CMAT Newsletter: November 2003

Wolfgang M. Hartmann

November 2003

1 General Remarks

During the last two months the major work concentrated on an alternative to the `arpack` function for the iterative computation of singular values and vectors for large and possibly sparse matrices. The new function `svdtrip` permits both, the explicit specification of a dense or sparse matrix or the use of an operator function for simulating a matrix-vector multiplication. The *Illustration* chapter on the end of this newsletter shows some tables comparing the computer times for computing singular triplets using `svdtrip` and `arpack` and a very limited comparison between the two versions of Kolmogorov-Smirnov tests for the uniform random distribution.

2 Modifications of Features

2.1 Fixed Bugs

A large number of bugs was fixed for the new CD which was shipped on October 20, 2003.
A very serious bug was fixed for the `arpack` function when the input matrix for svd is rectangular and sparse. It is strongly recommended that you ask me for a new CD if you want to run the `arpack` function for the SVD of rectangular matrices!
One of the known bugs is still not fixed: Using names of variables in option matrices of some functions are sometimes not recognized when functions are called inside functions. In such cases CMAT will issue an error that an object with that name could not be found and substitute a missing value instead. This was one of the reasons for changing the calling sequence of the `arpack` function. Matrix und function names are now provided by function arguments rather than by their names in the `option` argument.

2.2 New Runtime Option

The new runtime option `nalloc=i` can be used to extend the `listmem` output to more than ten largest memory allocations. For example, `options nalloc=25; lstmem();` will show the 25 largest allocations, assuming that so many already happened before.

2.3 The `%inc "file_name";` Alternative

The redirection of input for CMAT commands was already possible using the

`^#include "file_name"^`

clause compatible to the C preprocessor command. Here the ^ denotes a newline. In addition now the command

`^%inc "file_name";`

similar to that of the SAS language can be used. Note, that a newline (optionally followed only by blanks) must precede the `#` or the `%` symbols and the `#` command must be terminated by a newline. The may be spaces or tabs in front or behind the `#`, `%` and in front of the file name in quotes.

2.4 Changes in `kstest()` Function

Due to the new algorithm developed by paper by Marsaglia, Tsang, & Wang (2003) we were able to add a new test for univariate distribution of random numbers to the `kstest` function. The old version was identical to the Anderson-Darling form of the Cramer-von Mises-Kolmogorov-Smirnov test for univariate distribution which was also available in Marsaglia's *Diehard Battery* of tests for random generators. An additional string input option was added to the `kstest` function. The new syntax is now the following:

p = kstest(y<,"dist"<,"vers">>)

dist = "uni" :

vers="and" : Anderson-Darling approximative

vers="mtw" : Marsaglia-Tsang-Wang exact,JSS

dist = "nor" :

2.5 Changes in `arpack()` Function

We changed the input arguments for the `arpack` function making them more compatible with those of the new `svdtrip` function. The old function definition

< val,vec,par > = arpack("task",nv,ncv,optn)

< val,lvec,rvec,par > = arpack("task",nv,ncv,optn)

was changed to this definitions:

< val,vec,par > = arpack(a,"task",nv,ncv<,optn<,b>>)

< val,lvec,rvec,par > = arpack(a,"task",nv,ncv<,optn<,b>>)

and

< val,vec,par > = arpack(opfun,"task",nv,ncv<,optn>)

< val,lvec,rvec,par > = arpack(opfun,"task",nv,ncv<,optn>)

Here, the matrices **A** and **B** and the operator function `opfun` had to be specified formerly by entries of the `optn` matrix. Now they belong to the list of input arguments.

2.6 Changes in `glim()`, `glmixd()`, `glmod`, `cancor()`, `varsel()`, and `reg()`

If the main data set contains column names then they are used with the printed effect names. Not everything is perfect, but the printout now becomes more informative.

2.7 New Functions

New functions are implemented:

generead not finished yet.

hotell standard and robust Hotelling test (in work)

ksprob compute probability of Kolmogorov CDF

pls partial least squares and principal components regression

plslv partial least squares with latent variables (in work)

svdtrip uses four different iterative methods for computing the p largest singular values and corresponding left and right vectors.

3 New Developments

3.1 Function `generead`

a = generead("path",resp,"sep"<,optn<,code>>)

Even though heavily used already this function is still not finished yet since there are other input forms available and obviously nobody else than me is interested in this at this time.

