

# NAG Library Function Document

## nag\_running\_median\_smoother (g10cac)

### 1 Purpose

nag\_running\_median\_smoother (g10cac) computes a smoothed data sequence using running median smoothers.

### 2 Specification

```
#include <nag.h>
#include <nagg10.h>

void nag_running_median_smoother (Nag_Smooth_Type smoother, Integer n,
    const double y[], double smooth[], double rough[], NagError *fail)
```

### 3 Description

Given a sequence of  $n$  observations recorded at equally spaced intervals, nag\_running\_median\_smoother (g10cac) fits a smooth curve through the data using one of two smoothers. They are based on the use of running medians and averages to summarise the overlapping segments. The fit is called the smooth, the residuals the rough and they obey the following:

Data = Smooth + Rough

The two smoothers are :

1. 4253H, twice consisting of a running median of 4, then 2, then 5, then 3 followed by Hanning. Hanning is a running weighted average, the weights being 1/4, 1/2 and 1/4. The result of this smoothing is then reroughed by computing residuals, applying the same smoother to them and adding the result to the smooth of the first pass.
2. 3RSSH, twice consisting of a running median of 3, two splitting operations named S to improve the smooth sequence, each of which is followed by a running median of 3, and finally Hanning. The end points are dealt with using the method described by Velleman and Hoaglin (1981). The full smoother 3RSSH, twice is produced by reroughing as described above.

The compound smoother 4253H, twice is recommended. The smoother 3RSSH, twice is popular when calculating by hand as it requires simpler computations and is included for comparison purposes.

### 4 References

Tukey J W (1977) *Exploratory Data Analysis* Addison–Wesley

Velleman P F and Hoaglin D C (1981) *Applications, Basics, and Computing of Exploratory Data Analysis* Duxbury Press, Boston, MA

### 5 Arguments

1: **smoother** – Nag\_Smooth\_Type *Input*

*On entry:* **smoother** must specify the method to be used.

**smoother** = Nag\_4253H  
4253H, twice is used.

**smoother** = Nag\_3RSSH  
3RSSH, twice is used.

*Constraint:* **smoother** = Nag\_4253H or Nag\_3RSSH.

- 2: **n** – Integer *Input*  
*On entry:* the number,  $n$ , of the observations.  
*Constraint:*  $n > 6$ .  
 If  $n \leq 6$  then the sequence is not long enough to carry out smoothing.
- 3: **y[n]** – const double *Input*  
*On entry:* the sample observations.
- 4: **smooth[n]** – double *Output*  
*On exit:* contains the smooth.
- 5: **rough[n]** – double *Output*  
*On exit:* contains the rough.
- 6: **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\_BAD\_PARAM

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

### NE\_INT\_ARG\_LE

On entry,  $n = \langle value \rangle$ .  
 Constraint:  $n > 6$ .

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

nag\_running\_median\_smoother (g10cac) is not threaded in any implementation.

## 9 Further Comments

### 9.1 Internal Changes

Internal changes have been made to this function as follows:

At Mark 25: nag\_running\_median\_smoother (g10cac) is a smoothing function with two possible smoothing methods. The function was previously using the incorrect method (i.e., if you asked for method A you would get method B, and vice versa).

## 10 Example

The example program reads in a sequence of 49 data taken from Tukey (1977), above. Results are obtained using the two smoothing methods described.

## 10.1 Program Text

```

/* nag_running_median_smoother (g10cac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagg10.h>

int main(void)
{
    Integer exit_status = 0, i, n;
    NagError fail;
    Nag_Smooth_Type smoother;
    double *rough0 = 0, *smooth0 = 0, *rough1 = 0, *smooth1 = 0, *y = 0;

    INIT_FAIL(fail);

    printf("nag_running_median_smoother (g10cac) Example Program Results\n");
    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &n);
#else
    scanf("%" NAG_IFMT "", &n);
#endif
    if (n >= 1) {
        if (!(rough0 = NAG_ALLOC(n, double)) ||
            !(smooth0 = NAG_ALLOC(n, double)) ||
            !(rough1 = NAG_ALLOC(n, double)) ||
            !(smooth1 = NAG_ALLOC(n, double)) || !(y = NAG_ALLOC(n, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else {
        printf("Invalid n.\n");
        exit_status = 1;
        return exit_status;
    }
    for (i = 0; i < n; ++i)
#ifdef _WIN32
        scanf_s("%lf", &y[i]);
#else
        scanf("%lf", &y[i]);
#endif

    /* nag_running_median_smoother (g10cac).
     * Compute smoothed data sequence using running median smoothers
     */
    /* Smooth sequence using 3RSSH,twice */
    smoother = Nag_3RSSH;
    nag_running_median_smoother(smoother, n, y, smooth1, rough1, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_running_median_smoother (g10cac).\n%s\n",
            fail.message);
        exit_status = 1;
    }
}

