`kernel`: a GAUSS library for kernel estimation

Ruud H. Koning*

July 13, 1996

1 Introduction

The GAUSS-library `kernel` consists of the following files:

- `kfunctio.src`,
- `kernel.src`,

In this documentation we discuss installation of the library and the functions that are included. This manual assumes that the reader is familiar with kernel estimation. The methods implemented in this library are based on Härdle (1990) and Silverman (1986).

The library has three main procedures: `ukernel`, `mkernel` and `nw`. The procedure `ukernel` can be used to estimate the density of univariate data, the procedure `mkernel` estimates the density function of multivariate data. Finally, the procedure `nw` can be used to do nonparametric regression according to the Nadaraya-Watson method.

The kernel library is installed by the following steps:

1. delete the file `kernel.lcg` in the `\gauss\lib` directory (if this file exists),
2. copy the files `kfunctio.src` and `kernel.src` to any directory (for example `\gauss\src\kernel`),
3. start gauss and change the active directory to the subdirectoy that contains the files of the kernel library, for example by giving the command

   ```
   >> dos cd c:\gauss\src\kernel,
   ```

4. check if your `src` directory contains the file `pgraph.ext`. If this is not the case, you do probably do not have the `pgraph`-library installed. You will not be able to use the procedure `view_ukernel` and you must comment the line `#include pgraph.ext` in the file `kernel.src` out.
5. create the kernel library with the commands

*The latest version of this library and its documentation can be found on the World-Wide-Web page of the author: `www.xs4all.nl/~rhkoning/gauss.htm`. Please send comments and suggestions to `rhkoning@xs4all.nl`.

```
>>lib kernel kfunctio.src /addpath
>>lib kernel kernel.src /addpath
```

You should now find a file `kernel.lcg` in your `gauss\lib` subdirectory. The procedures of this library can now be used in your programs by activating the `kernel`- and `pgraph`-library using the gauss statement `lib kernel,pgraph;`

2 Procedures of the `kernel` library

2.1 `bandw1`

format:

```
h=bandw1(y);
```

input:

`y`: n-vector with datapoints.

output:

`h`: bandwidth (scalar) calculated according to

$$h = \frac{0.9 \min(s, IQR/1.34)}{n^{1/5}} \tag{1}$$

with s the sample standard deviation of the datapoints and IQR the interquartile range of the data points (see Silverman (1986), eq. (3.31)).

2.2 `k_bw` (biweight kernel)

format:

```
{f,d}=k_bw(u);
```

input:

`u`: $n \times k$-matrix with datapoints.

output:

`f`: $n \times k$-matrix with biweight kernel function evaluated in `u`.

`d`: $n \times k$-matrix with the derivative of the biweight kernel function evaluated in `u`.

The biweight kernel function is defined as

$$K(u) = \frac{15}{16}(1-u^2)^2 I_{|u|\leq 1} \tag{2}$$

and its derivative is

$$K'(u) = -\frac{15}{4}u(1-u^2)I_{|u|\leq 1}. \tag{3}$$

2.3 k_epan (Epanechnikov kernel)

format:

```
{f,d}=k_epan(u);
```

input:

u: $n \times k$-matrix with datapoints.

output:

f: $n \times k$-matrix with Epanechnikov kernel function evaluated in u.

d: $n \times k$-matrix with the derivative of the Epanechnikov kernel function evaluated in u.

The Epanechnikov kernel function is defined as

$$K(u) = \frac{3}{4\sqrt{5}}(1 - \frac{1}{5}u^2)I_{|u|\leq\sqrt{5}} \tag{4}$$

and its derivative is

$$K'(u) = \frac{3}{10\sqrt{5}}uI_{|u|\leq\sqrt{5}} \tag{5}$$

2.4 k_gauss (Gaussian kernel)

format:

```
{f,d}=k_gauss(u);
```

input:

u: $n \times k$-matrix with datapoints.

output:

f: $n \times k$-matrix with Gaussian kernel function evaluated in u.

d: $n \times k$-matrix with the derivative of the Gaussian kernel function evaluated in u.

