CMAT© Newsletter: December 2012

Wolfgang M. Hartmann

December 2012

Contents

1 General Remarks

This last six months not many new function were added. The development was shifted from the old Watcom C/C++ and Fortran compilers to the new *MS Visual C 2010* for C/C++ development and *Intel Parallel Studio XE 2013* for Fortran/Fortran90 compilation and mixed binding. Since the Debug version of this new platform has a much more sophisticated search for errors in the code, especially memory allocation bugs, we were lucky to find many bugs in our code and fixing hopefully all of them. The optimization phase for producing the Release version version takes more than thirty minutes and is for most applications by a factor of 2 to 4 faster than the debug version.
Comparison of three versions with different DELL Precision Desktops:

1. PC1: DELL Precision 1650 bought in 2012: running 32-bit Windows 7 Professional,
   one 250 GB hard drive: 3.5 inch Serial ATA (7.200 Rpm),
   3rd Gen Intel Core I7-3770 Processor (Quad Core 3.40 GHz Turbo, 8 MB w/HD Graphics 4000),
   4 GB RAM 1600 MHz DDR3 Non-ECC,
   using MS Visual Studio and Intel Parallel Studio XE 213, **compiled in Release mode** (optmization process takes about 30 minutes),
   size of the executable 21.5 MB

2. PC2: DELL Precision 3500 bought in 2012: running 64-bit Windows 7 Professional,
   two 300 GB hard drives: 15,000 Rpm SAS,
   Intel Xeon W3670 (Six Core 3.20 GHz, 12 MB Cache, 4.8 GT/s Intel QPI),
   12 GB RAM 1333 MHz DDR3 ECC UDIMM,
   using MS Visual Studio and Intel Parallel Studio XE 213, **compiled in Debug mode**,
   size of the executable 38.6 MB

3. PC3: DELL Precision 1500 bought in 2001: running 32-bit Windows XP Professional
   two 35 GB SCSI hard drives (10.000 Rpm) and an older Intel Pentium Processor,
   using very slow Watcom C/C++ and Fortran compilers **in Debug mode**,
   size of the executable 25.9 MB

Example in cmat/test	Time in Secs for PC1	Time in Secs for PC2	Time in Secs for PC3
tborut2.inp	2812	4785	6605
tlrallv2.inp	3418	6466	12874
tlrallv3.inp	1853	2752	9917
tlrallv4.inp	1450	2253	8027
tmicro2.inp	2736	5627	3576 ???
tmixrpana.inp	345	1382	4839
tmixrpan2.inp	342	1386	4532
trand2.inp	1799	9443	24425
tranfor2.inp	2389	3891	7570
tsvm12.inp	1156	3283	-
tsvm13.inp	1542	2781	-
tsvm23.inp	1162	5123	-
tsvm24.inp	18349	100491	-
tsmp3.inp	19262	81609	-

1.1 New Functions

horner computation of Horner scheme

kde one- and two-dimensional kernel density estimation

mvsvm multivariate SVM for Regression

permcomb permutation and combination

prefname generation of prefix names

rvm Relevance Vector Machines for classification and regression

sdcspm testing Srivastava's condition and computing measures for search performance

1.2 Fixed Bugs

A large number of bugs were fixed. Due to the compilation with quite different compilers we were able to identify and correct some older bugs.

4 Extensions to Various Functions

4.1 Extensions of `cfa()` Function

For multiple sample applications, now polychoric correlations together with weight (asymptotic covariance) matrices should be working correctly.

4.2 Extensions of `hbttest()` Function

In addition to the input of raw data we now permit the input of two sets of triplets $(mean, stdev, nobs)$ of data. For that kind of data input we must set the value of `optn[3]` to nonzero.
The following table contains only means and standard deviations of 4 groups of experiments (control an different treatments). For each of the measurements we have to add `nobs` = 4:

```
  cnam = [" rnam m1 s1 m2 s2 m3 s3 m4 s4 "];
  data= [" Ceramides_C16_0          0.18   0.00    0.39   0.01    0.33   0.01    0.13   0.02
           Ceramides_C18_0          0.02   0.00    0.06   0.00    0.05   0.00    0.01   0.00
           Ceramides_C20_0          0.06   0.00    0.11   0.01    0.11   0.00    0.06   0.01
           Ceramides_C22_0          1.23   0.07    1.36   0.05    1.58   0.06    1.20   0.15
           Ceramides_C23_0          0.32   0.01    0.40   0.01    0.43   0.01    0.30   0.04
           Ceramides_C24_1          1.19   0.06    1.44   0.05    1.53   0.05    1.21   0.17
           Ceramides_C24_0          1.29   0.08    1.44   0.05    1.60   0.05    1.20   0.16
           Ceramides_saturated      3.09   0.15    3.77   0.13    4.10   0.12    2.90   0.38
           Ceramides_unsaturated    1.19   0.06    1.44   0.05    1.53   0.05    1.21   0.17
           Ceramides_total          4.28   0.21    5.21   0.18    5.62   0.17    4.11   0.55
           SPM_16_1_OH              0.04   0.00    0.04   0.00    0.05   0.00    0.03   0.00
           SPM_18_1                 0.03   0.00    0.03   0.00    0.04   0.00    0.02   0.00
..........................................................................................
           Chol_CE_20_5             0.007  0.00    0.010  0.003   0.016  0.001   0.009  0.00
           Chol_CE_20_4             0.021  0.002   0.029  0.005   0.037  0.005   0.019  0.00
           Chol_saturated           0.48   0.04    0.60   0.05    0.63   0.05    0.41   0.07
           Chol_monounsaturated     0.69   0.03    0.95   0.13    0.98   0.07    0.53   0.11
           Chol_polyunsaturated     0.48   0.03    0.68   0.09    0.77   0.03    0.42   0.08
           Chol_total_unsaturated   1.17   0.03    1.62   0.22     .75   0.06    0.96   0.19
           Chol_total_CE            1.65   0.06    2.22   0.27    2.38   0.04    1.36   0.26
           Chol_Free               20.25  0.78   23.19   0.92   24.30   0.47    16.16  2.11
```

We are setting `optn[3]` to 1 indicating the input of triplet data:

```
optn = [ 0 ,   /* ipri */
         . ,   /* alpha, def=.05 */
         1 ];  /* job=1: input tripels (mean,stdev,nobs) */
nr = nrow(data); nc = ncol(data);
print "nr=", nr, " nc=", nc;
```

```
rnam = data[ ,1]; cnam = [" Mean Stdev Nobs "];
```

From the table above

```
print "Compare Group1 with Group2";
dat1 = data[, [2 3] ] -> cons(nr,1,4.);
dat1 = rname(dat1,rnam); dat1 = cname(dat1,cnam);
dat2 = data[, [4 5] ] -> cons(nr,1,4.);
dat2 = rname(dat2,rnam); dat2 = cname(dat2,cnam);
print "Data1 and Data2=",dat1,dat2;

< ttst,mom,ftest > = hbttest(dat1,dat2,optn);
print "t_Test Group 1 vs. Group 2", ttst;
```

When both variances are zero, the probability can be 0 or 1, depending on the fact whether the mean values are different or the same:

	Crit_T	T_Value	NDF	Prob
TTest_Ceramides_C16_0	2.4469	-42.000	6.0000	1e-008
Sattw_Ceramides_C16_0	.	-42.000	3.0000	0.0000
TTest_Ceramides_C18_0	2.4469	.	6.0000	0.00000
Sattw_Ceramides_C18_0	.	.	6.0000	0.00000
TTest_Ceramides_C20_0	2.4469	-10.0000	6.0000	0.0001
Sattw_Ceramides_C20_0	.	-10.0000	3.0000	0.0021
TTest_Ceramides_C22_0	2.4469	-3.0224	6.0000	0.0233
Sattw_Ceramides_C22_0	.	-3.0224	5.4289	0.0264
TTest_Ceramides_C23_0	2.4469	-11.314	6.0000	0.0000
Sattw_Ceramides_C23_0	.	-11.314	6.0000	0.0000
TTest_Ceramides_C24_1	2.4469	-6.4018	6.0000	0.0007
Sattw_Ceramides_C24_1	.	-6.4018	5.8110	0.0008
TTest_Ceramides_C24_0	2.4469	-3.1800	6.0000	0.0191
Sattw_Ceramides_C24_0	.	-3.1800	5.0335	0.0243
TTest_Ceramides_saturated	2.4469	-6.8516	6.0000	0.0005
Sattw_Ceramides_saturated	.	-6.8516	5.8812	0.0005
TTest_Ceramides_unsaturated	2.4469	-6.4018	6.0000	0.0007
Sattw_Ceramides_unsaturated	.	-6.4018	5.8110	0.0008
TTest_Ceramides_total	2.4469	-6.7248	6.0000	0.0005
Sattw_Ceramides_total	.	-6.7248	5.8629	0.0006
TTest_SPM_16_1_OH	2.4469	.	6.0000	1.00000
Sattw_SPM_16_1_OH	.	.	6.0000	1.00000
TTest_SPM_18_1	2.4469	.	6.0000	1.00000
Sattw_SPM_18_1	.	.	6.0000	1.00000
TTest_Chol_CE_20_5	2.4469	-2.0000	6.0000	0.0924

```
          Sattw_Chol_CE_20_5 |      .       -2.0000    3.0000   0.13933
          TTest_Chol_CE_20_4 |    2.4469      -2.9711    6.0000    0.0249
          Sattw_Chol_CE_20_4 |      .       -2.9711    3.9360    0.0419
       TTest_Chol_saturated  |    2.4469      -3.7482    6.0000    0.0095
       Sattw_Chol_saturated  |      .       -3.7482    5.7242    0.0104
  TTest_Chol_monounsaturated |    2.4469      -3.8976    6.0000    0.0080
  Sattw_Chol_monounsaturated |      .       -3.8976    3.3186    0.0249
  TTest_Chol_polyunsaturated |    2.4469      -4.2164    6.0000    0.0056
  Sattw_Chol_polyunsaturated |      .       -4.2164    3.6585    0.0163
TTest_Chol_total_unsaturated |    2.4469      -4.0534    6.0000    0.0067
Sattw_Chol_total_unsaturated |      .       -4.0534    3.1115    0.0252
         TTest_Chol_total_CE |    2.4469      -4.1217    6.0000    0.0062
         Sattw_Chol_total_CE |      .       -4.1217    3.2956    0.0216
             TTest_Chol_Free |    2.4469      -4.8750    6.0000    0.0028
             Sattw_Chol_Free |      .       -4.8750    5.8436    0.0030
```

We also fixed that small problem when this function returned missing values when one of the variances are zero. As long as at least one of the variances is larger than zero we now return the correct result.

4.3 Extensions to `sortrow()` Function

A new return argument `rank` has been added to the `sortrow()` function:

< ind,bmat,rank > = sortrow(amat<,key<,optn>>)

The returned argument `rank` is an N integer vector with the ranking of the original rows indicating the tied values, i.e. $rank[i] \geq rank[i-1]$, $i = 2, \ldots, N$. The last entry of the `rank` vector shows how many different rows the data have w.r.t. the key columns.

The first return argument `ind` is an N integer vector with the indices of the original rows, where `j = ind[i]` gives the row number j of the unsorted **A** for the row i in **B**.

The following shows a small illustrative example:

```
amat = [ 5  3  1,
         7  4  2,
         3  5  3,
         5  3  4,
         7  4  5,
         3  5  6,
         5  3  4];
keycols = [ 1 2 ];
```

```
print "Sort ascending";
< ind,bmat,rank > = sortrow(amat,keycols);
print "Ind=", ind;
print "Bmat=", bmat;
print "Rank=", rank;
```

```
print "Sort descending";
< ind,bmat,rank > = sortrow(amat,keycols,1);
print "Ind=", ind;
print "Bmat=", bmat;
print "Rank=", rank;
```

Sort ascending

Ind=

	1
1	3
2	6
3	1
4	4
5	7
6	2
7	5

Bmat=

	1	2	3
1	3	5	3
2	3	5	6
3	5	3	1
4	5	3	4
5	5	3	4
6	7	4	2
7	7	4	5

Rank=

	1
1	1
2	1
3	2
4	2
5	2
6	3
7	3

Sort descending

Ind=

	1
1	2
2	5
3	1
4	4
5	7
6	3
7	6

Bmat=

	1	2	3
1	7	4	2
2	7	4	5
3	5	3	1
4	5	3	4
5	5	3	4
6	3	5	3
7	3	5	6

Rank=

	1
1	1
2	1
3	2
4	2
5	2
6	3
7	3

4.4 Extensions to `encrypt()` Function

For the `encrypt()` funtion the *SHA-3* method was made available which encodes a password into a 64 char string password. Note, that SHA-2, and MD5 algorithms encode into 32-char passwords. The latter (MD5) is implemented but not recommended since it may be hacked by professional hackers.

4.5 Extensions to some rotation Functions

In functions like `cfa()` and `pca()` it was possible to use the runtime option `SECOND` for running the original Fortran functions by Ogasarawa (1998). In the past even when CMAT was executed in batch mode its output was always written on the screen. Now it is included properly into the `xxx.txt` file.

5 New Developments

5.1 The `horner()` Function

dc = horner(coef,xval<,optn>)

Purpose: Compute the result of the *Horner* algorithm. If `xval` is a scalar, the result is the same as that of the `polyval(coef,x)` function.

Input: **coef** Must be a n vector of real or complex values.

xval Must be a real or complex scalar or n or $n-1$ vector. If `xval` is a n vector the entry with largest index is not used (irrelevant).

optn For nonmissing `optn[1]` unequal zero the computation is done in reverse order.

Output: The only return is a real or complex scalar which is the result of the *Horner* algorithm.

Restrictions: 1. Neither of the first two input arguments may have missing values nor string values.

Relationships: polyval()

Examples: 1. Scalar `xval`:

```
/* y = ((2. * 2 + 3.) * 2. + 4.) * 2. + 5. = 2*8 + 3*4 + 4*2 + 5 = 41 */
coef = [ 2. 3. 4. 5. ];
xval = 2.;
y = horner(coef,xval);
print "Y=", y;
```

```
Y= 41.000
```

2. Vector `xval`:

```
/* y = ((2. * 5 + 3.) * 5. + 4.) * 10 + 5. = 2*250 + 3*50 + 4*10 + 5 = 695 */
coef = [ 2. 3. 4.  5. ];
xval = [ 5. 5. 10. 0. ]; /* the last entry is not used */
y = horner(coef,xval);
print "Y=", y;
```

```
Y= 695.000
```

3. Scalar `xval`: execute in reverse order:

```
/* y = ((5 * 2 + 4) * 2 + 3) * 2 + 2 = 5*8 + 4*4 + 3*2 + 2 = 64 */
coef = [ 2. 3. 4. 5. ];
xval = 2.;
y = horner(coef,xval,1);
print "Y=", y;
```

```
Y= 64.000
```

4. Vector `xval`: execute in reverse order:

```
/* y = ((5 * 10 + 4) * 5 + 3) * 5 + 2 = 5*250 + 4*25 + 3*5 + 2 = 1367 */
coef = [ 2. 3. 4.  5. ];
xval = [ 5. 5. 10. 0. ]; /* the last entry is not used */
y = horner(coef,xval,1);
print "Y=", y;
```

```
Y= 1367.00
```

5.2 The `kde()` Function

< gof,dens,mesh > = kde(data,optn)

Purpose: The `kde()` function can be used either for one- or two-dimensional Gaussian kernel density estimation with bandwith estimation. The implementation is based on a Matlab program by Botev et al. (2010).

Input: **data** must be either some N vector or some $N \times 2$ matrix of data for one- resp. two-dimensional kernel density estimation.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
`"abstol"`	real	an absolute termination criterion for the zero finding algorithm. The default is 10^{-8}.
`"maxf"`	int	upper range of the number of function evaluations for the zero finding algorithm. The default is 400.
`"maxit"`	int	upper range of the number of iterations for the zero finding algorithm. The default is 200.
`"ngrid"`	int	the number n of grid points of the mesh. The default is $n = 2^14$ for one- and $n = 2^8$ for two dimensional KDE.
`"reltol"`	real	a relative termination criterion for the zero finding algorithm. The default is 10^{-6}.

Output: **gof** column vector with some scalar results:

1. failure
2. computation time
3. number of grid points (same in two dimensions)
4. band width first dimension
5. band width second dimension
6. number of function calls in optimization
7. function value at zero (should be close to zero)
8. band width

dens is either a n vector or a $n \times n$ matrix of densities.

mesh is either a n vector or a $n \times n$ matrix of the x or (x, y) mesh.

Restrictions: 1. No missing values or complex data are permitted.

Relationships: fft(), ffti()

Examples: 1. One-dimensional KDE: Small data set

```
dat12 = [ 0.5377   1.8339  -2.2588   0.8622   0.3188
         -1.3077  -0.4336   0.3426   3.5784   2.7694
         32.3002  41.0698  36.4508  34.8739  36.4295
         34.5901  34.7517  37.9794  37.8181  37.8344
         55.6715  53.7925  55.7172  56.6302  55.4889
         56.0347  55.7269  54.6966  55.2939  54.2127 ] ;
 nc = ncol(dat12); print "nc=",nc;


 optn = [ "print"         2 ,
          "ngrid"       100 ,
          "maxfu"       200 ,
          "abstol"    1.e-8 ,
          "reltol"    1.e-6 ];
 < gof,dens,mesh > = kde(dat12,optn);
 print "GOF=",gof;
 print "Dens=",dens;
 print "Mesh=",mesh;
```

```
    Kernel Density Estimation of 30 Points in 1 Dimensions

  Dim=1: Minimum=   -8.1477 Maximum=   62.5191 Range=   70.6668

    Zero 0.000556002 found with F=0 after 13 function calls.
                      Bandwidth=1.6663
                       Computer Time=0


GOF=
           |         1
-----------------------
   Failure |   0.00000
      Time |   0.00000
  Bandw[1] |    1.6663
  Bandw[2] |       .
  GridPnts |    128.00
    Fcalls |    13.000
     Fzero |   0.00000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000


Dens=
   |         1         2         3         4         5         6
```

1	0.00005	0.00011	0.00028	0.00068	0.00147	0.00292
	7	8	9	10	11	12
1	0.00529	0.00884	0.01372	0.01988	0.02708	0.03487
	13	14	15	16	17	18
1	0.04253	0.04916	0.05378	0.05563	0.05438	0.05036
	19	20	21	22	23	24
1	0.04436	0.03736	0.03018	0.02339	0.01731	0.01213
	25	26	27	28	29	30
1	0.00797	0.00486	0.00272	0.00140	0.00065	0.00028

.....

