Subject: Re: kernel density estimator routine
Posted by David Grier on Tue, 28 Jun 2011 22:04:30 GMT
View Forum Message <> Reply to Message

Hi Dan,

The following routines should do what you want. References to the relevant literature are in the comments. There are three files:

akde.pro: adaptive kernel density estimator for N-dimensional data.
This calls ...

kde.pro: one- and N-dimensional kernel density estimators for
fixed (optimal) scale factors. This calls ...

iqr.pro: inter-quartile range of a one-dimensional data set.

All the best,

David

```
;=== snip here for akde.pro ===
;+
; NAME:
;    AKDE
;
; PURPOSE:
;    Estimate the probability density underlying a set of discrete
;    samples (measurements) using the adaptive kernel density estimator
method.
;
; CATEGORY:
;    Statistics
;
; CALLING SEQUENCE:
;    d = akde(x, t)
;
; INPUTS:
;    x: discrete samples of the desired distribution.
;
;    t: values for which the probability density is required.
;
; KEYWORD FLAGS:
;    By default, KDE uses the Epanechnikov kernel to compute
;    the kernel density estimate, this can be overridden by
;    setting one of the following flags:
;    GAUSSIAN: use Gaussian kernel
;    TRIANGULAR: use triangular kernel
```

```
;    BIWEIGHT: use biweight kernel
;
; OUTPUTS:
;    d: probability density estimated at each value specified by t
;
; RESTRICTIONS:
;    Multivariate estimates are only computed with Gaussian kernels.
;
; PROCEDURE:
;    d_i = (1/n) \sum_{j = 1}^n K((x_j - t_i)/h)
;
;    where h is the estimated optimal smoothing parameter and
;    where K(z) is the selected kernel.
;
; REFERENCE:
; B. W. Silverman,
; Density Estimation for Statistics and Data Analysis
; (CRC Press, Boca Raton, 1998)
;
; EXAMPLE:
;    IDL> x = randomn(seed, 1000)
;    IDL> t = 2. * findgen(100)/99.
;    IDL> d = akde(x,t)
;    IDL> plot, t, d
;    IDL> plot, t, histogram(x, min=0, max=2, nbins=100), /noerase
;
; MODIFICATION HISTORY:
; 09/18/2010 Written by David G. Grier, New York University
; 09/26/2010 DGG: Added COMPILE_OPT.  First version of AKDE_ND
;
; Copyright (c) 2010 David G. Grier
;
;-

function akde_nd, x, y

COMPILE_OPT IDL2, HIDDEN

sx = size(x, /dimensions)
sy = size(y, /dimensions)

nd = sx[0]                ; number of dimensions
nx = sx[1]                ; number of data points
ny = sy[1]                ; number of sampling points

fac = replicate(1., nx)

; Method described by Silverman Sec. 5.3.1
```

```
; 1. pilot estimate of the density at the data points
f = kde(x, x, scale = h)

; 2. local bandwidth factor
alpha = 0.5                     ; sensitivity parameter: 0 <=
alpha <= 1
g = exp(mean(alog(f), dimension = 1)) ; geometric mean density
lambda = ((g # fac)/f)^alpha        ;  Eq. (5.7): factor for each data
point

; 3. adaptive density estimate
res = fltarr(ny, /nozero)
hfac = (h # fac)              ; smoothing factor for each point

for j = 0, ny - 1 do begin
   z = total(0.5 * ((x - y[*,j] # fac) / (hfac * lambda))^2 , 1)
   res[j] = mean(exp(-z))
endfor

res[j] /= 2. * !pi * total(h^2)^(nd/2) ; normalization
return, res
end

function akde_1d, x, y, $
                  biweight = biweight, $
                  triangular = triangular, $
                  gaussian = gaussian

COMPILE_OPT IDL2, HIDDEN

nx = n_elements(x)            ; number of data points
ny = n_elements(y)            ; number of samples

; Method described by Silverman Sec. 5.3.1
; 1. pilot estimate of the density at the data points
f = kde(x, x, scale = h, $
       biweight=biweight, triangular=triangular, gaussian=gaussian)

; 2. local bandwidth factor
alpha = 0.5                   ; sensitivity parameter: 0 <= alpha <= 1
g = exp(mean(alog(f)))         ; geometric mean density
lambda = (g/f)^alpha          ; Eq. (5.7)

; 3. adaptive density estimate
t = x / h
s = y / h
res = findgen(ny)
if keyword_set(biweight) then begin
```

```
