Using Stata® and Python for nonparametric smoothing estimation

Isaías Hazarmabeth Salgado Ugarte

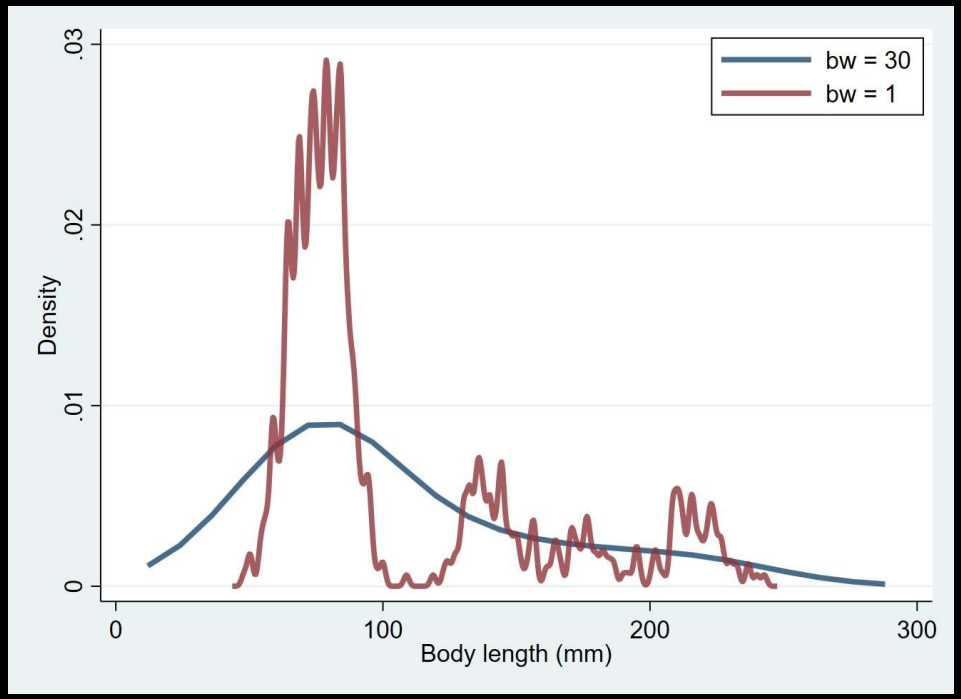
Verónica Mitsui Saito Quezada

Mitsui Myrna Salgado Saito

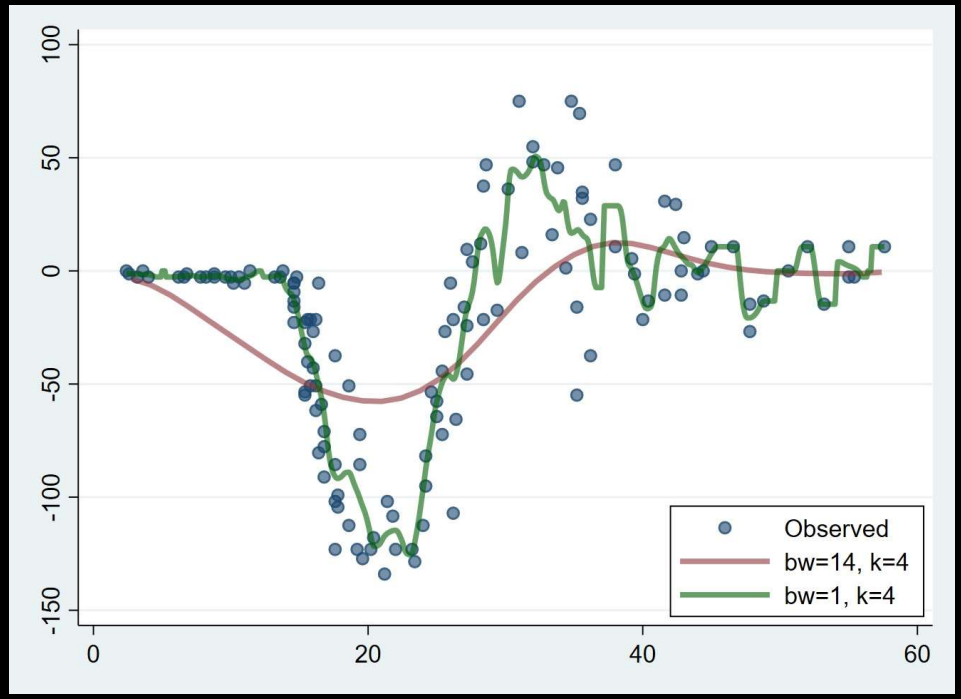
Noel Isaías Plascencia Díaz

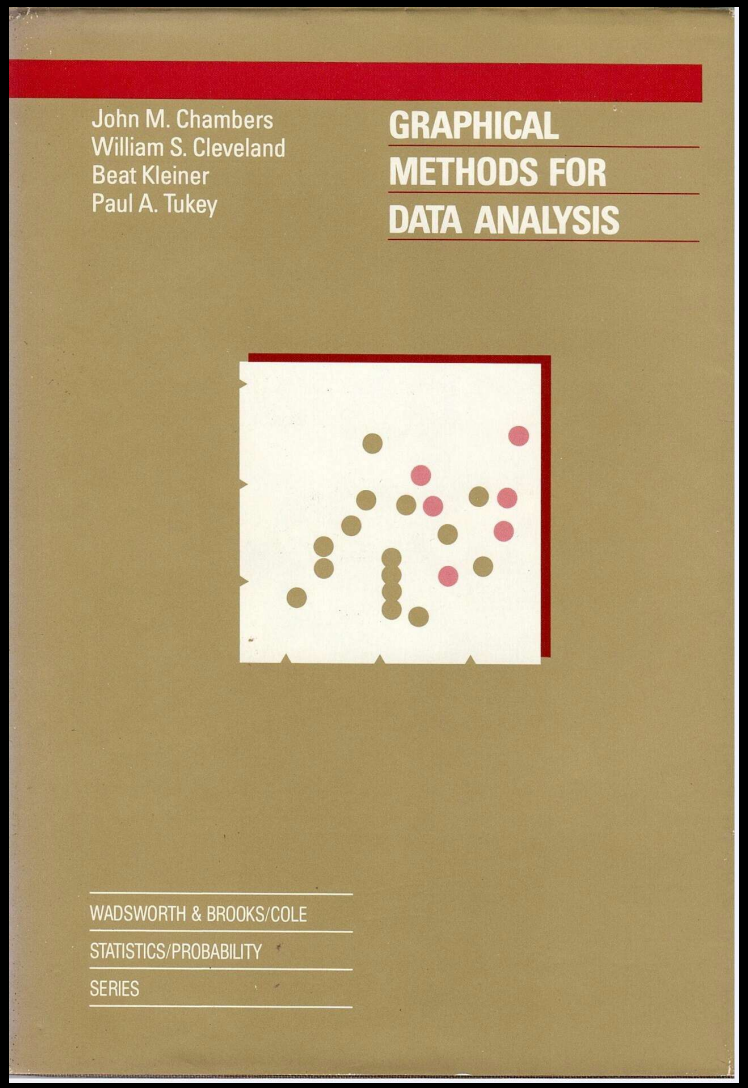
The choice of the bin/bandwidth (smoothing parameter) is one of the most relevant issues of nonparametric density/regression estimation.

Kernel density estimation (Gaussian)

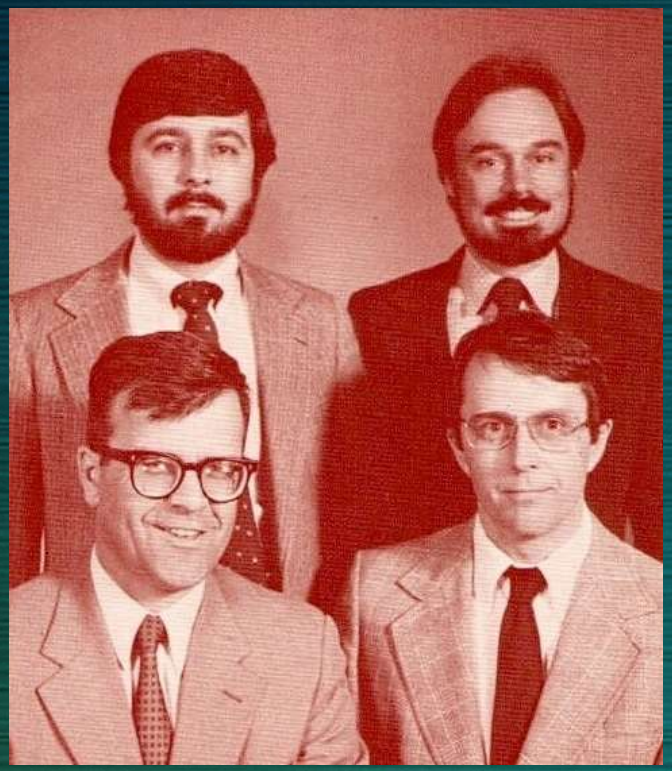


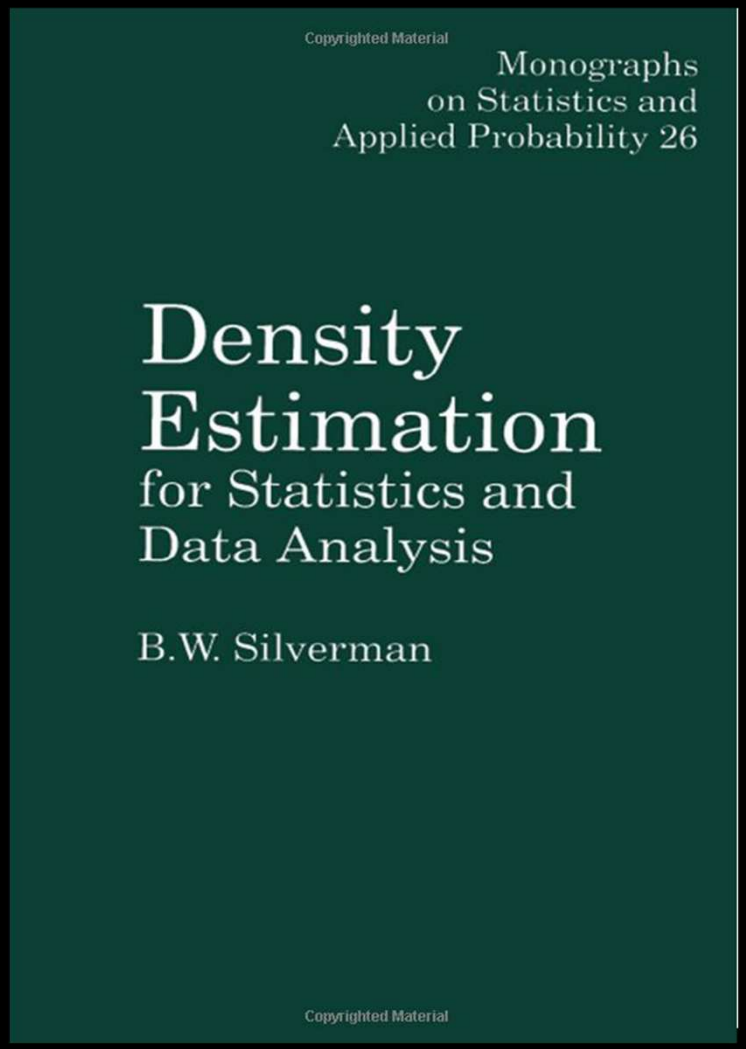
Kernel regression (Quartic)