The Gaussian kernel function is defined as

$$K(u) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}u^2) \tag{6}$$

and its derivative is

$$K'(u) = -\frac{u}{\sqrt{2\pi}}\exp(-\frac{1}{2}u^2). \tag{7}$$

2.5 `k_rect` (rectangular kernel)

format:

```
{f,d}=k_rect(u);
```

input:

`u`: $n \times k$-matrix with datapoints.

output:

`f`: $n \times k$-matrix with the rectangular kernel function evaluated in `u`.

`d`: $n \times k$-matrix with the derivative of the rectangular kernel function evaluated in `u`.

The rectangular kernel function is defined as

$$K'(u) = \frac{1}{2} I_{|u| \leq 1} \tag{8}$$

and its derivative is 0 everywhere.

2.6 `k_trian` (triangular kernel)

format:

```
{f,d}=k_trian(u);
```

input:

`u`: $n \times k$-matrix with datapoints.

output:

`f`: $n \times k$-matrix with the triangular kernel function evaluated in `u`.

`d`: $n \times k$-matrix with the derivative of the triangular kernel function evaluated in `u`.

The triangular kernel function is defined as

$$K(u) = (1 - |u|) I_{|u| \leq 1} \tag{9}$$

and its derivative is

$$K'(u) = -I_{0 < u \leq 1} + I_{-1 \leq u < 0}. \tag{10}$$

2.7 `mkernel`

format:

```
{f,d,h}=mkernel(x,z,h,w,&kf);
```

input:

`x`: $T \times k$-matrix with points where the k variate kernel is estimated.

`z`: $n \times k$-matrix with observed data points.

h: scalar, bandwidth. The same bandwidth is used for all k components.If `h=0` the bandwidth is calculated using

$$h = \left(\frac{4}{k+2}\right)^{1/(k+4)} n^{-1/(k+4)} \tag{11}$$

(see Silverman (1986), eq. (4.14)).

w: n-vector with weights.

&kf: pointer to a univariate kernel function that takes a datamatrix `z` as an argument and returns both the kernel function evaluated in `z` and the derivative of the kernel function evaluated in `z` (`{func,der} = kf(z)`).

output:

f: T-vector, the estimated density evaluated in `x`.

d: $T \times k$-matrix, the estimated derivative of the density evaluated in `x`.

h: scalar, with the bandwidth used to calculate `f` and `d`.

The density is estimated according to

$$\hat{f}(x_j) = \frac{1}{nh}\sum_{i=1}^{n} w_i K_m(\frac{x_j - z_i}{h}), \qquad j = 1, \ldots, T. \tag{12}$$

and the derivative is calculated as

$$\hat{f}'(x_j) = \frac{1}{nh^{1+k}}\sum_{i=1}^{n} w_i K_m'(\frac{x_j - z_i}{h}), \qquad j = 1, \ldots, T. \tag{13}$$

The multivariate kernel function $K_m(\cdot)$ is the product of (identical) univariate kernel functions: $K_m(y) = \prod_{i=1}^{k} K(y_i)$ with y_i the i-th element of the k-vector y. The pointer `&kf` points to $K(\cdot)$.

2.8 `nw`

format:

`{g,d,h}=nw(x,z,y,h,&kf)`

input:

x: T-vector with points where the regression function is estimated.

z: n-vector with observed data points of regressor.

y: n-vector with observed data points of dependent variable.

h: scalar, bandwidth. If `h=0` the bandwidth is calculated using the procedure `bandw1`.

&kf: pointer to kernel function that takes a datavector `z` as an argument and returns both the kernel function evaluated in `z` and the derivative of the kernel function evaluated in `z` (`{func,der} = kf(z)`).

output:

`g`: T-vector, estimated regression function evaluated in `x`.

`d`: T-vector, estimated derivative of the regression function evaluated in `x`.

`h`: scalar, the bandwidth used to calculate `g` and `d`.

