```
new;


  /*********************************************************
  **  Semiparametric estimation of a multiple-spell       **
  **  proportional hazards model                          **
  *********************************************************
  **  A Gauss program for Monte Carlo experiments         **
  **  [Estimation of baseline hazard: uncensored case]    **
  *********************************************************/


output file = ld_n-uc.out off;
format  /m1 /rd 12,4;
output off;

et = hsec;


    beta = 1;


   /*********************************************************************
  ***         ln(Ld_0(t)) = -x*beta - u + ep                    ***
  *********************************************************************
  ***  Model specification 1: a Weibull hazard                  ***
  *********************************************************************
  ***  Baseline hazard function  : ld_0(t) = alpha*y             ***
  ***  Integrated baseline hazard: Ld_0(t) = alpha*0.5*y^2       ***
  *********************************************************************
  ***  Model specification 2: a U-shaped hazard                 ***
  *********************************************************************
  ***   ld_0(t) = 0.05*(t/5)^(-2/3) + 0.57*(t/5)^5              ***
  ***   Ld_0(t) = 0.75*(t/5)^(1/3)  + 0.475*(t/5)^6             ***
  *********************************************************************/


specmat = {1,2};

i_spec = 1;
do until i_spec > rows(specmat);

spec = specmat[i_spec];

nmat = {100,500};

i_n = 1;
do until i_n > rows(nmat);

    n = nmat[i_n];              @ sample size @
```

```
if spec == 1; @--- model specification 1 ---@

   @--- Bandwidths ---@

if n==100;
       h1 = 3.0;
       h2 = 3.0;
       hz = 5.0;

elseif n==500;
       h1 = 1.5;
       h2 = 1.5;
       hz = 2.5;

endif;


   @--- Support of weighting functions ---@

      t2l = 3.5|0.5;
      z1l = 1|-1;
      z2l = 1|-1;

   @--- True ld_0 ---@

    alpha = 0.087;
     norm = ln(t2l[1]/t2l[2])/((t2l[1]-t2l[2])*alpha);  @ the integral of w(t)/ld(t) @

    t_min = 0.2;                    @ the minimum of evaluation points @
    t_max = 3.8;                    @ the maximum of evaluation points @
        m = 19;                     @ the number of evaluation points @
    t1mat = seqa(t_min,0.2,m);          @ evaluation points @
     ld_0 = alpha*t1mat;                @ true ld_0 @



elseif spec == 2; @--- model specification 2 ---@

   @--- Bandwidths ---@

if n==100;
       h1 = 2.8;
       h2 = 2.8;
       hz = 5.0;

elseif n==500;
       h1 = 1.3;
       h2 = 1.3;
       hz = 3.0;

endif;

   @--- Support of weighting functions ---@
```

```
    t2l = 5|1;
    z1l = 1|-1;
    z2l = 1|-1;

  @--- True ld_0 ---@

 _intord = 12;
   norm = intquad1(&ihazard,t2l[1]|t2l[2])/(t2l[1]-t2l[2]);   @ the integral of w(t)/ld(t) @

  t_min = 0.4;                              @ the minimum of evaluation points @
  t_max = 5.5;                              @ the maximum of evaluation points @
     m = 18;                               @ the number of evaluation points @
  t1mat = seqa(t_min,0.3,m);                    @ evaluation points @
   ld_0 = 0.05*(t1mat/5)^(-2/3) + 0.57*(t1mat/5)^5;   @ true ld_0 @


endif;

 /***************************************
 ***    Monte Carlo experiments       ***
 ***************************************/



  @--- Start of loop ---@


    seed = 12345;
  result = {};