	121	122	123	124	125	126
1	0.01144	0.00606	0.00291	0.00127	0.00050	0.00018
	127	128				
1	0.00006	0.00002				

Mesh=

	1	2	3	4	5	6
1	-8.1477	-7.5913	-7.0348	-6.4784	-5.9220	-5.3655
	7	8	9	10	11	12
1	-4.8091	-4.2527	-3.6962	-3.1398	-2.5834	-2.0270
	13	14	15	16	17	18
1	-1.4705	-0.91409	-0.35766	0.19877	0.75520	1.3116
	19	20	21	22	23	24
1	1.8681	2.4245	2.9809	3.5374	4.0938	4.6502

```
   |        25        26        27        28        29        30
---------------------------------------------------------------------
 1 |    5.2067    5.7631    6.3195    6.8760    7.4324    7.9888

.....................................................................

   |       121       122       123       124       125       126
---------------------------------------------------------------------
 1 |    58.624    59.181    59.737    60.293    60.850    61.406

   |       127       128
-----------------------
 1 |    61.963    62.519
```

2. One-dimensional KDE: Medium size data set N=300

```
print "Generate a Gaussian mixture with distant modes";
print "nor2 is Ziggurat method";
nr = 100; nc = 1;
dat1 = rand(nr,nc,'g',"nor2");
dat2 = 2. * rand(nr,nc,'g',"nor2") + 35.;
dat3 = rand(nr,nc,'g',"nor2") + 55.;
data = dat1 |> dat2 |> dat3;
print "nr=",nrow(data), " nc=",ncol(data);


ng = 1 << 14;
optn = [ "print"         2 ,
         "ngrid"        ng ,
         "maxfu"       200 ,
         "abstol"    1.e-8 ,
         "reltol"    1.e-6 ];
< gof,dens,mesh > = kde(data,optn);
print "GOF=",gof;
print "Dens=",dens[1:20];
print "Mesh=",mesh[1:20];
```

```
    Kernel Density Estimation of 300 Points in 1 Dimensions

 Dim=1: Minimum=  -8.42461 Maximum=   63.3614 Range=    71.786

   Zero 5.38656e-005 found with F=0 after 28 function calls.
                       Bandwidth=0.52686
                        Computer Time=0
```

```
GOF=
           |          1
-----------------------
   Failure |   0.000000
      Time |   0.000000
  Bandw[1] |   0.526860
  Bandw[2] |         .
  GridPnts |    16384.0
    Fcalls |    28.0000
     Fzero |   0.000000
    unused |   0.000000
    unused |   0.000000
    unused |   0.000000
```

3. Two-dimensional KDE: Small data set: N=40

```
dat21 = [ 0.5377    0.6715,  1.8339   -1.2075, -2.2588    0.7172,
          0.8622    1.6302,  0.3188    0.4889, -1.3077    1.0347,
         -0.4336    0.7269,  0.3426   -0.3034,  3.5784    0.2939,
          2.7694   -0.7873, -1.3499    0.8884,  3.0349   -1.1471,
          0.7254   -1.0689, -0.0631   -0.8095,  0.7147   -2.9443,
         -0.2050    1.4384, -0.1241    0.3252,  1.4897   -0.7549,
          1.4090    1.3703,  1.4172   -1.7115,  3.3978   -1.0891,
          3.2586    0.0326,  3.8192    0.5525,  3.8129    1.1006,
          2.6351    1.5442,  3.4699    0.0859,  3.3351   -1.4916,
          4.1277   -0.7423,  4.5933   -1.0616,  4.6093    2.3505,
          2.6363   -0.6156,  3.5774    0.7481,  2.2859   -0.1924,
          2.3865    0.8886,  3.4932   -0.7648,  5.0326   -1.4023,
          2.7303   -1.4224,  3.8714    0.4882,  3.2744   -0.1774,
          4.6174   -0.1961 ];
nr = nrow(dat22); print "nr=",nr;


optn = [ "print"         2 ,
         "ngrid"       100 ,
         "maxfu"       200 ,
         "abstol"    1.e-8 ,
         "reltol"    1.e-6 ];
< gof,dens,mesh > = kde(dat21,optn);
print "GOF=",gof;
print "Dens=",1.e4 * dens[1:20,1:20];
print "Mesh=",mesh[1:20,];


    Kernel Density Estimation of 40 Points in 2 Dimensions
```

```
   Dim=1: Minimum=  -4.08165 Maximum=   6.85545 Range=   10.9371
   Dim=2: Minimum=    -4.268 Maximum=    3.6742 Range=    7.9422

  Zero 0.010826 found with F=-6.245e-017 after 11 function calls.
                  Bandwidth= [0.945175 1.06386]
                        Computer Time=1
     \begin{verbatim}
GOF=
           |         1
-----------------------
   Failure |   0.00000
      Time |   1.00000
  Bandw[1] |   0.94518
  Bandw[2] |    1.0639
  GridPnts |    128.00
    Fcalls |    11.000
     Fzero |   -6e-017
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000


Dens=
   |         1         2         3         4         5
-------------------------------------------------------
 1 |   0.00068   0.00074   0.00086   0.00106   0.00137
 2 |   0.00071   0.00077   0.00090   0.00111   0.00142
 3 |   0.00078   0.00085   0.00098   0.00119   0.00152
 4 |   0.00089   0.00096   0.00110   0.00133   0.00167
 5 |   0.00105   0.00112   0.00127   0.00152   0.00189
 6 |   0.00126   0.00134   0.00151   0.00178   0.00218
 7 |   0.00153   0.00162   0.00181   0.00211   0.00256
 8 |   0.00189   0.00199   0.00220   0.00254   0.00304
 9 |   0.00233   0.00245   0.00270   0.00309   0.00366
10 |   0.00290   0.00304   0.00332   0.00377   0.00442
11 |   0.00361   0.00377   0.00410   0.00462   0.00538
12 |   0.00449   0.00468   0.00507   0.00569   0.00656
13 |   0.00560   0.00582   0.00628   0.00700   0.00803
14 |   0.00696   0.00723   0.00777   0.00863   0.00984
15 |   0.00864   0.00896   0.00962   0.01063   0.01206
16 |   0.01071   0.01109   0.01188   0.01309   0.01478
17 |   0.01325   0.01371   0.01464   0.01609   0.01811
18 |   0.01634   0.01690   0.01802   0.01976   0.02216
19 |   0.02011   0.02078   0.02213   0.02421   0.02707
20 |   0.02469   0.02549   0.02711   0.02960   0.03301
.......................................................
```

	16	17	18	19	20
1	0.04713	0.06449	0.08773	0.11860	0.15927
2	0.04757	0.06506	0.08845	0.11950	0.16041
3	0.04847	0.06619	0.08988	0.12132	0.16271
4	0.04985	0.06793	0.09208	0.12409	0.16620
5	0.05173	0.07030	0.09506	0.12784	0.17092
6	0.05416	0.07334	0.09888	0.13265	0.17696
7	0.05719	0.07713	0.10362	0.13859	0.18440
8	0.06090	0.08174	0.10937	0.14576	0.19335
9	0.06536	0.08727	0.11624	0.15429	0.20396
10	0.07070	0.09384	0.12435	0.16434	0.21639
11	0.07702	0.10160	0.13389	0.17607	0.23086
12	0.08450	0.11071	0.14503	0.18972	0.24760
13	0.09331	0.12139	0.15801	0.20554	0.26691
14	0.10367	0.13388	0.17311	0.22385	0.28913
15	0.11583	0.14846	0.19065	0.24499	0.31467
16	0.13012	0.16548	0.21100	0.26939	0.34398
17	0.14687	0.18534	0.23460	0.29753	0.37761
18	0.16650	0.20848	0.26198	0.32999	0.41619
19	0.18949	0.23546	0.29370	0.36741	0.46044
20	0.21641	0.26687	0.33047	0.41055	0.51118

Mesh=

	1	2
1	-4.0816	-4.2680
2	-3.9955	-4.2055
3	-3.9094	-4.1429
4	-3.8233	-4.0804
5	-3.7372	-4.0179
6	-3.6511	-3.9553
7	-3.5649	-3.8928
8	-3.4788	-3.8302
9	-3.3927	-3.7677
10	-3.3066	-3.7052
11	-3.2205	-3.6426
12	-3.1343	-3.5801
13	-3.0482	-3.5176
14	-2.9621	-3.4550
15	-2.8760	-3.3925
16	-2.7899	-3.3299
17	-2.7037	-3.2674
18	-2.6176	-3.2049

```
19 |   -2.5315   -3.1423
20 |   -2.4454   -3.0798
```

4. Two-dimensional KDE: Gaussian Mixture with distant modes: N=1000

```
nr = 500;
dat1 = rand(nr,2,'g',"nor2");
dat2 = rand(nr,1,'g',"nor2") + 3.5;
dat3 = rand(nr,1,'g',"nor2");
data = dat1 |> (dat2 -> dat3);
print "nr=",nrow(data), " nc=",ncol(data);


ng = 1 << 8;
optn = [ "print"         2 ,
         "ngrid"        ng ,
         "maxfu"       200 ,
         "abstol"    1.e-8 ,
         "reltol"    1.e-6 ];
< gof,dens,mesh > = kde(data,optn);
print "GOF=",gof;
print "Dens=",1.e4 * dens[1:20,1:20];
print "Mesh=",mesh[1:20,];
```

```
   Kernel Density Estimation of 1000 Points in 2 Dimensions

 Dim=1: Minimum=  -5.06934 Maximum=   8.28094 Range=   13.3503
 Dim=2: Minimum=   -4.9753 Maximum=   5.95415 Range=   10.9295

Zero 0.00111961 found with F=4.74881e-017 after 11 function calls.
                Bandwidth= [0.386585 0.446335]
                        Computer Time=5


GOF=
           |         1
----------------------
   Failure |   0.00000
      Time |    5.0000
  Bandw[1] |   0.38658
  Bandw[2] |   0.44634
  GridPnts |    256.00
    Fcalls |    11.000
     Fzero |    5e-017
    unused |   0.00000
    unused |   0.00000
```

```
  unused |   0.00000
```

5. Two-dimensional KDE: Gaussian Mixture with distant modes: N=300

```
nr = 100;
dat1 = rand(nr,1,'g',"nor2");
dat2 = .25 * rand(nr,1,'g',"nor2");
dat3 = rand(nr,1,'g',"nor2") + 18.;
dat4 = rand(nr,1,'g',"nor2");
dat5 = rand(nr,1,'g',"nor2") + 15.;
dat6 = .5 * rand(nr,1,'g',"nor2") - 18.;
data = (dat1 -> dat2) |> (dat3 -> dat4) |> (dat5 -> dat6);
print "nr=",nrow(data), " nc=",ncol(data);


ng = 1 << 8;
optn = [ "print"         2 ,
         "ngrid"        ng ,
         "maxfu"       200 ,
         "abstol"    1.e-8 ,
         "reltol"    1.e-6 ];
< gof,dens,mesh > = kde(data,optn);
print "GOF=",gof;
print "Dens=",1.e4 * dens[1:20,1:20];
print "Mesh=",mesh[1:20,];


    Kernel Density Estimation of 300 Points in 2 Dimensions

 Dim=1: Minimum=  -9.59144 Maximum=   26.2533 Range=   35.8447
 Dim=2: Minimum=  -24.9865 Maximum=   8.16401 Range=   33.1505

Zero 6.68993e-005 found with F=-5.83843e-017 after 11 function
                           calls.
               Bandwidth= [0.418039 0.221068]
                       Computer Time=5


GOF=
           |         1
----------------------
   Failure |   0.00000
      Time |    5.0000
  Bandw[1] |   0.41804
  Bandw[2] |   0.22107
  GridPnts |    256.00
    Fcalls |    11.000
```

```
    Fzero |   -6e-017
   unused |   0.00000
   unused |   0.00000
   unused |   0.00000
```

6. Two-dimensional KDE: Sinoidale Density: N=1000

```
nr = 1000;
dat1 = rand(nr,1,'g',"dmet"); /* Mersenne-Twister */
dat2 = rand(nr,1,'g',"nor2")/3. + sin(dat1);
data = dat1 -> dat2;
print "nr=",nrow(data), " nc=",ncol(data);


ng = 1 << 8;
optn = [ "print"         2 ,
         "ngrid"        ng ,
         "maxfu"       200 ,
         "abstol"    1.e-8 ,
         "reltol"    1.e-6 ];
< gof,dens,mesh > = kde(data,optn);
print "GOF=",gof;
print "Dens=",1.e4 * dens[1:20,1:20];
print "Mesh=",mesh[1:20,];
```

```
    Kernel Density Estimation of 1000 Points in 2 Dimensions

  Dim=1: Minimum= -0.249799 Maximum=   1.24968 Range=   1.49948
  Dim=2: Minimum=  -1.49531 Maximum=   2.46723 Range=   3.96253

Zero 0.00128787 found with F=-6.57027e-017 after 11 function calls.
                Bandwidth= [0.0503937 0.153184]
                        Computer Time=5


GOF=
           |         1
----------------------
   Failure |   0.00000
      Time |    5.0000
  Bandw[1] |     0.050
  Bandw[2] |   0.15318
  GridPnts |    256.00
    Fcalls |    11.000
     Fzero |   -7e-017
    unused |   0.00000
```

```
unused |   0.00000
unused |   0.00000
```

5.3 The `mvsvm()` Function

< gof,alpha,beta,wgts,resi > = mvsvm(xtrn,ytrn<,optn<,kfun>>)

Purpose: The `mvsvm()` function can be used for estimating a sparse weight matrix of multivariate SVM estimation. The implementation is based on a Matlab program by A. Thayananthan (2005). The model is:

$$\min_{W} \parallel \mathbf{Y} - \mathbf{K}(\mathbf{X})\mathbf{W} \parallel^2$$

where $\mathbf{X}$ and $\mathbf{Y}$ are $N \times p$ resp. $N \times m$ data matrices, with $2 \leq m < N$, and $\mathbf{K}$ is either an $N \times N$ or an $N \times N + 1$ (with an additional column of 1's for an intercept weight) kernel matrix. The estimated weight matrix is either $N \times m$ or $N + 1 \times m$, depending on the size of the kernel matrix.

Input: **xtrn** $N \times p$ training data set with set of predictors

ytrn $N \times m$ training data with $m \geq 2$ responses

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

kfun optional string argument specifying a user defined function for kernel evaluation. The function definition must be of the form: `function kernel(vi,vj)` where the input arguments `vi` and `vj` are observation vectors of the data and the function must be a real scalar. Note, that Mercer's condition must be satisfied by the functions kernel definition, i.e. the resulting matrices $\mathbf{Q}$ of the QP must be positive (semi)definite.

Content of the Options Matrix :

Option	Second Column	Meaning
"c"	real	regularization parameter C
"kern"	string	link function, default is linear
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function (def. scaling $[0, 1]$)
	"rbf2"	mod. Gaussian radial basis function (def. scaling $[0, 1]$)
	"rbfcs"	mod. Gaussian radial basis function (def. scaling $[0, 1]$)
	"erbf"	Exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function (def. scaling $[-\pi/2, \pi/2]$)
	"spli"	spline function (def. scaling $[0, 1]$)
	"anov"	anova function
	"curv"	curvspline function (def. scaling $[0, 1]$)
	"bspl"	Bspline function (def. scaling $[0, 1]$)
	"anob"	anova spline function (def. scaling $[0, 1]$)
	"ano1"	modified anova function (def. scaling $[0, 1]$)
	"ano2"	modified anova function (def. scaling $[0, 1]$)
	"ano3"	modified anova function (def. scaling $[0, 1]$)
"kfp1"	real	first parameter of kernel function
"kfp2"	real	second parameter of kernel function
"kfp3"	real	third parameter of kernel function
"maxit"	int	max. number iterations of optimization
"nobias"		no intercept is added (default is otherwise)
"nopr"		no printed output in ...txt
"print"	int	amount of general printed output (default is 0)

Output: **gof** column vector with some scalar results:

1. failure
2. total computation time
3. number of selected variables
4. total sum of squares error

alpha $N + 1$ or N vector

beta m vector

wgts $N + 1 \times m$ or $N \times m$ matrix of scoring weights $\mathbf{W}$

resi $N \times m$ matriox of residuals $\mathbf{Y} - \mathbf{K} * \mathbf{W}$

Restrictions:

1. The input arguments $\mathbf{X}$ and $\mathbf{Y}$ must not contain any missing values, strings or complex data.
2. The optional input argument `kfun()` must be a typical kernel function resulting in a positive definite matrix $\mathbf{K(X)}$.