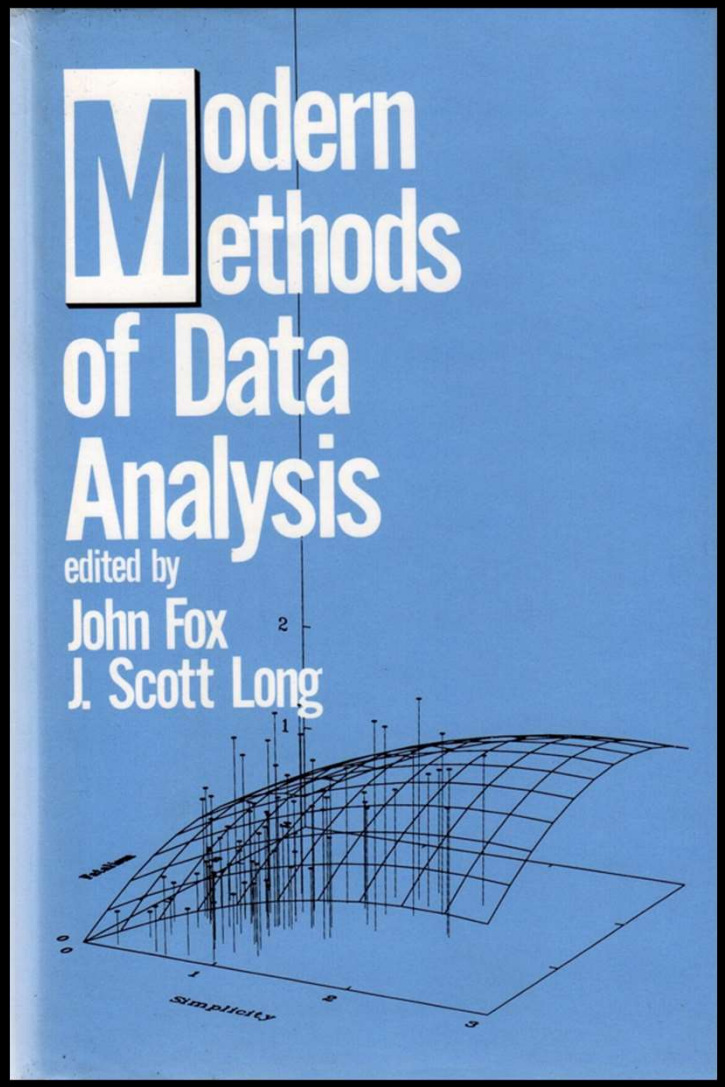


Density traces 1983



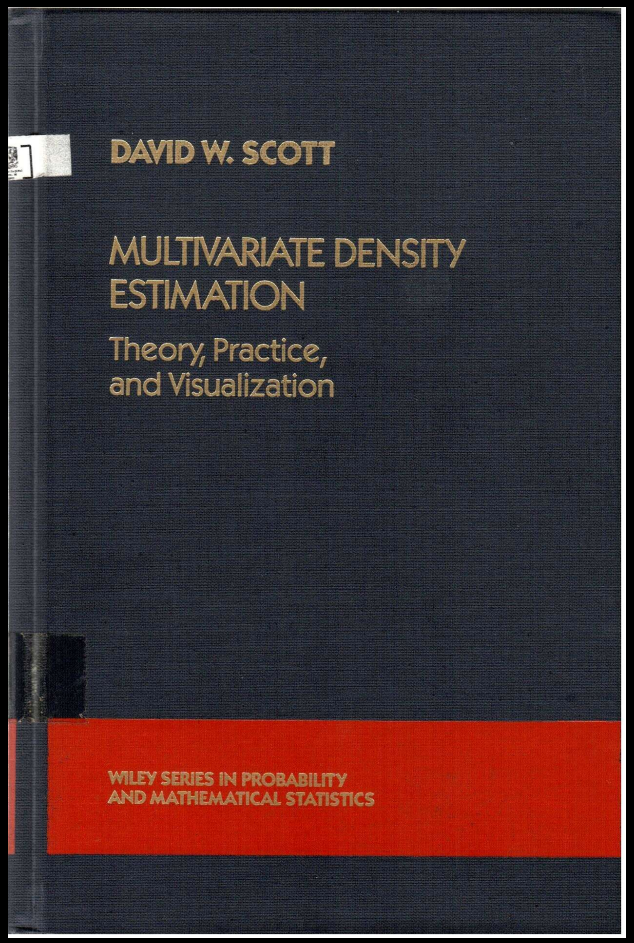


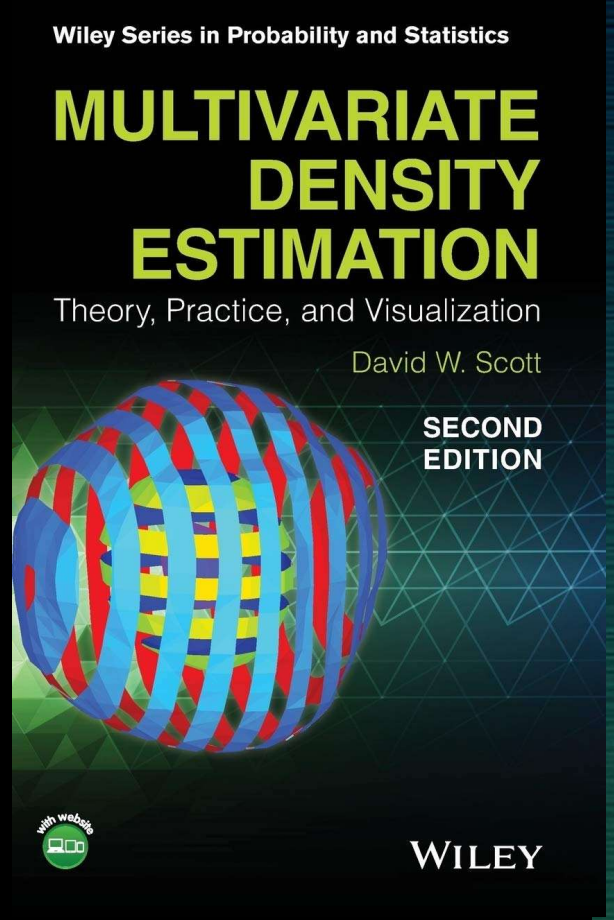
B.W. Silverman, 1986
Kernel density estimators (KDE's)



John Fox, 1990
Chapter 2:
"Describing univariate distributions"

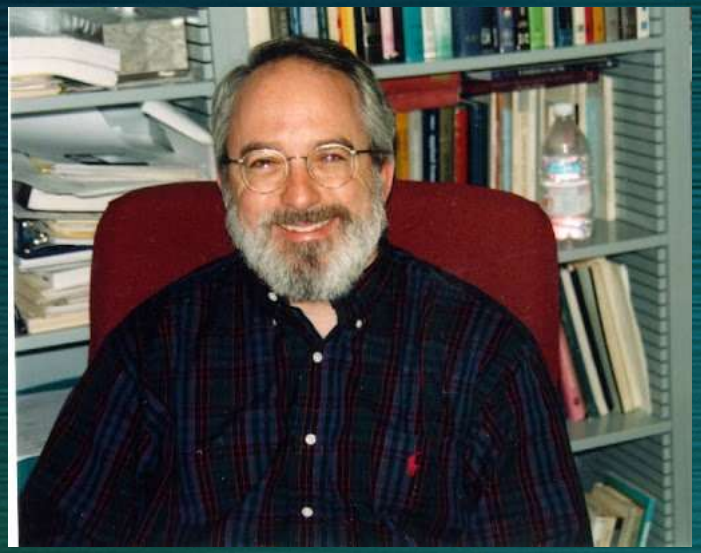
"... Nevertheless, (KDE's) are easily programmed ..."

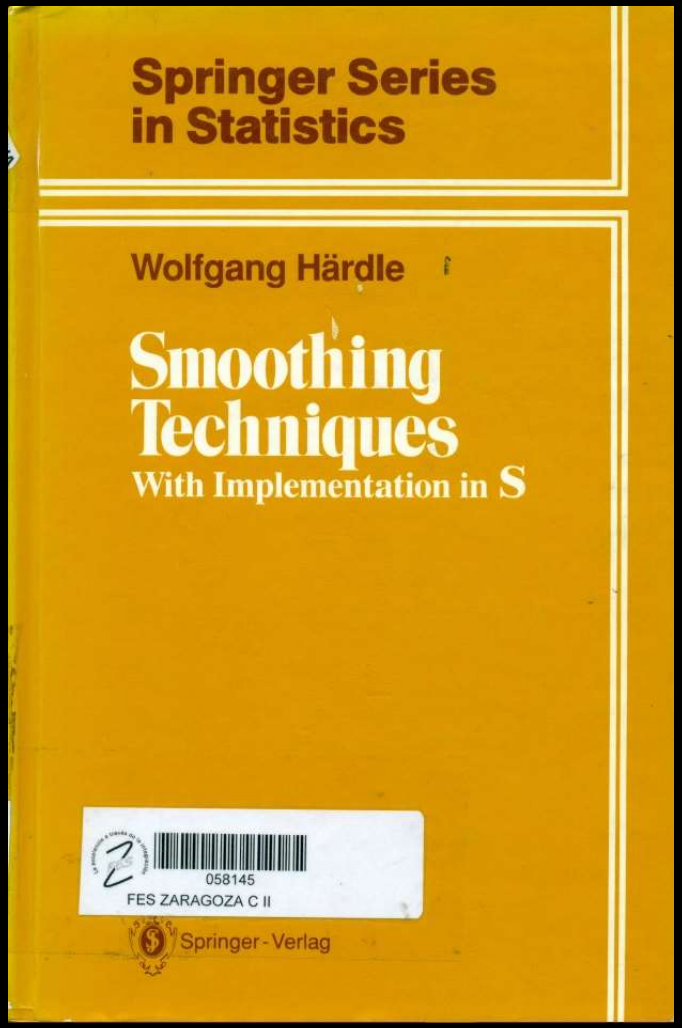




Dr. David W. Scott
1992, 2015

ASH routines in S



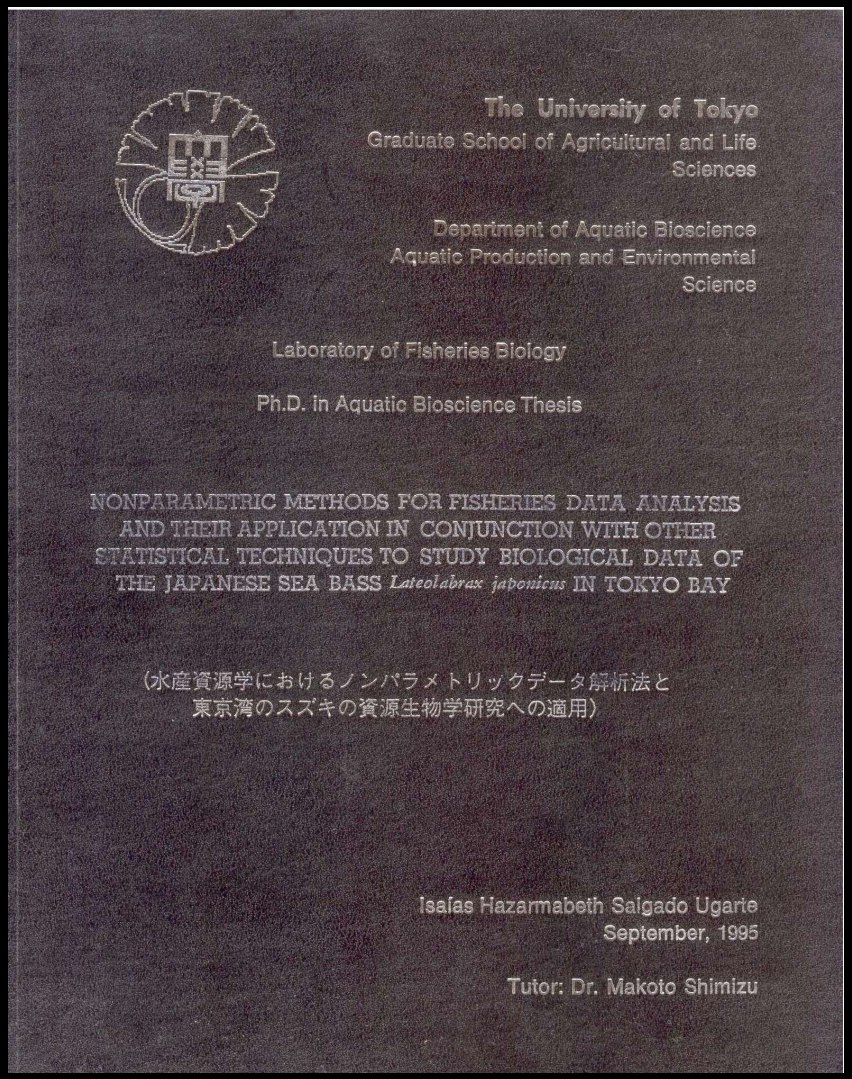


Dr. Wolfgang Härdle
S functions and C routines

Least squares Cross-Validation (C routine)

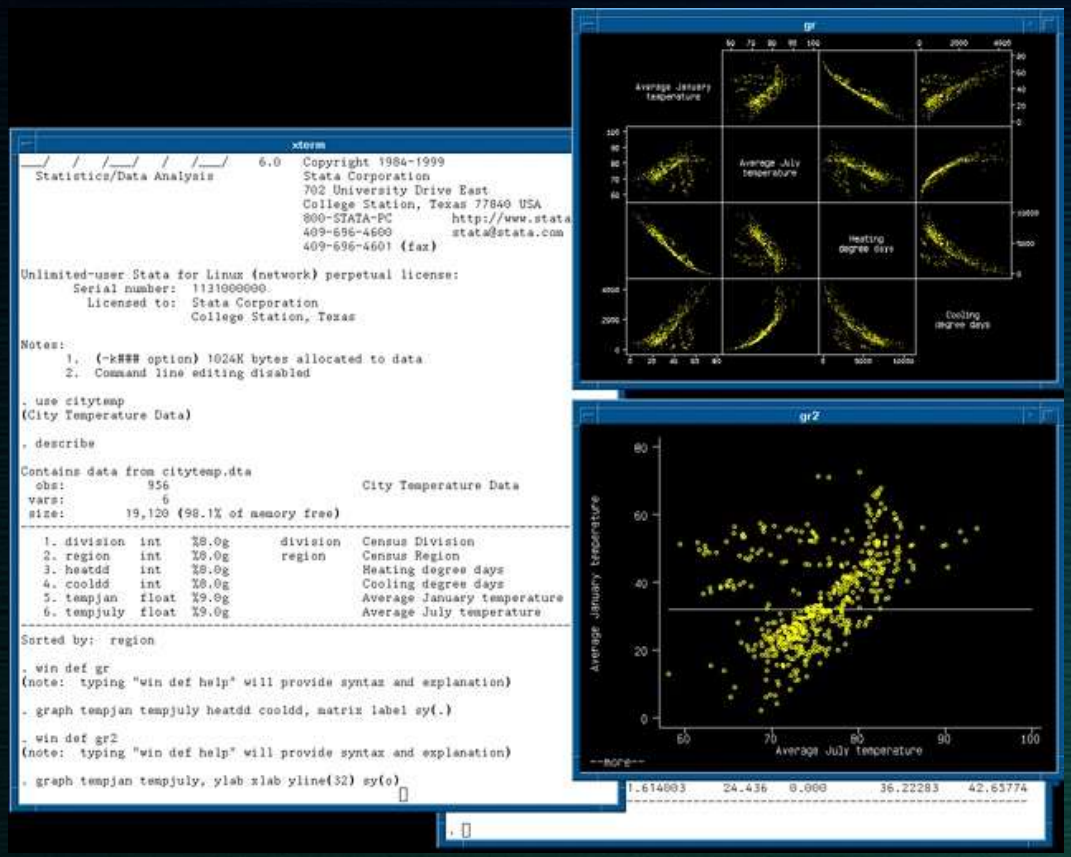
variables. When the routine is finished, we need only the particular estimate. Since we have labelled the belonging variable (by result=...), we can extract this variable with the option $result. Finally, the function manipulates the result to return a matrix to S, which contains the bin midpoints of the small bins and the belonging estimates in these midpoints. The listing of the C routine is as follows.

```
/* C-routine of WARPing-density-estimation for                    */
/* 1-dimensional data with 5 different kernels                    */
#include<stdio.h>
#include<math.h>

warping(x,fh,h,wM,origin,binmesh,counts,J,M,n,kernel)
/* x <==> data                                                    */
/* fh <==> vector == 0 will contain density estimates             */
/* h bandwidth                                                    */
/* wM <==> weights depending on the different kernels             */
/* origin <==> left border of first bin                           */
/* binmesh <==> vector == 0 protocol for non-empty bins           */
/* counts <==> counts of observations in non-empty bins           */
/* J <==> index of non-empty bins                                 */
/* M <==> number of small bins in one large bin                   */
/* n <==> number of data                                          */
/* kernel <==> type of kernel coded with 1 to 5                   */
double*x,*fh,*h,*wM,*origin;
long*M,*n,*kernel,*binmesh,*counts,*J;
{
  register long i,k,index;
  double delta,cm;
  long binnumber;
  delta=h[0]/(double)M[0];
  binnumber=0;
/* delta binwidth of small bins <==> h/M                          */
/* binnumber, integer to count the non-empty bins                 */
/* Binning the data                                               */
  for (i=0;i<n[0];i++)
  {
    index=floor((x[i]-origin[0])/delta);
/* the actual obs.  belongs to the bin with number index          */
    if (binmesh[index]==0)
/* if the actual observation is the first in that bin             */
    {
      binnumber++;
/* the number of non-empty bins increases by 1                    */
      counts[binnumber]=1;
/* the number of obs.  in this bin is 1 (at this time)            */
      J[binnumber]=index;
      binmesh[index]=binnumber;
/* protocol of the position of the new                            */
/* non-empty bin (in counts[] and J[])                            */
    }
    else
/* if the actual observation is not the first in this bin         */
    {
      counts[binmesh[index]]++;
/* increase the number of obs.  in belonging non-empty bin        */
    }
  }
/* The vector counts contains nonzero values in positions         */
/* 1,2,3,..,binnumber.  The other values are prespecified         */
/* as 0, since counts is installed as a 0-vector.  In             */
/* analogy J contains the indices of the non-empty bins.          */
/* Calculation of the kernel depending weights.                   */
/* The factor c(M) assures that sum(w.M(i/M),1-M,M-1)=M.          */
/* Note, in formula of cm the division by [n * h] does not        */
/* belong to the theoretical value of cm, but is a factor,        */
/* which has to be multiplied one time, since fh=1/(nh)*..        */
  switch ( *kernel )
  {
    case 1 :
/* uniform-kernel K(u)=c(M) * I(|u|<=1)                           */
/* c(M)=M/(2M-1)                                                  */
    {
      cm=(double)M[0]/((double)(2*M[0]-1)*(double)n[0]*h[0]);
      for (i=0;i<M[0];i++)
        wM[i]=cm;
      break;
    }
    case 2 :
/* Triangle - kernel K(u)=c(M)*(1- |u|) * I(|u|<=1)               */
/* original Average-Shifted-Histogram-kernel                      */
/* c(M)=1                                                         */
    {
      cm=1.0/((double)n[0]*h[0]);
      for (i=0;i<M[0];i++)
        wM[i]=cm*(1-(double)i/(double)M[0]);
      break;
    }
    case 3 :
/* Epanechnikov - kernel K(u)= c(M) * (1- u*u) * I(|u|<=1)        */
/* c(M)=3*M^2/(4*M^2-1)                                           */
    {
      cm=(double)(3*M[0]*M[0])/((double)(4*M[0]*M[0]-1)
                                        *(double)n[0]*h[0]);
      for (i=0;i<M[0];i++)
        wM[i]=cm*(1-pow((double)i/(double)M[0],2.0));
```