   n_rep = 100;    @ the number of replications @
     rep = 1;
 do until rep > n_rep;

 /***************************************
 ***    Generating data               ***
 ***************************************/

  x1 = rndns(n,1,seed);             @ covariates @
  x2 = rndn(n,1);
   u = (x1 + x2)/2;                @ fixed effects @


  ep = ln(ln(1/(1-rndu(n,2))));       @ error term that has the CDF F(ep) =  1-exp(-exp(ep)). @


if spec == 1;

 t_1 = sqrt((2/alpha)*exp(-x1*beta - u + ep[.,1]));  @ observed t1 @
 t_2 = sqrt((2/alpha)*exp(-x2*beta - u + ep[.,2]));  @ observed t2 @


elseif spec == 2;

  y1 = exp(-x1*beta - u + ep[.,1]);
```

```
    y2 = exp(-x2*beta - u + ep[.,2]);

   t_1 = {};
   t_2 = {};

    i = 1;
   do until i > n;

      coeff = zeros(19,1);
    coeff[1] = 0.475;
   coeff[18] = 0.75;
   coeff[19] = -y1[i];
        sol = polyroot(coeff);
        sol = maxc(sol.*(real(sol).>0).*(imag(sol).==0)); @ pick up the correct root @
        t_1 = t_1|5*sol^3;


   coeff[19] = -y2[i];
        sol = polyroot(coeff);
        sol = maxc(sol.*(real(sol).>0).*(imag(sol).==0)); @ pick up the correct root @
        t_2 = t_2|5*sol^3;


    i = i + 1;
    endo;

@-------------------------------------------------------------------
idea => To obtain the inverse of Ld_0, solve for x(=(t/5)^(1/3)):
            0.475*x^(18) + 0.75*x - y = 0.
-------------------------------------------------------------------@


   t_1 = real(t_1);  @ observed t1 @
   t_2 = real(t_2);  @ observed t2 @

endif;

   /*****************************************
   ***    Estimation of beta            ***
   *****************************************/


 {bhat, n_iter} = ple(t_1~t_2,x1~x2,beta);

if n_iter == 100;
output on;
" Oops, no convergence ";
output off;
endif;


   z_1 = x1*bhat;   @ estimated index @
   z_2 = x2*bhat;
```

```
  /*********************************************************
  ***  Estimation of ld_0 by the quadrature method      ***
  *********************************************************/


  ldmat = {};

     i = 1;
   do until i > rows(t1mat);

      t1 = t1mat[i];


  @----- Quadrature method -----@


   _intord = 3;

   _intrec = 0;
    ldhat = intquad3(&ftn_q1,t2l,z1l,z2l)/norm;

/* uncomment the following for the combined estimates */

@---@
   _intrec = 0;
    ldhat1 = intquad3(&ftn_q2,t2l,z1l,z2l)/norm;
     ldhat = 0.5*ldhat + 0.5*ldhat1;
@---@

     ldmat = ldmat|ldhat;

    " rep --- i --- ldhat " rep~i~ldhat;

    i = i + 1;
    endo;

 result = result|ldmat';

rep = rep + 1;
endo;

@-------------------------------------------------------@

if     (spec == 1) and (n == 100);     r_ld11e = result;  save r_ld11e;
elseif (spec == 1) and (n == 500);     r_ld12e = result;  save r_ld12e;
elseif (spec == 2) and (n == 100);     r_ld21e = result;  save r_ld21e;
elseif (spec == 2) and (n == 500);     r_ld22e = result;  save r_ld22e;
endif;


@-------------------------------------------------------@
```

```
library pgraph;
graphset;


   _pdate = "";
  _pltype = {3,6,6,6,6,6};

    xy(t1mat,ld_0~meanc(result));
    xy(t1mat,ld_0~median(result));
    xy(t1mat,ld_0~(result[1:5,.]'));


  /*********************************************************
  ***          Printing results                         ***
  *********************************************************/


output on;
" ======================================= ";
" sample size " n;
" the uncensored case ";
" specification " spec;