Relationships: svm(), sir(), varsel()smsvm()

Examples: 1. Data from the internet:

```
print "Generate Data: Use Gauss Kernel RBF";
xv = 10. * [ -1. : 0.02 : 1. ]‘; print "XV=",xv;

 options NOECHO;
#include "..\\tdata\\mvsvm_t.dat"
 options ECHO;

nr = nrow(yt); ny = ncol(yt);
print "Nr=,ny=",nr,ny;


print "My RBF: eta= -.5 / pk1^2 , Matlab code Gauss: eta= -1 / pk1^2";
width = 3. * sqrt(.5);
optn = [ "print"      3 ,
         "maxit"     10 ,
         "kern"    "rbf" ,
         "kfp1"   width ]; /* print optn; */
< gof,alfa,beta,wgts,resi > = mvsvm(xv,yt,optn);
```

Iter	Select	TotalSSE	Fmax	RMSE(Y1)	RMSE(Y2)
1	1	48.32758725	67800.52453	3.588081425	5.954263929
2	51	38.60504849	7536.876632	1.777424243	5.953638514
3	13	9.702589392	69.24220252	1.714728696	2.600441288
4	91	4.688247720	60.51018575	1.009470590	1.915519994
5	40	3.589827241	10.44012166	0.963878103	1.631185533
6	2	2.678441275	17.13222016	0.951298243	1.331718036
7	102	2.331049160	6.183055797	0.949824191	1.195359011
8	74	1.974893808	10.98694162	0.940848205	1.043886231
9	102	1.917453770	6.850161928	0.940601630	1.016229474
10	13	1.916082807	2.920740357	0.941140426	1.015055420

```
Number of Selected Observations: 8
Total Error Sum of Squares for 1.91608
Sparsity of Weight Matrix:    92.1569 %


                Goodness of Fit
                ***************

            Dense Column Vector (nrow=10)

  C |     Failure  TotalTime  ObsSelect  SSQ_Error     unused
```

```
              0          0  8.0000000  1.9160828          0

C |      unused     unused     unused     unused     unused
              0          0          0          0          0
```

Beta
****

Dense Matrix (2 by 2)

```
                    |       RESP1      RESP2
                  --------------------------
               Beta |   105.41626  90.309926
            SSQerror |   0.9411404  1.0150554
```

Alfa
****

Sparse Row Vector (ncol=102)

```
R |   Intercept      PRED1     PRED12     PRED39     PRED50
      0.8540182  6.0884528  8.9297176  13.778294  2.4251314

R |      PRED73     PRED90    PRED101
      9.2812462  1.6221995  2.3671207
```

Weights
****

Sparse Matrix (102 by 2)

```
          Intercept |       RESP1      RESP2
                    -0.3581995  1.1764712

              PRED1 |       RESP1      RESP2
                     0.0781061 -0.4808312

             PRED12 |       RESP1      RESP2
                     0.4003122 -0.1994516

             PRED39 |       RESP1      RESP2
                    -0.2136243 -0.0579393

             PRED50 |       RESP1      RESP2
```

```
                                1.4643996  0.3160864

                     PRED73 |       RESP1      RESP2
                               -0.1593335  0.6282074

                     PRED90 |       RESP1      RESP2
                                0.5867980  0.4280802

                    PRED101 |       RESP1      RESP2
                               -0.1085857  0.9346161


                      Time for Optimization: 0
```

2. Almost the same results with generated noise:

```
print "Generate Data: Use Gauss Kernel RBF";
nr = 101; ny = 2;
xv = 10. * [ -1. : 0.02 : 1. ]‘; print "XV=",xv;
y1 = sin(abs(xv)) ./ abs(xv);
y2 = .1 * xv + 1.5;
yt = y1 -> y2; /* print "YT=", yt; */


print "Noise=.1 and width=kfp1=3.";
nois = cons(nr,ny,.1);
nor1 = rand(nr,ny,’g’,"norm");
yt += nor1 .* nois; print "YT=", yt;


print "My RBF: eta= -.5 / pk1^2 , Matlab code Gauss: eta= -1 / pk1^2";
width = 3. * sqrt(.5);
optn = [ "print"      4 ,
         "maxit"     10 ,
         "kern"   "rbf" ,
         "kfp1"   width ]; /* print optn; */
< gof,alfa,beta,wgts,resi > = mvsvm(xv,yt,optn);


 Iter Select     TotalSSE        Fmax     RMSE(Y1)     RMSE(Y2)
    1      1  49.98086309  67675.25048  3.864225949  5.920187573
    2     52  38.97936680  7352.238573  1.985277616  5.919293842
    3     90  11.96052270  68.26522220  1.890436918  2.895992224
    4     12  5.943207460  62.38882339  1.011411099  2.218164793
    5     40  4.114215768  12.90335239  1.005925527  1.761343124
    6    102  3.275251428  15.78186726  0.959170071  1.534680489
    7     71  2.702273131  11.77751851  0.954071102  1.338664059
```

```
    8      2  2.334088435  9.064931101  0.952445114  1.194544574
    9    102  2.307990276  3.370089932  0.952412792  1.183596278
   10     40  2.316661539  3.173546326  0.957042145  1.183525189


          Number of Selected Observations: 7
        Total Error Sum of Squares for 2.31666
        Sparsity of Weight Matrix:    93.1373 %
```

3. Useer specified kernel function:

```
print "Generate Data: Use Gauss Kernel RBF";
xv = 10. * [ -1. : 0.02 : 1. ]‘; print "XV=",xv;

 options NOECHO;
#include "..\\tdata\\mvsvm_t.dat"
 options ECHO;


/* radial basis kernel function */
function krbf(vi,vj) {
  w = vi - vj; kp1 = 3.;
  fac = -1. / (kp1 * kp1);
  t = exp(fac * w * w‘);
  return(t);
}


print "Niter=20";
optn = [ "print"      3 ,
         "maxit"     20 ];
< gof,alfa,beta,wgts,resi > = mvsvm(xv,yt,optn,krbf);
```

Even ten more iterations don't improve much the fit:

```
 Iter Select     TotalSSE         Fmax     RMSE(Y1)     RMSE(Y2)
    1      1  48.32758725  67800.52453  3.588081425  5.954263929
    2     51  38.60504849  7536.876632  1.777424243  5.953638514
    3     13  9.702589392  69.24220252  1.714728696  2.600441288
    4     91  4.688247720  60.51018575  1.009470590  1.915519994
    5     40  3.589827241  10.44012166  0.963878103  1.631185533
    6      2  2.678441275  17.13222016  0.951298243  1.331718036
    7    102  2.331049160  6.183055797  0.949824191  1.195359011
    8     74  1.974893808  10.98694162  0.940848205  1.043886231
    9    102  1.917453770  6.850161928  0.940601630  1.016229474
```

10	13	1.916082807	2.920740357	0.941140426	1.015055420
11	78	1.863793787	2.815554657	0.938155580	0.991795289
12	51	1.862443519	1.546735197	0.937614196	0.991626512
13	91	1.860056099	1.188107693	0.936894973	0.991102371
14	85	1.851864278	0.525015239	0.933802811	0.989887159
15	91	1.849519800	0.646321250	0.933096298	0.989369040
16	86	1.848617261	0.239858372	0.932394351	0.989574675
17	40	1.849642206	0.249972997	0.932783338	0.989725947
18	35	1.844927010	0.320993954	0.929979394	0.989982493
19	40	1.848372035	0.364067546	0.929974966	0.991725062
20	102	1.848168388	0.210290218	0.929964152	0.991632525

Number of Selected Observations: 12
Total Error Sum of Squares for 1.84817
Sparsity of Weight Matrix: 88.2353 %

Beta
****

Dense Matrix (2 by 2)

	RESP1	RESP2
Beta	107.72977	94.731775
SSQerror	0.9299642	0.9916325

Alfa
****

Sparse Row Vector (ncol=102)

R	Intercept	PRED1	PRED12	PRED34	PRED39
	0.8540182	6.0884528	8.9297176	28.209071	139.90127

R	PRED50	PRED73	PRED77	PRED84	PRED85
	0.8590816	9.2812462	5.7122970	9.0741584	13.024997

R	PRED90	PRED101
	56.313924	1.5213575

Weights
****

Sparse Matrix (102 by 2)

Intercept	RESP1	RESP2
	-0.2287873	1.2062517
PRED1	RESP1	RESP2
	-0.0138703	-0.5112543
PRED12	RESP1	RESP2
	0.3480032	-0.2018783
PRED34	RESP1	RESP2
	-0.2380828	-0.0604445
PRED39	RESP1	RESP2
	-0.0433380	-0.0351027
PRED50	RESP1	RESP2
	1.3308852	0.3270169
PRED73	RESP1	RESP2
	-0.2536792	0.2580511
PRED77	RESP1	RESP2
	-0.1666725	0.3827200
PRED84	RESP1	RESP2
	0.2828395	0.1257793
PRED85	RESP1	RESP2
	0.2192362	0.0714420
PRED90	RESP1	RESP2
	0.0501825	0.0020802
PRED101	RESP1	RESP2
	-0.0156768	1.1302865

Time for Optimization: 0

5.4 The `permcomb()` Function

next = permcomb(sopt,nfol,n,k<,istart>)

Purpose: The `permcomb()` function can be used for generating a specific number of permutations or combinations with or without replacement. The implementation is based on a Matlab program by Mat Fig, Idaho National Lab.

Input: **sopt** the following string options specify the kind of combination or permutation:

- **cmbwo** combinations without replacement
- **cmbwr** combinations with replacement
- **prmwo** permutations without replacement
- **prmwr** permutations with replacement

nfol an integer specifying the number of required combinations or permutations, a missing value specifies all the remaining

n an integer specifying the number of items (length of set), cannot be missing

k an integer specifying the sample number, cannot be missing

istart the number at which the returned combinations or permutations must start, a missing value specifies the same as 0

Output: The only return argument `next` is either a k row vector or an $N \times k$ matrix containing one combination or permutation in each row.

Restrictions: The value of k cannot be larger than n.

Relationships: permute(), rperm()cperm(), combn()combn2()

Examples: 1. Example 1: All combinations without replications:

```
n = 5; k = 3; istrt = 0;
nxtcom = permcomb("cmbwo",.,n,k,istrt);
print "Combinations without Replications:",nxtcom;
```

```
[note] There are 10 combinations w/o rep for n=5 and k=3.
```

```
   |   1   2   3
---------------
 1 |   1   2   3
 2 |   1   2   4
 3 |   1   2   5
 4 |   1   3   4
```

```
 5 |   1   3   5
 6 |   1   4   5
 7 |   2   3   4
 8 |   2   3   5
 9 |   2   4   5
10 |   3   4   5
```

2. Example 2: All combinations with replications:

```
n = 5; k = 3; istrt = 0;
nxtcom = permcomb("cmbwr",.,n,k,istrt);
print "Combinations with Replications:",nxtcom;
```

```
[note] There are 35 combinations with rep for n=5 and k=3.
```

```
   |   1   2   3
----------------
 1 |   1   1   1
 2 |   1   1   2
 3 |   1   1   3
 4 |   1   1   4
 5 |   1   1   5
 6 |   1   2   2
 7 |   1   2   3
 8 |   1   2   4
 9 |   1   2   5
10 |   1   3   3
.................
33 |   4   4   5
34 |   4   5   5
35 |   5   5   5
```

3. Example 3: All permutations without replications:

```
n = 5; k = 3; istrt = 0;
nxtcom = permcomb("prmwo",.,n,k,istrt);
print "Permutations without Replications:",nxtcom;
```

```
[note] There are 60 permutations w/o rep for n=5 and k=3.
```

```
   |   1   2   3
----------------
 1 |   1   2   3
```

```
 2 |   2   1   3
 3 |   2   3   1
 4 |   1   3   2
 5 |   3   1   2
 6 |   3   2   1
 7 |   1   2   4
 8 |   2   1   4
 9 |   2   4   1
10 |   1   4   2
.................
58 |   3   5   4
59 |   5   3   4
60 |   5   4   3
```

4. Example 4: All permutations with replications:

```
n = 5; k = 3; istrt = 0;
nxtcom = permcomb("prmwr",.,n,k,istrt);
print "Permutations with Replications:",nxtcom;
```

```
[note] There are 125 permutations with rep for n=5 and k=3.
```

```
    |   1   2   3
-----------------
  1 |   1   1   1
  2 |   1   1   2
  3 |   1   1   3
  4 |   1   1   4
  5 |   1   1   5
  6 |   1   2   1
  7 |   1   2   2
  8 |   1   2   3
  9 |   1   2   4
 10 |   1   2   5
.................
123 |   5   5   3
124 |   5   5   4
125 |   5   5   5
```

5. Example 5: Some combinations without replications:

```
n = 3; k = 2; istrt = 2;
nxtcom = permcomb("cmbwo",.,n,k,istrt);
print "Combinations without Replications:",nxtcom;
```

```
[note] There are 3 combinations w/o rep for n=3 and k=2.
[note] There are only 1 combinations left that are returned.


   |   1   2
------------
 1 |   2   3
```

6. Example 6: Some combinations with replications:

```
n = 3; k = 2; istrt = 2;
nxtcom = permcomb("cmbwr",.,n,k,istrt);
print "Combinations with Replications:",nxtcom;
```

```
[note] There are 6 combinations with rep for n=3 and k=2.
[note] There are only 4 combinations left that are returned.


   |   1   2
------------
 1 |   1   3
 2 |   2   2
 3 |   2   3
 4 |   3   3
```

7. Example 7: Some permutations without replications:

```
n = 3; k = 2; istrt = 2;
nxtcom = permcomb("prmwo",.,n,k,istrt);
print "Permutations without Replications:",nxtcom;
```

```
[note] There are 6 permutations w/o rep for n=3 and k=2.
[note] There are only 4 combinations left that are returned.


   |   1   2
------------
 1 |   1   3
 2 |   3   1
 3 |   2   3
 4 |   3   2
```

8. Example 8: Some Permutations with replications:

```
  n = 3; k = 2; istrt = 2;
  nxtcom = permcomb("prmwr",.,n,k,istrt);
  print "Permutations with Replications:",nxtcom;
```

```
 [note] There are 9 permutations with rep for n=3 and k=2.
 [note] There are only 7 combinations left that are returned.
```

	1	2
1	1	3
2	2	1
3	2	2
4	2	3
5	3	1
6	3	2
7	3	3

5.5 The `prefname()` Function

names = prefname(pref,nind<,optn>)

Purpose: The `prefname()` function can be used to generate names with a prefix name and numerical suffix.

Input: **pref** must be a string scalar for the prefix of the set of names.

nind must be a positive integer scalar or a K vector of positive integers.

Output: Assuming the vector `nind` has K entries then the return object `names` is a string vector containing $nind[1] * nind[2] * ... * nind[K]$ names with K numerical suffixes ranging from 1 to $nind[k]$, $k = 1, \ldots, K$.

Restrictions:

1. Neither of the first two input arguments may have missing values.
2. The first input argument must be string and the second input argument must have positive integers.

Relationships:

Examples: 1. Only one suffix: $K = 1$:

```
inds = 10; pref = "var";
nam1 = prefname(pref,inds);
print "nam1=", nam1;
```

```
nam1=
   |       1
-----------
 1 |   var01
 2 |   var02
 3 |   var03
 4 |   var04
 5 |   var05
 6 |   var06
 7 |   var07
 8 |   var08
 9 |   var09
10 |   var10
```

Practically the same was possible before:

```
nams = [ "var1" : "var10" ] ;
print nams;
```

```
   |       1       2       3       4       5
---------------------------------------------
 1 |    var1    var2    var3    var4    var5

   |       6       7       8       9      10
---------------------------------------------
 1 |    var6    var7    var8    var9   var10
```

```
nams = [ "var_1" : "var_10" ] ;
print nams;
```

```
   |        1        2        3        4        5
-----------------------------------------------
 1 |    var_1    var_2    var_3    var_4    var_5

   |        6        7        8        9       10
-----------------------------------------------
 1 |    var_6    var_7    var_8    var_9   var_10
```

2. Two numerical suffixes $K = 2$:

```
inds = [ 10 10 ]; pref = "var";
nam2 = prefname(pref,inds);
print "nam2=", nam2;
```

```
    |          1
----------------
  1 |   var01_01
  2 |   var01_02
  3 |   var01_03
  4 |   var01_04
  5 |   var01_05
  6 |   var01_06
  7 |   var01_07
  8 |   var01_08
  9 |   var01_09
 10 |   var01_10
................
 91 |   var10_01
 92 |   var10_02
 93 |   var10_03
 94 |   var10_04
 95 |   var10_05
 96 |   var10_06
```

```
 97 |   var10_07
 98 |   var10_08
 99 |   var10_09
100 |   var10_10
```

3. Three numerical suffixes $K = 3$:

```
inds = [ 3#10 ]; pref = "var";
nam3 = prefname(pref,inds);
print "nam3=", nam3;
```

```
     |             1
-------------------
   1 |   var01_01_01
   2 |   var01_01_02
   3 |   var01_01_03
   4 |   var01_01_04
   5 |   var01_01_05
   6 |   var01_01_06
   7 |   var01_01_07
   8 |   var01_01_08
   9 |   var01_01_09
  10 |   var01_01_10
.....................
 991 |   var10_10_01
 992 |   var10_10_02
 993 |   var10_10_03
 994 |   var10_10_04
 995 |   var10_10_05
 996 |   var10_10_06
 997 |   var10_10_07
 998 |   var10_10_08
 999 |   var10_10_09
1000 |   var10_10_10
```

4. Four numerical suffixes $K = 4$:

```
inds = [ 4#10 ]; pref = "var";
nam4 = prefname(pref,inds);
print "nam4=", nam4[1:200];
```

```
    |                1
----------------------
  1 |   var01_01_01_01
```

```
  2 |   var01_01_01_02
  3 |   var01_01_01_03
  4 |   var01_01_01_04
  5 |   var01_01_01_05
  6 |   var01_01_01_06
  7 |   var01_01_01_07
  8 |   var01_01_01_08
  9 |   var01_01_01_09
 10 |   var01_01_01_10
....................
```

5.6 The `rvm()` Function

< gof,beta,resi > = rvm(xtrn,ytrn<,optn<,kfun>>)

Purpose: The `rvm()` function can be used for binary classification and regression using the algorithm of the relevance vector machine by M. E. Tipping(2001), R. Herbrich (2002). For classification, the data `ytrn` must be $(0, 1)$ binary, for regression they can be any real data. For other than *rbf* kernel functions this algorithm seems to have some problems.