3.2 Function `hotell`

< gof,parm > = hotell(x,mu0<,optn>)

Purpose: Compute T^2 statistics of One-Sample Hotelling test for hypothesis $H_0 : \mu = \mu_0$ for unknown **Sigma** is tested. Simultaneous confidence intervals for the estimated mean are computed as shown in Mardia, Kent, & Bibby (1979), pp. 144. There is also a robust version of the test available which is based on the MCD or MVE algorithm available in the `outl()` function.

Input: The first input argument must be a real $m \times n$ data matrix `X`. The second input argument must be a n vector mu_0 of assumed means. The third input argument is vector of some option parameters:

1 parameter specifying the amount of printed output (=0: no printed output);

2 specifying the classic or the robust version (=0: classic is default, =1: robust);

3 probability α for F test and confidence limits (default is $\alpha = .05$)

4 not used currently

5 parameter specifying the amount of printed output of the internal MCD call (=0: no printed output);

Output: The first output argument is a vector of some test statistics. The second output argument contains the estimated means and their confidence limits with respect to the default or specified α.

Restrictions:
1. No missing values in first two input arguments are permitted.
2. No string or complex data are permitted.
3. The first and second input arguments must be dimension compatible.

Relationships: outl()

Examples: 1. Philips Data (see Willems et.al., 2002): The robust version of the Hotelling test is used:

```
print "G. Willems: Philips Data: 677 by 6 matrix";
print "mu0 is the robust mean (data w/o outliers)";
options NOECHO;
phil = [
%inc "..\\tdata\\philips.dat";
       ];
options ECHO;
phil = shape(phil,.,6);
print "now=",nrow(phil);
```

We also want to see some of the MCD output:

```
optn = [  3 ,    /* printed output */
          1 ,    /* robust version */
         .05,    /* alpha */
          . ,
          2 ];   /* print some MCD output */
< gof,parm > = hotell(phil,mu0,optn);
```

```
             ************************************
             Fast MCD by Rousseeuw and Van Driessen
             ************************************
```

Number of Variables	6	Number of Observations	677
Specified Quantile	509	Default Quantile	342
Breakdown Value	24.96	Highest Breakdown	49.63
Total Number Subsets	2e+009	N Subset (Specified)	0

Variable	MinVal	1st Qu.	Median	Mean	3rd Qu.	MaxVal
V_1	-0.261000	-0.13000	-0.050000	-0.030607	0.02700	0.293000
V_2	-0.225000	-0.10050	-0.032000	-0.040270	9e-003	0.283000
V_3	0.353000	0.42100	0.436000	0.443362	0.46800	0.531000
V_4	1.876000	2.06650	2.116000	2.107959	2.15950	2.288000
V_5	0.342000	0.42000	0.436000	0.437334	0.45700	0.495000
V_6	-0.181000	-0.08750	-0.070000	-0.072672	-0.05200	0.168000

Variable	StdDev	MAD	Dispersion	S_n
V_1	0.11884369	0.08000000	0.11860818	0.12538969
V_2	0.08381061	0.05200000	0.07709532	0.08478732
V_3	0.02939197	0.02200000	0.03261725	0.03104888
V_4	0.06827701	0.04600000	0.06819970	0.07045707
V_5	0.02355676	0.01800000	0.02668684	0.02507794
V_6	0.03279899	0.01800000	0.02668684	0.02627213

```
      ****************************************************
      MCD Estimates (Obtained by Subsampling and Iteration)
      ****************************************************

MCD Location Estimate
---------------------

  1 :    -0.04364   -0.05019     0.4373      2.126     0.4359
  6 :    -0.06629
                Average of 509 Selected Points
```

```
**************************
Reweighted Robust Estimates
**************************

Reweighted Location Estimate
----------------------------

  1 :    -0.03848   -0.04547     0.4389      2.126      0.435
  6 :    -0.06563

Eigenvalues
-----------

  1 :     0.01949   0.004372   0.001882  0.0003945  0.0002078
  6 :   0.0001717
```

For space reason we skip most of the table with the 677 distances:

Classical Distances and Rousseeuw (Robust) Distances
Unsquared Mahalanobis Distance and
Unsquared Rousseeuw Distance of each Observation