    for j = 0, ny-1 do begin
      z = ((t - s[j])/lambda)^2
      w = where(z lt 1.)
      res[j] = 15. * total((1. - z[w])^2/lambda[w]) / (h * 16. * nx)
    endfor
endif $
else if keyword_set(triangular) then begin
    for j = 0, ny-1 do begin
      z = abs(t - s[j])/lambda
      w = where(z lt 1.)
      res[j] = total((1. - z[w])/lambda[w]) / (h * nx)
    endfor
endif $
else if keyword_set(gaussian) then begin
    for j = 0, ny-1 do begin
      z = 0.5 * ((t - s[j])/lambda)^2
      res[j] = mean(exp(-z)/lambda) / (h * sqrt(2.*!pi))
    endfor
endif $
else begin                      ; Epanechnikov
    for j = 0, ny-1 do begin
      z = ((t - s[j])/lambda)^2
      w = where(z lt 5)
      res[j] = 0.75 * total((1. - z[w]/5)/lambda[w]) / (h * sqrt(5.) * nx)
    endfor
endelse

return, res
end


function akde, x, y, $
               gaussian = gaussian, $
               biweight = biweight, $
               triangular = triangular

COMPILE_OPT IDL2

sx = size(x)
sy = size(y)

if sx[0] gt 2 then $
    message, "data must be organized as [ndims,npoints]"

if sy[0] ne sx[0] then $
    message, "inputs must have the same number of dimensions"

if (sx[0] eq 2) and (sx[1] ne sy[1]) then $
```

```
    message, "inputs must have the same number of dimensions"

ndims = (sx[0] eq 2) ? sx[1] : 1

if ndims gt 1 then $
    return, akde_nd(x, y) $
else $
    return, akde_1d(x, y, $
                    gaussian = gaussian, $
                    biweight = biweight, $
                    triangular = triangular)
end

;=== snip here for kde.pro ===
;+
; NAME:
;    KDE
;
; PURPOSE:
;    Estimate the probability density underlying a set of discrete
;    samples (measurements) using the kernel density estimator method.
;
; CATEGORY:
;    Statistics
;
; CALLING SEQUENCE:
;    d = kde(x, t)
;
; INPUTS:
;    x: discrete samples of the desired distribution.
;
;    t: values for which the probability density is required.
;
; KEYWORD PARAMETERS:
;    scale: smoothing factor, also called the bandwidth
;        used to compute the kernel density estimate
;
; KEYWORD FLAGS:
;    By default, KDE uses the Epanechnikov kernel to compute
;    the kernel density estimate, this can be overridden by
;    setting one of the following flags:
;    GAUSSIAN: use Gaussian kernel
;    TRIANGULAR: use triangular kernel
;    BIWEIGHT: use biweight kernel
;
; OUTPUTS:
;    d: probability density estimated at each value specified by t
;
```

```
; RESTRICTIONS:
;    Gaussian kernel used for multivariate systems.
;
; PROCEDURE:
;    d_i = (1/n) \sum_{j = 1}^n K((x_j - t_i)/h) / h
;
;    where h is the estimated optimal smoothing parameter and
;    where K(z) is the selected kernel.
;
; REFERENCE:
; B. W. Silverman,
; Density Estimation for Statistics and Data Analysis
; (CRC Press, Boca Raton, 1998)
;
; EXAMPLE:
;    IDL> x = randomn(seed, 1000)
;    IDL> t = 2. * findgen(100)/99.
;    IDL> d = kde(x,t)
;    IDL> plot, t, d
;    IDL> plot, t, histogram(x, min=0, max=2, nbins=100), /noerase
;
; MODIFICATION HISTORY:
; 09/18/2010 Written by David G. Grier, New York University
;
; Copyright (c) 2010 David G. Grier
;
;-

function kde_nd, x, y, $
                 scale = scale

COMPILE_OPT IDL2, HIDDEN

sx = size(x, /dimensions)
sy = size(y, /dimensions)

nd = sx[0]                  ; number of dimensions
nx = sx[1]                  ; number of data points
ny = sy[1]                  ; number of sampling points

; optimal smoothing parameter in each dimension
; Silverman Eqs. (3.30) and (3.31)
sx = stddev(x, dimension=2)
rx = fltarr(nd)
for d = 0, nd-1 do $
    rx[d] = iqr(x[d,*])
h = 0.9 * (sx < rx/1.34) / nx^0.2
scale = h
```

```
; density estimate
; Silverman Eq. (2.15) and Table 3.1
fac = replicate(1., nx)
res = fltarr(ny, /nozero)
hfac = h # fac
for j = 0, ny-1 do begin
   z = total(0.5 * ((x - y[*,j] # fac) / hfac)^2, 1)
   res[j] = mean(exp(-z))
endfor