The regression function is estimated according to

$$\hat{m}(x_j) = \frac{n^{-1}\sum_{i=1}^{n} y_i K(\frac{x_j - z_i}{h})}{n^{-1}\sum_{i=1}^{n} K(\frac{x_j - z_i}{h})}, \qquad j = 1, \ldots, T. \tag{14}$$

and the derivative is estimated as

$$\hat{m}'(x_j) = \frac{1}{nh^2}\sum_{i=1}^{n} K'(\frac{x_j - z_i}{h}) y_i \qquad j = 1, \ldots, T. \tag{15}$$

2.9 `ukernel`

format:

`{f,d,h}=ukernel(x,z,h,w,&kf);`

input:

`x`: T-vector with points where the kernel is estimated.

`z`: n-vector with observed data points.

`h`: scalar, bandwidth. If `h=0` the bandwidth is calculated using the procedure `bandw1`.

`w`: n-vector with weights.

`&kf`: pointer to kernel function that takes a datavector `z` as an argument and returns both the kernel function evaluated in `z` and the derivative of the kernel function evaluated in `z` (`{func,der} = kf(z)`).

output:

`f`: T-vector, the estimated density evaluated in `x`.

`d`: T-vector, the estimated derivative of the density evaluated in `x`.

`h`: scalar, the bandwidth used to calculate `f` and `d`.

The density is estimated according to

$$\hat{f}(x_j) = \frac{1}{nh}\sum_{i=1}^{n} w_i K(\frac{x_j - z_i}{h}), \qquad j = 1, \ldots, T. \tag{16}$$

and the derivative is calculated as

$$\hat{f}'(x_j) = \frac{1}{nh^2}\sum_{i=1}^{n} w_i K'(\frac{x_j - z_i}{h}), \qquad j = 1, \ldots, T. \tag{17}$$

2.10 `view_ukernel`

format:

`call view_ukernel(x,f,h);`

input:

`x`: T-vector with points where the kernel is estimated.

`f`: T-vector with estimated density in `x`.

`h`: scalar, bandwidth used to estimate `f`.

global variables:

All global variables of the `pgraph`-library can be used to create the graph.

remark:

This procedure works only in the DOS-version of gauss, it will not work in the UNIX-version! It is assumed that the `pgraph`-library has been activated before calling this procedure.

3 Examples

In a first example, we draw random numbers from a normal distribution and then we estimate the density function of these random numbers. To do so, we use the Gaussian kernel, the Epanechnikov kernel, and the rectangular kernel.

```
/* in this example we draw 1000 random numbers from a normal distribution
and we estimate the density using the kernel library. Three different graphs
will be created using different kernel funtions.
Note that the pgraph library must be activated in order to use view_ukernel.
*/

library kernel,pgraph;
nobs=1000;

r=rndn(nobs,1);

x_axis=seqa(-3,6/1000,1000);
{f1,d1,h1}=ukernel(x_axis,r,0,1,&k_gauss);
call view_ukernel(x_axis,f1,h1);
{f2,d2,h2}=ukernel(x_axis,r,0,1,&k_epan);
call view_ukernel(x_axis,f2,h2);
{f3,d3,h3}=ukernel(x_axis,r,0,1,&k_rect);
call view_ukernel(x_axis,f3,h3);
end;
```

References

Härdle, W. (1990). *Applied Nonparametric Regression*. Cambridge: Cambridge University Press.

Silverman, B.W. (1986). *Density Estimation*. London: Chapman and Hall.

A History

13 july 1996 version 1.0 documented and released on WWW.