Input: **xtrn** $N \times p$ training data set with set of predictors

ytrn $N \times m$ training data with $m \geq 2$ responses

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

kfun optional string argument specifying a user defined function for kernel evaluation. The function definition must be of the form: `function kernel(vi,vj)` where the input arguments `vi` and `vj` are observation vectors of the data and the function must be a real scalar. Note, that Mercer's condition must be satisfied by the functions kernel definition, i.e. the resulting matrices **Q** of the QP must be positive (semi)definite.

Content of the Options Matrix :

Option	Second Column	Meaning
"clas"		perform RVM binary classification
"c"	real	regularization parameter C
"fconv"	real	convergence criterion, def=1.e-5 for class, =1.e-6 for regression
"kern"	string	link function, default is linear
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function (def. scaling $[0, 1]$)
	"rbf2"	mod. Gaussian radial basis function (def. scaling $[0, 1]$)
	"rbfcs"	mod. Gaussian radial basis function (def. scaling $[0, 1]$)
	"erbf"	Exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function (def. scaling $[-\pi/2, \pi/2]$)
	"spli"	spline function (def. scaling $[0, 1]$)
	"anov"	anova function
	"curv"	curvspline function (def. scaling $[0, 1]$)
	"bspl"	Bspline function (def. scaling $[0, 1]$)
	"anob"	anova spline function (def. scaling $[0, 1]$)
	"ano1"	modified anova function (def. scaling $[0, 1]$)
	"ano2"	modified anova function (def. scaling $[0, 1]$)
	"ano3"	modified anova function (def. scaling $[0, 1]$)
"kfp1"	real	first parameter of kernel function
"kfp2"	real	second parameter of kernel function
"kfp3"	real	third parameter of kernel function
"maxit"	int	max. number iterations of optimization
"nobias"		no intercept is added (default is otherwise)
"nopr"		no printed output in ...txt
"print"	int	amount of general printed output (default is 0)
"regr"		perform RVM regression
"xrel"	real	threshold for zeroing weights, def=1.e-5 for class, =1.e-3 for regression

Output: gof column vector with some scalar results:

1. failure
2. total computation time
3. number of selected variables
4. number of kernel evaluations
5. value of termination criterion
6. total sum of squares error
7. for classification: value of Likelihood , for regression: value of σ^2
8. for classification only: accuracy value for cutoff=0.5

beta (usually sparse) m vector of weights

resi $N \times 2$ matrix of predicted values and residuals $\mathbf{Y} - \mathbf{K} * \beta$

Restrictions: 1. The input arguments **X** and **Y** must not contain any missing values, strings or complex data.

2. The optional input argument `kfun()` must be a typical kernel function resulting in a positive definite matrix $\mathbf{K(X)}$.

3. For other than *rbf* kernel functions this algorithm seems to have some problems.

Relationships: svm(), mvsvm(), smsvm()

Examples: 1. Classification : First N=60 observations of Heart data from Statlog Collection

```
data = rspfile("..\\tdata\\heart_60.dat");
```

First we show the in- and output of `svm()`:

```
modl = "1 = 2:14";
class = 1;
optn = [ "print"           2,
         "meth"       "redu",
         "ransam"           ,
         "samsiz"        30,
         "kern"       "rbf2",
         "c"                1.,
         "kfp1"  .076923076 ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

```
                    *****************
                    Model Information
                    *****************

Number Valid Observations   60
Response Variable          Y[1]
N Independent Variables     13
Support Vector C Classification
Model Without Linear Constraint
Estimation Method          RED
Kernel Function           RBF2
Gamma of RBF       0.076923076
Predictor Scaling [0,1]
Bias Uncorrected Predicted Values
```

```
                    ************************
                    Class Level Information
                    ************************

Class  Level    Value

Y[ 1]      2        1      -1
   Traindata stored incore: Mem need: 0.006 Mem spec: 2 Mb


            ***************************************
            Number of Observations for Class Levels
            ***************************************

            Variable       Value    Nobs  Proportion
            Y[ 1]              1      27   45.000000
                              -1      33   55.000000


         ---Start Training Cycle: Technique=  RSVM---

                      SSVM Iteration History

  iter        crit        cdif       gnorm      desc      step
     1  30.0000000           0  55.6060788 -30.49212         0
     2  14.7035614  15.2964386  3.14944223 -0.049947  1.000000
     3  14.6772238  0.02633758  0.41416886 -8.1e-004  1.000000
     4  14.6768208  4.030e-004  4.910e-014 -1.5e-028  1.000000


                ******************************
                Evaluation of Training Data Fit
                ******************************

   Index                                 Value       StdErr
   Absolute Classification Error            10            .
   Classification Accuracy           83.33333333          .
   Percent Concordant Pairs          67.34006734          .
   Percent Discordant Pairs          2.356902357          .
   Percent Tied Pairs                30.30303030          .
   c                                 0.824915825          .
   Goodman-Kruskal Gamma             0.932367150  0.048879486
   Kendall Tau_a                     0.327118644          .
   Kendall Tau_b                     0.664928475  0.095430834
   Stuart Tau_c                      0.643333333  0.098511139
   Somers D C|R                      0.649831650  0.098066914
   Somers D R|C                      0.680376028  0.095274482
```

```
                    Classification Table
                    --------------------

                      |    Predicted
               Observed |        1      -1
               ---------|-----------------
                     1 |       20       7
                    -1 |        3      30


 Regularization Parameter C . . . . . . . . . . . . . . . . . . . 1
 Kernel Function. . . . . . . . . . . . . . . . . . .          RBF2
 Kernel Parameter [1] . . . . . . . . . . . . . . . . . 0.0769231
 Norm of Longest Vector . . . . . . . . . . . . . . . . 1.41421
 Number Misclassifactions (Training Data) . . . . . . . . . .  10
 Number Fast Runs through TrainData . . . . . . . . . . . . . 3
 Number Slow Runs through TrainData . . . . . . . . . . . . . 6
 Total Number of Kernel Calls . . . . . . . . . . . . . .  5641
 Time for Optimization. . . . . . . . . . . . . . . . . . . . 0
 Total Processing Time. . . . . . . . . . . . . . . . . . . . 0
 Optimization Criterion . . . . . . . . . . . . . . . . 14.6768
 Infinity Norm of Gradient. . . . . . . . . . . .  4.91013e-014
 Geometric Margin . . . . . . . . . . 0.772191 (|w|^2= 6.70827)
 Number Support Vectors . . . . . . . . . . . . . 30 ( 50.00 %)
 Number Support Vectors on Margin . . . . . . . . . . . . . . 0
 Bias . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 0
 Radius of Sphere Around SV . . . . . . . . . . . . . . 1.07055
 Estimated VCdim of Classifier. . . . . . . . . . . . . 8.68821
 KT Threshold . . . . . . . . . . . . . . . . . . . . . . . . . 0


              Total Number of Kernel Calls: 5641
                   Time for Optimization: 0
                   Total Processing Time: 0
```

It follows the in- and output of `rvm()`:

```
xtrn = data[,2:14]; ytrn = data[,1];
ytrn = ytrn .== -1;


optn = [ "print"          3 ,
         "clas"             ,
         "xrel"       1.e-5 ,
         "fconv"      1.e-5 ,
```

```
         "maxit"       1000 ,
         "kern"      "rbf2" ,
         "kfp1"  .076923076 ];
< gof,beta,resi > = rvm(xtrn,ytrn,optn);
```

```
          ****************************************
          Relevance Vector Machine for Classification
          ****************************************

       Iter  Select    Criterion    SSquError   Likelihood
          1      61  0.141194632  7.268355831 -26.70366303
          2      61  0.202366147  6.780147086 -24.62901949
          3      60  0.455907615  6.502034595 -23.61470248
          4      55  0.865167955  6.279342082 -22.61905504
          5      54  1.068455496  6.173294844 -21.79522780
          6      49  0.837400464  6.169882251 -21.33050129
          7      40  0.488518581  6.188950474 -21.12400030
          8      35  0.250088006  6.200670946 -21.03494148
          9      32  0.129754707  6.203965163 -20.99252080
         10      29  0.075629528  6.202576375 -20.96824793
        ..................................................
        208       5  1.0694e-005  6.187476307 -20.91302366
        209       5  1.0286e-005  6.187475899 -20.91302333
        210       5  9.8939e-006  6.187475506 -20.91302300
```

```
                 *******************************
                 Evaluation of Training Data Fit
                 *******************************

    Index                                  Value       StdErr
    Absolute Classification Error              7            .
    Classification Accuracy            88.33333333          .
    Percent Concordant Pairs           77.44107744          .
    Percent Discordant Pairs           1.346801347          .
    Percent Tied Pairs                 21.21212121          .
    c                                  0.880471380          .
    Goodman-Kruskal Gamma              0.965811966  0.027302710
    Kendall Tau_a                      0.383050847          .
    Kendall Tau_b                      0.763949603  0.083703948
    Stuart Tau_c                       0.753333333  0.086126522
    Somers D C|R                       0.760942761  0.084725834
    Somers D R|C                       0.766968326  0.083540004
```

Classification Table

```
--------------------

          |    Predicted
 Observed |    Event   NoEvt
 ---------|-----------------
    Event |       23       4
    NoEvt |        3      30


Total Number of Kernel Calls: 3600
Number of Selected Observations: 5
Termination Criterion: 9.89392e-006
Total Error Sum of Squares: 6.18748
      Log Likelihood: -20.913
 Sparsity of Weights:    91.8033 %


                Goodness of Fit
                ***************

            Dense Column Vector (nrow=10)

C |     Failure  TotalTime  ObsSelect   N_Kernel  Term_Crit
              0          0  5.0000000  3600.0000  9.89e-006

C |   SSQ_Error Likelihood   Accuracy     unused     unused
      6.1874755 -20.913023  88.333333          0          0


                          Beta
                          ****
              Sparse Row Vector (ncol=61)

R |       PRED1     PRED10     PRED24     PRED28     PRED50
     -14.581109 -0.4626416  1.1011166  12.646665 -2.09e-004


              Time for Optimization: 0
```

2. Regression: Boston Housing Data: nobs=506, nvar=14

```
fid = fopen("..\\tdata\\housing.dat","r");
form = "%g %g %g %g %g %g %g %g %g %g %g %g %g %g";
hous = fscanf(fid,form,506,14);
vnam = [ "crim" "zn" "indus" "chas" "nox" "rm" "age"
         "dis" "rad" "tax" "ptrat" "b" "lstat" "medv" ];
hous = cname(hous,vnam);
```

```
/* print hous[1:10,; */
```

First we show the in- and output of `svm()`:

```
modl = "14 = 1:13";
optn = [ "print"           2 ,
         "pplan"             ,
         "meth"        "fqp" ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(hous,modl,optn);
```

```
                      *****************
                      Model Information
                      *****************

                Number Valid Observations  506
                Response Variable         Y[14]
                N Independent Variables     13
                Support Vector L2 Regression
                Model with 1 Linear Constraint
                Estimation Method          FQP
                Optimization Method     QPNUSP
                Kernel Function         Linear
                Use Unscaled Predictor
                Bias Corrected Predicted Values


          Memory needed for Kernel matrix: 0.97863 Mb
    Traindata stored incore: Mem need: 0.05 Mem spec: 2 Mb


          ---Start Training Cycle: Technique=  FQP---
 Regularization Parameter C . . . . . .  3.08629e-006
 Kernel Function. . . . . .         Linear
 Norm of Longest Vector . . . . . . 819.609
 Mean Squared Error (MSE, Training Data). . . . . .  63.365
 Mean Absolute Error (MAE, Training Data) . . . . . 5.68394
 Average Loss (Training Data) . . . . . 5.68394
 Average Positive Loss (Training Data). . . . . 4.39767 (N=327)
 Average Negative Loss (Training Data). . . . . 8.03373 (N=179)
 Number Fast Runs through TrainData . . . . . 6
 Number Slow Runs through TrainData . . . . . 8
 Total Number of Kernel Calls . . . . . .  258567
 Time for Optimization. . . . . 7
 Total Processing Time. . . . . 7
 Optimization Criterion . . . . . -0.00866023
```

```
Infinity Norm of Gradient. . . . . 0.0144648
Geometric Margin . . . . . 79.7907 (|w|^2= 0.000628284)
Number Support Vectors . . . . . 506 (100.00 %)
Number Support Vectors on Margin . . . . . 504
Bias . . . . . -2.555e-014
Radius of Sphere Around SV . . . . . 819.609
Estimated VCdim of Classifier. . . . . 423.056
Linear Kernel Constant (Fit) . . . . . 27.708
Linear Kernel Constant (PCE) . . . . . .
```

Linear Separating Plane (w*x = 27.708)

Dense Row Vector (ncol=14)

R	1	2	3	4	5
	-0.0021096	0.0082738	-0.0023612	4.01e-005	-3.57e-005

R	6	7	8	9	10
	5.47e-004	-0.0121544	3.48e-004	8.64e-005	-0.0174836

R	11	12	13	14
	-9.59e-004	0.0079062	-0.0057040	27.708037

Largest 13 Plane Coefficients (Sorted)

1	10	tax	-0.017483631
2	7	age	-0.012154443
3	2	zn	0.008273824
4	12	b	0.007906184
5	13	lstat	-0.005704010
6	3	indus	-0.002361205
7	1	crim	-0.002109603
8	11	ptrat	-0.000959021
9	6	rm	0.000547174
10	8	dis	0.000347989
11	9	rad	8.6398e-005
12	4	chas	4.0122e-005
13	5	nox	-3.5694e-005

Sparsity: 13 Nonzeros 0 Zeros (C=3.08629e-006)

Total Number of Kernel Calls: 258567

```
                  Time for Optimization: 7
                  Total Processing Time: 7
```

```
xtrn = hous[,1:13]; ytrn = hous[,14];
width = sqrt(.5);
optn = [ "print"          3 ,
         "regr"             ,
         "xrel"       1.e-3 ,
         "fconv"      1.e-6 ,
         "maxit"       1000 ,
         "kern"       "rbf" ,
         "kfp1"       width ]; /* print optn; */
< gof,beta,resi > = rvm(xtrn,ytrn,optn);
print "GOF=",gof;
print "Beta=",beta;
print "Resi=",resi[1:20,];
```

It follows the in- and output of `rvm()`:

```
xtrn = hous[,1:13]; ytrn = hous[,14];
width = sqrt(.5);
optn = [ "print"          3 ,
         "regr"             ,
         "xrel"       1.e-3 ,
         "fconv"      1.e-6 ,
         "maxit"       1000 ,
         "kern"       "rbf" ,
         "kfp1"       width ]; /* print optn; */
< gof,beta,resi > = rvm(xtrn,ytrn,optn);
```

Unfortunately, RVM regression does not seem to work very well for these data:

```
            ************************************
            Relevance Vector Machine for Regression
            ************************************

        Iter  Select    Criterion    SSquError        Sigma
           1     507  11.57935947  46959.79883  0.434079611
           2     484  6.182856659  2621.910879  0.169227480
           3     454  4.393084084  85.68619516  0.076813603
           4     454  2.009692260  18.14696696  0.055604990
           5     454  1.747219098  10.37736074  0.051641551
           6     454  1.115842067  8.669978522  0.050999796
```

7	454	0.306066382	8.376211994	0.051202358
8	454	0.099877460	8.340971866	0.051273633
9	454	0.038491213	8.334736557	0.051288900
10	454	0.013822656	8.333223408	0.051293597
11	454	0.004810109	8.332799227	0.051295179
12	454	0.001650997	8.332669951	0.051295719
13	454	0.000563201	8.332628597	0.051295903
14	454	0.000191576	8.332615004	0.051295966
15	454	6.5077e-005	8.332610469	0.051295987
16	454	2.2091e-005	8.332608944	0.051295994
17	454	7.4965e-006	8.332608430	0.051295997
18	454	2.5435e-006	8.332608255	0.051295998
19	454	8.6290e-007	8.332608196	0.051295998

Total Number of Kernel Calls: 256036
Number of Selected Observations: 454
Termination Criterion: 8.62901e-007
Total Error Sum of Squares: 8.33261
Sigma2: 0.051296
Sparsity of Weights: 10.4536 %

Goodness of Fit
****

Dense Column Vector (nrow=10)

C		Failure	TotalTime	ObsSelect	N_Kernel	Term_Crit
		0	0	454.00000	256036.00	8.63e-007

C		SSQ_Error	Sigma2	unused	unused	unused
		8.3326082	0.0512960	0	0	0

Beta
****

Dense Row Vector (ncol=507)

R		Intercept	PRED1	PRED2	PRED3	PRED4
		19.687724	4.3003812	1.8854584	15.008860	13.708537

R		PRED5	PRED6	PRED7	PRED8	PRED9
		16.509170	9.0065850	3.1963082	7.4053560	-3.1716338

R		PRED10	PRED11	PRED12	PRED13	PRED14

```
          -0.7226998 -4.6767819 -0.7226998  1.9867909  0.6403863

     R |      PRED15     PRED16     PRED17     PRED18     PRED19
          -1.4532725          0  3.3972447 -2.1642813          0
.....................................................................