res /= 2. * !pi * total(h^2)^(nd/2) ; normalization

return, res
end

function kde_1d, x, y, $
                 scale = scale, $
                 biweight = biweight, $
                 triangular = triangular, $
                 gaussian = gaussian

COMPILE_OPT IDL2, HIDDEN

nx = n_elements(x)             ; number of data points
ny = n_elements(y)             ; number of samples

; optimal smoothing parameter
; Silverman Eqs. (3.30) and (3.31)
sx = stddev(x)                 ; standard deviation
rx = iqr(x)                    ; interquartile range
h = 0.9 * (sx < rx/1.34) / nx^0.2
scale = h

; density estimate
; Silverman Eq. (2.15) and Table 3.1
t = x/h
s = y/h
res = fltarr(ny)               ; result

if keyword_set(biweight) then begin
   for j = 0, ny-1 do begin
      z = (t - s[j])^2
      w = where(z lt 1.)
      res[j] = 15. * total((1. - z[w])^2) / (h * 16. * nx)
   endfor
endif $
else if keyword_set(triangular) then begin
```

```
    for j = 0, ny-1 do begin
      z = abs(t - s[j])
      w = where(z lt 1.)
      res[j] = total(1. - z[w]) / (h * nx)
    endfor
endif $
else if keyword_set(gaussian) then begin
    for j = 0, ny-1 do begin
      z = 0.5 * (t - s[j])^2
      res[j] = mean(exp(-z)) / (h * sqrt(2.*!pi))
    endfor
endif $
else begin                      ; Epanechnikov
    for j = 0, ny-1 do begin
      z = (t - s[j])^2
      w = where(z lt 5)
      res[j] = 0.75 * total((1. - z[w]/5)) / (h * sqrt(5.) * nx)
    endfor
endelse

return, res
end

function kde, x, y, scale = scale, $
              gaussian = gaussian, $
              biweight = biweight, $
              triangular = triangular

COMPILE_OPT IDL2

sx = size(x)
sy = size(y)

if sx[0] gt 2 then $
    message, "data must be organized as [ndims,npoints]"

if sy[0] ne sx[0] then $
    message, "inputs must have the same number of dimensions"

if (sx[0] eq 2) and (sx[1] ne sy[1]) then $
    message, "inputs must have the same number of dimensions"

ndims = (sx[0] eq 2) ? sx[1] : 1

if ndims gt 1 then begin
    if keyword_set(biweight) or keyword_set(triangular) then $
      message, "Multidimensional: using Gaussian kernel", /inf
    res = kde_nd(x, y, scale = scale)
```

```
endif else $
   res = kde_1d(x, y, scale = scale, $
               gaussian = gaussian, $
               biweight = biweight, $
               triangular = triangular)

return, res
end

;=== snip here for iqr.pro ===
;+
; NAME:
;    IQR
;
; PURPOSE:
;    Computes the Inter-Quartile Range of a one-dimensional data set
;
; CATEGORY:
;    Statistics
;
; CALLING SEQUENCE:
;    r = iqr(x)
;
; INPUTS:
;    x: one-dimensional numerical data set
;
; OUTPUTS:
;    r: inter-quartile range of x
;
; RESTRICTIONS:
;    Considers only finite elements of x.
;    Considers only one-dimensional data sets.
;
; PROCEDURE:
;    The data are sorted into ascending order.  The lower quartile
;    is the median of the first half of the ordered data and the
;    upper quartile is the median of the second half.  The
;    interquartile range is the difference between the upper and
;    lower quartile values.
;
; MODIFICATION HISTORY:
; 09/18/2010 Written by David G. Grier, New York University
;
; Copyright (c) 2010 David G. Grier
;
;-

function iqr, data
```

```
; only consider valid numerical data
w = where(finite(data) eq 1, ndata)
if ndata lt 1 then begin
   message, "no finite data points", /inf
   return, 0.
endif
x = data[w]

min = min(data, max=max)

if min eq max then $
   return, 0.

x = x[sort(x)]
nhi = ndata/2
; if ndata is odd, the middle data point belongs to
; both the upper and lower quartiles ...
nlo = (ndata mod 2) eq 0 ? nhi - 1 : nhi

return, median(x[nhi:*], /even) - median(x[0:nlo], /even)

end

;;; === that's it! ===
```

On 06/28/2011 04:42 PM, Daniel Hartung wrote:
> Does anyone know of a kernel density estimator routine in IDL that is
> similar to the ksdensity funciton in MatLab?
>
> Thanks!
> Dan
