1 Purpose

nag_kernel_density_estim (g10bac) performs kernel density estimation using a Gaussian kernel.

2 Specification

```c
#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 \( b = n_sh \) 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 \( t_i \) with weights \( \xi_i \) (see Jones and Lotwick (1984));
2. compute the FFT of the weights \( \xi_i \) to give \( Y_i \);
3. compute \( \zeta_i = e^{-|t_i|^2}Y_i \) where \( s_i = 2\pi l/(b - a) \);
4. find the inverse FFT of \( \zeta \) to give \( \hat{f}(x) \).
4 References


5 Arguments

1: \( n \) – Integer

\( \text{Input} \)

\( On \ entry: \) the number of observations in the sample, \( n \).

\( Constraint: \ n > 0. \)

2: \( x[n] \) – const double

\( \text{Input} \)

\( On \ entry: \) the \( n \) observations, \( x_i \), for \( i = 1, 2, \ldots, n \).

3: \( \text{window} \) – double

\( \text{Input} \)

\( On \ entry: \) the window width, \( h \).

\( Constraint: \ \text{window} > 0.0. \)

4: \( \text{low} \) – double

\( \text{Input} \)

\( On \ entry: \) the lower limit of the interval on which the estimate is calculated, \( a \). For most applications \( \text{low} \) should be at least three window widths below the lowest data point.

\( Constraint: \ \text{low} < \text{high}. \)

5: \( \text{high} \) – double

\( \text{Input} \)

\( On \ entry: \) the upper limit of the interval on which the estimate is calculated, \( b \). For most applications \( \text{high} \) should be at least three window widths above the highest data point.

6: \( \text{ns} \) – Integer

\( \text{Input} \)

\( On \ entry: \) the number of points at which the estimate is calculated, \( n_s \).

\( Constraints: \)

\( \text{ns} \geq 2; \)

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.

7: \( \text{smooth}[\text{ns}] \) – double

\( \text{Output} \)

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

8: \( \text{t}[\text{ns}] \) – double

\( \text{Output} \)

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

9: \( \text{fail} \) – NagError *

\( \text{Input/Output} \)

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** = ⟨value⟩ while **low** = ⟨value⟩. 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** = ⟨value⟩.

Constraint: **n** > 0.

**NE_INT_ARG_LT**

On entry, **ns** = ⟨value⟩.

Constraint: **ns** ≥ 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** = ⟨value⟩.

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** (g05ske) and the density estimated on 100 points with a window width of 0.1.
10.1 Program Text

```c
#include <stdio.h>
#include <nag.h>
#include <naq_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;

    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, &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;
}

/* 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 */
#ifdef _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif
/* Read in the windowing information */
#ifdef _WIN32
  scanf_s("%lf ", &window);
#else
  scanf("%lf ", &window);
#endif
#ifdef _WIN32
  scanf_s("%lf, %lf", &low, &high);
#else
  scanf("%lf, %lf", &low, &high);
#endif

/* 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" 
"Value Value Value\n" 
"Value Value\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");
}
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

