CMAT Newsletter: July 2010

Wolfgang M. Hartmann

July 2010

Contents

1 General Remarks

The most important accomplishment of this development season was the implementation of a set of survival functions and an interface to the `gnuplot` software. Users now can generate graphs using *gnuplot's* scripting language. The *gnuplot* script can be either part of the CMAT script (using the `gnuplot ... gpend` syntax) or can be executed having script text files as input arguments for the `gpbatch` function. The *gnuplot* software must be available in a directory located at the same hierarchy level as the `cmat` directory.
The time spent on the *gnuplot* interface was small compared with my implementation of Breiman's *Random Forest* algorithm. For copyright reasons, I did decide to implement the algorithm very independently from the Breiman & Cutler Fortran code and found a few bugs in their Fortran code (among other it returns wrong Gini values when the variance importrance is computed).

1.1 New Functions

affrma RMA (Robust Multichip Average) by Bolstad et.al (2003) for the normalization of microarray data.

boruta implements the Boruta algorithm by Kursa and Rudnicki (2010) for feature selection. (boruta)

gpbatch run *gnuplot* in batch mode (executing *gnuplot* command script files). (gpbatch)

help shows the description pf a term in the reference manual by means of a call to *Acrobat Reader* (help)

maxn find the n largest entries of a set of input objects and their index locations (maxn)

minn find the n smallest entries of a set of input objects and their index locations (minn)

mdy returns the number of days since the year 0 (or optional since 1900, or 1960) for a date input of form; this function is mostly used to find the number of days between two specified dates; (mdy)

propurs 2-dimensional projection pursuit PCA algorithm by Friedman and Tukey (1974) (propurs)

ranfor implements a version of the Breiman and Cutler *Random Forest* modeling algorithm (ranfor)

rafprd uses the *Random Forest* model computed using the `ranfor` function for the prediction of additional data. (rafprd)

spawn executes a child process (spawn)

survcurv implements some algorithms for the computation of survival curves for censored data for Aalen's additive model and the Cox proportional hazards model for censored data (survcurv)

survreg implements some GLIM-like algorithms for survival regression, Aalen's additive model, and the Cox proportional hazards model for censored data (survreg)

system executes shell commands. (system)

zip7 compresses and decompresses files into .zip archives (zip7)

1.2 Fixed Bugs

A number of bugs were fixed, especially some for the `ksprob` function.

2 Modifications of Features

2.1 Modification of the `ksprob` Function

p = ksprob(n,x<,"bar"|"sle"|"fap"|"mtw">)

Several modifications and extensions to the Marsaglia, Tsang, & Wang (2003) Durbin-matrix algorithm for computing the Kolmogorov-Smirnov probability $p = F_n(x)$ were proposed by Simard, & L'Ecuyer (2010). In addition to the Marsaglia, Wand, & Tsang algorithm depending on the values of n and p some additional algorithms are applied:

- Ruben-Gambina formula
- Pomeranz algorithm
- Pelz-Good algorithm
- algorithms for the complementary $1 - F_n(x)$ problem.

This modifications ensure not only some gain on computational speed (especially for large n) they also ensure

1. for $n \leq 140$ between 13 and 15 correct digits
2. for $140 < n \leq 10^5$ at least 5 correct decimal digits
3. for $n > 10^5$, e.g. for $n = 10^6$:
   - at least 5 decimal digits for $F_n(x) > 10^{-33}$
   - at least 2 decimal digits for $F_n(x) > 10^{-120}$
   - at least 1 decimal digit for $F_n(x) > 10^{-230}$

The function `p = ksprob(n,d)` was extended by an optional third input argument which must be one of the following strings

"bar" computes the complimentary $1-F_n(x)$ problem by the Simard, & L'Ecuyer (2010) algorithm

"sle" is using the Simard, & L'Ecuyer (2010) algorithm for $p = F_n(x)$ (which is the default)

"fap" is using the Simard, & L'Ecuyer (2010) algorithm for $p = F_n(x)$, but permits a fast approximation for

$$n * x * x > 7.24 \quad \text{or} \quad n * x * x > 3.76 \quad \text{and} \quad n > 99$$

which ensures only 7 significant digits (which is also part of the Marsaglia, Tsang, & Wang)

"mtw" computes $p = F_n(x)$ using the original Marsaglia, Tsang, & Wang (2003) Durbin-matrix algorithm for all n and x.

The following examples show some impressive results for the complementary $1 - F_n(x)$ problem:

```
print "Pomeranz";
n = 120; d = .0874483967333; K = 0.30012;
p = ksprob(n,d,"bar");
print "2.1: n,d,p=", n,d,p;
```

```
2.1: n,d,p= 120 0.08745 0.3001
```

```
print "Pelz";
n = 500; d = .037527424; K = 0.47067;
p = ksprob(n,d,"bar");
print "2.2: n,d,p=", n,d,p;
```

```
2.2: n,d,p= 500 0.03753 0.4707
```

```
print "KSfbar";
n = 20; d = 0.8008915818; K = 2.5754e-14;
p = ksprob(n,d,"bar");
print "2.3: n,d,p=", n,d,p;
```

```
2.3: n,d,p= 20 0.8009 2.575e-014
```

```
 print "KSfbar";
 n = 20; d = 0.9004583223; K = 1.8250e-20;
 p = ksprob(n,d,"bar");
 print "2.4: n,d,p=", n,d,p;


2.4: n,d,p= 20 0.9005 1.825e-020
```

3 Extensions to the Language

3.1 `gnuplot` Statement

The `gnuplot ... gpend` syntax permits the input of code submitted to the *gnuplot* software. The following is an example:

```
   gnuplot
# Test Example for connecting gnuplot:
      plot [] [-2:2] sin(x), x, x-(x**3)/6
      pause -1 "Hit return to continue"
   gpend
```

Note, that comments must start with an `#` in the first location of a line.
The user must download a current version of the *gnuplot* software from the internet and unzip it into a `gnuplot` directory parallel to the c̃mat directory.
The use of some terminals (output devices in *gnuplot*) needs downloading of additional software from the internet:

svg needs the `SVGView.exe` software, e.g. downloadable for no charge from Adobe.

epslatex needs the *graphicx* software which is downloadable for no charge from the internet.

If a *gnuplot* script needs a data set (which should be an ASCI text data set) it may be generated from a CMAT data object using the `fprintf` function.
Some numerical algorithms related to graphics are available, like the `intpol()`, `quadip()`, and `fft()` functions. There are also a few printer plotting functions available in CMAT.

4 Extensions to Various Functions

4.1 Extension to the `min` and `max` Functions

We have extended the `val = min(a,...)` and `val = max(a,...)` functions for a second return argument to `< val,ind > = min(a,...)` and `< val,ind > = max(a,...)`.

1. For only one input argument `a` the returned value `val` is a numeric scalar which is the smallest resp. largest value of object `a`. The second return argument `ind` is a vector of two (for vectors and matrices `a`) or more (for tensors `a`) indices of the location of the smallest resp. largest value in the object.

2. For $k \geq 2$ input arguments `(a,b,...)`, `val` is a k vector and `ind` is a $k \times m$ matrix, where $m = \max(dim(a), dim(b), ...)$ and each row of `val` and `ind` corresponds to one of the k arguments.

Note, that the values of the indices are either nonnegative or positive depending on the value of the optional index base.

```
a = [ 7 6 1 2 3 4 5 ];
b = [ 2. 3.5 1.5 0.5 ];


< val,ind > = min(a,b);
print "Val=",val;
print "Ind=",ind;
```

```
Val=
   |         1
--------------
 1 |   1.00000
 2 |   0.50000
```

```
print "Data are sparse Matrices";
a = [ 1 0 0 2,
      4 0 0 3,
      5 0 0 0 ];
b = a';


< val,ind > = max(a,b);
print "Val=",val;
print "Ind=",ind;
```

```
Val=
   |   1
--------
 1 |   5
 2 |   5


Ind=
 S |   1   2
------------
 1 |   3
 2 |   1   3
```

A second example shows the application with a sparse tensor:

```
   print "Data are sparse Tensors";
   int at[4,4,4];
   at[1,2,3] = -4;
   at[2,1,4] =  3;
   at[3,3,2] = -2;
   at[4,4,2] =  1;
   print "At=", at;
   bt = at‘;


   < val,ind > = min(at,bt);
   print "Val=",val;
   print "Ind=",ind;
```

```
Val=
   |   1
--------
 1 |  -4
 2 |  -4


Ind=
   |   1   2   3
----------------
 1 |   1   2   3
 2 |   3   2   1
```

```
   < val,ind > = max(at,bt);
   print "Val=",val;
   print "Ind=",ind;
```

```
Val=
   |   1
--------
 1 |   3
 2 |   3


Ind=
   |   1   2   3
---------------
 1 |   2   1   4
 2 |   4   1   2
```

4.2 Extensions to the `nnmf` Function

4.2.1 Orthogonal Nonnegative Matrix Factorizations

For the $N \times n$ matrix $\mathbf{A}$ the unsymmetric nonnegative matrix factorization

$$\mathbf{A} = \mathbf{U}\mathbf{H}^T$$

can be extended to three different forms of orthogonal nonnegative matrix factorizations:

left-orthogonal factorization:

$$\mathbf{A} = \mathbf{U}\mathbf{H}^T$$

with column orthogonal $N \times p$ matrix $\mathbf{U}$, $\mathbf{U}^{\mathbf{U}} = I_p$, and $n \times p$ matrix $\mathbf{H}$

right-orthogonal factorization:

$$\mathbf{A} = \mathbf{U}\mathbf{H}^T$$

with $N \times p$ matrix $\mathbf{U}$, and column orthogonal $n \times p$ matrix $\mathbf{H}$, $\mathbf{H}^{\mathbf{H}} = I_p$

bi-orthogonal factorization:

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{H}^T$$

with column orthogonal $N \times p$ matrix $\mathbf{U}$, column orthogonal $n \times p$ matrix $\mathbf{H}$, and unsymmetric $p \times p$ matrix $\mathbf{W}$.

With our implementation, orthogonality is achieved only approximately. The examples show that true orthogonality is not easy to obtain within of a reasonable number of iterations. Ding, Peng, & Park (2006) show the relationship between orthogonal nonnegative matrix factorizations and *K-means* clustering. The algorithms are based on papers by Chris Ding et al.

4.2.2 Symmetric Nonnegative Matrix Factorization

Without additional constraints the nonnegative matrix factorization of a symmetric $n \times n$ matrix $\mathbf{C}$ results in an unsymmetric matrix $\mathbf{UH}^T$ for different matrices $\mathbf{U}$ and $\mathbf{H}$.
But there is also a need to factorize a symmetric $n \times n$ matrix $\mathbf{C}$ in symmetric ways corresponding to the following two model equations:

1. unweighted:
$$\mathbf{C} = \mathbf{HH}^T$$
2. weighted:
$$\mathbf{C} = \mathbf{HSH}^T$$

with $n \times p$ matrix $\mathbf{H}$ and symmetric $p \times p$ matrix $\mathbf{S}$ both with nonnegative entries, usually for $p \ll n$.
In addition the $n \times p$ matrix $\mathbf{H}$ can be constrained to become *approximately* column orthogonal $\mathbf{H^H} = I_p$. This is called either unweighted or weighted *orthogonal symmetric* nonnegative matrix factorization.
Note that the convergence behavior of the algorithms improves when the symmetric $n \times n$ matrix $\mathbf{C}$ is scaled and positive definite. Examples for such matrices are scaled cross product matrices like

$$\mathbf{D}^{-1}\mathbf{A}^T\mathbf{AD}^{-1}$$

(enforcing unit diagonal). Also diagonal dominant proximity matrices, like those (based on count data) computed by the Random Forest algorithm (`ranfor` function) are good examples. (Of course data matrices cannot be centered, since centering would introduce negative entries.)

4.2.3 New Options for the `nnmf` Function

Option	Second Column	Meaning
`"bet_h"`	real	step size (default=.5)
`"bet_w"`	real	step size (default=.5)
`"meth"`	"sym"	symmetric
`"meth"`	"ort_s"	symmetric orthogonal
`"meth"`	"ort_w"	left-orthogonal
`"meth"`	"ort_h"	right-orthogonal
`"meth"`	"ort_b"	bi-orthogonal
`"prest"`		only for SYM: include preprocessor step
`"rest"`	int	orthogonal methods: number of iterations
		when small loadings are set to zero, def=maxit/2

4.2.4 Some Examples

For all examples we use the 5871×175 sparse Reuters data set with 10432 nonzero elemnts:

```
print "Weighted Reuters Data Set in Sparse Form: (ir,ic,val)";
print "nrow=5871, ncol=175";
options NOECHO;
%inc "../tdata/reuters.dat";
options ECHO;
datb = shape(data,.,3);
print "Maxima=", max = datb[ <>, ];


nr = 5871; nc = max[2]; nf = 2;
datc = spmat(nr,nc,datb[,1],datb[,2],datb[,3]);
attrib(datc);
```

For the unsymmetric orthogonal methods the data is normalized columnwise using the diagonal of the crossproduct matrix:

```
/* column Normalization */
c = datc‘ * datc;
d = inv(sqrt(diag(c))); /* print d; */
data = datc * d; /* print "Data=",data; */
```

For the symmetric methods the crossproduct matrix is normalized for a positive (semi-)definite matrix:

```
c = datc‘ * datc;
d = diag(c);      /* print d; */
d = inv(sqrt(d)); /* print d; */
c = d * c * d; /* print "Data C=",c; */
```

By default, starting values are generated randomly. Needless to say that different starting values may change the results.

1. Left-orthogonal NNMF:

```
print "Random Starting Values: Left Orthogonal";
optn = [ "print"          2 ,      /* ipri */
         "meth"     "ort_u" ,      /* imet=ORT_U */
         "maxit"        400 ,      /* maxiter */
         "rest"          40 ,      /* start zeroing */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< u,h > = nnmf(data,nf,optn);
/* print "Matrix U:", u; */
```

```
print "Matrix H:", h;
print "Nonzeros: U=10729, H=324";
attrib(u);
attrib(h);
```

```
              Nonnegative Matrix Factorization
         Left-Orthogonal Method (Unweighted): U * H'

       Iter      L2Crit    DiffCrit      L1Crit     LooCrit

          1  39723.5152  277199.591  171102.653  0.66760106
          2  5100.64829  34622.8669  61284.6673  0.63364779
          3  783.292133  4317.35616  22820.5199  0.68093782
          4  247.570958  535.721175  9281.50246  0.69768090
          5  182.044342  65.5266162  4504.36186  0.70360769
          6  174.325002  7.71933954  2833.51670  0.70569132
          7  173.415061  0.90994132  2276.77503  0.70639134
          8  173.093135  0.32192613  2138.15714  0.70656603
          9  172.552836  0.54029916  2180.75065  0.70649216
         10  171.562845  0.98999035  2322.31149  0.70620972
         ..................................................
         79  151.230247  6.236e-005  1998.16080  0.67963651
         80  151.230194  5.273e-005  1998.24117  0.67964474
         81  151.230149  4.482e-005  1998.31541  0.67965252

       L2 Precision : 151.23   L1 Precision : 1998.32
                 Computer Time : 14 seconds
```

For true orthogonality both matrices, the left 5871×2 matrix $\mathbf{U}$ and the right 175×2 matrix $\mathbf{H}$ would have only one nonzero entry in each of its rows. Even though both matrices have many small entries they are not really sparse:

```
Nonzeros: U=10729, H=324
```

2. Right-orthogonal NNMF:

```
print "Random Starting Values: Right Orthogonal";
optn = [ "print"          2 ,      /* ipri */
         "meth"     "ort_h" ,      /* imet=ORT_H */
         "maxit"        400 ,      /* maxiter */
         "rest"          40 ,      /* start zeroing */
```

```
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< u,h > = nnmf(data,nf,optn);
/* print "Matrix U:", u; */
print "Matrix H:", h;
print "Nonzeros: U=10614, H=347";
attrib(u);
attrib(h);
```

```
            Nonnegative Matrix Factorization
         Right-Orthogonal Method (Unweighted) : U * H'

      Iter      L2Crit    DiffCrit      L1Crit     LooCrit

         1  40094.1007  276829.006  172026.477  0.66821197
         2  5229.11192  34864.9888  62177.6924  0.63010703
         3  813.007479  4416.10444  23461.7976  0.67714552
         4  249.014211  563.993267  9683.61609  0.69380230
         5  176.169657  72.8445538  4718.64927  0.69962985
         6  166.298151  9.87150662  2912.01275  0.70147824
         7  164.277317  2.02083384  2259.37913  0.70168566
         8  163.059872  1.21744491  2040.11947  0.70100766
         9  161.886984  1.17288803  1983.28893  0.69965894
        10  160.755271  1.13171262  1980.10409  0.69769559
        ..............................................
        72  151.174890  8.571e-005  1995.30670  0.67947893
        73  151.174809  8.031e-005  1995.30260  0.67948071
        74  151.174734  7.536e-005  1995.29857  0.67948242

       L2 Precision : 151.175   L1 Precision : 1995.3
                 Computer Time : 13 seconds
```

For true orthogonality both matrices, the left 5871×2 matrix $\mathbf{U}$ and the right 175×2 matrix $\mathbf{H}$ would have only one nonzero entry in each of its rows. Even though both matrices have many small entries they are not really sparse:

```
Nonzeros: U=10614, H=347
```

3. Bi-orthogonal NNMF:

```
print "Random Initial Values: Biorthogonal";
optn = [ "print"          2 ,      /* ipri */
         "meth"     "ort_b" ,      /* imet=ORT_B */
         "maxit"        400 ,      /* maxiter */
         "rest"          40 ,      /* start zeroing */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< u,h,s > = nnmf(data,nf,optn);
/* print "Matrix U:", u; */
print "Matrix H:", h;
print "Matrix S:", s;
print "Nonzeros: U=10440, H=332";
attrib(u);
attrib(h);
```

The $L2$ fit criterion is close for all three methods, but the bi-orthogonal method needs more computer time than the other methods:

```
                Nonnegative Matrix Factorization
           Bi-Orthogonal Method (Weighted) : U * S * H'

        Iter      L2Crit    DiffCrit      L1Crit     LooCrit

           1  519.081582  622864.815  17755.5235  0.68691593
           2  178.573418  340.508164  3774.16007  0.70414848
           3  174.108788  4.46463059  2204.28247  0.70608469
           4  174.030996  0.07779140  1833.72538  0.70654816
           5  174.046299 -0.01530259  1723.20678  0.70669723
           6  173.978791  0.06750725  1707.02666  0.70673623
           7  173.807529  0.17126248  1742.99876  0.70671340
           8  173.492479  0.31504974  1819.56049  0.70663211
           9  172.950253  0.54222630  1935.55963  0.70647320
          10  172.054329  0.89592366  2090.61737  0.70620304
          .................................................
         115  151.253296  8.762e-005  1993.55207  0.67916861
         116  151.253212  8.372e-005  1993.54410  0.67917317
         117  151.253132  8.003e-005  1993.53599  0.67917755

        L2 Precision : 151.253   L1 Precision : 1993.54
                  Computer Time : 60 seconds


Nonzeros: U=10440, H=332
```

4. Unweighted Symmetric NNMF:

```
print "Random Starting Values: Unweighted Symmetric";
optn = [ "print"          2 ,      /* ipri */
         "meth"       "sym" ,      /* imet=SYM */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< h1,h2 > = nnmf(c,nf,optn);
print "Matrix H1:", h1;
```

```
                Nonnegative Matrix Factorization
                  Unweighted Symmetric Method

       Iter      L2Crit    DiffCrit      L1Crit     LooCrit

          1  868.427230  8532.84687  3945.99159  0.99672121
          2  412.649759  455.777471  2045.75287  0.99298743
          3  379.393940  33.2558194  1810.72876  0.98837837
          4  362.694298  16.6996424  1812.62306  0.98596909
          5  352.915790  9.77850755  1807.89343  0.99234587
          6  346.251150  6.66464044  1797.18370  0.99520259
          7  341.219668  5.03148201  1784.93224  0.99659180
          8  337.080782  4.13888571  1771.55395  0.99731818
          9  333.293499  3.78728269  1756.40759  0.99771188
         10  329.356222  3.93727759  1738.56264  0.99791365
         ..............................................
        197  242.954295  9.089e-005  1285.06640  0.99611318
        198  242.954207  8.805e-005  1285.06502  0.99611304
        199  242.954121  8.533e-005  1285.06366  0.99611291

       L2 Precision : 242.954   L1 Precision : 1285.06
                 Computer Time : 1 seconds
```

5. Weighted Symmetric NNMF:

   The weighted forms have a third return argument `w`:

```
print "Random Starting Values: Weighted Symmetric";
optn = [ "print"          2 ,      /* ipri */
         "meth"       "sym" ,      /* imet=SYM */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
```

```
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< h1,h2,w > = nnmf(c,nf,optn);
print "Matrix H1:", h1;
print "Matrix W:", w;
```

```
                Nonnegative Matrix Factorization
                   Weighted Symmetric Method

       Iter      L2Crit    DiffCrit      L1Crit     LooCrit

          1  389.272120  19751.1684  1763.68317  0.98961999
          2  367.884194  21.3879258  1798.60266  0.98651441
          3  356.713304  11.1708899  1802.37432  0.99129544
          4  349.673306  7.03999828  1796.46011  0.99449473
          5  344.827954  4.84535222  1788.55868  0.99611463
          6  341.385223  3.44273063  1780.43246  0.99700481
          7  338.918094  2.46712873  1772.79506  0.99753572
          8  337.146123  1.77197103  1766.30473  0.99787680
          9  335.868766  1.27735696  1760.87662  0.99811022
         10  334.939482  0.92928482  1756.16895  0.99827818
         .................................................
        249  242.955097  9.176e-005  1285.12483  0.99611209
        250  242.955008  8.891e-005  1285.12255  0.99611200
        251  242.954922  8.615e-005  1285.12031  0.99611192

        L2 Precision : 242.955   L1 Precision : 1285.12
                  Computer Time : 1 seconds
```

6. Orthogonal Unweighted Symmetric NNMF:

```
print "Random Initial Values: Unweighted Orthogonal Symmetric";
optn = [ "print"          2 ,      /* ipri */
         "meth"     "ort_s" ,      /* imet=ORT_SYM */
         "maxit"       1000 ,      /* maxiter */
         "rest"          40 ,      /* start zeroing */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< h1,h2 > = nnmf(c,nf,optn);
print "Matrix H1:", h1;
print "Nonzeros: H=307";
attrib(h1);
```

```
          Nonnegative Matrix Factorization
        Unweighted Symmetric Orthogonal Method

      Iter      L2Crit    DiffCrit      L1Crit     LooCrit

         1  2259.70056  7141.57354  6628.19604  0.99933535
         2  713.173023  1546.52753  3388.13039  0.99950727
         3  445.978933  267.194090  2099.24934  0.99956075
         4  438.900438  7.07849530  1751.81854  0.99952877
         5  466.977008 -28.0765695  1762.27808  0.99941127
         6  488.871999 -21.8949911  1840.92579  0.99918774
         7  501.688628 -12.8166291  1902.40913  0.99915154
         8  508.428027 -6.73939900  1939.65889  0.99909721
         9  511.728199 -3.30017248  1960.70696  0.99900806
        10  513.214853 -1.48665316  1972.66115  0.99891304
        ..............................................
       138  495.415674  8.990e-005  1953.79059  0.99963167
       139  495.415588  8.677e-005  1953.79061  0.99963166
       140  495.415504  8.378e-005  1953.79063  0.99963166

       L2 Precision : 495.416   L1 Precision : 1953.79
                 Computer Time : 0 seconds
```

The 175×2 matrix $\mathbf{H}$ is rather dense and not even close to orthogonality:

```
Nonzeros: H=307
```

7. Orthogonal Weighted Symmetric NNMF:

```
print "Random Initial Values: Weighted Orthogonal Symmetric";
optn = [ "print"          2 ,      /* ipri */
         "meth"     "ort_s" ,      /* imet=ORT_SYM */
         "maxit"       1000 ,      /* maxiter */
         "rest"          40 ,      /* start zeroing */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< h1,h2,w > = nnmf(c,nf,optn);
print "Matrix H1:", h1;
print "Matrix W:", w;
print "Nonzeros: H=278";
attrib(h1);
```

```
          Nonnegative Matrix Factorization
        Weighted Symmetric Orthogonal Method

Iter      L2Crit    DiffCrit      L1Crit     LooCrit

   1  457.461468  19682.9790  1743.08502  0.99915313
   2  456.365644  1.09582356  1740.41104  0.99909605
   3  454.252450  2.11319401  1735.38873  0.99898037
   4  450.298403  3.95404674  1726.56224  0.99874696
   5  443.313876  6.98452767  1713.12614  0.99828021
   6  432.155746  11.1581294  1697.59346  0.99737617
   7  416.938481  15.2172657  1689.43154  0.99576303
   8  400.140690  16.7977909  1701.58023  0.99328551
   9  385.251922  14.8887676  1730.56309  0.99017607
  10  373.952795  11.2991270  1762.40363  0.98770858
  ..................................................
 318  244.485012  9.689e-005  1293.78206  0.99606003
 319  244.484916  9.547e-005  1293.78059  0.99606013
 320  244.484822  9.407e-005  1293.77914  0.99606023

 L2 Precision : 244.485   L1 Precision : 1293.78
           Computer Time : 2 seconds
```

The 175×2 matrix $\mathbf{H}$ is rather dense but slightly sparser than the unweighted method:

```
Nonzeros: H=278
```

5 New Developments

5.1 Function `affrma`

gof = affrma(data,optn<,ref>)

< gof,dnew > = affrma(data,optn<,ref>)

Purpose: The `affrma` function implements the Bioconductor RMA algorithm by Bolstad et.al (2003) for the normalization of microarray data. This algorithm is an alternative to the `affvsn` and `affarms` methods for obtaining column normalized microarray data. However, other than fitting the parameters of a Maximum Likelihood model, the RMA method applies robust Medianpolish (Tukey, 1977a, p. 179), see also the `mpolish` function in CMAT to the data.

