NAG C 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, $x_1, x_2, \ldots, x_n$, 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:

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

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:

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

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 (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 ξ_l (see Jones and Lotwick (1984));

(2) compute the FFT of the weights ξ_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 ζ_l to give $\hat{f}(x)$.

4 Parameters

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 $i = 1, 2, \ldots, n$.

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**[] ≥ 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 $l = 1, 2, \ldots, n_s$.

8: **t[ns]** – double *Output*

On exit: the points at which the estimate is calculated, t_l, for $l = 1, 2, \ldots, n_s$.

9: **fail** – NagError * *Input/Output*

The NAG error parameter (see the Essential Introduction).

5 Error Indicators and Warnings

NE_INT_ARG_LE

On entry, **n**[] must not be less than or equal to 0: **n**[] = <*value*>.

NE_INT_ARG_LT

On entry, **ns**[] must not be less than 2: **ns**[] = <*value*>.

NE_REAL_ARG_LE

On entry, **window**[] must not be less than or equal to 0.0: **window**[] = <*value*>.

NE_2_REAL_ARG_LE

On entry, **high**[] = <*value*> while **low**[] = <*value*>.
These parameters must satisfy **high**[] > **low**[].

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 data set. This may distort the density estimate in some cases.

NE_ALLOC_FAIL

Memory allocation failed.

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 consult NAG for assistance.

6 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.

6.1 Accuracy

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

6.2 References

Jones M C and Lotwick H W (1984) Remark AS R50. A remark on algorithm AS 176 *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

7 See Also

None.

8 Example

A sample of 1000 standard Normal (0,1) variates are generated using nag_random_normal (g05ddc) and the density estimated on 100 points with a window width of 0.1.

8.1 Program Text

```
/* nag_kernel_density_estim (g10bac) Example Program.
 *
 * Copyright 2000 Numerical Algorithms Group.
 *
 * Mark 6, 2000.
 */


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

int main(void)
{
  Integer i, init, increment, j, n, ns;
  Integer exit_status=0;
  double enda, endb, *s=0, high, low, *smooth=0, window, *x=0;
  Integer ifail, *isort=0;
  Boolean usefft;
  NagError fail;

  INIT_FAIL(fail);
  Vprintf("g10bac Example Program Results\n");

/*  Skip heading in data file */
  Vscanf("%*[^\n] ");

  Vscanf("%lf ", &window);
  Vscanf("%lf , %lf", &low, &high);
/*  Generate Normal (0,1) Distribution */
  n = 1000;
  ns = 100;
  if (!(x = NAG_ALLOC(n, double))
      || !(s = NAG_ALLOC(ns, double))
      || !(smooth = NAG_ALLOC(ns, double))
      || !(isort = NAG_ALLOC(ns, Integer)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  init = 0;
  g05cbc(init);
  enda = 0.0;
  endb = 1.0;
  for (i = 0; i < n; i++)
    x[i] = g05ddc(enda, endb);

/*  Perform kernel density estimation */
  usefft = FALSE;
  ifail = 0;
   g10bac(n, x, window, low, high, ns, smooth, s, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      Vprintf("Error from g10bac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

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


  increment = 25;
  for (i=1; i<= ns/4; i++)
    {
      printf("%10.4f %10.4f", s[i-1], smooth[i-1]);
      for (j=1; j <= 3; j++)
        {
          printf("%10.4f %10.4f", s[i-1+j*increment], smooth[i-1+j*increment]);
        }
      printf("\n");
    }
 END:
  if (x) NAG_FREE(x);
  if (s) NAG_FREE(s);
  if (smooth) NAG_FREE(smooth);
  if (isort) NAG_FREE(isort);
  return exit_status;
}
```

8.2 Program Data

```
g10bac Example Program Data
0.1
-4.0, 4.0
```

8.3 Program Results

```
g10bac Example Program Results
    Points     Density   Points     Density   Points     Density   Points     Density
                Value                Value                Value                Value

   -3.9600     0.0000   -1.9600     0.0508    0.0400     0.3698    2.0400     0.0464
   -3.8800     0.0001   -1.8800     0.0573    0.1200     0.3614    2.1200     0.0361
   -3.8000     0.0011   -1.8000     0.0763    0.2000     0.3393    2.2000     0.0344
   -3.7200     0.0037   -1.7200     0.0763    0.2800     0.3346    2.2800     0.0307
   -3.6400     0.0049   -1.6400     0.0719    0.3600     0.3618    2.3600     0.0207
   -3.5600     0.0023   -1.5600     0.0942    0.4400     0.3553    2.4400     0.0096
   -3.4800     0.0003   -1.4800     0.1292    0.5200     0.3312    2.5200     0.0071
   -3.4000     0.0000   -1.4000     0.1440    0.6000     0.3356    2.6000     0.0133
   -3.3200     0.0003   -1.3200     0.1659    0.6800     0.3496    2.6800     0.0162
   -3.2400     0.0021   -1.2400     0.2181    0.7600     0.3310    2.7600     0.0117
   -3.1600     0.0047   -1.1600     0.2511    0.8400     0.2922    2.8400     0.0074
   -3.0800     0.0039   -1.0800     0.2443    0.9200     0.2812    2.9200     0.0077
   -3.0000     0.0015   -1.0000     0.2443    1.0000     0.3011    3.0000     0.0073
   -2.9200     0.0012   -0.9200     0.2415    1.0800     0.2872    3.0800     0.0040
```

```
-2.8400     0.0038   -0.8400     0.2565   1.1600     0.2134   3.1600     0.0011
-2.7600     0.0062   -0.7600     0.2970   1.2400     0.1577   3.2400     0.0001
-2.6800     0.0115   -0.6800     0.3435   1.3200     0.1395   3.3200     0.0000
-2.6000     0.0218   -0.6000     0.3642   1.4000     0.1370   3.4000     0.0004
-2.5200     0.0231   -0.5200     0.3822   1.4800     0.1315   3.4800     0.0029
-2.4400     0.0191   -0.4400     0.4081   1.5600     0.1295   3.5600     0.0055
-2.3600     0.0230   -0.3600     0.4051   1.6400     0.1270   3.6400     0.0031
-2.2800     0.0297   -0.2800     0.3843   1.7200     0.1109   3.7200     0.0006
-2.2000     0.0316   -0.2000     0.3447   1.8000     0.0947   3.8000     0.0000
-2.1200     0.0417   -0.1200     0.3214   1.8800     0.0847   3.8800     0.0000
-2.0400     0.0536   -0.0400     0.3474   1.9600     0.0655   3.9600     0.0000
```