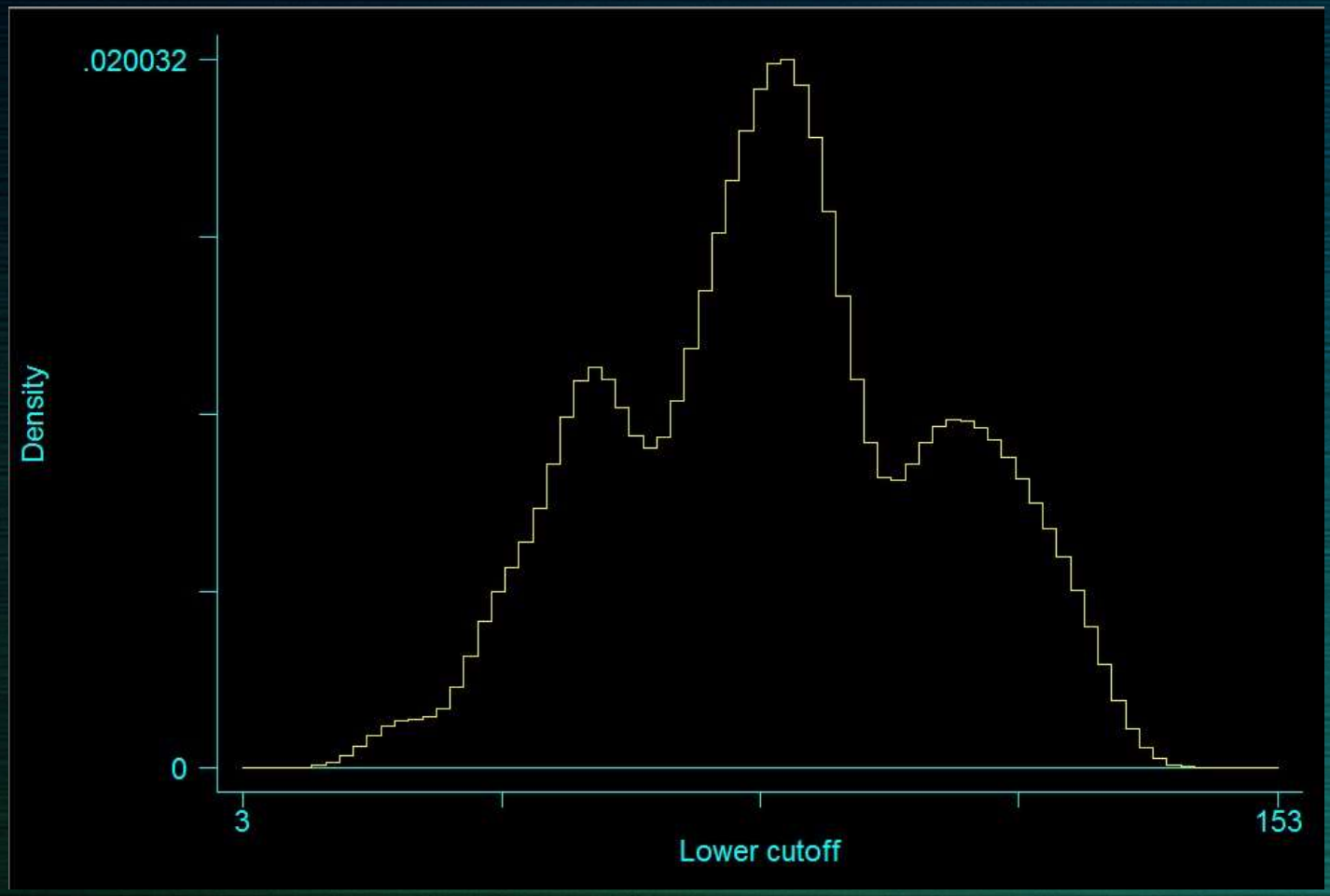
Isaías Hazarmabeth Salgado Ugarte

Stata ado programs and Turbo Pascal executables



Turbo Pascal

First combinations Stata-Turbo Pascal executables: ASH warpdenm.ado (TP warpingc.exe; warpingw.exe)



```
program warpingc(input,output);
{First written on November, 11, 1994; last correction November 18, 1994.
Authors: Isaias Hazarmabeth Salgado-Ugarte, Makoto Shimizu and
Toru Taniuchi. Reference: Stata Technical Bulletin XX, snp6.1

This version does not display results to screen, instead write them to
a text file in order to be used by Stata's command "infile" as a part
of three Stata's ado files (warpstep.ado, warpoly.ado and warping.ado).
This Pascal program performs Haerdle's WARPing density estimation
using modified algorithms from Haerdle (1991) "Smoothing Techniques
with Implementations in S. Springer-Verlag Series in Statistics
New York, and Scott (1992) "Multivariate Density Estimation: Theory,
Practice, and Visualization. John Wiley and Sons, New York.}

const
  maxsize = 2000;
  maxsize2 = 2000; {it is a really big limit!}

type
  vector = array[0..maxsize] of real;
  striname = string[30];
  vectorint = array[0..maxsize2] of integer;

var
  binumber,n,i,M,numbin,kerneltype,index   : integer;
  start, delta,h,origin,maxval,minval,cm   : real;
  binmesh,counts,J                         : vectorint;
  x,wM,fh,midval                           : vector;
  inpval,data,resul                        : text;


procedure InputValues;
begin
   assign(data,'_data2.raw');
   reset(data);
   i:=1;
   while (not eof(data)) do
     begin
        read(data,x[i]);
        i:=i+1;
     end;
   close(data);
   n:=i-2;
   assign(inpval,'_inpval.raw');
   reset(inpval);
   read(inpval,h,M,kerneltype);
   close(inpval);
end;

procedure InitialCalc;
var maxnoemp,ib: integer;

begin
   maxval:=x[n];
   minval:=x[1];
   if (kerneltype=6) then
      h:=4*h;
   delta:= h/M;
   numbin:= trunc((maxval-minval)/delta)+2*(M+1+round((M/10)+0.5));
   start:=(minval-h)-delta*0.1;
   if (start<0) then
      origin:=(round((start/delta)-0.5)-0.5)*delta
   else
      origin:=(trunc(start/delta)-0.5)*delta;
   if (n+1<numbin) then
      maxnoemp:=n+1

   for ib:= 0 to numbin+1 do
      begin
        binmesh[ib]:=0;
        counts[ib]:=0;
        J[ib]:=0;
      end;
end;

procedure BinmeshCalc;
var
   ia : integer;

begin
   for i:=1 to n do
    begin
      index:= trunc((x[i]-origin)/delta);
      if (binmesh[index]=0) then
         begin
            binumber:= binumber+1;
            J[binumber]:=index;
            counts[binumber]:=1;
            binmesh[index]:=binumber;
         end
      else
        counts[binmesh[index]]:=counts[binmesh[index]]+1;
    end;

    for ia:=0 to numbin-1 do
        midval[ia]:=((0.5+ia)*delta)+origin;
end;

procedure CreateWeight;
var
l: integer;
part1,part2 : real;

function CleverPower(x:real; n:integer): real;
  begin
  if n=1 then
    CleverPower:=x
  else if odd(n) then
    CleverPower:=x*sqr(CleverPower(x, n div 2))
  else
    CleverPower:=sqr(CleverPower(x, n div 2))
  end;

begin
  case kerneltype of
    1: begin      {uniform kernel}
         cm:=M/((2*M-1)*(n*h));
         for l:=0 to M-1 do
           wM[l]:= cm;
       end;
    2: begin {Triangle kernel (Original Average-Shifted-Histogram kernel)}
         cm:=1/(n*h);
         for l:=0 to M-1 do
           wM[l]:= cm*(1 -l/M);
       end;
    3: begin     {Epanechnikov kernel}
         cm:=3*CleverPower(M,2)/((4*CleverPower(M,2)-1)*(n*h));
         for l:=0 to M-1 do
           wM[l]:= cm*(1-CleverPower((l/M),2));
       end;
    4: begin    {Quartic Kernel}
         cm:=0.9375/((1-0.0625/(CleverPower(M,2)))*(n*h));
```

Turbo Pascal warping

Routines modified from C programs of Härdle, 1990

```
. bandw snow
```

Some practical number of bins and binwidth-bandwidth rules for univariate density estimation using histograms, frequency polygons (FP) and kernel density estimators

Sturges' number of bins =	6.9773
Oversmoothed number of bins <=	5.0133
FP oversmoothed number of bins <=	5.4091
Scott's optimal Gaussian binwidth =	20.8641
Freedman-Diaconis optimal robust binwidth =	17.4413
Terrell-Scott's oversmoothed binwidth >=	20.2262
Oversmoothed homoscedastic binwidth >=	22.2292
Oversmoothed robust binwidth >=	22.6999
FP optimal Gaussian binwidth =	22.2680
FP oversmoothed binwidth >=	24.1323

Gaussian kernel (6)

Silverman's optimal bandwidth =	9.3215
Haerdle's 'better' optimal bandwidth =	10.9787
Scott's oversmoothed bandwidth =	11.8487