Case	Mahalanobis Distances	Rousseeuw Distances	Weight
1	3.37945589992	3.64499828760	1
2	3.47703363255	3.58517283595	1
3	3.24836582084	3.40971146614	1
4	3.67614575384	3.78628453425	1
5	3.50254547104	3.71211967696	1
6	3.93200930657	4.04703005037	0
7	2.00178216624	2.12049190205	1
8	2.79930663954	3.41536803993	1
9	2.29235459497	2.63743471691	1
10	2.77963605581	3.24190543111	1
11	1.50156716209	1.67782030981	1
12	3.84887313509	4.53964699267	0
13	2.40767355089	3.01413776008	1
14	4.52524010957	5.45231601948	0
15	2.78890813966	3.39230787210	1
16	6.14860398847	7.84871588249	0
17	2.23231972523	2.72156949081	1
18	1.82533912621	2.05941937863	1
19	1.65689836676	1.77416563986	1
20	1.44235916677	1.54958564455	1
21	1.22823524325	1.28215587007	1
22	1.21417067382	1.32271675049	1
23	1.65521311129	1.86652623362	1
24	2.90932166453	3.56612140124	1
25	2.55208170785	3.31626643280	1

26	2.91182822166	4.41096543414	0
27	2.73163294428	3.51041404472	1
28	2.76990395389	3.32362313996	1
29	3.63497406481	4.49476041274	0
30	3.18982470325	3.75398072264	1
31	2.84163890983	3.36945245318	1
32	2.33318267085	2.61513772232	1
33	2.11753108117	2.33842179288	1
34	1.78728488825	2.20889204820	1
35	3.00280436914	3.06277756096	1
36	2.10092850688	2.22446130891	1
37	2.53571232483	2.81027484520	1
38	2.88322721974	3.17896563774	1
39	3.45750144581	3.86250972610	0
40	2.62446328337	2.98052116423	1

650	2.19084806632	2.19288959108	1
651	2.13123251275	2.07520556720	1
652	2.23095801536	2.19490138050	1
653	2.25471530731	2.19210623208	1
654	2.13084757369	2.35776144361	1
655	1.75251619522	1.69391830469	1
656	2.33699847936	2.64499332135	1
657	1.87695689391	1.82291003914	1
658	2.49811087972	2.46217649267	1
659	1.84710468223	1.77350825052	1
660	2.21840898526	2.13477039564	1
661	1.93571908343	2.12030831545	1
662	1.82973784716	1.78169953976	1
663	1.60428027018	1.74450232094	1
664	2.04155125618	2.03981755706	1
665	1.64301150666	1.73950780647	1
666	2.74377600237	2.89369904462	1
667	2.79262819744	2.84928328553	1
668	2.70298115928	2.79229421084	1
669	2.92780751405	3.05941698563	1
670	2.70387581100	2.75763291872	1
671	2.13799651546	2.26194770088	1
672	1.94064284733	2.06782110929	1
673	2.14647462287	2.14036547308	1
674	2.18530698765	2.29739442984	1
675	2.17539170787	2.17136961961	1
676	2.78908923140	2.86127960701	1
677	2.37803068518	2.53449291719	1

Distribution of Rousseeuw (Robust) Distances

MinRes	1st Qu.	Median	Mean	3rd Qu.	MaxRes

```
      0.3      2.06       2.48       3.01      3.23      12.3

              Default Cutoff Value= 3.80123
  The cutoff value is the square root of the 0.975 quantile
  of the chi square distribution with 6 degrees of freedom.
   N= 119 points have robust distances larger than cutoff.
Only points whose Rousseeuw distance is substantially larger
       than the cutoff should be considered outliers.


              ********************************
              Robust One-Sample Hotelling Test
              ********************************

      T^2_R= 5.82126 df1= 6 df2= 677 p-value=  0.4983
    Critical Cutoff Value= 13.7433 for Level Alpha= 0.05


   Reweighted MCD Location and  95.00% Confidence Limits
   ******************************************************

                  Dense Matrix (6 by 3)

              |        Mean      LowCI      UppCI
            ------------------------------------
          V_1 |  -0.0375498 -0.0567782 -0.0183215
          V_2 |  -0.0443523 -0.0576097 -0.0310949
          V_3 |   0.4390712  0.4345592  0.4435831
          V_4 |   2.1254466  2.1171390  2.1337543
          V_5 |   0.4350694  0.4316760  0.4384628
          V_6 |  -0.0657829 -0.0691625 -0.0624034
```