Input: **data** this should be a $N \times n$ matrix of microarray data where the rows correspond to features (genes) and the columns to samples which need to be normalized. The data may obtain a column for an ID variable specifying a strata. The column number of the ID variable must be specified by the `optn` argument.

optn The `optn` argument is 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.

ref not yet implemented.

Option	Second Column	Meaning
`"bgcor"`		perform background correction default is background correction
`"idvar"`	int	column number containing ID variable; default is no ID variable
`"medp"`	int	algorithm for mean or median polish default is =0: median polish, =1: mean polish
`"mpit"`	int	maximum number of median polish iterations; default is 10 for median and 1 for mean polish
`"mpeps"`	real	for terminating median polish; default is 1.e-2
`"nobg"`		do not perform background correction default is background correction
`"nsamp"`		size $n_r \leq N$ of a reduced data set, needed only when N is too large for the computer resources. default is $n_r = 0$
`"pdat"`	int	print input and fitted data sets
`"print"`	int	amount of printed output =0 specifies no printed output, default is 1

Output: **gof** a vector of scalar results, see below for content;

dnew this is the normalized $N \times n$ data set.

Restrictions: 1. The input data must be numeric and cannot contain any string or complex data.

2. Currently the input data must not have any missing values.

Relationships: affarms(), affvsn()

Examples: 1. :

5.2 Function `boruta`

< gof,stat,hist > = boruta(data,modl<,optn<,class<,cwgt> .. >)

Purpose: The `boruta` function implements a method of feature selection very similar to the algorithm of the R package `boruta` by Kursa & Rudnicki (2010). The algorithm is a simple rapper for the `randomForest` R package by A. Liang (2010):

1. For each run of the Random Forest algorithm the data is extended by additional columns with randomly permuted columns of the original data. There are at least five additional such columns added.
2. There are three initial stages in which the scaled Z scores of the variable importance computed by the R version of the random forest algorithm are compared and in which some of the original variables can be "rejected" since their scaled Z scores are in some range of the weighted Z of the added random variables. Only variables with significant larger Z scores are maintained as "tentative". The "rejected" variables are dropped for the successive analysis.
3. During a final stage ofa set of random forest runs with randomly permuted columns to the remaining original variables added, the size of the scaled Z scores decides whether variables are "rejected" or "confirmed" or may still stay "tentative".
4. Only if there are "tentative" variables left, an optional "roughfix" step can be performed which uses the univariate statistics of the Z scores of all or only the final set to decide whether a "tentative" variable should be "rejected" or "confirmed". No additional random forest are here computed.

Every time the `boruta` package calls the R package `randomForest` much overhead is executed including many memory allocations. Since this is done normally many times and could be very time consuming (the authors report a "few hours" computer time for processing the Madelon data set with $N = 2000$ observations and $p = 500$ variables) our C version of the simple algorithm can be much faster, since most of the overhead is done only once.

Input: **data** specifies the data with N rows and a set of n columns which can be defined as effects using the model string argument.

modl specifies which column corresponds to the response and which columns to the p effects of the linear model.

optn 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.

class an optional argument specifying which of the columns of the input data matrix **X** are CLASS variables. The response y must be a class variable and is many times binary.

cwgt only for classification: an optional argument specifying positive weights $w_j > 0$ for the levels (classes) j of the categorical response variable. For regression the argument is neglected.

Option	Second Column	Meaning
`"conf"`	real	significance level of test (default=.999)
`"maxrun"`	int	number
`"rough"`	string	specifies
	"all"	
	"final"	
—	—	following options refer to the random forest calls:
`"mtry"`	int	number of variables randomly selected at each node; default in R: $\max(floor(p/3), 1)$ for regression, $\max(floor(\sqrt{(}p)), 1)$ for classification
`"ndsize"`	int	no node with fewer cases will be split in R: def=5 for Regression, =1 for classification
`"nrnod"`	int	maximum number of nodes per tree def= 2*N+1 for vers. 5; 2*(N/ndsiz)+1 for vers. 4
`"ntree"`	int	number of trees to be produced here: def=10*N, but in [200,500]; in R: def=500
`"seed"`	int	seed for random generator (def=time of day)
`"vers"`	int	should be 0, 4, or 5 (default is 0):
	0	R version for the variable importance
	5	Fortran version 5 for the variable importance
	4	Fortran version 4 for the variable importance

(All int's in this table must not be negative.)

Output: gof This is a vector of scalar results, containing

1. indicates failure in execution
2. total time in seconds
3. total number n_{run} random forest calls
4. number "tentative" variable asignments
5. number "rejected" variable asignments
6. number "confirmed" variable asignments

stat This is a $p \times 6$ matrix with univariate statistics of the variable importance Z scores in its columns:

1. mean across all n_{run} random forest runs,
2. median across all n_{run} random forest runs,
3. minimum across all n_{run} random forest runs,
4. maximum across all n_{run} random forest runs,

5. Normalized (in $[0, 1]$) number of hits
6. coded final decision: 0: tentative, 1: rejected, 2: confirmed

hist this is a $n_{run} \times 4 + p$ matrix which contains some information about the Z scores of each of the n_{run} random forest calls. The first three columns refer to the Z scores of the et of randomly permuted variables which are added in each run. (This must be at least 5 columns.) The columns refer to:

1. the minimum of the Z scores of the randomly permuted columns
2. the mean of the Z scores of the randomly permuted columns
3. the maximum of the Z scores of the randomly permuted columns
4. the number of nonrejected variables at that time of stage
5. the remaining p columns contain the Z scores of the original variables at that stage; missing values indicate a variable which was already rejected earlier.

Restrictions: 1. For version 4 classification and regression the categorical variables cannot have more than 31 levels.

Relationships: ranfor(),varsel(), svm(),lrforw(), scad(),

Examples: 1. Random Forest Regression: Ozone Data, N=203, p=12:

```
   options NOECHO;
   ozon2 = [
#include "..\\tdata\\Ozone0.dat"
            ];
   options ECHO;

   ozon1 = shape(ozon2,.,14);
   ozone = ozon1[,2:14];


   cnam = [" V1:13 "];
   ozone = cname(ozone,cnam);
   print "nrow=",nrow(ozone);
   print "ozone[]=",ozone[1:5,];


   clas = [ 1:3 ];
   modl = "4 = 1:3 5:13";


   optn = [ "conf"        .999 ,
            "maxrun"        20 ,
            "rough"    "final" ,
            "print"          3 ,
            /* ranfor(): */
```

```
        "seed"         123 ,
        "ntree"        200 ,
        "mtry"           4 ,
        "ndsiz"          1 ];
< gof,stat,hist > = boruta(ozone,modl,optn,clas);


   Boruta Feature Selection (Random Forest Regression V=0)

   Number Observations . . . . . .          203
   Number Variables. . . . . .               12
   Number Initial Runs . . . . . .           10
   Number Final Runs . . . . . .             20
   Confidence. . . . . .                 0.999000
   Number of Trees . . . . . .              200
   Number Variables Selected per Node. . . . .  4
   Minimum Node Size . . . . . .             1
   Maximum Node Size . . . . . .           407
   Number Missing Values . . . . . .         0
   Seed for Random Generator . . . . . .   123


                  *****************
                  Model Information
                  *****************

               Number Valid Observations  203
               Response Variable          Y[4]
               N Independend Variables     12


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

Class  Level    Value

C[ 1]     12      1       2       3       4       5
                  6       7       8       9      10
                 11      12
C[ 2]     31      1       2       3       4       5
                  6       7       8       9      10
                 11      12      13      14      15
                 16      17      18      19      20
                 21      22      23      24      25
                 26      27      28      29      30
                 31
```

```
C[ 3]      5        1       2       3       4       5
```

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

                    Variable       Value    Nobs  Proportion
                    C[ 1]              1      17    8.374384
                                       2      17    8.374384
                                       3      21   10.344828
                                       4      21   10.344828
                                       5      10    4.926108
                                       6      17    8.374384
                                       7      13    6.403941
                                       8      15    7.389163
                                       9      16    7.881773
                                      10      18    8.866995
                                      11      17    8.374384
                                      12      21   10.344828
                    ........................................

                    C[ 3]              1      37   18.226601
                                       2      45   22.167488
                                       3      43   21.182266
                                       4      36   17.733990
                                       5      42   20.689655

                     Variable Assignment : Initial Round 1

                              1          V1   Tentative
                              2          V2   Tentative
                              3          V3   Tentative
                              4          V5   Tentative
                              5          V6   Tentative
                              6          V7   Tentative
                              7          V8   Tentative
                              8          V9   Tentative
                              9         V10   Tentative
                             10         V11   Tentative
                             11         V12   Tentative
                             12         V13   Tentative

                   Number Hits for 10 Runs : Initial Round 1
```

	RndVar_1	RndVar_2	RndVar_3	RndVar_4
V1	10.00000000	10.00000000	10.00000000	10.00000000
V2	0.000000000	0.000000000	0.000000000	1.000000000
V3	1.000000000	1.000000000	1.000000000	2.000000000
V5	10.00000000	10.00000000	10.00000000	10.00000000
V6	2.000000000	5.000000000	5.000000000	5.000000000
V7	10.00000000	10.00000000	10.00000000	10.00000000
V8	10.00000000	10.00000000	10.00000000	10.00000000
V9	10.00000000	10.00000000	10.00000000	10.00000000
V10	10.00000000	10.00000000	10.00000000	10.00000000
V11	10.00000000	10.00000000	10.00000000	10.00000000
V12	10.00000000	10.00000000	10.00000000	10.00000000
V13	10.00000000	10.00000000	10.00000000	10.00000000

Number Hits for 10 Runs : Initial Round 1

	RndVar_5
V1	10.00000000
V2	3.000000000
V3	2.000000000
V5	10.00000000
V6	6.000000000
V7	10.00000000
V8	10.00000000
V9	10.00000000
V10	10.00000000
V11	10.00000000
V12	10.00000000
V13	10.00000000

Variable Assignment : Initial Round 2

1	V1	Tentative
2	V2	Rejected
3	V3	Tentative
4	V5	Tentative
5	V6	Tentative
6	V7	Tentative
7	V8	Tentative
8	V9	Tentative
9	V10	Tentative
10	V11	Tentative
11	V12	Tentative
12	V13	Tentative

Number Rejected Variables: 1

Number Hits for 10 Runs : Initial Round 2

	RndVar_1	RndVar_2	RndVar_3	RndVar_4
V1	10.00000000	10.00000000	10.00000000	10.00000000
V2	0.000000000	0.000000000	0.000000000	0.000000000
V3	1.000000000	2.000000000	4.000000000	5.000000000
V5	10.00000000	10.00000000	10.00000000	10.00000000
V6	0.000000000	2.000000000	3.000000000	3.000000000
V7	10.00000000	10.00000000	10.00000000	10.00000000
V8	10.00000000	10.00000000	10.00000000	10.00000000
V9	10.00000000	10.00000000	10.00000000	10.00000000
V10	10.00000000	10.00000000	10.00000000	10.00000000
V11	9.000000000	10.00000000	10.00000000	10.00000000
V12	10.00000000	10.00000000	10.00000000	10.00000000
V13	8.000000000	9.000000000	10.00000000	10.00000000

Number Hits for 10 Runs : Initial Round 2

	RndVar_5
V1	10.00000000
V2	0.000000000
V3	6.000000000
V5	10.00000000
V6	4.000000000
V7	10.00000000
V8	10.00000000
V9	10.00000000
V10	10.00000000
V11	10.00000000
V12	10.00000000
V13	10.00000000

Variable Assignment : Initial Round 3

1	V1	Tentative
2	V2	Rejected
3	V3	Rejected
4	V5	Tentative
5	V6	Rejected
6	V7	Tentative
7	V8	Tentative
8	V9	Tentative

9	V10	Tentative
10	V11	Tentative
11	V12	Tentative
12	V13	Tentative

Number Rejected Variables: 3

Number Hits for 10 Runs : Initial Round 3

	RndVar_1	RndVar_2	RndVar_3	RndVar_4
V1	10.00000000	10.00000000	10.00000000	10.00000000
V2	0.000000000	0.000000000	0.000000000	0.000000000
V3	0.000000000	0.000000000	1.000000000	1.000000000
V5	10.00000000	10.00000000	10.00000000	10.00000000
V6	0.000000000	0.000000000	3.000000000	6.000000000
V7	10.00000000	10.00000000	10.00000000	10.00000000
V8	10.00000000	10.00000000	10.00000000	10.00000000
V9	10.00000000	10.00000000	10.00000000	10.00000000
V10	10.00000000	10.00000000	10.00000000	10.00000000
V11	10.00000000	10.00000000	10.00000000	10.00000000
V12	10.00000000	10.00000000	10.00000000	10.00000000
V13	10.00000000	10.00000000	10.00000000	10.00000000

Number Hits for 10 Runs : Initial Round 3

	RndVar_5
V1	10.00000000
V2	0.000000000
V3	1.000000000
V5	10.00000000
V6	6.000000000
V7	10.00000000
V8	10.00000000
V9	10.00000000
V10	10.00000000
V11	10.00000000
V12	10.00000000
V13	10.00000000

Variable Assignment : Final Round

1	V1	Confirmed
2	V2	Rejected
3	V3	Rejected
4	V5	Tentative

5	V6	Rejected
6	V7	Confirmed
7	V8	Tentative
8	V9	Confirmed
9	V10	Confirmed
10	V11	Confirmed
11	V12	Confirmed
12	V13	Confirmed

Number Rejected Variables: 3
Number Confirmed Variables: 7

Number Hits for 20 Runs : Final Round

	RndVar_1	RndVar_2	RndVar_3	RndVar_4
V1	20.00000000	20.00000000	20.00000000	20.00000000
V2	0.000000000	0.000000000	0.000000000	0.000000000
V3	0.000000000	0.000000000	0.000000000	0.000000000
V5	11.00000000	11.00000000	11.00000000	11.00000000
V6	0.000000000	0.000000000	0.000000000	0.000000000
V7	20.00000000	20.00000000	20.00000000	20.00000000
V8	14.00000000	14.00000000	14.00000000	15.00000000
V9	20.00000000	20.00000000	20.00000000	20.00000000
V10	20.00000000	20.00000000	20.00000000	20.00000000
V11	20.00000000	20.00000000	20.00000000	20.00000000
V12	20.00000000	20.00000000	20.00000000	20.00000000
V13	19.00000000	20.00000000	20.00000000	20.00000000

Number Hits for 20 Runs : Final Round

	RndVar_5
V1	20.00000000
V2	0.000000000
V3	0.000000000
V5	11.00000000
V6	0.000000000
V7	20.00000000
V8	18.00000000
V9	20.00000000
V10	20.00000000
V11	20.00000000
V12	20.00000000
V13	20.00000000

Statistics of Z Values (Variable Importance)

****

Dense Matrix (12 by 6)

	MeanZ	MedianZ	MinZ	MaxZ	NormHits
V1	5.6320452	5.6286652	3.5299774	7.7128760	1.0000000
V2	-0.5495120	-0.3748286	-2.3072629	0.7976992	0.0000000
V3	-0.1135942	-0.3503261	-1.3850808	1.9704112	0.0666667
V5	3.2344043	3.9806524	-1.8339903	6.0288468	0.8200000
V6	0.3869893	0.5281310	-2.1363626	2.4813815	0.0666667
V7	4.2037146	4.3104669	1.9254233	6.2269564	1.0000000
V8	7.0574033	8.2841007	-0.9768839	10.072950	0.8800000
V9	8.9876166	9.6200811	3.6913571	11.857071	1.0000000
V10	5.0609700	4.0642050	2.1771242	10.865764	1.0000000
V11	4.7934198	3.6403402	1.9504716	11.955347	0.9800000
V12	7.0735062	7.2463141	3.8521976	8.7257399	1.0000000
V13	3.6771414	3.5628032	1.3905682	6.5991323	0.9400000

	Decision
V1	2.0000000
V2	1.0000000
V3	1.0000000
V5	0.0000000
V6	1.0000000
V7	2.0000000
V8	0.0000000
V9	2.0000000
V10	2.0000000
V11	2.0000000
V12	2.0000000
V13	2.0000000

Variable Assignment : Roughfix

1	V1	Confirmed
2	V2	Rejected
3	V3	Rejected
4	V5	Confirmed
5	V6	Rejected
6	V7	Confirmed
7	V8	Confirmed
8	V9	Confirmed
9	V10	Confirmed

```
                    10         V11   Confirmed
                    11         V12   Confirmed
                    12         V13   Confirmed
                Number Rejected Variables: 3
                Number Confirmed Variables: 9


                 Total Computation Time: 308

GOF=
              |         1
------------------------
      Failure |   0.00000
    TotalTime |    308.00
    N_RF_runs |    50.000
  N_Tentative |   0.00000
   N_Rejected |    3.0000
  N_Confirmed |    9.0000
       unused |       .
       unused |       .
       unused |       .
       unused |       .
```

2. Random Forest Binary Classification: Madelon Data, N=2000, p=500: Among the 500 variables of this data set there are only 20 important attributes for the prediction. The remaining 480 variables must be rejected. The authors write about "a few hours" computation time.

```
   print "Madelon Data: nobs=2000, nvar=500";
   options NOECHO;
   madey = [
#include "..\\..\\tdata2\\UCI_Repos\\madelon\\mad_y_train.dat"
           ];


   madex = [
#include "..\\..\\tdata2\\UCI_Repos\\madelon\\mad_train.dat"
           ];
   options ECHO;


   madey = replace(madey,-1,0);
   made = shape(madex,.,500);
   made = made -> madey`;
   print "nrow=",nrow(made)," ncol=",ncol(made);
   free madex, madey;
```

```
modl = "501 = 1:500";
clas = 501;


modl = "501 = 1:500";
clas = 501;
optn = [ "conf"          .999 ,
         "maxrun"          20 ,
         "rough"      "final" ,
         "print"            3 ,
         /* ranfor(): */
         "ntree"          200 ,
         "mtry"            12 ,
         "ndsiz"            1 ];
< gof,stat,hist > = boruta(made,modl,optn,clas);
```

In each of the three initial stages we only had 10 Random Forest computations and in the final stage 20:

```
 Boruta Feature Selection (Random Forest Classification V=0)

   Number Observations . . . . . .          2000
   Number Variables. . . . . .               500
   Number Initial Runs . . . . . .            10
   Number Final Runs . . . . . .              20
   Confidence. . . . . .                  0.999000
   Number of Trees . . . . . .               200
   Number Variables Selected per Node. . . .  10
   Minimum Node Size . . . . . .               1
   Maximum Node Size . . . . . .            4001
   Number Missing Values . . . . . .           0
   Seed for Random Generator . . . . . .     123


                  *****************
                  Model Information
                  *****************

               Number Valid Observations 2000
               Response Variable        Y[501]
               N Independend Variables    500


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

```
  Class  Level    Value

Y[501]       2        0       1
```

During the first initial stage already 431 variables are "rejected" and only 69 stay "tentative":

```
              Variable Assignment : Initial Round 1

                    Rejected Variables : 431
  1   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 19  20  21  23  24  25  26  27  28  30  31  32  33  34  36  37
  ..........................................................
465 466 467 468 469 470 472 474 475 477 478 479 480 481 482 483
484 485 486 487 488 489 490 491 492 493 495 497 498 499 500


                    Tentative Variables : 69
  2   3  22  29  35  44  46  47  49  65  66  70 100 106 109 114
126 129 133 137 144 149 154 155 165 174 178 179 181 186 212 214
221 234 237 242 251 256 259 272 282 285 300 304 305 308 318 319
333 335 337 339 350 352 379 389 426 434 439 443 448 452 454 461
471 473 476 494 496
```

During the second initial stage another 25 variables are "rejected" and only 44 stay "tentative":

```
              Variable Assignment : Initial Round 2

                    Rejected Variables : 456
  1   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 19  20  21  22  23  24  25  26  27  28  30  31  32  33  34  36
  ..........................................................
475 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 495 496 497 498 499 500


                    Tentative Variables : 44
  2   3  29  35  44  49  65  66 106 114 129 154 165 174 179 181
212 214 234 242 251 256 272 282 300 304 305 319 333 335 337 339
350 379 426 434 439 443 452 454 461 473 476 494
```

During the third initial stage another 15 variables are "rejected" and only 29 stay "tentative":

```
            Variable Assignment : Initial Round 3

                    Rejected Variables : 471
  1   2   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 18  19  20  21  22  23  24  25  26  27  28  30  31  32  33  34
  ..........................................................
477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492
493 495 496 497 498 499 500


                    Tentative Variables : 29
3  29  49  65  66 106 129 154 165 174 242 282 300 319 333 335
337 339 350 379 434 439 443 452 454 461 473 476 494
```

In the final round another 8 variables are "rejected", 9 variables are being "confirmed" and only 12 stay "tentative":

```
                Variable Assignment : Final Round

                    Confirmed Variables : 9
 49 106 129 165 242 282 379 452 476


                    Rejected Variables : 479
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 17  18  19  20  21  22  23  24  25  26  27  28  30  31  32  33
  ..........................................................
485 486 487 488 489 490 491 492 493 495 496 497 498 499 500


                    Tentative Variables : 12
 29  65 154 319 335 337 339 434 443 454 473 494
```

Based on the statistics of the Z scores a "Roughfix" was applied which changed 11 of the "tentative" variables to "confirmed" and only one to "rejected". The 20 "confirmed" variables are the same correct ones as found with the R package:

```
                Variable Assignment : Roughfix

                    Confirmed Variables : 20
 29  49  65 106 129 154 165 242 282 319 337 339 379 434 443 452
454 473 476 494


                    Rejected Variables : 480
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
```

```
    17  18  19  20  21  22  23  24  25  26  27  28  30  31  32  33
     ................................................................
   484 485 486 487 488 489 490 491 492 493 495 496 497 498 499 500
```

The computation of the 50 Random Forest runs took slightly more than 100 minutes on my ten year old PC:

```
                  Total Computation Time: 6165


GOF=
               |         1
-------------------------
      Failure |   0.00000
    TotalTime |    6165.0
    N_RF_runs |    50.000
  N_Tentative |   0.00000
   N_Rejected |    480.00
  N_Confirmed |    20.000
       unused |       .
       unused |       .
       unused |       .
       unused |       .
```

5.3 Function `gpbatch`

rc = gpbatch(gpfiles)

Purpose: The `gpbatch` function is calling *gnuplot* with one or more input arguments which are the path names of files containing valid *gnuplot* input command scripts for plots. Conventionally those script files should have the extension `.gp`, another (less common) choice would be `.plt`.

The input argument `"-"` refers to an interactive input section. For example, the call

```
gpfiles = const(3,1,.);
gpfiles[1] = "setup.gp";
gpfiles[2] = "-";
gpfiles[3] = "teardown.gp";

rc=grnuplot(gpfiles);
```

is equivalent to the *gnuplot* shell command:
`gnuplot setup.gp - teardown.gp`.

If a *gnuplot* command script needs a data set (which should be an ASCI text data set) it may be generated from a CMAT data object using the `fprintf` function. Be aware, that blank lines in data files are interpreted by *gnuplot* (see Janert, 2009, pp. 30);

Some numerical algorithms related to graphics are available, like the `intpol()`, `quadip()`, and `fft()` functions. There are also a few printer plotting functions available in CMAT.

Input: The `gpfiles` input argument must be either a string scalar or a vector of strings specifying the `gnuplot` input scripts for batch processing. The strings must be valid path names pointing to the `.gp` files with the strings. As with *gnuplot* the `"-"` "file name" refers to an interactive input section.

Output: The only output argument of this function is an integer `rc` which is 0 if there were no problems encountered when executing the comman scripts, and 1 if there were problems. In addition there should be output files written by *gnuplot* containing plots provided by `terminal` and `output` commands in the script file.

Restrictions:
1. The *gnuplot* software must be available in a `gnuplot` directory at the same hierarchy level as the `cmat` directory.
2. For old DOS (not Windows NT) the length of the concatenated strings cannot exceed 128 characters.

