```
/*************************************************************/
/****************** ADEDIS3.PRG*****************************/
/*************************************************************/
/****** GAUSS PROGRAM TO CARRY OUT DIRECT SEMI- ************/
/*** PARAMETRIC ESTIMATION OF A SINGLE-INDEX MODEL WITH ***/
/***ONE OR MORE DISCRETE COVARIATES.  COMPUTES KERNEL *****/
/******* ESTIMATES IN DO LOOPS TO SAVE MEMORY. ************/
/*************************************************************/
/***REFERENCE:  J.L. HOROWITZ & W. HAERDLE, JASA, *********/
/**********91, 1632-1640, 1996 ****************************/
/*************************************************************/
/***** INSTRUCTIONS FOR USER-SUPPLIED INPUTS ARE **********/
/************ ON LINES 180-235 ****************************/
/*************************************************************/
/****** Joel Horowitz,  98-9-30 ***************************/
/*************************************************************/
/*************************************************************/

format 9,5;
library pgraph;
clear c0, c1, ic0, va,vb,xb,yb,hx,ytp;
output file = OUTPUT.OUT RESET;
"";
"   DIRECT SEMIPARAMETRIC ESTIMATION OF A SINGLE-INDEX MODEL";
"";
output off;
/*************************************************************/
/********** 2ND ORDER KERNEL WITH SUPPORT [-1,1] **********/
/*************************************************************/
 proc(1) = KERN2(v);
 local term;
 term = (15/16)*((1 - v^2)^2);
 term = term.*(abs(v) .<= 1);
 retp(term);
 endp;
/*************************************************************/
/********** 4TH ORDER KERNEL WITH SUPPORT [-1,1] **********/
/*************************************************************/
 proc(1) = KERN4(v,d);
 local term;
 if d < 1.5;  @ ****** Compute kernel ******* @
  term = (105/64)*(1 - 5*v^2 + 7*v^4 - 3*v^6);
  term = term.*(abs(v) .<= 1);
  else;       @ ***** Compute derivative ***** @
  term = (105/64)*(-10*v + 28*v^3 -18*v^5);
  term = term.*(abs(v) .<= 1);
  endif;
 retp(term);
 endp;
/*************************************************************/
/****** Kernel Smoother Routine ***************************/
/*************************************************************/
proc(1) = NADWAT(v);
```

```
local ii,term,nv,dx,dxy,sum, sum1;
  nv = cols(v);
  term = zeros(nv,1);
  ii = 1;
  do until ii > nv;
  dx = kern4((xb - v[ii])./hx,1);
  dxy = dx.*yb;
  sum = sumc(dxy);
  sum1 = sumc(dx);
  sum = sum./(sum1 + 1e-10);
  sum = c0.*(sum .< c0) + sum.*(c0 .<= sum).*(sum .<= c1)
        + c1.*(c1 .< sum);
  sum = sum/(c1 - c0);
  term[ii] = sum;
  ii = ii + 1;
  endo;
  retp(term');
  endp;
/*************************************************************/
/********* Kernel-Smoother with 2nd Order Kernel *********/
/*************************************************************/
proc(1) = NADWAT2(v);
local ii,term,nv,dx,dxy,sum, sum1;
  nv = cols(v);
  term = zeros(nv,1);
  ii = 1;
  do until ii > nv;
  dx = kern2((xb - v[ii])./hx);
  dxy = dx.*yb;
  sum = sumc(dxy);
  sum1 = sumc(dx);
  sum = sum./(sum1 + 1e-10);
  term[ii] = sum;
  ii = ii + 1;
  endo;
  retp(term);
  endp;
/*************************************************************/
/****** Density-Weighted ADE ****************************/
/*************************************************************/
  proc(1)=DEWADE(x,y,h);
  local wx,mm,m,i,j,jj,nx,dx,kew,dwx,b, term;