     R |     PRED500    PRED501    PRED502    PRED503    PRED504
          -2.1642588 -2.8699633  2.6933653  0.8561087  4.2000980

     R |     PRED505    PRED506
           2.2900948 -7.7811378

                    Time for Optimization: 2
```

5.7 The `sdcspm()` Function

< gof,klc,eklc,spm,spr,spc,wspm,wspr,wspc > = sdcspm(D<,rho<,optn>>)

Purpose: The `sdcspm()` function can be used for testing Srivastava's condition for a search design matrix **D** and to compute some measures for comparing the search performance of design matrices like

- KL: Kullback-Leibler search criterion (Talebi & Esmailzadeh, 2011a)
- EKL: Expected Kullback-Leibler search criterion (Talebi & Esmailzadeh, 2011a)
- SP: Search Probability (Ghosh & Teschmacher, 2002)
- WSP: Weighted Search Probability (Talebi & Esmailzadeh, 2011b)

These are the smallest values of the matrices **C**.

Input: Up to three input arguments can be specified:

D is an $N \times n$ design matrix **D** for $n < N$ and with entries in {-1,+1} and full column rank.

rho is either a missing value, a positive real scalar, or a k vector of positive real values; that specifies the parameter for the Kullback-Leibler and search probability measures

optn is a vector of runtime options, for content see below.

The content of the options vector:

1. amount of printed output (default: =0: no printed output)
2. number of interactions (must be integer in $\{2, 3, 4\}$, default=2)
3. parameter k for Srivastava's test, if specified as missing: the largest k is tried first and is stepwise reduced as long as Srivastava's condition is not satisfied
4. first parameter v for Gamma function (needed only for weighted search probability)
5. second parameter la for Gamma function (needed only for weighted search probability)
6. if $\neq 0$: plot graph of search probabilities for $r = 0, ..., \rho$ (if =0 or missing, do not plot)
7. $\neq 0$ output matrix **C** of Kullback-Leibler measures
8. $\neq 0$ output matrix **C** of expected Kullback-Leibler measures
9. $\neq 0$ output matrix **M** for search probability measures
10. $\neq 0$ output vector **r** for search probability measures
11. $\neq 0$ output matrix **C** of search probability measures

12. $\neq 0$ output matrix **M** for weighted search probability measures
13. $\neq 0$ output vector **r** for weighted search probability measures
14. $\neq 0$ output matrix **C** of weighted search probability measures

Output: There are up to nine output arguments:

gof a vector of scalar results, for content see below.

klc matrix **C** of Kullback-Leibler measures

eklc matrix **C** of expected Kullback-Leibler measures

spm $n2$ $timesn2$ matrix **M** of search probabilities

spr $n2$ vector r of search probabilities

spc $n2$ $timesn2$ matrix **C** of search probabilities

wspm $n2$ $timesn2$ matrix **M** of weighted search probabilities

wspr $n2$ vector r of weighted search probabilities

wspc $n2$ $timesn2$ matrix **C** of weighted search probabilities

The content of the `gof` vector for a successful run:

1. the largest value of k for which Srivastava's condition is satisfied
2. computer time in seconds
3. the Kullback-Leibler criterion for a search design
4. the extended Kullback-Leibler criterion for a search design
5. the search probability criterion for a search design
6. the weighted search probability criterion for a search design

Restrictions:
1. the $N \times n$ design matrix **D** must have only entries in {-1,+1}
2. if not missing ρ must have only positive real values

Relationships: promep(), urd1out()

Examples: The data are from the illustrative example by N. Esmailzadeh (2013) and are a design matrix **D** with four factors each at two levels:

```
dmat = [ -1  -1   1  -1 ,
          1  -1  -1  -1 ,
          1   1   1  -1 ,
         -1   1  -1   1 ,
         -1  -1  -1   1 ,
          1   1  -1  -1 ,
          1   1   1   1 ,
          1  -1   1   1 ,
         -1   1   1   1 ,
```

```
 -1   1  -1  -1 ,
  1   1  -1   1 ,
  1  -1  -1   1 ];
```

The matrix of 3 interactions is:

```
                Matrix of Interactions up to Order 3

        1      2      3      4      5      6      7      8      9
 1      1     -1      1     -1      1     -1      1     -1      1
 2     -1     -1     -1      1      1      1      1      1      1
 3      1      1     -1      1     -1     -1      1     -1     -1
 4     -1      1     -1     -1      1     -1      1     -1      1
 5      1      1     -1      1     -1     -1     -1      1      1
 6      1     -1     -1     -1     -1      1     -1     -1      1
 7      1      1      1      1      1      1      1      1      1
 8     -1      1      1     -1     -1      1     -1     -1      1
 9     -1     -1     -1      1      1      1     -1     -1     -1
10     -1      1      1     -1     -1      1      1      1     -1
11      1     -1      1     -1      1     -1     -1      1     -1
12     -1     -1      1      1     -1     -1      1     -1     -1
```

```
                Matrix of Interactions up to Order 3

                       10
                 1      1
                 2     -1
                 3     -1
                 4     -1
                 5      1
                 6      1
                 7      1
                 8     -1
                 9      1
                10      1
                11     -1
                12      1
```

1. Neither input argument 2 nor `optn[3]` is specified: Search for largest k for which augmented matrix $\mathbf{X}$ has full column rank:

```
optn = [ 5 ,   /* ipri */
         3 ,   /* nint */
```

```
          . ];  /* ktst */
 gof = sdcspm(dmat,.,optn);


          Srivastava's Condition not satisfied for k=7

Failing Combination
-------------------

  1 :         1        2        3        4        5
  6 :         7        8


Singular Values                                                    Page 3
---------------

  1 :           4          4          4          4          4
  6 :           4          4          4      2.828          2
 11 :           2  2.117e-016
        --------------- Reducing to: k=6 ---------------


          Srivastava's Condition not satisfied for k=6

Failing Combination
-------------------

  1 :         1        2        5        6        7
  6 :        10


Singular Values
---------------

  1 :           4          4          4          4          4
  6 :           4          4      3.291      2.274          2
 11 :   3.281e-016
        --------------- Reducing to: k=5 ---------------


Srivastava's Condition for a search design is satisfied for k=5.
```

`gof[2]` returns $k = 5$.

2. Checking whether or or not the design $\mathbf{D}$ is a search design: $k = 4$

```
 optn = [ 5 ,   /* ipri */
          3 ,   /* nint */
```

```
         4 ];  /* ktst */
gof = sdcspm(dmat,.,optn);

Srivastava's Condition for a search design is satisfied for k=4.
```

3. Checking whether or or not the design $\mathbf{D}$ is a search design: $k = 6$

```
optn = [ 5 ,   /* ipri */
         3 ,   /* nint */
         6 ];  /* ktst */
gof = sdcspm(dmat,.,optn);


         Srivastava's Condition not satisfied for k=6


Failing Combination
-------------------

  1 :         1        2        5        6        7
  6 :        10


Singular Values                                                    Page 10
---------------

  1 :           4          4          4          4          4
  6 :           4          4      3.291      2.274          2
 11 :   3.281e-016


       --------------- Reducing to: k=5 ---------------
Srivastava's Condition for a search design is satisfied for k=5.
```

4. Checking whether or or not the design $\mathbf{D}$ is a search design: $k = 8$
   This $k = 8$ generates an augmented matrix `X` with too many columns:

```
print "Choose ktst=2*k, k=4 as in JSS 1068";
optn = [ 5 ,   /* ipri */
         3 ,   /* nint */
         8 ];  /* ktst */
gof = sdcspm(dmat,.,optn);


[warning] file tsdcspm.inp, line 99: The number of columns (13) of
          the augmented design matrix cannot be larger than the
          number of rows (12) of the input design matrix.
```

```
Parameter k reduced to k=7.
```

5. Computing the KL, EKL, SP, WSP criterion for design D using SDC function:

```
print "KL= 42.6667, EKL = 10.6667, SP = 0.9995, WSP = 0.7726";
optn = [ 5  ,   /* ipri */
         2  ,   /* nint */
          . ,   /* ktst */
         2. ,   /* gam1 */
         5. ];  /* gam2 */
< gof,klm,eklm,spm,spr,spc,wspm,wspr,wspc >  = sdcspm(dmat,2.,optn);
```

```
     Kullback-Leibler Criterion . . . . . 42.666667
     Expected Kullback-Leibler Criterion. . . . . 10.666667
     Searching Probability. . . . . 0.9994546
     Weighted Searching Probability . . . . . 0.7725709
```

```
print "KL= 24, EKL= 10.6667, SP= 0.9928, WSP= 0.7726";
optn = [ 5  ,   /* ipri */
         3  ,   /* nint */
          . ,   /* ktst */
         2. ,   /* gam1 */
         5. ];  /* gam2 */
< gof,klm,eklm,spm,spr,spc,wspm,wspr,wspc >  = sdcspm(dmat,1.5,optn);
```

```
     Kullback-Leibler Criterion . . . . . 24.000000
     Expected Kullback-Leibler Criterion. . . . . 10.666667
     Searching Probability. . . . . 0.9928362
     Weighted Searching Probability . . . . . 0.7725709
```

6. Output of some matrices:

```
optn = [  5 ,   /* ipri */
          3 ,   /* nint */
          . ,   /* ktst */
          . ,   /* gam1 */
          . ,   /* gam2 */
          . ,   /* plot */
          . ,   /* print klc  */
          . ,   /* print eklc */
          1 ,   /* print spm  */
          1 ,   /* print spr  */
```

```
          1 ];  /* print spc  */
< gof,klm,eklm,spm,spr,spc > = sdcspm(dmat,1.5,optn);
```

Search Probability Matrix

	1	2	3	4
1	0	0.999468136	0.999468136	0.999468136
2	0.999468136	0	0.999468136	0.999468136
3	0.999468136	0.999468136	0	0.992836174
4	0.999468136	0.999468136	0.992836174	0
5	0.999468136	0.992836174	0.999468136	0.999468136
6	0.992836174	0.999468136	0.999468136	0.999468136
7	0.998524526	0.998524526	0.998524526	0.998524526
8	0.998524526	0.998524526	0.998524526	0.998524526
9	0.998524526	0.998524526	0.998524526	0.998524526
10	0.998524526	0.998524526	0.998524526	0.998524526

Search Probability Matrix

	5	6	7	8
1	0.999468136	0.992836174	0.997982668	0.997982668
2	0.992836174	0.999468136	0.997982668	0.997982668
3	0.999468136	0.999468136	0.997982668	0.997982668
4	0.999468136	0.999468136	0.997982668	0.997982668
5	0	0.999468136	0.997982668	0.997982668
6	0.999468136	0	0.997982668	0.997982668
7	0.998524526	0.998524526	0	0.999137442
8	0.998524526	0.998524526	0.999137442	0
9	0.998524526	0.998524526	0.999137442	0.999137442
10	0.998524526	0.998524526	0.999137442	0.999137442

Search Probability Matrix

	9	10
1	0.997982668	0.997982668
2	0.997982668	0.997982668
3	0.997982668	0.997982668
4	0.997982668	0.997982668
5	0.997982668	0.997982668
6	0.997982668	0.997982668
7	0.999137442	0.999137442
8	0.999137442	0.999137442
9	0	0.999137442
10	0.999137442	0

Search R Vector

	1
1	10.66666667
2	10.66666667
3	10.66666667
4	10.66666667
5	10.66666667
6	10.66666667
7	10.00000000
8	10.00000000
9	10.00000000
10	10.00000000

Search C Matrix

	1	2	3	4
1	0	2.309401077	2.309401077	2.309401077
2	2.309401077	0	2.309401077	2.309401077
3	2.309401077	2.309401077	0	1.632993162
4	2.309401077	2.309401077	1.632993162	0
5	2.309401077	1.632993162	2.309401077	2.309401077
6	1.632993162	2.309401077	2.309401077	2.309401077
7	1.989038106	1.989038106	1.989038106	1.989038106
8	1.989038106	1.989038106	1.989038106	1.989038106
9	1.989038106	1.989038106	1.989038106	1.989038106
10	1.989038106	1.989038106	1.989038106	1.989038106

Search C Matrix

	5	6	7	8
1	2.309401077	1.632993162	1.925877865	1.925877865
2	1.632993162	2.309401077	1.925877865	1.925877865
3	2.309401077	2.309401077	1.925877865	1.925877865
4	2.309401077	2.309401077	1.925877865	1.925877865
5	0	2.309401077	1.925877865	1.925877865
6	2.309401077	0	1.925877865	1.925877865
7	1.989038106	1.989038106	0	2.160246899
8	1.989038106	1.989038106	2.160246899	0
9	1.989038106	1.989038106	2.160246899	2.160246899
10	1.989038106	1.989038106	2.160246899	2.160246899

Search C Matrix

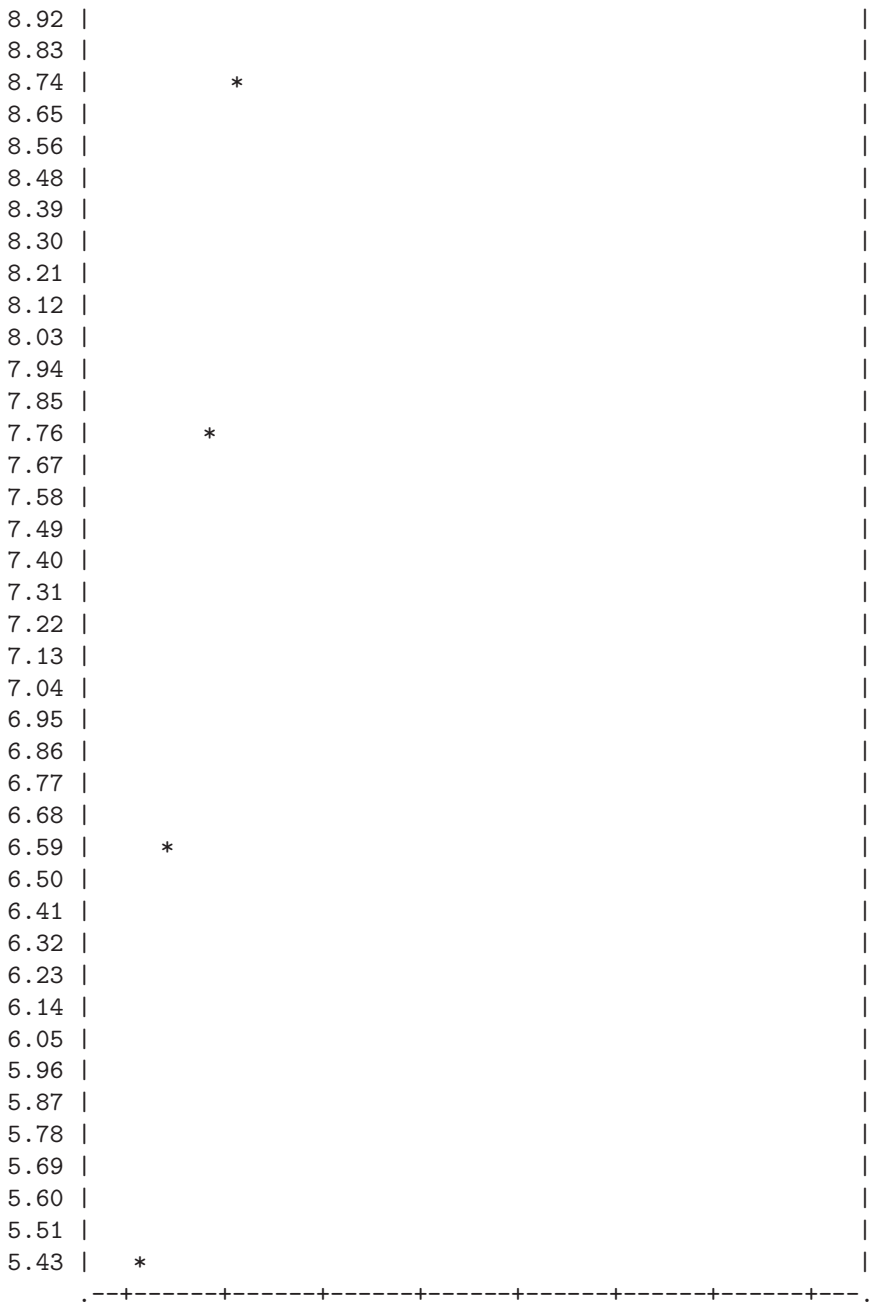
```
                          9           10
                 1  1.925877865  1.925877865
                 2  1.925877865  1.925877865
                 3  1.925877865  1.925877865
                 4  1.925877865  1.925877865
                 5  1.925877865  1.925877865
                 6  1.925877865  1.925877865
                 7  2.160246899  2.160246899
                 8  2.160246899  2.160246899
                 9           0  2.160246899
                10  2.160246899           0
```

```
    Kullback-Leibler Criterion . . . . . 24.000000
    Expected Kullback-Leibler Criterion. . . . . 10.666667
    Searching Probability. . . . . 0.9928362
    Weighted Searching Probability . . . . . .
```

```
print "Plotting SP curve";
/* Spplot(D,3.5,3) */
optn = [  5 ,   /* ipri */
          3 ,   /* nint */
          . ,   /* ktst */
          . ,   /* gam1 */
          . ,   /* gam2 */
          1 ];  /* plot */
gof = sdcspm(dmat,3.5,optn);
```

```
           Plot of Search Probabilities for Rho=3.5
           ****************************************