Relationships: `gnuplot ...  gpend`

Examples: 1. Show a plot on the screen using the `pause` statement:

```
print "Example 1: Simple Example with pause";
gpbatch("..\\tgplt\\gnuplt.gp");
```

```
plot [] [-2:2] sin(x), x, x-(x**3)/6
pause -1 "Hit return to continue"
```

2. Writing a `.jpg` file using the `jpeg` terminal:

```
print "Simple Example with jpeg terminal";
gpbatch("..\\tgplt\\gp_jpg.gp");
```

```
set terminal jpeg
set o "gp_sin.jpg"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

3. Writing a `.gif` file using the `gif` terminal:

```
print "Simple Example with gif terminal";
gpbatch("..\\tgplt\\gp_gif.gp");
```

```
set t gif
set o "gp_sin.gif"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

4. Writing a `.png` file using the `png` terminal:

```
print "Simple Example with png terminal";
gpbatch("..\\tgplt\\gp_png.gp");
```

```
set t png
set o "gp_sin.png"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

5. Writing a `.svg` file using the `svg` terminal:
Using the `svg` terminal assumes that an SVG viewer is available to the user. For example, the website for Adobe offers a `SVGView.exe` program free of charge.

```
   print "Simple Example with svg terminal";
   gpbatch("..\\tgplt\\gp_svg.gp");
```

```
set t svg
set o "gp_sin.svg"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

6. Writing a `.ps` file using the `postscript` terminal:

```
   print "Simple Example with postscript terminal";
   gpbatch("..\\tgplt\\gp_ps.gp");
```

```
set t postscript "Helvetica" 12
set t postscript color enhanced
set o "gp_sin.ps"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

7. Writing an `.eps` file using the `postscript eps` terminal:

```
   print "Simple Example with eps terminal";
   gpbatch("..\\tgplt\\gp_pseps.gp");
```

```
set t postscript "Helvetica" 12
set t postscript eps color enhanced
set o "gp_sin.eps"
plot [] [-2:2] sin(x), x, x-(x**3)/6
reset
```

Assuming package `graphicx` is included the following Latex commands will show the plot from the `gp_sin.eps` file in this manuscript:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5in]{gp_sin.eps}
\end{center}
\caption{.eps file included into document}
\end{figure}
```

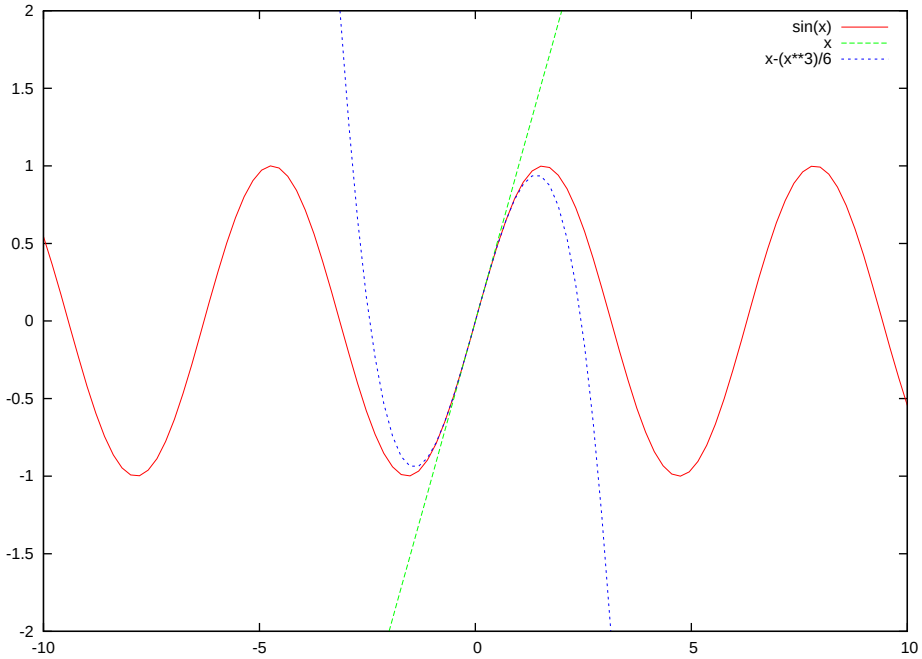

Figure 1: .eps file included into document

8. Submitting more than one input file:
   The following `gpbatch` call is equivalent to a DOS or Linux command line input of `gnuplot ..`

```
tgplt
gp_jpg.gp ..
tgplt
gp_gif.gp ..
tgplt
gp_png.gp
```

```
gpfils = [ "..\\tgplt\\gp_jpg.gp",
           "..\\tgplt\\gp_gif.gp",
           "..\\tgplt\\gp_png.gp",
           "..\\tgplt\\gp_svg.gp"];
gpbatch(gpfils);
```

5.4 Function `help`

help(page)

help(name<,"s">)

Purpose: The `help` function calls *Acrobat Reader* for opening the *CMAT Reference Manual* `cm_ref.pdf` at a specific location. There are three way of opening the document epending on the input argument(s):

- the first input argument is an intger specifying the page number where the *Reference Manual* should be opened;
- the first input argument is a string:
  - if there is a second string argument starting with "s", then *Acrobat Reader* is started with a search for the string in the first argument;
  - otherwise *Acrobat Reader* is started at a bookmark with name specified by string in the first argument.

The computation with CMAT is continued after closing the *Acrobat Reader* application.

Input: The first input argument can be

- an integer referring to a page number,
- or a string specifying a bookmark or search term.

A second argument is only valid when the first argument is specified as a string and specifies that a search is wanted.

Output: There is no output argument.

Restrictions:
1. The *Acrobat Reader* software must be available in a `Reader` directory at the same hierarchy level as the `cmat` directory.
2. The string input argument must agree with a (bookmark) term inside the *CMAT Reference Manual*.

Relationships: spawn(),

Examples:

```
help("scad");
/* first close the Acrobat Reader application */
help("inv");
/* second close the Acrobat Reader application */
```

5.5 Function `maxn`

< val,ind > = maxn(a,n)

Purpose: The `maxn` function finds the $n \geq 1$ largest values `val` of object `a` and its index locations `ind`. Assuming `a` has m dimensions where $m = 2$ for vectors and matrices.

Input: **a** must be an input object (scalar, vector, matrix, or tensor);

n must be a positive integer specifying the number of values.

Output: **val** a vector of the n largest values of `a`

loc a $n \times m$ matrix of index locations.

Restrictions: 1. If the object `a` has less than n nonmissing numeric values, missing values are returned.

Relationships: max()min() minn()

Examples: 1. Data is sparse vector:

```
a = [ 0 6 0 0 0 0 5 ];
< val,ind > = maxn(a,3);
print "Val=",val;
print "Ind=",ind;
```

```
Val=
   |   1
--------
 1 |   6
 2 |   5
 3 |   0
```

```
Ind=
   |   1   2
------------
 1 |   1   2
 2 |   1   7
 3 |   1   4
```

2. Data is sparse tensor:

```
print "Data are sparse Tensors";
int at[4,4,4];
at[1,2,3] = -4;
at[2,1,4] =  3;
at[3,3,2] = -2;
at[4,4,2] =  1;
print "At=", at;


< val,ind > = maxn(at,3);
print "Val=",val;
print "Ind=",ind;
```

Val=

	1
1	3
2	1
3	0

Ind=

	1	2	3
1	2	1	4
2	4	4	2
3	1	1	3

5.6 Function `mdy`

days = mdy(m,d,y<,sopt>)

Purpose: The `mdy` function finds the number of days starting from January 1 of the year 0 (or optional from Januar 1 of the year 1900) and a specified date. This function is mostly used to find the number of days between two specified dates, using the difference of two function calls.

Input: **m** must be the two digit month specification, $1 \leq m \leq 12$

d must be the two digit day specification, $1 \leq d \leq 31$

y must be either the two or four digit year specification, $y \geq 0$

sopt is an optional string option

"year2" two digits year input assuming 19th century date (def.)

"y1900" as "year2" but subtracts 1900 * 365 days

"y1960" as "year2" but subtracts 1960 * 365 days

"dm2y2" as "year2" but the inputs of day and month are swapped

"year4" four digit year input

"dm2y4" as "year4" but the inputs of day and month are swapped

Output: If not options `"y1900"` or `"y1960"` are specified, it returns the number of days starting from January 1 of the year 0. If option `"y1900"` is specified, it returns the number of days since January 1, 1900. If option `"y1960"` is specified, it returns the number of days since January 1, 1960. (This is similar to a function in SAS.)

Restrictions: 1. The first three arguments must be integer, which must be in valid ranges and must not be missing.

Relationships: date()

Examples:

```
day1 = mdy(01,10,37);
day2 = mdy(01,20,37);
print "day1,day2,dis=",day1,day2,day2-day1;

day1,day2,dis= 707015 707025 10


day3 = mdy(01,10,37,"y1900");
day4 = mdy(10,01,37,"dm2y2");
dif = day3 - day4 + 1900 * 365;
print "day3,day4,dif=",day3,day4,dif;


day3,day4,dif= 13515 707015 0
```

```
   day5 = mdy(01,10,1937,"year4");
   day6 = mdy(10,01,1937,"dm2y4");
   print "day5,day6,dif=",day5,day6,day5-day6;


day5,day6,dif= 707015 707015 0
```

5.7 Function `minn`

< val,ind > = minn(a,n)

Purpose: The `minn` function finds the $n \geq 1$ smallest values `val` of object `a` and its index locations `ind`. Assuming `a` has m dimensions where $m = 2$ for vectors and matrices.

Input: **a** must be an input object (scalar, vector, matrix, or tensor);

n must be a positive integer specifying the number of values.

Output: **val** a vector of the n smallest values of `a`

loc a $n \times m$ matrix of index locations.

Restrictions: 1. If the object `a` has less than n nonmissing numeric values, missing values are returned.

Relationships: min()max() minn()

Examples: 1. Data is lower triangular matrix:

```
a = [ 3 ,
     -4  5 ,
      5 -6 -7 ];


< val,ind > = minn(a,3);
print "Val=",val;
print "Ind=",ind;
```

```
Val=
   |   1
--------
 1 |  -7
 2 |  -6
 3 |  -4
```

```
Ind=
   |   1   2
------------
 1 |   3   3
 2 |   3   2
 3 |   2   1
```

2. Data is sparse matrix:

```
a = [ 1 0 0 2,
      4 0 0 3,
      5 0 0 0 ];


< val,ind > = minn(a,3);
print "Val=",val;
print "Ind=",ind;
```

```
Val=                                    Ind=
 Z |   1                                   |   1   2
--------                                ------------
 1 |   0                                 1 |   1   2
 2 |   0                                 2 |   1   3
 3 |   0                                 3 |   2   2
```

```
print "Data is sparse Tensor";
int at[4,4,4];
at[1,2,3] = -4;
at[2,1,4] =  3;
at[3,3,2] = -2;
at[4,4,2] =  1;
print "At=", at;


< val,ind > = minn(at,3);
print "Val=",val;
print "Ind=",ind;
```

```
Val=                                    Ind=
   |   1                                   |   1   2   3
--------                                ----------------
 1 |  -4                                 1 |   1   2   3
 2 |  -2                                 2 |   3   3   2
 3 |   0                                 3 |   1   1   3
```

5.8 Function `propurs`

< gof,ap,xpa > = propurs(data,nsol<,optn<,wgt>>)

Purpose: The `propurs` function is a new implementation of the *Projection Pursuit* algorithm by Friedman and Tukey (1974). A number of `nsol` twodimensional configurations is fitted from the PCA scores of the data. The fit is stagewise and the result for $n-1$ solutions is always contained as first ones in the result of the n solutiuons.

Input: data This argument must be a $N \times n$ numeric matrix. Neither string data or missing values are currently permitted.

nsol The number of two dimensional configurations which should be fitted.

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.

wgt

Option	Second Column	Meaning
`"fei"`	real	threshold for search space dimensionality, proportion of largest eigenvalue (must be in $(0,1)$, def=1.e-4)
`"maxi"`	int	maximum number of iterations, def=10
`"ndeg"`	int	degree of Legendre polynomial to be fitted must be at least 2 and at most 8, def=2
`"nei"`	int	maximum dimensionality of search space, maximum number of eigenvalues (must be in $[2,n]$, def=n)
`"print"`	int	indicates the amount of printed output (=0: no printed output, default is 0)
`"tol"`	real	termination tolerance for iteration (default is 0.01)
`"trim"`	real	trimming tolerance for or deleting a portion of outlying observations with high PCA scores (must be in $[0,1]$, default is 0.0)

Output: gof a vector of scalar information

ap contains $n \times$ `nsol` $*$ 2 matrix of `nsol` twodimensional configurations of the n columns (variables, features).

xpa contains $N \times$ `nsol` $*$ 2 matrix of `nsol` twodimensional configurations of the N rows (observations, items).

Restrictions: 1. The input data matrix should have no strings and no missing values.

2. The input data and the input weights should have the same number of rows (observations).

Relationships: pca(), mds(),

Examples: 1. Fit of the `sun1.dat` 100×6 data set for 1,2, and 3 configurations.

```
   options NOECHO;
   sun = [
#include "..\\tdata\\sun1.dat"
         ];
   options ECHO;
   sun = shape(sun,.,6);
   print "nr=",nrow(sun)," nc=",ncol(sun);


   optn = [ "ndeg"          4 ,
            "nei"           6 ,
            "fei"       1.e-4 ,
            "tol"       1.e-2 ,
            "trim"         0. ,
            "print"         3 ];


   nsol = 1;
   < gof,ap,xpa > = propurs(sun,nsol,optn);


    Projection Pursuit PCA for 1 Two Dimensional Configurations
                Degree of Legendre Polynomial: 4


Eigenvalues of Covariance Matrix (6 Selected)
---------------------------------------------

  1 :       18.73      8.744      2.661      1.331     0.5677
  6 :     0.06288

Cumulative Fraction of Eigenvalues
----------------------------------

  1 :      0.5835      0.856     0.9389     0.9804      0.998
  6 :           1


                  Dimension of search space: 6
  Pursuit Plane 1: After 8 Iterations Projection Index=0.111901
```

```
                Column Coordinates of 1 Solutions
                *********************************

                     Dense Matrix (6 by 2)

                       |       DIM_1      DIM_2
                      -------------------------
                 COL_1 |  -0.0398801 -0.5121022
                 COL_2 |   0.1964163 -0.4335036
                 COL_3 |   0.3185655 -0.4137896
                 COL_4 |   0.3406905 -0.4355579
                 COL_5 |  -0.8494883 -0.3844310
                 COL_6 |  -0.1436868 -0.2027468


                 Row Coordinates of 1 Solutions
                 ******************************

                    Dense Matrix (100 by 2)

                      |       DIM_1      DIM_2
                     -------------------------
                ROW_001 |  -2.9228813 -0.2365380
                ROW_002 |   2.7861978 -0.1014369
                ROW_003 |  -0.1479158 -0.2390063
                ROW_004 |   4.0952568 -0.2621754
                ROW_005 |  -2.8126590 -0.1823139
                ..............................
                ROW_095 |   1.3454200  0.5301987
                ROW_096 |   3.1442471 -0.1576579
                ROW_097 |  -1.9075112 -0.4069278
                ROW_098 |   3.4372284 -0.0624715
                ROW_099 |  -2.4647613 -0.1412313
                ROW_100 |   4.4002469  0.0358836


nsol = 2;
< gof,ap,xpa > = propurs(sun,nsol,optn);


 Projection Pursuit PCA for 2 Two Dimensional Configurations
             Degree of Legendre Polynomial: 4
```

The eigenvalue distribution is the same as above.

```
               Dimension of search space: 6
Pursuit Plane 1: After 8 Iterations Projection Index=0.111901
```

```
Pursuit Plane 2: After 4 Iterations Projection Index=0.0511867

             Column Coordinates of 2 Solutions
             *********************************

                  Dense Matrix (6 by 4)

          |     SOL_1_1    SOL_1_2    SOL_2_1    SOL_2_2
        ------------------------------------------------
    COL_1 |  -0.0398801 -0.5121022  0.4435321 -0.4745084
    COL_2 |   0.1964163 -0.4335036  0.2508022 -0.4831991
    COL_3 |   0.3185655 -0.4137896  0.1179745 -0.4894119
    COL_4 |   0.3406905 -0.4355579 -0.4216186 -0.4017203
    COL_5 |  -0.8494883 -0.3844310 -0.5571900 -0.3317061
    COL_6 |  -0.1436868 -0.2027468 -0.4880949 -0.1744372


nsol = 3;
< gof,ap,xpa > = propurs(sun,nsol,optn);


 Projection Pursuit PCA for 3 Two Dimensional Configurations
             Degree of Legendre Polynomial: 4


               Dimension of search space: 6
Pursuit Plane 1: After 8 Iterations Projection Index=0.111901
Pursuit Plane 2: After 4 Iterations Projection Index=0.0511867
Pursuit Plane 3: After 7 Iterations Projection Index=0.426709


             Column Coordinates of 3 Solutions
             *********************************

                  Dense Matrix (6 by 6)

      |     SOL_1_1    SOL_1_2    SOL_2_1    SOL_2_2    SOL_3_1
    ------------------------------------------------------------
COL_1 |  -0.0398801 -0.5121022  0.4435321 -0.4745084  0.1736464
COL_2 |   0.1964163 -0.4335036  0.2508022 -0.4831991  0.3812713
COL_3 |   0.3185655 -0.4137896  0.1179745 -0.4894119  0.5179395
COL_4 |   0.3406905 -0.4355579 -0.4216186 -0.4017203  0.7070876
COL_5 |  -0.8494883 -0.3844310 -0.5571900 -0.3317061 -0.1250592
COL_6 |  -0.1436868 -0.2027468 -0.4880949 -0.1744372  0.2015074


      |     SOL_3_2
```

```
    --------------
COL_1 |   0.6228442
COL_2 |   0.4758429
COL_3 |   0.3666939
COL_4 |  -0.1718889
COL_5 |  -0.3156187
COL_6 |  -0.3493040
```

5.9 Function `ranfor`

< gof,misc,pred,gini,prox,outl,intact,tmis,tprd,tgini > = ranfor(data,modl<,...>)

gof = ranfor(data,modl<,optn<,class<,cwgt<,test> . >)

Purpose: The `ranfor` function is a new implementation of the Random Forest algorithm by Breiman (2001). It is slightly different from that Fortran code by Breiman & Cutler, which is available on the internet. Currently, only *Random Forest* regression and classification are implemented. For classification the response y must be categorical with $K > 1$ levels, for regression the response is interval scaled. Unsupervised modeling maybe added later. This algorithm is different from many others in data mining. Especially the training result (error, classification table, predicted values etc.) relates to the so-called out-of-bag observations, i.e. observations which were **not** used with the bootstrap samples in the training. If the errors, classification table, and predicted values of all observations of the training set are needed, you should input the same data as the `test` argument or use the `rafprd()` function afterward with the same data.

The return argument seven, which is by default the $p \times p$ matrix `intact` of variable interaction indicators can also return some other object specified by the `"ret7"` option:

Option	Returned Object
"inter"	$p \times p$ interactions of variables (this is default)
"xfull"	$N \times p$ matrix of predictors with imputed values
"imput"	$nmis \times 3$ matrix of imputed values
"mdsca"	$N \times dimsc$ matrix of scaling coordinates
"proto"	prototypes (similar to Fortran version 5)
"parco"	parallel coordinates (similar to Fortran version 4)

Due to the model specification the p predictor effects must not necessarily correspond to columns of the input data matrix. Therefore, the returned imputed values, correspond to the analyzed p effects and not necessarily to the columns of the input data set.

Iterative refinement of the imputed values is computationally expensive and only done if the `"imput"` option is specified. Otherwise missing values are set to the following starting values:

- for numerical variables (interval scaled): set to the median of the nonmissing values,
- for categorical variable (nominal scaled): set to the most frequently used category.

The results of the `random forest` function correspond to different versions of the algorithm:

- R version of the algorithm by A. Liaw (2009); (except for computing the variable importance measurements and the proximity matrix the algorithm is the same as that of version 5).
- Fortran version 5 of the algorithm by Breiman and Cutler
- Fortran version 4 of the algorithm by Breiman and Cutler

For classification the function invokes the version 5 algorithm by default, however permits three different ways of computing and storing the large $N \times N$ proximity matrix

1. computing the large $N \times N$ matrix and storing incore in compact form (similar to version 4 of the Fortran program by Breiman and Cutler, and the R package by A. Liaw);
2. depending on the `"memthr"` option: computing the large $N \times N$ matrix and storing on utility file; after tree development the number of zeros (sparsity percentage) is known and if it compares positive to the `"memthr"` option, the sparse matrix is stored incore. If that is not possible and the matrix must remain on file, some of the operations, e.g. scaling coordinates, prototypes, and outlier detection may need more computation time and the proximity matrix is not returned.
3. computing a $N \times nrnn$ matrix of the $nrnn$ nearest of each observation $i = 1, \ldots, N$ and storing incore (similar to version 5 of Breiman and Cutler). (It looks like this approximation has some problems. It is also not used in the *randomForest* package of R by A. Liaw (2009).)

By default variable importance is computed like in the R package (Liaw, 2009). By specifying the `"vers" = 5` or `"vers" = 4` option similar results to those of the Breiman and Cutler Fortran program versions 5 and 4 can be obtained.

Input: **data** specifies the data with N rows and a set of n columns which can be defined as effects using the model string argument.

modl specifies which column corresponds to the response and which columns to the p effects of the linear model.

optn 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.

class an optional argument specifying which of the columns of the input data matrix $\mathbf{X}$ are CLASS variables. The response y must be a class variable and is many times binary.

cwgt only for classification: an optional argument specifying positive weights $w_j > 0$ for the levels (classes) j of the categorical response variable. For regression the argument must be missing.

test an optional $Ntst \times n$ test data set which is only evaluated (scored).

Option	Second Column	Meaning
`"dimsc"`	int	dimension d for Torgerson scaling (def=0) (needs proximity matrix)
`"fillit"`	int	number of outer iterations for the imputation of missing values, only valid with `"imput"` (def=1: if there are no missing values, def=5, if there are missing values)
`"impn"`		compute casewise variable importance (for plotting)
`"impo"`		compute importance for p variables
`"imput"`		perform iterative imputation algorithm (is costly, needs proximity matrix)
`"intact"`		compute $p \times p$ variable interactions
`"memthr"`	int	bound for switching incore storage of proximity to file (def=100000)
`"mtry"`	int	number of variables randomly selected at each node; default in R: $\max(floor(p/3), 1)$ for regression, $\max(floor(\sqrt{(}p)), 1)$ for classification
`"ncoor"`	int	number of cases with highest votes are returned (def=0); only for classification (this is Fortran version 4 feature)
`"ndsize"`	int	no node with fewer cases will be split (def=1)
`"nearn"`	int	number nearest neighbors of each observation: for computation of prototypes and for number of columns of rectangular (version 5) proximity matrix (def=0)
`"nprot"`	int	maximum number of prototypes to compute (def=0) only for calssification (needs proximity matrix)
`"nrnod"`	int	maximum number of nodes per tree; def= 2*N+1 for vers. 5; 2*(N/ndsiz)+1 for vers. 4
`"ntree"`	int	number of trees to be produced (def=10*N, but in [200,1000])
`"outl"`		detect outlier observations (needs proximity matrix)
`"outmod"`	string	path for model output file (binary) necessary if model is used later with function `rafprd()`

Option	Second Column	Meaning
"pcoor"		print tables with `ncoor` observations (this is Fortran version 4 feature)
"phist"	int	output of the tree history (def=1)
"pinit"	int	print information about input data
"pmis"		print $K \times K$ misclassification table
"pobs"		print $N \times (K + 3)$ predicted value table also prints N vector of outlier indicators
"pprox"		print large proximity matrix
"print"	int	amount of printed output (def=2)
"prox"		compute large $N \times N$ proximity matrix; needed for iterative imputation, outliers, scaling
"prox"	"rect"	compute rectangular (version 5) $N \times nearn$ proximity matrix (approximation)
"prox"	"file"	store large $N \times N$ proximity matrix on file; see also "memthr" option
"ret7"	string	what is returned in argument 7:
	"inter"	interactions of variables (this is default)
	"xfull"	$N \times p$ matrix of predictors with imputed values
	"imput"	$nmis \times 3$ matrix of imputed values
	"proto"	prototypes ("nprot" must be ¿ 0)
	"mdsca"	scaling coordinates ("dimsc" must be ¿ 0)
	"parco"	parallel coordinates ("ncoor" must be ¿ 0)
"seed"	int	seed for random generator (def=time of day)
"vers"	int	should be 0, 4, or 5 (default is 0):
	0	R version for variable importance
	5	Fortran version 5 for variable importance, proximities
	4	Fortran version 4

(All int's in this table must not be negative.) We think that there may be problems with the "nearn" specification of the proximity matrix and the prototypes as they are computed like in version 5 of the Breiman and Cutler code.

