

## NAG Library Function Document

### nag\_kernel\_density\_estim (g10bac)

#### 1 Purpose

nag\_kernel\_density\_estim (g10bac) performs kernel density estimation using a Gaussian kernel.

#### 2 Specification

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

void nag_kernel_density_estim (Integer n, const double x[], double window,
    double low, double high, Integer ns, double smooth[], double t[],
    NagError *fail)
```

#### 3 Description

Given a sample of  $n$  observations,  $x_1, x_2, \dots, x_n$ , from a distribution with unknown density function,  $f(x)$ , an estimate of the density function,  $\hat{f}(x)$ , may be required. The simplest form of density estimator is the histogram. This may be defined by:

$$\hat{f}(x) = \frac{1}{nh} n_j, \quad a + (j-1)h < x < a + jh, \quad j = 1, 2, \dots, n_s,$$

where  $n_j$  is the number of observations falling in the interval  $a + (j-1)h$  to  $a + jh$ ,  $a$  is the lower bound to the histogram and  $b = n_s h$  is the upper bound. The value  $h$  is known as the window width. To produce a smoother density estimate a kernel method can be used. A kernel function,  $K(t)$ , satisfies the conditions:

$$\int_{-\infty}^{\infty} K(t) dt = 1 \quad \text{and} \quad K(t) \geq 0.$$

The kernel density estimator is then defined as:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right).$$

The choice of  $K$  is usually not important but to ease the computational burden use can be made of the Gaussian kernel defined as:

$$K(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}.$$

The smoothness of the estimator depends on the window width  $h$ . The larger the value of  $h$  the smoother the density estimate. The value of  $h$  can be chosen by examining plots of the smoothed density for different values of  $h$  or by using cross-validation methods (see Silverman (1990)).

Silverman (1982) and Silverman (1990) show how the Gaussian kernel density estimator can be computed using a fast Fourier transform (FFT). In order to compute the kernel density estimate over the range  $a$  to  $b$  the following steps are required:

1. discretize the data to give  $n_s$  equally spaced points  $t_l$  with weights  $\xi_l$  (see Jones and Lotwick (1984));
2. compute the FFT of the weights  $\xi_l$  to give  $Y_l$ ;
3. compute  $\zeta_l = e^{-\frac{1}{2}h^2 s_l^2} Y_l$  where  $s_l = 2\pi l / (b - a)$ ;
4. find the inverse FFT of  $\zeta_l$  to give  $\hat{f}(x)$ .

## 4 References

Jones M C and Lotwick H W (1984) Remark AS R50. A remark on algorithm AS 176. Kernel density estimation using the Fast Fourier Transform *Appl. Statist.* **33** 120–122

Silverman B W (1982) Algorithm AS 176. Kernel density estimation using the fast Fourier transform *Appl. Statist.* **31** 93–99

Silverman B W (1990) *Density Estimation* Chapman and Hall

## 5 Arguments

- 1: **n** – Integer *Input*  
*On entry:* the number of observations in the sample,  $n$ .  
*Constraint:* **n** > 0.
  
- 2: **x[n]** – const double *Input*  
*On entry:* the  $n$  observations,  $x_i$ , for  $i = 1, 2, \dots, n$ .
  
- 3: **window** – double *Input*  
*On entry:* the window width,  $h$ .  
*Constraint:* **window** > 0.0.
  
- 4: **low** – double *Input*  
*On entry:* the lower limit of the interval on which the estimate is calculated,  $a$ . For most applications **low** should be at least three window widths below the lowest data point.  
*Constraint:* **low** < **high**.
  
- 5: **high** – double *Input*  
*On entry:* the upper limit of the interval on which the estimate is calculated,  $b$ . For most applications **high** should be at least three window widths above the highest data point.
  
- 6: **ns** – Integer *Input*  
*On entry:* the number of points at which the estimate is calculated,  $n_s$ .  
*Constraints:*  

**ns**  $\geq 2$ ;  
 The largest prime factor of **ns** must not exceed 19, and the total number of prime factors of **ns**, counting repetitions, must not exceed 20.
  
- 7: **smooth[ns]** – double *Output*  
*On exit:* the  $n_s$  values of the density estimate,  $\hat{f}(t_l)$ , for  $l = 1, 2, \dots, n_s$ .
  
- 8: **t[ns]** – double *Output*  
*On exit:* the points at which the estimate is calculated,  $t_l$ , for  $l = 1, 2, \dots, n_s$ .
  
- 9: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_2\_REAL\_ARG\_LE

On entry, **high** =  $\langle value \rangle$  while **low** =  $\langle value \rangle$ . These arguments must satisfy **high** > **low**.

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_C06\_FACTORS

At least one of the prime factors of **ns** is greater than 19 or **ns** has more than 20 prime factors.

### NE\_G10BA\_INTERVAL

On entry, the interval given by **low** to **high** does not extend beyond three **window** widths at either extreme of the dataset. This may distort the density estimate in some cases.

### NE\_INT\_ARG\_LE

On entry, **n** =  $\langle value \rangle$ .  
Constraint: **n** > 0.

### NE\_INT\_ARG\_LT

On entry, **ns** =  $\langle value \rangle$ .  
Constraint: **ns**  $\geq$  2.

### 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\_LE

On entry, **window** must not be less than or equal to 0.0: **window** =  $\langle value \rangle$ .

## 7 Accuracy

See Jones and Lotwick (1984) for a discussion of the accuracy of this method.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

The time for computing the weights of the discretized data is of order  $n$  while the time for computing the FFT is of order  $n_s \log(n_s)$  as is the time for computing the inverse of the FFT.

## 10 Example

A sample of 1000 standard Normal (0,1) variates are generated using `nag_rand_normal` (g05skc) and the density estimated on 100 points with a window width of 0.1.

## 10.1 Program Text

