

# CMAT Newsletter: December 2008

Wolfgang M. Hartmann

December 2008

## Contents

<b>1</b>	<b>General Remarks</b>	<b>3</b>
1.1	New Functions . . . . .	3
1.2	Fixed Bugs . . . . .	4
<b>2</b>	<b>Modifications of Features</b>	<b>4</b>
2.1	Modifications of the <code>loc</code> Function . . . . .	4
2.2	Modifications of the <code>sem</code> Function . . . . .	5
<b>3</b>	<b>Extensions to Various Functions</b>	<b>8</b>
3.1	Extensions to <code>quantile</code> Function . . . . .	8
3.2	Extensions to <code>nlp</code> Function . . . . .	10
3.2.1	NONDIF Techniques . . . . .	10
3.2.2	UOBYQA, NEWUOA, and BOBYQA Techniques . . . . .	22
<b>4</b>	<b>New Developments</b>	<b>36</b>
4.1	Function <code>affarms</code> . . . . .	36
4.2	Function <code>affvsn</code> . . . . .	39
4.3	Function <code>decrypt</code> . . . . .	43
4.4	Function <code>encrypt</code> . . . . .	46
4.5	Function <code>locatn</code> . . . . .	50
4.6	Function <code>log2</code> . . . . .	61
4.7	Function <code>mad</code> . . . . .	62
4.8	Function <code>median</code> . . . . .	65
4.9	Function <code>mpolish</code> . . . . .	67
4.10	Function <code>nlfit</code> . . . . .	72
4.11	Function <code>nlfitprd</code> . . . . .	122
<b>5</b>	<b>Illustration</b>	<b>132</b>
5.1	Random Generators for Normal and Exponential Variates . . . . .	132
5.1.1	Normal Variates $n = 1000$ . . . . .	133
5.1.2	Normal Variates $n = 10,000,000$ . . . . .	138

5.1.3	Exponential Variates $n = 1000$	143
5.1.4	Exponential Variates $n = 10,000,000$	148
5.2	Matching the Behavior of an Index by a Small Number of Assets	152
5.2.1	Training a Model with 2006 Data of IMKB30	152
5.2.2	Model Prediction for some January 2007 Data	158

# 1 General Remarks

A set of new functions dealing with the normalization of genetic microarray data (affymetrix chips) were implemented.

Also, some new nonsmooth optimization algorithms with an important application, location optimization, was implemented.

The new `nlfit` function is a very general approach for predictive modeling (data mining) where the response depends on predictor variables with unknown nonlinear model functions. For scoring additional data sets with the model obtained by the `nlfit` algorithm the new `nlfitprd()` function can be used.

Some important special situations for matrix concatenation were reprogrammed for speedup. Especially concatenating large sparse matrices is now much faster.

## 1.1 New Functions

**affarms** "Factor Analysis for Robust Microarray Summarization" (FARMS) implements an EM algorithm for estimating loadings and unique variances of the one factor model for the normalization of microarray data by Hochreiter et al. (2006)

**affvsn** "Variance Stabilizing Normalization" (VSN) algorithm for the column normalization of microarray data by Huber et al. (2002)

**decrypt** decrypts the content of a file or a directory of files

**encrypt** encrypts the content of a file or a directory of files

**locatn** assigning  $K$  optimal locations among  $n > K$  potential locations for servicing  $m$  clients.

**log2** computes the logarithm w.r.t. base 2.

**mad** computes the MAD of vector or columns or rows of matrix (is already similar in `univar()`);

**median** computes the Median of vector or columns or rows of matrix (is already similar in `univar()`);

**mpolish** computes mean and median polish of a data matrix (Tukey, 1977a, p. 179)

**nlfit** implements a stepwise nonlinear regression algorithm for a variety of nonlinear activation and link functions and different parametrizations.

**nlfitprd** performs scoring a test data set with the model obtained by the algorithm in `nlfit()`

## 1.2 Fixed Bugs

A number of bugs were fixed, especially for operations with sparse matrices, e.g. `vec2tri()`, the Kronecker product `@`, and with the `sem` function.

Almost two months were spent for debugging to get a new stable version. That was necessary since many new features were added since the last stable release in 2003 and the older code had not been tested well for compatibility.

## 2 Modifications of Features

### 2.1 Modifications of the `loc` Function

The string specifications of the `loc` function have changed to those similar to the Fortran syntax which is also used in SAS:

Old	New	Meaning
"miss"	"ms"	find locations of all missing values no third argument must be specified
"nznm"	"nz"	find locations of all nonzeros and nonmissings no third argument must be specified
"zero"	"eq"	find locations of all zeros no third argument is specified
"nonz"	"ne"	find locations of all nonzeros (includes locations of missing value) no third argument is specified
"equal"	"eq"	this is the default and must not be specified find locations of all entries in <b>a</b> which are equal to zero or a specified third argument
"unequ"	"ne"	find locations of all entries in <b>a</b> which are unequal to zero or a specified third argument
"great"	"gt"	find locations of all entries in <b>a</b> which are larger than zero or a specified third argument
"small"	"lt"	find locations of all entries in <b>a</b> which are smaller than zero or a specified third argument
not av.	"ge"	find locations of all entries in <b>a</b> which are larger or equal than zero or a specified third argument
not av.	"le"	find locations of all entries in <b>a</b> which are smaller or equal than zero or a specified third argument

Note, that the "nonz" and "unequ" both change to **ne** and "zero" and "equal" both change to **eq**. Comparisons to zero are specified by skipping the third (last) input argument.

## 2.2 Modifications of the **sem** Function

For "mean structure analysis" (the "msa" option) the null model was changed taking into account that the mean is estimated. That makes some large difference in the values of the  $\chi^2$  value of the null model and some other fit indices, like Bentler's CFI (comparative fit index) and TFI (Tuckers fit index). The results are now the same as you would get with M-Plus.

Due to the availability of object lists the first, second, and fifth input arguments of the **sem** function are now simpler. The **semdata**, **semram**, and **semwgt** statements for multiple sample analysis are no longer needed since we are now able to specify lists of input objects for more multiple samples.

For multiple sample analysis of correlation and covariance matrices the number of observations can be specified with a modified form of the **parms** input argument. The following is the description of the newly modified first five input arguments:

1. The first argument **data** can be either the name of a single data object or the name of a list of data objects for multiple sample analysis. Each of the input matrices must specify one of the
  - (a) **nobs** by **nvar** matrix of raw data
  - (b) symmetric **nvar** by **nvar** matrix of covariances or correlations.
  - (c) **nvar+1** by **nvar** matrix that contains a symmetric covariance or correlation matrix in its first **nvar** rows and a vector of mean values in its last row.
  - (d) **nvar+2** by **nvar** matrix that contains a symmetric covariance or correlation matrix in its first **nvar** rows, a vector of mean values afterward, and a vector of standard deviations in its last row.

For multiple sample analysis a single raw data matrix with an "idvar" column defining the sample number of that observation.

For raw data input the number of observations of each sample is determined from the data input. For correlation or covariance input the number of observations of each sample must be specified:

- for a single sample (single input matrix) by using the **optn** argument **nobs**.
- for multiple samples (list of input matrices) by using the **\_SAMP** and **\_NOBS** columns of the **parms** input argument.

2. The second argument **model** can be

- the name of an object specifying a *RAM* matrix.
- the name of a list of *RAM* matrix objects for multiple sample analysis.
- a string referring to a previously defined model statement:
  - ”**semeqs**” referring to a set of **semeqs**, **semvar**, and **semcov** statements,
  - ”**semcos**” referring to a **semcos** statement,
  - ”**semfact**” referring to a **semfact** statement.
  - ”**factor**” specifying an exploratory factor model  $LP + U$ .
- a string specifying the path to an INRAM data set.

That means, the COSAN and EQS specifications always need model statements. However, RAM can be specified directly using a model matrix object. An INRAM data set cannot have the name **ram**, **cosan** or **factor**.

3. 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.
4. An optional fourth **parms** input argument specifies a matrix of specific additional information:
  - (a) For parameters: initial values, reparametrization, and constraints: the column names of **parms** can be specified as a subset of the following:
    - \_PNAM** string specifying the name of the parameter for relating the remaining information of that row to that model parameter;
    - \_INIT** real value specifying the initial value of that parameter; a missing value indicates that the default setting should be used;
    - \_DEP** int value specifying the index of the return vector from a reparametrizing function specified as the seventh input argument;
    - \_INDEP** int value specifying the index of the input vector of a reparametrizing function specified as the seventh input argument;
    - \_LBC** real value specifying a lower bound for that parameter;
    - \_UBC** real value specifying an upper bound for that parameter;
    - \_LC** int value specifying the index of the parameter in a matrix of linear constraints specified as the seventh input argument;
    - \_NLC** int value specifying the index of the parameter in the input vector of a nonlinear constraints function specified as the eighth input argument.
  - (b) For multiple sample analysis of covariance or correlation matrices: number of observations, names and labels for samples: the column names of **parms** can be specified as a subset of the following:

**\_SAMP** integer specifying the sample number  
**\_NOBS** integer specifying the number of observations of the sample  
**\_SAMPNAM** string specifying a name for the sample  
**\_SAMPLAB** string specifying a label for the sample

For this content the "**\_PNAM**" column must always be present, the other columns are optional. Missing values indicate that default settings should be used.

5. An optional fifth `wmat` input argument specifies either a `nvar` by `nvar` weight matrix  $\mathbf{W}$  for GLS or DWLS estimation or a  $\binom{nvar}{2} \times \binom{nvar}{2}$  weight matrix for WLS estimation, where  $\binom{nvar}{2} = \frac{n(n+1)}{2}$ . For multiple sample analysis, lists of symmetric matrices must be specified.

Three additional return argument were added:

- `outwgt` returns one weight matrix or a list of `ns` weight matrices used in the analysis, where `ns` is the number of samples;
- `outcov` returns the  $p \times p$  covariance matrix of parameter estimates;
- `outjac` returns `ns` stacked Jacobian matrices.

```
< gof,est,resi,toteff,indeff,outwgt,outcov,outjac >
= sem(data,model,optn<,parms<,inwgt<,lc<,repar<,nlc,nlcb>>>>>)
```

The output is in a form that can be reused as the fifth input argument `inwgt` in a later `sem` call. (It could also be saved in a permanent data set using the `obj2fil` function.) An extensive testing including many new test examples required the fixing of a number of bugs.

### 3 Extensions to Various Functions

#### 3.1 Extensions to quantile Function

The old version `quant = quantile(a,k)` was defined with only two input arguments, the  $m \times n$  data `a` and a positive integer `k`. The new version was enhanced to match results of a similar function in the R language: `quant = quantile(a,k|prob<,optn<)` where the second argument can be a scalar or vector of real numbers, probabilities in  $[0, 1]$ . A third input argument `optn` was added which should be a vector specifying the following options:

1. specifies the amount of printed output, `optn[1]=0` is the default specifying no printed output.
2. specifies the type of the quantile which is valid only when the second input argument is a scalar or vector of probabilities. If the second input argument is a positive scalar  $k > 1$  the type should be specified as missing or zero. If the second input argument is real and inside  $[0, 1]$ , with `optn[2]=1, ..., 9` one of nine types of quantiles can be specified. The types are the same as for the quantile function in R, see below for a table.

Discontinuous Sample Quantiles	
Type	Description
1	inverse of empirical distribution function
2	like <code>type=1</code> but with averaging discontinuities
3	SAS definition: nearest even order statistic
Continuous Sample Quantiles	
4	$p(k) = k/n$ : linear interpolation of empirical cdf
5	$p(k) = (k - .5)/n$ : piecewise linear function where knots are values midway through the steps of the empirical cdf; popular with hydrologists
6	$p(k) = k/(n + 1)$ : SPSS and MINITAB definitions
7	$p(k) = (k - 1)/(n - 1)$ : here $p(k) = \text{mode}[F(x[k])]$ S and R version is completely back compatible: this is default;
8	$p(k) = (k - 1/3)/(n + 1/3)$ : here $p(k) = \text{median}[F(x[k])]$ resulting quantile is approximately median-unbiased regardless of the $x$ distribution
9	$p(k) = (k - 3/8)/(n + 1/4)$ resulting quantile is approximately unbiased for expected order statistic if $x$ is normally distributed

The following example is using 1001 normal distributed random values (generated from R):

```
rqunt = [
#include "..\tdata\rquant.dat"
];
print "nd=", nd = size(rqunt);
```



```

prob1 = [ 0. .25 .50 .75 1. ];
quan1 = quantile(rqunt,prob1);

```

```
nd= 1001
```

R Example for Quantile: Default setting

```

Quan1=
|          0%          25%          50%          75%          100%
-----
1 |  -3.5385  -0.64897  -0.05884   0.62202   3.3331

```

```

quan2 = cons(9,7,.);
optn = [ 2 , 1 ];
for (j = 1; j <= 9; j++) {
  optn[2] = j;
  quan2[j,] = quantile(rqunt,prob2,optn);
}
cnam = [ "0.1%" "0.5%" "1%" "2%" "5%" "10%" "50%" ];
rnam = [" typ1:9 "];
quan2 = cname(quan2,cnam);
quan2 = rname(quan2,rnam);
print "Quan2=", quan2;

```

```

Quan2=
|          0.1%          0.5%          1%          2%
-----
typ1 |  0.00000  0.00000  -3.5385  -3.5385
typ2 | -3.3387  -2.6462  -2.4810  -2.2290
typ3 | -3.5385  -2.7402  -2.4883  -2.2327
typ4 | -3.5383  -2.7397  -2.4883  -2.2326
typ5 | -3.4384  -2.6927  -2.4846  -2.2308
typ6 | -3.5381  -2.7393  -2.4882  -2.2326
typ7 | -3.3387  -2.6462  -2.4810  -2.2290
typ8 | -3.4716  -2.7082  -2.4858  -2.2314
typ9 | -3.4633  -2.7044  -2.4855  -2.2312
-----
|          5%          10%          50%
-----
typ1 |  -3.5385  -3.5385  -3.5385
typ2 |  -1.7035  -1.2892  -0.05884
typ3 |  -1.7079  -1.2892  -0.06516
typ4 |  -1.7077  -1.2892  -0.06200

```

typ5	-1.7055	-1.2892	-0.05884
typ6	-1.7074	-1.2892	-0.05884
typ7	-1.7035	-1.2892	-0.05884
typ8	-1.7061	-1.2892	-0.05884
typ9	-1.7060	-1.2892	-0.05884

## 3.2 Extensions to nlp Function

### 3.2.1 NONDIF Techniques

The new NONDIF optimization technique implements a set of nonsmooth subgradient techniques developed in the early 1990's by a team from the University of Bayreuth (Outrata, Schramm and Zowe, 1991) called Bundle-Trust-Region methods. (BTNCLC was mailed to me in August 2008.) The Fortran code of all routines (except BTNCLC) was mailed to me in February and May 1995 when I was still working for SAS. Due to some very tragic accident of Prof. Zowe the work on this software was not continued. The original Fortran package contains the following programs:

**BT** the unconstrained problem with convex function

**BTNC** the unconstrained problem with nonconvex function

**BTNCBCL** the bound constrained problem with nonconvex function

**BTCLC** the bound and linear constrained problem with convex function

**BTNCLC** the bound and linear constrained problem with nonconvex function

All methods use a QP solver written by K. Schittkowski which is based on software by J.M.D. Powell. Slightly modified versions of these algorithms were implemented in CMAT.

For unconstrained optimization, Einarsson (1998), and Madsen & Einarsson (1999) developed a different method based on stepwise LP which sometimes can compete with the BT and BTNC methods. The following additional options are relevant for the NONDIF algorithms:

**"vers"** this should be 1 for the Einarsson & Madsen algorithm (only unconstrained) or 2 for the BT algorithms (default is 2);

**"corrs"** is the number of gradients in the bundle (as larger as better), default is 5;

**"fconvex"** the objective function is convex.

Here a few examples:

1. Subgradient specification of example by Shor:

```

a = [ 0.0  2.0  1.0  1.0  3.0  0.0  1.0  1.0  0.0  1.0 ,
      0.0  1.0  2.0  4.0  2.0  2.0  1.0  0.0  0.0  1.0 ,
      0.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  2.0  2.0 ,
      0.0  1.0  1.0  2.0  0.0  0.0  1.0  2.0  1.0  0.0 ,
      0.0  3.0  2.0  2.0  1.0  1.0  1.0  1.0  0.0  0.0 ]';
b = [ 1.  5. 10.  2.  4.  3. 1.7 2.5 6.0 4.5 ];
m = nrow(a); n = ncol(a);
c = cons(m);

```

```

function fshor5(x) global(a,b,c) {
  m = nrow(a); n = ncol(a);
  for (i = 1; i <= m; i++)
    c[i] = abs(b[i] * ssq(x - a[i,]));

  /* get index k1 of max c[m] */
  k = c[>!]; k1 = k[1];
  crit = c[k1];
  return(crit);
}

```

```

function gshor5(x) global(a,b,c) {
  /* get index k1 of max c[m]: c[] should still be stored */
  k = c[>!]; k1 = k[1];
  grad = 2. * b[k1] * (x - a[k1,]);
  return(grad);
}

```

```

x0 = [ 4#0.  1.  ];
f0 = fshor5(x0);
print "Function at starting point",f0;
g0 = gshor5(x0);
print "Gradient at starting point",g0;

```

Function at starting point 80.000

Gradient at starting point

	1	2	3	4	5
1	-20.000	-40.000	-20.000	-20.000	-20.000

2. The Einarsson-Madsen algorithm:

```

x0 = [ 4#0. 1. ];

mopt = [ "tech"      "nondif" ,
         "vers"      1 , /* NONDIF version Madsen */
         "maxit"     1000 ,
         "maxfu"     5000 ,
         "print"     5 ];
< xr,rp,der1,der2 > = nlp(fshor5,x0,mopt,.,.,.,gshor5);

```

NonDifferentiable Method (Einarsson and Madsen, 1998)  
 Convexity of Objective Function NOT Assumed  
 User Specified Gradient

Iteration Start:

```

N. Variables          5
Criterion             110.0000000      Max Grad Entry  40.00000000

```

Iter	nfun	act	optcrit	norm(hk)	lambda	pred	rho
1	2	1	25.00000	1.000000	1.0000000	0.6071429	1.000000
2	10	4	25.00000	0.200000	2.0000000	-10.000000	0.200000
3	19	5	23.51537	0.500000	0.5000000	0.4017618	0.040833
4	27	6	23.51537	0.032465	0.5000000	-10.000000	0.032465
5	35	3	22.79268	0.125000	0.1250000	0.7954681	0.032465
6	44	6	22.67422	0.137706	0.2500000	0.3154945	5.0e-003
7	52	6	22.67422	5.0e-003	0.2500000	-10.000000	5.0e-003
8	59	3	22.63708	0.062500	0.0625000	0.3134346	1.3e-003
9	65	3	22.62049	0.062500	0.0625000	0.2157089	1.3e-003
10	74	3	22.60181	0.015625	0.0156250	0.7830801	3.1e-004
11	82	5	22.60181	6.3e-004	0.0312500	-10.000000	6.3e-004
12	89	3	22.60055	7.8e-003	0.0078125	0.5707412	1.6e-004
13	98	5	22.60028	7.8e-003	0.0078125	0.3947248	1.6e-004
14	105	0	22.60028	1.6e-004	0.0078125	1.00e-006	1.6e-004
15	112	0	22.60028	3.9e-005	0.0019531	-10.000000	3.9e-005
16	120	5	22.60018	4.9e-004	4.88e-004	0.2371260	9.8e-006
17	127	3	22.60018	2.4e-006	1.22e-004	-1.1439683	2.4e-006

Successful Termination After 17 Iterations

ABSGCONV convergence criterion satisfied.

```

Criterion             22.60017694      Max Grad Entry  2.4414e-006
N. Grad Storage       5
N. Function Calls     128      N. Gradient Calls    118
Preproces. Time       0      Time for Method      1
Effective Time        1

```

```

*****
Optimization Results
*****

```

Parameter Estimates

```

-----
Parameter      Estimate      Gradient
1 X_1          1.12388195  13.486583
2 X_2          0.97931517  11.751782
3 X_3          1.47603969 -6.2875237
4 X_4          0.92000741 -0.9599111
5 X_5          1.12409702  13.489164

```

Value of Objective Function = 22.6002

3. The unconstrained and convex case BT:

```

x0 = [ 4#0. 1. ];

mopt = [ "tech"      "nondif" ,
         "vers"      2 , /* NONDIF version BTR */
         "fconvex"   , /* f assumed convex */
         "corrs"     10 , /* number of corrections */
         "maxit"     1000 ,
         "maxfu"     5000 ,
         "print"     5 ];
< xr,rp,der1,der2 > = nlp(fshor5,x0,mopt,.,.,.,gshor5);
/* Fopt = 22.6002
   X = [ 1.1243 0.97965 1.4786 0.91989 1.1245 ]; */
print "Bundle TR MIN: XR=",xr;
print "Bundle TR MIN: RP=",rp;

```

```

*****
Optimization Start
*****

```

Parameter Estimates

```

-----
Parameter      Estimate      Gradient
1 X_1          0.00000000 -20.000000

```

2	X_2	0.00000000	-40.000000
3	X_3	0.00000000	-20.000000
4	X_4	0.00000000	-20.000000
5	X_5	1.00000000	-20.000000

Value of Objective Function = 80

Bundle Trust Region Method (Ostrata-Schramm-Zowe, 1991)  
 Convex Objective Function Assumed  
 User Specified Gradient

Iteration Start:

N. Variables	5
Criterion	80.00000000
Max Grad Entry	40.00000000

Iter	nfun	act	optcrit	maxgrad	gradnorm	alpha	rho
2	3	2	80.00000	40.00000	28.284271	226.50044	32.06403
2	4	1	80.00000	67.95013	56.568542	0.0000000	1.282561
3	5	1	37.80071	18.79501	30.120043	0.0000000	0.363613
4	6	2	37.80071	21.00906	16.420987	4.2711930	0.108075
4	7	2	37.80071	14.92391	14.303712	5.8060317	2.050056
4	8	2	37.80071	33.40431	14.471175	5.5502253	0.755402
5	9	3	29.00440	22.13278	10.438652	6.0458126	0.393061
6	10	4	29.00440	26.60915	7.7923853	6.9131913	0.219034
7	11	4	26.10894	15.32694	2.6369991	4.8226013	0.025084
8	12	4	24.34863	15.15838	1.8864657	2.3266175	0.012837
9	13	4	24.34863	12.39751	2.1901058	1.8572140	0.017302
10	14	4	23.89188	19.32857	1.2446275	1.2657621	5.6e-003
11	15	4	22.70130	13.74280	0.5903855	0.1312849	0.031433
12	16	5	22.70130	18.90823	0.2002678	0.1639584	3.6e-003
13	17	5	22.66608	15.15163	0.0861074	0.1161131	6.7e-004
14	18	4	22.66608	18.73240	0.0244589	0.0857739	5.4e-005
15	19	4	22.64333	12.10321	0.0792175	0.0496663	5.7e-004
16	20	5	22.61557	13.50355	0.0921757	0.0240104	7.7e-004
17	21	5	22.61096	18.71096	0.0280888	0.0177295	1.8e-003
18	22	6	22.61096	15.09778	0.0020142	0.0176058	9.1e-006
19	23	6	22.61096	14.99445	0.0268033	0.0146066	1.6e-003
20	24	6	22.61096	18.87746	0.0134045	0.0148242	4.1e-004
21	25	6	22.60278	12.09313	2.23e-004	0.0055238	1.1e-007
22	26	6	22.60278	13.56868	0.0078431	0.0045218	1.4e-004
23	27	6	22.60278	18.77953	0.0102646	0.0037998	2.4e-004
24	28	6	22.60160	13.52349	0.0077556	0.0021372	1.4e-004
25	29	6	22.60160	15.01907	0.0052699	0.0020531	6.3e-005
26	30	5	22.60072	18.75832	0.0032959	9.50e-004	2.4e-005

```

27 31 6 22.60058 13.49656 0.0033100 7.17e-004 2.5e-005
28 32 6 22.60058 15.00512 0.0032581 6.63e-004 2.4e-005
29 33 6 22.60058 12.07966 0.0032579 5.15e-004 2.4e-005
30 34 6 22.60058 18.74864 0.0029156 4.98e-004 1.9e-005
31 35 6 22.60029 13.50192 0.0014579 2.08e-004 4.8e-006
32 36 6 22.60023 18.75746 0.0013410 1.30e-004 4.1e-006
33 37 6 22.60023 12.08307 6.37e-004 1.07e-004 9.1e-007
34 38 6 22.60022 15.01196 0.0017627 8.47e-005 7.0e-006
35 39 6 22.60022 13.50390 2.60e-004 7.77e-005 1.5e-007
36 40 6 22.60020 13.49200 9.53e-004 5.23e-005 2.0e-006
37 41 6 22.60017 15.00538 6.15e-007 1.98e-005 8.5e-013

```

```

Successful Termination After      37 Iterations
GCONV convergence criterion satisfied.
Criterion      22.60017281      Max Grad Entry  15.00538265
N. Grad Storage      10
N. Function Calls    42      N. Gradient Calls      42
Preproces. Time      1      Time for Method      0
Effective Time      1

```

Objective function seems to be convex.

```

*****
Optimization Results
*****

```

Parameter Estimates

```

-----
Parameter      Estimate      Gradient
1 X_1          1.12432717 -15.005383
2 X_2          0.97965293 -8.1627765
3 X_3          1.47861688  3.8289350
4 X_4          0.91989151  7.3591321
5 X_5          1.12454714  0.9963771

```

Value of Objective Function = 22.6002

4. The unconstrained and nonconvex case BTNC:

```

x0 = [ 4#0. 1. ];

print "Bundle Trust-Region: MIN: NONConvex Algorithm";
mopt = [ "tech"      "nondif" ,

```

```

"vers"          2 , /* NONDIF version BTR */
"corrs"         10 , /* number of corrections */
"maxit"         1000 ,
"maxfu"         5000 ,
"print"         5 ];
< xr,rp,der1,der2 > = nlp(fshor5,x0,mopt,..,..,gshor5);

```

Bundle Trust Region Method (Outrata-Schramm-Zowe, 1991)  
 Convexity of Objective Function NOT Assumed  
 User Specified Gradient

Iteration Start:

```

N. Variables          5
Criterion             80.00000000          Max Grad Entry  40.00000000

```

Iter	nfun	act	optcrit	maxgrad	gradnrm	alpha	rho
2	3	2	80.00000	40.00000	28.284271	226.50044	32.06403
2	4	1	80.00000	67.95013	56.568542	0.0000000	1.282561
3	5	1	37.80071	18.79501	30.120043	0.0000000	0.363613
4	6	2	37.80071	21.00906	16.420987	4.2711930	0.108075
5	7	2	27.19197	14.92391	11.981370	0.6595729	0.230145
6	8	3	27.19197	21.79488	8.0233448	1.5248242	0.103205
7	9	4	24.38932	12.48165	6.3630811	1.2120491	0.064912
8	10	4	23.67483	14.12845	3.2516058	1.6198609	0.016951
9	11	4	23.67483	15.78242	3.0180048	0.9353277	0.014603
10	12	4	23.32867	19.11006	1.5084959	0.8052569	3.6e-003
11	13	4	22.77759	12.13325	0.8644411	0.2729511	1.2e-003
12	14	4	22.77759	13.74625	0.3939206	0.1930694	2.5e-004
13	15	4	22.64137	15.02575	0.0853155	0.0523139	1.2e-005
14	16	4	22.61430	18.76364	0.1898998	0.0197479	5.8e-005
15	17	4	22.61430	13.50106	0.0750897	0.0144816	9.0e-006
16	18	4	22.60080	13.49219	0.0073361	8.30e-004	3.5e-007
17	19	4	22.60071	12.08238	0.0102082	5.90e-004	6.7e-007
18	20	4	22.60030	18.75782	0.0035766	1.46e-004	8.2e-008
19	21	4	22.60021	14.99990	0.0070167	4.81e-005	1.3e-006
20	22	4	22.60018	18.75706	0.0024376	1.53e-005	1.5e-007
21	23	4	22.60018	15.00747	9.35e-004	1.36e-005	2.2e-008
22	24	4	22.60016	15.00476	2.68e-004	1.88e-006	7.4e-009
23	25	4	22.60016	12.08215	3.71e-004	8.86e-007	1.4e-008
24	26	5	22.60016	13.49310	2.83e-004	2.99e-007	8.2e-009
25	27	4	22.60016	13.49242	1.52e-004	6.31e-008	9.5e-009
26	28	5	22.60016	15.00554	3.11e-005	6.65e-008	4.0e-010

Successful Termination After 26 Iterations



```

GCONV convergence criterion satisfied.
Criterion      22.60016225      Max Grad Entry  15.00553962
N. Grad Storage      10
N. Function Calls    29      N. Gradient Calls      29
Preproces. Time      0      Time for Method      0
Effective Time      0
Objective function seems to be convex.