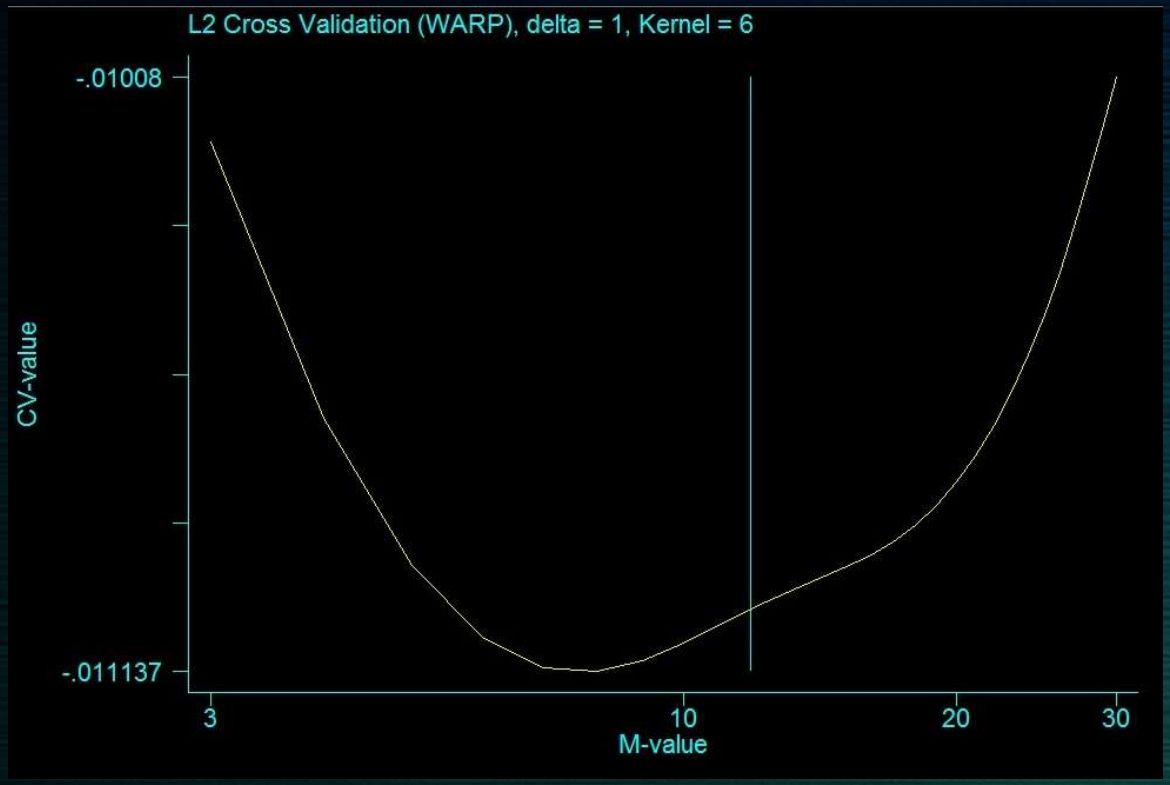
```
program l2cvwarf;
{First written: September 25, 1994. Last revised March 7, 1995.
Author: Isaias Hazarmabeth Salgado Ugarte.
Univ. of Tokyo, Fac. of Agriculture, Dept. of
Fisheries. Yayoi 1-1-1, Bunkyo-ku, Tokyo 113, Japan.
No results to screen, only to text file. Read data from _data2 and
input values from _inpval (ASCII). This Pascal program performs
least squares cross validation for warping density estimation, using
modified algorithms from Haerdle (1991) "Smoothing Techniques with
Implementations in S. Springer-Verlag Series in Statistics, New York }

const
  maxsize = 1000;
  maxsize2 = 1000;

type
  vector = array[0..maxsize] of real;
  striname = string[30];
  vectorint = array[0..maxsize2] of integer;

var
  n,i,M,mstart,mend,mnumber,numbin,kerneltype,indexi,nl  :integer;
  start,delta,h,origin,maxval,minval,cm,intsquare,estexp : real;
  bin,frequ,index,mv                                     : vectorint;
  x,kwe,fm,cv,hv                                         : vector;
  inpval,data,resul                                      : text;


procedure InputValues;
var i: integer;
begin
   assign(data,'_data2.raw');
   reset(data);
   i:=1;
   while (not eof(data)) do
    begin
       read(data,x[i]);
       i:=i+1;
    end;
   close(data);
   n:=i-2;
   assign(inpval,'_inpval.raw');
   reset(inpval);
   read(inpval,delta,kerneltype,mstart,mend);
   close(inpval);
end;

procedure InitialCalc;
var maxnoemp,ib: integer;

begin
   mnumber:=mend-mstart+1;
   maxval:=x[n];
   minval:=x[1];
   if kerneltype=6 then
      delta:=delta*4;
   start:=minval-delta*(Mend+0.1);
    if (start<0) then
      origin:=(round((start/delta)-0.5)-0.5)*delta
   else
      origin:=(trunc(start/delta)-0.5)*delta;
   numbin:=round(((maxval+delta*(Mend+0.1)-origin)/delta)+0.5)+1;
   if (n+1<numbin) then
      maxnoemp:=n+1
   else
      maxnoemp:=numbin;
   nl:=0;
   indexi:=0;
   for ib:= 1 to numbin+1 do
      begin
        bin[ib]:=0;
```

Turbo Pascal

l2cvwarf

Routines in Turbo Pascal modified

from C programs of Härdle, 1990



l2cvwarp.ado
l2cvwarw.ado

TP
- l2cvwars.exe (MSDOS)
- l2cvwarw.exe (MSW)

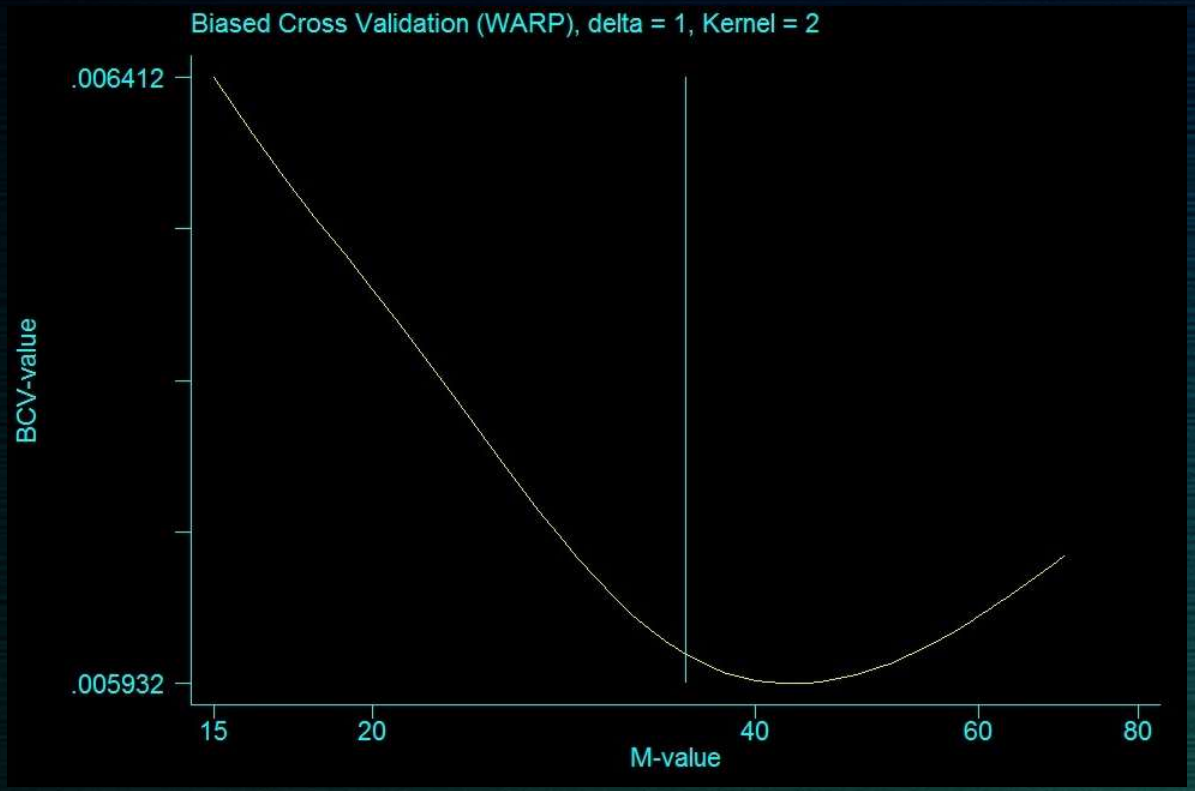

```
. l2cvwarp snow, d(1) k(6) ms(3) me(30) xlog xlab xline(11.85)

------------------------------------------------------------------------------
Least Squares Cross-validation for WARPing density estimation, Gaussian kernel
------------------------------------------------------------------------------
CV-value = -0.01113733       M-value =     8     Bandwidth =    8.0000
CV-value = -0.01113101       M-value =     7     Bandwidth =    7.0000
CV-value = -0.01111789       M-value =     9     Bandwidth =    9.0000
CV-value = -0.01108621       M-value =    10     Bandwidth =   10.0000
CV-value = -0.01107764       M-value =     6     Bandwidth =    6.0000
```

```
program bcvwarpf;
{First written: September 27, 1994. Last revised: March 7, 1995.
Authors: Isaias Hazarmabeth Salgado Ugarte.
Univ. of Tokyo, Fac. of Agriculture, Dept. of
Fisheries. Yayoi 1-1-1, Bunkyo-ku, Tokyo 113, Japan.
No results to screen, only to text file. This Pascal program performs
least squares cross validation for WARPing density estimation, using
modified algorithms from Haerdle (1991) "Smoothing Techniques with
Implementations in S. Springer-Verlag Series in Statistics, New York }

const
  maxsize = 1000;
  maxsize2 = 1000;

type
  vector = array[0..maxsize] of real;
  striname = string[30];
  vectorint = array[0..maxsize2] of integer;

var
  n,i,M,mstart,mend,mnumber,numbin,kerneltype,indexi,nl        :integer;
  start,delta,h,origin,maxval,minval,cm,score,tau,kappa,zeta   : real;
  bin,frequ,index,mv                                           : vectorint;
  x,kwe,fm,bcv,hv,ma                                           : vector;
  inpval,data,resul                                            : text;

function Power(x:real; n:integer): real;
  begin
  if n=1 then
    Power:=x
  else if odd(n) then
    Power:=x*sqr(Power(x, n div 2))
  else
    Power:=sqr(Power(x, n div 2))
  end;

procedure InputValues;
var i: integer;
begin
   assign(data,'_data2.raw');
   reset(data);
   i:=1;
   while (not eof(data)) do
    begin
       read(data,x[i]);
       i:=i+1;
    end;
   close(data);
   n:=i-2;
   assign(inpval,'_inpval.raw');
   reset(inpval);
   read(inpval,delta,kerneltype,mstart,mend);
   close(inpval);
end;

procedure InitialCalc;
var maxnoemp,ib: integer;

begin
   mnumber:=mend-mstart+1;
   maxval:=x[n];
   minval:=x[1];
   start:=minval-delta*(Mend+0.1);
    if (start<0) then
      origin:=(round((start/delta)-0.5)-0.5)*delta
   else
      maxnoemp:=numbin;
   nl:=0;
   indexi:=0;
   for ib:= 0 to numbin+1 do
      begin
        bin[ib]:=0;
        frequ[ib]:=0;
        index[ib]:=0;
      end;
end;

procedure BinningData;
var
   ia,bindx,bi : integer;
   sum1, sum2,interval : real;
begin
   for i:=1 to n do
    begin
      indexi:= trunc((x[i]-origin)/delta);
      if (bin[indexi]=0) then
         begin
            nl:= nl+1;
            frequ[nl]:=1;
            index[nl]:=indexi;
            bin[indexi]:=nl;
         end
      else
        frequ[bin[indexi]]:=frequ[bin[indexi]]+1;
    end;
end;

procedure CalcFunctional;
var
   divisor : real;

begin
   case kerneltype of
      1: begin {Quartic}
         tau:=(16*ma[4]+32*ma[2]-13)/(112*ma[2]+28);
         divisor:=(16*ma[4]-1)*(4*ma[2]+1);
         kappa:=15*ma[1]/7*(1-(32*ma[4]-28*ma[2]+1)/divisor);
         zeta:=30/7*(336*ma[3]-420*ma[2]+85*ma[1]+104)/divisor;
         end;
      2: begin {Triweight}
         tau:=(32*ma[6]+80*ma[4]+62*ma[2]-69)/(288*ma[4]+72*ma[2]+60);
         divisor:=(4*ma[2]-1)*Power((24*ma[4]+6*ma[2]+5),2);
         kappa:=350*ma[1]/143*(1-(4492.8*ma[8]-979.2*ma[6]+2124*ma[4]-866.4*ma[2]-
2755.2)/(3*divisor));
         zeta:=630*(128*ma[7]-448*ma[5]+560*ma[4]-280*ma[3]-280*ma[2]+320*ma[1]+35)/divisor;
         end;
      end; {case}
end;

procedure CreateWeight2;
var
l: integer;
part1,part2 : real;

begin
  case kerneltype of
    1: begin
         cm:=0.9375/((1-0.0625/(Power(M,4)))*(n*h));
         for l:=0 to M-1 do
           kwe[l]:=cm*Power((1-Power((l/M),2)),2);
       end;
    2: begin
```

Turbo Pascal

bcvwarpf

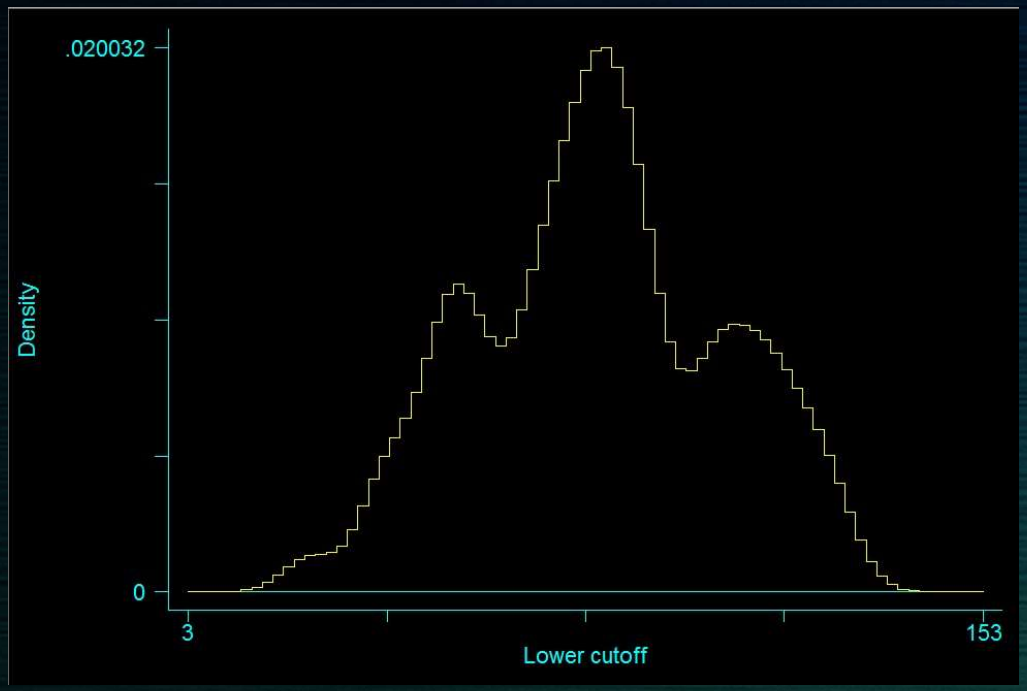
Routines in Turbo Pascal modified from C programs of Härdle, 1990