" no. of rep. "   n_rep;
" bandwidths h1 - h2 - hz ";
h1~h2~hz;
" integrated mse " meanc(meanc((result-ld_0')^2));
output off;


i_n = i_n + 1;
endo;

i_spec = i_spec + 1;
endo;


et = hsec - et;
output on;
" Elapsed Time, in minutes "  et/6000;
output off;


end;

/*************************************************************/
/******    procedure to evaluate the integral     *********/
/*************************************************************/
proc (1) = ihazard(t);
local hf;
hf = 0.05.*((t./5).^(-2/3)) + 0.57.*((t./5).^5);
hf = 1/hf;
retp(hf);
endp;
```

```
/*************************************************************/
/******    partial likelihood estimator of beta   *********/
/*************************************************************/

proc (2) = ple(t,x,beta0);
local n,d,r,b0,tol,max_i,i,tmp,score,d1,info,b1,cr;


      n = rows(t);
      d = x[.,2] - x[.,1];
      r = (t[.,2] .< t[.,1]);

     b0 = beta0;
    tol = 1e-10;
  max_i = 100;
     cr = 1;
      i = 1;
      do until (i > max_i) or (cr < tol);

    tmp = exp(d*b0);
  score = meanc((r - tmp./(1+tmp)).*d);
     d1 = d.*sqrt(tmp./((1+tmp)^2));
   info = d1'd1/n;

     b1 = b0 + invpd(info)*score;
     cr = maxc(abs(b1-b0));

    b0 = b1;

      i = i + 1;
      endo;



retp(b1,i);
endp;



/*************************************************************/
/******    Integrand for the quadrature method    *********/
/*************************************************************/
proc ftn_q1(t2,z1,z2);
local An,Bn,Rn,wt,f;

 An = (t_2 .> vecr(t2)').*kl2((vecr(t1)' - t_1)/h1).*kl4((vecr(z1)' - z_1)/hz).*kl4((vecr(z2)' - z_2)/hz);
 Bn = (t_1 .> vecr(t1)').*kl2((vecr(t2)' - t_2)/h2).*kl4((vecr(z1)' - z_1)/hz).*kl4((vecr(z2)' - z_2)/hz);

 Rn = (sumc(An)/h1)./(sumc(Bn)/h2);

 wt = (1/(t2l[1]-t2l[2]))*kl2(vecr(z1)).*kl2(vecr(z2));
```

```
 f = wt.*exp(vecr(z2)-vecr(z1)).*Rn;
 f = f';

retp(f);
endp;

/*************************************************************/
/******    Integrand for the quadrature method    *********/
/*************************************************************/
proc ftn_q2(t2,z1,z2);
local An,Bn,Rn,wt,f;

 An = (t_2 .> vecr(t1)').*kl2((vecr(t2)' - t_1)/h2).*kl4((vecr(z1)' - z_1)/hz).*kl4((vecr(z2)' - z_2)/hz);
 Bn = (t_1 .> vecr(t2)').*kl2((vecr(t1)' - t_2)/h1).*kl4((vecr(z1)' - z_1)/hz).*kl4((vecr(z2)' - z_2)/hz);

 Rn = (sumc(An)/h2)./(sumc(Bn)/h1);

 wt = (1/(t2l[1]-t2l[2]))*kl2(vecr(z1)).*kl2(vecr(z2));


 f = wt.*exp(vecr(z1)-vecr(z2))./Rn;
 f = f';

retp(f);
endp;




/*************************************************************/
/*************** 2nd Order Kernel Function ***************/
/*************************************************************/
proc(1) = kl2(v);
local term;
term = (15/16)*((1 - v^2)^2);
term = term.*(abs(v) .<= 1);
retp(term);
endp;


/*************************************************************/
/*************** 4th Order Kernel Function ***************/
/*************************************************************/
proc(1) = kl4(v);
local term;
term = (105/64)*(1 - 5*v^2 + 7*v^4 - 3*v^6);
term = term.*(abs(v) .<= 1);
retp(term);
endp;


/*************************************************************/
```

```
/*************** 6th Order Kernel Function ***************/
/*********************************************************/
proc(1) = kl6(v);
local term;
term = (315/2048)*(15 - 140*v^2 + 378*v^4 - 396*v^6 + 143*v^8);
term = term.*(abs(v) .<= 1);
retp(term);
endp;
```