```

```

    goto END;
}

/* Smooth sequence using 4253H,twice */
smoother = Nag_4253H;
nag_running_median_smoother(smoother, n, y, smooth0, rough0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_running_median_smoother (g10cac).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Display results */
printf("\n");
printf("
Index      Data      Using 3RSSH,twice      Using 4253H,twice\n");
printf("      Smooth      Rough      Smooth      Rough\n");
for (i = 0; i < n; ++i)
    printf("%4" NAG_IFMT " %10.1f %12.4f %12.4f %12.1f %12.1f\n", i, y[i],
        smooth1[i], rough1[i], smooth0[i], rough0[i]);
END:
NAG_FREE(rough0);
NAG_FREE(smooth0);
NAG_FREE(rough1);
NAG_FREE(smooth1);
NAG_FREE(y);
return exit_status;
}

```

## 10.2 Program Data

nag\_running\_median\_smoother (g10cac) Example Program Data

```

49
569.0 416.0 422.0 565.0 484.0 520.0 573.0 518.0 501.0 505.0
468.0 382.0 310.0 334.0 359.0 372.0 439.0 446.0 349.0 395.0
461.0 511.0 583.0 590.0 620.0 578.0 534.0 631.0 600.0 438.0
516.0 534.0 467.0 457.0 392.0 467.0 500.0 493.0 410.0 412.0
416.0 403.0 422.0 459.0 467.0 512.0 534.0 552.0 545.0

```

## 10.3 Program Results

nag\_running\_median\_smoother (g10cac) Example Program Results

Index	Data	Using 3RSSH,twice		Using 4253H,twice	
		Smooth	Rough	Smooth	Rough
0	569.0	416.0000	153.0000	491.4	77.6
1	416.0	416.0000	0.0000	491.4	-75.4
2	422.0	431.5000	-9.5000	491.4	-69.4
3	565.0	473.0000	92.0000	498.9	66.1
4	484.0	509.5000	-25.5000	514.9	-30.9
5	520.0	520.6875	-0.6875	524.7	-4.7
6	573.0	521.5625	51.4375	525.0	48.0
7	518.0	518.0000	0.0000	521.2	-3.2
8	501.0	510.0000	-9.0000	512.6	-11.6
9	505.0	496.5000	8.5000	493.2	11.8
10	468.0	455.2500	12.7500	449.7	18.3
11	382.0	387.5000	-5.5000	391.6	-9.6
12	310.0	339.7500	-29.7500	353.4	-43.4
13	334.0	334.9375	-0.9375	343.8	-9.8
14	359.0	353.9375	5.0625	355.2	3.8
15	372.0	376.1250	-4.1250	382.8	-10.8
16	439.0	392.2500	46.7500	405.5	33.5
17	446.0	396.2500	49.7500	411.9	34.1
18	349.0	403.0000	-54.0000	411.6	-62.6
19	395.0	427.2500	-32.2500	420.9	-25.9
20	461.0	461.3750	-0.3750	456.1	4.9
21	511.0	513.3125	-2.3125	513.9	-2.9
22	583.0	567.5625	15.4375	565.2	17.8
23	590.0	590.0000	0.0000	589.5	0.5
24	620.0	593.5000	26.5000	594.7	25.3

25	578.0	595.2500	-17.2500	594.6	-16.6
26	534.0	590.9375	-56.9375	591.8	-57.8
27	631.0	566.8125	64.1875	583.8	47.2
28	600.0	531.5000	68.5000	569.0	31.0
29	438.0	516.0000	-78.0000	546.3	-108.3
30	516.0	516.0000	0.0000	517.3	-1.3
31	534.0	501.8750	32.1250	489.6	44.4
32	467.0	473.6250	-6.6250	471.2	-4.2
33	457.0	457.0000	0.0000	463.5	-6.5
34	392.0	452.0000	-60.0000	464.2	-72.2
35	467.0	440.1250	26.8750	468.5	-1.5
36	500.0	421.3750	78.6250	470.6	29.4
37	493.0	412.0000	81.0000	462.3	30.7
38	410.0	412.0000	-2.0000	438.6	-28.6
39	412.0	412.0000	0.0000	416.1	-4.1
40	416.0	411.0625	4.9375	408.9	7.1
41	403.0	410.6875	-7.6875	412.2	-9.2
42	422.0	422.0000	0.0000	424.9	-2.9
43	459.0	446.6250	12.3750	448.1	10.9
44	467.0	476.3750	-9.3750	478.8	-11.8
45	512.0	509.0000	3.0000	510.0	2.0
46	534.0	534.0000	0.0000	534.1	-0.1
47	552.0	545.0000	7.0000	547.0	5.0
48	545.0	547.7500	-2.7500	550.9	-5.9

---