       .--+------+------+------+------+------+------+------+---.
 10.00 |                      * *  * *  * *  * *  * *  *  *   |
  9.91 |                   *                                  |
  9.82 |                 *                                    |
  9.73 |                                                      |
  9.64 |               *                                      |
  9.55 |                                                      |
  9.46 |                                                      |
  9.37 |                                                      |
  9.28 |             *                                        |
  9.19 |                                                      |
  9.10 |                                                      |
  9.01 |                                                      |
```



Page 23

The Vertical Scale has been Divided by 0.1
The Horicontal Scale has been Divided by 0.1

6 Illustrations

6.1 Two-Phase Logistic Modeling

6.1.1 Example 1: Data by Carroll, Gail & Lubin, (1993)

Note, that this code is available in the files `cmat/test/twophase.inp` and `twop_macro.inp`. The call `cmat twophase.inp` will generate the two output files `twophase.txt` and `twophase.log`.

1. Include the file with the functions for the estimation:

   This file should be available in the `cmat/test` directory.

   ```
   %inc "twop_macro.inp";
   ```

2. Two small data sets (Carroll, Gail & Lubin, 1993):

   The first data set is small but with large counts in column 3. The first column contains the strata numbers and column 2 the binary response (disease, case-control):

   ```
   print "Test1: Carroll Data";
   carr1 = [ 1  0  750 0 ,
             2  0  562 1 ,
             1  1  336 0 ,
             2  1  396 1 ];
   cnam1 = [" Strata Dy count Z "];
   carr1 = cname(carr1,cnam1);
   ```

   The larger data set for phase 2 contains the strata number in the first column, the response in the second column, and the smaller count numbers in the third row. Column 4 is unused and corresponds to the strata, column 6 contains the only predictor variable which is here binary.

   ```
   carr2 = [ 1  0  33 0  0 ,
             1  0  16 0  1 ,
             2  0  11 1  0 ,
             2  0  16 1  1 ,
             1  1  13 0  0 ,
             1  1   5 0  1 ,
             2  1   3 1  0 ,
             2  1  18 1  1 ];
   cnam2 = [" Strata Dy count Z X "];
   carr2 = cname(carr2,cnam2);
   ```

3. Setting the column numbers:

```
par = cons(20,1,.);
par[1] = cs = 1; par[2] = cy = 2;   /* column number strata, disease= y */
par[3] = cc = 3;                    /* column name count */
par[4] = cx = 5; par[5] = cx2 = 5;  /* first and last column number of cat. X vars *
nx = cx2 - cx + 1; nxp = nx + 1;    /* number covars=regr */


par[6] = ipri = 0;
par[7] = maxiter = 1000;   /* like in macro */
par[8] = comp = 1;         /* run comparison macro */
par[9] = eps = 1.e-10;     /* termination, usually 1.e-10 */
par[10] = prosp = 0;       /* never set different? */
```

4. Preparing the data for the parameter estimation:

```
ns = carr1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
nr1 = nrow(carr1); nr2 = nrow(carr2);
< par,nz01,m01,mz01,strat,vars > = twop_inp(par,carr1,carr2);
mres = inp2 = 0; free meth,pase,pase2,covms;
```

5. EM5 ML Estimation:

```
< par,parm,ase,cov > = twop_em5(par,nz01,m01,mz01,strat,vars,carr1,carr2);
if (par[20] >= 0) {
  mres++; str = "EM5 ML Estimation";
  if (mres == 1) meth = str; else meth = meth |> str;
  pase = pase -> (parm -> ase);
  pase2 = pase2 -> cons(ns,2,.);
  covms = covms |> cov;
}
```

6. Breslow-Cain Pseudo Likelihood:

```
< par,parm,ase,cov > = twop_bc(par,nz01,m01,mz01,strat,vars,carr1,carr2);
if (par[20] >= 0) {
  /* note: that BC method yields ns more parms for strata */
  mres++; str = "Breslow-Cain ML Estimation";
  if (mres == 1) meth = str; else meth = meth |> str;
  inp2 = 1;
  lo = ns + 1; up = ns + nxp;
```

```
  pase = pase -> (parm[lo:up] -> ase[lo:up]);
  pase2 = pase2 -> (parm[1:ns] -> ase[1:ns]);
  covms = covms |> cov[lo:up,lo:up];
}
```

7. Pseudo Likelihood Schill:

```
< par,parm,ase,cov > = twop_zlr(par,nz01,m01,mz01,strat,vars,carr1,carr2);
if (par[20] >= 0) {
  /* note: that ZLR method yields ns more parms for strata */
  mres++; str = "Schill Pseudo ML Estimation";
  if (mres == 1) meth = str; else meth = meth |> str;
  inp2 = 1;
  lo = ns + 1; up = ns + nxp;
  pase = pase -> (parm[lo:up] -> ase[lo:up]);
  pase2 = pase2 -> (parm[1:ns] -> ase[1:ns]);
  covms = covms |> cov[lo:up,lo:up];
}
```

8. WLR: Weighted Likelihood Estimation:

```
< par,parm,ase,cov > = twop_wlr(par,nz01,m01,mz01,strat,vars,carr1,carr2);
if (par[20] >= 0) {
  mres++; str = "Weighted Likelihood Estimation";
  if (mres == 1) meth = str; else meth = meth |> str;
  pase = pase -> (parm -> ase);
  pase2 = pase2 -> cons(ns,2,.);
  covms = covms |> cov;
}
```

9. Sample 2 ML Estimation:

```
< par,parm,ase,cov > = twop_s2(par,nz01,m01,mz01,strat,vars,carr1,carr2);
if (par[20] >= 0) {
  mres++; str = "Sample 2 ML Estimation";
  if (mres == 1) meth = str; else meth = meth |> str;
  pase = pase -> (parm -> ase);
  pase2 = pase2 -> cons(ns,2,.);
  covms = covms |> cov;
}
```

10. Code for printing the results:

```
   /* print meth; print pase; print pase2; */
   cnam = cn = [" Parameters AsymStdErr "];
   for (ic = 2; ic <= mres; ic++) cnam = cnam -> cn;
   pase  = cname(pase,cnam);

   options ls=120;
   ic1 = 1; ic2 = ic1 + 2; if (ic2 > mres) ic2 = mres;
newprt1:
   for (ic = ic1; ic <= ic2; ic++) print meth[ic];
   jc1 = 2 * ic1 - 1; jc2 = 2 * ic2;

   if (inp2) print "Estimates for Predictors";
   print pase[,jc1:jc2];
   if (inp2) {
     pase2 = cname(pase2,cnam);
     rnam = prefname("S_",ns);
     pase2 = rname(pase2,rnam);
     print "Additional Estimates for Strata";
     print pase2[,jc1:jc2];
   }
   if (ic2 < mres) {
     ic1 = ic2 + 1; ic2 = ic1 + 2;
     if (ic2 > mres) ic2 = mres;
     goto newprt1;
   }
```

11. Output of the results:

    (a) EM5 ML Estimation

    (b) Breslow-Cain ML Estimation

    (c) Schill Pseudo ML Estimation

```
Estimates for Predictors

        |   Parameters   AsymStdErr   Parameters   AsymStdErr   Parameters   AsymStdErr
--------------------------------------------------------------------------------------
  alpha |     -0.51352      0.14142     -0.30256      0.19631     -0.29962      0.19826
      X |      0.95792      0.23659      0.56875      0.37337      0.56422      0.37600


Additional Estimates for Strata

      |   Parameters   AsymStdErr   Parameters   AsymStdErr   Parameters   AsymStdErr
--------------------------------------------------------------------------------------
```

```
  S_1 |       .            .          -0.21943      0.04670     -0.21338      0.04472
  S_2 |       .            .           0.23344      0.04665      0.22738      0.04482
```

(a) Weighted Likelihood Estimation

(b) Sample 2 ML Estimation

```
Estimates for Predictors

        |   Parameters   AsymStdErr   Parameters   AsymStdErr
-------------------------------------------------------------
  alpha |     -0.31383      0.18685     -0.34443      0.21547
      X |      0.60808      0.35034      0.68136      0.39994
```

6.1.2 Example 2: Data by Pohlabeln et al. (2002)

1. Two larger data sets:

   The first data set is small but with large counts in column 3. The first column contains the strata and column 2 the binary response (disease):

```
print "Test2: HdA Data";
HdA1 = [ 1  0  347 ,
         2  0   62 ,
         3  0   42 ,
         4  0   42 ,
         5  0  210 ,
         6  0   48 ,
         7  0   46 ,
         8  0   42 ,
         1  1  135 ,
         2  1   19 ,
         3  1   32 ,
         4  1   28 ,
         5  1  360 ,
         6  1   80 ,
         7  1   89 ,
         8  1   96 ];
cnam = [" Strata Dy count "];
HdA1 = cname(HdA1,cnam); /* print "HdA1=",HdA1; */
```

   The second data set is so large that we are storing it in the `cmat/tdata` directory:

```
   options NOECHO;
%inc "..\\tdata\\HdA2.dat";
   options ECHO;
```

We permute the columns of the large data set so that the strata column is first, followed by that of the binary response, the count variable, and then all of the predictor variables. The predictor variables SMOKE1, SMOKE2, SMOKE3 are binary and FY is interval scaled:

```
   cnam = [" Obs    Dy    Strata    SMOKE     FY    count
             SMOKE1    SMOKE2    SMOKE3 "];
   HdA2 = cname(HdA2,cnam);
   ind = [ 3 2 6 7 8 9 5 4 ];
   HdA2 = HdA2[,ind];
   cnam = cnam[ind]; HdA2 = cname(HdA2,cnam);
   /* print "[1] HdA2=", HdA2[1:10,]; */
```

The number of different values in the columns of the predictor variables of data set 2 is important for estimating the amount of computer resources some applications require. (For interval scaled predictor variables some of the estimation techniques require the storage of large (sparse) matrices, which may become close to $n2 \times n2$, where $n2$ is the number of rows of the phase-2 data matrix.

```
   /* All predictor vars must be ordinal starting with level 0:
      Sort FY and replace by levels 0,1,2,... */
   fy = HdA2[,7];
   fy = ordinal(fy); print "Nrank=", nrnk = fy[<>] + 1;
   /* HdA2[,7] = fy; HdA2 = cname(HdA2,cnam); */
   print "[2] HdA2=", HdA2[1:10,];
```

2. Setting the column numbers:

```
   par = cons(20,1,.);
   par[1] = cs = 1; par[2] = cy = 2;   /* column number strata, disease= y */
   par[3] = cc = 3;                    /* column name count */
   /* response is Dy, predictors are SMOKE1,2,3,FY */
   par[4] = cx = 4; par[5] = cx2 = 7;  /* first and last column number of cat. X vars *
   nx = cx2 - cx + 1; nxp = nx + 1;    /* number covars=regr */

   par[6] = ipri = 0;
   par[7] = maxiter = 1000;  /* like in macro */
```

```
par[8] = comp = 1;        /* run comparison macro */
par[9] = eps = 1.e-10;    /* termination, usually 1.e-10 */
par[10] = prosp = 0;      /* never set different? */
```

3. Preparing the data for the parameter estimation:

```
ns = HdA1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
nr1 = nrow(HdA1); nr2 = nrow(HdA2);
print "nrow(HdA1,HdA2)=",nr1,nr2;
< par,nz01,m01,mz01,strat,vars > = twop_inp(par,HdA1,HdA2);
mres = inp2 = 0; free meth,pase,pase2,covms;
```

4. The code for calling the functions which obtain the parameter estimates and standard errors is very similar to that shown above for example 1.

5. Output of the results:

   (a) EM5 ML Estimation
   (b) Breslow-Cain ML Estimation
   (c) Schill Pseudo ML Estimation

Estimates for Predictors

	Parameters	AsymStdErr	Parameters	AsymStdErr	Parameters	AsymStdEr
alpha	-1.6159	0.45468	-1.6263	0.45578	-1.6272	0.4562
SMOKE1	0.84504	0.54383	0.94049	0.54140	0.94200	0.5417
SMOKE2	1.9399	0.47946	1.9808	0.47861	1.9839	0.4784
SMOKE3	2.4028	0.50382	2.4197	0.50837	2.4225	0.5086
FY	0.16389	0.05739	0.13211	0.07039	0.12783	0.0737

Additional Estimates for Strata

	Parameters	AsymStdErr	Parameters	AsymStdErr	Parameters	AsymStdErr
S_1	.	.	-0.94405	0.08891	-0.93738	0.08456
S_2	.	.	-1.1827	0.25764	-1.1463	0.22919
S_3	.	.	-0.27193	0.22951	-0.33047	0.20098
S_4	.	.	-0.40547	0.23904	-0.41292	0.20688
S_5	.	.	0.53900	0.07180	0.54064	0.06628
S_6	.	.	0.51083	0.17592	0.51708	0.15275
S_7	.	.	0.65999	0.17490	0.65840	0.14713
S_8	.	.	0.82668	0.17844	0.81449	0.14987

(a) Weighted Likelihood Estimation

(b) Sample 2 ML Estimation