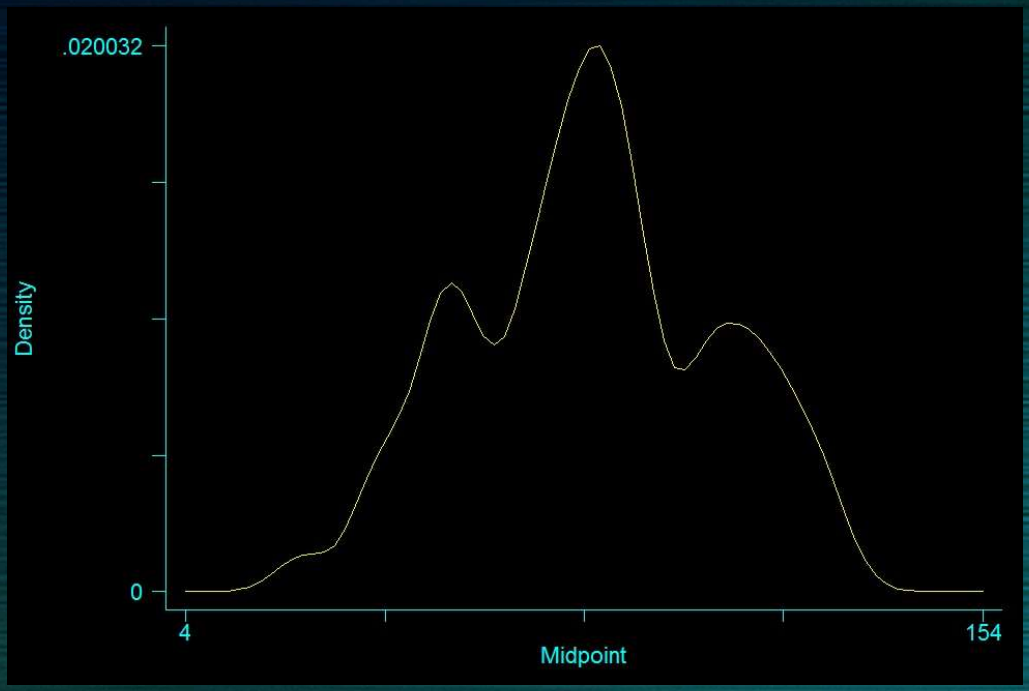


bcvwarp.ado
bcvwarw.ado

TP
- bcvwarpf.exe (MSDOS)
- bcvwarpw.exe (MSW)

```
. bcvwarp snow, d(1) k(2) ms(15) me(70) xlog xlab xline(35.29)

-------------------------------------------------------------------------------
Biased Cross-validation for WARPing density estimation, Triweight kernel
-------------------------------------------------------------------------------
Biased Cv-value = 0.00593182   M-value =    43   Bandwidth =  43.0000
Biased Cv-value = 0.00593193   M-value =    42   Bandwidth =  42.0000
Biased Cv-value = 0.00593229   M-value =    44   Bandwidth =  44.0000
Biased Cv-value = 0.00593269   M-value =    41   Bandwidth =  41.0000
Biased Cv-value = 0.00593329   M-value =    45   Bandwidth =  45.0000
```



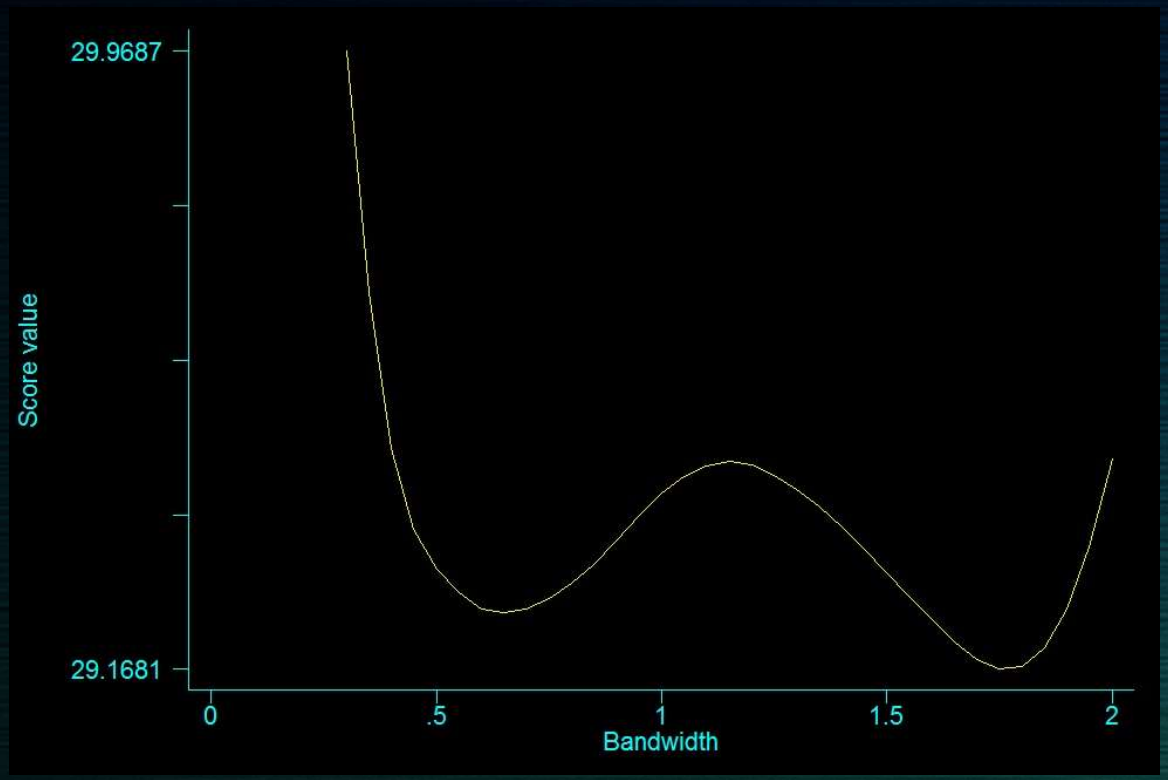


warpden.ado y warpdenw.ado
Turbo Pascal executables
- warpingc.exe (MS DOS)
- warpingw.exe (MS Windows)

Turbo Pascal gwarpren

Routines modified from C programs of Härdle, 1990

```
program gwarpren;
{First written: September 30, 1994. Last revised, November 11,1994.
Authors: Isaias Hazarmabeth Salgado Ugarte.
No results to screen, only to text file. This Pascal program calculates
the adjusted prediction error G(M) for warping regression Nadaraya-Watson
estimate, using modified algorithms and C programs modified from Haerdle
(1991) "Smoothing Techniques with Implementations in S. Springer-Verlag
Series in Statistics, New York }

const
  maxsize = 1000;
  maxsize2 = 1000;

type
  vector = array[0..maxsize] of real;
  vectorint = array[0..maxsize2] of integer;

var

n,selector,M,mstart,mend,mnumber,numbin,kerneltype,j,nl,binboundh,binboundl,nlweight,index2
:integer;
  start,delta,h,origin,maxval,minval,boundary,rm,fm,mm      : real;
  bin,counts,index,mv,indexweight                           : vectorint;
  x,y,ysum,ysquaresum,kwe,score,hv                          : vector;
  inpval,data,resul                                         : text;

function Power(x:real; n:integer): real;
  begin
  if n=1 then
    Power:=x
  else if odd(n) then
    Power:=x*sqr(Power(x, n div 2))
  else
    Power:=sqr(Power(x, n div 2))
  end;

function SelecFun (u,v :real): real;
begin
   case selector of
      1:   SelecFun:=1+2*u/v;
      2:   SelecFun:=1/Power((1-u/v),2);
      3:   SelecFun:=exp(2*u/v);
      4:   Selecfun:=(1.0+u/v)/(1.0-u/v);
      5:   Selecfun:=1.0/(1.0-2.0*u/v);
   end; {case}
end;

procedure InputValues;
var i: integer;
begin
   assign(data,'_data2.raw');
   reset(data);
   i:=1;
   while (not eof(data)) do
    begin
       read(data,y[i],x[i]);
       i:=i+1;
    end;
   close(data);
   n:=i-2;
   assign(inpval,'_inpval.raw');
   reset(inpval);
   read(inpval,delta,selector,kerneltype,mstart,mend,boundary);
   close(inpval);
end;

begin
   mnumber:=mend-mstart+1;
   maxval:=x[n];
   minval:=x[1];
   if kerneltype=6 then
      delta:=delta*4;
   start:=minval-delta*(Mend+0.1);
    if (start<0) then
      origin:=(round((start/delta)-0.5)-0.5)*delta
   else
      origin:=(trunc(start/delta)-0.5)*delta;
   numbin:=round(((maxval+delta*(Mend+0.1)-origin)/delta)+0.5)+1;
   binboundl:=trunc(((x[trunc(n*boundary/2)])-origin)/delta);
   binboundh:=trunc(((x[trunc(n-n*boundary/2)])-origin)/delta);
   if (n+1<numbin) then
      maxnoemp:=n+1
   else
      maxnoemp:=numbin;
   nl:=0;
   nlweight:=0;
   for ii:= 0 to numbin+1 do
      begin
        bin[ii]:=0;
        counts[ii]:=0;
        ysum[ii]:=0;
        ysquaresum[ii]:=0;
        kwe[ii]:=0;
        index[ii]:=0;
        score[ii]:=0;
      end;
end;


procedure BinningData;
var
ib : integer;
begin
   for ib:=1 to n do
    begin
      j:= trunc((x[ib]-origin)/delta);
      if (bin[j]=0) then
         begin
            nl:= nl+1;
            counts[nl]:=1;
            ysum[nl]:=y[ib];
            ysquaresum[nl]:=Power(y[ib],2);
            index[nl]:=j;
            bin[j]:=nl;
            if (j>=binboundl) and (j<=binboundh) then
               begin
                  nlweight:=nlweight+1;
                  indexweight[nlweight]:=nl;
               end
         end
      else
         begin
            counts[bin[j]]:=counts[bin[j]]+1;
            ysum[bin[j]]:=ysum[bin[j]]+y[ib];
            ysquaresum[bin[j]]:=ysquaresum[bin[j]]+Power(y[ib],2);
         end
    end;
end;


procedure CreateWeight;
var
```



gwarpreg.ado
gwarprew.ado

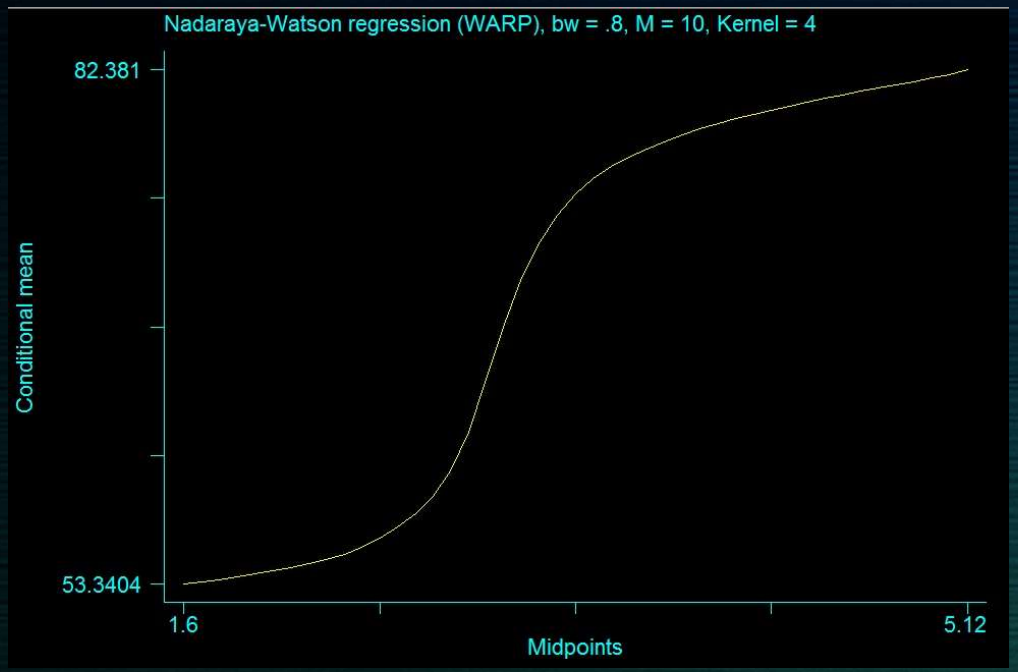
TP
- gwarpren.exe (MSDOS)
- gwarprew.exe (MSW)