```

```

*****
Optimization Results
*****

```

```

Parameter Estimates
-----

```

Parameter	Estimate	Gradient
1 X_1	1.12430755	-15.005540
2 X_2	0.97946424	-8.1642861
3 X_3	1.47763403	3.8210722
4 X_4	0.92018564	7.3614851
5 X_5	1.12429511	0.9943609

```

Value of Objective Function =      22.6002

```

5. The boundary constrained case BTNCBC:

```

x0 = [ 4#0. 1. ];
lbc = [ 5#0. ]; ubc = [ 4#1. 2. ];
bc = lbc' -> ubc';

print "Bundle Trust-Region: MIN: NONConvex Algorithm";
mopt = [ "tech"      "nondif" ,
        "vers"      2 , /* NONDIF version BTR */
        "corrs"     10 , /* number of corrections */
        "maxit"     1000 ,
        "maxfu"     5000 ,
        "print"     5 ];
< xr,rp,der1,der2 > = nlp(fshor5,x0,mopt,bc,...,gshor5);

```

```

*****

```

Optimization Start

\*\*\*\*\*

Parameter Estimates

-----

Parameter	Estimate	Gradient	Lower BC	Upper BC
1 X_1	0.00000000	-20.000000	0.0000000	1.0000000
2 X_2	0.00000000	-40.000000	0.0000000	1.0000000
3 X_3	0.00000000	-20.000000	0.0000000	1.0000000
4 X_4	0.00000000	-20.000000	0.0000000	1.0000000
5 X_5	1.00000000	-20.000000	0.0000000	2.0000000

Value of Objective Function = 80

Bundle Trust Region Method (Ostrata-Schramm-Zowe, 1991)

Convexity of Objective Function NOT Assumed

User Specified Gradient

Iteration Start:

N. Variables	5	N. Mask Constr.	0
N. Bound. Constr.	10	Max Grad Entry	40.00000000
Criterion	80.00000000		
N. Active Constraints	4		

Iter	nfun	act	optcrit	maxgrad	gradnm	alpha	rho
2	1	1	36.00000	40.00000	29.393877	0.0000000	0.270000
3	2	2	36.00000	21.20000	15.767152	2.7485825	0.077688
4	3	2	28.02512	18.77268	8.0318882	0.2014390	0.181438
5	4	3	28.02512	21.46389	6.1145740	1.3382217	0.105154
6	5	3	25.66330	13.83168	2.3641517	2.3610623	0.015720
7	6	4	24.99637	12.73736	1.0889734	2.0756864	0.030017
8	7	4	24.75232	19.86490	0.4504446	1.4993444	5.1e-003
9	8	4	24.75232	16.00000	0.7311917	0.9890784	0.013533
10	9	4	23.87084	12.85575	0.0134521	0.1386117	4.1e-005
11	10	4	23.87084	19.32959	0.0167079	0.1074100	6.4e-005
12	11	4	23.82708	16.00000	0.0085580	0.0475927	1.7e-005
13	12	4	23.79611	12.44946	7.64e-004	0.0145165	1.2e-006
14	13	4	23.78211	12.86109	3.41e-006	3.15e-004	2.2e-010
15	14	4	23.78205	19.28252	2.22e-006	2.00e-004	9.1e-011
16	15	4	23.78194	16.00000	4.50e-007	6.66e-005	3.4e-011

Successful Termination After 16 Iterations

```

GCONV convergence criterion satisfied.
Criterion      23.78194219          Max Grad Entry  16.00000000
N. Active Constraints    2          N. Grad Storage    10
N. Function Calls      16          N. Gradient Calls  16
Preproces. Time        0          Time for Method    0
Effective Time         0
Objective function seems to be convex.

```

```

*****
Optimization Results
*****

```

```

Parameter Estimates
-----

```

Parameter	Estimate	Gradient	Active BC
1 X_1	1.00000000	-16.000000	Upper BC
2 X_2	0.88835984	-8.8931213	
3 X_3	1.00000000	0.0000000	Upper BC
4 X_4	0.83940002	6.7152001	
5 X_5	1.07175865	0.5740692	

Value of Objective Function = 23.7819

6. The linear constrained convex case BTCLC:

```

x0 = [ 5#1. ];
lbc = [ 5#0. ]; ubc = [ 5#2. ];
bc = lbc' -> ubc';
lc = [ . 1. 0. 0. 0. 1. 2. ]; /* IC */

print "Bundle Trust-Region: MIN: NONConvex Algorithm";
mopt = [ "tech"      "nondif" ,
        "vers"      2 , /* NONDIF version BTR */
        "fconvex"   , /* f assumed convex */
        "corrs"     10 , /* number of corrections */
        "maxit"     30 ,
        "maxfu"     5000 ,
        "print"     5 ];
< xr,rp,der1,der2 > = nlp(fshor5,x0,mopt,bc,lc,.,gshor5);

```

\*\*\*\*\*  
 Optimization Start  
 \*\*\*\*\*

Parameter Estimates  
 -----

Parameter	Estimate	Gradient	Lower BC	Upper BC
1 X_1	1.00000000	-10.000000	0.0000000	2.0000000
2 X_2	1.00000000	0.0000000	0.0000000	2.0000000
3 X_3	1.00000000	0.0000000	0.0000000	2.0000000
4 X_4	1.00000000	0.0000000	0.0000000	2.0000000
5 X_5	1.00000000	-20.000000	0.0000000	2.0000000

Value of Objective Function = 25

Linear Constraints  
 -----

[ 1]ACT-2.0000000 <= - 1.00000 \* X\_1 - 1.00000 \* X\_5  
 ( 0.00000 )

Bundle Trust Region Method (Outrata-Schramm-Zowe, 1991)  
 Convex Objective Function Assumed  
 User Specified Gradient

Iteration Start:

N. Variables	5	N. Mask Constr.	0
N. Bound. Constr.	10	Lin. Equ. Constr.	0
N. Linear Constr.	1	Max Grad Entry	20.0000000
Criterion	25.00000000	N. Active Constraints	1

Iter	nfun	act	optcrit	maxgrad	gradnm	alpha	rho
1	0	1	25.00000	20.00000	6.32e-004	0.0000000	1.000000
1	1	1	25.00000	24.00000	0.0063182	0.0000000	1.000000
1	2	1	25.00000	24.00000	0.0631824	0.0000000	1.000000
1	3	1	25.00000	24.00000	0.6318237	0.0000000	1.000000
1	4	1	25.00000	24.00000	6.3182371	0.0000000	1.000000
1	5	1	25.00000	24.00000	7.0710678	0.0000000	0.012525
2	6	2	25.00000	16.89532	4.1408339	0.1921540	4.3e-003
3	7	2	24.46701	19.42406	2.9375024	0.0106646	0.019454
4	8	3	24.46701	13.78642	2.1750176	0.0348509	0.010665

5	9	3	24.22694	16.81854	1.8779548	0.0833776	8.0e-003
6	10	3	24.22694	18.69935	0.1232769	0.0659432	3.4e-005
7	11	3	24.22694	13.23451	0.1358220	0.0575472	4.2e-005
8	12	3	24.22694	18.97823	0.8717513	0.0066635	1.7e-003
9	13	3	24.18514	13.44354	0.0541816	0.0212196	6.0e-005
10	14	4	24.18258	16.98728	0.0101920	0.0145210	2.1e-006
11	15	3	24.18258	18.77699	0.2847616	0.0034695	1.6e-003
12	16	3	24.18258	13.69354	0.2050968	0.0053245	8.5e-004
13	17	3	24.17809	18.63972	0.0769498	0.0032218	1.2e-004
14	18	3	24.17809	13.57858	0.0402430	0.0032119	3.3e-005
15	19	3	24.17809	17.07058	0.1063510	0.0015768	2.3e-004
16	20	3	24.17653	18.70437	0.0337468	7.94e-004	2.3e-005
17	21	2	24.17574	18.68292	1.7878440	5.92e-005	0.583707
17	22	2	24.17574	21.99353	1.7878440	5.92e-005	5.8e-003
18	23	3	24.17574	13.92120	0.0599509	0.0062178	6.6e-006
19	24	3	24.17574	13.58818	0.0109461	8.71e-005	2.2e-007
20	25	3	24.17574	17.05213	0.0034029	4.49e-005	2.1e-008
21	26	3	24.17574	13.58131	0.0023912	3.74e-005	1.0e-008
22	27	3	24.17571	17.05405	7.78e-004	1.95e-006	1.0e-008
23	28	4	24.17570	18.68201	2.63e-004	2.58e-007	1.0e-008
24	29	4	24.17570	18.68244	0.0065299	0.0063248	6.3e-006
25	30	3	24.17570	13.59134	1.63e-004	7.36e-006	3.9e-009
26	31	3	24.17570	13.58076	2.38e-005	3.75e-008	8.4e-011

Successful Termination After 26 Iterations

GCONV convergence criterion satisfied.

Criterion	24.17577254	Max Grad Entry	13.58075529
N. Active Constraints	1	N. Grad Storage	10
N. Function Calls	32	N. Gradient Calls	32
Preproces. Time	0	Time for Method	0
Effective Time	0		

Objective function seems to be convex.

\*\*\*\*\*

Optimization Results

\*\*\*\*\*

Parameter Estimates

-----

Parameter	Estimate	Gradient	Active BC
1 X_1	0.86827039	10.419245	
2 X_2	1.08538160	13.024579	
3 X_3	1.12003319	-10.559602	

```

4 X_4      0.79458520 -2.4649776
5 X_5      1.13172961 13.580755

```

```

Value of Objective Function =      24.1758

```

```

Linear Constraints Evaluated at Solution
-----

```

```

[ 1]ACT  -1.00000 * X_1      - 1.00000 * X_5      + 2.00000
= -2.2204e-016

```

7. The linear constrained nonconvex case BTNCLC: Since the SHOR function is convex the same result is obtained as with the BLCLC algorithm.

### 3.2.2 UOBYQA, NEWUOA, and BOBYQA Techniques

Three new algorithms were added which performs Powell's Unconstrained and Bound constrained Optimization BY Quadratic Approximation (UOBYQA and POBYQA):

**UOBYQA** the original algorithm by Powell (2000): Report DAMTP 2000/NA14, University of Cambridge. This algorithm is using very much memory for large  $n$  but could be faster than the other.

**NEWUOA** the updated algorithm by Powell (2003): Report DAMTP 2003/NA03, University of Cambridge. This algorithm is using much less memory for large  $n$  but is usually slower than the other.

**BOBYQA** this is Powell's (2008) modification of the NEWUOA algorithm which permits the specification of inequality boundary constraints (masks, i.e. bounds where lower equal to upper bound are not permitted with BOBYQA).

The UOBYQA and NEWUOA algorithms are available by setting "tech" to UOBYQA. The NEWUOA is chosen by default (setting "vers" to 0), for selecting the UOBYQA algorithm "vers" must be set to 1. For NEWUOA the number of optimization points may be specified using the `intpoi` option which expects an integer in the  $[n + 2, (n + 1)(n + 2)/2]$  interval. For UOBYQA the number of interpolation points is set to  $(n * n + 3 * n + 2)/2$ . BOBYQA is an extension of NEWUOA for constraining the feasible region to a hyper cube with a finite length of edges.

At the end of the optimization we added some code for computing an approximate gradient by central finite differences for an idea how well the result satisfies optimality condition. With BOBYQA we also print the maximum constraint violation and the maximum gradient of the Lagrange function, i.e. the maximum of gradient values w.r.t. inactive variables (not at one of the bounds).

Testing confirms, the number of interpolation points has an impact on the memory allocation and numerical performance for NEWUOA and BOBYQA.

Memory requirements:

**UOBYQA**  $(n^4 + 8 * n^3 + 23 * n^2 + 42 * n + \max[2 * n^2 + 4, 18 * n])/4$  for a fixed number  $npt = (n * n + 3 * n + 2)/2$  of interpolation points

**NEWUOA**  $(npt + 11) * (npt + n) + n * (3 * n + 11)/2$  where  $npt$  is the specified number of points, by default  $npt = 2 * n + 1$

**BOBYQA**  $(npt + 5) * (npt + n) + 3 * n * (n + 5)/2$  where  $npt$  is the specified number of points, by default  $npt = 2 * n + 1$

These new algorithms are designed for problems where derivatives are not easily available or the function is not smooth (discontinuous first order derivatives). They are related to the COBYLA (Constrained Optimization BY Linear Approximation) algorithm which was developed by Powell in 1992.

The new algorithms compete especially with the Nelder-Mead algorithm. However, when comparing results we should have in mind that the results obtained from the Nelder-Mead implementation are not nearly as precise as those from UOBYQA, NEWUOA, and BOBYQA.

The first example features the Chebyquad function for  $n = 8$ :

```
print "\n *** Test NLPJOB: Chebyquad Function: m=n=8 ***\n";
n = 8; u1 = cons(n);
for (i = 2; i <= n; i+=2) u1[i] = 1. / (i * i - 1);

function fcheby81(x) global(u1) {
    f = cons(8);
    for (k = 1; k <= 8; k++) {
        t1 = 1.; t2 = 2. * x[k] - 1.; s = t2 + t2;
        for (i = 1; i <= 8; i++) {
            f[i] += t2;
            t = t2 * s - t1; t1 = t2; t2 = t;
        }
    }
    tt = 1. / 8.;
    f = tt * f + u1;
    crit = .5 * f[**];
    return(crit);
}

x0 = [ 1.:8. ] * .1111111111;
```

For comparison, we run the Nelder-Mead algorithm first:

```

mopt = [ "tech" "nmsimp" ,
        "print"   3 ];
< xr,rp > = nlp(fcbeby81,x0,mopt);
print "rp=",rp; print "xr=",xr;

```

Nelder-Mead Simplex Optimization

Iteration Start:

N. Variables               8  
Criterion                0.019308849

Iter	rest	nfun	act	optcrit	difcrit	std	delta	size
1	0	14	0	0.01931	94665.5	29279	1.0000	2.35901
.....								
65	0	508	0	2e-003	1e-005	4e-006	1.0000	4e-003
66	0	516	0	2e-003	1e-005	3e-006	1.0000	4e-003
67	0	522	0	2e-003	5e-006	2e-006	1.0000	4e-003
68	0	531	0	2e-003	2e-006	7e-007	1.0000	5e-003

Successful Termination After    68 Iterations

FCONV2 convergence criterion satisfied.

Criterion                0.001763053  
N. Function Calls           533                Preproces. Time                0  
Time for Method             1                Effective Time                1

\*\*\*\*\*  
Optimization Results  
\*\*\*\*\*

Parameter Estimates

-----

Parameter	Estimate
1 X_1	0.04298696
2 X_2	0.19245458
3 X_3	0.26551996
4 X_4	0.49898134
5 X_5	0.49879437
6 X_6	0.73178986
7 X_7	0.80718399
8 X_8	0.95685882

Value of Objective Function =    0.00176305



The default NEWUOA algorithm has a problem between iteration 3 and 4 which obviously results from the minimum number  $2n+1 = 17$  of interpolation points:

```
mopt = [ "tech" "uobyqa" ,
         "print" 3 ];
< xr,rp > = nlp(fcbeby81,x0,mopt);
print "rp=",rp; print "xr=",xr;
```

NEWUOA Algorithm by M.J.D. Powell (2004)

Iter	nfun	optcrit	difcrit	rho
1	18	0.01930885	0.00000000	0.05000000
2	37	0.01161401	0.00769483	0.00500000
3	98	0.00671276	0.00490125	5.000e-004
4	1249	0.00175929	0.00495347	5.000e-005
5	1349	0.00175844	8.473e-007	7.071e-006
6	1391	0.00175844	1.744e-009	1.000e-006
7	1426	0.00175844	5.344e-011	1.000e-006

Successful Termination After	7 Iterations
Criterion 0.001758437	Max Grad Entry 1.3873e-006
N. Function Calls 1427	N. Gradient Calls 1
Preproces. Time 0	Time for Method 4
Effective Time 4	

\*\*\*\*\*  
 Optimization Results  
 \*\*\*\*\*

Parameter Estimates  
 -----

Parameter	Estimate
1 X_1	0.04315280
2 X_2	0.19309098
3 X_3	0.26632889
4 X_4	0.50000037
5 X_5	0.50000005
6 X_6	0.73367128
7 X_7	0.80690931
8 X_8	0.95684718

Value of Objective Function = 0.00175844

This problem looks much milder when we are specifying some ineffective boundary constraints and use the BOBYQA technique:

```
/* Bound Const. Opt. BY Qadrat. Approx. : BOBYQA */
n = 8;
bc = cons(n,1,-100000.) -> cons(n,1,100000.);

print "Specified npt=17, minimum number of int points";
mopt = [ "tech" "bobyqa" ,
        "print"   4 ];
< xr,rp > = nlp(fcheby81,x0,mopt,bc);
```

BOBYQA Algorithm by M.J.D. Powell (2008)

Iter	nfun	optcrit	difcrit	rho
1	18	0.01930885	0.00000000	0.05000000
2	22	0.01930885	0.00000000	0.00500000
3	47	0.01358139	0.00572746	5.000e-004
4	226	0.00180071	0.01178068	5.000e-005
5	319	0.00175844	4.227e-005	7.071e-006
6	329	0.00175844	7.007e-010	1.000e-006
7	355	0.00175844	6.694e-011	1.000e-006

Successful Termination After	7 Iterations
Criterion 0.001758437	Max Grad Entry 9.6120e-007
Max Const Viol. 0.000000000	Max Grad LagF. 9.6120e-007
N. Active Constraints 0	
N. Function Calls 356	N. Gradient Calls 1
Preproces. Time 0	Time for Method 1
Effective Time 1	

Increasing the number of interpolation points to 32 shows a much better iteration history of NEWUOA:

```
print "specified number interpolation points: 32";
mopt = [ "tech" "uobyqa" ,
        "intpoi" 32 ,
        "print"   3 ];
< xr,rp > = nlp(fcheby81,x0,mopt);
print "rp=",rp; print "xr=";
```

NEWUOA Algorithm by M.J.D. Powell (2004)

Iter	nfun	optcrit	difcrit	rho
1	33	0.01930885	0.00000000	0.05000000
2	80	0.00725903	0.01204982	0.00500000
3	167	0.00325651	0.00400251	5.000e-004
4	625	0.00175847	0.00149805	5.000e-005
5	662	0.00175844	2.769e-008	7.071e-006
6	668	0.00175844	3.407e-010	1.000e-006
7	692	0.00175844	2.359e-010	1.000e-006

Successful Termination After 7 Iterations  
 Criterion 0.001758437 Max Grad Entry 7.2660e-007  
 N. Function Calls 693 N. Gradient Calls 1  
 Preproces. Time 0 Time for Method 2  
 Effective Time 2

\*\*\*\*\*  
 Optimization Results  
 \*\*\*\*\*

Parameter Estimates

-----

Parameter	Estimate
1 X_1	0.04315260
2 X_2	0.19309056
3 X_3	0.26632862
4 X_4	0.49999953
5 X_5	0.50000002
6 X_6	0.73367104
7 X_7	0.80690901
8 X_8	0.95684714

Value of Objective Function = 0.00175844

```
print "specified number interpolation points: 44";
mopt = [ "tech" "uobyqa" ,
        "intpoi" 44 ,
        "print" 3 ];
< xr,rp > = nlp(fcheby81,x0,mopt);
print "rp=",rp; print "xr=",xr;
```

NEWUOA Algorithm by M.J.D. Powell (2004)

Iter	nfun	optcrit	difcrit	rho
1	45	0.01930885	0.00000000	0.05000000
2	112	0.00541564	0.01389321	0.00500000
3	276	0.00176228	0.00365336	5.000e-004
4	321	0.00175844	3.837e-006	5.000e-005
5	342	0.00175844	9.477e-010	7.071e-006
6	345	0.00175844	2.608e-013	1.000e-006
7	362	0.00175844	1.498e-011	1.000e-006

```

Successful Termination After      7 Iterations
Criterion      0.001758437      Max Grad Entry  5.5522e-008
N. Function Calls      363      N. Gradient Calls      1
Preproces. Time      0      Time for Method      1
Effective Time      1
    
```

```

*****
Optimization Results
*****
    
```

Parameter Estimates

-----

Parameter	Estimate
1 X_1	0.04315274
2 X_2	0.19309079
3 X_3	0.26632868
4 X_4	0.49999996
5 X_5	0.49999994
6 X_6	0.73367125
7 X_7	0.80690912
8 X_8	0.95684722

Value of Objective Function = 0.00175844

The old UOBYQA algorithm performs excellent with the large number of interpolation points:

```

mopt = [ "tech" "uobyqa" ,
         "vers"   1 ,
         "print"  5 ];
< xr,rp > = nlp(fccheby81,x0,mopt);
    
```

```
print "rp=",rp; print "xr=",xr;
```

UOBYQA Algorithm by M.J.D. Powell (2002)

Iter	nfun	optcrit	difcrit	rho
1	46	0.01930885	0.00000000	0.05000000
2	136	0.00358936	0.01571949	0.00500000
3	236	0.00175990	0.00182946	5.000e-004
4	282	0.00175847	1.428e-006	5.000e-005
5	327	0.00175844	3.429e-008	7.071e-006
6	372	0.00175844	1.616e-010	1.000e-006
7	417	0.00175844	0.00000000	1.000e-006

Successful Termination After	7 Iterations		
Criterion	0.001758437	Max Grad Entry	2.8806e-008
N. Function Calls	418	N. Gradient Calls	1
Preproces. Time	0	Time for Method	1
Effective Time	1		

```
*****  
Optimization Results  
*****
```

Parameter Estimates

-----

Parameter	Estimate
1 X_1	0.04315276
2 X_2	0.19309084
3 X_3	0.26632871
4 X_4	0.50000000
5 X_5	0.50000000
6 X_6	0.73367129
7 X_7	0.80690916
8 X_8	0.95684724

Value of Objective Function = 0.00175844

The second example shows results for the Rosenbrock function for  $n = 2$ :

```
function frosbr1(x) {  
  /* crit = .5 * f' * f */  
  r1 = 10. * (x[2] - x[1] * x[1]);
```

```

r2 = 1. - x[1];
crit = .5 * (r1 * r1 + r2 * r2);
return(crit);
}

```

```

x0 = [ -1.2 1.];

```

```

mopt = [ "tech" "nmsimp" ,
        "print" 3 ];
< xr,rp > = nlp(frosbr1,x0,mopt);
print "rp=",rp; print "xr=",xr;

```

Nelder-Mead Simplex Optimization

Iteration Start:

N. Variables 2  
Criterion 12.10000000

Iter	rest	nfun	act	optcrit	difcrit	std	delta	size
1	0	12	0	2.34371	2.65588	1.0873	1.0000	0.38672
2	0	22	0	1.90558	0.14397	6e-002	1.0000	0.11365
3	0	32	0	1.47515	0.21064	9e-002	1.0000	0.37103
4	0	41	0	1.07103	0.17427	8e-002	1.0000	0.21831
5	0	51	0	0.78360	0.12162	5e-002	1.0000	0.09812
6	0	60	0	0.57881	0.05905	2e-002	1.0000	0.13249
7	0	69	0	0.33466	0.06830	3e-002	1.0000	0.14869
8	0	79	0	0.28183	5e-003	2e-003	1.0000	0.02794
9	0	88	0	0.21045	0.04213	2e-002	1.0000	0.06359
10	0	98	0	0.08917	0.06314	3e-002	1.0000	0.20860
11	0	108	0	0.02495	0.02190	9e-003	1.0000	0.15897
12	0	117	0	1e-002	0.01181	5e-003	1.0000	0.14038
13	0	125	0	5e-003	2e-003	9e-004	1.0000	0.10518
14	0	134	0	3e-004	4e-004	2e-004	1.0000	0.05466
15	0	143	0	1e-005	5e-005	2e-005	1.0000	0.01543
16	0	153	0	6e-007	9e-007	4e-007	1.0000	4e-003

Successful Termination After 16 Iterations

FCONV2 convergence criterion satisfied.

Criterion 6.1951e-007

N. Function Calls 155 Preproces. Time 0

Time for Method 0 Effective Time 0

\*\*\*\*\*

Optimization Results  
 \*\*\*\*\*

Parameter Estimates  
 -----

Parameter	Estimate
1 X_1	1.00098515
2 X_2	1.00191945

Value of Objective Function = 6.19514e-007

```
mopt = [ "tech" "uobyqa" ,
         "print"   5 ];
< xr,rp > = nlp(frosbr1,x0,mopt);
print "rp=",rp; print "xr=",xr;
```

NEWUOA Algorithm by M.J.D. Powell (2004)

Iter	nfun	optcrit	difcrit	rho
1	6	2.60000000	9.50000000	0.05000000
2	24	1.85569005	0.74430995	0.00500000
3	136	0.00343971	1.85225034	5.000e-004
4	163	5.532e-009	0.00343971	5.000e-005
5	171	1.077e-010	5.424e-009	7.071e-006
6	175	3.668e-014	1.077e-010	1.000e-006
7	180	4.339e-017	3.663e-014	1.000e-006

Successful Termination After	7 Iterations		
Criterion	5.9691e-019	Max Grad Entry	1.0871e-008
N. Function Calls	181	N. Gradient Calls	1
Preproces. Time	0	Time for Method	0
Effective Time	0		

\*\*\*\*\*  
 Optimization Results  
 \*\*\*\*\*

Parameter Estimates  
 -----

Parameter	Estimate
-----------	----------

```

1 X_1      1.00000000
2 X_2      1.00000000

```

Value of Objective Function = 5.96912e-019

```

mopt = [ "tech" "uobyqa" ,
         "vers"   1 ,
         "print"  5 ];
< xr,rp > = nlp(frosbr1,x0,mopt);
print "rp=",rp; print "xr=",xr;

```

UOBYQA Algorithm by M.J.D. Powell (2002)

Iter	nfun	optcrit	difcrit	rho
1	8	2.19261331	9.90738669	0.05000000
2	52	0.08467533	2.10793798	0.00500000
3	98	4.327e-007	0.08467490	5.000e-004
4	105	3.972e-009	4.288e-007	5.000e-005
5	110	8.791e-012	3.964e-009	7.071e-006
6	111	8.501e-014	8.706e-012	1.000e-006
7	115	7.764e-017	8.493e-014	1.000e-006

Successful Termination After		7 Iterations	
Criterion	3.9129e-021	Max Grad Entry	2.7737e-008
N. Function Calls	116	N. Gradient Calls	1
Preproces. Time	0	Time for Method	0
Effective Time	0		

```

*****
Optimization Results
*****

```

Parameter Estimates

```

-----
Parameter      Estimate
1 X_1          1.00000000
2 X_2          1.00000000

```

Value of Objective Function = 3.91288e-021



We conclude:

1. The Nelder-Mead algorithm converges fast for a rough precision, but will take a long time for high precision. Nelder-Mead does not need much memory and may also perform better for nonsmooth functions.
2. For high precision unconstrained optimization UOBYQA and NEWUOA are preferred to NMSIMP. For small  $n$  UOBYQA is preferred to NEWUOA, for large  $n$  UOBYQA may run out of memory.

For testing BOBYQA we ran the example which is attached to the software. First we specify the module for the objective function:

```
function fsrecip(x) {
  n = ncol(x);
  crit = 0.;
  for (i = 4; i <= n; i+=2)
  for (j = 2; j <= i-2; j+=2) {
    t1 = x[i-1] - x[j-1]; t2 = x[i] - x[j];
    tt = t1 * t1 + t2 * t2;
    if (tt < 1.e-6) tt = 1.e-6;
    crit += 1. / sqrt(tt);
  }
  return(crit);
}
```

The following is the CMAT specification for a simple run for  $n=10$ :

```
m = 5; n = 2 * m;
pi2 = 2. * macon("pi");
bc = cons(n,1,-1.) -> cons(n,1,1.);
x0 = cons(1,n,.);
for (j = k = 1; j <= m; j++, k+=2) {
  xin = (pi2 / (real)m) * j;
  x0[k] = cos(xin); x0[k+1] = sin(xin);
}
```

```
crit = fsrecip(x0);
print "F(x0)=",crit;
```

```
npt = 2*n + 1;
/* rhobeg = "instep" = 1.e-1;
   rhoend = "absxtol" = 1.e-6; */
mopt = [ "tech"    "bobyqa" ,
         "intpoi"   npt ,
```

```

    "instep"      .1 ,
    "absxtol"    1.e-6 ,
    "maxfun"     5000 ,
    "print"      4 ];
< xr,rp > = nlp(fsrecip,x0,mopt,bc);
print "m=",m," n=",n," npt=",npt;
print "rp=",rp; print "xr=",xr;

```

The first test run for m=5, i.e. n=10, gives the same results as they are reported by M. Powell (2008):

BOBYQA Algorithm by M.J.D. Powell (2008)

Iter	nfun	optcrit	difcrit	rho
1	44	5.60889786	1.27301174	0.01000000
2	59	5.60156025	0.00733762	0.00100000
3	73	5.60153398	2.627e-005	1.000e-004
4	79	5.60153397	6.357e-009	1.000e-005
5	94	5.60153397	0.00000000	1.000e-006
6	106	5.60153397	1.902e-011	1.000e-006

Successful Termination After	6 Iterations
Criterion 5.601533972	Max Grad Entry 1.338420835
Max Const Viol. 0.000000000	Max Grad LagF. 1.3362e-007
N. Active Constraints 9	
N. Function Calls 107	N. Gradient Calls 1
Preproces. Time 0	Time for Method 0
Effective Time 0	

\*\*\*\*\*  
 Optimization Results  
 \*\*\*\*\*

Parameter Estimates

Parameter	Estimate	Active BC
1 X_1	1.00000000	Upper BC
2 X_2	1.00000000	Upper BC
3 X_3	-1.00000000	Lower BC
4 X_4	1.00000000	Upper BC
5 X_5	-1.00000000	Lower BC
6 X_6	-1.00000000	Lower BC

7	X_7	1.00000000	Upper BC
8	X_8	-1.00000000	Lower BC
9	X_9	1.00000000	Upper BC
10	X_10	3.401e-008	

Value of Objective Function = 5.60153

The following CMAT input illustrates the entire test run for the example supplied with the software:

```

pi2 = 2. * macon("pi");
for (m = 5; m <= 10; m++) {
  n = 2 * m; x0 = cons(1,n,.);
  bc = cons(n,1,-1.) -> cons(n,1,1.);
  for (j = k = 1; j <= m; j++, k+=2) {
    xin = (pi2 / (real)m) * j;
    x0[k] = cos(xin); x0[k+1] = sin(xin);
  }

  crit = fsrecip(x0);
  print "m=",m," F(x0)=",crit;

  for (jc = 1; jc <= 2; jc++) {
    npt = (jc == 1) ? n + 6 : 2*n + 1;
    print "m=",m," jc=",jc;
    mopt = [ "tech"      "bobyqa" ,
             "intpoi"    npt ,
             "instep"    .1 ,
             "absxtol"   1.e-6 ,
             "maxfun"    5000 ,
             "print"     4 ];
    < xr,rp > = nlp(fsrecip,x0,mopt,bc);
    print "m=",m," jc=",jc," n=",n," npt=",npt;
    print "rp=",rp; print "xr=",xr;
  } }

```

We obtain the same results as are reported by Powell (2008).

