NAG Library Function Document

nag_kernel_density_gauss (g10bbc)

1 Purpose

nag_kernel_density_gauss (g10bbc) performs kernel density estimation using a Gaussian kernel.

2 Specification

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

void nag_kernel_density_gauss (Integer n, const double x[],
    Nag_WindowType wtype, double *window, double *slo, double *shi,
    Integer ns, double smooth[], double t[], Nag_Boolean fcall,
    double rcomm[], 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:

\[
\hat{f}(x) = \frac{1}{nhn_j} \quad a + (j-1)h < x < a + jh, \quad j = 1, 2, \ldots, n_s,
\]

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, \( b = n_s h \) is the upper bound and \( n_s \) is the total number of intervals. 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.

(i) Discretize the data to give \( n_s \) equally spaced points \( t_l \) with weights \( \xi_l \) (see Jones and Lotwick (1984)).

(ii) Compute the FFT of the weights \( \xi_l \) to give \( Y_l \).

(iii) Compute \( \zeta_l = e^{-\frac{1}{2}h^2 \xi_l^2} \) where \( s_l = 2\pi l/(b-a) \).

(iv) Find the inverse FFT of \( \zeta_l \) to give \( \hat{f}(x) \).
To compute the kernel density estimate for further values of $h$ only steps (iii) and (iv) need be repeated.

4 References


5 Arguments

1: $n$ – Integer

On entry: $n$, the number of observations in the sample.

If $\text{fcall} = \text{Nag\_FALSE}$, $n$ must be unchanged since the last call to nag_kernel_density_gauss (g10bbc).

Constraint: $n > 0$.

2: $x[n]$ – const double

On entry: $x_i$, for $i = 1, 2, \ldots, n$.

If $\text{fcall} = \text{Nag\_FALSE}$, $x$ must be unchanged since the last call to nag_kernel_density_gauss (g10bbc).

3: $\text{wtype}$ – Nag\_WindowType

On entry: how the window width, $h$, is to be calculated:

$\text{wtype} = \text{Nag\_WindowSupplied}$

$h$ is supplied in window.

$\text{wtype} = \text{Nag\_RuleOfThumb}$

$h$ is to be calculated from the data, with

$$h = m \times \left(\frac{0.9 \times \min(q_{75} - q_{25}, \sigma)}{n^{0.2}}\right)$$

where $q_{75} - q_{25}$ is the inter-quartile range and $\sigma$ the standard deviation of the sample, $x$, and $m$ is a multiplier supplied in window. The 25\% and 75\% quartiles, $q_{25}$ and $q_{75}$, are calculated using nag_double_quantiles (g01amc). This is the "rule-of-thumb" suggested by Silverman (1990).

Suggested value: $\text{wtype} = \text{Nag\_RuleOfThumb}$ and window = 1.0

Constraint: wtype = Nag\_WindowSupplied or Nag\_RuleOfThumb.

4: window – double *

On entry: if $\text{wtype} = \text{Nag\_WindowSupplied}$, then $h$, the window width. Otherwise, $m$, the multiplier used in the calculation of $h$.

Suggested value: window = 1.0 and $\text{wtype} = \text{Nag\_RuleOfThumb}$

On exit: $h$, the window width actually used.

Constraint: window > 0.0.

5: slo – double *

On entry: if slo < shi then $a$, the lower limit of the interval on which the estimate is calculated. Otherwise, $a$ and $b$, the lower and upper limits of the interval, are calculated as follows:
where $h$ is the window width.

For most applications $a$ should be at least three window widths below the lowest data point.

If $\text{fcall} = \text{Nag\_FALSE}$, $\text{slo}$ must be unchanged since the last call to `nag_kernel_density_gauss` (g10bbc).

**Suggested value:** $\text{slo} = 3.0$ and $\text{shi} = 0.0$ which would cause $a$ and $b$ to be set 3 window widths below and above the lowest and highest data points respectively.

*On exit:* $a$, the lower limit actually used.

6: $\text{shi} = \text{double *}$  
*Input/Output*

*On entry:* if $\text{slo} < \text{shi}$ then $b$, the upper limit of the interval on which the estimate is calculated. Otherwise a value for $b$ is calculated from the data as stated in the description of $\text{slo}$ and the value supplied in $\text{shi}$ is not used.

For most applications $b$ should be at least three window widths above the highest data point.

If $\text{fcall} = \text{Nag\_FALSE}$, $\text{shi}$ must be unchanged since the last call to `nag_kernel_density_gauss` (g10bbc).

*On exit:* $b$, the upper limit actually used.

7: $\text{ns} = \text{Integer}$  
*Input*

*On entry:* $n_s$, the number of points at which the estimate is calculated.

If $\text{fcall} = \text{Nag\_FALSE}$, $\text{ns}$ must be unchanged since the last call to `nag_kernel_density_gauss` (g10bbc).