Output: **gof** This is a vector of scalar results, containing

1. indicates failure in execution
2. total time in seconds
3. classification error for training
4. classification error for test data (if there is a test data set specified)
5. sparsity percentage of proximity matrix (if computed)
6. number of outliers (with outlyingness ≥ 10) (if computed)

misc $K \times K$ classification table for the training;

pred $N \times 3$ matrix for regression and for classification $Ntst \times (K+3)$ matrix containing in its columns:

1. input values of the response y
2. predicted values of training $\hat{y}$
3. number of times drawn in subsample
4. K columns with vote counts (for classification only)

gini $p \times 3 or 1$ matrix of variable importance; first column has Gini coefficients, second column has zscore (version 5) or average lowering of margin (version 4) when variable j is randomly permuted, third column has significance (version 5) or error (version 4)

prox $N \times N$ dense or sparse or rectangular $N \times nearn$ matrix of proximities this matrix (is symmetric and diagonal dominant, with 1's in the diagonal) and therefore positive semidefinite, i.e. $1 - prox$ can be used as "distance" matrix for the `mds()` multidimensional scaling function;

outl N vector of values indicating outlier observations; The printed output identifies an outlier when the outlyingness is at least 10.

intact by default the $p \times p$ matrix of pairwise variable interactions, but could be also one of the those depending on the `"ret7"` option:

- "ret7"="xfull": $N \times p$ matrix of predictors of the p effects for the final set of trees where missing values are replaced.
- "ret7"="imput": $nmis \times 3$ matrix of missing value imputations, where the first column corresponds to the observation, the second column to the effect number, and the third column to the imputed value of the $N \times p$ matrix $\mathbf{X}$.
- "ret7"="mdsca": $N \times dimsc$ matrix of point coordinates computed as the $dimsc$ principal components of the $1-prox$ distance matrix; CMAT permits you to run better versions of multidimensional scaling algorithms on such a distance matrix based on the output proximities.
- "ret7"="proto": only for classification: matrix of prototypes (assumes version 5 with nonzero `"nearn"` specification (may have problems).
- "ret7"="parco": only for classification: $N \times (3 + Ncoor)$ matrix cases with highest votes (this is called "parallel coordinates" in version 4 and is similar to prototypes in version 5).

tmis only for test data: $K \times K$ classification table for the test data

tprd only for test data: $N \times 3$ matrix for regression and for classification $Ntst \times (K+3)$ matrix containing in its columns:

1. input values of the response y_t
2. predicted values of training $\hat{y_t}$
3. number of trees

4. K columns with vote counts (for classification only)

tgini only for test data: the vector of Gini coefficients.

Restrictions: 1. Observations with missing values for the response are skipped on input.

2. The column-effect mapping (model and class specifications) of training and test data must be compatible.
3. For version 4 classification and regression the categorical variables cannot have more than 31 levels.
4. The proximity matrix may not be returned if the number of nonzeros is larger than the "memthr" option specifies.

Relationships: rafprd(), split(), boruta(), reg(), nlreg()

Examples: 1. Hepatitis Data: N=155, p=19: With 13 predictor class variables:

```
print "Hepatitis data: Nobs=155, nvar=19, Binary Y in col 1";
print "Y: die=32, live=123; var2: ordinal, var3:14 binary, var=15:20 real";

options NOECHO;
hep = [
#include "..\\tdata\\hepatitis.dat"
        ];
options ECHO;


clas = [ 1 3:14 20 ];
modl = "1 = 2:20";
cwgt = [ 3. 1. ];
```

The input data contain 167 missing values. When "imput" is not specified the imputed values are not iterated and stay fixed at the starting values:

- for numerical variables (interval scaled) at the median of the nonmissing values
- for categorical variable (nominal scaled) at the most frequently used category.

```
optn = [ "ntree"          100 ,
         "outmod"   "hepat.rf",
         "mtry"             4 ,
         "ndsiz"            1 ,
         "impo"               ,  /* variable importance */
         "impn"               ,  /* casewise var importance */
```

```
          "outl"               ,  /* look for outliers */
          "prox"               ,  /* compute proximities */
          "intact"             ,  /* compute var interactions */
          "dimsc"            2 ,  /* dimension for MDS */
          "nprot"            2 ,  /* compute prototypes */
          "print"            3 ,
          "pobs"               ,
          "phist"            1 ,
          "pinit"            3 ];
  < gof,misc,pred,gini,prox,outl,intact >
                   = ranfor(hep,modl,optn,clas,cwgt,hep);


              Random Forest Classification (V=0)

    Number Observations . . . . . .          155
    Number Observations in Test Set . . . . .          155
    Number Variables. . . . . .           19
    Number of Trees . . . . . .          100
    Number Variables Selected per Node. . . . .            4
    Minimum Node Size . . . . . .            1
    Maximum Node Size . . . . . .          311
    Method Proximity Matrix . . . . . .            1
    Number Missing Values . . . . . .          167
    Percentage of Missing Values. . . . . .         5.39
    Seed for Random Generator . . . . . .        32873


                    *****************
                    Model Information
                    *****************

                Number Valid Observations  155
                Observations Test Data     155
                Response Variable          Y[1]
                N Independend Variables     19


                     *************
                     Model Effects
                     *************

X2  + C3  + C4  + C5  + C6  + C7  + C8  + C9  + C10 + C11 + C12 +
          C13 + C14 + X15 + X16 + X17 + X18 + X19 + C20


                    ***********************
```

Class Level Information

Class	Level	Value	
Y[1]	2	1 2	
C[3]	2	1 2	
C[4]	3	1 2	_Missing
C[5]	2	1 2	
C[6]	3	1 2	_Missing
C[7]	3	1 2	_Missing
C[8]	3	1 2	_Missing
C[9]	3	1 2	_Missing
C[10]	3	1 2	_Missing
C[11]	3	1 2	_Missing
C[12]	3	1 2	_Missing
C[13]	3	1 2	_Missing
C[14]	3	1 2	_Missing
C[20]	2	1 2	

Simple Statistics

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
X[2]	155	41.200000	12.565878	0.3652940	-0.1062706
X[15]	149	1.4275168	1.2121490	2.8919125	10.237924
X[16]	126	105.32540	51.508109	1.3553074	2.0023636
X[17]	151	85.894040	89.650890	3.1785664	14.017483
X[18]	139	3.8172662	0.6515231	-0.1219955	1.3117814
X[19]	88	61.852273	22.875244	0.0223824	-0.5349362

The following table shows how many missing values each of the effects has:

Number of Observations for Class Levels

Variable	Value	Nobs	Proportion
Y[1]	1	32	20.645161
	2	123	79.354839
C[3]	1	139	89.677419
	2	16	10.322581

C[4]	1	76	49.032258
	2	78	50.322581
	_Missing	1	0.645161
C[5]	1	24	15.483871
	2	131	84.516129
C[6]	1	100	64.516129
	2	54	34.838710
	_Missing	1	0.645161
C[7]	1	61	39.354839
	2	93	60.000000
	_Missing	1	0.645161
C[8]	1	32	20.645161
	2	122	78.709677
	_Missing	1	0.645161
C[9]	1	25	16.129032
	2	120	77.419355
	_Missing	10	6.451613
C[10]	1	60	38.709677
	2	84	54.193548
	_Missing	11	7.096774
C[11]	1	30	19.354839
	2	120	77.419355
	_Missing	5	3.225806
C[12]	1	51	32.903226
	2	99	63.870968
	_Missing	5	3.225806
C[13]	1	20	12.903226
	2	130	83.870968
	_Missing	5	3.225806
C[14]	1	18	11.612903
	2	132	85.161290
	_Missing	5	3.225806
C[20]	1	85	54.838710
	2	70	45.161290

	Tree	Nodes	ClassError	Misc_1	Misc_2
Train	1	39	58.064516	21.875000	67.479675
Test			7.7419355	21.875000	4.0650407
Train	2	57	44.516129	31.250000	47.967480
Test			6.4516129	3.1250000	7.3170732
Train	3	33	40.645161	46.875000	39.024390
Test			12.258065	3.1250000	14.634146
Train	4	59	38.064516	43.750000	36.585366
Test			14.838710	0.0000000	18.699187
Train	5	45	32.258065	46.875000	28.455285
Test			5.1612903	3.1250000	5.6910569
.....					
Train	95	51	20.000000	15.625000	21.138211
Test			0.6451613	0.0000000	0.8130081
Train	96	51	20.000000	15.625000	21.138211
Test			0.6451613	0.0000000	0.8130081
Train	97	37	20.645161	15.625000	21.951220
Test			0.6451613	0.0000000	0.8130081
Train	98	49	20.645161	15.625000	21.951220
Test			0.6451613	0.0000000	0.8130081
Train	99	45	20.645161	15.625000	21.951220
Test			0.6451613	0.0000000	0.8130081
Train	100	61	20.645161	15.625000	21.951220
Test			0.6451613	0.0000000	0.8130081

The first column of the following table shows the starting values of the missing value imputation, which is here fixed since the `"imput"` option is not used:

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	39.00000000	1.354533377	0.003947246	0.027581804
2	0.000000000	0.197551464	0.003807211	0.010172265
3	1.000000000	0.077758181	0.001840026	0.008504854
4	1.000000000	0.038283187	0.000768371	0.005676922
5	0.000000000	0.712647911	0.006237628	0.021950620
6	1.000000000	1.100281205	0.008832703	0.027052898
7	1.000000000	0.124665044	0.002924928	0.016809186
8	1.000000000	0.068013539	-0.000338397	0.010540521
9	1.000000000	0.117056910	0.003527812	0.017403246

```
10  1.000000000  0.457485256  0.005325328  0.016942416
11  1.000000000  1.896921626  0.026367327  0.038062139
12  1.000000000  2.521820019  0.012748113  0.021263475
13  1.000000000  0.348153310  0.004985701  0.011894536
14  1.000000000  3.315629140  0.016150243  0.035014004
15 85.00000000  0.732539287  0.004746713  0.022774604
16 58.00000000  0.814251047 -0.001319800  0.019778780
17  4.000000000  2.879748305  0.018875890  0.033149039
18 60.00000000  1.470594380  0.007276054  0.020626935
19  0.000000000  0.772066811  0.006468555  0.024246521
```

Gini Coefficients (Test)

	1
X2	1.354533377
C3	0.197551464
C4	0.077758181
C5	0.038283187
C6	0.712647911
C7	1.100281205
C8	0.124665044
C9	0.068013539
C10	0.117056910
C11	0.457485256
C12	1.896921626
C13	2.521820019
C14	0.348153310
X15	3.315629140
X16	0.732539287
X17	0.814251047
X18	2.879748305
X19	1.470594380
C20	0.772066811

Note, the misclassification of the training relates to the out-of-box observations:

Classification Rate=20.6452 (Training)

	Y_1	Y_2
1	15.62500000	21.95121951

Final Error: 20.6452 (Training)

	YData_1	YData_2
YPred_1	27	27
YPred_2	5	96

The error is smaller when the model is applied to the entire test data:

Classification Rate=0.645161 (Test)

	Y_1	Y_2
1	0.000000000	0.813008130

Final Error: 0.645161 (Test)

	YData_1	YData_2
YPred_1	32	1
YPred_2	0	122

Predicted Values and Votes (Training)

Obs	Data	Pred	Nout	Vot_1	Vot_2
1	1	1	37	0.0000000	1.0000000
2	1	1	38	0.1578947	0.9473684
3	1	1	31	0.1935484	0.9354839
4	1	1	44	0.0000000	1.0000000
5	1	1	41	0.0000000	1.0000000
.....					
151	0	0	26	1.5000000	0.5000000
152	1	1	47	0.0638298	0.9787234
153	1	0	37	0.8918919	0.7027027
154	1	0	28	1.2857143	0.5714286
155	0	0	31	1.2580645	0.5806452

Predicted Values and Votes (Test)

Obs	Data	Pred	Nout	Vot_1	Vot_2
1	1	1	100	0.0000000	1.0000000
2	1	1	100	0.0600000	0.9800000
3	1	1	100	0.0600000	0.9800000
4	1	1	100	0.0000000	1.0000000
5	1	1	100	0.0000000	1.0000000
.....					
151	0	0	100	2.6100000	0.1300000
152	1	1	100	0.0300000	0.9900000

```
153     1     1   100  0.3300000  0.8900000
154     1     1   100  0.3600000  0.8800000
155     0     0   100  2.4600000  0.1800000
```

The proximity matrix is almost dense:

```
          Largest (Offdiagonal) Proximity Values

           Row  Column        Value
            71      66  1.000000000
            80      71  0.990000000
            80      66  0.990000000
            55      35  0.940000000
            45       5  0.910000000
            71      58  0.910000000
            80      58  0.910000000
            66      58  0.910000000
            45      43  0.910000000
            80       8  0.900000000
Sparsity of Proximity Matrix:  13.2010 % (10494 Nonzeros)
```

For space saving reason we drop here the output of the prototypes and report the following outlier observations:

```
                Table of N=8 Outliers

                 40  20.0000000 ***
                 63  14.8217796 ***
                 98  15.7020043 ***
                104  19.5131131 ***
                122  18.3793716 ***
                135  20.0000000 ***
                141  14.2400897 ***
                154  15.6078766 ***


              Pairwise Variable Interactions
              ******************************

            Dense Symmetric Matrix (19 by 19)

 S |          X2         C3         C4         C5         C6
 ---------------------------------------------------------------
X2 |   0.0000000
C3 |  -48.000000  0.0000000
C4 |  -18.000000 -49.000000  0.0000000
```

C5	-9.0000000	2.0000000	7.0000000	0.0000000	
C6	20.000000	-28.000000	-13.000000	-37.000000	0.0000000
C7	-31.000000	-29.000000	-1.0000000	14.000000	19.000000
C8	-20.000000	-14.000000	4.0000000	1.0000000	5.0000000
C9	-31.000000	3.0000000	-5.0000000	-13.000000	4.0000000
C10	5.0000000	28.000000	0.0000000	21.000000	-11.000000
C11	-31.000000	-21.000000	-13.000000	6.0000000	22.000000
C12	21.000000	-18.000000	-1.0000000	30.000000	44.000000
C13	-36.000000	-13.000000	-80.000000	3.0000000	59.000000
C14	0.0000000	3.0000000	-2.0000000	2.0000000	-71.000000
X15	-76.000000	-29.000000	9.0000000	-21.000000	-48.000000
X16	-59.000000	0.0000000	-36.000000	-12.000000	-56.000000
X17	-42.000000	20.000000	-48.000000	7.0000000	-43.000000
X18	-3.0000000	19.000000	-13.000000	-21.000000	-67.000000
X19	-56.000000	-25.000000	2.0000000	-23.000000	7.0000000
C20	-33.000000	-3.0000000	7.0000000	-12.000000	-90.000000

S	C7	C8	C9	C10	C11
C7	0.0000000				
C8	-60.000000	0.0000000			
C9	-7.0000000	9.0000000	0.0000000		
C10	-19.000000	-7.0000000	2.0000000	0.0000000	
C11	-137.00000	-13.000000	-14.000000	-15.000000	0.0000000
C12	31.000000	-30.000000	4.0000000	-48.000000	51.000000
C13	-20.000000	-10.000000	19.000000	-77.000000	-41.000000
C14	-13.000000	-30.000000	-2.0000000	-27.000000	7.0000000
X15	8.0000000	-21.000000	0.0000000	-24.000000	-7.0000000
X16	-43.000000	-72.000000	-10.000000	-22.000000	-27.000000
X17	-11.000000	22.000000	-18.000000	-16.000000	-28.000000
X18	0.0000000	-12.000000	-33.000000	-67.000000	-32.000000
X19	-33.000000	-15.000000	-44.000000	-23.000000	-36.000000
C20	-28.000000	-15.000000	-53.000000	-6.0000000	-39.000000

S	C12	C13	C14	X15	X16
C12	0.0000000				
C13	-72.000000	0.0000000			
C14	-3.0000000	-15.000000	0.0000000		
X15	-10.000000	22.000000	-70.000000	0.0000000	
X16	-29.000000	0.0000000	-12.000000	-15.000000	0.0000000
X17	36.000000	-25.000000	-35.000000	-28.000000	-34.000000
X18	-4.0000000	2.0000000	-33.000000	-4.0000000	-20.000000
X19	-43.000000	-28.000000	-28.000000	-11.000000	26.000000
C20	15.000000	1.0000000	-3.0000000	-22.000000	-37.000000

```
  S |         X17        X18        X19        C20
  ---------------------------------------------------
X17 |   0.0000000
X18 |  -63.000000  0.0000000
X19 |   1.0000000 -37.000000  0.0000000
C20 |  -27.000000 -14.000000 -5.0000000  0.0000000
```

The scaling coordinates are simply the principal components of the distance matrix $1 - Prox$:

```
Torgerson Scaling Coordinates (PComp of D=1-PROX)
************************************************

             Dense Matrix (155 by 2)

                |       DIM_1      DIM_2
               -------------------------
OBS_001 |  -0.3311319 -0.0501309
OBS_002 |  -0.0813852  0.1252977
OBS_003 |  -0.1021628  0.1381530
OBS_004 |  -0.3817703 -0.0492290
OBS_005 |  -0.3363352 -0.1282851
.............................
OBS_151 |   0.3242111 -0.1540918
OBS_152 |  -0.0403245  0.0425678
OBS_153 |   0.2432055  0.0695664
OBS_154 |   0.2217630 -0.0157302
OBS_155 |   0.2955053 -0.1271302
```

For space saving reason we drop here the output of the casewise variable importance. The training of this data is fast:

```
Total Computation Time: 0
```

2. Use the "imput" option for iterative refinement of the imputed values. In addition we specify an uncommonly high number of 10 (default is 5) iterations for imputation. Since "imput" is specified the "prox" option is set by the program but issuing a note. (Since the "outl" option is not selected, the outl" return argument is set to missing.)

```
optn = [ "ntree"          500 ,
         "mtry"             4 ,
         "ndsiz"            1 ,
```

```
          "impo"               ,  /* variable importance */
          "ret7"       "imput" ,  /* return imputation than intact */
          "imput"              ,  /* iterative imputation */
          "fillit"          10 ,  /* number iterations */
          "print"            3 ,
          "phist"            0 ,
          "pinit"            3 ];
< gof,misc,pred,gini,prox,outl,imp >
            = ranfor(hep,modl,optn,clas,cwgt,hep);
```

The following table shows in its columns some indicators for the size of changes in the imputed values at each imputation:

(a) the mean change of the entire set of missing values for all of the numerical variables

(b) the maximum change of the entire set of missing values for all of the numerical variables

(c) the relative number of changes in the selected levels for the missing values in categorical variables.

Imputation by 10 Iterations

Iter	DXmean	DXmax	DXmisclas	ErrTrain
1	8.25135470	37.3353383	0.02222222	0.21290323
2	3.18788595	28.0927746	0.02222222	0.19354839
3	0.92160096	14.2229811	0.00000000	0.19354839
4	0.53240146	3.02134399	0.00000000	0.18064516
5	0.56456849	4.18271330	0.00000000	0.17419355
6	0.53091375	4.82520543	0.00000000	0.16129032
7	0.49253864	3.47362420	0.00000000	0.15483871
8	0.52282913	5.49471898	0.00000000	0.15483871
9	0.51697582	5.07543486	0.00000000	0.14838710

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	39.00000000	0.923327953	0.001973098	0.009260545
2	0.000000000	0.152967788	0.004660357	0.004765331
3	1.000000000	0.110931394	-0.001585069	0.004662419
4	1.000000000	0.044339773	0.000347500	0.002800799
5	0.000000000	0.654602587	0.004988979	0.006924656
6	1.000000000	0.737584949	0.004691918	0.007754369
7	1.000000000	0.072019791	0.000375684	0.004268473
8	1.000000000	0.078382901	0.001132391	0.003718546

9	1.000000000	0.082320217	0.001613253	0.005797327
10	1.000000000	0.198357037	0.000576319	0.004620204
11	1.000000000	1.551782672	0.013082089	0.012781069
12	1.000000000	1.261441283	0.008227639	0.008417217
13	1.000000000	0.517402827	0.005128088	0.007179135
14	1.000000000	2.647910367	0.011139976	0.012042018
15	85.00000000	1.039758321	0.002556627	0.008716539
16	58.00000000	0.406029720	-0.003089068	0.007125573
17	4.000000000	3.393839406	0.020411536	0.014671474
18	60.00000000	4.323431378	0.040097555	0.020562039
19	0.000000000	0.803569636	0.010513497	0.010589962

Classification Rate=16.129 (Training)

	Y_1	Y_2
1	18.75000000	15.44715447

Final Error: 16.129 (Training)

	YData_1	YData_2
YPred_1	26	19
YPred_2	6	104

Classification Rate=1.93548 (Test)

	Y_1	Y_2
1	3.125000000	1.626016260

Final Error: 1.93548 (Test)

	YData_1	YData_2
YPred_1	31	2
YPred_2	1	121

Largest (Offdiagonal) Proximity Values

Row	Column	Value
80	66	0.988000000
80	71	0.986000000
71	66	0.978000000
45	8	0.956000000
10	5	0.952000000
10	8	0.942000000

```
                          55      35  0.938000000
                          45       5  0.926000000
                          45      10  0.914000000
                          57      45  0.902000000
    Sparsity of Proximity Matrix:   3.5567 % (11660 Nonzeros)


                    Total Computation Time: 11
```

3. Cancer Remission Data: N=27, n=7: Using Test Data (first 10 observations of training data)

```
 remis = [ 1  .8  .83  .66 1.9 1.1    .996 ,  1  .9  .36  .32 1.4  .74   .992 ,
           0  .8  .88  .7   .8  .176  .982 ,  0 1.   .87  .87  .7 1.053  .986 ,
           1  .9  .75  .68 1.3  .519  .98  ,  0 1.   .65  .65  .6  .519  .982 ,
           1  .95 .97  .92 1.  1.23   .992 ,  0  .95 .87  .83 1.9 1.354 1.02  ,
           0 1.   .45  .45  .8  .322  .999 ,  0  .95 .36  .34  .5  .0   1.038 ,
           0  .85 .39  .33  .7  .279  .988 ,  0  .7  .76  .53 1.2  .146  .982 ,
           0  .8  .46  .37  .4  .38  1.006 ,  0  .2  .39  .08  .8  .114  .99  ,
           0 1.   .9   .9  1.1 1.037  .99  ,  1 1.   .84  .84 1.9 2.064 1.02  ,
           0  .65 .42  .27  .5  .114 1.014 ,  0 1.   .75  .75 1.  1.322 1.004 ,
           0  .5  .44  .22  .6  .114  .99  ,  1 1.   .63  .63 1.1 1.072  .986 ,
           0 1.   .33  .33  .4  .176 1.01  ,  0  .9  .93  .84  .6 1.591 1.02  ,
           1 1.   .58  .58 1.   .531 1.002 ,  0  .95 .32  .3  1.6  .886  .988 ,
           1 1.   .6   .6  1.7  .964  .99  ,  1 1.   .69  .69  .9  .398  .986 ,
           0 1.   .73  .73  .7  .398  .986 ];
   temis = remis[1:10,];


   /* Change the response event coding like SAS */
   remis[,1] = !remis[,1]; print "Remis=",remis;
   temis[,1] = !temis[,1]; print "Temis=",temis;

   cnam = [" remiss cell smear infil li blast temp "];
   remis = cname(remis,cnam);
   temis = cname(temis,cnam);
```

