univariate time series using spectral smoothing by the trapezium frequency (Daniell) window.

2 Specification

```
#include <nag.h>
#include <nagg13.h>

void nag_tsa_spectrum_univar (Integer nx, NagMeanOrTrend mt_correction,
    double px, Integer mw, double pw, Integer l, Integer kc,
    Nag_LoggedSpectra lg_spect, const double x[], double **g, Integer *ng,
    double stats[], NagError *fail)
```

3 Description

The supplied time series may be mean or trend corrected (by least squares), and tapered, the tapering factors being those of the split cosine bell:

$$\begin{aligned} & \frac{1}{2} \left(1 - \cos \left(\frac{\pi(t-\frac{1}{2})}{T} \right) \right), & 1 \leq t \leq T \\ & \frac{1}{2} \left(1 - \cos \left(\frac{\pi(n-t+\frac{1}{2})}{T} \right) \right), & n+1-T \leq t \leq n \\ & 1, & \text{otherwise} \end{aligned}$$

where $T = \lfloor \frac{np}{2} \rfloor$ and p is the tapering proportion.

The unsmoothed sample spectrum

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

is then calculated for frequency values

$$\omega_k = \frac{2\pi k}{K}, k = 0, 1, \dots, [K/2]$$

where $[]$ denotes the integer part.

The smoothed spectrum is returned as a subset of these frequencies for which K is a multiple of a chosen value r , i.e.,

$$\omega_{rl} = \nu_l = \frac{2\pi l}{L}, l = 0, 1, \dots, [L/2]$$

where $K = r \times L$. You will normally fix L first, then choose r so that K is sufficiently large to provide an adequate representation for the unsmoothed spectrum, i.e., $K \geq 2 \times n$. It is possible to take $L = K$, i.e., $r = 1$.

The smoothing is defined by a trapezium window whose shape is supplied by the function

$$\begin{aligned} W(\alpha) &= 1, & |\alpha| \leq p \\ W(\alpha) &= \frac{1-|\alpha|}{1-p}, & p < |\alpha| \leq 1 \end{aligned}$$

the proportion p being supplied by you.

The width of the window is fixed as $2\pi/M$ by supplying M . A set of averaging weights are constructed:

$$W_k = g \times W\left(\frac{\omega_k M}{\pi}\right), 0 \leq \omega_k \leq \frac{\pi}{M}$$

where g is a normalizing constant, and the smoothed spectrum obtained is

$$\hat{f}(\nu_l) = \sum_{|\omega_k| < \frac{\pi}{M}} W_k f^*(\nu_l + \omega_k).$$

If no smoothing is required M should be set to n , in which case the values returned are $\hat{f}(\nu_l) = f^*(\nu_l)$. Otherwise, in order that the smoothing approximates well to an integration, it is essential that $K \gg M$, and preferable, but not essential, that K be a multiple of M . A choice of $L > M$ would normally be required to supply an adequate description of the smoothed spectrum. Typical choices of $L \simeq n$ and $K \simeq 4n$ should be adequate for usual smoothing situations when $M < n/5$.

The sampling distribution of $\hat{f}(\omega)$ is approximately that of a scaled χ_d^2 variate, whose degrees of freedom d is provided by the function, together with multiplying limits mu , ml from which approximate 95% confidence intervals for the true spectrum $f(\omega)$ may be constructed as $[ml \times \hat{f}(\omega), mu \times \hat{f}(\omega)]$. Alternatively, $\log \hat{f}(\omega)$ may be returned, with additive limits.

The bandwidth b of the corresponding smoothing window in the frequency domain is also provided. Spectrum estimates separated by (angular) frequencies much greater than b may be assumed to be independent.

4 References

Bloomfield P (1976) *Fourier Analysis of Time Series: An Introduction* Wiley

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

5 Arguments

- 1: **nx** – Integer *Input*
On entry: the length of the time series, n .
Constraint: **nx** ≥ 1 .
- 2: **mt_correction** – NagMeanOrTrend *Input*
On entry: whether the data are to be initially mean or trend corrected.
mt_correction = Nag_NoCorrection for no correction, **mt_correction** = Nag_Mean for mean correction, **mt_correction** = Nag_Trend for trend correction.
Constraint: **mt_correction** = Nag_NoCorrection, Nag_Mean or Nag_Trend.
- 3: **px** – double *Input*
On entry: the proportion of the data (totalled over both ends) to be initially tapered by the split cosine bell taper. (A value of 0.0 implies no tapering).
Constraint: $0.0 \leq \mathbf{px} \leq 1.0$.
- 4: **mw** – Integer *Input*
On entry: the value of M which determines the frequency width of the smoothing window as $2\pi/M$. A value of n implies no smoothing is to be carried out.
Constraint: $1 \leq \mathbf{mw} \leq \mathbf{nx}$.