## 4 New Developments

### 4.1 Function affarms

---

```
< gof,scor,lod,phi > = affarms(data,optn)
```

**Purpose:** This function performs an EM algorithm for estimating loadings and unique variances for the one factor model. The algorithm is especially designed for data which have a covariance matrices with only positive entries. This algorithm is almost the same as that of FARMS ("Factor Analysis for Robust Microarray Summarization") in the Bioconductor package of R but permits a few more options.

**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; depending on the option [3], the data may have to be strictly positive values (for the  $\log_2()$  transformation).

**optn** this is a vector of options, to maintain default values, the corresponding location should be set to a missing value.

The entries of the option vector specify:

1. the amount of printed output, can have int values 0,1, or 2, there is no printed output for `optn[1]=0`, `optn[1]=1` is default;
2. the version of the algorithm, `optn[2]=2` is the default and takes the least amount of memory allocation; `optn[2]=0` is the algorithm as implemented in R and needs most of the memory but permits to specify `optn[4]` setting negative correlations to zero;
3. if nonzero, this performs the  $\log_2$  transformation on the data (excluding negative data values), specifying zero will not apply the  $\log_2$  data transformation;
4. if nonzero and only for `optn[2]=0` the algorithm will set negative correlations to zero; for some really bad data sets, this will, however, generate an indefinite correlation matrix affecting the convergence of the EM algorithm;
5. the maximum number of EM iterations, default is 100;
6. the weight hyperparameter, default is 8. like for the R program;
7. the  $\mu$  hyperparameter, default is 0. like for the R program;
8. the scale hyperparameter, default is 1.5 like for the R program;
9. the tol hyperparameter as a termination tolerance for the iterations, default is 1.e-5 like for the R program;

**Output: gof** this is a vector returning some scalar information

**scor** this is an  $N$  vector of factor scores (weights)

**lod** this is an  $n$  vector of factor loadings  
**phi** this is an  $n$  vector of unique variances ( $\Phi$  is a diagonal matrix)

**Restrictions:**

**Relationships:** affvsn(), factor(), sem()

**Examples:** The  $20 \times 12$  data set SpikeIn is used from the *Bioconductor* package of R. Note, that there are more parameters to estimate (12 Loadings and 12 unique variances) than the data matrix has rows (20).

```

print "\n *** Test AFFARMS Function: SpikeIn\n";

#include "..\tdata\SpikeIn.dat"

print "AFFARMS: EM Factor method with log2 transform";
par = [ 3 , /* ipri */
        0 , /* ivrs=0: original */
        1 , /* iltr: log2 transform */
        1 , /* inul: set neg cors zero */
        1000 , /* maxi */
        8. , /* weight */
        0. , /* rmu */
        1.5 , /* scale */
        1.e-5 ]; /* tol */
< gof,expr,load,phi > = affarms(SpikeIn,par);

*****
Factor Analysis for Robust Mircoarray Summarization
*****

Number of Rows of Data. . . . . 20
Number of Columns of Data . . . . . 12
Number of Estimates . . . . . 24
Version of Algorithm. . . . . 0
Perform LOG2 Transform. . . . . Yes
Set Negative Correlations to Zero . . . . . Yes
Weight Parameter. . . . . 8.00000000
Mu Parameter. . . . . 0.00000000
Scale Parameter . . . . . 1.50000000
Maximum Number Iterations . . . . . 1000
Termination Tolerance . . . . . 1.0000e-005

```

\*\*\*\*\*

Iteration History for EM Algorithm

\*\*\*\*\*

Iter	Crit	CDiff
1	2.16402809	-2.16402809
2	1.41036860	0.75365949
3	1.09204379	0.31832481
4	1.01172312	0.08032067
5	0.99136063	0.02036249
6	0.97813464	0.01322599
7	0.96021672	0.01791792
8	0.93967842	0.02053830
9	0.92259122	0.01708719
10	0.91183425	0.01075698
.....		
80	0.91199423	-1.587e-005
81	0.91200936	-1.513e-005
82	0.91202378	-1.442e-005
83	0.91203752	-1.375e-005
84	0.91205063	-1.310e-005
85	0.91206312	-1.249e-005
86	0.91207503	-1.191e-005
87	0.91208639	-1.136e-005
88	0.91209722	-1.083e-005
89	0.91210754	-1.033e-005
90	0.91211739	-9.847e-006

Factor Scores

\*\*\*\*\*

Dense Column Vector (nrow=20)

C	Row_01	Row_02	Row_03	Row_04	Row_05
	7.9442043	7.7240800	10.272986	7.2886465	9.3966277
C	Row_06	Row_07	Row_08	Row_09	Row_10
	9.7247808	10.835891	9.9884600	10.128579	8.9622183
C	Row_11	Row_12	Row_13	Row_14	Row_15
	8.4960030	7.6794851	7.7761396	7.8356786	9.9490772
C	Row_16	Row_17	Row_18	Row_19	Row_20
	9.0497738	10.223175	8.1496402	8.2425788	9.7166332

### Unrotated Factor Loadings

\*\*\*\*\*

Dense Column Vector (nrow=12)

```
C |      0.50      0.75      1.00      1.50      2.00
    0.2472566  0.1833615  0.2912959  0.2249604  0.3516306

C |      3.00      5.00     12.50     25.00     50.00
    0.3307532  0.2845124  0.2832056  0.2413829  0.2467902

C |     75.00    150.00
    0.2097712  0.2106438
```

### Unique Variances

\*\*\*\*\*

Dense Column Vector (nrow=12)

```
C |      0.50      0.75      1.00      1.50      2.00
    0.0657080  0.0333292  0.0117516  0.0036930  0.0034226

C |      3.00      5.00     12.50     25.00     50.00
    0.1883581  0.3573857  0.4219595  0.5632054  0.5633828

C |     75.00    150.00
    0.4961404  0.2817855
```

## 4.2 Function `affvsn`

---

```
< gof,dnew,mu > = affvsn(data,optn<,strata<,ref>>)
```

**Purpose:** The `vsn` function implements the Huber et al. (2003) algorithm for "Variance Stabilizing Normalization" of the columns of a matrix of microarray data. The implementation is very similar to that of the `Bioconductor` function in R. The algorithm's outer cycle is a fast version of LTS (Least Trimmed Squares, see Rousseeuw & Leroy, 1987) for the robust estimation of a nonlinear model predicting the values of a new data set with normalized columns. Each iteration selects a new subset of rows, its size is defined by the user specification of the quantile `optn[2]`. The innermost part of the algorithm consists in the estimation of the parameters of a nonlinear model by means of an optimization algorithm. If there are no row strata specified, the algorithm computes  $npar = 2 * n$  optimal

parameter estimates. When there are  $n_s$  row strata specified the number of estimates increases to  $npar = 2 * n_s * n$  model parameters. Test computations show that the results of the optimizations may differ considerably depending on the initial values. The model obviously does not restrict the parameter estimates to a unique solution.

**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.

**optn** this is a vector of options, to maintain default values, the corresponding location should be set to a missing value.

**strata** The rows can be divided into  $n_s$  groups (strata, clusters) for which separate sets of parameters are estimated. If specified **strata** should be an integer vector with  $n_s + 1$  monoton increasing values indicating the start and end index of each strata. It is required that  $strata[n_s + 1] = m$ .

**ref**

The entries of the option vector specify:

1. the amount of printed output, can have int values 0,1, or 2, there is no printed output for `optn[1]=0`, `optn[1]=1` is default;
2. the quantile determining the subsample size for the fast LTS algorithm, which must be in  $(0, 1]$ . Default is `optn[2]=.9`. For `optn[2]=1`. no LTS subsampling is done and the algorithm terminates with estimates for the complete data set of  $N$  rows.
3. the size  $n_r < N$  of a reduced data set, usually needed only when  $N$  is too large for the computer resources. Other than in R default is  $n_r = N$  which is the same as selecting `optn[3]=0`.
4. an integer specifying the number of LTS iterations which is only useful for `optn[2] < 1`.. Default is `optn[4]=7`.
5. a real value in  $(0, 1)$  for an LTS termination criterion. Default is `optn[5]=1.e-4`.
6. the amount of printed output for the optimization. Default is `optn[6]=0`, i.e. no printed optimization history.
7. an integer specifying the number of iterations for each optimization. Default is `optn[7]=1000`.
8. a real value in  $(0, 1)$  for an absolute gradient criterion for terminating the optimization. Default is `optn[8]=2.e-4`.
9. a real value in  $(0, 1)$  for a relative gradient criterion for terminating the optimization. Default is `optn[9]=1.e-8`.

**Output: gof**

**dnew**



mu

Restrictions: 1.

Relationships: affarms()

Examples: 1. :

```
print "\n *** Test AFFVSN Function: AffyBatch\n";

#include "..\\tdata\\affybatch.dat"

affb1 = affb0[1:1000,];
affb2 = affb0[1:100,];
free affb0;
print "AFFB2=",affb2[1:10,];

print "AFFVSN: Nonlinear Variance Stabilizing Normalization";
par = [ 3 , /* ipri */
       .9 , /* quantile */
       0 , /* nsamp: subsampling */
       7 , /* itlts */
       1.e-4 , /* epslts */
       3 , /* popt */
       1000 , /* maxi */
       1.e-3 , /* agtl */
       2.e-4 ]; /* gtol */
< gof,dnew,mu > = affvsn(affb2,par);

*****
VSN: Column Normalization by Variance Stabilization
*****

Number of Rows of Data . . . . . 100
Number of Columns of Data . . . . . 3
Number of Estimates . . . . . 6
LTS Quantile . . . . . 0.900000000
LTS Iterations . . . . . 7
LTS Tolerance . . . . . 0.000100000

Name          Mean      Std Dev   Skewness   Kurtosis
-----
Col_1         529.2770000  400.294769  0.03555787 -2.01056548
Col_2         349.2160000  220.212221  0.02998792 -1.95555250
Col_3         544.9800000  443.067630  0.05430788 -1.98246485
```

\*\*\*\*\*  
 LTS Iteration History  
 \*\*\*\*\*

Iter	Nselect	Nchanged	MaxChange
0	100	0	.
1	90	0	0.00000000

Transformed Data Set  
 \*\*\*\*\*

Dense Matrix (100 by 3)

	Col_1	Col_2	Col_3
Row_001	10.128050	10.118748	9.9004655
Row_002	8.5891019	8.9712471	8.6346973
Row_003	10.194177	10.240599	10.029783
Row_004	8.5881425	8.7989034	8.6092785
Row_005	10.196277	10.030345	9.9352053
Row_006	8.5992966	8.4322128	8.5978359
Row_007	10.114787	10.138327	10.036156
Row_008	8.6393696	8.7800815	8.6318950
Row_009	10.191758	10.202046	10.135753
Row_010	8.6659850	9.0978430	8.6486277
.....			
Row_090	8.6322506	8.7421965	8.5891940
Row_091	10.046225	10.162433	10.022633
Row_092	8.5906996	8.5421280	8.6206311
Row_093	9.9411850	10.038704	10.056152
Row_094	8.6291443	8.5362364	8.5949611
Row_095	10.048901	10.152839	10.130998
Row_096	8.6750500	8.6070829	8.5834039
Row_097	10.097697	10.141241	10.144333
Row_098	8.5992966	8.7396351	8.5929452
Row_099	10.107774	10.086455	10.239449
Row_100	8.5881425	8.7189785	8.6064264

Vector of Means  
 \*\*\*\*\*

Dense Column Vector (nrow=100)

C	Row_001	Row_002	Row_003	Row_004	Row_005
	6.8296883	5.9165321	6.9029990	5.8706176	6.8330531

C	Row_006	Row_007	Row_008	Row_009	Row_010
	5.7858274	6.8624985	5.8833303	6.9180166	5.9667643
.....					
C	Row_091	Row_092	Row_093	Row_094	Row_095
	6.8491028	5.8145037	6.8039903	5.8160940	6.8725418
C	Row_096	Row_097	Row_098	Row_099	Row_100
	5.8403992	6.8842173	5.8557270	6.8958640	5.8514920

Scalar Standard Deviation 0.00154819  
 Number of LTS Iterations (Optimizations) 2

### 4.3 Function decrypt

---

`decrypt(ofil,ifil<,pwd>)`

**Purpose:** The `decrypt` function can be used to decrypt formerly encrypted files or all encrypted files of a specified directory. It is assumed that the correct password used in the encryption of the file is still available. There are two ways to specify this password:

- as the same full (with at least 128 characters) string with the third input argument `pwd` that was used at the encryption.
- if the third input argument is not specified, then the output password file from the encrypting run is read from the `cmat save` directory.

The function `decrypt` is useful only for input files which were generated by function `encrypt`.

**Input: ofil** This must be a string specifying the path

- either for an output directory
- or for an output file.

**ifil** This must be a string specifying the path

- either for an input directory
- or for an input file.

The file should have an extension, preferred but not necessarily `.enc`.

**pwd** If this argument is specified it must be a string with (at least) 128 characters which is the same as has been used to obtain the encrypted file `ifil`. If this argument is not specified, it is assumed that the `cmat` `save` directory contains the password file written when using the `encrypt` function.

**Output:** The path name for the (hopefully readable) output file is specified as the first input argument.

- Restrictions:**
1. To differentiate between file and directory names the function assumes that file names include an extension at the end of the string separated from the name with a dot.
  2. Pathnames to files and directories must not contain any white space.
  3. If the 128 character password (file) is lost, an encrypted file cannot be decrypted anymore.
  4. The password and the encrypted files are written in binary mode.

**Relationships:** `encrypt()`

**Examples:** 1. File Encryption:

(a) No password specified:

The encrypted file is written to `spaeth1.enc` and the 128 byte password file is written to the `..\save` directory.

```
ipath = "..\tdata\spaeth.dat";
opath = "spaeth1.enc";
encrypt(opath,ipath);
```

The password file written during the encryption is picked up in the `..\save` directory and the `spaeth1.enc` file is decrypted to the `spaeth1.txt` file:

```
ipath = "spaeth1.enc";
opath = "spaeth1.txt";
decrypt(opath,ipath);
```

(b) Short Password specified:

The first 6 of the 128 bytes of the password are user specified, the remaining 122 bytes are computer generated when the file `spaeth.dat` is encrypted to the file `spaeth2.enc`:

```
ipath = "..\tdata\spaeth.dat";
opath = "spaeth2.enc";
passw = "bully";
encrypt(opath,ipath,passw);
```

For decryption the password file must be available in the ..\save directory for reading:

```
ipath = "spaeth2.enc";
opath = "spaeth2.txt";
decrypt(opath,ipath);
```

(c) Long Password specified:

The entire 128 byte password is user specified. No password file is written during the encryption:

```
print "Long Password specified: Length=128";
ipath = "..\tdata\spaeth.dat";
opath = "spaeth3.enc";
pass1 = "bullybummybullybummybullybummybullybummybullybummy";
pass2 = "bullybummybullybummybullybummybullybummybullybummy";
pass3 = "bullybummybullybummybullybum";
passw = strcat(pass1,strcat(pass2,pass3));
encrypt(opath,ipath,passw);
```

If the entire password is specified for decryption no password file is picked up:

```
ipath = "spaeth3.enc";
opath = "spaeth3.txt";
decrypt(opath,ipath,passw);
```

## 2. Directory Encryption:

(a) No password specified:

All files of the ..\csrc\ode directory are encrypted and written to the ..\csrc\ode\odenc directory with an .enc extension. If the directory does not exist at the specified location it will be created.

```
print "*** De- and Encrypt Directory ***";
print "No password specified";
ipath = "..\csrc\ode";
opath = "..\csrc\ode\odenc";
encrypt(opath,ipath);
```

All encrypted files in the ..\csrc\ode\odenc are decrypted and written to the ..\csrc\ode\oddec directory with an .dec extension:

```
ipath = "..\csrc\ode\odenc";
opath = "..\csrc\ode\oddec";
decrypt(opath,ipath);
```

(b) Long Password specified:

```
print "Long Password specified: Length=128";
ipath = "..\\csrc\\ode";
opath = "..\\csrc\\ode\\odenc";
pass1 = "bullybummybullybummybullybummybullybummybullybummy";
pass2 = "bullybummybullybummybullybummybullybummybullybummy";
pass3 = "bullybummybullybummybullybum";
passw = strcat(pass1, strcat(pass2, pass3));
encrypt(opath, ipath, passw);

ipath = "..\\csrc\\ode\\odenc";
opath = "..\\csrc\\ode\\oddec";
decrypt(opath, ipath, passw);
```

## 4.4 Function encrypt

---

encrypt(ofil, ifil<, pwd>)
----------------------------

**Purpose:** The `encrypt` function can be used for the save encryption of the content of files. You can provide a password string (no more than 128 characters are being used). If no password or a shorter password is provided it will be automatically generated up to a length of 128 characters.

If the specified password is shorter than 128 characters it is written to the `cmat`

`save` directory as a file with the file or directory name for the base and `.enc` as extension. To be on the save side, the password file could be saved on some external memory away from the encrypted file, especially when the computer is connected to the internet. If the specified password is at least 128 characters long no password file is written and the user must specify the same string for the `decrypt` function. Preferred are sentences from a book with white space removed.

**Input: ofil** This must be a string specifying the path

- either for an output directory
- or for an output file.

The output file should have an extension, preferred but not necessary `.enc`.

**ifil** This must be a string specifying the path

- either for an input directory
- or for an input file.

**pwd** If this argument is specified it must be a string, no more than 128 characters are used. If the string is shorter than 128 characters, it will be extended to 128 characters for a complete password. If this argument is not specified, a 128 character password is machine generated. If no password string or a password string with less than 128 characters is specified a password file is written to the **cmat save** directory which is then picked up for the decryption of the file.

**Output:** The path name for the encrypted output file is specified as the first input argument.

- Restrictions:**
1. To differentiate between file and directory names the function assumes that file names include an extension at the end of the string separated from the name with a dot.
  2. Pathnames to files and directories must not contain any white space.
  3. If the 128 character password (file) is lost, an encrypted file cannot be decrypted anymore.
  4. The password and the encrypted files are written in binary mode.

**Relationships:** decrypt()

**Examples:** 1. File Encryption:

- (a) No password specified:

The encrypted file is written to **spaeth1.enc** and the 128 byte password file is written to the **..\save** directory.

```
ipath = "..\tdata\spaeth.dat";
opath = "spaeth1.enc";
encrypt(opath,ipath);
```

The password file written during the encryption is picked up in the **..\save** directory and the **spaeth1.enc** file is decrypted to the **spaeth1.txt** file:

```
ipath = "spaeth1.enc";
opath = "spaeth1.txt";
decrypt(opath,ipath);
```

- (b) Short Password specified:

The first 6 of the 128 bytes of the password are user specified, the remaining 122 bytes are computer generated when the file **spaeth.dat** is encrypted to the file **spaeth2.enc**:

```
ipath = "..\tdata\spaeth.dat";
opath = "spaeth2.enc";
passw = "bully";
encrypt(opath,ipath,passw);
```

For decryption the password file must be available in the ..\save directory for reading:

```
ipath = "spaeth2.enc";
opath = "spaeth2.txt";
decrypt(opath,ipath);
```

(c) Long Password specified:

The entire 128 byte password is user specified. No password file is written during the encryption:

```
print "Long Password specified: Length=128";
ipath = "..\tdata\spaeth.dat";
opath = "spaeth3.enc";
pass1 = "bullybummybullybummybullybummybullybummybullybummy";
pass2 = "bullybummybullybummybullybummybullybummybullybummy";
pass3 = "bullybummybullybummybullybum";
passw = strcat(pass1,strcat(pass2,pass3));
encrypt(opath,ipath,passw);
```

If the entire password is specified for decryption no password file is picked up:

```
ipath = "spaeth3.enc";
opath = "spaeth3.txt";
decrypt(opath,ipath,passw);
```

## 2. Directory Encryption:

(a) No password specified:

All files of the ..\csrc\ode directory are encrypted and written to the ..\csrc\ode\odenc directory with an .enc extension. If the directory does not exist at the specified location it will be created.

```
print "*** De- and Encrypt Directory ***";
print "No password specified";
ipath = "..\csrc\ode";
opath = "..\csrc\ode\odenc";
encrypt(opath,ipath);
```

All encrypted files in the ..\csrc\ode\odenc are decrypted and written to the ..\csrc\ode\oddec directory with an .dec extension:

```
ipath = "..\csrc\ode\odenc";
opath = "..\csrc\ode\oddec";
decrypt(opath,ipath);
```



(b) Long Password specified:

```
print "Long Password specified: Length=128";
ipath = "..\\csrc\\ode";
opath = "..\\csrc\\ode\\odenc";
pass1 = "bullybummybullybummybullybummybullybummybullybummy";
pass2 = "bullybummybullybummybullybummybullybummybullybummy";
pass3 = "bullybummybullybummybullybum";
passw = strcat(pass1, strcat(pass2, pass3));
encrypt(opath, ipath, passw);

ipath = "..\\csrc\\ode\\odenc";
opath = "..\\csrc\\ode\\oddec";
decrypt(opath, ipath, passw);
```

## 4.5 Function locatn

`<gof,xind,yind,ofun,lm> = locatn(cmat<,par<,dvec>>)`

**Purpose:** For a given  $m \times n$  data matrix  $\mathbf{X} = (x_{ij})$ , an  $n$  vector  $d = (d_j)$ , which can be zero, and a specified integer  $K > 1$  the `locatn` function implements an algorithm that computes an approximate solution to the following integer LP:

$$z = \max \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} - \sum_{j \in J} d_j y_j$$

subject to:

$$\sum_{j \in J} x_{ij} = 1 \quad , \quad i \in I,$$

$$1 \leq \sum_{j \in J} y_j \leq K,$$

$$0 \leq x_{ij} \leq y_j \leq 1 \quad , \quad i \in I, j \in J,$$

$$x_{ij}, y_j \text{ are integral, } i \in I, j \in J.$$

In a location problem there are  $m$  clients  $i$  which should be assigned to  $K \leq n$  locations out of a total of  $n$  potential locations  $j$ .

There are a number of different algorithms solving this problem:

1. An optimal solution can be computed by an integer LP. However, for very large  $m$  and  $n$  it is very time consuming to solve the integer LP exactly. This is not tried here.
2. A greedy algorithm only approximates the exact solution. A heuristic is implemented here.
3. Also the here implemented Lagrangean relaxation obtains only an approximation to the exact solution.

A value for  $K \leq n$  can be specified ( $n$  is the default). However, it may happen that the objective function is not increasing anymore for  $K_r \leq K$  selected locations. For the greedy algorithm the optimal solution refers to only  $K_r$  locations.

Since the optimal solution is from a nonsmooth problem involving the `max` function in the objective, multiple solutions can be expected. For larger estimation problems the solutions usually are not so different.

**Input:** `cmat` is an  $m \times n$  matrix  $\mathbf{C} = (c_{ij})$  specifying the profit which is made when client  $i$  is using the facility at location  $j$ .

`par` is an option vector which should be initialized with missing values for defaults. The following entries can be specified:

1.  $K$  the number of facilities to select; default is  $n$ ;
2. specifies the amount of printed output; default is zero, i.e. no printed output.
3. specifies the method:
  - (a) `par[2]=0`: this is the greedy algorithm (heuristic)
  - (b) `par[2]=1`: this is Lagrangean relaxation algorithm (optimization of a nonsmooth function)
  - (c) `par[2]=2`: first the greedy algorithm is used for starting estimates of the Lagrangean multipliers for the nonsmooth optimization (Lagrangean relaxation)
4. specifies the amount of printed output for the nonsmooth optimization algorithm; default is zero, i.e. no printed output.

**dvec** is an  $n$  vector specifying the cost of maintaining a potential facility at location  $j$ . This argument is optional. If not specified it is assumed that  $d_j = 0, j = 1, \dots, n$ .

**Output: gof** this is vector of some scalar results, like

1. failure of the run,
2. computation time,
3. a possibly modified value  $K_r$  for  $K$ ,
4. the function value  $z_g$  for the solution at  $K_r$ , this is a lower bound of the optimal solution  $z_{LP}$
5. the function value at  $K_r + 1$  which should be an upper bound  $z_d$  of the optimal solution  $z_{LP}$
6. a lowest bound  $z_r$

**xind** this is an  $m$  vector  $j = \text{xind}[i]$  which defines the best location  $j$  for the client  $i$ ; it only refers to  $K_r \leq K$  locations

**yind** is a vector with  $n$  integers  $1, \dots, n$  specifying a rank order for selecting  $K_r$  of the  $n$  locations for servicing the clients, where  $j_1 = \text{yind}[1]$  is the first selected location  $j_1$  and  $j_n = \text{yind}[n]$  gives the last selected location  $j_n$ .

**ofun** this is a  $n$  vector with the approximative function values  $z_g$  when there are  $1, \dots, n$  locations are selected. The  $z_g$  is a lower bound,  $z_g \leq z_{LP}$ , for the objective function value  $z_{LP}$  maximizing  $z$ . Note, that these function values must not necessarily monoton increase and may decrease or stagnate after  $K_r$  when too many facilities  $K$  are selected.

**lm** this is an  $m$  vector  $u_i$  which gives the Lagrange multipliers for  $K_r$  selected locations.

**Restrictions:** The data cannot contain any missing values.

**Relationships:** `lp()`, `nlp()`

**Examples:** 1. Very small illustrative example:  $m = n = 4$  with  $d = 0$ :

```
c = [ 0 11 6 9 ,
      7 0 8 2 ,
      7 3 0 3 ,
      10 9 4 0 ];
optn = [ 2 , /* K */
         2 ];
< xind,yind,ofun,lm > = locatn(c,optn);
```

```
Number Selected Locations . . . . . 2
Lowest Bound zr . . . . . 0.0000000
Greedy Algorithm zg . . . . . 35.0000000
Upper Bound zd. . . . . 36.0000000
```

Assignment of Clients to Locations  $X_{ij}$  for  $K_{max}=2$

```

Client_1      1
Client_1      2
Client_2      1
Client_3      1
Client_4      1
```

Selected Locations  $Y_j$  in Order of Relevance

```

      Loc_1      Loc_2      Loc_3      Loc_4
1      1          2          3          4
```

Objective Function Values for Stepwise Algorithm

```

      Loc_1      Loc_2      Loc_3      Loc_4
1 24.00000000 35.00000000 36.00000000 36.00000000
```

Lagrange Multipliers  $U$  for  $K_{max}=2$

```

Client_1      1
Client_1      11.0000000
Client_2      7.0000000
Client_3      7.0000000
Client_4      10.0000000
```

Here we only show the `gof` return:

```

GOF=
      |          1
-----
Failure | 0.00000
Time    | 0.00000
Kmax    | 2.0000
z_g     | 35.000
z_d     | 36.000
z_r     | 0.00000
unused  | .
unused  | .