Some of the following options are default and may not be specified explicitly:

```
   clas = 1;
   modl = "1 = 2:7";


   optn = [ "ntree"      100 ,
            "mtry"         3 ,
```

```
        "ndsiz"        1 ,
        "impo"          ,  /* compute var importance */
        "outl"          ,  /* look for outliers */
        "prox"          ,  /* compute proximities */
        "intact"        ,  /* compute var interactions */
        "dimsc"        2 ,  /* dimension for MDS */
        "pprox"         ,  /* print proximitiy matrix */
        "print"        3 ,
        "phist"        1 ,  /* print search history */
        "pinit"        3 ];
< gof,misc,pred,gini,prox,outl,intact,tmisc,tpred,tgini >
              = ranfor(remis,modl,optn,clas,.,temis);


           Random Forest Classification (V=0)

  Number Observations . . . . . .          27
  Number Observations in Test Set . . . . .          10
  Number Variables. . . . . .          6
  Number of Trees . . . . . .         100
  Number Variables Selected per Node. . . . .          3
  Minimum Node Size . . . . . .          1
  Maximum Node Size . . . . . .          55
  Method Proximity Matrix . . . . . .          1
  Number Missing Values . . . . . .          0
  Seed for Random Generator . . . . . .       32873


                ****************
                Model Information
                ****************

              Number Valid Observations   27
              Observations Test Data      10
              Response Variable          Y[1]
              N Independend Variables      6


                *************
                Model Effects
                *************

             X2 + X3 + X4 + X5 + X6 + X7


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

****

Class	Level	Value
Y[1]	2	0
		1

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
X[2]	27	0.8814815	0.1866445	-2.3686287	6.3400640
X[3]	27	0.6351852	0.2140519	-0.0675912	-1.4973157
X[4]	27	0.5707407	0.2375666	-0.2982745	-1.0350292
X[5]	27	1.0037037	0.4677947	0.7319276	-0.5094833
X[6]	27	0.6888519	0.5358045	0.7423387	-0.0753490
X[7]	27	0.9970000	0.0148609	1.1428113	0.6986903

	Tree	Nodes	ClassError	Misc_1	Misc_2
Train	1	17	55.555556	0.0000000	83.333333
Test			10.000000	0.0000000	16.666667
Train	2	13	51.851852	0.0000000	77.777778
Test			20.000000	0.0000000	33.333333
Train	3	11	55.555556	22.222222	72.222222
Test			10.000000	0.0000000	16.666667
Train	4	13	51.851852	22.222222	66.666667
Test			10.000000	0.0000000	16.666667
Train	5	15	40.740741	22.222222	50.000000
Test			0.0000000	0.0000000	0.0000000
.....					
Train	95	7	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000
Train	96	9	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000
Train	97	11	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000
Train	98	9	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000
Train	99	13	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000
Train	100	15	33.333333	44.444444	27.777778
Test			0.0000000	0.0000000	0.0000000

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	0.950000000	0.262087130	-0.007393218	0.016121437
2	0.630000000	0.814872597	-0.009965229	0.022925437
3	0.600000000	0.763838751	-0.008987734	0.026370646
4	0.800000000	2.474271366	0.049662393	0.044345252
5	0.519000000	1.145335994	0.007060384	0.025381721
6	0.990000000	0.539594163	-0.019718975	0.019499971

Gini Coefficients (Test)

	1
cell	0.263618534
smear	0.825561114
infil	0.776061526
li	2.460867265
blast	1.128106730
temp	0.545784831

Classification Rate=33.3333 (Training)

	Y_1	Y_2
1	44.44444444	27.77777778

Final Error: 33.3333 (Training)

	YData_1	YData_2
YPred_1	5	5
YPred_2	4	13

Classification Rate=0 (Test)

	Y_1	Y_2
1	0.000000000	0.000000000

Final Error Rate: 0 (Test)

	YData_1	YData_2
YPred_1	4	0
YPred_2	0	6

Largest (Offdiagonal) Proximity Values

Row	Column	Value
19	14	1.000000000
14	11	0.950000000
21	10	0.950000000
19	11	0.950000000
17	13	0.920000000
19	17	0.910000000
17	14	0.910000000
21	9	0.880000000
17	10	0.870000000
17	11	0.870000000

Sparsity of Proximity Matrix: 7.1429 % (351 Nonzeros)

Table of N=1 Outliers

Obs	Outlier Out
24	13.07262102

Pairwise Variable Interactions

****

Dense Symmetric Matrix (6 by 6)

S	cell	smear	infil	li	blast
cell	0.0000000				
smear	3.0000000	0.0000000			
infil	10.000000	-14.000000	0.0000000		
li	-7.0000000	-12.000000	-1.0000000	0.0000000	
blast	-15.000000	-12.000000	-29.000000	8.0000000	0.0000000
temp	-14.000000	4.0000000	-5.0000000	-16.000000	1.0000000

S	temp
temp	0.0000000

Torgerson Scaling Coordinates (PComp of D=1-PROX)

****

Dense Matrix (27 by 2)

```
          |       DIM_1      DIM_2
     -------------------------
OBS_01 |   0.3855735 -0.1080561
OBS_02 |   0.3288342 -0.2915156
OBS_03 |  -0.3139045  0.0449208
OBS_04 |  -0.1467126  0.3061019
OBS_05 |   0.3847756 -0.2106480
OBS_06 |  -0.2015833  0.0306407
OBS_07 |   0.3019970  0.2482275
OBS_08 |   0.2535534  0.4477960
OBS_09 |  -0.3923091 -0.0826623
OBS_10 |  -0.4007642 -0.0519209
OBS_11 |  -0.4196114 -0.1111152
OBS_12 |   0.0796861 -0.1793010
OBS_13 |  -0.3942971 -0.0715258
OBS_14 |  -0.4155913 -0.1183136
OBS_15 |   0.2945605  0.2983732


OBS_16 |   0.2792858  0.3146518
OBS_17 |  -0.4100879 -0.0908294
OBS_18 |   0.3012624  0.3060072
OBS_19 |  -0.4155913 -0.1183136
OBS_20 |   0.4317832 -0.2222870
OBS_21 |  -0.3949787 -0.0545405
OBS_22 |  -0.1453496  0.3629172
OBS_23 |   0.4087960 -0.2176505
OBS_24 |   0.2112363 -0.1964902
OBS_25 |   0.4329943 -0.2205227
OBS_26 |   0.1769678 -0.1823614
OBS_27 |  -0.2205252  0.1684176


  Total Computation Time: 0
```

4. Random Forest Regression: Boston Housing Data: N=506, p=14

```
print "Boston Housing Data";
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[,5] = 100. * hous[,5];
```

```
ind = [ 1:3 5:14 ];
hous2 = hous[,ind];
print "Hous2=",hous2[1:5,];


modl = "13 = 1:12";


print "Random Forest Regression with Test: ndsiz=1";
optn = [ "ntree"          100 ,
         "outmod"   "bhous.rf",
         "mtry"             4 ,
         "ndsiz"            1 ,
         "prox"               ,
         "outl"               ,
         "impo"               ,
         "intact"             ,
         "print"            3 ,
         "pobs"               ,
         "phist"            1 ,
         "pinit"            3 ];
< gof,misc,pred,gini,prox,outl,intact >
                        = ranfor(hous2,modl,optn,.,.,hous2);
```

The default value of the "memthr" options is smaller than the size of the full proximity matrix. The proximity matrix is stored on utility file:

```
[note] Proximity matrix is not allocated and not returned since it
       has more entries (128271) than the MEMTHR=100000 definition
       permits.
```

```
                    Random Forest Regression

    Number Observations . . . . . .          506
    Number Variables. . . . . .               12
    Number of Trees . . . . . .              100
    Number Variables Selected per Node. . . .   4
    Minimum Node Size . . . . . .               1
    Maximum Node Size . . . . . .            1013
    Method Proximity Matrix . . . . . .         2
    Number Missing Values . . . . . .           0
    Seed for Random Generator . . . . . .   32873


                  **********
```

Model Information
****

Number Valid Observations 506
Response Variable Y[13]
N Independend Variables 12

****
Model Effects
****

X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 + X12

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[13]	506	22.532806	9.1971041	1.1080984	1.4951969
X[1]	506	3.6135236	8.6015451	5.2231487	37.130508
X[2]	506	11.363636	23.322453	2.2256663	4.0315101
X[3]	506	11.136779	6.8603530	0.2950215	-1.2335397
X[4]	506	55.469506	11.587768	0.7293079	-0.0646673
X[5]	506	6.2846344	0.7026172	0.4036121	1.8915002
X[6]	506	68.574901	28.148862	-0.5989626	-0.9677156
X[7]	506	3.7950427	2.1057101	1.0117806	0.4879412
X[8]	506	9.5494071	8.7072594	1.0048146	-0.8672320
X[9]	506	408.23715	168.53712	0.6699559	-1.1424080
X[10]	506	18.455534	2.1649458	-0.8023248	-0.2850920
X[11]	506	356.67403	91.294863	-2.8903738	7.2268177
X[12]	506	12.653063	7.1410615	0.9064601	0.4932395

Tree	Nodes	Test_Err	Train_Err	Err/Yvar
1	741	44.972984	19.251601	0.2280467
2	765	22.003765	24.331437	0.2882204
3	693	15.336278	30.985907	0.3670465
4	727	13.817961	41.067790	0.4864725
5	713	11.958102	36.305018	0.4300546
.....				
96	743	6.6457741	15.763092	0.1867232
97	711	6.6025784	15.697570	0.1859471
98	719	6.5784983	15.692292	0.1858846

99	711	6.5817229	15.708347	0.1860747
100	733	6.6077837	15.769115	0.1867946

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	0.253870010	3.495731460	555.2858302	20890.26460
2	0.000000000	0.420953934	-3.388629472	1440.643865
3	9.689999580	6.858681607	448.6420839	10793.68861
4	53.79999876	7.221612929	775.3151528	22453.17562
5	6.208000183	31.96598351	2543.797665	92185.77265
6	77.30000305	2.341002816	336.8280483	9444.407549
7	3.199199915	3.509960852	534.4835184	19033.21918
8	5.000000000	0.488814425	190.2772169	4827.302732
9	330.0000000	2.759218233	365.0079824	7627.842836
10	19.00000000	8.079781674	576.5932252	14433.46542
11	391.4299927	1.214194625	264.1729710	6551.636015
12	11.34000015	31.64406394	4354.767849	333284.7698

Final Error: Training=15.7691 Test=6.60778

Predicted Values (Training)

Obs	Data	Pred	Nout
1	24.0000000	27.7023254	43
2	21.6000004	23.4928572	28
3	34.7000008	32.1911114	45
4	33.4000015	32.9580648	31
5	36.2000008	31.9829274	41
.....			
501	16.7999992	22.3184209	38
502	22.3999996	22.2307693	26
503	20.6000004	17.9424999	40
504	23.8999996	25.0666667	39
505	22.0000000	22.9999998	34
506	11.8999996	21.4964288	28

Predicted Values (Test)

Obs	Data	Pred	Nout
1	24.0000000	25.2649999	100
2	21.6000004	22.3900002	100

```
  3  34.7000008  33.1890004       100
  4  33.4000015  31.7920009       100
  5  36.2000008  32.0650006       100
  ..................................
501  16.7999992  20.0129996       100
502  22.3999996  23.4559998       100
503  20.6000004  20.2170001       100
504  23.8999996  24.5929998       100
505  22.0000000  22.1949999       100
506  11.8999996  15.8779999       100
```

The large Proximity matrix is 90 % sparse and can therefore be moved from the file into core memory after the tree generation. The computation of outliers, prototypes and the MDS coordinates is much faster for a sparse incore matrix than for a large matrix on a utility file:

```
          Largest (Offdiagonal) Proximity Values

                Row  Column        Value
                170     169  0.520000000
                491     490  0.490000000
                144     143  0.480000000
                322     321  0.460000000
                335     334  0.460000000
                384     383  0.450000000
                138     129  0.430000000
                236     231  0.420000000
                102     101  0.390000000
                125     123  0.380000000
Sparsity of Proximity Matrix:  90.8319 % (11760 Nonzeros)


                Table of N=1 Outliers
                 266  10.2013166 ***


             Pairwise Variable Interactions
             ******************************

           Dense Symmetric Matrix (12 by 12)

 S |          1          2          3          4          5
---------------------------------------------------------------
 1 |   0.0000000
 2 |  -11.000000  0.0000000
```

```
 3 |   10.000000 -36.000000  0.0000000
 4 |   18.000000 -10.000000 -3.0000000  0.0000000
 5 |  -31.000000 -10.000000 -7.0000000 -27.000000  0.0000000
 6 |  -18.000000  3.0000000  22.000000  9.0000000 -25.000000
 7 |   5.0000000 -4.0000000  3.0000000 -4.0000000 -9.0000000
 8 |   2.0000000 -12.000000 -27.000000 -12.000000 -14.000000
 9 |   3.0000000  0.0000000  28.000000 -12.000000 -10.000000
10 |   8.0000000 -17.000000 -18.000000  32.000000  4.0000000
11 |  -24.000000  15.000000  19.000000 -7.0000000 -3.0000000
12 |  -12.000000 -2.0000000  1.0000000 -18.000000  8.0000000


 S |           6          7          8          9         10
 ------------------------------------------------------------
 6 |   0.0000000
 7 |   3.0000000  0.0000000
 8 |   10.000000  3.0000000  0.0000000
 9 |  -15.000000 -10.000000 -14.000000  0.0000000
10 |   14.000000 -23.000000 -7.0000000  21.000000  0.0000000
11 |  -35.000000  3.0000000 -10.000000 -5.0000000 -2.0000000
12 |  -9.0000000 -28.000000 -12.000000 -12.000000 -19.000000


 S |          11         12
 --------------------------
11 |   0.0000000
12 |  -15.000000  0.0000000


                Total Computation Time: 4
```

We now use the model stored on file `"bhous.rf"` to score the same data:

```
print "Run Prediction with the same data";
optn = [ "prox"               ,
         "print"            3 ,
         "pobs"               ,
         "phist"            1 ,
         "pinit"            3 ];
< gof,misc,pred,gini,prox > = rafprd("bhous.rf",hous2,modl,optn);


                Random Forest Regression

   Number Observations . . . . . .          506
   Number Variables. . . . . .               12
```

```
Number of Trees . . . . . .          100
Number Variables Selected per Node. . . . .          4
Minimum Node Size . . . . . .          1
Maximum Node Size . . . . . .       1013
Method Proximity Matrix . . . . . .          1
Number Missing Values . . . . . .          0
Seed for Random Generator . . . . . .      32873
```

Skipping some of the output we show the scoring history across the 100 trees:

```
     Tree Nodes  MSQ_Error   Err/Yvar

        1   741  44.972984  0.5327318
        2   765  22.003765  0.2606477
        3   693  15.336278  0.1816674
        4   727  13.817961  0.1636820
        5   713  11.958102  0.1416508
      ..............................
       95   721  6.6431564  0.0786922
       96   743  6.6457741  0.0787232
       97   711  6.6025784  0.0782115
       98   719  6.5784983  0.0779262
       99   711  6.5817229  0.0779644
      100   733  6.6077837  0.0782731


    Final Error: 6.60778 (Validation)
```

The proximity matrix is the same as for training. The predicted values look much better now:

```
        Imputation, Gini, and Variable Importance

  Var       FilMis        GiniC      Z-Score       Signif

    1  0.253870010  3.433297846  10.09661988  1.056644702
    2  0.000000000  0.422703203  0.969120525  0.277214543
    3  9.689999580  6.825719229  6.737655578  0.913881674
    4  53.79999876  7.332901245  12.14435204  1.194826872
    5  6.208000183  32.00099467  32.03024713  1.698681096
    6  77.30000305  2.287716094  6.802336117  0.875277729
    7  3.199199915  3.470495844  11.32671343  1.556311014
    8  5.000000000  0.496643739  2.855670228  0.549024759
    9  330.0000000  2.736705955  5.336565919  0.793066284
   10  19.00000000  8.076260218  9.395708340  1.082347442
```

11	391.4299927	1.198354263	4.037045655	0.657530611
12	11.34000015	31.71820769	49.44383491	3.594009626

Predicted Values (Validation)

Obs	Data	Pred	Nout
1	24.00000000	27.70232544	43
2	21.60000038	23.49285718	28
3	34.70000076	32.19111137	45
4	33.40000153	32.95806485	31
5	36.20000076	31.98292742	41
6	28.70000076	27.15609760	41
7	22.89999962	20.43571436	42
8	27.10000038	19.54864881	37
9	16.50000000	19.49761915	42
10	18.89999962	21.60263162	38
...	...	...	...
501	16.79999924	22.31842091	38
502	22.39999962	22.23076930	26
503	20.60000038	17.94249990	40
504	23.89999962	25.06666670	39
505	22.00000000	22.99999983	34
506	11.89999962	21.49642876	28

Largest (Offdiagonal) Proximity Values

Row	Column	Value
170	169	0.520000000
491	490	0.490000000
144	143	0.480000000
335	334	0.460000000
322	321	0.460000000
384	383	0.450000000
138	129	0.430000000
236	231	0.420000000
102	101	0.390000000
205	204	0.380000000

Sparsity of Proximity Matrix: 90.8319 % (11760 Nonzeros)

Table of N=1 Outliers

Obs	Outlier Out

```
      266  10.20131660

Total Computation Time: 0
```

5.10 Function `rafprd`

< gof,misc,pred,gini,prox,outl,intact > = rafprd(infor,data<,...> . >)

gof = rafprd(infor,data,modl<,optn<,class<,cwgt> . >)

Purpose: The `rafprd` function is used for evaluating (scoring) a Random Forest model generated with training by the `ranfor()` function for a data set which has the same column - effects mapping as was used for the training. Similar to the `random forest` function the results may correspond to different versions of the algorithm:

- R version of the algorithm by A. Liaw (2009); (except for computing the variable importance measurements and the proximity matrix the algorithm is the same as that of version 5).
- Fortran version 5 of the algorithm by Breiman and Cutler (since the *sparse* computation of the rectangular proximity matrix as in version 5 of the Fortran program seems to have some problems, the proximity matrix is by default computed as in version 4)
- Fortran version 4 of the algorithm by Breiman and Cutler

Input: **infor** string specifying the path to for model input file generated earlier with `ranfor()`

data specifies the data with N rows and a set of n columns which can be defined as effects using the model string argument. The column - effects mapping should be the same as was used by the training.

modl specifies which column corresponds to the response and which columns to the p effects of the linear model. Usually this input is the same as for the training.

optn 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.

class only for classification: an optional argument specifying which of the columns of the input data matrix $\mathbf{X}$ are CLASS variables. The response y must be a class variable and is many times binary. Usually this input is the same as for the training.

cwgt an optional argument specifying positive weights $w_j > 0$ for the levels j of the response variable. Usually this input is the same as for the training.

Option	Second Column	Meaning
"dimsc"	int	dimension d for Torgerson scaling (def=0)
		(needs proximity matrix)
"impo"		compute importance for p variables
"impn"		compute casewise variable importance (for plotting)
"intact"		compute $p \times p$ variable interactions
"memthr"	int	bound for switching incore storage of proximity to file
"nearn"	int	number of columns of proximity matrix (def=0)
		if > 0: computes rectangular prox matrix as in version 5
"nprot"	int	maximum number of prototypes to compute (def=0)
		(needs proximity matrix)
"outl"		detect outlier observations (needs proximity matrix)
"phist"	int	output of the tree history (def=1)
"pinit"	int	print information about input data
"pmis"		print $K \times K$ misclassification table
"pobs"		print $N \times (K + 3)$ predicted value table
		also prints N vector of outlier indicators
"pprox"		print large proximity matrix
"print"	int	amount of printed output (def=2)
"prox"		compute large $N \times N$ proximity matrix;
		needed for iterative imputation, outliers, scaling
"prox"	"rect"	compute rectangular (version 5) $N \times nearn$ proximity matrix (approximation)
"prox"	"file"	store large $N \times N$ proximity matrix on file; see also "memthr" option
"ret7"	string	what is returned in argument 7:
	"inter"	interactions of variables (this is default)
	"proto"	prototypes ("nprot" must be ¿ 0)
	"mdsca"	scaling coordinates ("dimsc" must be ¿ 0)
	"parco"	parallel coordinates ("ncoor" must be ¿ 0)
"seed"	int	seed for random generator (def=time of day)
"vers"	int	should be 0, 4, or 5 (default is 0):
	0	R version for the variable importance
	5	Fortran version 5 for the variable importance
	4	Fortran version 4 for the variable importance

Note, that options for specifying the tree generation cannot be specified here. They are specified by the model file input. Among others this prevents from applying the scoring algorithm of version 4 to a tree structure generated by version 5 algorithm and vice versa.

Output: **gof** This is a vector of scalar results, containing

1. indicates failure in execution
2. total time in seconds
3. classification error

misc $K \times K$ classification table for the training;

pred $N \times (K+3)$ matrix containing in its columns:

1. input values of the response y
2. predicted values of training $\hat{y}$
3. K columns with observation counts

gini p vector with Gini coefficients

prox $N \times N$ or $N \times nrnn$ matrix of proximities this matrix is symmetric and diagonal dominant (with 1's in the diagonal) and therefore positive semidefinite (and therefore can be used for the multidimensional scaling function);

outl N vector of values indicating outlier observations; The printed output identifies an outlier when the outlyingness is at least 10.

intact by default the $p \times p$ matrix of pairwise variable interactions, but could be also one of the those depending on the `"ret7"` option:

- "ret7"="mdsca": $N \times dimsc$ matrix of point coordinates computed as the $dimsc$ principal components of the $1-prox$ distance matrix; CMAT permits you to run better versions of multidimensional scaling algorithms on such a distance matrix based on the output proximities;
- "ret7"="proto": matrix of prototypes (assumes version 5 with nonzero `"nearn"` specification (may have problems);
- "ret7"="parco": $N \times (3+Ncoor)$ matrix cases with highest votes (this is called "parallel coordinates" in version 4 and is similar to prototypes in version 5).

Restrictions: 1. The column-effect mapping (model and class specifications) of training and test data must be compatible.

2.

Relationships: ranfor(), split(), reg(), nlreg()

Examples: 1. Glass data: Nobs=214, nvar=9, Y in col 11:

```
   options NOECHO;
   glass = [
#include "..\\tdata\\glass.dat"
           ];
   options ECHO;
```

```
glass2 = shape(glass,.,11);
glass2 = glass2[,2:11];
cnam = [" RI Na Mg Al Si K Ca Ba Fe Type "];
glass2 = cname(glass2,cnam);
```

Categorical weights are specified with no theoretical background

```
modl = "10 = 1:9";
clas = 10;
cwgt = [ 2. 2 2 1 2 1. ];
```

Training the model is done with the `ranfor()` function:

```
optn = [ "ntree"          100 ,
         "outmod"   "glass.rf",
         "mtry"             4 ,
         "ndsiz"            1 ,
         "prox"               ,
         "impo"               ,
         "outl"               ,
         "intact"             ,
         "dimsc"            2 ,  /* dimension for MDS */
         "print"            3 ,
         "phist"            1 ,
         "pinit"            3 ];
< gof,misc,pred,gini,prox,outl,intact,tmisc,tpred,tgini > =
                ranfor(glass2,modl,optn,clas,cwgt,glass2);
```

Here, we show only a small portion of the output:

```
              Random Forest Classification (V=0)

   Number Observations . . . . . .          214
   Number Observations in Test Set .          214
   Number Variables. . . . . .                9
   Number of Trees . . . . . .              100
   Number Variables Selected per Node.        4
   Minimum Node Size . . . . . .              1
   Maximum Node Size . . . . . .            429
   Method Proximity Matrix . . . . .          1
   Number Missing Values . . . . . .          0
   Seed for Random Generator . . . .      60249

                  *****************
```

Model Information
****

Number Valid Observations 214
Response Variable Y[10]
N Independend Variables 9

****
Model Effects
****

X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9

****
Class Level Information
****

Class	Level	Value
Y[10]	6	1 2 3 5 6 7

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
X[1]	214	1.5183654	0.0030369	1.6254303	4.9318001
X[2]	214	13.407850	0.8166036	0.4541815	3.0522324
X[3]	214	2.6845327	1.4424078	-1.1525593	-0.4103190
X[4]	214	1.4449065	0.4992696	0.9072898	2.0605690
X[5]	214	72.650935	0.7745458	-0.7304472	2.9679025
X[6]	214	0.4970561	0.6521918	6.5516483	54.689699
X[7]	214	8.9569626	1.4231535	2.0470539	6.6819780
X[8]	214	0.1750467	0.4972193	3.4164246	12.541084
X[9]	214	0.0570093	0.0974387	1.7543275	2.6620156

****
Class Level Information
****

Class	Level	Value