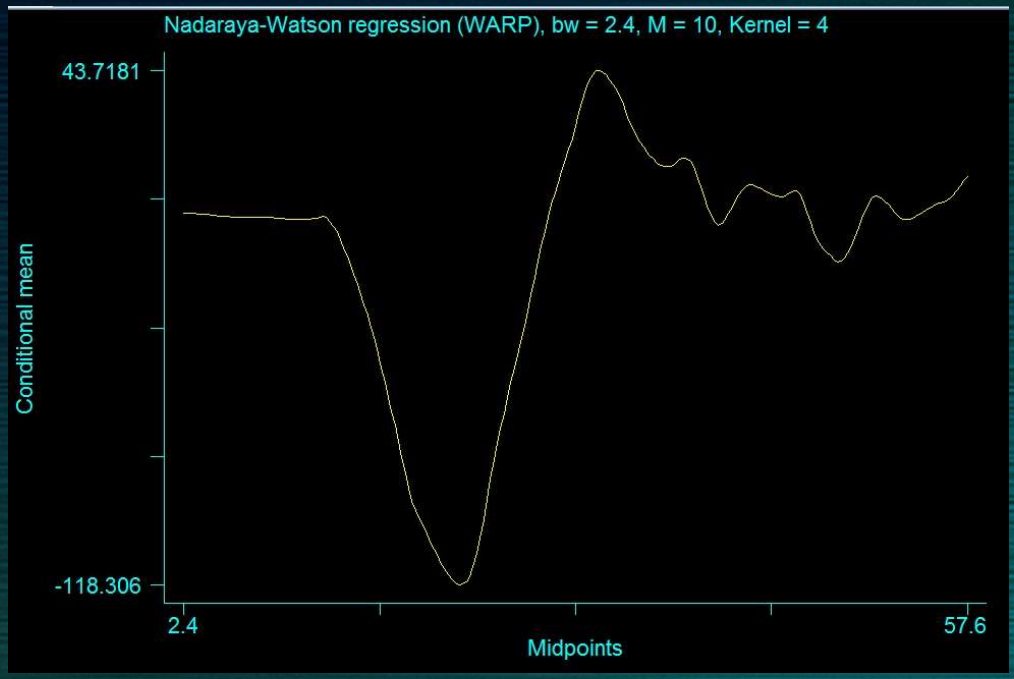
```
. gwarpreg wait dura, d(.05) s(2) k(4) ms(6) me(40)

------------------------------------------------------------------------------
Adjusted Prediction error G(M) for WARPing Nadaraya-Watson regression
------------------------------------------------------------------------------
M-value = 35       Score value = 29.168085       Bandwidth =    1.7500
M-value = 36       Score value = 29.170967       Bandwidth =    1.8000
M-value = 34       Score value = 29.18079        Bandwidth =    1.7000
M-value = 37       Score value = 29.195848       Bandwidth =    1.8500
M-value = 33       Score value = 29.203459       Bandwidth =    1.6500
```

```
program warpregf;
{First written on November 11, 1994; last revised November 11, 1994.
Author: Isaias Hazarmabeth Salgado-Ugarte.
This version do not display results to screen, instead write them to
a text file in order to be used by Stata's command "infile" as a part
of a Stata's ado file (warpreg.ado).
This Pascal program performs Nayarada-Watson Nonparametric regression
employing Haerdle's WARPing approximation using modified algorithms
taken from Haerdle (1991) "Smoothing Techniques with Implementations
in S. Springer-Verlag Series in Statistics New York}

const
  maxsize = 1000;
  maxsize2 = 1000; {it is a really big limit!}

type
  vector = array[0..maxsize] of real;
  vectorint = array[0..maxsize2] of integer;

var
  indexstart,indexend,nl,n,i,M,numbin,kerneltype,j   : integer;
  start, delta,h,origin,maxval,minval                : real;
  bin,frequ,index                                    : vectorint;
  x,y,ysum,kwe,fm,rm,mm,midval                       : vector;
  inpval,data,resul                                  : text;


procedure InputValues;
begin
   assign(data,'_data2.raw');
   reset(data);
   i:=1;
   while (not eof(data)) do
    begin
       read(data,y[i],x[i]);
       i:=i+1;
    end;
   close(data);
   n:=i-2;
   assign(inpval,'_inpval.raw');
   reset(inpval);
   read(inpval,h,M,kerneltype);
   close(inpval);
end;


procedure InitialCalc;
var maxnoemp,ia,ib: integer;

begin
   maxval:=x[n];
   minval:=x[1];
   if kerneltype=6 then
      h:=4*h;
   delta:= h/M;
   start:=minval-h-delta*0.1;
   if (start<0) then
      origin:=(round((start/delta)-0.5)-0.5)*delta
   else
      origin:=(trunc(start/delta)-0.5)*delta;
   numbin:=trunc(((maxval+h+delta*0.1-origin)/delta)+0.5)+1;
   if (n+1<numbin) then
      maxnoemp:=n+1
   else
      maxnoemp:=numbin;
   j:=0;
   for ib:= 0 to numbin+1 do
      begin
        bin[ib]:=0;
        frequ[ib]:=0;
        index[ib]:=0;
      end;
end;

procedure BinningData;
begin
   for i:=1 to n do
    begin
      j:= trunc((x[i]-origin)/delta);
      if (bin[j]=0) then
         begin
            nl:= nl+1;
            index[nl]:=j;
            frequ[nl]:=1;
            ysum[nl]:=y[i];
            bin[j]:=nl;
         end
      else
         begin
           frequ[bin[j]]:=frequ[bin[j]]+1;
           ysum[bin[j]]:=ysum[bin[j]]+y[i];
         end
    end;
end;

procedure CreateWeight; {Note: weights are not normalized}
var
l: integer;

function Power(x:real; n:integer): real;
  begin
  if n=1 then
    Power:=x
  else if odd(n) then
    Power:=x*sqr(Power(x, n div 2))
  else
    Power:=sqr(Power(x, n div 2))
  end;

begin
  case kerneltype of
    1: begin      {uniform kernel}
         for l:=0 to M-1 do
           kwe[l]:= 1;
       end;
    2: begin {Triangle kernel (Original Average-Shifted-Histogram kernel)}
         for l:=0 to M-1 do
           kwe[l]:= 1 -l/M;
       end;
    3: begin     {Epanechnikov kernel}
         for l:=0 to M-1 do
           kwe[l]:= 1-Power((l/M),2);
       end;
    4: begin   {Quartic Kernel}
         for l:=0 to M-1 do
           kwe[l]:=Power((1-Power((l/M),2)),2);
       end;
    5: begin    {Triweight kernel}
         for l:=0 to M-1 do
           kwe[l]:=Power((1-Power((l/M),2)),3);
```

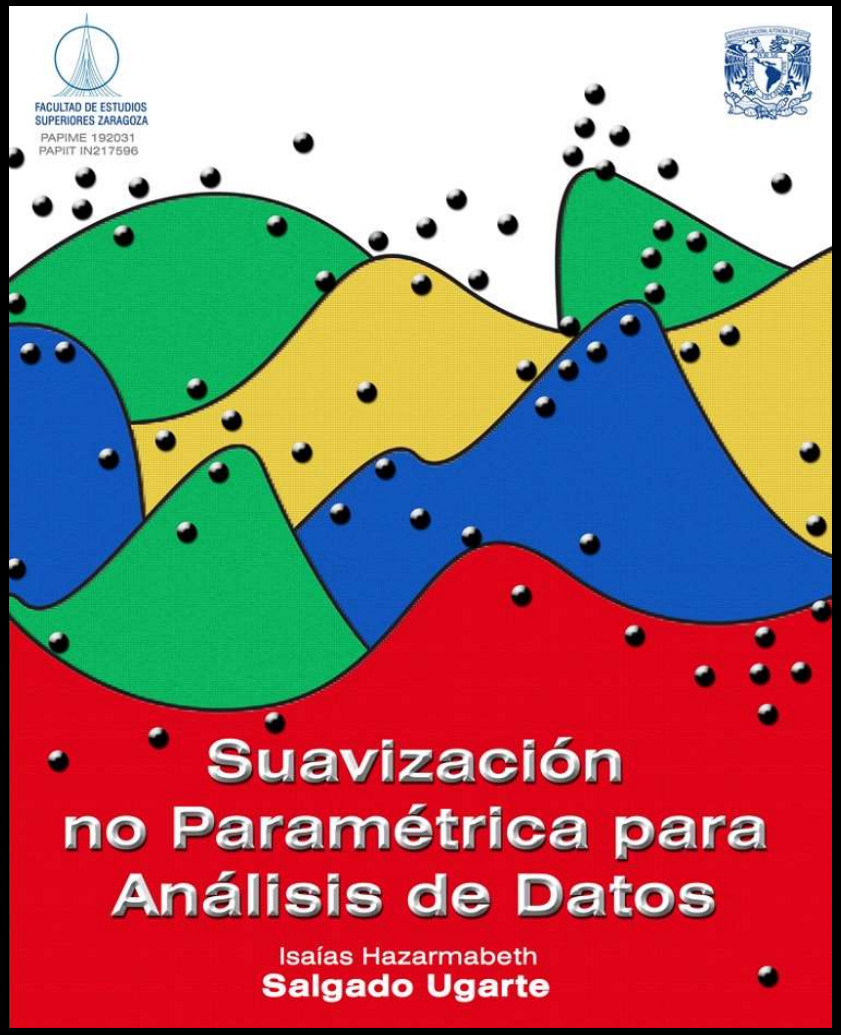
Turbo Pascal warpregf

Routines modified from C programs of Härdle, 1990



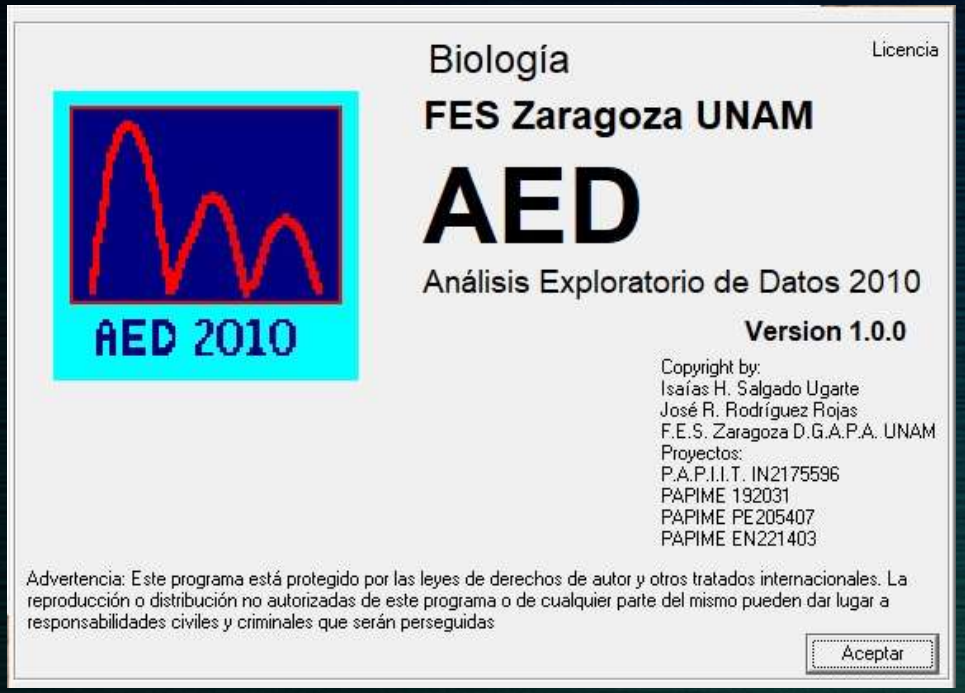


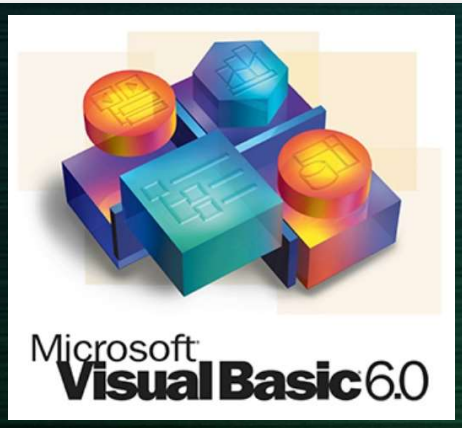
warpreg.ado y warpregw.ado
Turbo Pascal executables
- warpregf.exe (MS DOS)
- warpregw.exe (MS Windows)



2002

Routines in Stata and Turbo Pascal; EDK2000 MSVB





MS Visual Basic routines

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd


# # Import Data

# In[2]:


data = np.loadtxt('_data2.raw')


# In[3]:


n = len(data)


# # Model Parameters

# In[4]:

params = np.loadtxt('_inpval.raw')

h = float(params[0])
M = int(params[1])
kernel_type = int(params[2])


# # Initial Calc

# In[5]:


maxval = data[-1];
minval = data[0];

if kernel_type == 6:
    h = 4*h

delta = h/M
numbin = int((maxval-minval)/delta)+2*(M+1+round((M/10)+0.5))
start = (minval-h)-delta*0.1

if start<0:
    origin = (round((start/delta)-0.5)-0.5)*delta

else:
    origin = (int(start/delta)-0.5)*delta

if n+1<numbin:
    maxnoemp = n+1

else:
    maxnoemp = numbin

binumber = 0
index = 0

binmesh = np.zeros(numbin+1)
counts = np.zeros(numbin+1)
J = np.zeros(numbin+1)


# # Binmesh Calc

# In[6]:


for i in range(0,n):
    index = int((data[i]-origin)/delta)
    if binmesh[index]==0:
        binumber = binumber+1
        J[binumber] = index
        counts[binumber] = 1
        binmesh[index] = binumber
    else:
        counts[int(binmesh[index])] = counts[int(binmesh[index])]+1

midval = []
for ia in range(0,numbin):
    value = ((0.5+ia)*delta) + origin
    midval.append(value)


# # Create Weight

# In[7]:


def CreateWeight(kerneltype,M):

    if kerneltype == 1:
        cm = M/((2*M-1)*(n*h))
        wM = [cm for l in range(0,M)]

    if kerneltype == 2:
        cm = 1/(n*h)
        wM = [cm*(1-l/M) for l in range(0,M)]

    if kerneltype == 3:
        cm = 3*(M**2)/((4*(M**2)-1)*(n*h))
```

Python scripts

(warping.py)

```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   16.1   Copyright 1985-2019 StataCorp LLC
  Statistics/Data analysis             StataCorp
                                       4905 Lakeway Drive
     Special Edition                   College Station, Texas 77845 USA
                                       800-STATA-PC        https://www.stata.com
                                       979-696-4600        stata@stata.com
                                       979-696-4601 (fax)

Stata license: 10-student lab perpetual
Serial number: 401606284495
  Licensed to: Laboratorio de Biometria y Biologia Pesquera
               FES Zaragoza UNAM