```

2. A slightly larger estimation problem with the swimming team data from the SAS/OR manual for the PROC ASSIGN function:

```

c0 = [ 1 35.1 36.7 28.3 36.1 ,
      1 34.6 32.6 26.9 26.2 ,
      1 31.3 33.9 27.1 31.2 ,
      1 28.6 34.1 29.1 30.3 ,
      1 32.9 32.2 26.6 24.0 ,
      1 27.8 32.5 27.8 27.0 ,
      2 26.3 27.6 23.5 22.4 ,
      2 29.0 24.0 27.9 25.4 ,
      2 27.2 33.8 25.2 24.1 ,
      2 27.0 29.2 23.0 21.9 ];

cnam = [" sex back breast fly freest "];
rnam = [" sue karen jan andrea carol ellen jim mike sam clayton "];
c0 = cname(c0,cnam);
c0 = rname(c0,rnam); print c0;

c = c0[,2:5];
cnam = [" back breast fly freest "];
rnam = [" sue karen jan andrea carol ellen jim mike sam clayton "];
c = cname(c,cnam);
c = rname(c,rnam); print c;
m = nrow(c); n = ncol(c);

```

First we only run the greedy algorithm:

```

optn = [ 2 , /* Kmax */
        3 ];
gof = locatn(c,optn);

```

```

Number Selected Locations . . . . . 2
Lowest Bound zr . . . . . 253.60000
Greedy Algorithm zg . . . . . 324.30000
Upper Bound zd for Heuristic. . . . . 324.30000

```

Assignment of Clients to Locations  $X_{ij}$  for  $K_{max}=2$

```

sue 1
karen 2
jan 1
andrea 2
carol 2
ellen 1
jim 2
mike 2
sam 1
clayton 2

```

Selected Locations  $Y_j$  in Order of Relevance

```

breast back fly freest
1 2 1 3 4

```

Objective Function Values for Stepwise Algorithm

```

1 316.600000 324.300000 324.300000 324.300000

```

Lagrange Multipliers  $U$  for  $K_{max}=2$

```

sue 1
sue 36.700000
karen 34.600000
jan 33.900000
andrea 34.100000
carol 32.900000
ellen 32.500000
jim 27.600000
mike 29.000000
sam 33.800000
clayton 29.200000

```

Now, we only use the Lagrangean relaxation algorithm:

```

optn = [ 2 , /* Kmax */
        3 , /* ipri */
        1 , /* imet: LR */
        3 ]; /* poprt */
< gof,xind,yind,ofun,lm > = locatn(c,optn);

```

```

*****
Optimization Start
*****

```

```

Parameter Estimates
-----

```

Parameter	Estimate	Gradient
1	1.00000000	-3.00000000
2	1.00000000	-3.00000000
3	1.00000000	-3.00000000
4	1.00000000	-3.00000000
5	1.00000000	-3.00000000
6	1.00000000	-3.00000000
7	1.00000000	-3.00000000
8	1.00000000	-3.00000000
9	1.00000000	-3.00000000
10	1.00000000	-3.00000000

Value of Objective Function = 1120.4

Bundle Trust Region Method (Outrata-Schramm-Zowe, 1991)  
Convex Objective Function Assumed  
Internally Specified Gradient

Iteration Start:

```

N. Variables      10
Criterion         1120.400000      Max Grad Entry  3.000000000

```

Iter	nfun	act	optcrit	maxgrad	gradnrm	alpha	rho
2	1	2	1120.400	3.000000	3.71e-004	832.76741	3853.088
2	2	2	1120.400	2.000000	5.86e-005	832.79486	3853.088
2	3	2	1120.400	2.000000	9.24e-006	832.79919	3853.088
2	4	2	1120.400	2.000000	1.46e-006	832.79987	3853.088

2	5	2	1120.400	2.000000	7.83e-007	832.79993	3853.088
3	6	3	353.2800	2.000000	1.84e-007	65.679996	213.5575
4	7	4	353.2800	3.000000	5.20e-007	65.679970	1697.445
5	8	5	353.2800	3.000000	1.07e-006	65.679872	7194.059
6	9	3	353.2800	3.000000	1.1321008	49.587670	8.1e+013
7	10	2	353.2800	3.000000	0.7497560	49.887988	3.5e+011
7	11	2	353.2800	3.000000	0.7497560	49.887887	3.5e+009
7	12	2	353.2800	3.000000	0.7497561	49.886872	35352845
7	13	2	353.2800	3.000000	0.7497627	49.876720	353534.7
7	14	2	353.2800	3.000000	0.7504265	49.775201	3541.609
7	15	2	353.2800	3.000000	1.4595171	37.252621	133.9685
8	16	2	338.7183	3.000000	0.9080501	32.922616	51.85659
9	17	3	338.7183	1.000000	1.2785762	17.787648	102.8105
10	18	4	338.7183	1.000000	0.8328594	17.169615	43.62423
.....							
18	27	1	324.3000	0.000000	0.0000000	0.0000000	0.000000

Successful Termination After 18 Iterations

ABSGCONV convergence criterion satisfied.

Criterion	324.3000000	Max Grad Entry	0.000000000
N. Grad Storage	10		
N. Function Calls	28	N. Gradient Calls	28
Preproces. Time	0	Time for Method	0
Effective Time	0		

Objective function seems to be convex.

\*\*\*\*\*  
 Optimization Results  
 \*\*\*\*\*

Parameter Estimates

Parameter	Estimate	Gradient
1	36.2682404	0.0000000
2	33.6400467	0.0000000
3	31.7469605	0.0000000
4	33.7973778	0.0000000
5	32.5500000	0.0000000
6	29.7953133	0.0000000
7	27.5861164	0.0000000
8	28.0852817	0.0000000
9	29.0149622	0.0000000
10	28.0974999	0.0000000



Value of Objective Function = 324.3

Number Selected Locations . . . . . 2  
Lowest Bound zr . . . . . 253.60000  
Greedy Algorithm zg . . . . .  
Upper Bound zd for Lagr. Relaxation . . . . . 324.30000

Assignment of Clients to Locations X<sub>ij</sub> for K<sub>max</sub>=2

\*\*\*\*\*

Dense Column Vector (nrow=10)

C | sue karen jan andrea carol ellen jim  
2 1 2 2 1 2 2

C | mike sam clayton  
1 2 2

Selected Locations Y<sub>j</sub> in Order of Relevance

\*\*\*\*\*

Dense Row Vector (ncol=4)

R | breast back fly freest  
2 1 3 4

Lagrange Multipliers U for K<sub>max</sub>=2

\*\*\*\*\*

Dense Column Vector (nrow=10)

C | sue karen jan andrea carol  
36.268240 33.640047 31.746960 33.797378 32.550000

C | ellen jim mike sam clayton  
29.795313 27.586116 28.085282 29.014962 28.097500

With specifying `optn[3]=2` a combined algorithm is run: using the greedy stepwise algorithm for initial estimates of the Lagrangean multipliers and then using nonsmooth optimization for the Lagrangean relaxation:

```

optn = [ 2 , /* Kmax */
        3 , /* ipri */
        2 , /* imet: both */
        3 ]; /* popt */
< gof,xind,yind,ofun,lm > = locatn(c,optn);

```

Assignment of Clients to Locations X<sub>ij</sub> for K<sub>max</sub>=2  
(Initial) Solution by Greedy Algorithm

	1
sue	2
karen	1
jan	2
andrea	2
carol	1
ellen	2
jim	2
mike	1
sam	2
clayton	2

Lagrange Multipliers U for K<sub>max</sub>=2  
(Initial) Solution by Greedy Algorithm

	1
sue	36.700000
karen	34.600000
jan	33.900000
andrea	34.100000
carol	32.900000
ellen	32.500000
jim	27.600000
mike	29.000000
sam	33.800000
clayton	29.200000

Bundle Trust Region Method (Outrata-Schramm-Zowe, 1991)  
Convex Objective Function Assumed  
Internally Specified Gradient

Iteration Start:

N. Variables	10	
Criterion	324.300000	Max Grad Entry 0.00000000

```

Iter  nfun act  optcrit  maxgrad  gradnrm  alpha  rho

Successful Termination After      0 Iterations
ABSGCONV convergence criterion satisfied.
Criterion      324.3000000          Max Grad Entry  0.000000000
N. Grad Storage      10
N. Function Calls    1          N. Gradient Calls    1
Preproces. Time      0          Time for Method      1
Effective Time       1
Objective function seems to be convex.

```

```

*****
Optimization Results
*****

```

```

Parameter Estimates
-----

```

Parameter	Estimate	Gradient
1	36.7000000	0.0000000
2	34.6000000	0.0000000
3	33.9000000	0.0000000
4	34.1000000	0.0000000
5	32.9000000	0.0000000
6	32.5000000	0.0000000
7	27.6000000	0.0000000
8	29.0000000	0.0000000
9	33.8000000	0.0000000
10	29.2000000	0.0000000

Value of Objective Function = 324.3

```

Number Selected Locations . . . . . 2
Lowest Bound zr . . . . . 253.60000
Greedy Algorithm zg . . . . . 324.30000
Upper Bound zd for Heuristic. . . . . 324.30000
Upper Bound zd for Lagr. Relaxation . . . . . 324.30000

```

```

Assignment of Clients to Locations X_ij for K_max=2
*****

```

```

Dense Column Vector (nrow=10)
C |      sue   karen   jan  andrea  carol  ellen  jim
   |      2     1     2     2     1     2     2
C |  mike    sam clayton
   |     1     2     2

```

Selected Locations Y\_j in Order of Relevance  
 \*\*\*\*\*

```

Dense Row Vector (ncol=4)
R |      back breast  fly freest
   |          1     2     3     4

```

Objective Function Values for Stepwise Algorithm

```

      back      breast      fly      freest
1  316.600000  324.300000  324.300000  324.300000

```

Lagrange Multipliers U for K\_max=2  
 \*\*\*\*\*

```

Dense Column Vector (nrow=10)
C |      sue      karen      jan      andrea      carol
   |  36.700000  34.600000  33.900000  34.100000  32.900000
C |      ellen      jim      mike      sam      clayton
   |  32.500000  27.600000  29.000000  33.800000  29.200000

```

As we can easily verify the values of the Lagrange Multipliers obtained from the greedy algorithm are optimal. Therefore, we use bad starting values here:

## 4.6 Function log2

---

$$y = \text{log2}(z)$$

**Purpose:** Returns an approximation of the base-2 logarithm function.

**Input:** The argument  $z$  must be numeric. If the argument  $z$  is negative or complex, the result is complex.

$$\text{log2}(z) = \ln(z) / \ln(2)$$

**Output:** A missing value is returned if argument  $z$  is string or missing value. If the argument is vector or matrix, the `log2` function is computed elementwise.

**Restrictions:** If argument  $z \leq 0$ , a missing value is returned.

**Relationships:** `log10()`, `log()`

**Examples:** `a = log2(5);` produces `a = 2.3219`.

## 4.7 Function mad

---

```
v = mad(a<,optn>)
```

**Purpose:** This function returns the unscaled or scaled MAD (median absolute deviation)

- as a scalar for an input vector **a** or
- for  $m \times n$  input matrix **a** either a  $n$  vector for columnwise (default) or a  $m$  vector for rowwise (see `optn[2]`) treatment.

For the MAD function see also `univar()` function.

**Input:** **a** should be a real or int matrix or vector.

`optn` is a vector of options which should be initialized to missing for the default options.

1. amount of printed output (default=0, no output)
2. =0: colwise treatment, =1: rowwise treatment (default=0)
3. get scaled MAD (multiply with factor 1.48)

**Output:** The result is either a real scalar or a vector depending on the input **a**.

**Restrictions:** The data **a** may have missing values.

**Relationships:** `univar()`, `median()`

**Examples:** `print "Heart data, D.M. Hawkins (1994)";`

```
a= [ 1 42.8 40.0 37,  
    2 63.5 93.5 50,  
    3 37.5 35.5 34,  
    4 39.5 30.0 36,  
    5 45.5 52.0 43,  
    6 38.5 17.0 28,  
    7 43.0 38.5 37,  
    8 22.5  8.5 20,  
    9 37.0 33.0 34,  
   10 23.5  9.5 30,  
   11 33.0 21.0 38,  
   12 58.0 79.0 47 ];  
aa = a[,2:4];
```

```
sopt= [ "ari" "med" "mad" ];  
c1 = univar(aa,sopt);  
print "\n Compare Measures with MAD and MEDIAN\n",c1;
```

Compare Measures with MAD and MEDIAN

	Var_1	Var_2	Var_3
Ari_Mean	40.358	38.125	36.167
Median	39.000	34.250	36.500
MAD	5.0000	15.250	4.5000

```
opt = cons(3,1,.);
opt[1] = 1; /* ipri */
mad1 = mad(aa,opt);
print "MAD1=",mad1;
```

Unscaled Columnwise MAD  
\*\*\*\*\*

Dense Row Vector (ncol=3)

R	1	2	3
	5.0000000	15.250000	4.5000000

```
opt = cons(3,1,.);
opt[1] = 1; /* ipri */
opt[3] = 1; /* scaling */
mad2 = mad(aa,opt);
print "MAD2=",mad2;
```

Scaled Columnwise MAD  
\*\*\*\*\*

Dense Row Vector (ncol=3)

R	1	2	3
	7.4130111	22.609684	6.6717100

```
opt = cons(3,1,.);
opt[1] = 1; /* ipri */
opt[2] = 1; /* rowwise */
```

```
mad3 = mad(aa,opt);
print "MAD3=",mad3;
```

Unscaled Rowwise MAD  
\*\*\*\*\*

Dense Column Vector (nrow=12)

C	1	2	3	4	5
	2.800000	13.500000	1.500000	3.500000	2.500000
C	6	7	8	9	10
	10.500000	1.500000	2.500000	1.000000	6.500000
C	11	12			
	5.000000	11.000000			



## 4.8 Function median

---

```
v = median(a<,optn>)
```

**Purpose:** This function returns median

- as a scalar for an input vector **a** or
- for  $m \times n$  input matrix **a** either a  $n$  vector for columnwise (default) or a  $m$  vector for rowwise (see `optn[2]`) treatment.

For the median function see also `univar()` function.

**Input:** **a** should be a real or int matrix or vector.

`optn` is a vector of options which should be initialized to missing for the default options.

1. amount of printed output (default=0, no output)
2. =0: colwise treatment, =1: rowwise treatment (default=0)

**Output:** The result is either a real scalar or a vector depending on the input **a**.

**Restrictions:** The data **a** may have missing values.

**Relationships:** `univar()`, `quantile()`, `mad()`

**Examples:** `print "Heart data, D.M. Hawkins (1994)";`

```
a= [ 1 42.8 40.0 37,  
     2 63.5 93.5 50,  
     3 37.5 35.5 34,  
     4 39.5 30.0 36,  
     5 45.5 52.0 43,  
     6 38.5 17.0 28,  
     7 43.0 38.5 37,  
     8 22.5  8.5 20,  
     9 37.0 33.0 34,  
    10 23.5  9.5 30,  
    11 33.0 21.0 38,  
    12 58.0 79.0 47 ];  
aa = a[,2:4];
```

```
sopt= [ "ari" "med" "mad" ];  
c1 = univar(aa,sopt);  
print "\n Compare Measures with MAD and MEDIAN\n",c1;
```

Compare Measures with MAD and MEDIAN

	Var_1	Var_2	Var_3
Ari_Mean	40.358	38.125	36.167
Median	39.000	34.250	36.500
MAD	5.0000	15.250	4.5000

```
print "***** MEDIAN *****";
opt = cons(3,1,.);
opt[1] = 1; /* ipri */
med1 = median(aa,opt);
print "MED1=",med1;
```

Columnwise Median  
\*\*\*\*\*

Dense Row Vector (ncol=3)

R	1	2	3
	39.000000	34.250000	36.500000

```
opt = cons(3,1,.);
opt[1] = 1; /* ipri */
opt[2] = 1; /* rowwise */
med3 = median(aa,opt);
print "MED3=",med3;
```

Rowwise Median  
\*\*\*\*\*

Dense Column Vector (nrow=12)

C	1	2	3	4	5
	40.000000	63.500000	35.500000	36.000000	45.500000
C	6	7	8	9	10
	28.000000	38.500000	20.000000	34.000000	23.500000
C	11	12			
	33.000000	58.000000			



Total Effect = 54.4

Median Polished Data

\*\*\*\*\*

Dense Matrix (3 by 7)

```
      |      month1      month2      month3      month4      month5
-----|-----
place1 |  3.4000000  3.3000000  1.7000000  0.0000000 -0.6000000
place2 |  0.0000000  0.0000000  0.0000000  0.0000000  1.5000000
place3 | -11.6000000 -7.2000000 -4.2000000  0.0000000  0.0000000

      |      month6      month7
-----|-----
place1 | -1.3000000  0.0000000
place2 |  4.2000000  5.8000000
place3 |  0.0000000 -0.7000000
```

Row Effects

\*\*\*\*\*

Dense Column Vector (nrow=3)

```
  C |      place1      place2      place3
     | -14.9000000  0.0000000  28.9000000
```

Column Effects

\*\*\*\*\*

Dense Row Vector (ncol=7)

```
  R |      month1      month2      month3      month4      month5
     | -14.0000000 -13.5000000 -7.9000000  0.0000000  10.2000000

  R |      month6      month7
     | 20.4000000  25.7000000
```

2. More General: More iterations until convergence:

```
print "More General: More iterations to convergence";
opt = cons(4,1,.);
opt[1] = 1;          /* ipri */
opt[3] = 100;       /* maxit */
```

```

opt[4] = .0001;          /* eps */
< bb,reff,ceff > = mpolish(tukey,opt);
print "Median Polish=",bb;
print "RowEffects=",reff;
print "ColEffects=",ceff;

```

Median polish algorithm converged after 2 iterations.  
Total Effect = 54.4

Median Polished Data  
\*\*\*\*\*

Dense Matrix (3 by 7)

	month1	month2	month3	month4	month5
place1	3.4000000	3.3000000	1.7000000	0.0000000	-1.3000000
place2	0.0000000	0.0000000	0.0000000	0.0000000	0.8000000
place3	-10.9000000	-6.5000000	-3.5000000	0.7000000	0.0000000

  

	month6	month7
place1	-2.0000000	0.0000000
place2	3.5000000	5.8000000
place3	0.0000000	0.0000000

Row Effects  
\*\*\*\*\*

Dense Column Vector (nrow=3)

C	place1	place2	place3
	-14.9000000	0.0000000	28.2000000

Column Effects  
\*\*\*\*\*

Dense Row Vector (ncol=7)

R	month1	month2	month3	month4	month5
	-14.0000000	-13.5000000	-7.9000000	0.0000000	10.9000000

  

R	month6	month7

21.100000 25.700000

3. Mean Polish needs only one iteration:

```
print "Mean Polish needs only one iteration";
opt = cons(4,1,.);
opt[1] = 1;          /* ipri */
opt[2] = 1;          /* mean polish */
opt[3] = 2;          /* maxit */
opt[4] = .0001;     /* eps */
< cc,reff,ceff > = mpolish(tukey,opt);
print "Mean Polish=",cc;
print "RowEffects=",reff;
print "ColEffects=",ceff;
```

Total Effect = 61.781

Mean Polished Data  
\*\*\*\*\*

Dense Matrix (3 by 7)

	month1	month2	month3	month4	month5
place1	4.9333333	3.4000000	1.3333333	-1.2000000	-2.1000000
place2	0.8190476	-0.6142857	-1.0809524	-1.9142857	-0.7142857
place3	-5.7523810	-2.7857143	-0.2523810	3.1142857	2.8142857

  

	month6	month7
place1	-3.4666667	-2.9000000
place2	1.3190476	2.1857143
place3	2.1476190	0.7142857

Row Effects  
\*\*\*\*\*

Dense Column Vector (nrow=3)

C	place1	place2	place3
	-18.366667	-2.7523810	21.119048

Column Effects

\*\*\*\*\*

Dense Row Vector (ncol=7)

```
R |      month1      month2      month3      month4      month5
   | -19.447619 -17.514286 -11.447619 -2.7142857  7.7857143
R |      month6      month7
   | 18.652381  24.685714
```

## 4.10 Function nlfitt

```
gof = nlfitt(train,modl<,optn<,class<,fun1<,fun2<, actf<,link<,test> .. >)
```

```
<gof,parm,fit,tabs,stat,scor,tscor> = nlfitt(train,modl<,optn<,class<,...> .. >)
```

**Purpose:** The `nlfitt` function performs stagewise nonlinear regression for predictive modeling data mining  $y = F(x)$  of large data sets. For some special applications this function is similar to *PROC DMNEURL* in the SAS Enterprise Miner product.

Lets assume the training data set  $\mathbf{X}$  has  $N$  observations (rows). The  $n$  predictor effects (variables) can be selected via a `model` string from the columns of  $\mathbf{X}$ .

In each stage two linear or nonlinear functions are applied, an activation function  $f(z)$  and a link function  $g(f(z))$  to either the original response  $y$  (in its first stage) or its residuals  $r = \hat{y} - y$  in all subsequent stages.

Until now, the response  $y$  (in data mining terms: "target") must be either binary (discrete) or interval scaled (continuous).

1. For the first stage, the link function can be user specified. The default link function for the first stage would be logistic for binary data and identity for interval scaled data.
2. Since the following stages work on residuals, an identity link function is always used, i.e.  $g()$  is not really effective anymore.

A set of  $m$  nonlinear functions can be specified for the activation function. In each stage each of the activation functions is tried on the data and the best fitting functions are selected for computing expected values and residuals at that stage.

This is a function designed for nonlinear data mining of very large data sets and therefore has three levels of complexity depending on the size of the input data set:

1. The nonlinear fit can be applied directly to the original  $N \times n$  data set. Normally that would assume that all data are stored incore and the number  $n$  of predictors is small. Each model scoring needs an entire run through the  $N \times n$  original data set, and since there are usually many optimizations involved that could be a costly but precise approach to model fit. This approach can be specified by setting the `maxvec` option to zero.
2. The nonlinear fit can be applied to a small number of best fitting principal components, i.e. a subset of the the data sets eigenvectors with good prediction. Normally, that would assume that the  $N \times p$ , where  $p \ll n$ , principal component scores can be stored incore. Each model scoring needs a run through the  $N \times p$  principal component



scores which are hopefully stored incore. This approach can be specified by setting the `maxvec` option to greater than zero and the `npoint` option to zero.

3. If in addition to a larger number  $n$  of predictors (effects), the number of observations  $N$  is large, the principal component scores can be bucketed into a  $p$  dimensional frequency table by discretization of the score values. This approach is selected by setting both, the `maxvec` and the `npoint` options to values greater than zero.

Of course, fitting principal component scores or even categorized principal component scores of the predictors has an impact on the goodness of model fit.

For interval target, also the response ( $y$ ) can be bucketed in quantiles (by default percentiles) for showing observed-predicted accuracy tables. However, the  $y$  bucketing for interval response has no influence on the model fit except when the `minmis` option is specified for optimizing the accuracy rate or the `selcr` option is specified to `acc` for selecting the function with best accuracy rate in each stage.

The following model parametrizations, ordered with increasing complexity, are covered by the `nlfit` function. If there is principal component bucketing specified, the effects  $x_j$  are based on a set of principal components depending on stage  $k$  and  $nc$  is substituted for  $n$ :

**single** has  $n + 1$  parameters  $e$  to fit in each stage  $k$

$$y = g_k(f_k(e_0 + \sum_{j=1}^n e_j x_j^k))$$

all activation functions  $f(z)$  are applied but only the best is selected.

**separate** has  $2n + 1$  parameters  $e$  to fit in each stage

$$y = g_k(e_0 + \sum_{j=1}^n e_j f_k(e_{n+j} x_j^k))$$

The activation functions  $f$  are the same for each model effect. Each specified activation function  $f(z)$  is applied but only the best is selected.

**stepwise** has  $2n + 1$  parameters  $e$  to fit in each stage and is similar to *separate*. Starting from estimates  $e$  of the *separate* model in stage  $k$ , stepwise each model effect is tried with all other specified activation functions:

$$y = g_k(e_0 + \sum_{j=1}^n e_j f_k^b(e_{n+j} x_j^k))$$

Here the  $b$  in  $f_k^b$  indicates that the best activation function is chosen from a specified set of functions.

**multiple** has  $2n + 1$  parameters  $e$  to fit in each stage and has the same parametrization as *stepwise*. It also starts with the estimates of the *separate* solution. However, different from *stepwise*, each of the  $m$  functions is tried for each of the  $n$  effects before the function with the best fit is exchanged.

The *single* and *separate* models need only a constant number of  $m$  optimizations in each stage. The *stepwise* model needs  $m*(n+1)$  optimizations in each stage, whereas the *multiple* model needs a multiple of  $m * (n + 1)$  optimizations in each stage. The results of the *stepwise* approach may depend on the order of the effects in the model, whereas the order of effects should have no influence at the results of the *multiple* approach. If there is only one activation function specified, the optimal estimates for the *stepwise* and *multiple* models are the same as for the *separate*.

Currently, the following activation functions can be specified:

1	"lin"	<b>linear</b>	$s$	NA
2	"squ"	<b>square</b>	$s^2$	$(a + b * x) * x$
3	"tan"	<b>tanh</b>	$\tanh(s)$	$a * \tanh(b * x)$
4	"arc"	<b>arctan</b>	$\text{atan}(s)$	$a * \text{atan}(b * x)$
5	"log"	<b>logist</b>	$\exp(s)/(1. + \exp(s))$	$\exp(a * x)/(1. + \exp(b * x))$
6	"gau"	<b>gauss</b>	$\exp(-s^2)$	$a * \exp(-(b * x)^2)$
7	"sin"	<b>sin</b>	$\sin(s)$	$a * \sin(b * x)$
8	"cos"	<b>cos</b>	$\cos(s)$	$a * \cos(b * x)$
9	"exp"	<b>exp</b>	$\exp(s)$	$a * \exp(b * x)$

Column 3 of this table shows the formulas for the single model with  $s = e_0 + \sum_{j=1}^n e_j x_j$  and column 4 the formulas for each effect  $x_j$  of the *separate*, *stepwise*, and *multiple* models. Note, that for the *separate*, *stepwise*, and the *multiple* model the square function includes the parametrization of the linear function.

The following link functions can be specified for use in the first stage:

"ide"	<b>identity</b>	$x$
"log"	<b>logist</b>	$\exp(x)/(1. + \exp(x))$
"rec"	<b>reciprocal</b>	$1/x$

**Input: train** This is an  $N \times nc$  matrix containing  $N$  numerical observations of  $nc$  variables. From this matrix one column is selected for the observed response  $y$  and some other columns are selected for  $n$  predictors specified by the `mod1` argument.

**model** : 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.

- class** specifies which of the columns of the input data matrix  $\mathbf{X}$  are nominally scaled CLASS variables. (This argument can be missing value.)
- fun1** A set of  $m$  activation functions  $f$  can be specified as shown in the table above. If **fun1** is specified to missing, all activation functions will be applied and the best are selected in each stage.
- fun2** Only one link function  $g$  can be specified for the first stage as shown in the table above. If **fun2** is specified to missing, the **log** function will be used for binary response and the identity is used for interval scaled response.
- actf** specifies the name of a user defined activation function. This function must be defined by the user before the call of **nlfit** and should have the syntax:
- link** specifies the name of a user defined link function. This function must be defined by the user before the call of **nlfit** and should have the syntax:
- test** This is an  $N_t \times nc$  matrix containing  $N_t$  numerical observations of  $nc$  variables. Note, that the column should correspond to those of **train** since the model specification applies to both in the same way. This data set is not used for the model, it is only used for scoring the predicted values.

Option	Second Column	Meaning
"cut"	real	cutoff value, default=0.5
"fcrit"		use $F$ criterion (and not $R^2$ )
"freq"	int	column number of frequency variable
"inipnt"		
"maxcom"	int	only valid for principal component bucketing: maximum number of selected components in [2,8]
"mincom"	int	only valid for principal component bucketing: minimum number of selected components in [2,8]
"maxvec"	int	only valid for principal component bucketing maximum number of computed eigenvectors $\geq 2$
"maxst"	int	maximum number of stages, $\geq 1$ , default=10
"memsiz"	int	
"minmis"		minimize misclassification
"model"	string	kind of nonlinear fit model
	"sing"	single parametrization of model
	"sepa"	separate parametrization of model
	"mult"	
"npoint"	int	only valid for principal component bucketing: number of $X$ buckets, must be in [5,19]
"nbest"	int	
"nobstat"		do not print observational info (residuals etc.)
"nopr"		do not print
"pini"		print initial values
"popt"	int	print optimization histories
"ppar"		print parameter estimates
"pres"		print residuals
"print"	int	amount of printed output
"ptab"	int	print accuracy tables
"pvec"		print all "maxvec" eigenvectors
"selcr"	string	type of criterion for selecting best fit
	"sse"	use SSE criterion, is default
	"acc"	use classification accuracy
"seed"	int	seed for random generator
"sing"	real	threshold for singularity, default=1.e-8
"stopr2"	real	determines number of selected components, default=5.e-5
"vers"	string	type of algorithm for PCA see below
"ypct"	int	only valid for bucketing interval response number of $Y$ buckets (percentiles), default=10

Both, interval predictors ( $X$ ) and interval response ( $Y$ ), can be discretized into buckets using the "npoint" and "ypct" options. The default for the "npoint" option depends on the number  $nc_0$  of selected components in the first stage:

$$\text{npoint} = \begin{cases} 17 & \text{for } nc_0 \leq 3 \\ 15 & \text{for } nc_0 = 4, 5 \\ 13 & \text{for } nc_0 = 6 \\ 11 & \text{for } nc_0 = 7 \\ 9 & \text{otherwise} \end{cases}$$

The "maxst" option specifies an upper bound for the number of stages of estimation. If "maxst" is not specified, the default is 10. When a missing value is specified, the multistage estimation process is terminated

- if the sum-of-squares residual in the component selection process changes by less than 1%
- or when an upper range of 100 stages are processed.

That means, not specifying "maxst" or specifying a missing value are treated differently. Large values for "maxst" may result in numerical problems: the discretization error may be too large and the fit criterion does no longer improve and can actually become worse. In such a case the stagewise process is terminated with the last good stage.

Principal component bucketing of the predictors is only default for more than 10 predictors (model effects). This version of the algorithm reduces the number of runs through the data set and is usually recommended for  $n > 10$ . The options "maxvec", "maxcom", "mincom", and "npnt" specify the algorithmic details for the bucketing of principal components:

The "maxvec" option specifies an upper bound for the number of eigenvectors made available for selection. The default is:

$$\text{"maxvec"} = \begin{cases} 0 & \text{for } n < 10 \\ \text{MIN}(n, 40) & \text{for } n \geq 10 \end{cases}$$

No principal component bucketing is performed if "maxvec" is set to zero. Smaller values than  $\text{MIN}(n, 40)$  should be used only if there are memory problems for storing the eigenvectors when too many variables are included in the analysis. The specified value for "maxvec" cannot be smaller than that for "mincom".

The "maxcom" option specifies an upper bound for the number of components selected for predicting the target in each stage. Good values for "maxcom" are inbetween 3 and 5. Note, that the computer time and core memory will increase superlinear for larger values than 5. There is one memory allocation which takes  $n^m$  long integer values, where  $n$  is the value specified with the "npoint" option and  $m$  is the value specified by the "maxcom" option. The following table lists values of  $4n^m/1000000$  for specific combinations of  $(n, m)$ . This is the actual memory requirement in mb assuming that a long integer takes 4 bytes storage:

n	m=3	m=4	m=5	m=6	m=7	m=8
5	0	0	0	0	0	2
7	0	0	0	0	3	23
9	0	0	0	2	19	172*
11	0	0	1	7	78*	857
13	0	0	2	19*	250	3263
15	0	0*	3*	46	683	10252
17	0*	0	6	97	1641	27903
19	0	1	10	188	3575	67934

The trailing asterisk indicates the default number of points for a given number of components. Therefore, values larger than 8 are reduced to this upper range. It seems to be better to increase the value  $i$  of the `MAXSTAGE=i` option when higher precision is requested.