  nx = rows(x); dx = cols(x);
  b = zeros(dx,1);
  term = zeros(nx,1);
  j = 1;
  do while j <= dx;
  i = 1;
  do while i <= nx;
  jj = 1;
  m = y[i].*ones(1,nx);
  do while jj <= dx;
    kew = (x[i,jj] - x[.,jj]')/h;
    if jj == j;
```

```
    wx = kern4(kew,2);
    else;
    wx = kern4(kew,1);
    endif;
    m = m.*wx;
    jj = jj + 1;
    endo;
    term[i] = sumc(m');
    i = i + 1;
    endo;
    b[j] = sumc(term);
    j = j + 1;
    endo;
  b = -2*b/((nx^2)*h^(dx + 1));
  retp(b);
  endp;
/*************************************************************/
/********** R FUNCTION FOR VARIANCE OF BETA *************/
/********* COMPUTES R(Y,X) FOR A SPECIFIED Z ************/
/*************************************************************/
 proc(1) = rfunc(y,x,h);
   local wx,mm,m,i,j,jj,nx,dx,dwx,kew,r;
  nx = rows(x); dx = cols(x); r = zeros(nx,dx);
  r = zeros(nx,dx);
  j = 1;
  do while j <= dx;
  i = 1;
  do while i <= nx;
  jj = 1;
  m = ones(1,nx);
  do while jj <= dx;
    kew = (x[i,jj] - x[.,jj]')/h;
    if jj == j;
    wx = kern4(kew,2);
    else;
    wx = kern4(kew,1);
    endif;
    m = m.*wx;
    jj = jj + 1;
    endo;
    m = (y[i] - y').*m;
    r[i,j] = -meanc(m')/(h^(dx + 1));
    i = i + 1;
    endo;
    j = j + 1;
    endo;  @ j loop @
  retp(r);
  endp;
/*************************************************************/
/*** PROCEDURE TO COMPUTE P(V|Z), G(V|Z), AND G'(V|Z) ***/
/*************************************************************/
  proc(3) = gzv(y,v,h);
   local i,wv,nv,pv,gv,dgv,kew;
   nv = rows(v);
```

```
   pv = zeros(nv,1);
   gv = zeros(nv,1);
   dgv = zeros(nv,1);
   i = 1;
   do until i > nv;
   kew = (v[i] - v')/h;
   wv = kern4(kew,1);
   pv[i] = meanc(wv')/h;
   gv[i] = meanc(y.*wv')./(h*pv[i]);
   wv = kern4(kew,2);
   dgv[i] = meanc(y.*wv')./(pv[i]*h^2);
   i = i + 1;
   endo;
  retp(pv,gv,dgv);
  endp;
/*************************************************************************/
/************************** MAIN PROGRAM *********************************/
/*************************************************************************/

  DATASET = "PUT PATH TO DATA HERE";  @ **** IDENTIFY INPUT DATA SET **** @
  OPEN FY=^DATASET;
@
  READ THE DATA AND SET UP ARRAYS
@
  N = ROWSF(FY);
  YZ=READR(FY,N);
  CLOSE(FY);

@ ***** TELL THE PROGRAM WHETHER THERE ARE ANY DISCRETE COVARIATES ***** @

   DISCR = 1;  @ ***** 0 IF NO DISCRETE COVARIATES, 1 OTHERWISE ***** @

@ ***** IDENTIFY AND STANDARDIZE THE CONTINUOUS INDEPENDENT VARIABLES ***** @

   X = YZ[.,2:3];   @ ***** Modify this to choose right X from data ***** @
   X = (X - MEANC(X)')./STDC(X)';

@ ***** IDENTIFY THE DEPENDENT VARIABLE ***** @

   Y = YZ[.,1];   @ ***** Modify to choose right Y from data ***** @

@ ****** IDENTIFY THE DISCRETE INDEPENDENT VARIABLES ***** @

    Z = YZ[.,4];  @ ***** Modify this to choose right Z from data ***** @

  n = rows(x);   @ number of observations @
  dx = cols(x);  @ dimension of x @
  dz = 1;
  if discr > 0;
   dz = cols(z); @ dimension of z if there are discrete covariates @
   endif;

@  ******** SET PARAMETERS OF PROGRAM ********* @
```

```
  _intord = 12;  @ number of quadrature knots  @

@
  FACTORS THAT MAY BE USED TO ADJUST RULE-OF-THUMB BANDWIDTHS.  HFAC MULTIPLIES
  RULE-OF-THUMB BANDWIDTH FOR ESTIMATING COEFFICENTS OF DISCRETE COVARIATES.
  HDFAC ADJUSTS BANDWIDTH FOR ESTIMATING COEFFICIENTS OF CONTINUOUS COVARIATES.