```
Y[ 1]      2        1       2
C[ 3]      2        1       2
C[ 4]      3        1       2 _Missing
C[ 5]      2        1       2
C[ 6]      3        1       2 _Missing
C[ 7]      3        1       2 _Missing
C[ 8]      3        1       2 _Missing
C[ 9]      3        1       2 _Missing
C[10]      3        1       2 _Missing
C[11]      3        1       2 _Missing
C[12]      3        1       2 _Missing
C[13]      3        1       2 _Missing
C[14]      3        1       2 _Missing
C[20]      2        1       2
```

	Tree	Nodes	ClassError	Misc_1	Misc_2	Misc_3
Train	1	83	58.411215	1.89e-006	5.78e-006	3.14e-006
				8.40e-007	inf	inf
Test			14.953271	12.857143	19.736842	17.647059
				7.6923077	22.222222	6.8965517
Train	2	91	47.196262	1.68e-006	4.47e-006	-6.29e-007
				8.40e-007	inf	inf
Test			14.953271	2.8571429	22.368421	29.411765
				23.076923	11.111111	13.793103
Train	3	79	42.990654	1.87e-006	3.90e-006	-6.81e-007
				7.46e-007	inf	inf
Test			8.4112150	2.8571429	2.6315789	23.529412
				23.076923	0.0000000	24.137931
Train	4	79	43.925234	1.87e-006	3.81e-006	-4.11e-006
				8.38e-007	inf	inf
Test			4.2056075	0.0000000	1.3157895	17.647059
				7.6923077	0.0000000	13.793103
Train	5	85	39.252336	9.21e-007	3.62e-006	-3.81e-006
				4.66e-007	1.51e-006	1.03e-006
Test			3.7383178	0.0000000	1.3157895	5.8823529
				15.384615	0.0000000	13.793103
.....						
Train	96	89	23.831776	-6.20e-007	1.67e-006	1.81e-006
				6.48e-007	9.56e-007	4.65e-007
Test			0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	0.0000000	0.0000000
Train	97	87	23.831776	1.62e-006	1.67e-006	1.81e-006
				6.48e-007	-2.40e-007	4.65e-007
Test			0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	0.0000000	0.0000000

Train	98	81	23.831776	1.62e-006	1.67e-006	1.81e-006
				6.48e-007	-2.40e-007	4.65e-007
Test			0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	0.0000000	0.0000000
Train	99	73	24.299065	7.70e-007	1.76e-006	-6.19e-007
				6.48e-007	2.64e-007	4.65e-007
Test			0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	0.0000000	0.0000000
Train	100	85	24.299065	7.70e-007	1.76e-006	-6.19e-007
				6.48e-007	-7.08e-007	4.65e-007
Test			0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	0.0000000	0.0000000

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	1.517680000	1.614010740	0.119664677	0.121653464
2	13.30000000	0.590207517	0.051841677	0.065746461
3	3.480000000	1.537231094	0.145092030	0.148154936
4	1.360000000	2.079278015	0.129048860	0.133958186
5	72.79000000	0.319406155	0.030904780	0.046015201
6	0.550000000	0.560602166	0.053265029	0.063960094
7	8.600000000	1.291104953	0.085913182	0.094843245
8	0.000000000	0.772841751	0.072115282	0.079522374
9	0.000000000	0.235317608	0.009976845	0.020538579

Classification Rate=24.2991 (Training)

	Y_1	Y_2	Y_3	Y_4	Y_5
1	7.6968e-007	1.7599e-006	-6.1912e-007	6.4840e-007	-7.0800e-007

Classification Rate=24.2991 (Training)

	Y_6
1	4.6470e-007

Final Error: 24.2991 (Training)

	YData_1	YData_2	YData_3	YData_4	YData_5	YData_6
YPred_1	63	13	7	0	0	1
YPred_2	6	57	4	6	2	4
YPred_3	1	2	6	0	1	0

YPred_4	0	1	0	6	0	0
YPred_5	0	3	0	0	6	0
YPred_6	0	0	0	1	0	24

Classification Rate=0 (Test)

	Y_1	Y_2	Y_3	Y_4	Y_5
1	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000

Classification Rate=0 (Test)

	Y_6
1	0.000000000

Final Error: 0 (Test)

	YData_1	YData_2	YData_3	YData_4	YData_5	YData_6
YPred_1	70	0	0	0	0	0
YPred_2	0	76	0	0	0	0
YPred_3	0	0	17	0	0	0
YPred_4	0	0	0	13	0	0
YPred_5	0	0	0	0	9	0
YPred_6	0	0	0	0	0	29

Largest (Offdiagonal) Proximity Values

Row	Column	Value
40	39	1.000000000
203	201	0.990000000
205	203	0.990000000
205	192	0.990000000
213	211	0.990000000
213	195	0.990000000
211	207	0.990000000
211	205	0.990000000
200	199	0.990000000
205	201	0.980000000

Sparsity of Proximity Matrix: 56.0704 % (10106 Nonzeros)

Table of N=11 Outliers

Obs	Outlier	Out

```
  3  20.00000000
  6  20.00000000
 11  17.82831902
 13  19.17120764
 36  20.00000000
 57  12.91951136
 98  13.16850337
127  20.00000000
188  20.00000000
189  20.00000000
202  20.00000000
```

The model output file "glass.rf" is now used with the same data for scoring:

```
print "Run Prediction with the same data";
optn = [ "print"            3 ,
         "pobs"               ,
         "prox"               ,
         "impo"               ,
         "outl"               ,
         "intact"             ,
         "dimsc"            2 ,  /* dimension for MDS */
         "phist"            1 ,
         "pinit"            3 ];
< gof,misc,pred,gini,prox,outl,intact > =
       rafprd("glass.rf",glass2,modl,optn,clas,cwgt);


             Random Forest Classification (V=0)

   Number Observations . . . . . .          214
   Number Variables. . . . . .              9
   Number of Trees . . . . . .            100
   Number Variables Selected per Node. . . .  4
   Minimum Node Size . . . . . .            1
   Maximum Node Size . . . . . .          429
   Method Proximity Matrix . . . . . .      1
   Number Missing Values . . . . . .        0
   Seed for Random Generator . . . . . .  60249
```

We skip the initial output:

```
Tree Nodes ClassError     Misc_1     Misc_2     Misc_3     Misc_4
   1    83  14.953271  12.857143  19.736842  17.647059  7.6923077
                      22.222222  6.8965517
```

```
  2    91  14.953271  2.8571429  22.368421  29.411765  23.076923
                      11.111111  13.793103
  3    79  8.4112150  2.8571429  2.6315789  23.529412  23.076923
                      0.0000000  24.137931
  4    79  4.2056075  0.0000000  1.3157895  17.647059  7.6923077
                      0.0000000  13.793103
  5    85  3.7383178  0.0000000  1.3157895  5.8823529  15.384615
                      0.0000000  13.793103
              ......................................
 96    89  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
                      0.0000000  0.0000000
 97    87  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
                      0.0000000  0.0000000
 98    81  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
                      0.0000000  0.0000000
 99    73  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
                      0.0000000  0.0000000
100    85  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
                      0.0000000  0.0000000
```

Classification Rate=0 (Validation)

```
          Y_1          Y_2          Y_3          Y_4          Y_5
1 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

Classification Rate=0 (Validation)

```
                          Y_6
                1 0.000000000
```

Final Error: 0 (Validation)

```
          YData_1  YData_2  YData_3  YData_4  YData_5  YData_6
YPred_1        70        0        0        0        0        0
YPred_2         0       76        0        0        0        0
YPred_3         0        0       17        0        0        0
YPred_4         0        0        0       13        0        0
YPred_5         0        0        0        0        9        0
YPred_6         0        0        0        0        0       29
```

Predicted Values and Votes (Validation)

```
Obs  Data  Pred  Nout      Vot_1      Vot_2      Vot_3      Vot_4
```

1	0	0	214	1.5200000	0.2000000	0.2800000	0.0000000
				0.0000000	0.0000000		
2	0	0	214	1.7000000	0.2000000	0.0800000	0.0000000
				0.0200000	0.0000000		
3	0	0	214	1.1800000	0.8200000	0.0000000	0.0000000
				0.0000000	0.0000000		
4	0	0	214	1.7200000	0.2800000	0.0000000	0.0000000
				0.0000000	0.0000000		
5	0	0	214	1.8400000	0.1600000	0.0000000	0.0000000
				0.0000000	0.0000000		
	.....						
211	5	5	214	0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	1.0000000		
212	5	5	214	0.0000000	0.0200000	0.0000000	0.0100000
				0.1600000	0.9000000		
213	5	5	214	0.0000000	0.0000000	0.0000000	0.0000000
				0.0000000	1.0000000		
214	5	5	214	0.0000000	0.0000000	0.0000000	0.0100000
				0.0000000	0.9900000		

Largest (Offdiagonal) Proximity Values

Row	Column	Value
40	39	1.000000000
203	201	0.990000000
205	203	0.990000000
205	192	0.990000000
213	211	0.990000000
213	195	0.990000000
211	207	0.990000000
211	205	0.990000000
200	199	0.990000000
205	201	0.980000000

Sparsity of Proximity Matrix: 56.0704 % (10106 Nonzeros)

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	1.517680000	1.614010740	0.046168224	0.079426642
2	13.30000000	0.590207517	0.020233645	0.042328482
3	3.480000000	1.537231094	0.053224299	0.088624951
4	1.360000000	2.079278015	0.048364486	0.082883047
5	72.79000000	0.319406155	0.012383178	0.030292333
6	0.550000000	0.560602166	0.022102804	0.043720359

7	8.600000000	1.291104953	0.031074766	0.058230670
8	0.000000000	0.772841751	0.026121495	0.048377401
9	0.000000000	0.235317608	0.004252336	0.012354065

Pairwise Variable Interactions
****

Dense Symmetric Matrix (9 by 9)

S	RI	Na	Mg	Al	Si
RI	0.0000000				
Na	-4.0000000	0.0000000			
Mg	-29.000000	-7.0000000	0.0000000		
Al	30.000000	-19.000000	22.000000	0.0000000	
Si	-12.000000	-23.000000	-7.0000000	-5.0000000	0.0000000
K	3.0000000	11.000000	-23.000000	-44.000000	-17.000000
Ca	-1.0000000	-32.000000	-9.0000000	2.0000000	2.0000000
Ba	-5.0000000	25.000000	11.000000	-38.000000	-14.000000
Fe	-29.000000	-17.000000	-24.000000	-7.0000000	-1.0000000

S	K	Ca	Ba	Fe
K	0.0000000			
Ca	24.000000	0.0000000		
Ba	-2.0000000	9.0000000	0.0000000	
Fe	-7.0000000	-30.000000	-2.0000000	0.0000000

Torgerson Scaling Coordinates (PComp of D=1-PROX)
****

Dense Matrix (214 by 2)

	DIM_1	DIM_2
OBS_001	0.0281695	-0.0420772
OBS_002	0.1268872	-0.1418484
OBS_003	0.0618151	0.2143094
OBS_004	0.1360817	-0.1503379
OBS_005	0.1769051	-0.2265034
.....		
OBS_210	-0.5656604	-0.0516936
OBS_211	-0.6168983	-0.0567496
OBS_212	-0.4897823	-0.0447789

```
OBS_213 |  -0.6175621 -0.0568163
OBS_214 |  -0.6088032 -0.0559165


   Total Computation Time: 0
```

2. Random Forest Regression: Ozone Data: Subset: nobs=40, nvar=12

```
   options NOECHO;
   ozon2 = [
#include "..\\tdata\\Ozone2.dat"
           ];
   options ECHO;

   ozon1 = shape(ozon2,.,14);
   ozone = ozon1[,2:14];

   cnam = [" V1:13 "];
   ozone = cname(ozone,cnam);
   print "nrow=",nrow(ozone);
   print "ozone[]=",ozone[1:5,];


   clas = [ 1:3 ];
   modl = "4 = 1:3 5:13";
```

The training is done with the `ranfor` function:

```
   optn = [ "ntree"          500 ,
            "outmod"   "ozon.rf",
            "mtry"             4 ,
            "ndsiz"            1 ,
            "prox"               ,
            "impo"               ,  /* compute var importance */
            "outl"               ,  /* compute outliers */
            "intact"             ,  /* compute var interactions */
            "dimsc"            2 ,  /* dimension for MDS */
            "print"            3 ,
            "phist"            1 ,
            "pinit"            3 ];
    < gof,misc,pred,gini,prox,outl,intact > = ranfor(ozone,modl,optn,clas);
```

We now use the model stored on file `"ozon.rf"` to score the same data:

```
optn = [ "prox"               ,
         "impo"               ,  /* compute var importance */
         "outl"               ,
         "intact"             ,  /* compute var interactions */
         "print"           3 ,
         "pobs"               ,
         "phist"           1 ,
         "pinit"           3 ];
< gof,misc,pred,gini,prox,outl,intact >
         = rafprd("ozon.rf",ozone,modl,optn,clas);
```

Random Forest Regression

Number Observations	203
Number Variables.	12
Number of Trees	500
Number Variables Selected per Node.	4
Minimum Node Size	1
Maximum Node Size	407
Method Proximity Matrix	1
Number Missing Values	0
Seed for Random Generator	32873

****
Model Information
****

Number Valid Observations	203
Response Variable	Y[4]
N Independend Variables	12

****
Model Effects
****

```
C1  + C2  + C3  + X5  + X6  + X7  + X8  + X9  + X10 + X11 + X12 +
                          X13
```

****
Class Level Information
****

Class	Level	Value

C[1]	12	1	2	3	4	5
		6	7	8	9	10
		11	12			
C[2]	31	1	2	3	4	5
		6	7	8	9	10
		11	12	13	14	15
		16	17	18	19	20
		21	22	23	24	25
		26	27	28	29	30
		31				
C[3]	5	1	2	3	4	5

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[4]	203	11.374384	8.1900528	0.9684100	0.0979959
X[5]	203	5746.1576	113.02771	-0.9570358	1.2589801
X[6]	203	4.8669951	2.1054019	0.0968375	0.4182790
X[7]	203	57.610837	20.847662	-0.6986799	-0.6793142
X[8]	203	61.113300	14.206468	0.0062592	-0.4718326
X[9]	203	56.542956	11.742668	-0.1033238	-0.4965024
X[10]	203	2601.7488	1859.8894	0.2372582	-1.6365634
X[11]	203	14.428571	36.317202	-0.1329615	-0.6701010
X[12]	203	60.692709	14.124734	-0.1900330	-0.6312401
X[13]	203	122.19704	81.171317	0.8127794	-0.0688468

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

Variable	Value	Nobs	Proportion
C[1]	1	17	8.374384
	2	17	8.374384
	3	21	10.344828
	4	21	10.344828
	5	10	4.926108
	6	17	8.374384
	7	13	6.403941
	8	15	7.389163
	9	16	7.881773
	10	18	8.866995
	11	17	8.374384

```
                    12      21   10.344828
C[ 2]                1       5    2.463054
.....................................

C[ 3]                1      37   18.226601
                     2      45   22.167488
                     3      43   21.182266
                     4      36   17.733990
                     5      42   20.689655
```

Skipping some of the output we show the scoring history across the 100 trees:

```
  Tree Nodes  MSQ_Error   Err/Yvar

     1   301  54.009852  0.8091784
     2   295  25.338670  0.3796252
     3   299  15.773399  0.2363179
     4   273  13.258005  0.1986321
     5   301  11.831133  0.1772546
   ..............................
   495   299  8.1432909  0.1220032
   496   279  8.1453933  0.1220347
   497   275  8.1539960  0.1221636
   498   293  8.1612523  0.1222723
   499   307  8.1653113  0.1223331
   500   275  8.1573327  0.1222136


Final Error: 8.15733 (Validation)


  Predicted Values (Validation)

Obs        Data        Pred      Nout

  1  5.00000000  6.68000000       500
  2  6.00000000  6.66400000       500
  3  4.00000000  5.57800000       500
  4  4.00000000  5.56600000       500
  5  6.00000000  7.10200000       500
  ..................................
200  2.00000000  5.43200000       500
201  3.00000000  4.70000000       500
202  5.00000000  5.78400000       500
203  1.00000000  3.25000000       500
```

Imputation, Gini, and Variable Importance

Var	FilMis	GiniC	Z-Score	Signif
1	2.000000000	7.530477376	9.477428571	0.451891238
2	8.000000000	4.149524302	10.87394089	0.352428590
3	1.000000000	0.203899742	1.874640394	0.178103067
4	5760.000000	6.053438993	4.816600985	0.413696700
5	5.000000000	0.114483045	1.227162562	0.145117831
6	63.00000000	1.727151027	4.403438424	0.320661651
7	61.00000000	25.09305200	18.36463054	0.945616689
8	56.30000000	30.76396966	25.12057143	1.046950725
9	2047.000000	4.925102752	5.380768473	0.391242503
10	18.00000000	0.959085425	4.402768473	0.291867835
11	60.80000000	16.32288698	14.16638424	0.789767268
12	100.0000000	2.156928704	3.665586207	0.347586723

Largest (Offdiagonal) Proximity Values

Row	Column	Value
37	36	0.246000000
201	192	0.214000000
103	102	0.202000000
187	184	0.198000000
174	173	0.198000000
192	191	0.196000000
131	130	0.178000000
12	8	0.174000000
176	168	0.172000000
74	73	0.168000000

Sparsity of Proximity Matrix: 52.5548 % (9824 Nonzeros)

Pairwise Variable Interactions

****

Dense Symmetric Matrix (12 by 12)

S	1	2	3	4	5
1	0.0000000				
2	-2.0000000	0.0000000			
3	-17.000000	-9.0000000	0.0000000		

```
 4 |  -24.000000  3.0000000 -19.000000  0.0000000
 5 |  -16.000000 -6.0000000 -15.000000 -10.000000  0.0000000
 6 |   6.0000000 -20.000000 -11.000000  6.0000000 -11.000000
 7 |  -3.0000000 -29.000000 -15.000000  5.0000000 -15.000000
 8 |   15.000000 -20.000000 -17.000000  11.000000 -11.000000
 9 |  -7.0000000 -11.000000  1.0000000  7.0000000 -14.000000
10 |   1.0000000 -16.000000  2.0000000 -5.0000000 -10.000000
11 |  -19.000000 -11.000000 -6.0000000  15.000000 -12.000000
12 |  -1.0000000 -16.000000 -8.0000000 -4.0000000 -14.000000


 S |           6          7          8          9         10
 ------------------------------------------------------------
 6 |   0.0000000
 7 |  -16.000000  0.0000000
 8 |  -13.000000  26.000000  0.0000000
 9 |  -9.0000000 -8.0000000 -25.000000  0.0000000
10 |   21.000000  4.0000000 -20.000000  4.0000000  0.0000000
11 |   10.000000  27.000000  8.0000000  30.000000 -27.000000
12 |   3.0000000 -1.0000000  2.0000000  30.000000  1.0000000


 S |          11         12
 --------------------------
11 |   0.0000000
12 |  -27.000000  0.0000000
```

Note, processing the 500 trees is and computing all statistics is time consuming:

```
Total Computation Time: 6
```

5.11 Function `spawn`

rc = spawn(path,args<,optn>)

Purpose: The `spawn` function executes a child process of an executable `path`.

Input: path

args This must be either a string scalar or a vector of strings. By convention the first argument must be the name of the of the program to be executed.

optn This optional argument must be a string scalar

wait The invoked program is loaded into available memory, is executed, and then the original program resumes execution. This is the default.

conc Causes the current program to execute concurrently with the new child process.

over The invoked program replaces the original program in memory and is executed. No return is made to the original program.

Output: The function only returns an integer indicating the correct (rc=0) or failed termination of the child process. All other output by the child process should be in the form of files.

Restrictions:

1. The executable which `path` refers to must be available.
2. For old DOS (not Windows NT) the length of the concatenated strings cannot exceed 128 characters.

Relationships: system(), help()

Examples: 1. The following two inputs are equivalent:

```
print "The following is equivalent";
irc = spawn("..\\..\\gnuplot\\bin\\wgnuplot_pipes.exe",
            "wgnuplot_pipes.exe","..\\tgplt\\gnuplt.gp",
            "wait");


print "Gnuplot Example with pause";
gpbatch("..\\tgplt\\gnuplt.gp");
```

5.12 Function `survcurv`

< gof,curv,dase > = survcurv(sopt,data,modl,optn<,clas>)

Purpose: The `survcurv` function estimates the function values of survival curves for ng groups of data. Following different models are implemented:

- Aalen's additive model: the groups of observations are defined by a data column which is specified with the `"group"` options argument. (There can be no `"strata"` option for Aalen's model.)
- Cox's proportional hazards model: there are two forms, one for the stratified PH model and one for the simply grouped model, i.e. you may specify either the `"strata"` or the `"group"` column with the options argument.

Many of the options of the `survreg()` function, especially those for the optimization technique, are valid for Cox' PH model since the algorithm executes internally a specific form of this function. (For Aalen's model such specifications are ignored.)

Input: **sopt** This is a string option specifying the regression method (model) which is applied to the data:

"aalen" Aalen's additive model

"phcox" Cox's proportional hazards model

data This is a $N \times nc$ data set which contains in its nc columns:

- the dependent time (stop) variable
- the binary censor variable
- a set of n independent covariate variables
- an optional start variable
- an optional strata variable (for groups of observations)
- an optional offset variable
- an optional weight variable

modl : The analysis model is specified in form of a string, e.g. `model= "3=1 2"`, containing column numbers for variables. The syntax of the `model` string argument is the same as for the `glmod()` function except for the additional *events / trial* response specification. **????**

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. The option statement is not optional, since at least the columns of the censor variable and the group or strata variable must be specified.

class specifies which of the columns of the input data matrix **X** are nominally scaled CLASS variables. (This argument can be missing value.)

Option	Second Column	Meaning
`"alpha"`	real	probability for confidence intervals, default=.05
`"absftol"`	real	absolute function criterion for terminating see function `nlp()`
`"absgtol"`	real	absolute gradient criterion for terminating see function `nlp()`
`"absxtol"`	real	absolute x criterion for terminating see function `nlp()`
`"cens"`	int	column number of census variable
`"fda"`	'f' or 'c'	use finite differences for gradient and Hessian
`"fdg"`	'f' or 'c'	use finite differences for gradient
`"fdh"`	'f' or 'c'	use finite differences for Hessian
`"ftol"`	real	relative function criterion for terminating see function `nlp()`
`"group"`	int	column number of group variable
`"gtol"`	real	relative gradient criterion for terminating see function `nlp()`
`"lsmet"`	string	solving the LS problem for Aalen's model
	"crt"	using the $\mathbf{X}^T\mathbf{X}matrix$ (def)
	"svd"	using the `svd()` method
`"maxf"`	int	maximum number function calls see function `nlp()`
`"maxi"`	int	maximum number iterations see function `nlp()`
`"offset"`	int	column number of offset variable
`"orig"`		use TTs optimization algorithm
`"phist"`	int	print iteration history (def=0)
`"phmet"`	string	estimation method
	"bresl"	Breslow's treatment of ties
	"efron"	Efron's treatment of ties
	"exact"	exact treatment of ties

Option	Second Column	Meaning
"pinit"	int	printed output of input (def=0)
"print"	int	amount of printed output (def=0)
"psurv"		print survival curves and ASE's
"robcov"		use robust covariance matrix
"seed"	int	seed for random generator
"sing"	real	singularity criterion, default=10^{-8}
"stat"	int	column number of census variable
"start"	int	column number of start time variable
"strat"	int	column number of strata variable
		(not valid for Aalen's model)
"techop"	string	optimization technique
		see function `nlp()`
"weight"	int	column number of weight variable
"wgt"	int	column number of weight variable
"xtol"	real	relative x criterion for terminating
		see function `nlp()`

Output: Assuming there are Nev observations with events (for which the binary census variable is 1) and ng groups of observations (indicated either by the group or strata ID variable).

gof this is vector of some scalar results, see below for content.

curv contains the $Nev \times nc1$ matrix with survival curves and its asymptotic standard errors. The $nc1$ columns are:

1. the time value of the event (x coordinate in the (x, y) plot)
2. the corresponding row number in the input data
3. the rank of the linear system (valid only for Aalen's model)
4. ng columns for the curve function, i.e. y values of the (x, y) plot
5. ng columns for the asymptotic standard errors of the curve values

dase contains the $Nev \times nc2$ matrix with asymptotic standard errors of the $nc2$ pairwise differences of the survival curves. The $nng = n * (ng - 1)/2$ columns correspond to the pairs:

$$(1, 2), ...(1, ng), (2, 3), ..., (2, ng), ..., (ng - 1, ng)$$

The content of `gof` is:

1.

Restrictions:
1. For Aalen's model, no strata can be specified.
2.

Relationships: survreg()

Examples:
1. The following data are being used: The data have unique event times.

```
sarcom = [ 1 0 1  153, 1 0 1  945, 1 0 1 1400, 1 0 1 1887, 1 0 0 2557,
           1 0 0 3134, 1 0 0 3226, 1 0 0 3348, 1 0 0 3501, 1 0 0 3743,
           1 1 1  151, 1 1 1  152, 1 1 1  212, 1 1 1  214, 1 1 1  242,
           1 1 1  243, 1 1 1  244, 1 1 1  245, 1 1 1  249, 1 1 1  273,
           1 1 1  336, 1 1 1  337, 1 1 1  396, 1 1 1  427, 1 1 1  457,
           1 1 1  761, 1 1 1 1249, 1 1 1 1310, 1 1 0 2708, 2 0 1   31,
           2 0 1  335, 2 0 1  366, 2 0 1  426, 2 0 1  456, 2 0 1  578,
           2 0 1  589, 2 0 0  762, 2 0 1  792, 2 0 0  913, 2 0 0  914,
           2 0 1  974, 2 0 1 1005, 2 0 1 1035, 2 0 0 1065, 2 0 1 1096,
           2 0 0 1107, 2 0 0 1219, 2 0 0 1250, 2 0 0 1312, 2 0 0 1403,
           2 0 0 1461, 2 0 0 1553, 2 0 0 1645, 2 0 0 1706, 2 0 0 1734,
           2 0 0 1826, 2 0 0 1948, 2 0 0 1949, 2 0 0 1979, 2 0 0 2222,
           2 0 0 2374, 2 0 0 2435, 2 0 0 2465, 2 0 0 2526, 2 1 1    0,
           2 1 1   91, 2 1 1  183, 2 1 1  334, 2 1 1  338, 2 1 1  365,
           2 1 1  391, 2 1 1  518, 2 1 1  547, 2 1 1  608, 2 1 1  609,
           2 1 1  651 ];
  cnam = [" group ldh event time "];
  sarcom = cname(sarcom,cnam);
```

(a) PHCOX Stratified Model:

```
modl = "4 = 2";
optn = [ "status"         3 ,
         "strata"         1 ,
         "pinit"          2 ,
         "psurv"            ,
         "phist"          1 ,
         "print"          3 ];
< gof,curv,dase > = survcurv("phcox",sarcom,modl,optn);
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;
```

```
      Survival Curves by Cox PH Model (Stratified: Efron)
                  Time Data: Right Censored

     Number Rows of Training Data. . . . . .          76
     Number Columns of Matrix  . . . . . .             4
     Number Training Observations. . . . . .          76
     Number Independent Covariates . . . . .           1
     Number (Train) Strata . . . . . .                 2
     Number (Train) Events . . . . . .                46
     Number Unique Time Events . . . . . .            46
     Linear Solver Method. . . . . .                 CRT

            Newton-Raphson Ridge Optimization
```

Without Parameter Scaling
Internally Specified Gradient
Internally Specified Hessian

Iteration Start:

N. Variables	1		
Criterion	147.3410167	Max Grad Entry	16.90946472

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	129.0962	18.24486	1.02554	0	0.92053
2	0	3	0	129.0155	0.080670	0.02970	0	0.98057
3	0	4	0	129.0154	6.5e-005	2e-005	0	1.00047
4	0	5	0	129.0154	3.3e-011	1e-011	0	0.99927

Successful Termination After 4 Iterations
GCONV convergence criterion satisfied.