```
Estimates for Predictors

         |   Parameters   AsymStdErr   Parameters   AsymStdErr
-------------------------------------------------------------
   alpha |      -1.6296      0.44913      -1.7106      0.47797
  SMOKE1 |      0.90879      0.53278      0.89373      0.53473
  SMOKE2 |       1.9789      0.47023       2.0175      0.52330
  SMOKE3 |       2.4498      0.49940       2.4417      0.55246
      FY |      0.11708      0.06081      0.14560      0.08239
```

6.1.3 Listing of the file `twop_macro.inp`

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

function twop_inp(par,dat1,dat2)
{
   /* Input part for every estimation : see macro prep */
   par[20] = 0;
   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5]; /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;         /* number covars=regr */

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   /* nr1,nr2: number rows of data */
   nr1 = nrow(dat1); nr2 = nrow(dat2);
   ipri = par[6];
   if (ipri) print "nx,ns,nr1,nr2=",nx,ns,nr1,nr2;
   cnam1 = cname(dat1); cnam2 = cname(dat2);
   xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   nz01 = m01 = mz01 = strat = .;

   /*--- Phase 1 preparation ------------------------------*/
   /* n0: number controls, n1: number events */
   /* nz0[ns]: number controls in strata, nz1[ns]: in events */
   n0 = n1 = 0.; nz0 = nz1 = cons(ns,1,0.);
   for (ir = 1; ir <= nr1; ir++) {
     if (dat1[ir,cc] == 0)
     print "Warning: zero count data in sample 1: row=",ir;
     if (dat1[ir,cy] == 0) {
       n0 += dat1[ir,cc];
```

```
    nz0[dat1[ir,cs]] += dat1[ir,cc];
  } else {
    n1 += dat1[ir,cc];
    nz1[dat1[ir,cs]] += dat1[ir,cc];
} }
par[12] = n0; par[13] = n1;
nz01 = nz0 -> nz1;
if (ipri) print "N0,N1=", n0,n1;
if (ipri) print "nz0,nz1=",nz01;

/* this is what the (PROC MEANS BY &strat &caco &covar) call is doing:
   stored into vectors mz0,mz1[nsx] and not into tensor
   tmp2[nr2,4+nx]: strata,covar,count1,count2 */
nd = 1 + nx;
nc = ic1 = 3 + nx; ic0 = 2 + nx;
tmp2 = cons(nr2,nc,0.);
for (ir = 1; ir <= nr2; ir++) {
  if (dat2[ir,cc] == 0)
  print "Warning: zero count data in sample 2: row=",ir;
  iy = dat2[ir,cy];
  tmp2[ir,1] = dat2[ir,cs]; /* strata starts at 1 */
  if (nx == 1) tmp2[ir,2] = dat2[ir,cx];
  else tmp2[ir,2:nd] = dat2[ir,cx:cx2];
  /* count in last column: no sort */
  if (iy == 0) tmp2[ir,ic0] = dat2[ir,cc];
  else tmp2[ir,ic1] = dat2[ir,cc];
}
if (ipri > 3) {
  cnam = "strata" -> xnam -> [" count0 count1 "];
  tmp2 = cname(tmp2,cnam);
  print "Before Sort Tmp2=",tmp2;
}

/* sort tmp2 wrt. temp2[,1] and compress for nsx */
ind = [ 1:nd ];
< ord,tmp2,rank > = sortrow(tmp2,ind);
if (ipri > 3) print "After Sort Tmp2=",tmp2;
nrnk = rank[nr2];
if (ipri > 1) print "nrnk,rank=",nrnk,rank;

nsx = 1; i1 = rank[1];
for (ir = 2; ir <= nr2; ir++) {
  i2 = rank[ir];
  if (i1 == i2) {
    /* print "same key: row and ind=", ir,i1; */
    tmp2[nsx,ic0] += tmp2[ir,ic0];
```

```
      tmp2[nsx,ic1] += tmp2[ir,ic1];
    } else {
      nsx++;
      tmp2[nsx,] = tmp2[ir,];
      i1 = rank[ir];
  } }
  if (ipri) print "nsx=",nsx;
  tmp2 = tmp2[1:nsx,];

  par[11] = nsx; nsx2 = nsx + nsx;
  strat = tmp2[,1];
  mz0 = tmp2[,ic0]; mz1 = tmp2[,ic1];
  mz01 = mz0 -> mz1;
  if (ipri) print "strat,mz0,mz1=",strat -> mz01;
  vars = (nx == 1) ? tmp2[,2] : tmp2[,2:nd];
  if (ipri > 1) print "vars=",vars;

  /* this is what the (PROC MEANS BY &strat &caco) call is doing */
  /* get m0[ns] and m1[ns] */
  m0 = m1 = cons(ns,1,0.);
  for (ir = 1; ir <= nr2; ir++) {
    j = dat2[ir,cs]; iy = dat2[ir,cy];
    if (iy == 0) m0[j] += dat2[ir,cc];
    else m1[j] += dat2[ir,cc];
  }
  m01 = m0 -> m1;
  if (ipri) print "m0,m1=",m01;
leave:
  return(par,nz01,m01,mz01,strat,vars);
}

/*********************************************************************/
/************** EM5 ML Estimation ************************************/
/*********************************************************************/

function twop_em5(par,nz01,m01,mz01,strat,vars,dat1,dat2)
{
  print "****************** Into EM5 ML Estimation ***************";
  ipri = par[6];
  if (ipri > 1) {
    print "nz01,m01=",nz01,m01;
    print "mz01,strat,vars=",mz01 -> strat -> vars;  /* [nsx],[nsx,nx] */
  }
  parm = cov = ase = .;
  par[20] = 0;
```

```
   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5];   /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;           /* number covars=regr */
   nxp = nx + 1;                /* number parms */

   maxiter = par[7]; comp = par[8];
   eps = par[9]; prosp = par[10];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   nr1 = nrow(dat1); nr2 = nrow(dat2);
   cnam1 = cname(dat1); cnam2 = cname(dat2);
   xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];
   nsx = par[11]; nsx2 = nsx + nsx;
   n0 = par[12]; n1 = par[13];
   if (ipri > 1) print "ns,nsx,n0,n1=",ns,nsx,n0,n1;

   nz0 = nz01[,1]; nz1 = nz01[,2];  /* [ns] */
   m0  = m01[,1];  m1  = m01[,2];   /* [ns] */
   mz0 = mz01[,1]; mz1 = mz01[,2];  /* [nsx] */

/*-------------------------------------------------------------------*/

   /* get nm0[ns]= nz0[ns] - m0[ns] and nm1[ns]= nz1[ns] - m1[ns] */
   nm0 = nm1 = cons(ns);
   for (i = 1; i <= ns; i++) {
     nm0[i] = nz0[i] - m0[i];
     nm1[i] = nz1[i] - m1[i];
   }
   if (ipri > 2) print "nm0,nm1=",nm0 -> nm1;

   /* m_com[nsx2] */
   m_com = mz0 |> mz1;
   /* get m_inc[nsx2], nz0[nsx2] */
   m_inc = nz10 = cons(nsx2);
   l = 1;
   for (j = 1; j <= 2; j++)
   for (i = 1; i <= nsx; i++, l++) {
     is = strat[i];
     m_inc[l] = (j == 1) ? nm0[is] : nm1[is];
     nz10[l]  = (j == 1) ? nz0[is] : nz1[is];
   }
   if (ipri > 2) print "m_com,m_inc,nz10=",m_com -> m_inc -> nz10;

   /* get ss2[nsx2], sy, sz2[nsx2], vars[nsx2,nx] */
   sy = cons(nsx,1,0.) |> cons(nsx,1,1.);
   sz = ss2 = cons(nsx2);
```

```
l = 1;
for (j = 0; j <   2; j++)
for (i = 1; i <= nsx; i++, l++) {
  is = strat[i]; sz[l] = is;
  ss2[l] = 2 * is - 1 + j;
}
if (ipri > 2) print "sy,sz,ss2=",sy -> sz -> ss2;

/* parm[1+nx+nsx] */
np = 1 + nx + nsx;
/* make new function names = prefname(pref,n) */
parmc = prefname("delta_",nsx);
pnam = "alpha" -> xnam -> parmc‘;
v1 = (nx == 1) ? 0. : cons(nx,1,0.);
t = log(n0 / nsx); v2 = cons(nsx,1,t);
parm = log(n1 / n0) |> v1 |> v2;
parm = rname(parm,pnam);
if (ipri > 1) print "parm=",parm;

/* get large sparse design matrix:
   xdes[nsx2,np]= (sy[nsx2],sy[nsx2]#vars[nsx2,nx],ide[nsx2,nsx]) */
hh = cons(nsx,nx,0.) |> vars;
ndes = [ 1:nsx 1:nsx ];
xdes = sy -> hh -> design(ndes);
if (ipri > 3) print "Xdesign=",xdes;

/* get: mu[nsx2], B[nsx2,ns2] */
mu = exp(xdes * parm);
if (ipri > 2) print "mu=",mu;
B = design(ss2); BB = B * B‘;
if (ipri > 3) print "Bdesign=",B;

for (iter = 1; iter <= maxiter; iter++) {
  /* Update weight vector */
  wgt = mu ./ (BB * mu);
  /* Update pseudo-counts mv[nsx2] */
  mv = m_com + diag(wgt) * m_inc;
  mlogm = cons(nsx2,1,0.);
  for (i = 1; i <= nsx; i++)
  if (mv[i] > 0.) mlogm[i] = mv[i] * log(mv[i]);
  v1 = mlogm - mv .* log(mu) - mv + mu;
  devold = 2. * v1[+]; thold = parm;
  rhs = xdes‘* (mv - mu);
  if (iter < 10) {
    sym = xdes‘ * diag(mu) * xdes;
  } else {
```

```
    WW = (wgt * wgt') .* BB;
    sym = xdes' * (diag(mu) - diag(m_inc) * (diag(wgt) - WW)) * xdes;
  }
  /* if (ipri > 3)
     print "Solving linear system with", sym; */

  parm += sym \ rhs;
  for (jter = 1; jter <= 3; jter++) {
    mu = exp(xdes * parm);
    v1 = mlogm - mv .* log(mu) - mv + mu;
    dev = 2. * v1[+];
    if (dev < devold+eps) break;
    parm = .5 * (parm + thold);
  }
  v2 = abs(parm - thold); crit = v2[+];
  if (crit < eps) break;
}
par[19] = crit;
if (iter == maxiter) {
  print "Warning maxiter=",maxiter," iterations without convergence: crit=", crit;
  par[20] = -1; goto leave;
}
if (ipri)
print "Convergence with crit=",crit," after",iter," iterations.";

/* compute COV matrix */
ww = (wgt * wgt') .* BB;
cov = inv(xdes' * (diag(mu) - diag(m_inc) * (diag(wgt) - ww)) * xdes);
if (ipri > 1) print "EM5: parm=",parm;
if (ipri > 2) print "EM5: Cov=",cov;

/* modification of intercept due to retrospective phase one sample */
if (prosp == 0) {
  parm[1] -= log(n1 / n0);
  cov[1,1] -= 1. / n0 + 1. / n1;
}

/* Wrapping it up */
ind = [ 1:nxp ];
parm = parm[ind];
pnam = "alpha" -> xnam; parm = rname(parm,pnam);
cov = cov[ind,ind];
ase = dia2vec(cov);
for (j = 1; j <= nxp; j++)
ase[j] = (ase[j] <= 0.) ? 0. : sqrt(ase[j]);
ase = rname(ase,pnam);
```

```
leave:
   if (ipri > 1) print "Result EM5: parm, ase=", parm -> ase;
   return(par,parm,ase,cov);
}

/*********************************************************************/
/************** Pseudo Likelihood Breslow-Cain ***********************/
/*********************************************************************/

function twop_bc(par,nz01,m01,mz01,strat,vars,dat1,dat2)
{
   print "*** Into Breslow-Cain Pseudo Likelihood Estimation ************";
   ipri = par[6];
   if (ipri > 1) {
     print "nz01,m01=",nz01,m01;
     print "mz01,strat,vars=",mz01 -> strat -> vars;  /* [nsx],[nsx,nx] */
   }
   parm = cov = ase = .;
   par[20] = 0;

   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5]; /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;         /* number covars=regr */
   nxp = 1 + nx;              /* number parms */

   maxiter = par[7]; comp = par[8];
   eps = par[9]; prosp = par[10];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   nr1 = nrow(dat1); nr2 = nrow(dat2);
   cnam1 = cname(dat1); cnam2 = cname(dat2);
   xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];
   nsx = par[11]; nsx2 = nsx + nsx;
   n0 = par[12]; n1 = par[13];
   ntot = (real)n0 + n1;
   if (ipri > 2) print "ns,nxp,n0,n1=",ns,nxp,n0,n1;

   nz0 = nz01[,1]; nz1 = nz01[,2];  /* [ns] */
   m0  = m01[,1];  m1  = m01[,2];   /* [ns] */
   mz0 = mz01[,1]; mz1 = mz01[,2];  /* [nsx] */

/*-------------------------------------------------------------------*/

   offs1 = anz1 = cons(ns,1,0.);
   st1 = cons(ns,ns,0.);
   tt = log(n1 / n0);
```

```
for (is = 1; is <= ns; is++) {
  offs1[is] = (1. - prosp) * tt;
  anz1[is]  = nz0[is] + nz1[is];
  for (i = 1; i <= ns; i++) st1[is,i] = (i == is) ? 1. : 0.;
}

vnam = prefname("s_",ns)‘; if (ipri > 2) print "vnam=", vnam;
cnam = [" offs1 nz1 anz1 "] -> vnam;
tmp1 = offs1 -> nz1 -> anz1 -> st1;
tmp1 = cname(tmp1,cnam);
if (ipri > 2) print "Tmp1=",tmp1;

pri = (ipri > 1) ? 5 : 0;
optn = [ "print"         pri ,
         "noint"             ,
         "offset"          1 ,
         "link"      "logit" ,
         "dist"      "binom" ,
         "trial"           3 ,
         "tech"     "trureg" ];
modl1 = sprintf("2/3 = %i : %i",4,3+ns);
if (ipri > 2) print "Modl=",modl1;
< gof1,parm1,sterr1,conf1,cov1 > = glim(tmp1,modl1,optn);
parm1 = rname(parm1,vnam);
if (ipri > 1) print "Parm1=",parm1;

offs2 = anz2 = alfa = cons(nsx,1,0.);
st2 = cons(nsx,ns,0.);
for (i = 1; i <= nsx; i++) {
  is = strat[i];
  offs2[i]  = log((real)m1[is] / m0[is]) - parm1[is];
  anz2[i]   = mz0[i] + mz1[i];
  st2[i,is] = alfa[i] = 1.;
}

pnam = "alpha" -> xnam;
cnam = [" offs2 mz1 anz2 "] -> pnam;

pred = alfa -> vars;
tmp2 = offs2 -> mz1 -> anz2 -> pred;
tmp2 = cname(tmp2,cnam);
if (ipri > 2) print "Tmp2=",tmp2;

pri = (ipri > 1) ? 5 : 0;
optn = [ "print"         pri ,
         "noint"             ,
```

```
         "offset"          1 ,
         "link"      "logit" ,
         "dist"      "binom" ,
         "trial"           3 ,
         "tech"     "trureg" ];
modl2 = sprintf("2/3 = %i : %i",4,4+nx);
if (ipri > 2) print "Modl=",modl2;
< gof2,parm2,sterr2,conf2,cov2 > = glim(tmp2,modl2,optn);
parm2 = rname(parm2,pnam);
if (ipri > 1) print "Parm2=",parm2;

/* Stage 1: dim=ns */
w1 = nz1 |> nz0; z1 = st1 |> st1;
xd1 = st1 |> st1; of1 = offs1 |> offs1;
eta1 = of1 + xd1 * parm1;
p1 = 1. / (1. + exp(-eta1)); f1 = p1 .* (1. - p1);
if (ipri > 2) print "eta1,p1,f1=", eta1 -> p1 -> f1;
xpxi = z1‘ * (w1 .* f1 .* z1);
if (ipri > 3) print "XPXi=",xpxi;

/* Stage 2: dim=nxp */
w2 = mz1 |> mz0; z2 = st2 |> st2; x2 = pred |> pred;
xd2 = pred |> pred; of2 = offs2 |> offs2;
eta2 = of2 + xd2 * parm2;
p2 = 1. / (1. + exp(-eta2)); f2 = p2 .* (1. - p2);
if (ipri > 2) print "eta2,p2,f2=", eta2 -> p2 -> f2;

/* Stage 1: dim=ns2 */
rho = ntot / n0 + ntot / n1;
vmat = ntot / (z1‘* (f1 .* w1)); if (ipri > 3) print "Vmat=", vmat;
wmat = diag(vmat) - (1. - prosp) * rho * cons(ns,ns,1.);
cov1 = wmat / ntot;
ase1 = dia2vec(cov1);
for (j = 1; j <= ns; j++)
ase1[j] = (ase1[j] <= 0.) ? 0. : sqrt(ase1[j]);
if (ipri > 2) print "ASE1=",ase1;

/* Stage 2: dim=ns+nxp */
ndim = ns + nxp;
x12 = cons(ns2,nxp,0.);
xdes = (z1 -> x12) |> (-z2 -> x2);
wgt = w1 |> w2; fmat = f1 |> f2;
hmat = (xdes‘ * (wgt .* fmat .* xdes)) / ntot;
if (ipri > 3) print "Hmat=",hmat;

/* modification of variance due to 2-stage setup */
```

```
   m20 = st2' * mz0; m21 = st2' * mz1;
   vec = ntot / m20 + ntot / m21;
   qmat = diag(vec); if (ipri > 3) print "Qmat=", qmat;
   nsp = ns + 1;
   ind1 = [ 1 : ns ]; ind2 = [ nsp : ndim ];
   h12 = hmat[ind1,ind2]; h22 = hmat[ind2,ind2]; h22i = inv(h22);
   cov2 = (h22i * (h22 - h12'* (qmat - wmat) * h12) * h22i) / ntot;
   if (ipri > 3) print "COV2=", cov2;

   /* Wrapping it up */
   ase2 = dia2vec(cov2);
   for (j = 1; j <= nxp; j++)
   ase2[j] = (ase2[j] <= 0.) ? 0. : sqrt(ase2[j]);
   /* print "ASE2=",ase2; */

   cov = cov2 \> xpxi;
   parm = parm1 |> parm2; ase = ase1 |> ase2;
leave:
   if (ipri > 1) print "Result BC: parm, ase=", parm -> ase;
   return(par,parm,ase,cov);
}

/*************************************************************************/
/************** Pseudo Likelihood Schill ***************************/
/*************************************************************************/

function twop_zlr(par,nz01,m01,mz01,strat,vars,dat1,dat2)
{
   print "********** Into Schill Pseudo Likelihood Estimation *************";
   ipri = par[6];
   if (ipri > 1) {
     print "nz01,m01=",nz01,m01;
     print "mz01,strat,vars=",mz01 -> strat -> vars;  /* [nsx],[nsx,nx] */
   }
   parm = cov = ase = .;
   par[20] = 0;

   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5]; /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;         /* number covars=regr */
   nxp = 1 + nx;              /* number parms */

   maxiter = par[7]; comp = par[8];
   eps = par[9]; prosp = par[10];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
```

```
  nr1 = nrow(dat1); nr2 = nrow(dat2);
  cnam1 = cname(dat1); cnam2 = cname(dat2);
  xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];

  nsx = par[11]; nsx2 = nsx + nsx;
  n0 = par[12]; n1 = par[13];
  ntot = (real)n0 + n1;
  if (ipri > 2) print "ns,nxp,n0,n1=",ns,nxp,n0,n1;

  nz0 = nz01[,1]; nz1 = nz01[,2];  /* [ns] */
  m0  = m01[,1];  m1  = m01[,2];   /* [ns] */
  mz0 = mz01[,1]; mz1 = mz01[,2];  /* [nsx] */

/*--------------------------------------------------------------------*/

  nssx = ns + nsx; nss2 = nssx + nssx;
  offst = event = trial = alpha = cons(nssx,1,0.);
  st = cons(nssx,ns,0.);
  tt = log(n1 / n0);
  for (is = 1; is <= ns; is++) {
    offst[is] = (1. - prosp) * tt;
    event[is] = nz1[is];
    trial[is] = nz0[is] + nz1[is];
    st[is,is] = 1.; alpha[is] = 0.;
  }
  l = ns + 1;
  for (i = 1; i <= nsx; i++, l++) {
    is = strat[i];
    offst[l] = log((real)m1[is] / m0[is]);
    event[l] = mz1[i];
    trial[l] = mz0[i] + mz1[i];
    st[l,is] = -1.; alpha[l] = 1.;
  }