  EACH FACTOR CONVERTS A RULE-OF-THUMB BANDWIDTH TO FACTOR*BANDWIDTH
@

  hfac = 1.0;    @ Factor that may be used to adjust bandwidth  @
  hdfac = 1.0;   @ Factor that may be used to adjust bandwidths @

/******** WRITE THE MATRIX OF POSSIBLE VALUES OF Z IF THERE ARE ***********/
/****** DISCRETE COVARIATES.  EACH ROW OF ZMAT IS A MASS POINT OF Z ********/
/*************************************************************************/

    if discr > 0;
     zmat = {0,1};  @ ***** PUT YOUR MATRIX HERE ***** @
     endif;

@ ********* END OF USER-SET PARAMETERS ********* @

  tstart = date;
  indx = seqa(1,1,n);
  nz = 1;
  if discr == 0;
   zmat = 1;
   endif;
  nz = rows(zmat);
  indvec = ones(n,1);
  wgt = zeros(nz,1);
  cntz = zeros(nz,1);
  ic0 = 0;
  b = -1;
  jj = 1;
  do until jj > nz;
    tmp = zmat[jj,.].*indvec;
    ind = selif(indvec, (z .== tmp));
    cntz[jj] = sumc(ind);
    wgt[jj] = sumc(ind)/n;
    jj = jj + 1;
    endo;
  tmp = meanc(cntz);
  hdewad = 5*hdfac*(cntz^(-1/6));
   if dx > 1;
   jj = 1;
   do until jj > nz;
      h = hdewad[jj];
      "DEWADE: jj = ";; jj;
      tmp = zmat[jj,.].*indvec;
      ind = selif(indx,(z .== tmp));
      if jj == 1;
         delta = wgt[jj]*DEWADE(x[ind,.],y[ind,.],h);
         else ;
```

```
        delta = delta~(wgt[jj]*DEWADE(x[ind,.],y[ind,.],h));
        endif ;
      jj = jj+ 1;
      endo ;

   delta=sumc(delta') ;    @ collect the mean over bins @
   b = delta./delta[1];    @ normalize so that first coeff. is 1 @
   ENDIF;

    clear vtmp, ytmp, gz, out;

  "Estimate of beta:  ";; b';

   if discr == 0;
     goto nodiscr;
     endif;

/***********  FIND THE LIMITS OF INTEGRATION *********/

  jj = 1;
  lim1 = zeros(nz,1);
  lim2 = zeros(nz,1);
  do until jj > nz;
    tmp = zmat[jj,.].*indvec;
    ind = selif(indx,(z .== tmp));
    xb = x[ind,.]*b;
    yb = y[ind];
    hx = hfac*stdc(xb)*rows(xb)^(-1/7.5) ;
    lim1[jj]  =  minc(xb) + hx;
    lim2[jj]  =  maxc(xb) - hx;
    jj = jj + 1;
    endo;
  lim1 = maxc(lim1);
  lim2 = minc(lim2);
  if lim2 < lim1;
     output on;
     "Upper Limit Is Below Lower Limit.  HFAC = ";; hfac;
     "";
     endif;
     "";
     "Limits of integration for hfac =";; hfac;; ":";; lim2;; lim1;
     output off;
  limv = lim2|lim1;


/******************** FIND C0 AND C1 ********************/

  jj = 1;
  c0try = zeros(nz,1);
  c1try = zeros(nz,1);
  do until jj > nz;
    tmp = zmat[jj,.].*indvec;
    ind = selif(indx,(z .== tmp));
    xb = x[ind,.]*b;
```

```
    yb = y[ind,.] ;
    hx = 2*hfac*stdc(xb)*rows(xb)^(-1/5) ;
    h2 = hx;
    gtemp = nadwat2(xb');
    gtemp = gtemp';

/*
@ ***** REMOVE COMMENT SYMBOLS TO GRAPH G(V) FOR EACH Z ***** @

      out = xb~gtemp';
      out = sortc(out,1);
       LIBRARY PGRAPH;
       GRAPHSET;
       TITLE("G(V) VS. V FOR FIXED Z");
       XY(out[.,1], out[.,2]);
*/

/*
@ ***** ALTERNATIVE WAY TO COMPUTE C0 AND C1 ***** @

    glo = selif(gtemp,(xb .<= lim1));
    c0try[jj] = maxc(glo);
    glo = selif (gtemp, (xb .>= lim2));
    c1try[jj] = minc(glo);
*/

    temp = minindc(abs(xb - lim1));
    c0try[jj] = gtemp[temp];
    temp = minindc(abs(xb - lim2));
    c1try[jj] = gtemp[temp];
    jj = jj + 1;
  endo;
  c0 = maxc(c0try);
  c1 = minc(c1try);

  "c0 and c1:  ";; c0;; c1;


/************** DO THE INTEGRAL **********************/

  jj = 1;
  integ = zeros(nz,1);
  do until jj > nz;
    tmp = zmat[jj,.].*indvec;
    ind = selif(indx,(z .== tmp));