Criterion	129.0154183	Max Grad Entry	1.0710e-011
Ridge (lambda)	0	Act.dF/Pred.dF	0.999269124
N. Function Calls	6	N. Gradient Calls	5
N. Hessian Calls	6	Preproces. Time	0
Time for Method	0	Effective Time	0

Survival Curves and Asymptotic Standard Errors
Stratified PH Model

N	EventTime	Curv1	ASE_1	Curv2	ASE_2
1	0	1.00000000	0	0.97101858	0.02821614
2	31.00000000	1.00000000	0	0.94179151	0.03896484
3	91.00000000	1.00000000	0	0.91400357	0.04565551
4	151.0000000	0.97693348	0.02258343	0.91400357	0.04565551
5	152.0000000	0.95371485	0.03135041	0.91400357	0.04565551
6	153.0000000	0.93032833	0.03766660	0.91400357	0.04565551
7	183.0000000	0.93032833	0.03766660	0.88593458	0.05087544
8	212.0000000	0.90789026	0.04214453	0.88593458	0.05087544
9	214.0000000	0.88526463	0.04585838	0.88593458	0.05087544
10	242.0000000	0.86242942	0.04898681	0.88593458	0.05087544
	.....				
40	1005.000000	0.54927861	0.06447870	0.42315698	0.05154254
41	1035.000000	0.54927861	0.06447870	0.40006982	0.05100989
42	1096.000000	0.54927861	0.06447870	0.37748346	0.05089957
43	1249.000000	0.52057597	0.06461041	0.37748346	0.05089957
44	1310.000000	0.48778793	0.06569888	0.37748346	0.05089957
45	1400.000000	0.44699676	0.06849861	0.37748346	0.05089957

```
46 1887.000000  0.41189082  0.07038976  0.37748346  0.05089957

      Asymptotic Standard Errors of Pairwise Differences

                          N      DASE21
                          1  0.02821614
                          2  0.03896484
                          3  0.04565551
                          4  0.05097895
                          5  0.05546492
                          6  0.05930617
                          7  0.06343837
                          8  0.06624596
                          9  0.06872003
                         10  0.07089890
                         ..............
                         40  0.07949721
                         41  0.07863768
                         42  0.07810133
                         43  0.07769785
                         44  0.07806150
                         45  0.07995940
                         46  0.08098638
```

(b) PHCOX UNstratified Model:

```
modl = "4 = 2";
optn = [ "status"          3 ,
         "group"           1 ,
         "pinit"           2 ,
         "psurv"             ,
         "phist"           1 ,
         "print"           3 ];
< gof,curv,dase > = survcurv("phcox",sarcom,modl,optn);
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;

      Survival Curves by Cox PH Model (Grouped: Efron)
                Time Data: Right Censored

   Number Rows of Training Data. . . . . .          76
   Number Columns of Matrix  . . . . . .             4
   Number Training Observations. . . . . .          76
   Number Independent Covariates . . . . . .         1
   Number (Train) Strata . . . . . .                 2
```

```
  Number (Train) Events . . . . . .          46
  Number Unique Time Events . . . . . .      46
  Linear Solver Method. . . . . .           CRT

           Newton-Raphson Ridge Optimization
               Without Parameter Scaling
             Internally Specified Gradient
             Internally Specified Hessian


 Iteration Start:
 N. Variables             2
 Criterion       179.7007234       Max Grad Entry  19.31170860


Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  160.4187 19.28207 4.21314       0  0.78528
   2    0    3   0  159.3512 1.067464 0.40962       0  0.93437
   3    0    4   0  159.3420 9.2e-003  2e-003       0  1.00373
   4    0    5   0  159.3420 2.9e-007  8e-008       0  1.00002

 Successful Termination After     4 Iterations
 GCONV convergence criterion satisfied.
 Criterion       159.3419643       Max Grad Entry  7.6534e-008
 Ridge (lambda)            0       Act.dF/Pred.dF  1.000022327
 N. Function Calls         6       N. Gradient Calls         5
 N. Hessian Calls          6       Preproces. Time           0
 Time for Method           0       Effective Time            0

       Survival Curves and Asymptotic Standard Errors
                   Unstratified PH Model

   N   EventTime       Curv1       ASE_1       Curv2       ASE_2
   1           0  0.98776219  0.01224809  0.98604848  0.01397378
   2 31.00000000  0.97543547  0.01737710  0.97204043  0.01983589
   3 91.00000000  0.96334316  0.02115831  0.95834305  0.02415444
   4 151.0000000  0.95115816  0.02436833  0.94458561  0.02785930
   5 152.0000000  0.93892440  0.02720867  0.93081908  0.03110024
   6 153.0000000  0.92663923  0.02978878  0.91704163  0.03398658
   7 183.0000000  0.91461168  0.03204762  0.90359926  0.03647234
   8 212.0000000  0.90247858  0.03413763  0.89008578  0.03881049
   9 214.0000000  0.89028732  0.03611129  0.87655552  0.04094344
  10 242.0000000  0.87803440  0.03799023  0.86300592  0.04288505
  ..........................................................
  40 1005.000000  0.48429578  0.06137452  0.45546612  0.04974913
```

```
41 1035.000000  0.47141875  0.06117477  0.44284229  0.04959597
42 1096.000000  0.45888137  0.06104894  0.43054344  0.04957210
43 1249.000000  0.44637889  0.06102344  0.41826154  0.04976186
44 1310.000000  0.43219509  0.06127904  0.40429555  0.05011418
45 1400.000000  0.41548110  0.06192765  0.38777806  0.05063531
46 1887.000000  0.39472011  0.06328377  0.36714809  0.05270673
```

Asymptotic Standard Errors of Pairwise Differences

```
                          N      DASE21
                          1  0.00455514
                          2  0.00870382
                          3  0.01265317
                          4  0.01652685
                          5  0.02029192
                          6  0.02394657
                          7  0.02740397
                          8  0.03078131
                          9  0.03404661
                         10  0.03719788
                         ..............
                         40  0.06914751
                         41  0.06850547
                         42  0.06791152
                         43  0.06737410
                         44  0.06686503
                         45  0.06644190
                         46  0.06623028
```

(c) Aalen's Grouped Model:

```
modl = "4 = 2";
optn = [ "status"          3 ,
         "strata"          1 ,
         "lsmet"       "crt" ,
         "pinit"           2 ,
         "psurv"             ,
         "phist"           1 ,
         "print"           3 ];
< gof,curv,dase > = survcurv("aalen",sarcom,modl,optn);
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;
```

Survival Curves by Aalen's Additive Model
Time Data: Right Censored

Number Rows of Training Data.	76
Number Columns of Matrix	4
Number Training Observations.	76
Number Independent Covariates	1
Number (Train) Strata	2
Number (Train) Events	46
Number Unique Time Events	46
Linear Solver Method.	CRT

Survival Curves and Asymptotic Standard Errors

N	EventTime	Curv1	ASE_1	Curv2	ASE_2
1	0	1.01224213	0.01259761	0.97206332	0.02727306
2	31.00000000	1.00808290	0.01302818	0.95358824	0.03254267
3	91.00000000	1.02155121	0.01958911	0.92556296	0.04150639
4	151.0000000	0.99338172	0.03385780	0.92204027	0.04119764
5	152.0000000	0.96492360	0.04327196	0.91850944	0.04087186
6	153.0000000	0.91954493	0.06035813	0.92489868	0.04226488
7	183.0000000	0.93393737	0.06347897	0.89629665	0.04885998
8	212.0000000	0.90550458	0.06782197	0.89245830	0.04832242
9	214.0000000	0.87673497	0.07164607	0.88860302	0.04776249
10	242.0000000	0.84760121	0.07502662	0.88473485	0.04717792
...					
40	1005.000000	0.57075469	0.12139222	0.44544705	0.06118247
41	1035.000000	0.57075469	0.12139222	0.42565273	0.06158201
42	1096.000000	0.57075469	0.12139222	0.40489341	0.06197823
43	1249.000000	0.55664085	0.12149518	0.39488104	0.05888633
44	1310.000000	0.54260346	0.12151844	0.38492292	0.05576851
45	1400.000000	0.48138847	0.12181123	0.38696680	0.05665218
46	1887.000000	0.42018560	0.12014636	0.38963795	0.05789604

Asymptotic Standard Errors of Pairwise Differences

N	DASE21
1	0.03987067
2	0.04176061
3	0.05813542
4	0.06242868
5	0.06624499
6	0.08232308
7	0.09267043
8	0.09399858
9	0.09516840
10	0.09618682

```
..............
40  0.15544516
41  0.15478913
42  0.15409006
43  0.15125024
44  0.14841946
45  0.14830321
46  0.14692977
```

2. The following data are being used: The event time data are not unique and have ties.

```
  options NOECHO;
#include "..\\tdata\\myeloma1.dat"
  options ECHO;

  cnam = [" Time Vstatus LogBUN HGB Platelet Age LogWBC Frac
           LogPBM Protein SCalc "];
  myeloma1 = cname(myeloma1,cnam);
```

3. PHCOX Stratified Model:

```
 modl = "1 = 3 4 11";
 optn = [ "status"          2 ,
          "strata"          5 ,
          "phmet"   "breslow" ,
          "pinit"           2 ,
          "psurv"             ,
          "phist"           1 ,
          "print"           3 ];
< gof,curv,dase > = survcurv("phcox",myeloma1,modl,optn);
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;
```

```
     Survival Curves by Cox PH Model (Stratified: Breslow)
                  Time Data: Right Censored

    Number Rows of Training Data. . . . . .          65
    Number Columns of Matrix  . . . . . .            11
    Number Training Observations. . . . . .          65
    Number Independent Covariates . . . . . .         3
    Number (Train) Strata . . . . . .                 2
    Number (Train) Events . . . . . .                48
    Number Unique Time Events . . . . . .            31
```

```
  Linear Solver Method. . . . . .          CRT

            Newton-Raphson Ridge Optimization
                Without Parameter Scaling
              Internally Specified Gradient
              Internally Specified Hessian


 Iteration Start:
 N. Variables              3
 Criterion       134.3237580        Max Grad Entry  24.47419208


Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  129.0019 5.321887 1.72979       0  0.92307
   2    0    3   0  128.9474 0.054484 0.01812       0  1.00046
   3    0    4   0  128.9474 2.0e-006  3e-006       0  1.00010

 Successful Termination After     3 Iterations
 ABSGCONV convergence criterion satisfied.
 Criterion       128.9473858        Max Grad Entry  3.0068e-006
 Ridge (lambda)            0        Act.dF/Pred.dF  1.000096850
 N. Function Calls         5        N. Gradient Calls         5
 N. Hessian Calls          5        Preproces. Time           0
 Time for Method           0        Effective Time            0

       Survival Curves and Asymptotic Standard Errors
                    Stratified PH Model

   N   EventTime       Curv1       ASE_1       Curv2       ASE_2
   1 1.250000000  1.00000000           0  0.96379281  0.02485122
   2 2.000000000  0.84786712  0.09543890  0.94355413  0.03115596
   3 3.000000000  0.84786712  0.09543890  0.92312631  0.03615444
   4 5.000000000  0.84786712  0.09543890  0.87815597  0.04537173
   5 6.000000000  0.76819110  0.11200580  0.80968433  0.05473468
   6 7.000000000  0.68335806  0.12581690  0.76658034  0.05815070
   7 9.000000000  0.68335806  0.12581690  0.74292054  0.06003354
   8 11.00000000  0.68335806  0.12581690  0.63190110  0.06433984
   9 13.00000000  0.60369364  0.13319847  0.63190110  0.06433984
  10 14.00000000  0.60369364  0.13319847  0.60782353  0.06504415
 ..............................................................
  25 54.00000000  0.30290149  0.15500969  0.23767411  0.05680451
  26 58.00000000  0.30290149  0.15500969  0.20912368  0.05612389
  27 66.00000000  0.30290149  0.15500969  0.18186667  0.05470034
  28 67.00000000  0.30290149  0.15500969  0.14937586  0.05293671
```

```
29 88.00000000  0.30290149  0.15500969  0.11189865  0.05057417
30 89.00000000  0.30290149  0.15500969  0.07677660  0.04502436
31 92.00000000  0.30290149  0.15500969  0.04603549  0.03648082
```

```
   Asymptotic Standard Errors of Pairwise Differences

                          N      DASE21
                          1  0.02482635
                          2  0.10054299
                          3  0.10226045
                          4  0.10587399
                          5  0.12497045
                          6  0.13933318
                          7  0.14035859
                          8  0.14182193
                          9  0.14865680
                         10  0.14933421
                         ..............
                         25  0.16067266
                         26  0.15957475
                         27  0.15813387
                         28  0.15781800
                         29  0.15702726
                         30  0.15512079
                         31  0.15190661
```

4. PHCOX UNStratified Model:

```
modl = "1 = 3 4 11";
optn = [ "status"          2 ,
         "group"           5 ,
         "phmet"   "breslow" ,
         "pinit"           2 ,
         "psurv"             ,
         "phist"           1 ,
         "print"           3 ];
< gof,curv,dase > = survcurv("phcox",myeloma1,modl,optn);
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;
```

```
      Survival Curves by Cox PH Model (Grouped: Breslow)
                  Time Data: Right Censored
```

Number Rows of Training Data.	65
Number Columns of Matrix	11
Number Training Observations.	65
Number Independent Covariates	3
Number (Train) Strata	2
Number (Train) Events	48
Number Unique Time Events	31
Linear Solver Method.	CRT

Newton-Raphson Ridge Optimization
Without Parameter Scaling
Internally Specified Gradient
Internally Specified Hessian

Iteration Start:			
N. Variables	4		
Criterion	154.8579914	Max Grad Entry	38.41628820

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	148.4967	6.361260	6.29624	0	0.82235
2	0	3	0	148.0385	0.458232	0.40675	0	1.03639
3	0	4	0	148.0366	1.9e-003	3e-003	0	1.00424
4	0	5	0	148.0366	8.0e-008	1e-007	0	1.00003

Successful Termination After	4 Iterations		
GCONV convergence criterion satisfied.			
Criterion	148.0365801	Max Grad Entry	1.3163e-007
Ridge (lambda)	0	Act.dF/Pred.dF	1.000029469
N. Function Calls	6	N. Gradient Calls	5
N. Hessian Calls	6	Preproces. Time	0
Time for Method	0	Effective Time	0

Survival Curves and Asymptotic Standard Errors
Unstratified PH Model

N	EventTime	Curv1	ASE_1	Curv2	ASE_2
1	1.250000000	0.96915705	0.02430031	0.97016679	0.02072382
2	2.000000000	0.92028521	0.04372573	0.92279040	0.03316854
3	3.000000000	0.90354985	0.04963598	0.90653746	0.03631797
4	5.000000000	0.86740524	0.06135344	0.87138274	0.04264279
5	6.000000000	0.79535893	0.08073682	0.80109592	0.05176289
6	7.000000000	0.74294447	0.09339887	0.74978151	0.05566873
7	9.000000000	0.72399824	0.09766183	0.73119545	0.05702671

```
 8 11.00000000  0.63420154  0.11359732  0.64283433  0.06108579
 9 13.00000000  0.61516172  0.11625366  0.62404115  0.06184589
10 14.00000000  0.59495727  0.11939584  0.60407624  0.06245098
.....................................................................
25 54.00000000  0.22366568  0.12907322  0.23275284  0.05666072
26 58.00000000  0.19609288  0.12565262  0.20474637  0.05584722
27 66.00000000  0.16983952  0.12088897  0.17799068  0.05431679
28 67.00000000  0.13863635  0.11199773  0.14605470  0.05243488
29 88.00000000  0.10295821  0.09916756  0.10931000  0.04989635
30 89.00000000  0.06969335  0.08198021  0.07475037  0.04426390
31 92.00000000  0.04075495  0.06156080  0.04433028  0.03557657
```

```
   Asymptotic Standard Errors of Pairwise Differences

                          N      DASE21
                          1  0.01422770
                          2  0.03522336
                          3  0.04199786
                          4  0.05587488
                          5  0.08050455
                          6  0.09590843
                          7  0.10094487
                          8  0.12096176
                          9  0.12438772
                         10  0.12771714
                         ..............
                         25  0.12616287
                         26  0.11993757
                         27  0.11275064
                         28  0.10234372
                         29  0.08726519
                         30  0.06907524
                         31  0.04836435
```

5. Aalen's Model:

```
modl = "1 = 3 4";
optn = [ "status"          2 ,
         "group"           5 ,
         "lsmet"       "crt" ,
         "pinit"           2 ,
         "psurv"             ,
         "phist"           1 ,
         "print"           3 ];
< gof,curv,dase > = survcurv("aalen",myeloma1,modl,optn);
```

```
print "Gof=",gof;
print "Curv=",curv;
print "DASE=",dase;
```

```
          Survival Curves by Aalen's Additive Model
                  Time Data: Right Censored

    Number Rows of Training Data. . . . . .          65
    Number Columns of Matrix  . . . . . .            11
    Number Training Observations. . . . . .          65
    Number Independent Covariates . . . . . .         2
    Number (Train) Strata . . . . . .                 2
    Number (Train) Events . . . . . .                48
    Number Unique Time Events . . . . . .            31
    Linear Solver Method. . . . . .                 CRT
```

The log output shows that 8 of the 31 unique time events have singular coeffient matrices and the curves are shortened correspondingly:

```
[warning] file tsurvcur2.inp, line 104: From 31 unique time events
          8 have singular linear systems.
```

```
        Survival Curves and Asymptotic Standard Errors

    N   EventTime       Curv1       ASE_1       Curv2       ASE_2
    1 1.250000000  0.99726953  0.01739743  0.96790524  0.02089373
    2 2.000000000  0.78960953  0.13977781  0.95213445  0.02818393
    3 3.000000000  0.82567652  0.15139862  0.92823445  0.03532805
    4 5.000000000  0.87435348  0.16772624  0.88167394  0.04244341
    5 6.000000000  0.76545660  0.20348277  0.82452901  0.04992492
    6 7.000000000  0.66336920  0.19783122  0.78451707  0.05251282
    7 9.000000000  0.68689128  0.20628907  0.76044865  0.05328398
    8 11.00000000  0.68087858  0.21368671  0.67066128  0.06202558
    9 13.00000000  0.58015689  0.20441215  0.67420354  0.06524085
   10 14.00000000  0.54376016  0.19369126  0.65842089  0.06593166
   ..............................................................
   20 35.00000000  0.21947675  0.15701178  0.41716302  0.07848692
   21 37.00000000  0.25310581  0.18859996  0.38253879  0.07375257
   22 41.00000000  0.26520032  0.20267762  0.33412381  0.07415039
   23 51.00000000  0.09756175  0.12279099  0.33412381  0.07415039
   24 52.00000000           0           .           0           .
   25 54.00000000           0           .           0           .
   26 58.00000000           0           .           0           .
   27 66.00000000           0           .           0           .
```

28	67.00000000	0	.	0	.
29	88.00000000	0	.	0	.
30	89.00000000	0	.	0	.
31	92.00000000	0	.	0	.

Asymptotic Standard Errors of Pairwise Differences

N	DASE21
1	0.02822760
2	0.14771687
3	0.16510944
4	0.18860357
5	0.22630188
6	0.21780285
7	0.22880719
8	0.23781090
9	0.22895554
10	0.21700051
.....	
20	0.18662062
21	0.21438860
22	0.23075850
23	0.15175890
24	.
25	.
26	.
27	.
28	.
29	.
30	.
31	.

5.13 Function `survreg`

< gof,parm,cov,res,tres > = survreg(sopt,data,modl,optn<,clas<,xini<,test> . >)

Purpose: The `survreg` function estimates different methods for survival data which contain one or two time and a binary censor variable.

- Aalen's additive model
- Cox's proportional hazards model
- GLIM-like: extreme distribution
- GLIM-like: logistic distribution
- GLIM-like: Gaussian distribution
- GLIM-like: Weibull distribution (log(extreme))
- GLIM-like: loglogistic distribution (log(logistic))
- GLIM-like: lognormal distribution (log(Gaussian))
- GLIM-like: exponential distribution (this is Weibull distribution for $scale = 1$)
- GLIM-like: Rayleigh distribution (this is Weibull distribution for $scale = .5$)

The implementation is similar to the `survival` package by Terry Therneau (2000) in R. Similar models are estimated by the SAS PROCs PHREG and LIFEREG. Note that Aalen's method has an analytical solution and does not need a general optimization technique for the estimation of the parameters.

Input: **sopt** This is a string option specifying the regression method (model) which is applied to the data:

"aalen" Aalen's additive model

"phcox" Cox's proportional hazards model

"extre" extreme distribution

"logis" logistic distribution

"gauss" Gaussian distribution

"weibu" Weibull distribution

"loglo" loglogistic distribution

"logno" lognormal distribution

"expon" exponential distribution (Weibull with fixed scale of 1)

"rayle" Rayleigh distribution (Weibull with fixed scale of .5)

data This is a $N \times nc$ data set which contains in its nc columns:

- the dependent time (stop) variable
- the binary censor variable

- a set of n independent covariate variables
- an optional start variable
- an optional strata variable (for groups of observations)
- an optional offset variable
- an optional weight variable

modl : The analysis model is specified in form of a string, e.g. `model= "3=1 2"`, containing column numbers for variables. The syntax of the `model` string argument is the same as for the `glmod()` function except for the additional *events / trial* response specification. **????**

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. The option statement is not optional, since at least the column of the censor variable must be specified.

class specifies which of the columns of the input data matrix **X** are nominally scaled CLASS variables. (This argument can be missing value.)

xini specifies a vector of starting values for the parameter estimation

test An optional test data set with the same column structure can be "scored". Input data, residuals and predicted values are returned in the `tres` output argument. Currently, test set scoring is not implemented with Aalen's method.

Option	Second Column	Meaning
`"alpha"`	real	probability for confidence intervals, default=.05
`"absftol"`	real	absolute function criterion for terminating see function `nlp()`
`"absgtol"`	real	absolute gradient criterion for terminating see function `nlp()`
`"absxtol"`	real	absolute x criterion for terminating see function `nlp()`
`"cens"`	int	column number of census variable
`"estsc"`		estimate scale parameter (valid only for GLIM-like models)
`"fda"`	'f' or 'c'	use finite differences for gradient and Hessian
`"fdg"`	'f' or 'c'	use finite differences for gradient
`"fdh"`	'f' or 'c'	use finite differences for Hessian
`"ftol"`	real	relative function criterion for terminating see function `nlp()`
`"gtol"`	real	relative gradient criterion for terminating see function `nlp()`
`"maxf"`	int	maximum number function calls see function `nlp()`
`"maxi"`	int	maximum number iterations see function `nlp()`
`"nowgtr"`		do not weight residuals
`"nopht"`		do not perform proportional hazards test (valid only for `"phcox"`)
`"offset"`	int	column number of offset variable
`"orig"`		use TTs optimization algorithm
`"phist"`	int	print iteration history (def=0)
`"phmet"`	string	estimation method (only for `"phcox"`)
	"bresl"	Breslow's treatment of ties
	"efron"	Efron's treatment of ties
	"exact"	exact treatment of ties
`"phtest"`	string	transformation for PH test (valid only for `"phcox"`)
	"km"	Kaplan-Meier transform
	"lo"	log transform
	"ra"	rank transform
	"id"	identity transform

Option	Second Column	Meaning
"pinit"	int	printed output of input (def=0)
"print"	int	amount of printed output (def=0)
"ptyp"	string	type of predicted values (see below)
"robcov"		use robust covariance matrix
"rtyp"	string	type of residuals (see below)
"scal"	real	set scale (fixed or initial value)
		(valid only for GLIM-like models)
"seed"	int	seed for random generator
"sing"	real	singularity criterion, default=10^{-8}
"stat"	int	column number of census variable
"start"	int	column number of start time variable
"strat"	int	column number of strata variable
		(not valid for Aalen's model)
"techop"	string	optimization technique
		see function `nlp()`
"weight"	int	column number of weight variable
"wgt"	int	column number of weight variable
"xtol"	real	relative x criterion for terminating
		see function `nlp()`
"ytype"	string	form of census variable
	"right"	right censored
	"left"	right censored
	"right"	right censored
	"count"	count censored

Note, for Exponential and Rayleigh distribution you cannot use the "estsc" option or "scale" values different from its definitions (1 for exponential and .5 for Rayleigh).