- 5: **pw** – double *Input*
On entry: the shape argument, p , of the trapezium frequency window.
 A value of 0.0 gives a triangular window, and a value of 1.0 a rectangular window.
 If **mw** = **nx** (i.e., no smoothing is carried out), then **pw** is not used.
Constraint: $0.0 \leq \mathbf{pw} \leq 1.0$. if **mw** \neq **nx**.
- 6: **l** – Integer *Input*
On entry: the frequency division, L , of smoothed spectral estimates as $2\pi/L$.
Constraints:
 $\mathbf{l} \geq 1$;
 \mathbf{l} must be a factor of **kc** (see below).
- 7: **kc** – Integer *Input*
On entry: the order of the fast Fourier transform (FFT), K , used to calculate the spectral estimates. **kc** should be a multiple of small primes such as 2^m where m is the smallest integer such that $2^m \geq 2n$, provided $m \leq 20$.
Constraints:
 $\mathbf{kc} \geq 2 \times \mathbf{nx}$;
kc must be a multiple of **l**. The largest prime factor of **kc** must not exceed 19, and the total number of prime factors of **kc**, counting repetitions, must not exceed 20. These two restrictions are imposed by the internal FFT algorithm used.
- 8: **lg_spect** – Nag_LoggedSpectra *Input*
On entry: indicates whether unlogged or logged spectral estimates and confidence limits are required. **lg_spect** = Nag_Unlogged for unlogged. **lg_spect** = Nag_Logged for logged.
Constraint: **lg_spect** = Nag_Unlogged or Nag_Logged.
- 9: **x[kc]** – const double *Input*
On entry: the n data points.
- 10: **g** – double ** *Output*
On exit: vector which contains the **ng** spectral estimates $\hat{f}(\omega_i)$, for $i = 0, 1, \dots, [L/2]$, in **g**[0] to **g**[**ng** – 1] (logged if **lg_spect** = Nag_Logged). The memory for this vector is allocated internally. If no memory is allocated to **g** (e.g., when an input error is detected) then **g** will be **NULL** on return. If repeated calls to this function are required then **NAG_FREE** should be used to free the memory in between calls.
- 11: **ng** – Integer * *Output*
On exit: the number of spectral estimates, $[L/2] + 1$, in **g**.
- 12: **stats**[4] – double *Output*
On exit: four associated statistics. These are the degrees of freedom in **stats**[0], the lower and upper 95% confidence limit factors in **stats**[1] and **stats**[2] respectively (logged if **lg_spect** = Nag_Logged), and the bandwidth in **stats**[3].
- 13: **fail** – NagError * *Input/Output*
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_2_INT_ARG_CONS

On entry, **kc** = $\langle value \rangle$ while **l** = $\langle value \rangle$. These arguments must satisfy $kc \% l = 0$ when $l > 0$.

On entry, **kc** = $\langle value \rangle$ while **nx** = $\langle value \rangle$. These arguments must satisfy $kc \geq 2 \times nx$ when $nx > 0$.

NE_2_INT_ARG_GT

On entry, **mw** = $\langle value \rangle$ while **nx** = $\langle value \rangle$. These arguments must satisfy $mw \leq nx$.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **lg_spect** had an illegal value.

On entry, argument **mt_correction** had an illegal value.

NE_CONFID_LIMIT_FACT

The calculation of confidence limit factors has failed. Spectral estimates (logged if requested) are returned in **g**, and degrees of freedom and bandwidth in **stats**.

NE_FACTOR_GT

At least one of the prime factors of **kc** is greater than 19.

NE_INT_ARG_LT

On entry, **l** = $\langle value \rangle$.

Constraint: $l \geq 1$.

On entry, **mw** = $\langle value \rangle$.

Constraint: $mw \geq 1$.

On entry, **nx** = $\langle value \rangle$.

Constraint: $nx \geq 1$.

On entry, **pw** must not be less than 0.0: **pw** = $\langle value \rangle$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_REAL_ARG_GT

On entry, **pw** must not be greater than 1.0: **pw** = $\langle value \rangle$.

On entry, **px** must not be greater than 1.0: **px** = $\langle value \rangle$.

NE_REAL_ARG_LT

On entry, **px** must not be less than 0.0: **px** = $\langle value \rangle$.

NE_SPECTRAL_ESTIM_NEG

One or more spectral estimates are negative. Unlogged spectral estimates are returned in **g** and the degrees of freedom, unlogged confidence limit factors and bandwidth in **stats**.

NE_TOO_MANY_FACTORS

kc has more than 20 prime factors.

7 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.

8 Parallelism and Performance

`nag_tsa_spectrum_univar` (g13cbc) is not threaded in any implementation.