The `"mincom"` option specifies a lower bound for the number of components selected for predicting the target in each stage. The default is 2. The specified value for `"mincom"` cannot be larger than that for `"maxcom"`. The `"mincom"` specification may permit the selection of components which otherwise would be rejected by the `"stopr2"` option. The `nlfit` function may override the specified value when the rank of the  $X'X$  matrix is less than the specified value.

The `"stopr2"` option specifies a lower value for the incremental model  $R^2$  value at which the variable selection process is stopped. The `"stopr2"` criterion is used only for the  $R^2$  values of the components selected in the range specified by the `"mincom"` and `"maxcom"` values. The default is  $5e - 5$ .

This is from the `pca` document: Using the `version` option we can select an algorithm among the eight available implementations. The first four compute eigenvalue decompositions of symmetric cross product (covariance or correlation) matrices (either  $X^T X$  or  $XX^T$  whichever is the smaller one) and the other four methods use the singular value decomposition of the raw matrix  $X$ :

- "cevd1"** (11) all values, all vectors dense eigen value decomposition (EVD): the algorithm is based on the eigenvalue decomposition of a dense symmetric  $X^T X$  or  $XX^T$  matrix and can be applied to raw input data and also to  $n \times n$  input covariance or correlation matrices;
- "cevd2"** (12) all values, few vectors dense EVD: the algorithm is based on the eigenvalue decomposition of a dense symmetric  $X^T X$  or  $XX^T$  matrix and can be applied to raw input data and also to  $n \times n$  input covariance or correlation matrices;
- "cevd3"** (13) selected triplets Lapack EVD: the algorithm is based on the eigenvalue decomposition of a dense symmetric  $X^T X$  or  $XX^T$  matrix and can be applied to raw input data and also to  $n \times n$  input covariance or correlation matrices;

”**cev4**” (14) selected triplets Arpack EVD: the algorithm is appropriate for large and sparse data matrix  $\mathbf{X}$  and is based on the eigenvalue decomposition of a dense symmetric  $X^T X$  or  $XX^T$  matrix and can be applied to raw input data and also to  $n \times n$  input covariance or correlation matrices;

”**rsv1**” (16) standard dense singular value decomposition (SVD): the algorithm is suitable for medium sized dense data matrices  $\mathbf{X}$ , and cannot be applied to  $n \times n$  input covariance or correlation matrices;

”**rsv2**” (17) selected triplets Arpack SVD: the algorithm is suitable for large sized sparse data matrices  $\mathbf{X}$  and a relatively small number of factors, and cannot be applied to  $n \times n$  input covariance or correlation matrices;

”**rsv3**” (18) selected triplets Block Lanczos SVD: the algorithm is suitable for large sized sparse data matrices  $\mathbf{X}$ , and cannot be applied to  $n \times n$  input covariance or correlation matrices;

”**rsv4**” (19) selected triplets subspace iteration SVD: the algorithm is suitable for large sized sparse data matrices  $\mathbf{X}$ , and cannot be applied to  $n \times n$  input covariance or correlation matrices.

The eigenvalue algorithms need memory in  $O(n^2) + O(n * m)$  or  $O(N^2) + O(N * m) + O(n * m)$  whereas the singular value algorithms need memory  $O(N * m) + O(n * m)$  for dense  $\mathbf{X}$  and  $O(nzer) + O(n * m)$  for sparse  $\mathbf{X}$ .

In addition most of the options for the **nlp** functions can be specified.

**Output:** **gof** this is vector of some scalar results, like

**parm** contains the optimal parameters for all stages.

**fit** contains a table of fit indices for all stages.

**tabs** contains an accuracy table for the final solution.

**stat** contains means and standard deviations of the original data and results of the PCA, all or a subset of eigenvalues and eigenvectors.

**scor** This is an  $N \times 4$  matrix containing the  $N$  predicted model values and residuals obtained by scoring the training data using the optimal model weights from the training data of each stage.

**tscor**  $N_t \times 4$  matrix containing  $N_t$  predicted model values and residuals obtained by scoring the test data using the optimal model weights from the training data of each stage.

**Restrictions:** 1.

**Relationships:** `nlfprd()`

**Examples:** 1. Interval Response  $y$ : Predicting Reactions of Chemical Process

```

print "Predicting Reactions of Chemical Process: nrow=20 > ncol=15";
print "SAS PLSEG1: time1-time5 temp1-temp5 pres1-pres5 yield1-yield5";

options NOECHO;
#include "..\\tdata\\chem_pls.dat"
options ECHO;
print "Data", process;

```

(a) Separate Model, using raw data (maxvec=0):

All three models, the multiple, the stepwise and the separate model obtain perfect fit with the first stage of fitting. Therefore, we only show the output of the separate model. Neither X nor Y bucketing is specified. The data are fit directly without using principal components. That means that there are no accuracy tables available.

```

print "Separate Activation Model: maxvec=0 ypct=0";
modl = "16 = 1:15";
optn = [ "print"          3 ,
        "ptab"           1 ,
        "pres"           ,
        "popt"           0 ,
        "ypct"           0 ,
        "maxvec"         0 ,
        "maxstag"        6 ];
< gof,parm,fit,tabs,stat,scor > = nlfitt(process,modl,optn);

```

```

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

```

```

Number Valid Observations  20
Response Variable          Y[16]
N Independent Variables    15
NOBS w/o Missing Target   20
Interval Target            yield1
Target Minimum:           0.0000e+000
Target Maximum:           6.2900e+001
* Separate Activation Model *
First Link Function        IDENTITY
Selection Criterion        SSE
Optimize                   SSE
Max. Estimation Stages    6
Number Y Percentiles      0
Max. Number Components    0
No Princ. Component Reduction

```



Store Input Data Incore  
 Store Eigenvectors Incore

\*\*\*\*\*  
 Model Effects  
 \*\*\*\*\*

X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 +  
 X12 + X13 + X14 + X15

\*\*\*\*\*  
 Simple Statistics  
 \*\*\*\*\*

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[16]	20	36.900000	19.611302	0.0579980	-1.2749133
X[ 1]	20	4.7050000	2.3164117	-0.3026435	-0.7653976
X[ 2]	20	3.9950000	2.2670349	0.1657809	-1.1747032
X[ 3]	20	4.6000000	2.8121915	0.2509782	-1.3687120
X[ 4]	20	5.8000000	3.0378316	0.0012891	-1.2392063
X[ 5]	20	3.2000000	1.5471536	-0.2310823	-0.7400643
X[ 6]	20	16.100000	12.311313	0.6794396	-0.6680792
X[ 7]	20	27.000000	19.633484	0.6818937	-0.5312877
X[ 8]	20	36.050000	18.138285	-0.2982325	-0.8111074
X[ 9]	20	28.950000	18.613733	0.1194999	-1.0417223
X[10]	20	24.950000	18.591099	0.8148850	-0.2585900
X[11]	20	0.2605000	0.1689433	0.3648949	-1.1293015
X[12]	20	0.4320000	0.2227011	-0.0435753	-1.1859672
X[13]	20	0.3100000	0.2310161	0.5728630	-0.9656921
X[14]	20	0.1400000	0.0915941	0.0867574	-1.2117938
X[15]	20	0.2505000	0.1784576	0.5069831	-1.0127073

Only one stage is needed to fit the data to full precision:

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0  
 \*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	2.54e-023	4.75e-005	3.24e-004	0.0016440	0.0380617	1.93e-004
Iter	1	11	4	5	10	4
Gmax	8.03e-014	2.90e-004	3.90e-004	1.08e-004	4.12e-004	3.52e-004

	COS	EXP
Crit	5.29e-011	2.37e-004

Iter 9 6  
 Gmax 4.03e-006 1.90e-004

\*\*\*\*\*  
 Activation Ordered by SSE Criterion  
 \*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE
1	SQUARE	4.013e-018	2.003e-009	2.536e-023
7	COS	8.371e-006	0.00289321	5.289e-011
2	TANH	7.51726506	2.74176313	4.750e-005
6	SIN	30.5528832	5.52746626	1.931e-004
8	EXP	37.4557626	6.12011132	2.367e-004
3	ARCTAN	51.2075901	7.15594788	3.236e-004
4	LOGIST	260.170058	16.1297879	0.00164398
5	GAUSS	6023.51281	77.6112931	0.03806173

SSE of Best Solution= 4.01304e-018 at Stage 0

\*\*\*\*\*  
 Summary Table Across Stages  
 \*\*\*\*\*

Stage	Activ.	Link	SSE	RMSE	Accur.
0	SQUARE	IDENTITY	4.013e-018	2.003e-009	.

Stage	Activ.	Link	AIC	SBC
0	SQUARE	IDENTITY	-799.054288	-768.186588

Time for Optimization: 1  
 Total Processing Time: 1  
 Number of Optimizations : 9  
 Number of Runs through Data : 304

Expected Values and Residuals of Training Data  
 \*\*\*\*\*

Dense Matrix (20 by 4)

		Stage	Observed	Predicted	Residual
1		0.0000000	37.900000	37.900000	-6.50e-011
2		0.0000000	62.900000	62.900000	5.18e-010
3		0.0000000	17.400000	17.400000	2.69e-010
4		0.0000000	25.900000	25.900000	-3.73e-010
5		0.0000000	62.600000	62.600000	1.72e-010

```

6 | 0.0000000 18.700000 18.700000 1.60e-010
7 | 0.0000000 59.200000 59.200000 6.04e-011
8 | 0.0000000 26.600000 26.600000 3.56e-010
9 | 0.0000000 55.100000 55.100000 -1.56e-010
10 | 0.0000000 41.200000 41.200000 -1.01e-010
11 | 0.0000000 60.700000 60.700000 -7.16e-010
12 | 0.0000000 61.700000 61.700000 4.69e-010
13 | 0.0000000 38.400000 38.400000 9.48e-010
14 | 0.0000000 16.800000 16.800000 -2.67e-010
15 | 0.0000000 59.300000 59.300000 -3.94e-010
16 | 0.0000000 0.0000000 -5.01e-011 5.01e-011
17 | 0.0000000 21.900000 21.900000 2.15e-010
18 | 0.0000000 23.300000 23.300000 -1.16e-009
19 | 0.0000000 24.800000 24.800000 -1.73e-010
20 | 0.0000000 23.600000 23.600000 1.87e-010

```

(b) Single Model, using raw data (maxvec=0):

Neither X nor Y bucketing is specified. The data are fit directly without using principal components. Naturally, the single activation model does not fit as well as the separate, stepwise, or multiple models do:

```

print "Single Activation Model: maxvec=0 ypct=0";
modl = "16 = 1:15";
optn = [ "print"      3 ,
        "ptab"      1 ,
        "pres"      ,
        "popt"      0 ,
        "ypct"      0 ,
        "model"     "sing" ,
        "maxvec"    0 ,
        "maxstag"   6 ];
< gof,parm,fit,tabs,stat,scor > = nlfite(process,modl,optn);

```

We only report stages 0 and 5 here:

```

*****
Summary of All Optimizations in Stage=0
*****

```

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	0.0019483	6.30e-004	3.59e-004	2.82e-004	0.0013487	2.86e-004
Iter	8	4	3	2	6	3
Gmax	3.54e-004	1.49e-004	3.26e-004	1.52e-004	3.15e-004	1.00e-004

COS      EXP

Crit 0.0041252 0.0011081  
 Iter 5 3  
 Gmax 2.42e-004 1.03e-004

\*\*\*\*\*  
 Activation Ordered by SSE Criterion  
 \*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE
4	LOGIST	44.5537827	3.33743100	2.815e-004
6	SIN	45.2489991	3.36336881	2.859e-004
3	ARCTAN	56.7871451	3.76786230	3.588e-004
2	TANH	99.7200842	4.99299720	6.301e-004
8	EXP	175.366436	6.62129964	0.00110812
5	GAUSS	213.439687	7.30478759	0.00134870
1	SQUARE	308.324923	8.77959171	0.00194826
7	COS	652.832242	12.7752910	0.00412516

SSE of Best Solution= 44.5538 at Stage 0

And after skipping stages 1,2,3,4 here is stage 5:

\*\*\*\*\*  
 Summary of All Optimizations in Stage=5  
 \*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	3.33e-005	3.31e-005	3.31e-005	1.44e-004	6.83e-005	3.31e-005
Iter	6	5	4	3	4	3
Gmax	3.50e-004	3.97e-008	2.94e-008	2.19e-004	3.60e-004	1.47e-006

	COS	EXP
Crit	0.0145885	2.45e-004
Iter	4	4
Gmax	3.30e-005	4.24e-004

\*\*\*\*\*  
 Activation Ordered by SSE Criterion (Stage 5)  
 \*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE
3	ARCTAN	5.24137740	2.28940547	3.312e-005
2	TANH	5.24137740	2.28940547	3.312e-005
6	SIN	5.24138055	2.28940616	3.312e-005
1	SQUARE	5.26348922	2.29422955	3.326e-005
5	GAUSS	10.8050541	3.28710420	6.828e-005
4	LOGIST	22.8230775	4.77735047	1.442e-004

8 EXP 38.7623713 6.22594341 2.449e-004  
 7 COS 2308.72724 48.0492168 0.01458852  
 SSE of Best Solution= 5.24138 at Stage 5

\*\*\*\*\*  
 Summary Table Across Stages  
 \*\*\*\*\*

Stage	Activ.	Link	SSE	RMSE	Accur.
0	LOGIST	IDENTITY	44.5537827	3.33743100	.
1	SQUARE	IDENTITY	25.0040796	5.00040794	.
2	TANH	IDENTITY	12.5770568	3.54641464	.
3	SQUARE	IDENTITY	9.44258749	3.07287935	.
4	TANH	IDENTITY	5.24137746	2.28940548	.
5	ARCTAN	IDENTITY	5.24137740	2.28940547	.

Stage	Activ.	Link	AIC	SBC
0	LOGIST	IDENTITY	48.0192957	63.9510121
1	SQUARE	IDENTITY	68.4661344	100.329567
2	TANH	IDENTITY	86.7228398	134.517989
3	SQUARE	IDENTITY	112.989955	176.716821
4	TANH	IDENTITY	133.217041	212.875623
5	ARCTAN	IDENTITY	165.217041	260.807339

Time for Optimization: 0  
 Total Processing Time: 0  
 Number of Optimizations : 54  
 Number of Runs through Data : 1531

Expected Values and Residuals of Training Data

\*\*\*\*\*

Dense Matrix (20 by 4)

	Stage	Observed	Predicted	Residual
1	5.0000000	37.900000	38.698525	-0.7985255
2	5.0000000	62.900000	62.500177	0.3998225
3	5.0000000	17.400000	16.625611	0.7743889
4	5.0000000	25.900000	25.543547	0.3564535
5	5.0000000	62.600000	62.141869	0.4581309
6	5.0000000	18.700000	18.720500	-0.0205003
7	5.0000000	59.200000	59.495363	-0.2953628
8	5.0000000	26.600000	26.439281	0.1607192
9	5.0000000	55.100000	55.135300	-0.0353000

10		5.0000000	41.200000	41.066577	0.1334229
11		5.0000000	60.700000	61.355670	-0.6556697
12		5.0000000	61.700000	61.122133	0.5778668
13		5.0000000	38.400000	37.914270	0.4857302
14		5.0000000	16.800000	16.883578	-0.0835776
15		5.0000000	59.300000	59.935070	-0.6350696
16		5.0000000	0.0000000	0.8054136	-0.8054136
17		5.0000000	21.900000	21.179015	0.7209847
18		5.0000000	23.300000	23.294299	0.0057009
19		5.0000000	24.800000	24.662266	0.1377335
20		5.0000000	23.600000	24.481555	-0.8815549

(c) Multiple Model, using principal components (npoint=0):

We only show the output of the multiple model. The results of the stepwise and separate models are of similar quality, however, the result of the single model is significantly worse. Neither X nor Y bucketing is specified. That means that there are no accuracy tables available.

```

modl = "16 = 1:15";
optn = [ "print"      3 ,
        "ptab"       1 ,
        "pres"       ,
        "popt"        0 ,
        "ypct"        0 ,
        "model"      "mult" ,
        "maxvec"      10 ,
        "npoint"      0 ,
        "maxstag"     6 ,
        "maxcomp"     3 ];
< gof,parm,fit,tabs,stat,scor > = nlfitt(process,modl,optn);

```

```

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

```

```

Number Valid Observations  20
Response Variable          Y[16]
N Independent Variables     15
NOBS w/o Missing Target    20
Interval Target            yield1
Target Minimum:            0.0000e+000
Target Maximum:            6.2900e+001
* Multiple Activation Model *
PCA Common All Evals and Evecs
First Link Function        IDENTITY

```

```

Selection Criterion      SSE
Optimize                SSE
Max. Estimation Stages 6
Number Y Percentiles   0
Max. Number Components 3
Max. N. Eigenvectors   10
Min. Number Components 1
Minimum R2 Value        5e-005
No Princ. Component Bucketing
Store Input Data Incore
Store Eigenvectors Incore

```

Some of the output is skipped, we report here only the output of the first and last stage:

```

*****
Component Selection: SS(y) and R2 (SS_total=8.73005)
*****

```

Comp	Eigval	R-Square	F Value	p-Value	SSE
1	240.257463	0.20432395	4.87906486	0.0340	6.94629228
2	27.0982152	0.00416054	0.09461556	0.8283	6.90997057
3	17.5908661	0.00108341	0.02330104	0.9462	6.90051239

```

*****
Summary of All Optimizations in Stage=0
*****

```

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	4.12e-004	4.10e-004	4.12e-004	4.33e-004	0.0460731	4.21e-004
Iter	1	18	19	14	6	6
Gmax	7.41e-014	4.32e-006	2.75e-004	2.48e-004	3.58e-004	1.78e-004

	COS	EXP
Crit	0.0423173	4.16e-004
Iter	12	55
Gmax	3.10e-004	1.25e-004

```

*****
Activation Ordered by SSE Criterion
*****

```

Run	Activ.	SSE	RMSE	MSE
2	TANH	64.8976538	2.23430688	4.101e-004
1	SQUARE	65.1232697	2.23818728	4.115e-004
3	ARCTAN	65.2609999	2.24055281	4.124e-004

8 EXP            65.7875760   2.24957391   4.157e-004  
 6 SIN            66.6398675   2.26409886   4.211e-004  
 4 LOGIST        68.5852374   2.29690825   4.334e-004  
 7 COS            6696.98842   22.6969812   0.04231733  
 5 GAUSS         7291.36912   23.6827901   0.04607314  
 SSE of Best Solution= 64.8977 at Stage 0  
 Change Function TANH to SQUARE for Component 1

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 1  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	4.06e-004	4.10e-004	4.31e-004	4.21e-004	0.0382883
	Iter	7	0	4	15	14
	Gmax	1.35e-004	4.32e-006	2.12e-004	3.82e-004	4.42e-004
CMP2	Crit	4.15e-004	4.10e-004	4.16e-004	4.19e-004	0.0012089
	Iter	4	0	3	3	10
	Gmax	4.56e-004	4.32e-006	2.61e-004	4.67e-004	3.24e-004
CMP3	Crit	4.11e-004	4.10e-004	4.10e-004	4.10e-004	6.31e-004
	Iter	1	0	1	3	3
	Gmax	2.58e-004	4.32e-006	1.67e-004	1.62e-004	3.80e-004
		SIN	COS	EXP		
CMP1	Crit	4.33e-004	0.0378923	4.33e-004		
	Iter	5	9	19		
	Gmax	2.91e-004	4.07e-004	1.05e-004		
CMP2	Crit	4.13e-004	0.0012063	4.18e-004		
	Iter	5	6	7		
	Gmax	1.15e-004	4.75e-005	2.34e-005		
CMP3	Crit	4.11e-004	6.31e-004	4.11e-004		
	Iter	4	4	5		
	Gmax	7.15e-005	3.87e-004	3.17e-004		

Change Function TANH to SIN for Component 2

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 2  
 \*\*\*\*\*

SQUARE            TANH            ARCTAN            LOGIST            GAUSS



CMP1	Crit	4.06e-004	4.11e-004	4.13e-004	4.15e-004	0.0382590
	Iter	0	16	7	17	19
	Gmax	1.35e-004	2.85e-004	4.51e-005	1.56e-004	3.47e-004
CMP2	Crit	4.12e-004	4.06e-004	4.09e-004	4.06e-004	0.0011283
	Iter	6	0	12	7	8
	Gmax	1.50e-004	1.35e-004	2.24e-004	3.85e-004	1.67e-004
CMP3	Crit	4.09e-004	4.06e-004	4.07e-004	4.06e-004	6.16e-004
	Iter	2	0	6	7	4
	Gmax	1.43e-004	1.35e-004	7.79e-005	1.26e-004	4.39e-004

		SIN	COS	EXP
CMP1	Crit	4.35e-004	0.0378929	6.57e-004
	Iter	5	17	23
	Gmax	3.32e-004	4.44e-004	4.94e-004
CMP2	Crit	4.02e-004	0.0011253	4.22e-004
	Iter	18	10	5
	Gmax	4.28e-004	4.01e-004	3.18e-004
CMP3	Crit	4.03e-004	6.16e-004	4.09e-004
	Iter	10	3	4
	Gmax	2.34e-005	3.38e-004	2.45e-004

Change Function TANH to SIN for Component 3

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 3  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	4.02e-004	4.51e-004	4.07e-004	5.31e-004	0.0382616
	Iter	0	6	12	10	18
	Gmax	4.28e-004	3.74e-004	4.07e-004	3.96e-004	4.51e-004
CMP2	Crit	4.12e-004	4.06e-004	4.09e-004	4.06e-004	0.0011271
	Iter	10	6	8	7	13
	Gmax	3.00e-004	3.08e-004	7.62e-005	3.24e-004	4.74e-004
CMP3	Crit	4.05e-004	4.02e-004	4.03e-004	4.03e-004	6.11e-004
	Iter	1	0	6	4	5
	Gmax	3.55e-004	4.28e-004	2.49e-004	2.89e-004	2.52e-004

	SIN	COS	EXP
--	-----	-----	-----

CMP1	Crit	4.32e-004	0.0378767	6.26e-004
	Iter	5	7	14
	Gmax	3.43e-004	2.56e-004	4.37e-004
CMP2	Crit	4.02e-004	0.0011257	4.18e-004
	Iter	0	12	6
	Gmax	4.28e-004	2.14e-004	6.90e-005
CMP3	Crit	3.99e-004	6.11e-004	4.05e-004
	Iter	7	2	3
	Gmax	1.02e-005	4.00e-004	6.95e-006

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 4  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	3.99e-004	4.05e-004	4.05e-004	4.54e-004	0.0365131
	Iter	0	5	11	12	14
	Gmax	1.02e-005	6.11e-008	1.48e-004	1.90e-004	4.79e-004
CMP2	Crit	4.12e-004	4.03e-004	4.06e-004	4.05e-004	0.0011268
	Iter	7	8	6	4	8
	Gmax	9.27e-005	2.56e-004	3.16e-004	4.20e-004	5.90e-005
CMP3	Crit	4.05e-004	4.04e-004	4.04e-004	4.03e-004	6.11e-004
	Iter	2	6	5	4	4
	Gmax	5.14e-006	3.94e-004	4.34e-004	3.61e-004	3.85e-006
		SIN	COS	EXP		
CMP1	Crit	4.22e-004	0.0362236	4.89e-004		
	Iter	5	11	5		
	Gmax	2.30e-004	2.27e-004	2.62e-004		
CMP2	Crit	3.99e-004	0.0011263	4.17e-004		
	Iter	0	5	7		
	Gmax	1.02e-005	2.59e-004	4.67e-005		
CMP3	Crit	3.99e-004	6.11e-004	4.05e-004		
	Iter	0	3	3		
	Gmax	1.02e-005	3.98e-004	6.81e-006		

SSE of Best Solution= 64.8977 at Stage 0

\*\*\*\*\*

Stage 0 : Criterion SSE= 0.000398685

\*\*\*\*\*

Comp	Activation
1	SQUARE
2	SIN
3	SIN

And here comes the output for stage 5:

\*\*\*\*\*

Component Selection: SS(y) and R2 (Stage=5)

\*\*\*\*\*

Comp	Eigval	R-Square	F Value	p-Value	SSE
1	240.257463	0.02115417	0.41061544	0.7918	0.00112312
6	0.00897240	0.00580324	0.10735222	0.7953	0.00111646
4	0.01268220	0.00478525	0.08401616	0.8005	0.00111097

\*\*\*\*\*

Summary of All Optimizations in Stage=5

\*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	2.26e-005	2.83e-005	2.82e-005	2.78e-005	2.28e-005	2.81e-005
Iter	1	5	6	2	5	6
Gmax	2.40e-014	1.97e-004	3.67e-004	7.72e-005	3.71e-004	4.20e-005

	COS	EXP
Crit	2.28e-005	2.80e-005
Iter	8	4
Gmax	2.88e-004	2.59e-004

\*\*\*\*\*

Activation Ordered by SSE Criterion (Stage 5)

\*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE
1	SQUARE	3.57796291	1.89155040	2.261e-005
7	COS	3.60706383	1.89922717	2.279e-005
5	GAUSS	3.61509185	1.90133949	2.284e-005
4	LOGIST	4.39907204	2.09739649	2.780e-005
8	EXP	4.42508371	2.10358829	2.796e-005
6	SIN	4.44731954	2.10886689	2.810e-005
3	ARCTAN	4.45653901	2.11105164	2.816e-005
2	TANH	4.47689078	2.11586644	2.829e-005

SSE of Best Solution= 3.57796 at Stage 5

\*\*\*\*\*  
Summary of All Optimizations in Stage=5 in Iteration 1  
\*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	2.26e-005	2.59e-005	2.61e-005	2.75e-005	2.78e-005
	Iter	0	3	4	3	3
	Gmax	2.40e-014	6.31e-005	3.15e-004	1.55e-004	5.80e-005
CMP2	Crit	2.26e-005	2.35e-005	2.35e-005	2.34e-005	3.03e-005
	Iter	0	2	2	3	2
	Gmax	2.40e-014	9.04e-005	9.04e-005	5.02e-005	1.69e-006
CMP3	Crit	2.26e-005	2.30e-005	2.30e-005	2.29e-005	2.26e-005
	Iter	0	2	2	3	4
	Gmax	2.40e-014	3.88e-004	3.88e-004	1.83e-004	1.82e-004
		SIN	COS	EXP		
CMP1	Crit	2.68e-005	2.79e-005	2.70e-005		
	Iter	1	4	4		
	Gmax	4.08e-004	1.53e-004	4.16e-004		
CMP2	Crit	2.35e-005	2.28e-005	2.34e-005		
	Iter	2	7	10		
	Gmax	1.87e-004	4.13e-004	2.70e-004		
CMP3	Crit	2.26e-005	2.26e-005	2.44e-005		
	Iter	3	5	3		
	Gmax	1.52e-004	2.74e-004	3.07e-004		

SSE of Best Solution= 3.57796 at Stage 5

\*\*\*\*\*  
Stage 5 : Criterion SSE= 2.26086e-005  
\*\*\*\*\*

Comp	Activation
1	SQUARE
2	SQUARE
3	SQUARE

\*\*\*\*\*  
Summary Table Across Stages

\*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	64.8976538	2.23430688	.	37.5415840	44.5117099
1	29.6012151	2.22115642	.	35.8416627	49.7819145
2	22.1516413	4.70655301	.	44.0435864	64.9539641
3	13.0233475	3.60878754	.	47.4202287	75.3007324
4	4.53954590	2.13062101	.	40.3418942	75.1925238
5	3.57796291	1.89155040	.	49.5812269	91.4019824

Stage	Link	CMP1	CMP2	CMP3
0	IDENTITY	SQUARE	SIN	SIN
1	IDENTITY	SQUARE	SQUARE	SQUARE
2	IDENTITY	SQUARE	GAUSS	SQUARE
3	IDENTITY	LOGIST	SQUARE	GAUSS
4	IDENTITY	SQUARE	SQUARE	SQUARE
5	IDENTITY	SQUARE	SQUARE	SQUARE

Time for Optimization: 1  
Total Processing Time: 1  
Number of Optimizations : 372  
Number of Runs through Data : 12140

(d) Multiple Model, using principal components Bucketing:

We only show the output of the multiple model. Default bucketing of principal component scores for three components is being used:

```
print "Multiple Activation Model: maxvec=10 ypct=0";  
modl = "16 = 1:15";  
optn = [ "print"      3 ,  
         "ptab"      1 ,  
         "pres"      ,  
         "popt"      0 ,  
         "ypct"      0 ,  
         "model"     "mult" ,  
         "maxvec"    10 ,  
         "maxstag"   6 ,  
         "maxcomp"   3 ];  
< gof,parm,fit,tabs,stat,scor > = nlfit(process,modl,optn);
```

\*\*\*\*\*  
Model Information  
\*\*\*\*\*

Number Valid Observations 20

```

Response Variable      Y[16]
N Independent Variables  15
NOBS w/o Missing Target  20
Interval Target      yield1
Target Minimum:      0.0000e+000
Target Maximum:      6.2900e+001
* Multiple Activation Model *
PCA Common All Evals and Evecs
First Link Function  IDENTITY
Selection Criterion  SSE
Optimize            SSE
Max. Estimation Stages  6
Number Y Percentiles  0
Max. Number Components  3
Max. N. Eigenvectors  10
Min. Number Components  1
Minimum R2 Value      5e-005
Store Input Data Incore
Store Eigenvectors Incore

```

Again, we only show some output of the first and last stage:

```

*****
Component Selection: SS(y) and R2 (SS_total=8.73005)
*****

```

Comp	Eigval	R-Square	F Value	p-Value	SSE
1	240.257463	0.20432395	4.87906486	0.0340	6.94629228
2	27.0982152	0.00416054	0.09461556	0.8283	6.90997057
3	17.5908661	0.00108341	0.02330104	0.9462	6.90051239

Number of X Grid Points (Buckets): 17  
Distinctive Patterns=20  
Sparsity Percentage=0.407083

```

*****
Summary of All Optimizations in Stage=0
*****

```

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	5.76e-004	5.99e-004	6.23e-004	5.40e-004	0.0458084	6.25e-004
Iter	1	5	4	17	9	6
Gmax	9.30e-014	1.94e-004	2.22e-004	4.02e-004	4.77e-004	3.87e-005

	COS	EXP
Crit	0.0457532	5.72e-004

Iter 10 200  
 Gmax 3.95e-004 0.0044020

\*\*\*\*\*  
 Activation Ordered by Approximate Fit Criterion (Stage 0)  
 \*\*\*\*\*

Run	Activation	Criterion	ASSE	Accuracy
4	LOGIST	5.399e-004	85.4501733	.
8	EXP	5.725e-004	90.5961080	.
1	SQUARE	5.761e-004	91.1784088	.
2	TANH	5.988e-004	94.7620178	.
3	ARCTAN	6.229e-004	98.5856877	.
6	SIN	6.253e-004	98.9635592	.
7	COS	0.04575321	7240.73814	.
5	GAUSS	0.04580838	7249.46880	.