```

/* nag_kernel_density_estim (g10bac) Example Program.
 *
 * Copyright 2000 Numerical Algorithms Group.
 *
 * Mark 6, 2000.
 * Mark 7b revised, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>
#include <nagg05.h>
#include <nagg10.h>

int main(void)
{
    /* Integer scalar and array declarations */
    Integer    exit_status = 0, i, increment, j, lstate;
    Integer    *state = 0, *isort = 0;

    /* NAG structures */
    NagError    fail;

    /* Double scalar and array declarations */
    double     high, low, window;
    double     *s = 0, *smooth = 0, *x = 0;

    /* Choose the base generator */
    Nag_BaseRNG genid = Nag_Basic;
    Integer    subid = 0;

    /* Set the seed */
    Integer    seed[] = { 1762543 };
    Integer    lseed = 1;

    /* Set the distribution parameters for the simulated data */
    double     xmu = 0.0e0;
    double     var = 1.0e0;

    /* Generate 1000 data points in the simulated data */
    Integer    n = 1000;

    /* Number of points at which to estimate density */
    Integer    ns = 100;

    INIT_FAIL(fail);

    printf("nag_kernel_density_estim (g10bac) Example Program Results\n");

    /* Get the length of the state array */
    lstate = -1;
    nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }

    /* Allocate some memory for the arrays */
    if (!(x = NAG_ALLOC(n, double))
        || !(s = NAG_ALLOC(ns, double))
        || !(state = NAG_ALLOC(lstate, Integer))
        || !(smooth = NAG_ALLOC(ns, double))
        || !(isort = NAG_ALLOC(ns, Integer)))
    {

```

```

    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Initialise the generator to a repeatable sequence */
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &state, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* Generate the variates */
nag_rand_normal(n, xmu, var, state, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_normal (g05skc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* Skip heading in data file */
scanf("%*[\n] ");
/* Read in the windowing information */
scanf("%lf ", &window);
scanf("%lf, %lf", &low, &high);