**Suggested value:** $\text{ns} = 512$

**Constraints:**

$\text{ns} \geq 2$;

The largest prime factor of $\text{ns}$ must not exceed 19, and the total number of prime factors of $\text{ns}$, counting repetitions, must not exceed 20.

8: $\text{smooth[ns]} = \text{double}$  
*Output*

*On exit:* $\hat{f}(t_l)$, for $l = 1, 2, \ldots, n_s$, the $n_s$ values of the density estimate.

9: $\text{t[ns]} = \text{double}$  
*Output*

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

10: $\text{fcall} = \text{Nag\_Boolean}$  
*Input*

*On entry:* If $\text{fcall} = \text{Nag\_TRUE}$ then the values of $Y_l$ are to be calculated by this call to `nag_kernel_density_gauss` (g10bbc), otherwise it is assumed that the values of $Y_l$ were calculated by a previous call to this routine and the relevant information is stored in $\text{rcomm}$.

11: $\text{rcomm[ns + 20]} = \text{double}$  
*Communication Array*

*On entry:* communication array, used to store information between calls to `nag_kernel_density_gauss` (g10bbc).

If $\text{fcall} = \text{Nag\_FALSE}$, $\text{rcomm}$ must be unchanged since the last call to `nag_kernel_density_gauss` (g10bbc).
On exit: the last ns elements of rcomm contain the fast Fourier transform of the weights of the
discretized data, that is \( rcomm[l + 19] = Y_l \), for \( l = 1, 2, \ldots, n_s \).

12: fail = NagError *

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

NE_ILLEGAL_COMM

rcomm has been corrupted between calls.

NE_INT

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n > 0 \).

On entry, \( n_s = \langle \text{value} \rangle \).
Constraint: \( n_s \geq 2 \).

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_PREV_CALL

On entry, \( n = \langle \text{value} \rangle \).

On entry at previous call, \( n = \langle \text{value} \rangle \).
Constraint: if \( \text{fcall} = \text{Nag_FALSE} \), \( n \) must be unchanged since previous call.

On entry, \( n_s = \langle \text{value} \rangle \).

On entry at previous call, \( n_s = \langle \text{value} \rangle \).
Constraint: if \( \text{fcall} = \text{Nag_FALSE} \), \( n_s \) must be unchanged since previous call.

On entry, \( shi = \langle \text{value} \rangle \).
On exit from previous call, \( shi = \langle \text{value} \rangle \).
Constraint: if \( \text{fcall} = \text{Nag_FALSE} \), \( shi \) must be unchanged since previous call.

On entry, \( slo = \langle \text{value} \rangle \).
On exit from previous call, \( slo = \langle \text{value} \rangle \).
Constraint: if \( \text{fcall} = \text{Nag_FALSE} \), \( slo \) must be unchanged since previous call.

NE_PRIME_FACTOR

On entry, \( n_s = \langle \text{value} \rangle \).

Constraint: Largest prime factor of \( n_s \) must not exceed 19.
On entry, \( \text{ns} = \langle \text{value} \rangle \).
Constraint: Total number of prime factors of \( \text{ns} \) must not exceed 20.

**NE_REAL**

On entry, \( \text{window} = \langle \text{value} \rangle \).
Constraint: \( \text{window} > 0.0 \).

**NW_POTENTIAL_PROBLEM**

On entry, \( \text{slo} = \langle \text{value} \rangle \) and \( \text{shi} = \langle \text{value} \rangle \).
On entry, \( \text{min(x)} = \langle \text{value} \rangle \) and \( \text{max(x)} = \langle \text{value} \rangle \).
Expected values of at least \( \langle \text{value} \rangle \) and \( \langle \text{value} \rangle \) for \( \text{slo} \) and \( \text{shi} \).
All output values have been returned.

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

Data is read from a file and the density estimated. The first 20 values are then printed.

10.1 Program Text

```c
/* nag_kernel_density_gauss (g10bbc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 24, 2013. */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nagg01.h>
#include <nagg10.h>

int main(void)
{
/* Integer scalar and array declarations */
    Integer n, ns, i;
    Integer exit_status = 0;

/* Nag Types */
    NagError fail;
    Nag_Boolean fcall;
    Nag_WindowType wtype;

/* Double scalar and array declarations */
    double shi, slo, window;
    double *rcomm = 0, *smooth = 0, *t = 0, *x = 0;

/* Character scalar and array declarations */
    char cwtype[40];
```
/* Initialise the error structure */
INIT_FAIL(fail);

printf("nag_kernel_density_gauss (g10bbc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif

/* Read in density estimation information */
#ifdef _WIN32
    scanf_s("%39s %lf %lf %lf %NAG_IFMT"%*[\n ] ", cwtype, _countof(cwtype),
            &window, &slo, &shi, &ns);
#else
    scanf("%39s %lf %lf %lf %NAG_IFMT"%*[\n ] ", cwtype, &window, &slo, &shi,
            &ns);
#endif
wtype = (Nag_WindowType) nag_enum_name_to_value(cwtype);