  if (ipri > 2) print "offs,event,trial=",offst -> event -> trial;
  if (ipri > 3) print "st=",st;

  vnam = prefname("s_",ns)`; /* print "vnam=", vnam; */
  pnam = vnam -> "alpha" -> xnam;
  cnam = [" offst event trial "] -> pnam;

  xvar = cons(ns,nx,0.) |> vars;
  pred = alpha -> xvar;
  tmp1 = offst -> event -> trial -> st -> pred;
  tmp1 = cname(tmp1,cnam);
  if (ipri > 3) print "Tmp1=",tmp1;
```

```
np = ns + nxp;
pri = (ipri > 1) ? 5 : 0;
optn = [ "print"         pri ,
         "noint"             ,
         "offset"          1 ,
         "link"      "logit" ,
         "dist"      "binom" ,
         "trial"           3 ,
         "tech"     "trureg" ];
modl = sprintf("2/3 = %i : %i",4,4+ns+nx);
if (ipri > 2) print "Modl=",modl;
< gof,parm,sterr,conf,cov > = glim(tmp1,modl,optn);
parm = rname(parm,pnam);
if (ipri > 1) print "Parm=",parm;

ns1 = ns + 1;
y1 = cons(ns,1,1.) |> cons(ns,1,0.);
y2 = cons(nsx,1,1.) |> cons(nsx,1,0.);
yv = y1 |> y2;
wv = nz1 |> nz0 |> mz1 |> mz0;
x1 = st[1:ns,]‘ -> alpha[1:ns] -> cons(ns,nx,0.);
x2 = st[ns1:nssx,] -> alpha[ns1:nssx] -> vars;
xv = x1 |> x1 |> x2 |> x2;
if (ipri > 2) print "Xv=", xv;

of1 = offst[1:ns]; of2 = offst[ns1:nssx];
offs = of1 |> of1 |> of2 |> of2;
if (ipri > 2) print "offs=",offs;

x1 = st[1:ns,]‘; x2 = -st[ns1:nssx,]‘;
if (ipri > 2) print "X1,X2=", x1,x2;
m20 = x2 * mz0; m21 = x2 * mz1;
if (ipri > 2) print "m20,m21=",m20,m21;

/* PCL-variance 2-phase logistic regression by Schill */
eta = offs + xv * parm;
pv = 1. / (1. + exp(-eta));
fv = pv .* (1. - pv); if (ipri > 2) print "eta,fv=",eta -> fv;
covp = inv(xv‘ * (wv .* fv .* xv));
dia = dia2vec(covp);
for (j = 1; j <= np; j++)
dia[j] = (dia[j] <= 0.) ? 0. : 1. / sqrt(dia[j]);
corp = diag(dia) * covp * diag(dia);
hmat = (xv‘* (wv .* fv .* xv)) / ntot;
if (ipri > 3) print "Hmat=",hmat;
```

```
   /* Modification due to 2-phase setup, Diss. p29,30 */
   f1 = fv[1:ns] .* (nz1 + nz0);
   vv = diag((x1 * f1) / ntot); if (ipri > 2) print "vv=",vv;
   vv = diag(vv);
   ta = hmat[1:ns,1:ns] - vv; tb = hmat[1:ns,ns1:np];
   qmat = ntot / diag(m20) + ntot / diag(m21);
   if (ipri > 2) print "qmat=",qmat;

   /* nxp = np - ns: (ta,tb)[ns,np] */
   up = ta -> tb; lo = cons(nxp,np,0.);
   ri = up |> lo;
   rho = ntot / n0 + ntot / n1; if (ipri > 2) print "rho=",rho;
   vvt = vv[,+] @ vv[+,]; if (ipri > 3) print "VVT=",vvt;

   ds1 = (1. - prosp) * (((rho * vvt) -> cons(ns,nxp,0.)) |> lo);
   ds2 = ri' * ((qmat -> cons(ns,nxp,0.)) |> lo) * ri;
   if (ipri > 2) print "ds1,ds2=",ds1,ds2;

   hi = inv(hmat);
   cov = (hi * (hmat - ds1 - ds2) * hi) / ntot;
   if (ipri > 2) print "COV=", cov;

   /* Wrapping it up */
   ase = dia2vec(cov);
   for (j = 1; j <= np; j++)
   ase[j] = (ase[j] <= 0.) ? 0. : sqrt(ase[j]);
   ase = rname(ase,pnam);

leave:
   if (ipri > 1) print "Result ZLR: parm, ase=", parm -> ase;
   return(par,parm,ase,cov);
}

/*********************************************************************/
/************ WLR: Weighted Likelihood Estimation ******************/
/*********************************************************************/

function twop_wlr(par,nz01,m01,mz01,strat,vars,dat1,dat2)
{
   print "************ Into WLR Weighted Likelihood Estimation ***************";
   ipri = par[6];
   if (ipri > 1) {
     print "nz01,m01=",nz01,m01;
     print "mz01,strat,vars=",mz01 -> strat -> vars;  /* [nsx],[nsx,nx] */
   }
   parm = cov = ase = .;
```

```
   par[20] = 0;

   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5]; /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;         /* number covars=regr */
   nxp = 1 + nx;              /* number parms */

   maxiter = par[7]; comp = par[8];
   eps = par[9]; prosp = par[10];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   nr1 = nrow(dat1); nr2 = nrow(dat2);
   cnam1 = cname(dat1); cnam2 = cname(dat2);
   xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];

   nsx = par[11]; nsx2 = nsx + nsx;
   n0 = par[12]; n1 = par[13];
   if (ipri > 2) print "ns,nsx,n0,n1=",ns,nsx,n0,n1;

   nz0 = nz01[,1]; nz1 = nz01[,2];  /* [ns] */
   m0  = m01[,1];  m1  = m01[,2];   /* [ns] */
   mz0 = mz01[,1]; mz1 = mz01[,2];  /* [nsx] */

/*-------------------------------------------------------------------*/

   nm0 = nm1 = cons(ns);
   for (i = 1; i <= ns; i++) {
     nm0[i] = nz0[i] - m0[i];
     nm1[i] = nz1[i] - m1[i];
   }
   if (ipri > 2) print "nm0,nm1=",nm0 -> nm1;

   /* get m_com[nsx2],m_inc[nsx2], nz0[nsx2] */
   m_com = mz0 |> mz1;
   m_inc = nz10 = cons(nsx2);
   l = 1L;
   for (j = 0; j <= 1; j++)
   for (i = 1; i <= nsx; i++, l++) {
     is = strat[i];
     m_inc[l] = (j) ? nm1[is] : nm0[is];
     nz10[l]  = (j) ? nz1[is] : nz0[is];
   }
   if (ipri > 2) print "m_com,m_inc,nz10=",m_com,m_inc -> nz10;

/*-------------------------------------------------------------------*/
```

```
  /* Logistic regression needs only sample 2 data [nsx] */
  w0 = w1 = anz = cons(nsx,1,0.);
  for (i = 1; i <= nsx; i++) {
    is = strat[i];
    w0[i]  = (real)nz0[is] / (real)m0[is] * mz0[i];
    w1[i]  = (real)nz1[is] / (real)m1[is] * mz1[i];
    anz[i] = w0[i] + w1[i];
  }
  if (ipri > 2) print "w0,w1,anz=",w0 -> w1 -> anz;
  tmp1 = w1 -> anz -> vars;
  if (ipri > 3) print "Tmp1=",tmp1;

  pri = (ipri > 1) ? 5 : 0;
  optn = [ "print"         pri ,
           "link"      "logit" ,
           "dist"      "binom" ,
           "trial"           2 ,
           "tech"     "trureg" ];
  modl = (nx == 1) ? "1/2 = 3"
                   : sprintf("1/2= %i:%i",3,2+nx);
  if (ipri > 2) print "modl=",modl;
  < gof,parm,sterr,conf,cov > = glim(tmp1,modl,optn);
  pnam = "alpha" -> xnam;
  parm = rname(parm,pnam);
  if (ipri > 1) print "pnam,parm=",pnam,parm;


/*-------------------------------------------------------------------*/

  /* Sort (strat,yv,) w.r.t key = (strat=1,...,ns, caco=1,0) */
  key = cons(nsx2); tmp2 = cons(nsx2,3);
  l = 1;
  for (j = 0; j <= 1; j++)
  for (i = 1; i <= nsx; i++, l++) {
    is = strat[i];
    key[l] = (is - 1.) * 2. + (1. - j);
    tmp2[l,1] = (j) ? nz1[is] : nz0[is];
    tmp2[l,2] = (j) ? m1[is] : m0[is];
    tmp2[l,3] = (j) ? mz1[i] : mz0[i];
  }

  /* tmp1 = (key,caco,strat) (nz01,m01,mz01,m_com,vars) */
  caco = cons(nsx,1,0.) |> cons(nsx,1,1.);
  vrs = vars |> vars;
  nc = 6 + nx; str = strat |> strat;
  tmp1 = key -> caco -> str -> tmp2 -> m_com -> vrs;
```

```
cnam = [" key caco strat nz01 m01 mz01 m_com "] -> xnam;
tmp1 = cname(tmp1,cnam);
if (ipri > 3) print "For sorting:", tmp1;
< ord,tmp1,rank > = sortrow(tmp1,1);
if (ipri > 3) print "Sort rank=", rank;
if (ipri > 3) print "After sorting:", tmp1;
caco = tmp1[,2]; strat = tmp1[,3];
m_com = tmp1[,7];
if (nx == 1) vars = tmp1[,8];
else { ux = 7+nx; vars = tmp1[,8:ux]; }
if (ipri > 2) print "VARS=", vars;

w0 = w1 = ww = cons(nsx,1,0.);
mm = mcsd = misd = cons(nsx2);
ll = 1;
for (is = 1; is <= ns; is++) {
  ca = c0 = 0.;
  for (i = 1; i <= nsx2; i++)
  if (strat[i] == is) {
    iy = caco[i];
    if (iy) ca += m_com[i]; else c0 += m_com[i];
  }
  if (ipri > 2) print "ca,c0=",ca,c0;

  l = ll;
  for (i = k = 1; i <= nsx2; i++)
  if (strat[i] == is) {
    iy = caco[i];
    if (iy) w1[k] = m_com[i] / ca; else w0[k] = m_com[i] / c0;
    ww[k] = w1[k] + w0[k];
    k++; l++;
  }
  if (ipri > 2) print "strata,N=",is,l;
  if (ipri > 5) print "w1,w0,ww=", w1 -> w0 -> ww;

  for (i = k = 1; i <= nsx2; i++)
  if (strat[i] == is) {
    iy = caco[i];
    mcsd[ll] = (iy) ? ca : c0;
    misd[ll] = (iy) ? nm1[is] : nm0[is];
    mm[ll]   = m_com[i] + misd[ll] * ww[k];
    k++; ll++;
} }
if (ipri > 2) print "mm,mcom,mcomsd,mincsd=",mm -> m_com -> mcsd -> misd;

/* Design matrix for logistic model */
```

```
xdes = cons(nsx2,1,1.) -> vars;

/* variance for weighted logistic regression according to Reilly and Pepe () */
pv = 1. / (1. + exp(-xdes * parm));
fv = pv .* (1. - pv);
/* ww = mm ./ fv; xx = fv .* xdes; */

qv = (mcsd + misd) ./ mcsd;
ww = (caco - pv) .* (caco - pv);
a1 = xdes‘ * (m_com .* qv .* (qv - 1.) .* ww .* xdes);
if (ipri > 2) print "A1=",a1;

ns2 = ns + ns;
m01 = q01 = cons(ns2);
u01 = cons(ns2,nxp);
for (is = k = 1; is <= ns; is++) {
  m0i = m1i = q0i = q1i = cons(nsx2);
  u0i = u1i = cons(nsx2,nxp);
  for (i = 1; i <= nsx2; i++)
  if (strat[i] == is) {
    iy = caco[i];
    if (iy) m1i[i] = mcsd[i]; else m0i[i] = mcsd[i];
    if (iy) q1i[i] = qv[i]; else q0i[i] = qv[i];
    if (iy) u1i[i] = (iy - pv[i]) .* m_com[i] .* xdes[i,];
       else u0i[i] = (iy - pv[i]) .* m_com[i] .* xdes[i,];
  }
  /* if (ipri > 2) print "u01,u1i=",u0i,u1i; */
  m01[k] = m1i[<>]; m01[k+1] = m0i[<>];
  q01[k] = q1i[<>]; q01[k+1] = q0i[<>];
  u01[k,]   = u1i[+,] ./ m01[k];
  u01[k+1,] = u0i[+,] ./ m01[k+1];
  k += 2;
}
if (ipri > 2) print "m01,q01,u01=",m01 -> q01 -> u01;

a2 = u01‘* (q01 .* (q01 - 1.) .* m01 .* u01);
if (ipri > 3) print "A2=",a2;
hm = inv(xdes‘ * (m_com .* qv .* fv .* xdes));
if (ipri > 3) print "HM=",hm;
cov = hm + hm * (a1 - a2) * hm;
if (ipri > 2) print "COV=", cov;

/* modification of intercept due to retrospective phase one sample */
if (prosp == 0) {
  parm[1] -= log(n1 / n0);
  cov[1,1] -= 1. / n0 + 1. / n1;
```

```
   }

   /* Wrapping it up */
   ase = dia2vec(cov);
   for (j = 1; j <= nxp; j++)
   ase[j] = (ase[j] <= 0.) ? 0. : sqrt(ase[j]);
   ase = rname(ase,pnam);
leave:
   if (ipri > 1) print "Result WLR: parm, ase=", parm -> ase;
   return(par,parm,ase,cov);
}

/*************************************************************************/
/************** S2 Complete Case ML ***********************************/
/*************************************************************************/

function twop_s2(par,nz01,m01,mz01,strat,vars,dat1,dat2)
{
   print "************ Into S2 Likelihood Estimation ***************";
   ipri = par[6];
   if (ipri > 1) {
     print "nz01,m01=",nz01,m01;
     print "mz01,strat,vars=",mz01 -> strat -> vars;  /* [nsx],[nsx,nx] */
   }
   parm = cov = ase = .;
   par[20] = 0;

   cs = par[1]; cy = par[2]; cc = par[3];
   cx = par[4]; cx2 = par[5]; /* first and last column number of cat. X vars */
   nx = cx2 - cx + 1;         /* number covars=regr */
   nxp = 1 + nx;              /* number parms */

   maxiter = par[7]; comp = par[8];
   eps = par[9]; prosp = par[10];

   ns = dat1[<>,cs]; ns2 = ns + ns; /* ns: number strata */
   nr1 = nrow(dat1); nr2 = nrow(dat2);
   cnam1 = cname(dat1); cnam2 = cname(dat2);
   xnam = (nx == 1) ? cnam2[cx] : cnam2[cx:cx2];

   nsx = par[11]; nsx2 = nsx + nsx;
   n0 = par[12]; n1 = par[13];
   if (ipri > 2) print "ns,nsx,n0,n1=",ns,nsx,n0,n1;

   nz0 = nz01[,1]; nz1 = nz01[,2];  /* [ns] */
   m0  = m01[,1];  m1  = m01[,2];   /* [ns] */
```

```
   mz0 = mz01[,1]; mz1 = mz01[,2];  /* [nsx] */

/*-------------------------------------------------------------------*/

   offst = event = trial = alpha = cons(nsx,1,0.);
   st = cons(nsx,ns,0.);
   sm20 = m0[+]; sm21 = m1[+]; tt = log((real)sm21 / sm20);
   if (ipri > 2) print "sm20,sm21=", sm20,sm21;
   for (i = 1; i <= nsx; i++) {
     is = strat[i];
     offst[i] = tt;
     event[i] = mz1[i];
     trial[i] = mz0[i] + mz1[i];
     st[i,is] = alpha[i] = 1.;
   }
   if (ipri > 2) print "offs,event,trial,alpha=",offst -> event -> trial -> alpha;
   if (ipri > 3) print "st=",st;

   pnam = "alpha" -> xnam;
   cnam = [" offst event trial "] -> pnam;

   pred = alpha -> vars;
   tmp1 = offst -> event -> trial -> pred;
   tmp1 = cname(tmp1,cnam);
   if (ipri > 3) print "Tmp1=",tmp1;

   np = nxp;
   pri = (ipri > 1) ? 5 : 0;
   optn = [ "print"         pri ,
            "noint"             ,
            "offset"          1 ,
            "link"      "logit" ,
            "dist"      "binom" ,
            "trial"           3 ,
            "tech"     "trureg" ];
   modl = sprintf("2/3 = %i : %i",4,4+nx);
   if (ipri > 2) print "Modl=",modl;
   < gof,parm,sterr,conf,cov > = glim(tmp1,modl,optn);
   pnam = prefname("est_",np)‘; /* print "pnam=", pnam; */
   parm = rname(parm,pnam);

   yv = cons(nsx,1,1.) |>  cons(nsx,1,0.);
   wgt = mz1 |> mz0;
   off = offst |> offst;
   pred = alpha -> vars;
   xdes = pred |> pred; if (ipri > 3)  print "Xdes=",xdes;
```

```
   eta = off + xdes * parm;
   pv = 1. / (1 + exp(eta)); fv = pv .* (1. - pv);
   if (ipri > 2) print "eta,fv=",eta -> fv;
   cov = inv(xdes‘ * (wgt .* fv .* xdes));

   /* Sample 2 is always retrospective : correction */
   cov[1,1] = cov[1,1] - (1. / sm20 + 1. / sm21);
   if (ipri > 2) print "COV=", cov;

   /* Wrapping it up */
   pnam = "alpha" -> xnam; parm = rname(parm,pnam);
   ase = dia2vec(cov);
   for (j = 1; j <= np; j++)
   ase[j] = (ase[j] <= 0.) ? 0. : sqrt(ase[j]);
   ase = rname(ase,pnam);
leave:
   if (ipri > 1) print "Result S2: parm, ase=", parm -> ase;
   return(par,parm,ase,cov);
}

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

7 The Bibliography

References

[1] Botev, Z. I., Grotowski, J. F., & Kroese, D. P. (2010), "Kernel Density Estimation Via Diffusion", *Annals of Statistics*, **38**, 2916-2957.

[2] Breslow, N. (2005), "Case-Control Study, Two-phase", in P. Armitage (ed.), *Encyclopedia of Biostatistics*, pp. 734-741, New York: J. Wiley & Sons.

[3] Cain, K.C. & Breslow, N.E. (1988), "Logistic regression analysis and efficient design for two-stage studies", *American Journal of Epidemiology*, **128**, 1198-1206.

[4] Carroll, R., Gail, M., Lubin, J. (1993), "Case-control studies with errors in covariates", *Journal of the American Stat. Association*, **88**, 185-199.

[5] Esmailzadeh, N. (2013), "Two level search designs in Matlab", paper and software submitted to JSS.

[6] Fig, M. (2010), Matlab toolbox for permutations and combinations.

[7] Ghosh, S. & Teschmacher, L. (2002), "Comparisons of search designs using search probabilities", *Journal of Statistical Planning and Inference*, **104**, 439-458.

[8] Herbrich, R. (2002), *Learning Kernel Classifiers: Theory and Algorithms*, Cambridge and London: MIT Press.

[9] Pohlabeln, H., Wild, P., Schill, W., Ahrens, W., Jahn, I., Bohn-Audorff, U., Jöckel, K.H. (2002), "Asbestos fibreyears and lung cancer: A two-phase case-control study with expert exposure assessment", *Occupational and Environmental Medicine*, **59**, 410-414.

[10] Schill, W., Enders, D., & Drescher, K. (2013), "sas-twophase-package: A SAS Package for logistic two-phase studies", paper and software submitted to JSS. Software can be downloaded from: `www.bips.uni-bremen.de/sastwophase`

[11] Schill, W., Wild, P., & Pigeot, I. (2007), "A planning tool for two-phase case-control studies", *Computer Programs and Methods in Biomedicine*, **88**, 175-181.

[12] Srivastava, J. (1975), "Designs for searching non-negligible effects", in: *A Survey of Statistical Design and Linear Models*, 507-519. Amsterdam: Elsevier, North Holland.

[13] Talebi, H. & Esmailzadeh, N. (2011a), "Using Kullback-Leibler distance for performance evaluation of search designs", *Bulletin of the Iranian Mathematical Society*, **37**, 269-279.

[14] Talebi, H. & Esmailzadeh, N. (2011b), "Weighted searching probability for classes of equivalent search design comparison", *Communication in Statistics: Theory and Methods*, **40**, 635-647.

[15] Thayananthan, A. (2005) "Template based pose estimation and tracking of 3D hand motion", PhD Thesis, Dept. of Engineering, University of Cambridge.

[16] Tipping, M. E. (2001), "Sparse Bayesian learning and the relevance vector machine", *The Journal of Machine Learning Research*, **1**, 211-244.