/* Perform kernel density estimation */
/* nag_kernel_density_estim (g10bac).
 * Kernel density estimate using Gaussian kernel
 */
nag_kernel_density_estim(n, x, window, low, high, ns, smooth, s, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_kernel_density_estim (g10bac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

printf("\n  Points Density Points Density Points  "
       "Density Points Density\n");
printf("      Value      Value      Value      "
       "Value      "
       "Value      Value\n\n");

increment = 25;
for (i = 1; i <= ns/4; i++)
{
    printf("%9.4f %7.4f", s[i-1], smooth[i-1]);
    for (j = 1; j <= 3; j++)
    {
        printf("%9.4f %7.4f", s[i-1+j*increment],
               smooth[i-1+j*increment]);
    }
    printf("\n");
}
END:
NAG_FREE(x);
NAG_FREE(s);
NAG_FREE(smooth);
NAG_FREE(isort);
NAG_FREE(state);
return exit_status;
}

```

## 10.2 Program Data

```
nag_kernel_density_estim (g10bac) Example Program Data
0.1
-4.0, 4.0
```

## 10.3 Program Results

```
nag_kernel_density_estim (g10bac) Example Program Results
```

Points	Density Value	Points	Density Value	Points	Density Value	Points	Density Value
-3.9600	0.0000	-1.9600	0.0274	0.0400	0.3682	2.0400	0.0476
-3.8800	0.0000	-1.8800	0.0455	0.1200	0.3864	2.1200	0.0467
-3.8000	0.0000	-1.8000	0.0646	0.2000	0.4285	2.2000	0.0366
-3.7200	0.0000	-1.7200	0.0771	0.2800	0.4436	2.2800	0.0282
-3.6400	0.0000	-1.6400	0.0904	0.3600	0.4409	2.3600	0.0332
-3.5600	0.0000	-1.5600	0.0998	0.4400	0.4068	2.4400	0.0357
-3.4800	0.0000	-1.4800	0.1052	0.5200	0.3654	2.5200	0.0312
-3.4000	0.0000	-1.4000	0.1275	0.6000	0.3516	2.6000	0.0273
-3.3200	0.0000	-1.3200	0.1732	0.6800	0.3366	2.6800	0.0223
-3.2400	0.0002	-1.2400	0.2267	0.7600	0.3038	2.7600	0.0138
-3.1600	0.0017	-1.1600	0.2434	0.8400	0.2577	2.8400	0.0057
-3.0800	0.0043	-1.0800	0.2353	0.9200	0.2096	2.9200	0.0031
-3.0000	0.0043	-1.0000	0.2299	1.0000	0.1935	3.0000	0.0060
-2.9200	0.0017	-0.9200	0.2462	1.0800	0.2032	3.0800	0.0085
-2.8400	0.0002	-0.8400	0.3012	1.1600	0.2235	3.1600	0.0061
-2.7600	0.0004	-0.7600	0.3398	1.2400	0.2301	3.2400	0.0020
-2.6800	0.0030	-0.6800	0.3321	1.3200	0.2145	3.3200	0.0003
-2.6000	0.0082	-0.6000	0.3032	1.4000	0.1814	3.4000	0.0000
-2.5200	0.0094	-0.5200	0.2780	1.4800	0.1371	3.4800	0.0000
-2.4400	0.0073	-0.4400	0.3188	1.5600	0.1196	3.5600	0.0000
-2.3600	0.0165	-0.3600	0.3850	1.6400	0.1268	3.6400	0.0000
-2.2800	0.0283	-0.2800	0.4016	1.7200	0.1118	3.7200	0.0000
-2.2000	0.0223	-0.2000	0.3829	1.8000	0.0846	3.8000	0.0000
-2.1200	0.0161	-0.1200	0.3646	1.8800	0.0621	3.8800	0.0000
-2.0400	0.0178	-0.0400	0.3687	1.9600	0.0487	3.9600	0.0000

---