B GAUSS Code

B.1 kernel.src

```
/* kernel.src version 1.0 13 july 1996
Source file with functions of the kernel library:
    ukernel     procedure for univariate kernel estimation
    mkernel     procedure for multivariate kernel estimation
    bandw1      procedure for automatic bandwidth selection
    nw          procedure for nonparametric regression
    viewuknl    procedere for plotting the estimated density

*/

/* comment the following line out if you don't have the pgraph library
installed. You will not be able to use the procedure view_ukernel */
#include pgraph.ext

/* procedure to calculate a univariate kernel density estimate and its
derivative
usage:      {f, d, h} = ukernel(x, z, h, w, &kf);
input:      x:      T vector where density is to be estimated
            z:      n vector with observed data points
            h:      scalar bandwidth, if h<=0 bandwidth is determined by
                    procedure bandw1(z)
            w:      n vector with weights
            &kf:    pointer to weighting function
output:     f:      T vector with estimated density
            d:      T vector with estimated derivative
            h:      scaler bandwidth
*/

proc (3)=ukernel(x, z,h,w, &kf);
 local arg, i, n, f, d, k, kff, kfd, pkff,argl,argh,kffl,kffh,fl,fh,
  hl,hh;
 local kf:proc;

 /* error checking here */
 if (cols(x)>1);
  errorlog "ukernel.g: x has too many columns";
  retp(-1,-1,-1 );
 endif;
 if (cols(z)>1);
  errorlog "ukernel.g: z has too many columns";
  retp( -1,-1,-1 );
 endif;

 /* initialization */
 i = 1;
 n = rows(x);
 k = cols(x);
```

```
 /* determine bandwidth */
 if (h<=0);
  h=bandw1(z);
 endif;

 f=zeros(n,1);
 d=zeros(n,1);
 do while (i<=n);
  arg = (x[i]-z)/h;
  {kff, kfd} = kf(arg);
  f[i] = meanc(kff.*w)/h;
  d[i] = meanc(kfd.*w)/(h^2);
  i=i+1;
 endo;
 retp( f,d,h );
endp;


/* procedure to calculate a multivariate kernel density estimate and its
derivative
usage:      {f, d, h} = mkernel(x, z, h, w, &kf);
input:      x:      T x k matrix where density is to be estimated
            z:      n x k matrix with observed data points
            h:      scalar, bandwidth parameter (same bandwidth for all
                    components), if h<=0 it is determined by eq 4.14 of
                    Silverman
            w:      n vector with weights
            &kf:    pointer to weighting function
output:     f:      T vector with estimated density
            d:      T x k matrix with estimated derivatives
            h:      scalar, bandwidth used
*/

proc (3)=mkernel(x, z, h, w, &kf);
 local arg,i,n,f,d,k,kff,kfd,pkff,p,deneq0;
 local kf:proc;

 i = 1;
 n = rows(x);
 k = cols(x);
 if (h<=0);
   p=1/(k+4);
   h=(4/(k+2))^p*n^(-p);
 endif;
 f = zeros(n,1);
 d = zeros(n,k);
 do while (i<=n);
  arg = (x[i,.]-z)/h;
  {kff, kfd} = kf(arg);
  pkff = prodc(kff');
  f[i] = meanc(pkff.*w)/(h^k);
  deneq0=kff .==0;
  kff=kff+deneq0;
  d[i,.] = meanc((kfd.*pkff.*w)./kff)'/(h^(k+1));
  i = i+1;
 endo;
```

```
 retp( f,d,h );
endp;


/* computes Nadaraya-Watson kernel regression estimator

usage:  {g,d,h} = nw(x,z,y,h,&kf);
input:  x:          T-vector where regression function is evaluated
        z:          n-vector with data on independent variable
        y:          n-vector with data on dependent variable
        h:          scalar, bandwidth, if h<=0 the bandwidth is determined
                    automatically using procedure bandw1
        &kf:        pointer to univariate kernel density estimator
output: g:          T-vector with values of regression evaluated in x
        d:          T-vector with estimated derivative in x
        h:          scalar bandwidth used
*/

proc (3)=nw(x, z, y,h,&kf);
    local num, numder, denom, denomder, g, s, n, k, d,v;
    local kf: proc;

 n = rows(x);
 k = cols(x);
 g = zeros(n,1);
 d = zeros(n,1);

 if (k ne 1);
  print "error in nw.g: too many columns in x";
  retp( g, d );
 endif;
 if (cols(z) ne 1);
  print "error in nw.g: too many columns in z";
  retp( g, d );
 endif;

 {num,numder,h} = ukernel(x,z,h,y,&kf);
 {denom,denomder,h} = ukernel(x,z,h,1,&kf);
 s = (denom.==0);
 g = num./(denom+s);
 d = numder./(denom+s) - num.*denomder./(denom^2+s);
 retp( (1-s).*g, (1-s).*d,h );
endp;

/* bandw1
procedure to calculate the optimal bandwidth in kernel estimation of a density.
The optimal bandwidth is calculated according to eq. 3.31 of Silverman (1986)

usage:      h=bandw1(y);
input:      y:      n-vector whose density will be estimated;
output:     h:      scalar, optimal bandwidth choice;

*/

proc bandw1(y);
 local s, ys,n, a, iqr, qi1, qi3;
```

```
 if (cols(y)>1);
  errorlog "input error in bandw1.g: too many columns";
  retp( -1 );
 endif;
 s=sqrt(vcx(y));
 n=rows(y);
 ys=sortc(y,1);
 qi1=round(0.25*n);
 qi3=round(0.75*n);
 iqr=ys[qi3]-ys[qi1];
 retp( 0.9*minc(s|(iqr/1.34))/n^0.2 );
endp;

/* view_ukernel
procedure to plot kernel density estimate. Library pgraph must be activated.

usage: call view_ukernel(x,f,h);
input:      x:  n-vector with data points
            f:  n x k matrix (k=1 or k=3) with estimated densities for
                different bandwidths
            h:  k-vector (k=1 or k=3) with bandwidths
output:     none
globals:    all globals of the pgraph library

*/

proc (0)=view_ukernel(x,f,h);
 local data,k,xlow,xhigh,flow,fhigh,xlegend,ylegend;

 /* error checking */
 if (cols(f) ne rows(h));
  errorlog "error in viewkrnl.g: rows(f) unequal to cols(h)";
  retp();
 endif;
 /* set global variables pgraph */
 _pdate=0;

 k=cols(f);
 xlow=minc(x);
 xhigh=maxc(x);
 flow=minc(minc(f));
 fhigh=maxc(maxc(f));
 xlegend=xlow;
 ylegend=flow+0.8*(fhigh-flow);
 _plegctl=1|4|xlegend|ylegend;
 if (k==1);
  _plegstr="h=" $+ ftos(h,"%*.*lf",5,3);
  else;
  _plegstr="h=" $+ ftos(h[1],"%*.*lf",5,3) $+ "\0h="
   $+ftos(h[2],"%*.*lf",5,3)$+"\0h="$+ ftos(h[3],"%*.*lf",5,3);
 endif;
 data=sortc(x~f,1);
 xy(data[.,1],data[.,2:cols(data)]);
endp;
```