<table>
<thead>
<tr>
<th>Points Value</th>
<th>Density Value</th>
<th>Points Value</th>
<th>Density Value</th>
<th>Points Value</th>
<th>Density Value</th>
<th>Points Value</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.9600</td>
<td>0.0000</td>
<td>-1.9600</td>
<td>0.0274</td>
<td>0.0400</td>
<td>0.3682</td>
<td>2.0400</td>
<td>0.0476</td>
</tr>
<tr>
<td>-3.8800</td>
<td>0.0000</td>
<td>-1.8800</td>
<td>0.0455</td>
<td>0.1200</td>
<td>0.3864</td>
<td>2.1200</td>
<td>0.0467</td>
</tr>
<tr>
<td>-3.8000</td>
<td>0.0000</td>
<td>-1.8000</td>
<td>0.0646</td>
<td>0.2000</td>
<td>0.4285</td>
<td>2.2000</td>
<td>0.0366</td>
</tr>
<tr>
<td>-3.7200</td>
<td>0.0000</td>
<td>-1.7200</td>
<td>0.0771</td>
<td>0.2800</td>
<td>0.4436</td>
<td>2.2800</td>
<td>0.0282</td>
</tr>
<tr>
<td>-3.6400</td>
<td>0.0000</td>
<td>-1.6400</td>
<td>0.0904</td>
<td>0.3600</td>
<td>0.4409</td>
<td>2.3600</td>
<td>0.0332</td>
</tr>
<tr>
<td>-3.5600</td>
<td>0.0000</td>
<td>-1.5600</td>
<td>0.0998</td>
<td>0.4400</td>
<td>0.4068</td>
<td>2.4400</td>
<td>0.0357</td>
</tr>
<tr>
<td>-3.4800</td>
<td>0.0000</td>
<td>-1.4800</td>
<td>0.1052</td>
<td>0.5200</td>
<td>0.3654</td>
<td>2.5200</td>
<td>0.0312</td>
</tr>
<tr>
<td>-3.4000</td>
<td>0.0000</td>
<td>-1.4000</td>
<td>0.1275</td>
<td>0.6000</td>
<td>0.3516</td>
<td>2.6000</td>
<td>0.0273</td>
</tr>
<tr>
<td>-3.3200</td>
<td>0.0000</td>
<td>-1.3200</td>
<td>0.1732</td>
<td>0.6800</td>
<td>0.3366</td>
<td>2.6800</td>
<td>0.0223</td>
</tr>
<tr>
<td>-3.2400</td>
<td>0.0000</td>
<td>-1.2400</td>
<td>0.2267</td>
<td>0.7600</td>
<td>0.3038</td>
<td>2.7600</td>
<td>0.0139</td>
</tr>
<tr>
<td>-3.1600</td>
<td>0.0017</td>
<td>-1.1600</td>
<td>0.2434</td>
<td>0.8400</td>
<td>0.2577</td>
<td>2.8400</td>
<td>0.0057</td>
</tr>
<tr>
<td>-3.0800</td>
<td>0.0043</td>
<td>-1.0800</td>
<td>0.2353</td>
<td>0.9200</td>
<td>0.2096</td>
<td>2.9200</td>
<td>0.0031</td>
</tr>
<tr>
<td>-3.0000</td>
<td>0.0043</td>
<td>-1.0000</td>
<td>0.2299</td>
<td>1.0000</td>
<td>0.1935</td>
<td>3.0000</td>
<td>0.0060</td>
</tr>
<tr>
<td>-2.9200</td>
<td>0.0017</td>
<td>-0.9200</td>
<td>0.2462</td>
<td>1.0800</td>
<td>0.2032</td>
<td>3.0800</td>
<td>0.0085</td>
</tr>
<tr>
<td>-2.8400</td>
<td>0.0002</td>
<td>-0.8400</td>
<td>0.3012</td>
<td>1.1600</td>
<td>0.2235</td>
<td>3.1600</td>
<td>0.0061</td>
</tr>
<tr>
<td>-2.7600</td>
<td>0.0004</td>
<td>-0.7600</td>
<td>0.3398</td>
<td>1.2400</td>
<td>0.2301</td>
<td>3.2400</td>
<td>0.0020</td>
</tr>
<tr>
<td>-2.6800</td>
<td>0.0030</td>
<td>-0.6800</td>
<td>0.3321</td>
<td>1.3200</td>
<td>0.2145</td>
<td>3.3200</td>
<td>0.0003</td>
</tr>
<tr>
<td>-2.6000</td>
<td>0.0082</td>
<td>-0.6000</td>
<td>0.3032</td>
<td>1.4000</td>
<td>0.1834</td>
<td>3.4000</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.5200</td>
<td>0.0094</td>
<td>-0.5200</td>
<td>0.2780</td>
<td>1.4800</td>
<td>0.1371</td>
<td>3.4800</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.4400</td>
<td>0.0073</td>
<td>-0.4400</td>
<td>0.3188</td>
<td>1.5600</td>
<td>0.1196</td>
<td>3.5600</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.3600</td>
<td>0.0165</td>
<td>-0.3600</td>
<td>0.3850</td>
<td>1.6400</td>
<td>0.1268</td>
<td>3.6400</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.2800</td>
<td>0.0283</td>
<td>-0.2800</td>
<td>0.4016</td>
<td>1.7200</td>
<td>0.1118</td>
<td>3.7200</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.2000</td>
<td>0.0223</td>
<td>-0.2000</td>
<td>0.3829</td>
<td>1.8000</td>
<td>0.0846</td>
<td>3.8000</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.1200</td>
<td>0.0161</td>
<td>-0.1200</td>
<td>0.3646</td>
<td>1.8800</td>
<td>0.0621</td>
<td>3.8800</td>
<td>0.0000</td>
</tr>
<tr>
<td>-2.0400</td>
<td>0.0178</td>
<td>-0.0400</td>
<td>0.3687</td>
<td>1.9600</td>
<td>0.0487</td>
<td>3.9600</td>
<td>0.0000</td>
</tr>
</tbody>
</table>