9 Further Comments

`nag_tsa_spectrum_univar` (g13cbc) carries out a FFT of length **kc** to calculate the sample spectrum. The time taken by the function for this is approximately proportional to $\mathbf{kc} \times \log(\mathbf{kc})$ (but see Section 9 in `nag_sum_fft_realherm_1d` (c06pac) for further details).

10 Example

The example program reads a time series of length 131. It selects the mean correction option, a tapering proportion of 0.2, the option of no smoothing and a frequency division for logged spectral estimates of $2\pi/100$. It then calls `nag_tsa_spectrum_univar` (g13cbc) to calculate the univariate spectrum and prints the logged spectrum together with 95% confidence limits. The program then selects a smoothing window with frequency width $2\pi/30$ and shape argument 0.5 and recalculates and prints the logged spectrum and 95% confidence limits.

10.1 Program Text

```

/* nag_tsa_spectrum_univar (g13cbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <nagg13.h>

#define KCMAX 400
#define NXMAX KCMAX/2

int main(void)
{
    Integer exit_status = 0, i, kc, l, mw, ng, nx;
    NagError fail;
    double *g, pw, px, *stats = 0, *x = 0, *xh = 0;

    INIT_FAIL(fail);

    printf("nag_tsa_spectrum_univar (g13cbc) Example Program Results\n");
    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}

```

```

#ifdef _WIN32
    scanf_s("%" NAG_IFMT " ", &nx);
#else
    scanf("%" NAG_IFMT " ", &nx);
#endif
if (nx >= 1 && nx <= NXMAX) {
    if (!(stats = NAG_ALLOC(4, double)) ||
        !(x = NAG_ALLOC(KCMAX, double)) || !(xh = NAG_ALLOC(NXMAX, double))
        )
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else {
    printf("Invalid nx.\n");
    exit_status = 1;
    return exit_status;
}
for (i = 1; i <= nx; ++i)
#ifdef _WIN32
    scanf_s("%lf ", &xh[i - 1]);
#else
    scanf("%lf ", &xh[i - 1]);
#endif
px = 0.2;
mw = nx;
pw = 0.5;
kc = KCMAX;
l = 100;
#ifdef _WIN32
    while ((scanf_s("%" NAG_IFMT " ", &mw)) != EOF)
#else
    while ((scanf("%" NAG_IFMT " ", &mw)) != EOF)
#endif
    {
        if (mw > 0 && mw <= nx) {
            for (i = 1; i <= nx; ++i)
                x[i - 1] = xh[i - 1];

            /* nag_tsa_spectrum_univar (g13cbc).
             * Univariate time series, smoothed sample spectrum using
             * spectral smoothing by the trapezium frequency (Daniell)
             * window
             */
            nag_tsa_spectrum_univar(nx, Nag_Mean, px, mw, pw, l, kc, Nag_Logged,
                                   x, &g, &ng, stats, &fail);
            if (fail.code != NE_NOERROR) {
                printf("Error from nag_tsa_spectrum_univar (g13cbc).\n%s\n",
                       fail.message);
                exit_status = 1;
                goto END;
            }
        }

        if (mw == nx)
            printf("\n No smoothing\n\n");
        else
            printf("\n Frequency width of smoothing window = "
                   "1/" NAG_IFMT "\n\n", mw);
        printf(" Degrees of freedom =%4.1f      Bandwidth =%7.4f\n\n",
               stats[0], stats[3]);
        printf(" 95 percent confidence limits -      Lower =%7.4f "
               "Upper =%7.4f\n\n", stats[1], stats[2]);
        printf("      Spectrum      Spectrum      Spectrum\n");
        printf("      Spectrum\n");
        printf("      estimate      estimate      estimate\n");
        printf("      estimate\n\n");
        for (i = 1; i <= ng; ++i)
            printf("%5" NAG_IFMT "%10.4f%s", i, g[i - 1],

```


	estimate		estimate		estimate		estimate
1	-0.1776	2	-0.4561	3	-0.1784	4	1.9042
5	2.1094	6	1.7061	7	-0.7659	8	-1.4734
9	-1.5939	10	-2.1157	11	-2.9151	12	-2.7055
13	-2.8200	14	-3.4077	15	-3.8813	16	-3.6607
17	-4.0601	18	-4.4756	19	-4.2700	20	-4.3092
21	-4.5711	22	-4.8111	23	-4.5658	24	-4.7285
25	-5.4386	26	-5.5081	27	-5.2325	28	-5.0262
29	-4.4539	30	-4.4764	31	-4.9152	32	-5.8492
33	-5.5872	34	-4.9804	35	-4.8904	36	-5.2666
37	-5.7643	38	-5.8620	39	-5.5011	40	-5.7129
41	-6.3894	42	-6.4027	43	-6.1352	44	-6.5766
45	-7.3676	46	-7.1405	47	-6.1674	48	-5.8600
49	-6.1036	50	-6.2673	51	-6.4321		