\*\*\*\*\*  
 Activation Ordered by SSE Criterion  
 \*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE
2	TANH	76.8632792	2.43157561	4.857e-004
4	LOGIST	77.7983525	2.44632145	4.916e-004
3	ARCTAN	78.4286668	2.45621138	4.956e-004
6	SIN	78.8109809	2.46219072	4.980e-004
1	SQUARE	80.0313278	2.48118036	5.057e-004
8	EXP	81.3006266	2.50077875	5.137e-004
5	GAUSS	7292.75289	23.6850373	0.04608188
7	COS	7302.66535	23.7011284	0.04614452

SSE of Best Solution= 76.8633 at Stage 0  
 Change Function TANH to SIN for Component 1

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 1  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	5.76e-004	5.99e-004	5.63e-004	6.10e-004	0.0401760
	Iter	1	0	6	16	12
	Gmax	5.22e-014	1.94e-004	4.90e-004	3.96e-004	4.41e-004
CMP2	Crit	5.76e-004	5.99e-004	5.53e-004	5.67e-004	0.0450015
	Iter	1	0	8	6	200
	Gmax	4.20e-014	1.94e-004	2.21e-004	1.98e-004	0.0175463

CMP3	Crit	5.76e-004	5.99e-004	5.57e-004	5.68e-004	0.0460505
	Iter	1	0	4	6	176
	Gmax	1.53e-014	1.94e-004	9.95e-005	1.42e-004	4.81e-004

		SIN	COS	EXP
CMP1	Crit	6.11e-004	0.0434539	5.72e-004
	Iter	7	11	200
	Gmax	1.12e-004	3.70e-004	0.0162759

CMP2	Crit	6.24e-004	0.0375703	6.06e-004
	Iter	6	7	200
	Gmax	2.02e-004	3.88e-004	0.0920057

CMP3	Crit	6.18e-004	0.0412250	6.06e-004
	Iter	3	200	61
	Gmax	3.82e-004	0.0026512	2.27e-004

Change Function TANH to SIN for Component 3

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 2  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	5.76e-004	5.39e-004	5.44e-004	0.0086240	0.0400618
	Iter	1	6	10	4	16
	Gmax	6.04e-014	4.98e-004	1.87e-004	7.02e-015	2.23e-004

CMP2	Crit	5.76e-004	5.38e-004	5.54e-004	6.29e-004	0.0420818
	Iter	1	9	12	4	8
	Gmax	6.06e-014	2.69e-004	4.08e-004	1.78e-004	1.19e-004

CMP3	Crit	5.76e-004	5.38e-004	5.53e-004	5.39e-004	0.0400913
	Iter	1	9	4	8	7
	Gmax	1.90e-014	2.69e-004	2.14e-004	1.13e-004	2.80e-004

		SIN	COS	EXP
CMP1	Crit	6.11e-004	0.0405229	5.72e-004
	Iter	0	17	200
	Gmax	1.12e-004	4.60e-004	0.0051141

CMP2	Crit	6.11e-004	0.0373768	5.72e-004
	Iter	5	13	200
	Gmax	1.90e-004	1.82e-004	0.0676009



CMP3 Crit 6.14e-004 0.0440514 6.04e-004  
 Iter 2 19 61  
 Gmax 4.42e-004 4.85e-004 2.89e-004

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0 in Iteration 3  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	5.76e-004	5.38e-004	5.47e-004	5.60e-004	0.0389485
	Iter	1	11	11	16	27
	Gmax	1.23e-013	8.03e-005	2.72e-004	4.10e-005	2.83e-004

CMP2	Crit	5.76e-004	5.38e-004	5.96e-004	5.38e-004	0.0420129
	Iter	1	6	7	8	6
	Gmax	4.11e-014	2.00e-004	4.14e-004	2.61e-005	2.17e-004

CMP3	Crit	5.76e-004	5.38e-004	5.50e-004	5.38e-004	0.0435022
	Iter	1	5	4	8	8
	Gmax	3.79e-014	1.75e-004	4.67e-004	1.78e-004	3.10e-004

		SIN	COS	EXP
CMP1	Crit	6.14e-004	0.0351958	5.72e-004
	Iter	0	23	200
	Gmax	4.42e-004	2.73e-004	0.0557026

CMP2	Crit	6.11e-004	0.0373844	6.06e-004
	Iter	5	12	200
	Gmax	1.87e-005	2.91e-004	0.1120499

CMP3	Crit	6.14e-004	0.0208049	5.72e-004
	Iter	0	9	200
	Gmax	4.42e-004	1.63e-004	0.0145815

SSE of Best Solution= 76.8633 at Stage 0

\*\*\*\*\*  
 Stage 0 : Criterion SSE= 0.000458508  
 \*\*\*\*\*

Comp	Activation
1	SIN
2	TANH

The output at stage 5 follows:

```
*****
Component Selection: SS(y) and R2 (Stage=5)
*****

Comp      Eigval      R-Square      F Value p-Value      SSE
4  0.01268220  0.00890252  0.17066729  0.9825  0.00584529
6  0.00897240  0.00852444  0.15616133  0.9829  0.00579502
3  17.5908661  0.00684632  0.11928284  0.9834  0.00575464
      Distinctive Patterns=17
      Sparsity Percentage=0.346021

*****
Summary of All Optimizations in Stage=5
*****

      SQUARE      TANH      ARCTAN      LOGIST      GAUSS      SIN
Crit 1.44e-004 1.45e-004 1.45e-004 1.46e-004 1.47e-004 1.47e-004
Iter      1      2      2      2      2      3
Gmax 9.33e-015 2.58e-004 3.53e-005 2.16e-005 3.66e-004 9.14e-005

      COS      EXP
Crit 1.46e-004 1.45e-004
Iter      5      1
Gmax 4.06e-004 1.71e-004
```

```
*****
Activation Ordered by Approximate Fit Criterion (Stage 5)
*****
```

Run	Activation	Criterion	ASSE	Accuracy
1	SQUARE	1.445e-004	22.8629642	.
8	EXP	1.449e-004	22.9384869	.
2	TANH	1.451e-004	22.9680589	.
3	ARCTAN	1.453e-004	23.0019995	.
7	COS	1.460e-004	23.1083155	.
4	LOGIST	1.461e-004	23.1201595	.
6	SIN	1.467e-004	23.2176601	.
5	GAUSS	1.471e-004	23.2735464	.

```
*****
Activation Ordered by SSE Criterion (Stage 5)
*****
```

Run	Activ.	SSE	RMSE	MSE
8	EXP	23.1895118	4.81554896	1.465e-004
5	GAUSS	23.2191005	4.81862018	1.467e-004
2	TANH	23.3069705	4.82772933	1.473e-004
3	ARCTAN	23.3116816	4.82821723	1.473e-004
4	LOGIST	23.3979350	4.83714120	1.478e-004
6	SIN	23.4876412	4.84640498	1.484e-004
7	COS	23.9227882	4.89109275	1.512e-004
1	SQUARE	6193.11500	78.6963468	0.03913343

SSE of Best Solution= 23.1895 at Stage 5

\*\*\*\*\*  
Summary of All Optimizations in Stage=5 in Iteration 1  
\*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	1.44e-004	1.46e-004	1.47e-004	1.45e-004	1.46e-004
	Iter	1	6	5	3	3
	Gmax	1.76e-015	3.06e-004	4.70e-004	1.53e-004	4.75e-004
CMP2	Crit	1.44e-004	1.47e-004	1.45e-004	1.45e-004	1.46e-004
	Iter	1	3	3	2	6
	Gmax	1.81e-015	1.79e-004	2.15e-004	1.13e-004	9.59e-005
CMP3	Crit	1.44e-004	1.45e-004	1.45e-004	1.48e-004	1.46e-004
	Iter	1	2	2	2	7
	Gmax	8.51e-015	5.14e-005	3.70e-005	5.99e-005	1.77e-004
CMP1		SIN	COS	EXP		
	Crit	1.52e-004	1.25e-004	1.45e-004		
	Iter	3	4	0		
CMP2	Crit	1.46e-004	1.46e-004	1.45e-004		
	Iter	3	6	0		
	Gmax	2.22e-004	2.40e-004	1.71e-004		
CMP3	Crit	1.47e-004	1.46e-004	1.45e-004		
	Iter	3	6	0		
	Gmax	8.28e-005	1.30e-004	1.71e-004		

SSE of Best Solution= 23.1895 at Stage 5

\*\*\*\*\*

Stage 5 : Criterion SSE= 0.000146531

\*\*\*\*\*

Comp Activation

1 EXP

2 EXP

3 EXP

\*\*\*\*\*

Summary Table Across Stages

\*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	76.8632792	2.43157561	.	40.9259195	47.8960454
1	31.7937683	2.30194730	.	37.2707606	51.2110125
2	28.3867422	5.32792100	.	49.0037988	69.9141765
3	24.2218786	4.92157278	.	59.8304805	87.7109842
4	23.3341130	4.83053961	.	73.0836818	107.934311
5	23.1895118	4.81554896	.	86.9593565	128.780112

Stage	Link	CMP1	CMP2	CMP3
0	IDENTITY	SIN	TANH	SIN
1	IDENTITY	SQUARE	SQUARE	SQUARE
2	IDENTITY	SIN	SIN	SIN
3	IDENTITY	SQUARE	SQUARE	SQUARE
4	IDENTITY	SIN	SIN	SIN
5	IDENTITY	EXP	EXP	EXP

Time for Optimization: 5

Total Processing Time: 7

Number of Optimizations : 240

Number of Runs through Data : 57

Expected Values and Residuals of Training Data

\*\*\*\*\*

Dense Matrix (20 by 4)

	Stage	Observed	Predicted	Residual
1	5.0000000	37.900000	37.676557	0.2234431
2	5.0000000	62.900000	62.400752	0.4992483
3	5.0000000	17.400000	15.190759	2.2092406
4	5.0000000	25.900000	26.413678	-0.5136775
5	5.0000000	62.600000	62.139951	0.4600486

```

6 | 5.000000 18.700000 18.672629 0.0273710
7 | 5.000000 59.200000 59.983398 -0.7833977
8 | 5.000000 26.600000 28.187412 -1.5874124
9 | 5.000000 55.100000 55.140799 -0.0407992
10 | 5.000000 41.200000 41.215449 -0.0154488
11 | 5.000000 60.700000 61.183270 -0.4832703
12 | 5.000000 61.700000 59.842839 1.8571605
13 | 5.000000 38.400000 38.349388 0.0506116
14 | 5.000000 16.800000 16.548941 0.2510588
15 | 5.000000 59.300000 59.444958 -0.1449582
16 | 5.000000 0.000000 1.0845181 -1.0845181
17 | 5.000000 21.900000 20.664132 1.2358678
18 | 5.000000 23.300000 25.670269 -2.3702686
19 | 5.000000 24.800000 23.293784 1.5062164
20 | 5.000000 23.600000 23.404475 0.1955251

```

## 2. Binary Response $y$ : Cancer Remission Data

The data set has only 20 rows (observations), the response (target) in column 1 and 6 predictor variables corresponding to columns 2 to 7:

```

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 ,
          .....
          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;

```

(a) Multiple Model, using raw data (maxvec=0):

```

print "Multiple Activation Model: SELCR=SSE, MAXVEC=0";
clas = 1;
modl = "1 = 2:7";
optn = [ "print" 3 ,

```

```

"ptab"          2 ,
"pini"          ,
"pres"          ,
"popt"          0 ,
"model"        "mult" ,
"selcr"        "sse" ,
"maxvec"       0 ,
"maxstag"      6 ];
< gof,parm,fit,tabs,stat,scor,tscor > =
      nlfit(remis,modl,optn,clas,,,,,temis);

```

```

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

```

```

Number Valid Observations  27
Response Variable          Y[1]
N Independent Variables     6
NOBS w/o Missing Target    27
Binary Target              remiss
* Multiple Activation Model *
First Link Function        LOGIST
Selection Criterion        SSE
Optimize                   SSE
Max. Estimation Stages    6
Max. Number Components    0
No Princ. Component Reduction
Store Input Data Incore
Store Eigenvectors Incore

```

```

*****
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

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

Variable	Value	Nobs	Proportion
Y[1]	0	9	33.333333
	1	18	66.666667

The following shows the results of the first stage:

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0  
 \*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	0.0377872	0.0290892	0.0185281	0.0389415	0.0317960	4.28e-006
Iter	3	16	38	65	95	28
Gmax	1.72e-004	3.72e-004	2.82e-004	4.27e-004	4.70e-004	9.54e-005

	COS	EXP
Crit	0.0186852	.
Iter	19	61
Gmax	4.91e-004	0.0000000

\*\*\*\*\*  
 Classification Table for CUTOFF = 0.5000  
 \*\*\*\*\*

Activ.	Acc.	Observed	Predicted	
			remiss01	remiss02
SIN	100.0000	remiss01	9.0000000	0.0000000
		remiss02	0.0000000	18.000000
ARCTAN	96.2963	remiss01	8.0000000	1.0000000
		remiss02	0.0000000	18.000000

```

COS      96.2963 remiss01 9.000000 0.000000
          remiss02 1.000000 17.000000
TANH     96.2963 remiss01 8.000000 1.000000
          remiss02 0.000000 18.000000
GAUSS    88.8889 remiss01 8.000000 1.000000
          remiss02 2.000000 16.000000
SQUARE   92.5926 remiss01 8.000000 1.000000
          remiss02 1.000000 17.000000
LOGIST   88.8889 remiss01 8.000000 1.000000
          remiss02 2.000000 16.000000

```

```

*****
Activation Ordered by SSE Criterion
*****

```

```

Run  Activ.      SSE          RMSE          MSE  Accur.
 6 SIN          2.313e-004  0.00406433  4.283e-006 100.0000
 3 ARCTAN       1.00051485  0.26733003  0.01852805 96.2963
 7 COS          1.00900263  0.26846157  0.01868523 96.2963
 2 TANH         1.57081506  0.33496429  0.02908917 96.2963
 5 GAUSS        1.71698434  0.35020242  0.03179601 88.8889
 1 SQUARE       2.04050611  0.38177276  0.03778715 92.5926
 4 LOGIST       2.10283905  0.38756005  0.03894146 88.8889
 8 EXP          .          .          .          .
SSE of Best Solution= 0.000231263 at Stage 0
Change Function SIN to ARCTAN for Variable 6

```

```

*****
Summary of All Optimizations in Stage=0 in Iteration 1
*****

```

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
VAR1	Crit	0.0370381	0.0185194	0.0185268	0.0185214	0.0370374
	Iter	20	9	14	16	6
	Gmax	4.36e-004	1.60e-005	3.66e-004	8.65e-005	1.83e-004
VAR2	Crit	0.0925905	0.0740761	0.0740758	0.0740743	0.1296296
	Iter	4	5	6	8	3
	Gmax	1.91e-004	4.19e-004	3.08e-004	7.91e-005	1.67e-008
VAR3	Crit	0.1296249	0.1296250	0.1296250	0.1296248	0.1296245
	Iter	0	0	0	0	0
	Gmax	2.13e-004	2.10e-004	2.10e-004	2.19e-004	2.30e-004
VAR4	Crit	0.1111100	0.1296305	0.1296308	0.1296299	0.1296305



	Iter	6	2	2	4	3
	Gmax	6.02e-005	1.77e-004	3.67e-004	1.40e-004	1.72e-004
VAR5	Crit	0.0555617	0.0370376	0.0370378	0.0925934	0.0740742
	Iter	10	8	16	2	4
	Gmax	4.49e-004	8.10e-005	1.85e-004	4.50e-004	4.65e-006
VAR6	Crit	0.0370375	0.0187311	1.36e-006	0.0740743	0.0925926
	Iter	10	9	10	5	1
	Gmax	1.96e-004	2.75e-004	3.87e-004	8.21e-005	1.74e-007

		SIN	COS	EXP
VAR1	Crit	4.28e-006	0.0185205	.
	Iter	0	10	1
	Gmax	9.54e-005	4.31e-004	8.37e-051
VAR2	Crit	4.28e-006	0.0740754	.
	Iter	0	9	1
	Gmax	9.54e-005	9.88e-005	1.60e-009
VAR3	Crit	4.28e-006	0.1296220	0.1296245
	Iter	0	0	0
	Gmax	9.54e-005	3.43e-004	2.30e-004
VAR4	Crit	4.28e-006	0.1296289	0.1481481
	Iter	0	2	1
	Gmax	9.54e-005	9.05e-005	1.30e-036
VAR5	Crit	4.28e-006	0.0925764	.
	Iter	0	7	1
	Gmax	9.54e-005	3.63e-004	0.0000000
VAR6	Crit	4.28e-006	0.0925926	.
	Iter	0	1	1
	Gmax	9.54e-005	1.98e-013	0.0000000

\*\*\*\*\*  
Summary of All Optimizations in Stage=0 in Iteration 2  
\*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
VAR1	Crit	0.0555561	0.0555571	0.0555571	0.0740755	0.0740743
	Iter	7	4	4	2	3
	Gmax	8.47e-005	2.43e-004	2.46e-004	2.45e-004	2.98e-004

VAR2	Crit	0.1111111	0.0925932	0.1481481	0.1481482	0.1481481
	Iter	1	9	1	5	1
	Gmax	2.97e-110	6.88e-005	1.51e-011	1.69e-004	0.0000000
VAR3	Crit	0.0925959	0.1111112	0.0740745	0.0925926	0.0740749
	Iter	12	1	8	9	12
	Gmax	2.92e-004	4.99e-005	9.48e-005	6.52e-006	4.25e-004
VAR4	Crit	0.1296308	0.1481497	0.1296296	0.1296304	0.1296305
	Iter	6	3	2	6	5
	Gmax	3.21e-004	4.53e-004	1.31e-008	7.72e-005	9.36e-005
VAR5	Crit	0.0925924	0.0555582	0.0740752	0.0555569	0.0555564
	Iter	2	5	1	5	4
	Gmax	5.41e-006	1.07e-004	1.78e-004	3.46e-004	1.66e-004
VAR6	Crit	0.0925933	0.0925933	1.36e-006	0.0925932	0.0925932
	Iter	0	0	0	0	0
	Gmax	1.56e-004	1.62e-004	3.87e-004	1.51e-004	1.37e-004

		SIN	COS	EXP
VAR1	Crit	1.36e-006	0.0555641	.
	Iter	0	11	1
	Gmax	3.87e-004	3.45e-004	0.0000000
VAR2	Crit	1.36e-006	0.1296299	.
	Iter	0	5	1
	Gmax	3.87e-004	2.05e-004	0.0000000
VAR3	Crit	1.36e-006	0.0925926	.
	Iter	0	1	1
	Gmax	3.87e-004	4.55e-014	0.0000000
VAR4	Crit	1.36e-006	0.1111116	0.1111115
	Iter	0	11	5
	Gmax	3.87e-004	1.16e-004	8.57e-005
VAR5	Crit	1.36e-006	0.0555600	.
	Iter	0	8	1
	Gmax	3.87e-004	2.26e-004	0.0000000
VAR6	Crit	0.0925933	0.0925929	0.0925932
	Iter	0	0	0

Gmax 1.64e-004 6.19e-005 1.37e-004

SSE of Best Solution= 7.36372e-005 at Stage 0

\*\*\*\*\*  
Stage 0 : Criterion SSE= 0.000231263  
\*\*\*\*\*

N	Variable	Activation
1	cell	SIN
2	smear	SIN
3	infil	SIN
4	li	SIN
5	blast	SIN
6	temp	ARCTAN

Following the result output of stage 2:

\*\*\*\*\*  
Stage 2 : Criterion SSE= 2.13712e-005  
\*\*\*\*\*

N	Variable	Activation
1	cell	SQUARE
2	smear	SQUARE
3	infil	SQUARE
4	li	SQUARE
5	blast	SQUARE
6	temp	SQUARE

Stage 3 cannot improve the fit of stage 2

[note] file tnlfit.inp, line 75: New SSE = 2.13712e-005 in stage 2  
is not improved compared to 2.13712e-005.

[warning] file tnlfit.inp, line 75: Stagewise estimation process  
terminated. The results of the last stage are ignored.

\*\*\*\*\*  
Summary Table Across Stages  
\*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	7.364e-005	0.00229343	100.0000	-319.929318	-303.083439
1	2.137e-005	0.00462290	100.0000	-327.331185	-293.639427

Stage	Link	VAR1	VAR2	VAR3	VAR4	VAR5
-------	------	------	------	------	------	------

```

0 LOGIST SIN SIN SIN SIN SIN
  ARCTAN
1 IDENTITY SQUARE SQUARE SQUARE SQUARE SQUARE
  SQUARE

```

```

Time for Optimization: 0
Total Processing Time: 0
Number of Optimizations : 222
Number of Runs through Data : 7248

```

```

Classification Table: Accuracy=100
*****

```

Dense Matrix (2 by 2)

```

      | remiss01 remiss02
-----|-----
remiss01 |      9      0
remiss02 |      0     18

```

```

Expected Values and Residuals of Training Data
*****

```

Dense Matrix (27 by 4)

```

      | Stage Observed Predicted Residual
-----|-----
1 | 1.000000 1.000000 0.9998054 1.95e-004
2 | 1.000000 1.000000 0.9986656 0.0013344
3 | 1.000000 0.000000 6.94e-004 -6.94e-004
4 | 1.000000 0.000000 0.0000000 0.0000000
5 | 1.000000 1.000000 1.0000000 0.0000000
6 | 1.000000 0.000000 0.0000000 0.0000000
7 | 1.000000 1.000000 0.9996613 3.39e-004
8 | 1.000000 0.000000 5.79e-004 -5.79e-004
9 | 1.000000 0.000000 5.49e-004 -5.49e-004
10 | 1.000000 0.000000 1.35e-004 -1.35e-004
11 | 1.000000 0.000000 0.0000000 0.0000000
12 | 1.000000 0.000000 0.0000000 0.0000000
13 | 1.000000 0.000000 0.0000000 0.0000000
14 | 1.000000 0.000000 0.0000000 0.0000000
15 | 1.000000 0.000000 5.26e-004 -5.26e-004
16 | 1.000000 1.000000 0.9995817 4.18e-004
17 | 1.000000 0.000000 0.0000000 0.0000000
18 | 1.000000 0.000000 6.11e-004 -6.11e-004

```

```

19 | 1.0000000 0.0000000 0.0031245 -0.0031245
20 | 1.0000000 1.0000000 0.9998968 1.03e-004
21 | 1.0000000 0.0000000 9.29e-005 -9.29e-005
22 | 1.0000000 0.0000000 4.22e-004 -4.22e-004
23 | 1.0000000 1.0000000 1.0000000 0.0000000
24 | 1.0000000 0.0000000 8.27e-004 -8.27e-004
25 | 1.0000000 1.0000000 1.0000000 0.0000000
26 | 1.0000000 1.0000000 0.9994903 5.10e-004
27 | 1.0000000 0.0000000 4.60e-005 -4.60e-005

```

(b) Multiple Model, with principal components reduction, but no bucketing (npoint=0):

```

print "Multiple Activation Model: SELCR=SSE, MAXVEC=5";
clas = 1;
modl = "1 = 2:7";
optn = [ "print"      3 ,
        "ptab"      2 ,
        "pini"      ,
        "pvec"      ,
        "pres"      ,
        "popt"      0 ,
        "model"    "mult" ,
        "selcr"    "sse" ,
        "maxvec"   5 ,
        "npoint"   0 ,
        "maxstag"  6 ,
        "maxcomp"  3 ];
< gof,parm,fit,tabs,stat,scor > = nlfitt(remis,modl,optn,clas);

```

```

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

```

```

Number Valid Observations  27
Response Variable          Y[1]
N Independent Variables     6
NOBS w/o Missing Target    27
Binary Target              remiss
* Multiple Activation Model *
PCA Common All Evals and Evecs
First Link Function        LOGIST
Selection Criterion         SSE
Optimize                    SSE
Max. Estimation Stages     6
Max. Number Components     3

```

```

Max. N. Eigenvectors      5
Min. Number Components    1
Minimum R2 Value          5e-005
No Princ. Component Bucketing
Store Input Data Incore
Store Eigenvectors Incore

```

We only report some of the output from stage 5:

```

*****
Component Selection: SS(y) and R2 (Stage=5)
*****

```

Comp	Eigval	R-Square	F Value	p-Value	SSE
1	80.1891249	0.01972939	0.52328836	0.7660	0.45869397
5	5.36548838	0.00346819	0.08876391	0.7682	0.45707111
2	29.1239970	0.00246242	0.06065449	0.9937	0.45591888

```

*****
Summary of All Optimizations in Stage=5
*****

```

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	0.0084314	0.0084165	0.0084404	0.0082174	0.0082270	0.0083533
Iter	1	5	7	8	5	8
Gmax	6.17e-015	1.70e-004	1.03e-004	4.39e-004	2.87e-004	8.48e-005

	COS	EXP
Crit	0.0086392	0.0084394
Iter	8	1
Gmax	1.41e-004	1.74e-004

```

*****
Classification Table for CUTOFF = 0.5000
*****

```

Activ.	Acc.	Observed	Predicted	
			remiss01	remiss02
LOGIST	96.2963	remiss01	8.0000000	1.0000000
		remiss02	0.0000000	18.0000000
GAUSS	96.2963	remiss01	8.0000000	1.0000000
		remiss02	0.0000000	18.0000000
SIN	96.2963	remiss01	8.0000000	1.0000000
		remiss02	0.0000000	18.0000000
TANH	96.2963	remiss01	8.0000000	1.0000000
		remiss02	0.0000000	18.0000000

```

SQUARE  96.2963 remiss01 8.0000000 1.0000000
          remiss02 0.0000000 18.0000000
ARCTAN  96.2963 remiss01 8.0000000 1.0000000
          remiss02 0.0000000 18.0000000
COS     96.2963 remiss01 8.0000000 1.0000000
          remiss02 0.0000000 18.0000000

```

```

*****
Activation Ordered by SSE Criterion (Stage 5)
*****

```

Run	Activ.	SSE	RMSE	MSE
4	LOGIST	0.44374017	0.66613825	0.00821741
5	GAUSS	0.44425931	0.66652780	0.00822702
6	SIN	0.45107784	0.67162329	0.00835329
2	TANH	0.45449298	0.67416095	0.00841654
1	SQUARE	0.45529808	0.67475779	0.00843145
3	ARCTAN	0.45578074	0.67511535	0.00844038
7	COS	0.46651618	0.68301990	0.00863919
8	EXP	.	.	.