2. Head Data by Frets: see Mardia, Kent, Bibby (1979, p. 121, 126)

```
frets = [ 191  155    179  145,   195  149    201  152,
          181  148    185  149,   183  153    188  149,
          176  144    171  142,   208  157    192  152,
          189  150    190  149,   197  159    189  152,
          188  152    197  159,   192  150    187  151,
          179  158    186  148,   183  147    174  147,
          174  150    185  152,   190  159    195  157,
          188  151    187  158,   163  137    161  130,
          195  155    183  158,   186  153    173  148,
          181  145    182  146,   175  140    165  137,
          192  154    185  152,   174  143    178  147,
          176  139    176  143,   197  167    200  158,
          190  163    187  150 ];


fret2 = frets[,[1 3]];
mu0 = [ 182, 182];
```

```
optn = [  3 ,    /* ipri */
          0 ,    /* irob*/
         .05 ];  /* alpha */
< gof,parm > = hotell(fret2,mu0,optn);
```

The hypothesis is accepted:

```
                 *********************************
                 Classic One-Sample Hotelling Test
                 *********************************

          T^2_R= 1.92532 df1= 2 df2= 23 p-value=  0.1686
              F(df1=2,df2=23,alfa= 0.0500) = 3.42213


                  Estimates and Confidence Limits
                  *******************************

                       Dense Matrix (2 by 3)

                    |        Mean      LowCI      UppCI
                   -------------------------------------
                 V_1 |   185.72000  180.50245  190.93755
                 V_2 |   183.84000  178.47364  189.20636
```

Note, the 1979 edition of Mardia, Kent, & Bibby uses on page 126 in example 5.2.2. a divisor of $p = 3$ which should be $p = 2$. Even though the T^2 value here is higher, we are coming to the same conclusion of accepting the hypothesis.

3. Weight of Cork in 4 directions: Mardia, Kent, Bibby (1979, p. 12)

```
cork = [  72  66  76  77,    60  53  66  63,
          56  57  64  58,    41  29  36  38,
          32  32  35  36,    30  35  34  26,
          39  39  31  27,    42  43  31  25,
          37  40  31  25,    33  29  27  36,
          32  30  34  28,    63  45  74  63,
          54  46  60  52,    47  51  52  43,
          91  79 100  75,    56  68  47  50,
          79  65  70  61,    81  80  68  58,
          78  55  67  60,    46  38  37  38,
          39  35  34  37,    32  30  30  32,
          60  50  67  54,    35  37  48  39,
          39  36  39  31,    50  34  37  40,
          43  37  39  50,    48  54  57  43 ];


/* contrast tests mean=0 for X * cont' */
cont = [ 1  -1   1  -1 ,
         1   0  -1   0 ,
         0   1   0  -1 ];
```

```
cork2 = cork * cont';
sopt = [ "ari" "med" ];
print mu = univar(cork2,sopt);


/* MKB example 5.3.1, p.133 */
/* CI's: MKB p. 145 */
mu0 = cons(1,3,0.);
optn = [  3 ,    /* ipri */
          0 ,    /* irob*/
         .01 ];  /* alpha */
< gof,parm > = hotell(cork2,mu0,optn);
```

```
          |     Var_1     Var_2     Var_3
----------------------------------------
  Ari_Mean |   8.85714   0.85714   1.00000
  Median   |   7.00000   1.00000  -1.50000
```

The hypothesis for zero mean is rejected:

```
                 *********************************
                 Classic One-Sample Hotelling Test
                 *********************************

          T^2_R= 6.40186 df1= 3 df2= 25 p-value=  0.0023
              F(df1=3,df2=25,alfa= 0.0100) = 4.67546


                  Estimates and Confidence Limits
                  *******************************

                       Dense Matrix (3 by 3)

                    |        Mean      LowCI      UppCI
                   -------------------------------------
                V_1 |   8.8571429  0.5121074  17.202178
                V_2 |   0.8571429 -5.0057279  6.7200136
                V_3 |   1.0000000 -6.4716163  8.4716163
```

3.3 Function `ksprob`

p = ksprob(n,d)

Purpose: Compute Probability of Kolmogorov CDF : The algorithm by Marsaglia, Tsang, & Wang (JSS, 2003) is used.