Types of Predicted Values: "ptyp"

PH Cox Estimation	
"te"	terms of linear predictors (def.)
"lp"	linear predictors
"ri"	risk score
"ex"	expectated number of events
GLIM Estimation	
"re"	response (def.) : [N,1]
"lp"	linear predictors : [N,1]
"te"	terms of linear predictors : [N,n-1]
"qu"	quantile predictors: [N,nq]
"uq"	Uquantile predictors: [N,nq]

Types of Residuals: `"rtyp"`

PH Cox Estimation	
"mart"	Martingale residuals (def) : [N,1]
"devi"	deviance residuals : [N,1]
"part"	partial residuals : [N,1]
"dfbe"	DFBE residuals : [N,n]
"dfbs"	DFBS residuals : [N,n]
"scor"	score residuals : [N,n]
"scho"	Schoenfeld residuals : [N,n]
"scas"	scaled Schoenfeld residuals : [N,n]
GLIM Estimation	
"resp"	response residuals (def.) : [N,1]
"devi"	deviance residuals : [N,1]
"work"	working residuals : [N,1]
"dfbe"	DFBE residuals : [N,p]
"dfbs"	DFBS residuals : [N,p]
"ldca"	LD case residuals : [N,1]
"ldre"	LD response residuals : [N,1]
"ldsh"	LD shape residuals : [N,1]

1. for `"aalen"` and `"phcox"`:
   - setting `scal` has no impact.
   - `"ytyp"` can only be `"right"` (with only stop time) and `"count"` (with input start and stop time)
2. for GLIM-like models:
   - For `"expon"` and `"rayle"`, the scale must be fixed (1.0 and .5).
   - For stratified GLIM-like models, the scale is not being used.
   - For each strata an additional parameter is estimated.
   - `"ytyp"` cannot be `"count"`; for `"right"` censoring only stop time, for `"left"` censored data only start time, for interval censored data both start and stop time can be specified.

Output: **gof** this is vector of some scalar results, see below for content.

parm contains the n optimal parameters β_j for each of the independent covariates, together with their risk ratios ($exp(\beta_j)$), z and p values, and asymptotic standard error.

cov contains the $n \times n$ covariance matrix of parameter estimates.

res contains the input data, predicted values, and residuals. Currently for Aalen's method predicted values and residuals are not computed and returned.

tres contains predicted values and residuals of a specified test data set. Currently, test set scoring is not implemented with Aalen's method which means that this return is missing.

The content of `gof` is:

1.

Restrictions: 1. Currently there are no residuals or predicted values for Aalen's method.

2. For GLIM estimation you can have stratified models only together with scale estimation. For k strata there are k additional weight parameters to estimate. A specified `"scale"` value is used as starting estimate for the weights.

Relationships: reg(), glim(), survfor(), survcurv()

Examples: 1. PH Cox Estimation: Ovarian Cancer Data: UNStratified Example: nvar=4:

```
print "Ovarian Data: UNStratified Example: nvar=4";
ovar = [ 1      59      1 72.3315        2  1       1,
         2     115      1 74.4932        2  1       1,
         3     156      1 66.4658        2  1       2,
         ..........................................
         24    353      1 63.2192        1  2       2,
         25    365      1 64.4247        2  2       1,
         26    377      0 58.3096        1  2       1 ];

  cnam = [" N futime fustat age resid_ds rx ecog_ps "];
  ovar = cname(ovar,cnam);
  nr = nrow(ovar); nc = ncol(ovar);
  print "Interval Data: nr=",nr," nc=",nc;
  print ovar[1:10,];


  modl = "2 = 4:7";
  optn = [ "status"       3 ,
           "phmet"  "efron" ,
           "rtyp"    "devi" ,
           "pinit"        2 ,
           "phist"        1 ,
           "print"        3 ];
< gof,parm,covm,resi > = survreg("phcox",ovar,modl,optn);
print "Gof=",gof;
print "Parm=",parm;
print "Covm=",covm;
print "Resi=",resi;
```

```
     Cox Proportional Hazards Regression (Method by Efron)
                  Time Data: Right Censored

    Number Rows of Training Data. . . . . .          26
```

```
  Number Columns of Matrix  . . . . . .          7
  Number Training Observations. . . . . .        26
  Number Independent Covariates . . . . . .       4
  Number (Train) Events . . . . . .              12
  Seed for Random Generator . . . . . .         234
```

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[2]	26	599.53846	339.68847	0.4394487	-0.7630457
X[4]	26	56.165442	10.100361	0.0461065	-0.5306959
X[5]	26	1.5769231	0.5038315	-0.3307984	-2.0553360
X[6]	26	1.5000000	0.5099020	0	-2.1739130
X[7]	26	1.4615385	0.5083911	0.1639158	-2.1447981

Specific Training Data

Nrow	Nobs	Status	Time
1	1	1	59.000000
2	2	1	115.00000
3	3	1	156.00000
.....			
24	24	1	353.00000
25	25	1	365.00000
26	26	0	377.00000

Training Covariates

	age	resid_ds	rx	ecog_ps
OBS_01	72.33150000	2.000000000	1.000000000	1.000000000
OBS_02	74.49320000	2.000000000	1.000000000	1.000000000
OBS_03	66.46580000	2.000000000	1.000000000	2.000000000
.....				
OBS_24	63.21920000	1.000000000	2.000000000	2.000000000
OBS_25	64.42470000	2.000000000	2.000000000	1.000000000
OBS_26	58.30960000	1.000000000	2.000000000	1.000000000

Newton-Raphson Ridge Optimization
Without Parameter Scaling
Internally Specified Gradient
Internally Specified Hessian

```
 Iteration Start:
 N. Variables              4
 Criterion       34.98494037        Max Grad Entry  100.2389003


Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  27.84642 7.138518 16.5039       0  0.68597
   2    0    3   0  26.49962 1.346804 2.36532       0  0.87613
   3    0    4   0  26.46330 0.036322 0.01929       0  1.00377
   4    0    5   0  26.46329 2.6e-006  7e-006       0  1.00006

 Successful Termination After     4 Iterations
 ABSGCONV convergence criterion satisfied.
 Criterion       26.46329352        Max Grad Entry  7.2365e-006
 Ridge (lambda)            0        Act.dF/Pred.dF  1.000064104
 N. Function Calls         6        N. Gradient Calls         5
 N. Hessian Calls          6        Preproces. Time           0
 Time for Method           0        Effective Time            0


             Test Statistics for Training Data

     -2 Log L: Without: 69.9699 With Covariates: 52.9266
        R_squared: 0.818106 Max(R_squared): 0.999085
     Likelihood Ratio Test: 17.0433 df=4 Prob=0.00189587
     Score (logrank) Test: 20.8128 df=4 Prob=0.000344912
           Wald Test: 14.2501 df=4 Prob=0.00653826
                   AIC=60.9266 SBC=62.1369

 Global PH Test (Kaplan-Meier): ChiSqu=3.36086 Prob=0.499348


            Coefficients, ASEs, Z, and P Values

     Variable        Coef   AsyStdErr     Z_Value     P_Value
          age  0.12481307  0.04689036  2.66180652  0.00777225
     resid_ds  0.82618635  0.78961103  1.04632069  0.29541300
           rx -0.91449982  0.65331561 -1.39978259  0.16157843
      ecog_ps  0.33621161  0.64392305  0.52213011  0.60157974


          Risk (exp(coef)) and Confidence Intervals

         Var       Coef  Exp(Coef)   CI_Lower   CI_Upper
           1  0.1248131  1.1329367  1.0334572  1.2419919
```

```
          2  0.8261864  2.2845895  0.4860602  10.738072
          3 -0.9144998  0.4007170  0.1113614  1.4419196
          4  0.3362116  1.3996352  0.3961928  4.9445087


      Proportional Hazards Test (Kaplan-Meier Transform)

          Variable         Rho      ChiSqu     P_Value
               age -0.03988567  0.02617998  0.87146154
         resid_ds -0.14168273  0.24625825  0.61972217
               rx  0.13246201  0.20011980  0.65462416
          ecog_ps  0.48447739  1.88192366  0.17011567


          Covariance Matrix of Parameter Estimates
          ****************************************

              Dense Symmetric Matrix (4 by 4)

          S |         age   resid_ds         rx    ecog_ps
          ----------------------------------------------
        age |   0.0021987
   resid_ds |  -0.0132674  0.6234856
         rx |   0.0051495 -0.0939057  0.4268213
    ecog_ps |  -0.0038605  0.1554163 -0.0862510  0.4146369


       Training Data, Predicted Values, and Residuals
       **********************************************

                   Dense Matrix (26 by 8)

         |       Event      Start       Time       LinP  P_LPterms
    -------------------------------------------------------------
OBS_01 |   1.0000000          0  59.000000  2.6693509  2.6693509
OBS_02 |   1.0000000          0  115.00000  2.9391594  2.9391594
OBS_03 |   1.0000000          0  156.00000  2.2734466  2.2734466
.............................................................
OBS_24 |   1.0000000          0  353.00000  0.1275423  0.1275423
OBS_25 |   1.0000000          0  365.00000  0.7679792  0.7679792
OBS_26 |          0          0  377.00000 -0.8214516 -0.8214516


         |       P_ASE   R_Martin R_Deviance
    -----------------------------------
OBS_01 |   0.7831646  0.8410328  1.4128160
OBS_02 |   0.8657932  0.5442439  0.6950591
```

```
OBS_03 |   0.7192268  0.5967083  0.7891600
.....................................
OBS_24 |   0.7549817  0.8211167  1.3415700
OBS_25 |   0.6125701  0.5513655  0.7073631
OBS_26 |   0.8005724 -0.0915402 -0.4278789


                  Total Computation Time: 0
```

The score residuals:

```
       |       P_ASE   R_Martin  R_Score_1  R_Score_2  R_Score_3
     ------------------------------------------------------------
OBS_01 |   0.7831646  0.8410328  2.2650331  0.0568636 -0.1056538
OBS_02 |   0.8657932  0.5442439  3.0252549  0.0464131 -0.0862366
OBS_03 |   0.7192268  0.5967083 -0.0685129  0.0713143 -0.1325036
.............................................................
OBS_24 |   0.7322996  0.8211167  3.9608430 -0.5656692  0.3464895
OBS_25 |   0.7684894  0.5513655  4.3002574  0.1524126  0.2241752
OBS_26 |   0.6536023 -0.0915402  0.3149064  0.0709177 -0.0521220


       |   R_Score_4
     ---------------
OBS_01 |  -0.4266169
OBS_02 |  -0.3482128
OBS_03 |   0.0616753
..................
OBS_24 |   0.4490741
OBS_25 |  -0.2039044
OBS_26 |   0.0484562
```

Schoenfeld residuals are nonmissing only for events:

```
       |       P_ASE   R_Martin  R_Schoe_1  R_Schoe_2  R_Schoe_3
     ------------------------------------------------------------
OBS_01 |   0.7831646  0.8410328  2.6931568  0.0676116 -0.1256239
OBS_02 |   0.8657932  0.5442439  5.3639019  0.0803912 -0.1493686
OBS_03 |   0.7192268  0.5967083 -0.8987740  0.1068399 -0.1985109
.............................................................
OBS_24 |   0.7322996  0.8211167  4.0610442 -0.7058765  0.4535120
OBS_25 |   0.7684894  0.5513655  5.5003587  0.2534827  0.4796229
OBS_26 |   0.6536023 -0.0915402     .          .          .


       |   R_Schoe_4
```

```
    ---------------
OBS_01 |  -0.5072536
OBS_02 |  -0.6031317
OBS_03 |   0.1984379
.....................
OBS_24 |   0.5282024
OBS_25 |  -0.4413864
OBS_26 |      .
```

2. PH Cox Estimation: Ovarian Data: Stratified Example: nvar=2:

```
 modl = "2 = 4 7";
 optn = [ "strat"        6 ,
          "status"       3 ,
          "phmet"  "efron" ,
          "pinit"        2 ,
          "phist"        1 ,
          "print"        3 ];
< gof,parm,covm,resi > = survreg("phcox",ovar,modl,optn);
print "Gof=",gof;
print "Parm=",parm;
print "Covm=",covm;
print "Resi=",resi;
```

```
     Cox Proportional Hazards Regression (Method by Efron)
                  Time Data: Right Censored

    Number Rows of Training Data. . . . . .          26
    Number Columns of Matrix  . . . . . .             7
    Number Training Observations. . . . . .          26
    Number Independent Covariates . . . . .           2
    Number (Train) Strata . . . . . .                 2
    Number (Train) Events . . . . . .                12
    Seed for Random Generator . . . . . .           234


                    Specific Training Data

                Nrow  Nobs Strata Status      Time
                   1     1      0      1 59.000000
                   2     2      0      1 115.00000
                   3     3      0      1 156.00000
                   ..............................
                  24    24      1      1 353.00000
                  25    25      1      1 365.00000
```

```
    26    26      1      0 377.00000
```

Strata Frequencies for Training Data

Strata	Freqency	CumFreq	Nevents
0	13	13	7
1	13	26	5

Training Covariates

	age	ecog_ps
OBS_01	72.33150000	1.000000000
OBS_02	74.49320000	1.000000000
OBS_03	66.46580000	2.000000000
.....	.....	.....
OBS_24	63.21920000	2.000000000
OBS_25	64.42470000	1.000000000
OBS_26	58.30960000	1.000000000

Newton-Raphson Ridge Optimization
Without Parameter Scaling
Internally Specified Gradient
Internally Specified Hessian

Iteration Start:			
N. Variables	2		
Criterion	26.90887268	Max Grad Entry	104.6791318

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	20.65629	6.252583	10.2205	0	1.02140
2	0	3	0	20.55354	0.102745	0.87744	0	1.05724
3	0	4	0	20.55269	8.5e-004	8e-003	0	1.00641
4	0	5	0	20.55269	8.0e-008	8e-007	0	1.00006

Successful Termination After	4 Iterations		
GCONV convergence criterion satisfied.			
Criterion	20.55268978	Max Grad Entry	7.9862e-007
Ridge (lambda)	0	Act.dF/Pred.dF	1.000063240
N. Function Calls	6	N. Gradient Calls	5
N. Hessian Calls	6	Preproces. Time	0
Time for Method	0	Effective Time	0

Test Statistics for Training Data

-2 Log L: Without: 53.8177 With Covariates: 41.1054
R_squared: 0.30456 Max(R_squared): 0.785114
Likelihood Ratio Test: 12.7124 df=2 Prob=0.00173598
Score (logrank) Test: 12.2431 df=2 Prob=0.00219503
Wald Test: 8.43157 df=2 Prob=0.0147608
AIC=45.1054 SBC=48.2161

Global PH Test (Kaplan-Meier): ChiSqu=1.88404 Prob=0.38984

Coefficients, ASEs, Z, and P Values

Variable	Coef	AsyStdErr	Z_Value	P_Value
age	0.13852785	0.04801415	2.88514644	0.00391232
ecog_ps	-0.09669743	0.62994270	-0.15350194	0.87800246

Risk (exp(coef)) and Confidence Intervals

Var	Coef	Exp(Coef)	CI_Lower	CI_Upper
1	0.1385278	1.1485817	1.0454233	1.2619193
2	-0.0966974	0.9078306	0.2641172	3.1204199

Proportional Hazards Test (Kaplan-Meier Transform)

Variable	Rho	ChiSqu	P_Value
age	-0.02034256	0.00466864	0.94552497
ecog_ps	0.51701863	1.85869534	0.17277545

Covariance Matrix of Parameter Estimates

****

Dense Symmetric Matrix (2 by 2)

S	age	ecog_ps
age	0.0023054	
ecog_ps	-0.0049991	0.3968278

Training Data, Predicted Values, and Residuals

****

Dense Matrix (26 by 8)

```
          |       Event     Strata      Start       Time       LinP
     -------------------------------------------------------------
OBS_01 |   1.0000000          0          0  59.000000  2.2840788
OBS_02 |   1.0000000          0          0  115.00000  2.5835344
OBS_03 |   1.0000000          0          0  156.00000  1.3748185
.............................................................
OBS_24 |   1.0000000  1.0000000          0  353.00000  0.9250740
OBS_25 |   1.0000000  1.0000000          0  365.00000  1.1887668
OBS_26 |           0  1.0000000          0  377.00000  0.3416551


          |   P_LPterms      P_ASE   R_Martin
     -----------------------------------
OBS_01 |   2.2840788  0.8727063  0.7839421
OBS_02 |   2.5835344  0.9713348  0.3366863
OBS_03 |   1.3748185  0.5515392  0.6252141
.....................................
OBS_24 |  -0.2682553  0.2930349  0.8520456
OBS_25 |  -1.6121555  0.5992499  0.5813637
OBS_26 |   0.4221902  0.3516331 -0.1794490


                    Total Computation Time: 0
```

3. Aalen's Model: Lung Cancer Data: N=228 with one observation missing:

```
  print "Lung Cancer Data Example";
  options NOECHO;
#include "..\\tdata\\lung.dat"
  options ECHO;


  nr = nrow(lung);  nc = ncol(lung);
  print "LUNG: nr=",nrow(lung)," nc=",ncol(lung);
  lung[,4] = (lung[,4] .== 2.);


  cnam = [" N inst time status age sex ph_ecog ph_karno pat_karno
            meal_cal wt_loss "];
  lung = cname(lung,cnam);
  print "LUNG=",lung[1:10,];


  print "fit = aareg(Surv(time, status) ~ age + sex + ph.ecog,";
  print " data=lung, nmin=1, taper=1:10 )";
```

```
modl = "3 = 5 6 7";
optn = [ "status"       4 ,
         "nmin"         1 ,
         "pinit"        2 ,
         "phist"        1 ,
         "print"        3 ];
< gof,parm,covm,resi > = survreg("aalen",lung,modl,optn);
print "Gof=",gof;
print "Parm=",parm;
print "Covm=",covm;
```

Survival Regression by Aalen's Additive Model
Test Type: Aalen
Time Data: Right Censored

Number Rows of Training Data.	228
Number Columns of Matrix	11
Rows of Train Data with Missing Values. . .	1
Number Training Observations.	227
Number Independent Covariates	3
Number (Train) Events	164

ChiSquared=26.7609 DF=3 Prob= 0.0000

Coefficients, ASEs, Z, and P Values

Variable	Slope	Coef	AsyStdErr	Z_Value	P_Value
Intercept	0.0052477	0.0059854	4.8e-003	1.259483	0.2079
age	4.29e-005	7.03e-005	7.2e-005	0.969643	0.3322
sex	-0.0033033	-0.0040289	1.2e-003	-3.299540	0.0010
ph_ecog	0.0031461	0.0038107	1.0e-003	3.704676	0.0002

Covariance Matrix of Parameter Estimates
****

Dense Symmetric Matrix (4 by 4)

S	Intercept	age	sex	ph_ecog
Intercept	2.3720996			
age	-152.70046	12023.778		
sex	-3.7440302	17.854357	35.049321	
ph_ecog	2.2576720	-283.50098	-5.4212658	86.187939

```
Total Computation Time: 0
```

5.14 Function `system`

$< irc, str >$ = system(commands)

Purpose: The `system` function either determines if a command processor (shell) is available on the CMAT running PC or tries to execute shell commands.

Input: The only input argument is either

- missing value: Tests the availability of a command processor. In Windows that could be `COMMAND.COM` or `CMD.EXE`. `rc = system(.);` returns zero when there is no command processor available, otherwise it returns 1.
- only one string: the string contains only one shell command which is executed.
- an n vector of strings: each string refers to one command and the commands are excuted in the order of the strings in the vector.

Output: There are two different situations:

1. The first output argument of this function is an integer `irc` which is 0 if there were no problems encountered when executing the commands, and $irc > 0$ if there were problems executing command.
2. If there is a second output argument specified, the output of the shell command is redirected from default standard output into a string object which can be processed afterward. For example, the `grep` command in Unix or Linux could be used to find a specific file name, which then could be processed.

Restrictions: 1. Only commands that are valid for an existing command processor (shell) should be entered.

2. For DOS the length of the concatenated strings cannot exceed 128 characters.

Relationships: spawn()

Examples: 1. Output is written to standard output:

```
rc = system("dir");              /* show directory content */
rc = system("date");             /* show date */
```

2. Output to string `str` or file `date.txt`:

```
<irc,str> = system("dir");         /* save output in string */
irc = system("date > date.txt");   /* save output in file date.txt */
```

3. Use string vector input:

```
lpc[1] = "date";                          /* show date */
lpc[2] = "lpr -P myprinter plot.png";     /* send file to printer */
rc = system(lpc);
```

5.15 Function `zip7`

irc = zip7("command")

irc = zip7("what","archive"<,file<,option>>)

Purpose: The `zip7` function relates to a (set of) `system()` call(s) for the *7-zip* software by Igor Pavlov. Note, using this function assumes that the excutable program `7za.exe` is available in the `cmat_util` directory. The standalone program `7za.exe` is missing some compression methods which are available in `7z.exe` due to copyright restrictions. The `cmat_util` directory also contains the `7zip.chm` HTML document which documents much more in detail the features of *7-zip* than we are able to do here.

Input: There are two forms of the call:

- only one input argument: the one string refers to the entire input of a single call of *7-zip*. See also `7zip.chm` HTML document in the `cmat_util` directory which documnts much more in detail the features of *7-zip*.
- at least two input arguments: the input command for `7-zip` is splitted off into two to four input arguments specifying the following:

  "what" specifies the task:

  "ext" extract files from archive to current or with -o specified output directory no file name arg: all files with file name arg: specifies which files (if a file would be overwritten the user is prompted: y(es),n(o),a(lways),s(kip),q(uit)
  -ai,-an,-ao,-ax,-i,-o,-p,-r,-so,-t,-x,-y

  "xfp" extract with full path to current or with -o specified output directory
  -ai,-an,-ao,-ax,-i,-o,-p,-r,-so,-t,-x,-y

  "add" add specified files to archive files can be specified with wildcards entire directories: file string must end with /
  -i,-m,-p,-r,-sfx,-si,-so,-ssw,-t,-u,-v,-w,-x

  "del" delete specified files in archive files can be specified with wildcards
  -i,-m,-p,-r,-u,-v,-w,-x

  "upd" update and add files that are not in archive
  -i,-m,-p,-r,-sfx,-si,-ssw,-t,-u,-w,-x

  "tst" test selected files from archive
  -ai,-an,-ax,-i,-p,-r,-x

  "lst" list files from archive: needs only the archive argument, no file name argument
  -ai,-an,-ax,-i,-sit,-p,-r,-t,-x

"archive" one string (path to file)

file scalar or vector of strings (path to one or more). If that argument is a vector of strings, the program is applied multiple times for every file string once (this argument may be missing)

option scalar or vector of strings (this argument may be missing)

- **-ai** include archive filenames
- **-an** disable parsing of archive name
- **-ao** overwrite mode
- **-ax** exclude archive filenames
- **-i** include filenames
- **-m** set compression method: Zip, Gzip, Bzip2, 7z, LZMA, PPMd, BC32
- **-o** set output directory
- **-p** set password (if archived files were compressed with password they can only be decompressed using the correct password)
- **-r** recurse subdirectories; there are three ways to treat wildcards in filenames: -r enable recurse subdirectories -r- diable recurse subdirectories, -r0 enable recurse subdirectories only for wildcard names
- **-scs** set charset for list files
- **-seml** set archive by email
- **-slp** set large pages mode
- **-slt** show technical information
- **-sfx** create SFX archive
- **-si** read data from stdin
- **-so** write data to stdout
- **-ssc** set sensitive case mode
- **-ssw** compress files open for writing
- **-t** type of archive
- **-u** update options
- **-v** create volumes
- **-w** set working directory
- **-x** exlude filenames
- **-y** assume yes on all queries

If that argument is a vector of strings, the strings are being concatenated into a single option string.

Remember the fact, that filenames with spaces or wildcards must be written into double quotes. Since the arguments for this function are strings, quotes inside strings must be preceded by backslash, i.e. \".

Output:

Restrictions: 1. The file name specification may contain wildcards which are standard except for one important difference: the expression `*.*` means here matching **all files that have an extension**. If you want to refer to **all files** (in a specified directory and maybe subdirectories) you must use the simple `*` wildcard.

Relationships: system(), spawn()

Examples: 1. use the "add" option for compressing file into archive:

```
print "Compress ode.txt to arch.zip";
irc = zip7("add","arch.zip","tode.txt");
```

2. use the "ext" option for decompressing file from archive:

```
print "DECompress arch.zip to ode_new.txt";
irc = zip7("ext","arch.zip","tode.txt","-y");


irc = zip7("ext","arch.zip","tode.txt","-o..\\tgplt");
```

3. use the "ext" option for decompressing file from archive and store in different directory:

```
print "Quotes inside strings must be preceded by backslash";
irc = zip7("ext","arch.zip","tode.txt","-y -o\"..\\tgplt\"");
```