```

Accuracy of Best Solution= 96.2963 at Stage 5
Change Function LOGIST to SIN for Component 1

```

```

*****
Summary of All Optimizations in Stage=5 in Iteration 1
*****

```

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	0.0081605	0.0081539	0.0081617	0.0082174	0.0083378
	Iter	4	4	4	0	6
	Gmax	3.78e-004	1.36e-004	4.60e-004	4.39e-004	3.19e-004
CMP2	Crit	0.0084976	0.0084175	0.0084176	0.0082174	0.0084194
	Iter	5	1	1	0	5
	Gmax	4.71e-004	4.18e-004	4.29e-004	4.39e-004	1.48e-004
CMP3	Crit	0.0081450	0.0081872	0.0081543	0.0082174	0.0081666
	Iter	4	4	5	0	5
	Gmax	2.66e-004	2.07e-004	8.11e-005	4.39e-004	1.77e-004
CMP1		SIN	COS	EXP		
	Crit	0.0079459	0.0083776	0.0081677		
	Iter	6	5	2		
	Gmax	2.44e-004	3.30e-004	2.46e-004		

CMP2 Crit 0.0084165 0.0085347 0.0084182  
 Iter 8 5 3  
 Gmax 4.61e-004 1.55e-004 4.35e-004

CMP3 Crit 0.0081485 0.0081844 0.0081634  
 Iter 7 6 5  
 Gmax 2.84e-005 6.81e-005 3.23e-004

Change Function LOGIST to SQUARE for Component 3

\*\*\*\*\*  
 Summary of All Optimizations in Stage=5 in Iteration 2  
 \*\*\*\*\*

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	0.0081372	0.0081551	0.0081555	0.0081490	0.0083183
	Iter	5	3	7	4	5
	Gmax	2.26e-004	4.64e-004	6.79e-005	3.05e-004	3.93e-004

CMP2	Crit	0.0080682	0.0079480	0.0080966	0.0079459	0.0080676
	Iter	4	8	1	0	5
	Gmax	6.87e-005	1.11e-004	4.58e-004	2.44e-004	2.11e-004

CMP3	Crit	0.0079413	0.0079456	0.0079484	0.0079459	0.0079585
	Iter	2	7	5	0	6
	Gmax	1.35e-005	6.53e-005	1.06e-004	2.44e-004	4.50e-005

		SIN	COS	EXP
CMP1	Crit	0.0079459	0.0083182	0.0081633
	Iter	0	4	4
	Gmax	2.44e-004	1.13e-004	4.68e-004

CMP2	Crit	0.0080977	0.0080682	0.0080961
	Iter	2	3	1
	Gmax	2.45e-004	1.72e-004	8.33e-005

CMP3	Crit	0.0079478	0.0079613	0.0079509
	Iter	4	3	3
	Gmax	6.21e-005	3.67e-004	2.04e-004

\*\*\*\*\*  
 Summary of All Optimizations in Stage=5 in Iteration 3  
 \*\*\*\*\*

SQUARE	TANH	ARCTAN	LOGIST	GAUSS
--------	------	--------	--------	-------



CMP1	Crit	0.0081313	0.0081448	0.0081535	0.0081452	0.0083184
	Iter	1	2	2	5	5
	Gmax	1.09e-013	9.23e-006	4.86e-005	9.56e-005	1.63e-004
CMP2	Crit	0.0080555	0.0080939	0.0080940	0.0079413	0.0081015
	Iter	1	3	3	0	4
	Gmax	9.32e-005	3.99e-005	3.92e-005	1.35e-005	2.32e-005
CMP3	Crit	0.0079413	0.0079456	0.0079481	0.0079536	0.0079626
	Iter	0	7	5	1	5
	Gmax	1.35e-005	6.81e-005	1.10e-004	2.43e-004	4.51e-004

		SIN	COS	EXP
CMP1	Crit	0.0079413	0.0083177	0.0081588
	Iter	0	7	4
	Gmax	1.35e-005	4.49e-006	8.75e-005
CMP2	Crit	0.0080953	0.0081015	0.0080971
	Iter	4	5	2
	Gmax	1.52e-004	2.09e-004	3.41e-004
CMP3	Crit	0.0079537	0.0079594	0.0079421
	Iter	3	5	3
	Gmax	3.92e-004	4.87e-004	1.02e-004

SSE of Best Solution= 0.428829 at Stage 5

\*\*\*\*\*  
Stage 5 : Criterion SSE= 0.44374  
\*\*\*\*\*

Comp	Activation
1	SIN
2	LOGIST
3	SQUARE

\*\*\*\*\*  
Summary Table Across Stages  
\*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	1.00012427	0.22362069	96.2963	-74.9842404	-65.9133824
1	0.78845590	0.24627313	96.2963	-67.4049229	-49.2632068
2	0.57849749	0.31050966	96.2963	-61.7652641	-34.5526900
3	0.52816143	0.72674716	96.2963	-50.2231344	-13.9397021

```

4 0.46792586 0.68405107 96.2963 -39.4926214 5.86166893
5 0.42882855 0.65485002 96.2963 -27.8484441 26.5767043

```

```

Stage   Link      CMP1      CMP2      CMP3
  0  LOGIST    SQUARE     SIN      SIN
  1  IDENTITY  SQUARE     SIN      LOGIST
  2  IDENTITY     SIN     GAUSS    GAUSS
  3  IDENTITY  SQUARE    SQUARE    LOGIST
  4  IDENTITY     EXP    SQUARE    SQUARE
  5  IDENTITY     SIN    LOGIST    SQUARE

```

```

Time for Optimization: 1
Total Processing Time: 1
Number of Optimizations : 444
Number of Runs through Data : 16646

```

Classification Table: Accuracy=96.2963

\*\*\*\*\*

Dense Matrix (2 by 2)

```

          | remiss01 remiss02
          -----
remiss01 |          8          1
remiss02 |          0         18

```

(c) Multiple Model, with principal components bucketing:

```

print "Multiple Activation Model: SELCR=SSE, MAXVEC=5";
clas = 1;
modl = "1 = 2:7";
optn = [ "print"      3 ,
        "ptab"       2 ,
        "pres"       ,
        "popt"       0 ,
        "model"     "mult" ,
        "selcr"     "sse" ,
        "maxvec"    5 ,
        "maxstag"   6 ,
        "maxcomp"   2 ];
< gof,parm,fit,tabs,stat,scor > = nlfite(remis,modl,optn,clas);

```

We again skip part of the output for space reasons:

\*\*\*\*\*

Model Information

\*\*\*\*\*

Number Valid Observations 27  
 Response Variable Y[1]  
 N Independent Variables 6  
 NOBS w/o Missing Target 27  
 Binary Target remiss  
 \* Multiple Activation Model \*  
 PCA Common All Evals and Evecs  
 First Link Function LOGIST  
 Selection Criterion SSE  
 Optimize SSE  
 Max. Estimation Stages 6  
 Max. Number Components 2  
 Max. N. Eigenvectors 5  
 Min. Number Components 1  
 Minimum R2 Value 5e-005  
 Store Input Data Incore  
 Store Eigenvectors Incore

\*\*\*\*\*  
 Component Selection: SS(y) and R2 (SS\_total=18)  
 \*\*\*\*\*

Comp	Eigval	R-Square	F Value	p-Value	SSE
1	80.1891249	0.05085237	1.39299883	0.2496	17.0846574
3	23.7758194	0.03338429	0.91137862	0.3549	16.4837403

Number of X Grid Points (Buckets): 17  
 Distinctive Patterns=25  
 Sparsity Percentage=4.32526

\*\*\*\*\*  
 Summary of All Optimizations in Stage=0  
 \*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	0.0723355	0.0671329	0.0674801	0.0694130	0.1046118	0.0801554
Iter	6	8	11	4	21	12
Gmax	1.45e-004	3.54e-004	2.54e-004	5.66e-005	1.26e-004	8.35e-005

	COS	EXP
Crit	0.1016970	0.0699034
Iter	6	7
Gmax	3.01e-004	3.56e-004

\*\*\*\*\*  
 Activation Ordered by Approximate Fit Criterion (Stage 0)  
 \*\*\*\*\*

Run	Activation	Criterion	ASSE	Accuracy
2	TANH	0.06713291	3.62517714	81.4815
3	ARCTAN	0.06748015	3.64392784	81.4815
4	LOGIST	0.06941303	3.74830361	81.4815
8	EXP	0.06990336	3.77478141	74.0741
1	SQUARE	0.07233554	3.90611904	74.0741
6	SIN	0.08015544	4.32839353	77.7778
7	COS	0.10169696	5.49163593	70.3704
5	GAUSS	0.10461184	5.64903958	66.6667

\*\*\*\*\*  
 Classification Table for CUTOFF = 0.5000  
 \*\*\*\*\*

Activ.	Acc.	Observed	Predicted	
			remiss01	remiss02
EXP	74.0741	remiss01	7.0000000	2.0000000
		remiss02	5.0000000	13.0000000
SQUARE	74.0741	remiss01	5.0000000	4.0000000
		remiss02	3.0000000	15.0000000
LOGIST	81.4815	remiss01	5.0000000	4.0000000
		remiss02	1.0000000	17.0000000
TANH	77.7778	remiss01	5.0000000	4.0000000
		remiss02	2.0000000	16.0000000
ARCTAN	77.7778	remiss01	5.0000000	4.0000000
		remiss02	2.0000000	16.0000000
SIN	81.4815	remiss01	5.0000000	4.0000000
		remiss02	1.0000000	17.0000000
COS	70.3704	remiss01	3.0000000	6.0000000
		remiss02	2.0000000	16.0000000
GAUSS	62.9630	remiss01	1.0000000	8.0000000
		remiss02	2.0000000	16.0000000

\*\*\*\*\*  
 Activation Ordered by SSE Criterion  
 \*\*\*\*\*

Run	Activ.	SSE	RMSE	MSE	AMSE	Accur.
8	EXP	3.70788814	0.41053669	0.06866460	0.06990336	74.0741
1	SQUARE	3.86308933	0.41904053	0.07153869	0.07233554	74.0741

```

4 LOGIST  4.24118164 0.43906831 0.07854040 0.06941303 81.4815
2 TANH    4.24816041 0.43942940 0.07866964 0.06713291 77.7778
3 ARCTAN  4.34019354 0.44416385 0.08037395 0.06748015 77.7778
6 SIN     4.44755240 0.44962370 0.08236208 0.08015544 81.4815
7 COS     5.60428780 0.50471809 0.10378311 0.10169696 70.3704
5 GAUSS   5.62453183 0.50562885 0.10415800 0.10461184 62.9630
Accuracy of Best Solution= 74.0741 at Stage 0

```

```

*****
Summary of All Optimizations in Stage=0 in Iteration 1
*****

```

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	0.0649673	0.0670419	0.0783064	0.0672019	0.1111111
	Iter	8	11	55	5	4
	Gmax	1.61e-004	3.23e-004	1.81e-004	4.94e-004	1.09e-006

CMP2	Crit	0.0835914	0.0691836	0.0675029	0.0692244	0.1110832
	Iter	3	5	10	4	4
	Gmax	2.46e-004	3.03e-005	4.99e-004	4.53e-004	2.10e-005

		SIN	COS	EXP
CMP1	Crit	0.0726330	0.1110835	0.0699034
	Iter	16	8	0
	Gmax	3.53e-004	6.26e-005	3.56e-004

CMP2	Crit	0.0801842	0.1016641	0.0699034
	Iter	11	9	0
	Gmax	2.61e-004	2.14e-004	3.56e-004

SSE of Best Solution= 3.70789 at Stage 0

```

*****
Stage 0 : Criterion SSE= 3.70789
*****

```

Comp	Activation
1	EXP
2	EXP

It follows the output from stage 5:

```

*****
Component Selection: SS(y) and R2 (Stage=5)
*****

```

Comp	Eigval	R-Square	F Value	p-Value	SSE
5	5.36548838	0.03516909	0.94772708	0.3393	2.22521296
3	23.7758194	0.00598830	0.15613360	0.9565	2.21140200

Distinctive Patterns=27  
Sparsity Percentage=4.67128

\*\*\*\*\*  
Summary of All Optimizations in Stage=5  
\*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	0.0408744	0.0411457	0.0411454	0.0412183	0.0426447	0.0411415
Iter	1	3	3	8	9	3
Gmax	3.22e-016	4.27e-005	2.74e-005	3.49e-004	3.56e-004	6.92e-005

	COS	EXP
Crit	0.0426108	0.0389814
Iter	8	200
Gmax	2.29e-004	2908733.4

\*\*\*\*\*  
Activation Ordered by Approximate Fit Criterion (Stage 5)  
\*\*\*\*\*

Run	Activation	Criterion	ASSE	Accuracy
8	EXP	0.03898138	2.10499437	96.2963
1	SQUARE	0.04087441	2.20721818	96.2963
6	SIN	0.04114150	2.22164080	96.2963
3	ARCTAN	0.04114542	2.22185250	96.2963
2	TANH	0.04114569	2.22186734	96.2963
4	LOGIST	0.04121828	2.22578709	96.2963
7	COS	0.04261081	2.30098352	96.2963
5	GAUSS	0.04264474	2.30281595	96.2963

\*\*\*\*\*  
Classification Table for CUTOFF = 0.5000  
\*\*\*\*\*

Activ.	Acc.	Observed	Predicted	
			remiss01	remiss02
EXP	96.2963	remiss01	9.0000000	0.0000000
		remiss02	1.0000000	17.000000
SQUARE	96.2963	remiss01	9.0000000	0.0000000
		remiss02	1.0000000	17.000000
SIN	96.2963	remiss01	9.0000000	0.0000000

```

remiss02 1.000000 17.000000
ARCTAN 96.2963 remiss01 9.000000 0.000000
remiss02 1.000000 17.000000
TANH 96.2963 remiss01 9.000000 0.000000
remiss02 1.000000 17.000000
LOGIST 96.2963 remiss01 9.000000 0.000000
remiss02 1.000000 17.000000
COS 96.2963 remiss01 9.000000 0.000000
remiss02 1.000000 17.000000
GAUSS 96.2963 remiss01 9.000000 0.000000
remiss02 1.000000 17.000000

```

```

*****
Activation Ordered by SSE Criterion (Stage 5)
*****

```

Run	Activ.	SSE	RMSE	MSE	AMSE	Accur.
8	EXP	2.15642868	1.46847836	0.03993386	0.03898138	96.2963
1	SQUARE	2.19788725	1.48252732	0.04070162	0.04087441	96.2963
6	SIN	2.21179892	1.48721179	0.04095924	0.04114150	96.2963
3	ARCTAN	2.21217623	1.48733864	0.04096623	0.04114542	96.2963
2	TANH	2.21219157	1.48734380	0.04096651	0.04114569	96.2963
4	LOGIST	2.21762328	1.48916865	0.04106710	0.04121828	96.2963
7	COS	2.30211274	1.51727148	0.04263172	0.04261081	96.2963
5	GAUSS	2.30582769	1.51849521	0.04270051	0.04264474	96.2963

Accuracy of Best Solution= 96.2963 at Stage 5

```

*****
Summary of All Optimizations in Stage=5 in Iteration 1
*****

```

		SQUARE	TANH	ARCTAN	LOGIST	GAUSS
CMP1	Crit	0.0408744	0.0413393	0.0412130	0.0417229	0.0314523
	Iter	1	4	14	2	11
	Gmax	7.94e-015	4.21e-004	4.51e-004	2.77e-004	3.60e-004
CMP2	Crit	0.0408744	0.0411746	0.0411644	0.0416577	0.0325506
	Iter	1	11	12	10	4
	Gmax	9.87e-016	4.81e-004	3.77e-004	4.23e-004	2.50e-004
CMP1		SIN	COS	EXP		
	Crit	0.0355200	0.0422787	0.0389807		
	Iter	12	6	200		
	Gmax	3.18e-005	9.45e-005	5.86e+009		

CMP2 Crit 0.0411565 0.0426102 0.0389807  
 Iter 12 7 200  
 Gmax 4.47e-004 1.10e-005 5.86e+009

SSE of Best Solution= 2.15643 at Stage 5

\*\*\*\*\*  
 Stage 5 : Criterion SSE= 2.15643  
 \*\*\*\*\*

Comp Activation  
 1 EXP  
 2 EXP

\*\*\*\*\*  
 Summary Table Across Stages  
 \*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	3.70788814	0.41053669	74.0741	-43.6051084	-37.1259241
1	2.91380424	0.41400513	85.1852	-40.1121882	-27.1538195
2	2.79653007	0.48274649	81.4815	-31.2213519	-11.7837989
3	2.52411377	0.60048954	92.5926	-23.9885648	1.92817250
4	2.30632429	1.07385388	96.2963	-16.4249093	15.9710124
5	2.15642868	1.46847836	96.2963	-8.23935183	30.6357541

Stage	Link	CMP1	CMP2
0	LOGIST	EXP	EXP
1	IDENTITY	SIN	SIN
2	IDENTITY	SQUARE	SQUARE
3	IDENTITY	SQUARE	SQUARE
4	IDENTITY	EXP	EXP
5	IDENTITY	EXP	EXP

Time for Optimization: 0  
 Total Processing Time: 1  
 Number of Optimizations : 160  
 Number of Runs through Data : 53

Classification Table: Accuracy=96.2963

\*\*\*\*\*

Dense Matrix (2 by 2)

| remiss01 remiss02



```

-----
remiss01 |          9          0
remiss02 |          1          17

```

Expected Values and Residuals of Training Data

\*\*\*\*\*

Dense Matrix (27 by 4)

	Stage	Observed	Predicted	Residual
1	5.000000	1.000000	0.7612488	0.2387512
2	5.000000	1.000000	0.5678584	0.4321416
3	5.000000	0.000000	0.0000000	0.0000000
4	5.000000	0.000000	0.0000000	0.0000000
5	5.000000	1.000000	0.8070445	0.1929555
6	5.000000	0.000000	0.2119425	-0.2119425
7	5.000000	1.000000	0.5281022	0.4718978
8	5.000000	0.000000	0.4600456	-0.4600456
9	5.000000	0.000000	0.3445774	-0.3445774
10	5.000000	0.000000	0.0000000	0.0000000
11	5.000000	0.000000	0.2929989	-0.2929989
12	5.000000	0.000000	0.1887860	-0.1887860
13	5.000000	0.000000	0.0000000	0.0000000
14	5.000000	0.000000	0.1739215	-0.1739215
15	5.000000	0.000000	0.7107106	-0.7107106
16	5.000000	1.000000	0.8690338	0.1309662
17	5.000000	0.000000	0.0000000	0.0000000
18	5.000000	0.000000	0.3639751	-0.3639751
19	5.000000	0.000000	0.0841418	-0.0841418
20	5.000000	1.000000	0.8476975	0.1523025
21	5.000000	0.000000	0.2084748	-0.2084748
22	5.000000	0.000000	0.1001781	-0.1001781
23	5.000000	1.000000	0.5988034	0.4011966
24	5.000000	0.000000	0.0890046	-0.0890046
25	5.000000	1.000000	1.0000000	0.0000000
26	5.000000	1.000000	0.6419939	0.3580061
27	5.000000	0.000000	0.0741363	-0.0741363

## 4.11 Function `nlfitprd`

---

```
gof = nlfitprd(data,parm,stat,modl<,optn<,class actf<,link> .. >)
```

```
<gof,scor,fit,tabs> = nlfitprd(data,parm,stat,modl<,optn<,...> .. >)
```

**Purpose:** The `nlfitprd` function performs scoring of additional data sets by using the model information returned from the `nlfit()` function. The `parm` and `stat` output arguments from the training run of `nlfit()` are being used for creating the model for scoring the input data set.

**Input: data** This is an  $N \times nc$  matrix containing  $N$  numerical observations of  $nc$  variables. The  $nc$  columns should have the same meaning as those of the training data set used for the prior `nlfit()` call complying to the same model specification.

**parm** contains the `parm` output argument from the training run with `nlfit()`.

**stat** contains the `stat` output argument from the training run with `nlfit()`.

**modl** : This must be the same model specification as was used at the prior `nlfit()` call.

**optn** This must be the same options matrix specification as was used at the prior `nlfit()` call.

**class** If specified in the training run, this must be the same specification of columns for CLASS variables as was used at the prior `nlfit()` call.

**actf** must be identical to the specification for the training run with `nlfit()`.

**link** must be identical to the specification for the training run with `nlfit()`.

**Output: gof** this is vector of some scalar results.

**scor** This is an  $N \times 4$  matrix containing the  $N$  predicted model values and residuals obtained by scoring the input data using the optimal model weights from the prior `nlfit()` call.

**fit** contains a table of fit indices for the input data.

**tabs** contains an accuracy table for the input data.

**Restrictions:** 1.

**Relationships:** `nlfit()`

**Examples:** 1. Binary Response  $y$ : Cancer Remission Data

The fit of the following input was illustrated documenting the `nlfit()` function:

```

print "Multiple Activation Model: SELCR=SSE, MAXVEC=0";
clas = 1;
modl = "1 = 2:7";
optn = [ "print"      3 ,
         "ptab"      2 ,
         "pfit"      ,
         "pini"      ,
         "pres"      ,
         "popt"      0 ,
         "model"    "mult" ,
         "selcr"    "sse" ,
         "maxvec"   0 ,
         "maxstag"  6 ];
< gof,parm,fit,tabs,stat,scor,tscor > =
      nlfit(remis,modl,optn,clas,.,.,.,.,temis);

```

Here, `temis` contains only the first ten observations of the data set `remis`. Using the `parm` and `stat` output from the `nlfit()` call we can now score the small `temis` data set separately:

```

< gof,scor,fit,tabs > = nlfitprd(temis,parm,stat,modl,optn,clas);
print "GOF=", gof;
print "Scor=", scor;
print "Fit=", fit;
print "Tabs=", tabs;

```

```

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

```

```

Number Valid Observations  10
Response Variable          Y[1]
N Independent Variables    6
NOBS w/o Missing Target   10
Binary Target              remiss
* Multiple Activation Model *
PCA Common All Evals and Evecs
First Link Function        LOGIST
Selection Criterion        SSE
Optimize                   SSE
Max. Estimation Stages    6
Max. Number Components    0
No Princ. Component Reduction
Input Data Remain in File
Store Eigenvectors Incore

```

\*\*\*\*\*  
 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]	10	0.9250000	0.0754615	-0.8484373	-0.4007818
X[3]	10	0.6990000	0.2306248	-0.6256798	-1.3503414
X[4]	10	0.6420000	0.2112292	-0.4092350	-1.0102023
X[5]	10	1.0900000	0.5130519	0.6891446	-0.9111471
X[6]	10	0.7013000	0.4675931	-0.0322886	-1.4121809
X[7]	10	0.9967000	0.0186312	1.5175245	1.8000133

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

Variable	Value	Nobs	Proportion
Y[1]	0	4	40.000000
	1	6	60.000000

Classification Table: Accuracy=100

\*\*\*\*\*

Dense Matrix (2 by 2)

	remiss01	remiss02
remiss01	4	0

remiss02 | 0 6

Model Fit Across Stages

\*\*\*\*\*

Dense Matrix (2 by 6)

	Stage	SSE	RMSE	Accuracy	AIC
STAG_1	0.0000000	7.86e-007	8.86e-004	100.00000	-137.59272
STAG_2	1.0000000	4.48e-006	0.0021163	100.00000	-120.18729

	SBC
STAG_1	-133.65911
STAG_2	-116.25368

Expected Values and Residuals of Training Data

\*\*\*\*\*

Dense Matrix (10 by 4)

	Stage	Observed	Predicted	Residual
1	1.0000000	1.0000000	0.9998054	1.95e-004
2	1.0000000	1.0000000	0.9986656	0.0013344
3	1.0000000	0.0000000	6.94e-004	-6.94e-004
4	1.0000000	0.0000000	0.0000000	0.0000000
5	1.0000000	1.0000000	1.0000000	0.0000000
6	1.0000000	0.0000000	0.0000000	0.0000000
7	1.0000000	1.0000000	0.9996613	3.39e-004
8	1.0000000	0.0000000	5.79e-004	-5.79e-004
9	1.0000000	0.0000000	5.49e-004	-5.49e-004
10	1.0000000	0.0000000	1.35e-004	-1.35e-004

Time for Optimization: 0  
Total Processing Time: 0  
Number of Optimizations : 0  
Number of Runs through Data : 5

2. Predicting Biological Activity: nrow=16 ; ncol=27:

This test example is taken from the PROC PLS chapter of the SAS/STAT manual. There are more predictor variables (26) than observations (16):

```

biotrn = ["
EM1  2766 2610 3306 3630 3600 3438 3213 3051 2907 2844 2796
      2787 2760 2754 2670 2520 2310 2100 1917 1755 1602 1467
      1353 1260 1167 1101 1017                3.0110 0.0000 0.00,
EM2  1492 1419 1369 1158 958 887 905 929 920 887 800
      710 617 535 451 368 296 241 190 157 128 106
      89 70 65 56 50                0.0000 0.4005 0.00,
.....
EM16 4017 4725 6090 6570 6354 5895 5346 4911 4611 4422 4314
      4287 4224 4110 3915 3600 3240 2913 2598 2325 2088 1917
      1734 1587 1452 1356 1257                3.1620 0.7012 60.00 "];

nr = nrow(biotrn); nc = ncol(biotrn);
print "nrow=",nr," ncol=",nc;
cnam = [ "obsnam" "v1":"v27" "ls" "ha" "dt" ];
biotrn = cname(biotrn,cnam);
biotrn = biotrn[,2:31];
cnam = [ "v1":"v27" "ls" "ha" "dt" ];
biotrn = cname(biotrn,cnam); /* print "Biotrn=",biotrn; */

```

The following test data set has only missing response values:

```

biotst = ["
EM17 3933 4518 5637 6006 5721 5187 4641 4149 3789
      3579 3447 3381 3327 3234 3078 2832 2571 2274
      2040 1818 1629 1470 1350 1245 1134 1050 987 . . . ,
EM25 2904 2997 3255 3150 2922 2778 2700 2646 2571
      2487 2370 2250 2127 2052 1713 1419 1200 984
      795 648 525 426 351 291 240 204 162 . . . "];

cnam = [ "obsnam" "v1":"v27" "ls" "ha" "dt" ];
biotst = cname(biotst,cnam);
biotst = biotst[,2:31];
cnam = [ "v1":"v27" "ls" "ha" "dt" ];
biotst = cname(biotst,cnam); /* print "Biotst=",biotst; */

```

```

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

```

```

print "Multiple Activation Model: Perfect Fit";
modl = "28 = 1:27";
optn = [ "print" 3 ,

```

```

      "ptab"          1 ,
      "pfit"         ,
      "pres"         ,
      "model"        "mult" ,
      "pvec"         ,
      "popt"         0 ,
      "maxvec"       0 ,
      "maxstag"      6 ];
< gof,parm,fit,tabs,stat,scor,tscor > =
      nlfit(biotrn,modl,optn,.,.,.,.,,biotst);

```

```

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

```

```

Number Valid Observations  16
Observations Test Data    2
Response Variable          Y[28]
N Independent Variables    27
NOBS w/o Missing Target   16
Interval Target           1s
Target Minimum:           0.0000e+000
Target Maximum:           4.1320e+000
* Multiple Activation Model *
First Link Function       IDENTITY
Selection Criterion       SSE
Optimize                  SSE
Max. Estimation Stages   6
Number Y Percentiles     10
Max. Number Components
No Princ. Component Reduction
Store Input Data Incore
Store Eigenvectors Incore

```

```

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

```

```

X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 +
X12 + X13 + X14 + X15 + X16 + X17 + X18 + X19 + X20 + X21 + X22 +
X23 + X24 + X25 + X26 + X27

```

```

*****
Simple Statistics

```

\*\*\*\*\*

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[28]	16	2.2521125	1.2945280	-0.3821260	-0.7711713
X[ 1]	16	3666.7500	1044.7061	-0.3870322	-0.4008330
X[ 2]	16	3892.1250	1231.9569	-0.4025267	-0.6899339
.....					
X[27]	16	861.43750	434.34701	-0.6249492	-0.4081236

\*\*\*\*\*  
Percentiles of Target ls in [0 : 4.132]  
\*\*\*\*\*

N	Nobs	Y Value	Label
1	2	0.0000	ls_0000
2	3	1.1160	ls_2701
3	5	1.4820	ls_3587
4	6	2.1600	ls_5227
5	8	2.4280	ls_5876
6	10	3.0110	ls_7287
7	11	3.1620	ls_7652
8	13	3.1900	ls_7720
9	14	4.0240	ls_9739
10	16	4.1320	ls_1000

Already the first stage shows perfect fit of the square function:

\*\*\*\*\*  
Summary of All Optimizations in Stage=0  
\*\*\*\*\*

	SQUARE	TANH	ARCTAN	LOGIST	GAUSS	SIN
Crit	1.04e-024	9.93e-010	6.53e-011	1.72e-006	0.0205245	8.18e-014
Iter	1	5	5	6	65	6
Gmax	5.98e-014	1.12e-005	6.41e-006	3.00e-004	2.54e-004	2.98e-007

  

	COS	EXP
Crit	1.37e-012	8.88e-006
Iter	4	7
Gmax	6.56e-007	8.60e-005

\*\*\*\*\*  
Activation Ordered by SSE Criterion  
\*\*\*\*\*



Run	Activ.	SSE	RMSE	MSE	Accur.
1	SQUARE	5.688e-022	2.385e-011	1.041e-024	1.0000
6	SIN	4.469e-011	6.685e-006	8.179e-014	1.0000
7	COS	7.474e-010	2.734e-005	1.368e-012	1.0000
3	ARCTAN	3.568e-008	1.889e-004	6.531e-011	1.0000
2	TANH	5.423e-007	7.364e-004	9.926e-010	1.0000
4	LOGIST	9.405e-004	0.03066722	1.721e-006	1.0000
8	EXP	0.00485042	0.06964493	8.878e-006	1.0000
5	GAUSS	11.2135282	3.34866066	0.02052446	0.4898

SSE of Best Solution= 5.68841e-022 at Stage 0

\*\*\*\*\*  
Summary Table Across Stages  
\*\*\*\*\*

Stage	SSE	RMSE	Accur.	AIC	SBC
0	5.688e-022	2.385e-011	1.0000	-717.056490	-674.564110

Stage	Link	VAR1	VAR2	VAR3	VAR4	VAR5
0	IDENTITY	SQUARE	SQUARE	SQUARE	SQUARE	SQUARE
		SQUARE	SQUARE	SQUARE	SQUARE	SQUARE
		SQUARE	SQUARE	SQUARE	SQUARE	SQUARE
		SQUARE	SQUARE	SQUARE	SQUARE	SQUARE
		SQUARE	SQUARE	SQUARE	SQUARE	SQUARE
		SQUARE	SQUARE	SQUARE	SQUARE	SQUARE

Time for Optimization: 1  
Total Processing Time: 1  
Number of Optimizations : 9  
Number of Runs through Data : 401

The following are the scor and tscor results:

Scor=	Stage	Observed	Predicted	Residual
1	0.00000	3.0110	3.0110	2e-012
2	0.00000	0.00000	-1e-012	1e-012
3	0.00000	0.00000	-5e-012	5e-012
4	0.00000	1.4820	1.4820	-2e-012
5	0.00000	1.1160	1.1160	-5e-012
6	0.00000	3.3970	3.3970	-2e-012
7	0.00000	2.4280	2.4280	-4e-012

8	0.00000	4.0240	4.0240	5e-012
9	0.00000	2.2750	2.2750	1e-011
10	0.00000	0.95880	0.95880	-1e-011
11	0.00000	3.1900	3.1900	-3e-012
12	0.00000	4.1320	4.1320	-7e-012
13	0.00000	2.1600	2.1600	-2e-012
14	0.00000	3.0940	3.0940	8e-012
15	0.00000	1.6040	1.6040	4e-012
16	0.00000	3.1620	3.1620	-2e-012

Note, the observed values are missing and so are the residuals:

```
TScor=
```

U	Stage	Observed	Predicted	Residual
-----				
1	0.00000	.	2.1402	.
2	0	.	-1.3116	.