B.2 `kfunctio.src`

```
/* This file contains some kernel functions. All functions take an
n x k matrix u as their argument and return an n x k matrix with the
function evaluated in each point of x and an n x k matrix d with the
derivative of the function in each point of x.
    k_bw:     biweight kernel function
    k_epan:   Epanechnikov kernel
    k_gauss:  Gaussian kernel
    k_triang: triangular kernel
    k_rect:   rectangular kernel

usage:
    {f,d}=k_bw(u);
*/

proc (2)=k_bw(u);
 local select;
 select = abs(u).<=1;
 retp( 15/16*((1-u.^2).^2).*select, -15/4*u.*(1-u.^2).*select);
endp;

proc (2)=k_gauss(u);
    retp( pdfn(u), -u.*pdfn(u) );
endp;

proc (2)=k_epan(u);
    local c, s;
    s = abs(u).<sqrt(5);
    c = 0.75/sqrt(5);
    retp( c*(1-0.2*u.^2).*s, -0.4*c*u.*s );
endp;

proc (2)=k_rect(u);
    retp( 0.5*(abs(u).<1), 0*u );
endp;

proc (2)=k_trian(u);
    local s, a;
    a = abs(u);
    s = a.<1;
    retp( (1-a).*s, (-(u.>=0) + (u.<=0)).*s );
endp;
```