    xb = x[ind,.]*b;
    yb = y[ind,.] ;
    hx = hfac*stdc(xb)*rows(xb)^(-1/7.5) ;
    h2 = hx;
    integ[jj] = intquad1(&nadwat,limv);
    jj = jj + 1;
  endo;

@  Now compute pairwise differences of integrals and z's @
```

```
 di =  integ[2] - integ[1];
 jj = 3;
 do until jj > nz;
   tmp = integ[jj] - integ[1];
   di = di|tmp;
   jj = jj + 1;
   endo;

@ careful here about same ordering as in enumeration of i's @

 jj = 2;
 do until jj > nz;
   if jj == 2;
     w = zmat[jj,.] - zmat[1,.];
     else;
     w = w|(zmat[jj,.] - zmat[1,.]);
     endif;
   jj = jj + 1;
   endo;

 alphahat = inv(w'w)*w'di;

 nodiscr:

/*********************************************************/
/********** COMPUTE THE VARIANCES ********************/
/*********************************************************/

 if discr == 0;
   c0 = -99999;
   c1 = 99999;
   lim1 = -99999;
   lim2 = 99999;
   endif;

 jj = 1;
 do until jj > nz;
    h = hdewad[jj];
    tmp = zmat[jj,.].*indvec;
    ind = selif(indx,(z .== tmp));
    vtmp = x[ind,.]*b;
    ytmp = y[ind];
    hx = hfac*stdc(vtmp)*rows(vtmp)^(-1/7.5) ;
    h2 = hx;
    {pv,gv,dgv} = gzv(ytmp,vtmp,h2);

    gtmp = 0;
    if dx > 1;
     gtmp = (lim1 .<= vtmp).*(vtmp .<= lim2).*(c0 .<= gv).*(gv .<= c1)
            .*dgv.*x[ind,2:dx]./pv;
     gtmp = meanc(gtmp);   @ ***** GAMMA ***** @
     endif;
    temp = (c0 .<= gv).*(gv .<= c1).*(ytmp - gv)./(wgt[jj]*pv); @ LAMBDA @
```

```
    if jj == 1;
      if dx > 1;
      psi = RFUNC(y[ind,.],x[ind,.],h);
      gam = gtmp;
      endif;
      term = temp;
      nn = n - cntz[jj];
      tvec = ones(cntz[jj],1)|zeros(nn,1);
      last = 0;
      else ;
      term = term|temp;
      last = last + cntz[jj-1];
      nn = n - last - cntz[jj];
      if jj < nz;
       tempp = zeros(last,1)|ones(cntz[jj],1)|zeros(nn,1);
       else;
       tempp = zeros(last,1)|ones(cntz[jj],1);
       endif;
      tvec = tvec~tempp;
      if dx > 1;
        tpp = (RFUNC(y[ind],x[ind,.],h));
        psi = psi|(RFUNC(y[ind],x[ind,.],h));
        gam = gam~gtmp;
        endif;
      endif;
    jj = jj+ 1;
    endo;
psigam = 0;
if dx > 1;
 if discr > 0;
  gam = -(gam[.,2:nz] - gam[.,1]);
  else;
  gam = 0;
  endif;
 psi = psi[.,2:dx] - psi[.,1].*b[2:dx]';
 psi = 2*psi/delta[1];
 psigam = psi*gam;
 endif;
if discr > 0;
 term = term.*tvec;
 term = (term[.,2:nz] - term[.,1] + psigam);
 invw = invpd(w'w);
 va = diag(invw*w'(term'term/n)*w*invw);
 va = va/n;
 endif;
vb = diag(psi'psi/n);
vb = vb/n;

@ *****  COMPUTE T STATISTICS FOR TESTING ALPHA AND BETA ***** @

 if dx >= 2; tb = b[2:dx]./sqrt(vb); endif;
 if discr > 0;
  tstat = alphahat'./sqrt(va');
  endif;
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
output on;
""; "";
"         beta  = " ;; b' ;
 if discr > 0;
  "         alpha = " ;; alphahat' ;
  "         c0    = " ;; c0;
  "         c1    ="  ;; c1;
  endif;
"      h(DEWADE) = " ;; h ;
 if discr > 0;
  "      h(NADWAT) = " ;; h2 ;
  endif;
"      hdfac     = " ;; hdfac;
"      h Factor  = ";;  hfac;
 "Sample size:  ";; n;
 "No. of knots in quadrature:  ";; _intord;

if dx >= 2;
"Estimated std. dev. of bhat[2:dx]:  ";; sqrt(vb');
endif;
 if discr > 0;
  "Estimated std. dev. of alphahat:  ";; sqrt(va');
  endif;
 if dx >= 2;
  "T statistics for bhat[2:dx]:  ";; tb';
  endif;
 if discr > 0;
  "T statistics for alphahat:  ";; tstat;
  endif;
 trun = ethsec(tstart,date)/360000;
 "Running time:  ";; trun;
  output off;
```