```
< gof,scor,fit,tabs > = nlfprd(biotst,parm,stat,modl,optn);
print "GOF=", gof;
print "Scor=",scor;
print "Fit=",fit;
print "Tabs=",tabs;
```

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

```
Number Valid Observations    0
Response Variable             Y[28]
N Independent Variables       27
NOBS w/o Missing Target      0
Interval Target               1s
Target Values are all Missing
* Multiple Activation Model *
PCA Common All Evals and Evecs
First Link Function          IDENTITY
Selection Criterion           SSE
Optimize                      SSE
Max. Estimation Stages        6
Number Y Percentiles          10
Max. Number Components        0
No Princ. Component Reduction
Input Data Remain in File
```

Store Eigenvectors Incore

Expected Values and Residuals of Training Data  
\*\*\*\*\*

Dense Matrix (2 by 4)

		Stage	Observed	Predicted	Residual
1		0.0000000	.	2.1402421	.
2		0.0000000	.	-1.3115915	.

Time for Optimization: 0  
Total Processing Time: 0  
Number of Optimizations : 0  
Number of Runs through Data : 4

## 5 Illustration

### 5.1 Random Generators for Normal and Exponential Variates

The `rand` function has the following syntax:

```
a = rand(nr<,nc<,mtyp<,dist<,>>>>)
```

where the fourth input argument should be a string specifying the random distribution. Here we only illustrate some use when `mtyp` is equal to 'g' and `dist` specifies normal and exponential variates.

For normal variates the following four algorithms can be specified:

"norm" `randlib` version

"nor2" old or new Ziggurat method (Marsaglia & Tsang, 2000)

"nor3" Fishman (1996), p.190: Box and Muller (1958)

"nor4" Fishman (1996), p.191: Ahrens and Dieter (1988)

For exponential variates the following four algorithms can be specified:

"expo" `randlib` version

"exp2" old or new Ziggurat method (Marsaglia & Tsang, 2000)

"exp3" Fishman (1996), p.188

"exp4" Fishman (1996), p.189

The four algorithms for each of the two distributions are compared in its quantiles and histograms for  $n = 1000$  and its computation times for  $n = 10,000,000$ .

### 5.1.1 Normal Variates $n = 1000$

```
nr = 1000; nc = 1;
dti1 = time("clock");
nor1 = rand(nr,nc,'g',"norm");
print "DTI1=", dti1 = time("clock") - dti1;
print "NOR1=",nor1[1:10];
qual = quantile(nor1,4,1);
optn = [ 1, ., -3.5, 3.5 ];
his1 = histogram(nor1,9,optn);
print "Histogram 1:", his1;
titl = "Histogram of NOR1";
histplot(his1,titl,2);
```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |          1          2          3          4          5
   -2.7497151 -0.6463936  0.0262144  0.6864117  3.2940445
```

\*\*\*\*\*  
Histogram of NOR1  
\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1  1.0000000
2  30.0000000 ****
3  94.0000000 *****
4  200.000000 *****
5  329.000000 *****
6  218.000000 *****
7  96.0000000 *****
8  25.0000000 ***
9  7.0000000
```

```

dti2 = time("clock");
nor2 = rand(nr,nc,'g',"nor2");
print "DTI2=", dti2 = time("clock") - dti2;
qua2 = quantile(nor2,4,1);
optn = [ 1, ., -3.5, 3.5 ];
his2 = histogram(nor2,9,optn);
print "Histogram 2:", his2;
titl = "Histogram of NOR2";
histplot(his2,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
   -3.4114823 -0.6717438  0.0598014  0.6574269  3.2105293

```

\*\*\*\*\*  
Histogram of NOR2  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1  1.0000000
2  24.0000000 ***
3 102.0000000 *****
4 206.0000000 *****
5 312.0000000 *****
6 236.0000000 *****
7  90.0000000 *****
8  23.0000000 ***
9   6.0000000

```

```

dti3 = time("clock");
nor3 = rand(nr,nc,'g',"nor3");
print "DTI3=", dti3 = time("clock") - dti3;
qua3 = quantile(nor3,4,1);
optn = [ 1, ., -3.5, 3.5 ];
his3 = histogrm(nor3,9,optn);
print "Histogram 3:", his3;
titl = "Histogram of NOR3";
histplot(his3,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
  -2.9706513 -0.6528997 -0.0010251  0.6639345  3.1173451

```

\*\*\*\*\*  
Histogram of NOR3  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1  3.0000000
2  25.0000000 ***
3  103.000000 *****
4  206.000000 *****
5  318.000000 *****
6  223.000000 *****
7  100.000000 *****
8  18.0000000 **
9  4.00000000

```

```

dti4 = time("clock");
nor4 = rand(nr,nc,'g',"nor4");
print "DTI4=", dti4 = time("clock") - dti4;
qua4 = quantile(nor4,4,1);
optn = [ 1, ., -3.5, 3.5 ];
his4 = histgrm(nor4,9,optn);
print "Histogram 4:", his4;
titl = "Histogram of NOR4";
histplot(his4,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
   -3.4590557 -0.7275371 -0.0268197  0.6846597  3.2676747

```

\*\*\*\*\*  
Histogram of NOR4  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1  3.0000000
2 14.0000000 *
3 102.000000 *****
4 249.000000 *****
5 292.000000 *****
6 218.000000 *****
7  88.000000 *****
8  25.000000 ***
9   9.0000000

```



```

quan = qua1 |> qua2 |> qua3 |> qua4;
print "Quantile Normal: for n=",nr,quan;

dtim = [ dti1 dti2 dti3 dti4 ];
cnam = [" dti1:dti4 "];
dtim = cname(dtim,cnam);
print "Clock Time Normal: for n=",nr,dtim;

```

```

Quantile Normal: for n= 1000
|          1          2          3          4          5
-----
1 |   -2.7497  -0.64639  0.02621  0.68641  3.2940
2 |   -3.4115  -0.67174  0.05980  0.65743  3.2105
3 |   -2.9707  -0.65290  -0.00103  0.66393  3.1173
4 |   -3.4591  -0.72754  -0.02682  0.68466  3.2677

```

```

Clock Time Normal: for n= 1000
Z |   dti1   dti2   dti3   dti4
-----
1 |     0     0     0     0

```

5.1.2 Normal Variates  $n = 10,000,000$

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

R	1	2	3	4	5
	-5.0354061	-0.6745115	3.78e-004	0.6748498	5.2947044

\*\*\*\*\*  
Histogram of NOR1  
\*\*\*\*\*

N	Value	
1	31849.0000	
2	226805.000	***
3	959306.000	*****
4	2267508.00	*****
5	3026974.00	*****
6	2270607.00	*****
7	957995.000	*****
8	226657.000	***
9	32299.0000	

Quantiles

\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    -5.1308284 -0.6744744  1.60e-004  0.6750277  5.2687731
```

\*\*\*\*\*

Histogram of NOR2

\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1  32305.0000
2  227502.000 ***
3  958172.000 *****
4  2268852.00 *****
5  3024687.00 *****
6  2270800.00 *****
7  957641.000 *****
8  227731.000 ***
9  32310.0000
```

Quantiles

\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    -5.7917526 -0.6745050 -1.23e-004  0.6741979  5.2084704
```

\*\*\*\*\*

Histogram of NOR3

\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1  32083.0000
2  226099.000 ***
3  957532.000 *****
4  2271038.00 *****
5  3028156.00 *****
6  2268366.00 *****
7  958390.000 *****
8  225610.000 ***
9  32726.0000
```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    -5.3474886 -0.6744193  8.33e-005  0.6744601  5.1062581
```

\*\*\*\*\*  
Histogram of NOR4  
\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1  32486.0000
2  226817.000 ***
3  957185.000 *****
4  2269175.00 *****
5  3026564.00 *****
6  2270251.00 *****
7  957770.000 *****
8  227232.000 ***
9  32520.0000
```

```

Quantile Normal: for n= 10000000
|          1          2          3          4          5
-----
1 | -5.0354 -0.67451  0.00038  0.67485  5.2947
2 | -5.1308 -0.67447  0.00016  0.67503  5.2688
3 | -5.7918 -0.67450 -0.00012  0.67420  5.2085
4 | -5.3475 -0.67442  0.00008  0.67446  5.1063

```

```

Clock Time Normal: for n= 10000000
|      dti1      dti2      dti3      dti4
-----
1 |   6.3590   3.4380   5.3590   7.5160

```

### 5.1.3 Exponential Variates $n = 1000$

```
nr = 1000; nc = 1;
dti1 = time("clock");
exp1 = rand(nr,nc,'g',"expo");
print "EXP1=", dti1 = time("clock") - dti1;
print "EXP1=",exp1[1:10];
qual = quantile(exp1,4,1);
optn = [ 1, ., 0.];
his1 = histogram(exp1,7,optn);
print "Histogram 1:", his1;
titl = "Histogram of EXP1";
histplot(his1,titl,2);
```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    5.25e-005  0.2760464  0.7352111  1.3306999  9.9587917
```

\*\*\*\*\*  
Histogram of EXP1  
\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1 774.000000 *****
2 168.000000 *****
3 44.000000 **
4 9.00000000
5 3.00000000
6 1.00000000
7 1.00000000
```

```

dti2 = time("clock");
exp2 = rand(nr,nc,'g',"exp2");
print "DTI2=", dti2 = time("clock") - dti2;
qua2 = quantile(exp2,4,1);
optn = [ 1, ., 0.];
his2 = histogramm(exp2,7,optn);
print "Histogram 2:", his2;
titl = "Histogram of EXP2";
histplot(his2,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
   1.46e-004  0.3021851  0.7227114  1.4628875  9.9001846

```

\*\*\*\*\*  
Histogram of EXP2  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1 736.000000 *****
2 200.000000 *****
3 49.0000000 ***
4 9.00000000
5 4.00000000
6 0.00000000
7 2.00000000

```



```

dti3 = time("clock");
exp3 = rand(nr,nc,'g',"exp3");
print "DTI3=", dti3 = time("clock") - dti3;
qua3 = quantile(exp3,4,1);
optn = [ 1, ., 0.];
his3 = histogramm(exp3,7,optn);
print "Histogram 3:", his3;
titl = "Histogram of EXP3";
histplot(his3,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
    0.0017192  0.2876566  0.7043681  1.4808905  7.1233840

```

\*\*\*\*\*  
Histogram of EXP3  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1 638.000000 *****
2 217.000000 *****
3 96.0000000 *****
4 33.0000000 **
5 12.0000000
6 2.00000000
7 2.00000000

```

```

dti4 = time("clock");
exp4 = rand(nr,nc,'g',"exp4");
print "DTI4=", dti4 = time("clock") - dti4;
qua4 = quantile(exp4,4,1);
optn = [ 1, ., 0.];
his4 = histogrm(exp4,7,optn);
print "Histogram 4:", his4;
titl = "Histogram of EXP4";
histplot(his4,titl,2);

```

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```

R |           1           2           3           4           5
   1.65e-004  0.3129143  0.7887144  1.6437414  16.666641

```

\*\*\*\*\*  
Histogram of EXP4  
\*\*\*\*\*

```

N      Value +-----+-----+-----+-----+
1 868.000000 *****
2 100.000000 *****
3 22.0000000 *
4 3.00000000
5 5.00000000
6 1.00000000
7 1.00000000

```

```

quan = qua1 |> qua2 |> qua3 |> qua4;
print "Quantile Exponential: for n=",nr,quan;

dtim = [ dti1 dti2 dti3 dti4 ];
cnam = [" dti1:dti4 "];
dtim = cname(dtim,cnam);
print "Clock Time Exponential: for n=",nr,dtim;

```

```

Quantile Exponential: for n= 1000
|          1          2          3          4          5
-----
1 |    0.0001    0.27605    0.73521    1.3307    9.9588
2 |    0.0001    0.30219    0.72271    1.4629    9.9002
3 |    0.0017    0.28766    0.70437    1.4809    7.1234
4 |    0.0002    0.31291    0.78871    1.6437    16.667

```

```

Clock Time Exponential: for n= 1000
|          dti1          dti2          dti3          dti4
-----
1 |    0.06300    0.00000    0.00000    0.00000

```

5.1.4 Exponential Variates  $n = 10,000,000$

Quantiles  
\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    1.19e-007  0.2878412  0.6932124  1.3865857  16.017792
```

\*\*\*\*\*  
Histogram of EXP1  
\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1 8984688.00 *****
2 912752.000 ****
3 92251.0000
4 9312.00000
5 910.000000
6 74.0000000
7 13.0000000
```

Quantiles

\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    3.72e-008  0.2879419  0.6935915  1.3862879  18.616442
```

\*\*\*\*\*

Histogram of EXP2

\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1 9299740.00 *****
2 651563.000 ***
3 45263.0000
4 3180.00000
5 236.000000
6 17.0000000
7 1.00000000
```

Quantiles

\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    9.22e-008  0.2877077  0.6937414  1.3868181  15.336960
```

\*\*\*\*\*

Histogram of EXP3

\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1 8881318.00 *****
2 993876.000 *****
3 110708.000
4 12521.0000
5 1403.00000
6 158.000000
7 16.0000000
```

Quantiles

\*\*\*\*\*

Dense Row Vector (ncol=5)

```
R |           1           2           3           4           5
    1.73e-007  0.2936155  0.7126597  1.4470499  24.590923
```

\*\*\*\*\*

Histogram of EXP4

\*\*\*\*\*

```
N      Value +-----+-----+-----+-----+
1 9548899.00 *****
2 381194.000 *
3 52805.0000
4 16346.0000
5 741.000000
6 14.0000000
7 1.00000000
```

```

Quantile Exponential: for n= 10000000
|          1          2          3          4          5
-----
1 |    1e-007    0.28784    0.69321    1.3866    16.018
2 |    4e-008    0.28794    0.69359    1.3863    18.616
3 |    9e-008    0.28771    0.69374    1.3868    15.337
4 |    2e-007    0.29362    0.71266    1.4470    24.591

```

```

Clock Time Exponential: for n= 10000000
|          dti1          dti2          dti3          dti4
-----
1 |    6.6570    3.6410    4.4690    4.3600

```

## 5.2 Matching the Behavior of an Index by a Small Number of Assets

### 5.2.1 Training a Model with 2006 Data of IMKB30

The `locatn()` function can be used to select a smaller set of securities from a stock index to match the behavior of the entire index (see Cornuejols & Tütüncü, 2006). We tested this using the Turkish IMKB30 stock index data for the year 2006 which had  $N = 250$  rows (observations) and of course  $n = 30$  columns (stocks).

Our approach is summarized as follows:

1. The first subproblem is the computation of a  $30 \times 30$  correlation matrix. The data which was available for us showed a large number of missing values. We considered the following two choices for computing the correlation matrix:
  - Using the `emcov()` function to obtain ML estimates for mean vector and covariance matrix of data with missing values. From the estimate of the covariance matrix we obtain the correlation matrix by standardization.
  - First we apply the `impute()` function for the imputation of missing values into the raw data set. Then we can use the `bivar()` function for the computation of the common correlation matrix of the imputed data set. As an alternative we can also use a *shrunked* covariance matrix (Ledoit & Wolf, 2003) by applying the `covshrk()` function to the imputed data set and then standardize this matrix.
2. After we have obtained a  $30 \times 30$  correlation matrix of the  $n = 30$  securities of the IMKB30 for the year 2006, we can now apply the `locatn()` function for selecting a subset of  $k < 30$  securities as representatives for the stock index.



- As the most simple model we can now apply a linear least squares regression model for the prediction of the 2006 index values from the stock values, i.e.

$$\min_{\beta} (y - X\beta)^T (y - X\beta)$$

where  $y = index$  is the vector of observed response values and  $\mathbf{X}$  is the  $N \times n$  predictor matrix with the closing values of the 30 stocks for the year 2006, and  $\beta$  is an  $n = 30$  vector of linear regression coefficients. We must impose nonnegativity bounds on the regression weights, i.e.  $\beta_j \geq 0$ .

- The same way we can formulate models for the  $k < n$  reduced data, and  $\beta$  now has dimension  $k < n = 30$ .
- Instead of using linear least squares regression we could also use the more robust LAV ( $L_1$ ) regression or even LMS or LTS. But imposing nonnegativity bounds on the estimates would be more difficult.
- After obtaining (linear least squares) regression coefficients  $\beta$  we can now compute *predicted* model values  $\hat{y}$  by plugging  $\beta$  into the linear model:

$$\hat{y} = X\beta$$

We can use the residuals  $y - \hat{y}$  for testing the fit of the linear model. Or we simply compute the correlation between the  $N=250$   $y$  and  $\hat{y}$  values. Even for the complete index of  $n = 30$  stocks the linear model will not be perfect. And we may expect that models for smaller  $k < n$  usually fit less well than for larger values of  $k$ .

- Using the same vector  $\beta$  we can also test how the model would work for some 2007 data ( $Z, z$ ) of the IMKB30 which were not used in the modeling. In data mining, this is called "test set scoring".

The following shows the CMAT input for some of these steps:

- Creating correlation matrices:

The file `imkb30_ind.dat` in the `tdata` directory contains a  $250 \times 4$  matrix with the opening, the high, the low, and the closing values of the IMKB30 index. Here we only use the closing value in column 4:

```

cnam = [" Open High Low Close "];
%inc "..\..\cmat\tdata\imkb30_ind.dat";

ind06 = imkb30_06[,4];
m = nrow(imkb30_06); n = ncol(imkb30_06);
print "nrow=",m," ncol=",n;
print "Index[1:10]=",ind06[1:10];

```

The file `imkb30.dat` in the `tdata` directory contains a  $250 \times 30$  matrix with the closing values of the  $n = 30$  stocks of the IMKB30. We read those data into the `imkb30` matrix and attach the column, i.e. stock names:

```
print "Daily Prices of IMKB30 Stock Fund in Turkey in 2006";
%inc "..\..\cmat\tdata\imkb30.dat";
m = nrow(imkb30); n = ncol(imkb30);
print "nrow=",m," ncol=",n;

cnam= [" AKBNK ARCLK DENIZ DOAS DOHOL DYHOL EREGL FINBN
        FORTS GARAN GSDHO HURGZ ISCTR ISGYO KCHOL MIGRS
        PETKM PTOFS SAHOL SISE SKBNK TCELL THYAO TOASO
        TSKB TUPRS ULKER VAKBN VESTL YKBNK "];
imkb30 = cname(imkb30,cnam);
imkb30 = rname(imkb30,rnam);
```

Apply the `emcov()` algorithm for ML estimates of the mean  $\mu$  and covariance matrix `cov1` for the data in `imkb30` which has missing values.

```
optn = [ 3 . 0. 1 . .0001 ];
< cov1,mu,b > = emcov(imkb30,optn);
/* print "EMCOV Result=",cov1; */
```

This algorithm converges in eight iterations:

```
EM Estimation of Covariance Matrix
      Iter   MaxChange
      1  0.107793358
      2  0.001865704
      3  0.001128260
      4  0.000690407
      5  0.000417762
      6  0.000251645
      7  0.000151387
      8  9.1088e-005
```

Convergence of EM estimation after 8 iterations.

Standardize the covariance `cov1` to a correlation matrix `corr1`:

```
var = sqrt(diag(cov1));
corr1 = inv(var) * cov1 * inv(var);
```

```
corr1 = cname(corr1,cnam);
corr1 = rname(corr1,cnam);
```

Apply `impute()` on `imkb30` for the imputation of missing values and obtain `imkbf1`:

```

ubc = imkb30[<>,];
lbc = imkb30[><,]; /* print "bounds=", lbc, ubc; */
bounds = lbc' -> ubc'; print bounds;

optn = [ "print"          1 ,
         "ppatt"         1 ,
         "cent"           ,
         "scal"           ,
         "start"         "linreg" ,
         "pinit"         1 ,
         "seed"          123 ,
         "tol"           1.e-3 ,
         "maxit"         30 ];
imkbf1 = impute(imkb30,"linreg",optn);
```

There are 21 missing values in the data to be estimated:

```

                Rows with Most Missing Values
                *****

Row:   19  20  21  22  23  27  61  88  91  92
Mis:   2   2   2   2   2   1   1   1   1   1
```

Apply `bivar()` to obtain the common correlation matrix `corr2` from the nonmissing data:

```

cov2 = bivar(imkbf1,"cov");
var = sqrt(diag(cov2));
corr2 = inv(var) * cov2 * inv(var);
corr2 = cname(corr2,cnam);
corr2 = rname(corr2,cnam);
```

Apply `covshr()` to obtain the shrunken covariance matrix `cov3` and standardize to the `corr3` correlation matrix:

```

< cov3,delta,fcov,scov,smu > = covshr(imkbf1,"mark");
var = sqrt(diag(cov3));
corr3 = inv(var) * cov3 * inv(var);
corr3 = cname(corr3,cnam);
corr3 = rname(corr3,cnam);
/* print "Imputation and Shrink CORR3=", corr3; */

```

2. Check how the linear model works for all  $n = 30$  stocks:

The following code shows how to solve a linear least squares regression problem with boundary constraints:

```

print "Formulation as Quadratic Model";
qmat = imkbf1' * imkbf1; qvec = -ind06' * imkbf1;
ubc = 10000;
lubc = cons(n,1,0.) -> cons(n,1,ubc);
/* lc = ubc -> cons(1,n,1.) -> ubc; */
lc = .;
x0 = cons(1,n,ubc/n);
x0 = cname(x0,cnam);
optn = [ "qpnuasp"      ,
        "print"       3 ];
< xr,gof > = qp(qmat,qvec,lc,optn,lubc,x0);
wgts = xr'; pred = imkbf1 * wgts;

```

After computing the predicted model values we compute the correlation between observed and predicted values:

```

ss4 = ssq(pred - ind06); rms = sqrt(ss4 / m);
print "BC Constrained Solution QR: Weights=", wgts;
print "Sum of weights=",wgts[+];
print "Solution QR: SSQ, RMS=", ss4, rms;
cuna = [" Observed Predicted "];
curv3 = ind06 -> pred; curv3 = cname(curv3,cuna);
print "Observed vs. predicted",curv3;
/* corr = 0.9998 */
corr = bivar(pred->ind06,"cor");
print "Corr=", corr[1,2];

```

Of course the fit is not perfect, but pretty close. Here are some of the observed vs. predicted values:

	Observed	Predicted
--	----------	-----------

1	50551.0	50562.3
2	51837.7	51890.9
3	52835.7	52972.6
4	53247.9	53320.2
5	53520.2	53597.3
6	56253.9	56335.9
7	56985.7	57097.7
8	55060.4	55069.8
9	56432.6	56471.3
10	57532.0	57441.3
.....		
240	50019.5	50015.1
241	49508.2	49511.0
242	48583.3	48563.1
243	48444.4	48411.2
244	48611.1	48575.7
245	48767.7	48732.4
246	48181.7	48166.1
247	48255.2	48305.4
248	47960.8	47970.2
249	48648.1	48737.7
250	48551.4	48721.4

And even the sum of squares of the residuals,  $SSE = 1009604$ , is very large the correlation is close to one:

Corr= 0.9999

### 3. Compute submodels for $k = 10(2)24$ selected stocks:

The following CMAT code shows a cycle for evaluating subset solutions with  $k = 10(2)24$  stocks:

```
print "Run the entire cycle for K=10(2)24: QP without EC";
options ps=2000;
ubc = 10000.;
nt = 8; corm = cons(nt,1,.);
res1 = res2 = cons(n,nt,.);
curv = ind06; wgts = cons(n,nt,.);
for (it = 1, k = 10; it <= nt; it++, k+=2) {
  /* [1] get xmat from solving location problem */
  < gof,xind,yind > = locatn(corr1,k,1);
  res1[,it] = xind;
```

```

res2[,it] = cnam[xind]';
iind = yind[1:k];
zmat = imkbfi[,iind]; /* print "ZMAT=",zmat; */

/* [2] Compute new weights for zmat[n,n]:
      zmat[n,n] contains many zero columns */
qmat = zmat' * zmat; qvec = -ind06' * zmat;
lubc = cons(k,1,0.) -> cons(k,1,ubc);
/* lc = ubc -> cons(1,k,1.) -> ubc; */
x0 = cons(1,k,ubc/k);
x0 = cname(x0,cnam[iind]);
optn = [ "qpnusp"      ,
        "print"      1 ];
< xr,gof > = qp(qmat,qvec,..,optn,lubc,x0);
iwgt = cons(n,1,0.);
iwgt[iind] = xr; iprd = zmat * xr';
wgt[,it] = iwgt; curv = curv -> iprd;
cork = bivar(ind06 -> iprd,"cor");
corm[it] = cork[1,2];
print "it=",it," k=",k," corr=",corm[it];
}

```

The following are the correlations between the observed values of the index and the predicted values from the linear model applied to a subset of  $k < n$  stocks which was selected by the `locatn()` algorithm:

```

Correlations of each portfolio with Index: corm=
      |          1
-----|-----
K10 | 0.99347
K12 | 0.99457
K14 | 0.99504
K16 | 0.99522
K18 | 0.99684
K20 | 0.99868
K22 | 0.99901
K24 | 0.99903

```

### 5.2.2 Model Prediction for some January 2007 Data

Here a rather serious problem arises: Obviously, there are two securities, ENKAI and IHLAS, from the 2006 IMKB30 replaced by new stocks, GSDHO and HURGZ, in the 2007 IMKB30. That means we can only model the 28 stocks

which are together contained in the 2006 and 2007 IMKB30 index. The subset correlation matrix is in `corr28` and the imputed data set is in `imkb28`.

```
ind28 = [ 1:10 13:30 ];
cnam28 = cnam[ind28];
imkb28 = imkb1[,ind28];
imkb28 = cname(imkb28,cnam28);
corr28 = corr1[ind28,ind28];
corr28 = cname(corr28,cnam28);
corr28 = rname(corr28,cnam28);
```

We selected data of the IMKB for January 4, which is the first day of the new year 2007, for January 15, 25, and 31. The observed responses are in `ind_04`, `ind_15`, `ind_25`, `ind_31` and the predictors are in `iprd04`, `iprd15`, `iprd25`, `iprd31`.

We first want to see how good the linear model works for the entire index of  $n = 28$  stocks. Even though the model fit of the 28 stocks for 2006 correlates with  $Corr = 0.9999$  as for  $n = 30$  stocks, the predicted values for the 2007 dates are very different from the observed values:

```
Forecast January 04: obs: 48551 pred: 47525.92
Forecast January 15: obs: 47626 pred: 44767.89
Forecast January 25: obs: 53033 pred: 50740.81
Forecast January 31: obs: 51852 pred: 47328.74
```

Therefore we cannot expect that submodel with  $k < 28$  are fitting much better the model trained in 2006 and applied to data in 2007.

```
it= 1 k= 10 corr= 0.9935
Forecast January 04: obs: 48551 pred: 59099.85
Forecast January 15: obs: 47626 pred: 42635.46
Forecast January 25: obs: 53033 pred: 47258.77
Forecast January 31: obs: 51852 pred: 46454.94
```

```
it= 2 k= 12 corr= 0.9946
Forecast January 04: obs: 48551 pred: 57699.87
Forecast January 15: obs: 47626 pred: 43263.22
Forecast January 25: obs: 53033 pred: 48173.58
Forecast January 31: obs: 51852 pred: 47155.68
```

```
it= 3 k= 14 corr= 0.9950
Forecast January 04: obs: 48551 pred: 57289.51
Forecast January 15: obs: 47626 pred: 42542.56
```

Forecast January 25: obs: 53033 pred: 47325.22  
Forecast January 31: obs: 51852 pred: 46286.16

it= 4 k= 16 corr= 0.9952  
Forecast January 04: obs: 48551 pred: 55753.10  
Forecast January 15: obs: 47626 pred: 42019.56  
Forecast January 25: obs: 53033 pred: 46691.82  
Forecast January 31: obs: 51852 pred: 45735.22

it= 5 k= 18 corr= 0.9968  
Forecast January 04: obs: 48551 pred: 56661.55  
Forecast January 15: obs: 47626 pred: 45857.03  
Forecast January 25: obs: 53033 pred: 52191.79  
Forecast January 31: obs: 51852 pred: 50791.67

it= 6 k= 20 corr= 0.9987  
Forecast January 04: obs: 48551 pred: 51244.17  
Forecast January 15: obs: 47626 pred: 46552.94  
Forecast January 25: obs: 53033 pred: 52961.40  
Forecast January 31: obs: 51852 pred: 51026.05

it= 7 k= 22 corr= 0.9990  
Forecast January 04: obs: 48551 pred: 54063.39  
Forecast January 15: obs: 47626 pred: 49247.62  
Forecast January 25: obs: 53033 pred: 56106.50  
Forecast January 31: obs: 51852 pred: 50866.83

it= 8 k= 24 corr= 0.9992  
Forecast January 04: obs: 48551 pred: 54222.42  
Forecast January 15: obs: 47626 pred: 50319.77  
Forecast January 25: obs: 53033 pred: 57205.34  
Forecast January 31: obs: 51852 pred: 51626.67

It almost looks like that the 2006 model fits especially bad the first of the January 2007 days. At later days the model fit seems to get better.