Input: **n** sample size, can be scalar or $nr \times nc$ matrix,

d statistics, can be scalar or $nr \times nc$ matrix.

Note, if both input arguments are vectors or matrices, they must have the same dimension. In that case, the computations are performed for corresponding pairs of entries $p_{ij} = ksprob(n_{ij}, d_{ij})$ and the result p will have the same dimension $nr \times nc$.

Output: The result is either a scalar or an object $p = Prob(d < D_n)$, where

$$D_n = max(x_1 - \frac{0}{n}, x_2 - \frac{1}{n}, \ldots, x_n - \frac{n-1}{n}, \frac{1}{n} - x_1, \frac{2}{n} - x_2, \ldots, \frac{n}{n} - x_n)$$

The Anderson-Darling version of the Cramer-von Mises-Kolmogorov-Smirnov test for univariate distribution uses

$$A_n = -n - \frac{1}{n}[ln(x_1 z_1) + 3ln(x_2 z_2) + 5ln(x_3 z_3) + \cdots + (2n-1)ln(x_n z_n)]$$

with $z_i = 1 - x_{n+1-i}$.

Restrictions: 1. No string or complex data are permitted.

2. For any missing value in n and d a missing value is returned in v.

Relationships: kstest()

Examples: 1. Example of Paper by Marsaglia, Tsang and Wang, JSS 2003:

```
n = 2000; d = .04;
p = ksprob(n,d);
print "n,d,p=", n,d,p;


n = 2000; d = .06;
p = ksprob(n,d);
print "n,d,p=", n,d,p;


n = 16000; d = .016;
p = ksprob(n,d);
print "n,d,p=", n,d,p;
```

```
n,d,p= 2000 0.04000 0.9968
n,d,p= 2000 0.06000 1.0000
n,d,p= 16000 0.01600 0.9995
```

2. High prob Examples:

```
/* n=100,d=0.100000, K=0.7473072429936096 */
n = 100; d = .1;
p = ksprob(n,d);
print "n,d,p=", n,d,p;
```

```
n,d,p= 100 0.10000 0.7473
```

```
 /* n=100,d=0.200000, K=0.9994446415625527 */
 n = 100; d = .2;
 p = ksprob(n,d);
 print "n,d,p=", n,d,p;


n,d,p= 100 0.2000 0.9994


 /* n=100,d=0.300000, K=0.9999999800931475 */
 n = 100; d = .3;
 p = ksprob(n,d);
 print "n,d,p=", n,d,p;


n,d,p= 100 0.3000 1.0000


 /*--- Vector input d[3]: vector output ---*/
 n = 100; d = [ .1 .2 .3 ];
 p = ksprob(n,d);
 print "n,d,p=", n,d,p;


n,d,p= 100
   |         1         2         3
-----------------------------------
 1 |   0.10000   0.20000   0.30000

   |         1         2         3
-----------------------------------
 1 |   0.74731   0.99944   1.00000
```

3.4 Function `pls`

< rms,parm,yprd,vrms,trms,tprd > = pls(xtrn,yind,xind<,icmp<,optn<,xtst>>>)

Purpose: The `pls` function performs three forms of partial least squares (PLS) algorithms and one method of principal components regression (PCR). The three version of PLS are

- Kernel PLS (Lindgren, Geladi, & Wold, 1993; Dayal & McGregor, 1997)
- SIMPLS (de Jong, 1993; de Jong & Ter Braak, 1994)
- NIPALS (Wold, 1966; Dayal & McGregor, 1997)

Input: **xtrn** the $n \times m$ training data set containing X and Y variables in its columns.

yind ny column indices for the responses (Y variables): must relate to columns of `xtrn` and `xtst`

xind nx column indices for the predictors (X variables): must relate to columns of `xtrn` and `xtst`

icmp scalar or nc vector of the number of components (factors, latent variables) of interest (by default a scalar equal the number of predictors)

optn option vector (see below for details)

xtst (optional) $t \times m$ test data set with columns compatible to the training data set

Content of the options vector **1** amount of printed output (=0: no output, =1: print only summary tables, =2: componentwise tables added, =3: parameters added (see `optn[13]`, =4: predicted values)

2 method: =0: PCR: principal components regression =1: PLS: partial least squares