/* Read in the size of the dataset */
#ifdef _WIN32
    scanf("%NAG_IFMT"%*[\n ] ", &n);
#else
    scanf("%NAG_IFMT"%*[\n ] ", &n);
#endif
if (!(smooth = NAG_ALLOC(ns, double)) ||
    !(t = NAG_ALLOC(ns, double)) ||
    !(rcomm = NAG_ALLOC(ns+20, double)) ||
    !(x = NAG_ALLOC(n, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Only calling the routine once */
fcall = Nag_TRUE;

/* Read in data */
for (i = 0; i < n; i++)
{
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif

/* Call nag_kernel_density_gauss (g10bbc) to perform kernel *
* density estimation */
nag_kernel_density_gauss(n, x, wtype, &window, &slo, &shi, ns, smooth, t, fcall, rcomm, &fail);
if (fail.code != NE_NOERROR || fail.code != NW_POTENTIAL_PROBLEM)
{
    printf("Error from nag_kernel_density_gauss (g10bbc).\n\n", fail.message);
    exit_status = -1;
    goto END;
}
/* Display the summary of results */
printf("Window Width Used = %13.4e\n", window);
printf("Interval = (%13.4e,%13.4e)\n", slo, shi);
printf("\n");
printf("First %"NAG_IFMT" output values:\n", MIN(ns,20));
printf("\n");
printf(" Time Density\n");
printf(" Point Estimate\n");
printf(" ---------------------------\n");
for (i = 0; i < MIN(20,ns); i++)
    printf(" %13.3e %13.3e\n", t[i], smooth[i]);

END:
NAG_FREE(smooth);
NAG_FREE(t);
NAG_FREE(rcomm);
NAG_FREE(x);

return exit_status;
}

10.2 Program Data

nag_kernel_density_gauss (g10bbc) Example Program Data
Nag_RuleOfThumb  1.0  3.0  0.0  512 :: wtype,window,slo,shi,ns
  100
  0.114 -0.232 -0.570  1.853 -0.994
-0.374 -1.028  0.509  0.881 -0.453
  0.588 -0.625 -1.622 -0.567  0.421
-0.475  0.054  0.817  1.015  0.608
-1.353 -0.912 -1.136  1.067  0.121
-0.075 -0.745  1.217 -1.058 -0.894
  1.026 -0.967 -1.065  0.513  0.969
  0.582 -0.985  0.097  0.416 -0.514
  0.898 -0.154  0.617 -0.436 -1.212
-1.571  0.210 -1.101  1.018 -1.702
-2.230 -0.648 -0.350  0.446 -2.667
  0.094 -0.380 -2.852 -0.888 -1.481
-0.359 -0.554  1.531  0.052 -1.715
  1.255 -0.540  0.362 -0.654 -0.272
-1.810  0.269 -1.918  0.001  1.240
-0.368 -0.647 -2.282  0.498  0.001
-3.059 -1.171  0.566  0.948  0.925
  0.825  0.130  0.930  0.523  0.443
-0.649  0.554 -2.823  0.158 -1.180
  0.610  0.877  0.791 -0.078  1.412 :: End of x

10.3 Program Results

nag_kernel_density_gauss (g10bbc) Example Program Results
Window Width Used =  3.7638e-01
Interval = (  -4.1882e+00,  2.9822e+00)

First 20 output values:

<table>
<thead>
<tr>
<th>Time</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Estimate</td>
<td></td>
</tr>
<tr>
<td>-4.181e+00</td>
<td>3.828e-06</td>
</tr>
<tr>
<td>-4.167e+00</td>
<td>4.031e-06</td>
</tr>
<tr>
<td>-4.153e+00</td>
<td>4.423e-06</td>
</tr>
<tr>
<td>-4.139e+00</td>
<td>5.021e-06</td>
</tr>
<tr>
<td>-4.125e+00</td>
<td>5.846e-06</td>
</tr>
<tr>
<td>-4.111e+00</td>
<td>6.928e-06</td>
</tr>
<tr>
<td>-4.097e+00</td>
<td>8.305e-06</td>
</tr>
<tr>
<td>-4.083e+00</td>
<td>1.002e-05</td>
</tr>
<tr>
<td>-4.069e+00</td>
<td>1.215e-05</td>
</tr>
<tr>
<td>-4.055e+00</td>
<td>1.474e-05</td>
</tr>
</tbody>
</table>
This plot shows the estimated density function for the example data for several window widths.

**Example Program**
Gaussian Kernel Density Estimation

```
-4.041e+00  1.788e-05
-4.027e+00  2.168e-05
-4.013e+00  2.624e-05
-3.999e+00  3.170e-05
-3.985e+00  3.821e-05
-3.971e+00  4.596e-05
-3.957e+00  5.514e-05
-3.943e+00  6.599e-05
-3.929e+00  7.877e-05
-3.915e+00  9.380e-05
```