Notes:
      1. Unicode is supported; see help unicode_advice.
      2. Maximum number of variables is set to 5,000; see help set_maxvar.
```

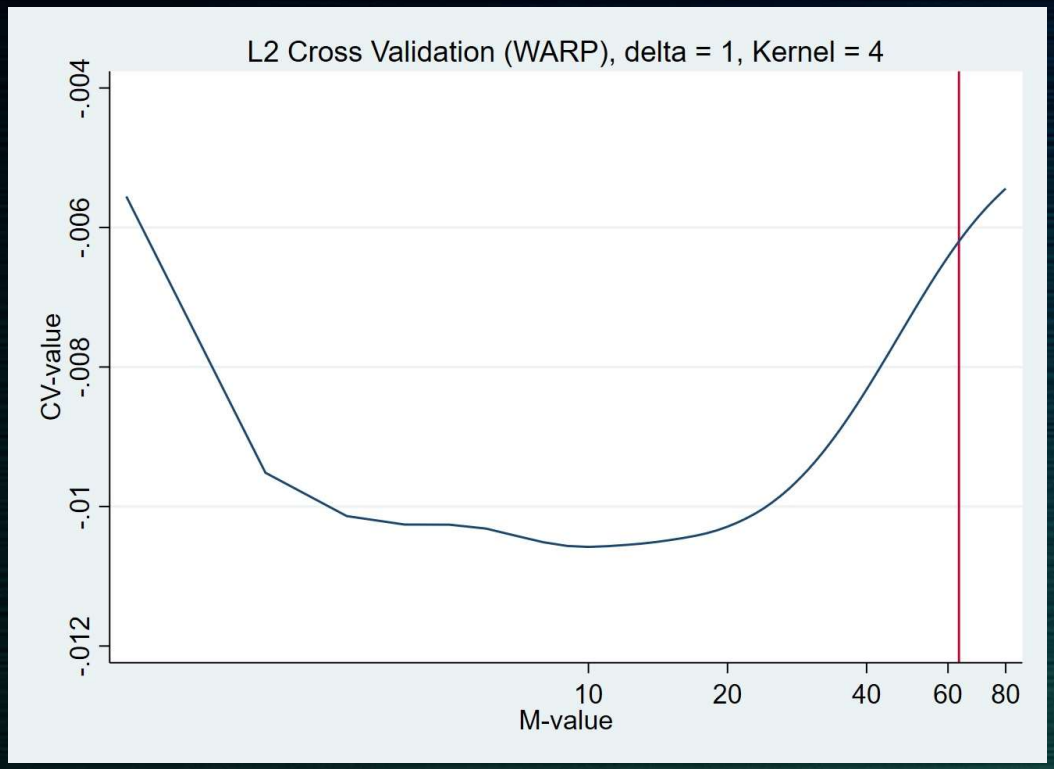
Stata 16.1

Python

```
  ___  ____  ____  ____  ____ ®
 /__    /   ____/   /   ____/      17.0
___/   /   /___/   /   /___/       MP—Parallel Edition

 Statistics and Data Science       Copyright 1985-2021 StataCorp LLC
                                   StataCorp
                                   4905 Lakeway Drive
                                   College Station, Texas 77845 USA
                                   800-STATA-PC        https://www.stata.com
                                   979-696-4600        stata@stata.com

Stata license: 13-user 2-core network, expiring  8 Nov 2021
Serial number: 501709348149
  Licensed to: Isaías Hazarmabeth Salgado-Ugarte
               FES Zaragoza UNAM

Notes:
      1. Unicode is supported; see help unicode_advice.
      2. More than 2 billion observations are allowed; see help obs_advice.
      3. Maximum number of variables is set to 5,000; see help set_maxvar.
```

Stata 17.0

Python

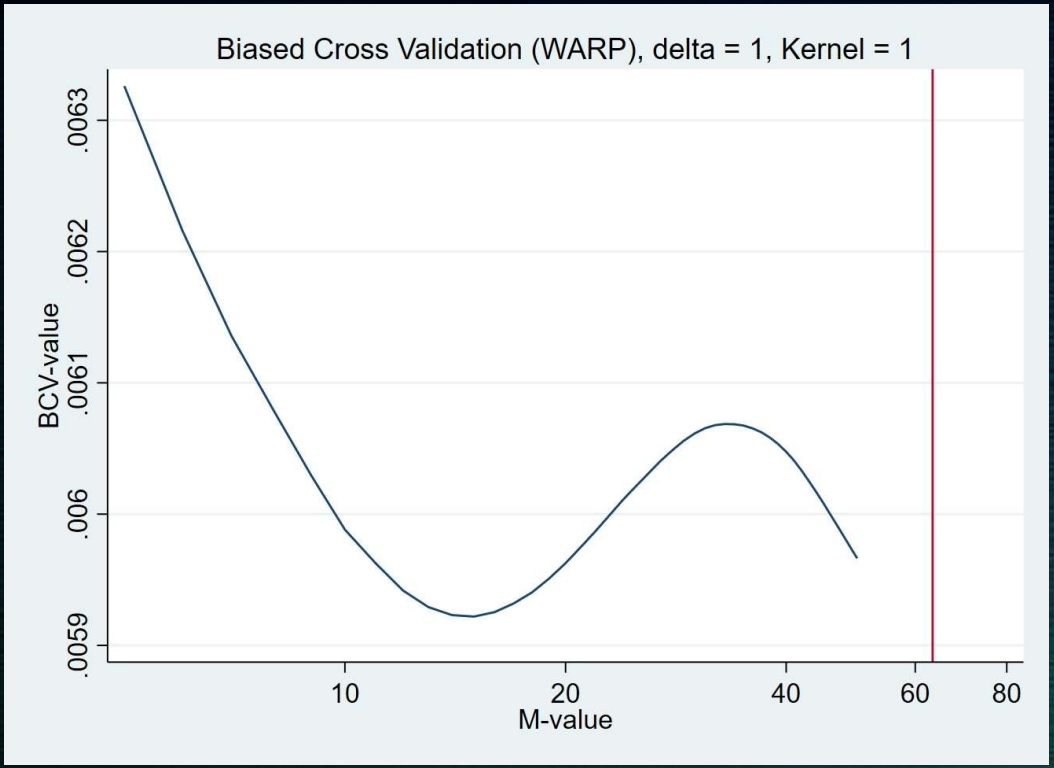


l2cvwarpy.ado
Quartic kernel

Python

- l2cvwarpy.py

```
. l2cvwarpy bodlen, d(1) k(4) mstart(1) mend(80) xscale(log) xline(63.37) xlab(10 20 40 60 80)

-------------------------------------------------------------------------------
Least Squares Cross-validation for WARPing density estimation, Quartic kernel
-------------------------------------------------------------------------------
CV-value = -0.01057915       M-value =    10     Bandwidth =  10.0000
CV-value = -0.01056818       M-value =    11     Bandwidth =  11.0000
CV-value = -0.01056505       M-value =     9     Bandwidth =   9.0000
CV-value = -0.01055124       M-value =    12     Bandwidth =  12.0000
CV-value = -0.01053272       M-value =    13     Bandwidth =  13.0000
```



bcvwarpy.ado

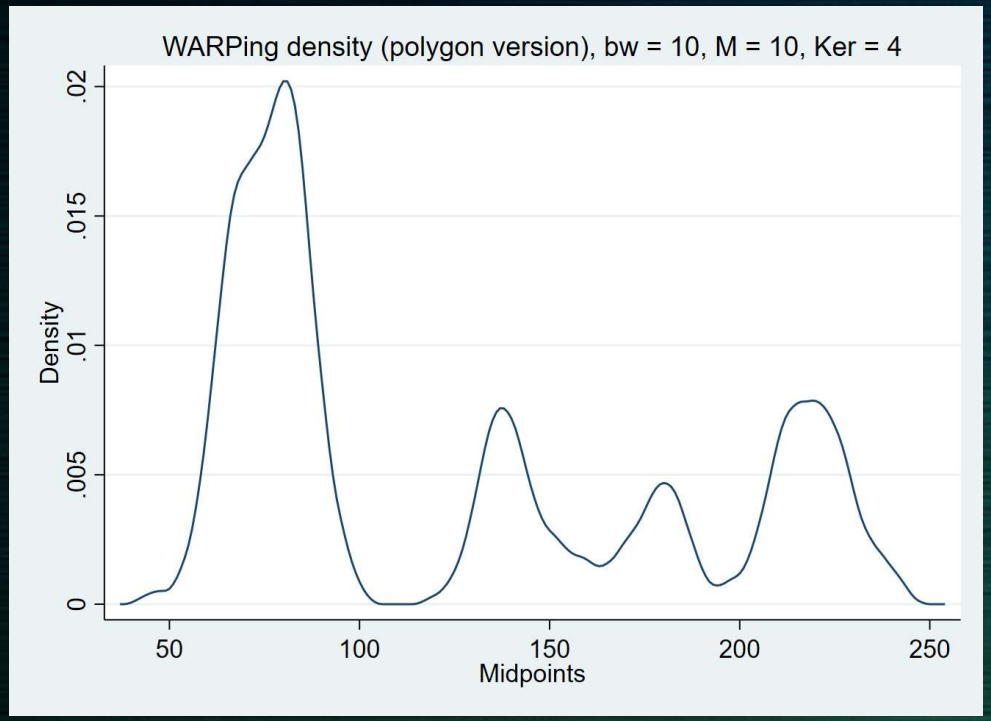
Quartic kernel (15)

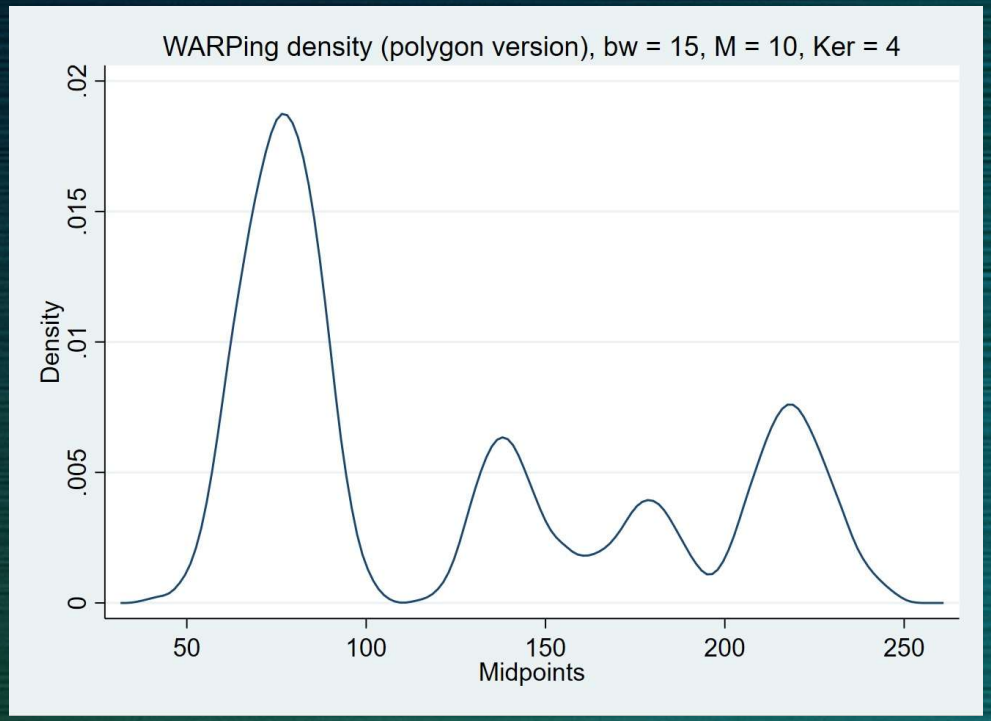
Python

- bcvwarpy.py

```
. bcvwarpy bodlen, delta(1) k(1) mstart(5) mend(50) xscale(log) xline(63.37) xlab(10 20 40 60 80)

------------------------------------------------------------------------------
Biased Cross-validation for WARPing density estimation, Quartic kernel
------------------------------------------------------------------------------
Biased Cv-value = 0.00592197   M-value =    15   Bandwidth =  15.0000
Biased Cv-value = 0.00592309   M-value =    14   Bandwidth =  14.0000
Biased Cv-value = 0.00592532   M-value =    16   Bandwidth =  16.0000
Biased Cv-value = 0.00592916   M-value =    13   Bandwidth =  13.0000
Biased Cv-value = 0.00593203   M-value =    17   Bandwidth =  17.0000
```

Combinations Stata ado files – Python scripts warpdenpy.ado (Python warping.py)





help gwarprpy [STB-30: snp10 (original version)]

Title

gwarprpy — Adjusted prediction error for WARPing regression (Nadaraya-Watson estimator) (Python script version)

Syntax

gwarprpy {xvar yvar} [*if*] [*in*] , **delta**(#) **selec**(#) **kercode**(#) [**mstart**(#) **mend**(#) **bound**(#) **gen**(*cvval mval hval*) **grsel**(#) **nograph** **graph_options**]

Description

gwarprpy calculates the adjusted prediction error of the WARPing approximation of the Nadaraya-Watson with several penalizing functions in order to find an optimal bandwidth for kernel regression. As a default this command draws a graph of score vs. mval but the user may shift to the score vs. bandwidth graph with the option **grsel**, and displays a table with the five minimal values for the score and corresponding mval and bandwidths.

Options

delta(#) specifies the desired accuracy for estimation (equivalent to the small bin width in WARPing density estimation).

{opt select{(#) refers to the penalizing function according to the following numeric selector codes:

1 = Shibata's model selector
2 = Generalized cross-validation
3 = Akaike's information criterion
4 = Finite prediction error
5 = Rice's T

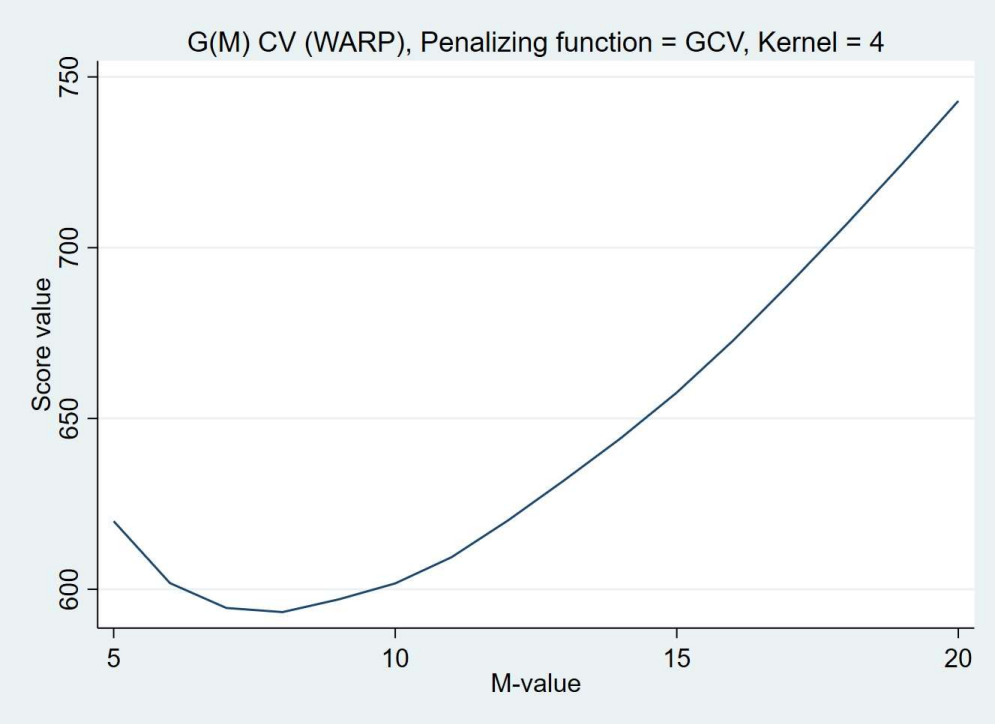
kercode(#) specifies the weight function (kernel) to calculate the required univariate densities according to the following numerical codes:

1 = Uniform
2 = Triangle
3 = Epanechnikov
4 = Quartic (Biweight)
5 = Triweight
6 = Gaussian

mstart(#) permits to specify the initial value for the range of mval (equivalent to the bandwidth) used for the score minimum search. As a default **mstart** = 5.

Selector functions for bandwidth selection in Kernel regression

1.- Shibata’s model selector
2.- Generalized cross-validation
3.- Akaike’s information criterion
4.- Finite prediction error
5.- Rice’s T



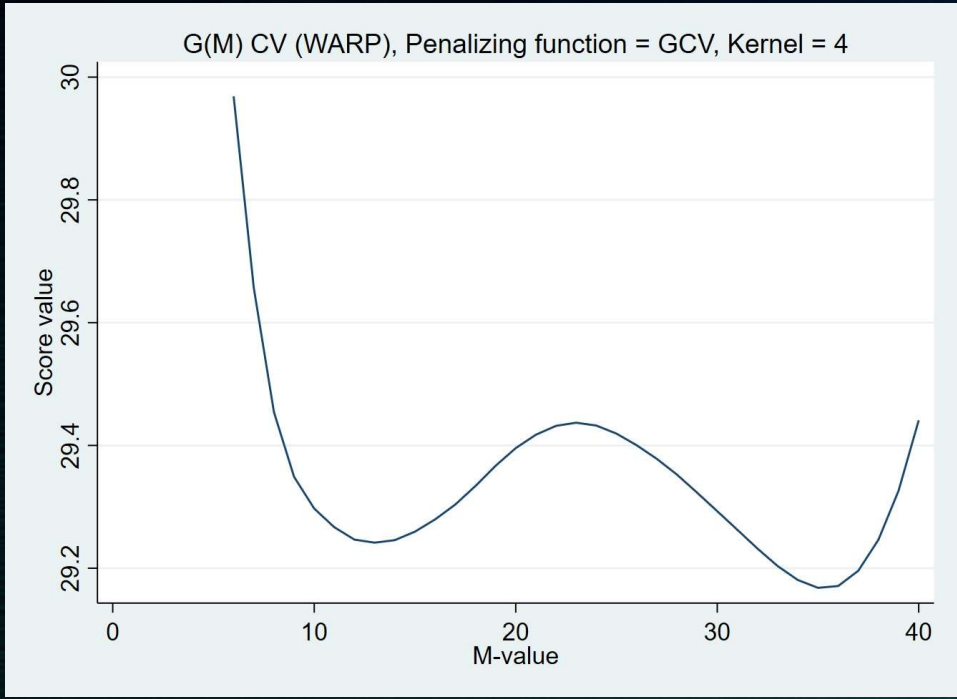
gwarprpy.ado

Generalized Cross-validation

Python

- gwarpreg.py

```
. gwarprpy accel time, d(.3) s(2) k(4) mstart(5) mend(20)

------------------------------------------------------------------------------
G(M) CV (WARP) N-W regression, PF = GCV, Kernel = 4, Delta = .3
------------------------------------------------------------------------------
M-value = 8         Score value = 593.3219        Bandwidth =   2.4000
M-value = 7         Score value = 594.52936       Bandwidth =   2.1000
M-value = 9         Score value = 597.05951       Bandwidth =   2.7000
M-value = 10        Score value = 601.71497       Bandwidth =   3.0000
M-value = 6         Score value = 601.81152       Bandwidth =   1.8000
```



gwarprpy.ado

Generalized Cross-validation

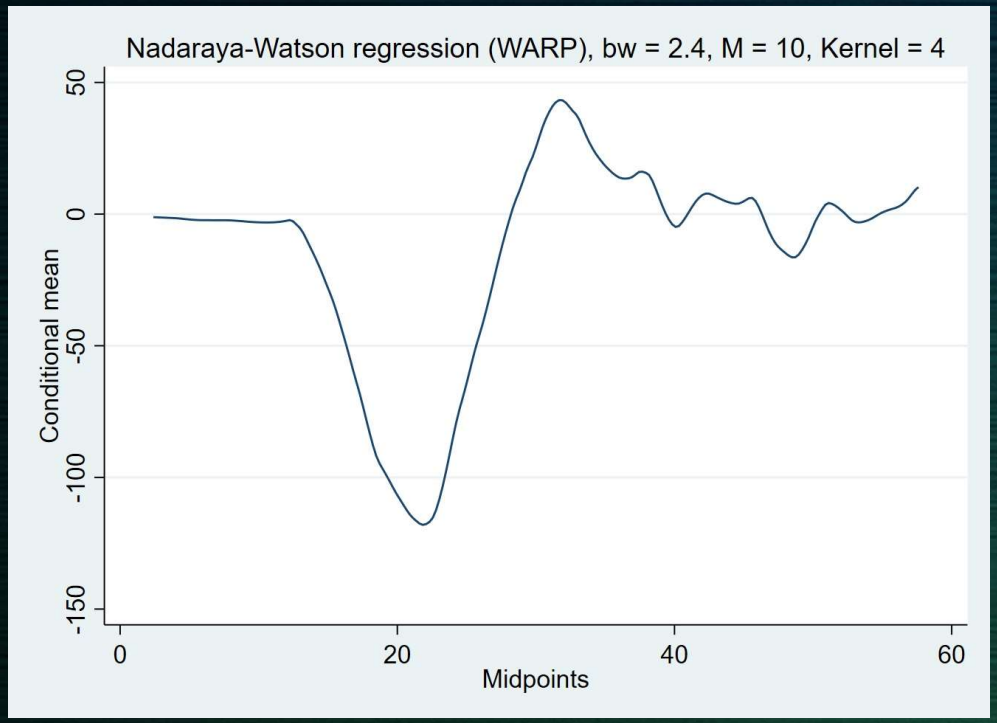
Python

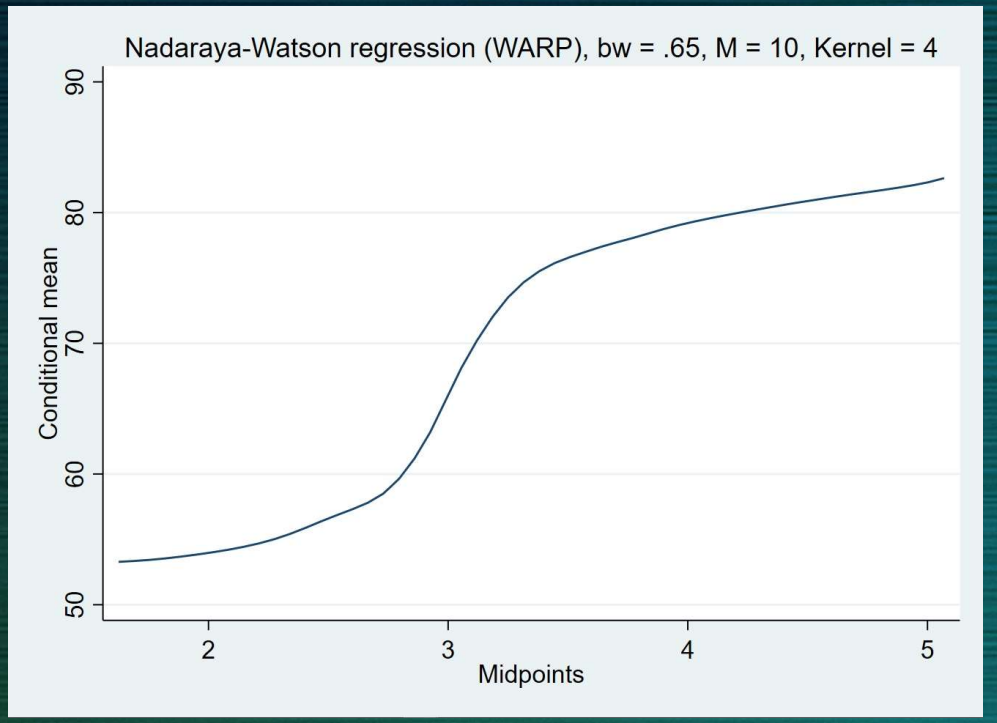
- gwarpreg.py

```
. gwarprpy wait dura, d(.05) s(2) k(4) mstart(6) mend(40)

------------------------------------------------------------------------
G(M) CV (WARP) N-W regression, PF = GCV, Kernel = 4, Delta = .05
------------------------------------------------------------------------
M-value = 35       Score value = 29.168085       Bandwidth =   1.7500
M-value = 36       Score value = 29.170967       Bandwidth =   1.8000
M-value = 34       Score value = 29.18079        Bandwidth =   1.7000
M-value = 37       Score value = 29.195848       Bandwidth =   1.8500
M-value = 33       Score value = 29.203459       Bandwidth =   1.6500
```

```
. gwarprpy wait dura, d(.05) s(2) k(4) mstart(6) mend(20)

------------------------------------------------------------------------
G(M) CV (WARP) N-W regression, PF = GCV, Kernel = 4, Delta = .05
------------------------------------------------------------------------
M-value = 13       Score value = 29.241667       Bandwidth =   0.6500
M-value = 14       Score value = 29.24571        Bandwidth =   0.7000
M-value = 12       Score value = 29.246544       Bandwidth =   0.6000
M-value = 15       Score value = 29.259604       Bandwidth =   0.7500
M-value = 11       Score value = 29.266678       Bandwidth =   0.5500
```

Combinations Stata ado files – Python scripts warpregpy.ado (Python warpregpy.py)

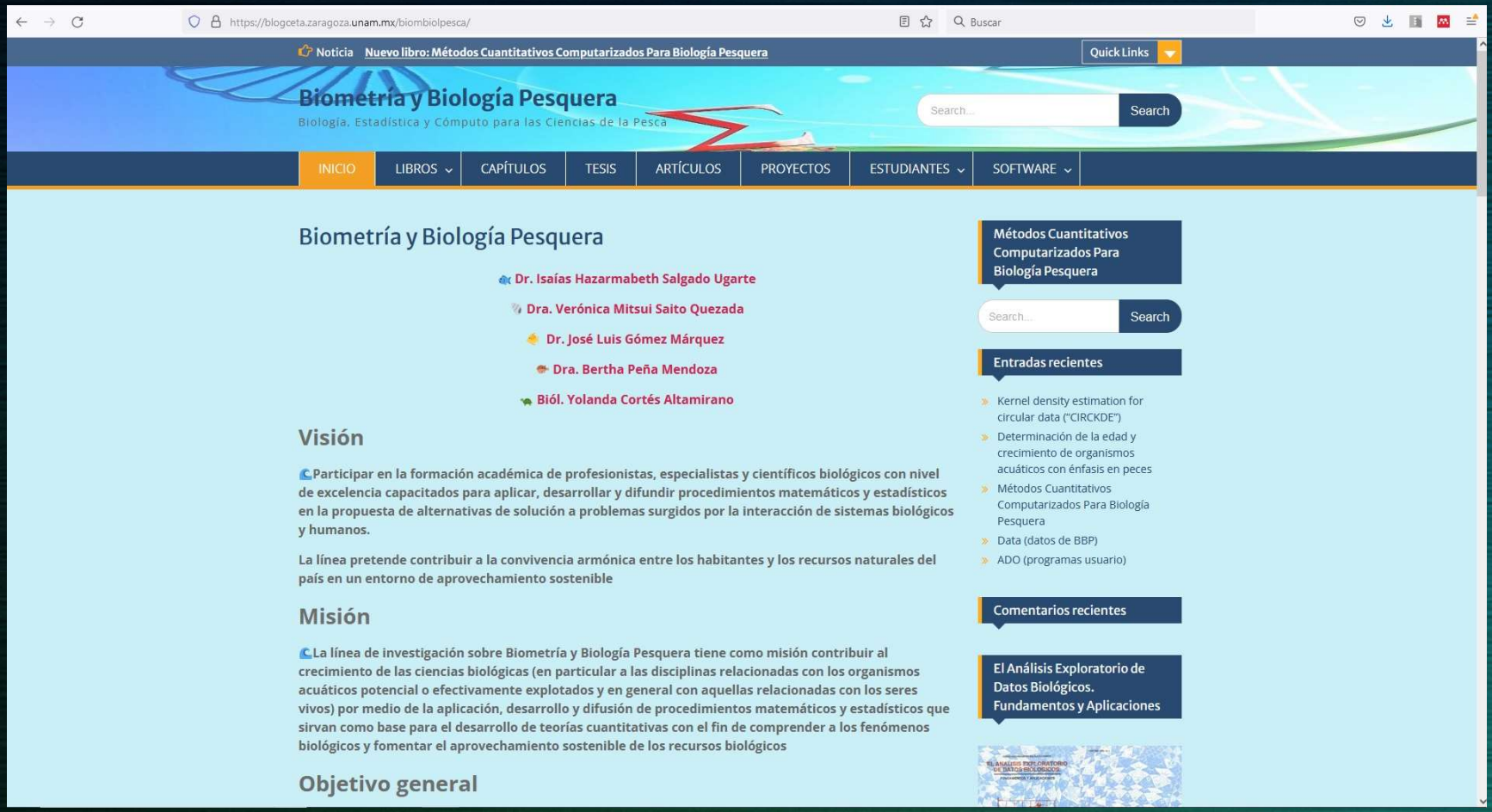




Isaías Hazarmabeth Salgado Ugarte
Verónica Mitsui Saito Quezada

Stata ado programs and Python scripts

https://blogceta.zaragoza.unam.mx/biombiolpesca/



The End

Thank you very much