3 version of the PLS algorithm: =0: SIMPLS, =1: Kernel PLS, =2: NIPALS

4 currently not used

5 cross validation: =1: leave-one-out, =2: block cross validation, =3: split cross validation;

6 only for CV: size of left-out part

7 only for CV: fold number

8 ppar: print parameter matrix Beta (`optn[1]`¿=3)

9 pypr: print predicted values (`optn[1]`¿=4)

10 currently not used

11 max number of iterations for NIPALS algorithm

12 convergence criterion for NIPALS algorithm

Output: **rms** contains explained variance, root-mean square error (RMS) and R^2 for each response Y corresponding to training; for $ny > 1$ this is $3 * nc \times ny$ matrix, for $ny = 1$ this is $nc \times 3$ matrix;

parm matrix of parameters for all nc component solutions (this is $nx \times nc * ny$ matrix)

yprd matrix of predicted values for training (this is $n \times nc * ny$ matrix)

vrms contains explained variance, root-mean square error (RMS) and R^2 for each response Y corresponding to cross validation; for $ny > 1$ this is $3 * nc \times ny$ matrix, for $ny = 1$ this is $nc \times 3$ matrix;

trms contains explained variance, root-mean square error (RMS) and R^2 for each response Y corresponding to test data; for $ny > 1$ this is $2 * nc \times ny$ matrix, for $ny = 1$ this is $nc \times 2$ matrix;

tprd matrix of predicted values for test data (this is $t \times nc * ny$ matrix)

Restrictions:

1. No string or complex data are permitted.
2. Some of the input arguments must be compatible.
3. The training data should have no observations with missing values. If there are observations with missing values in the X variables of the test data set those observations are skipped and not scored. The Y variables of the test data set can have missing values.

Relationships: svd(), odr(), reg()

Examples: 1. Middle-Aged Men in Health Fitness Club (Linnerud):

```
print "Middle-Aged Men in Health Fitness Club";
print "SAS/STAT: Linnerud, NC State University";
fit = [ 191  36  50   5  162   60,   189  37  52   2  110   60,
        193  38  58  12  101  101,   162  35  62  12  105   37,
        189  35  46  13  155   58,   182  36  56   4  101   42,
        211  38  56   8  101   38,   167  34  60   6  125   40,
        176  31  74  15  200   40,   154  33  56  17  251  250,
        169  34  50  17  120   38,   166  33  52  13  210  115,
        154  34  64  14  215  105,   247  46  50   1   50   50,
        193  36  46   6   70   31,   202  37  62  12  210  120,
        176  37  54   4   60   25,   157  32  52  11  230   80,
```

```
       156  33  54  15  225   73,   138  33  68   2  110   43 ];

cnam = [ "weight" "waist" "pulse" "chins" "situps" "jumps" ];
fit = cname(fit,cnam);
```

Here, three PCR analyses for one, two, and three factors are computed. Specifying `optn[5]=2` and `optn[7]=10` will also perform a 10-fold block cross classification on the training data:

```
/*--- (Y4,Y5,Y6) = f(X1,X2,X3) ---*/
xind = [ 1:3 ]; yind = [ 4:6 ];
cmp  = [ 1:3 ];
optn = cons(7,1,.);
optn[1] = 3;     /* ipri     */
optn[2] = 0;     /* imet=PCR */
optn[5] = 2;     /* block  CRV */
optn[7] = 10;    /* CRV fold */
< rms,bmat,yprd,cgof > = pls(fit,yind,xind,cmp,optn);
print rms,bmat,yprd,cgof;
```

```
              ******************************
              Principal Components Regression
              ******************************

X Dimension . . . . . .        3
Y Dimension . . . . . .        3
Nobs Training Data. . . .     20
Number Analyses . . . . .      3
Max Latent Variables. . .      3
Cross Validation. . . . . .  Block
Cross Validation Fold . . .   10
Cross Validation Size . . .    2
```

The following componentwise output is shown only for `optn[1]` greater than 1. The parameter matrices are only shown for `optn[1]` greater than 2 and the predicted values are shown only for `optn[1]` greater than 3. For larger problems only the summary output for `optn[1]` equal 1 is recommended which arranged so that it reports about the progress with an increasing number of factors.

```
                 -------------------------
                 Analysis of Training Data
                 -------------------------

              Training Run 1: Number Factors: 1
              ---------------------------------

          Response         RMS          R2       VarExp
          1  chins  4.73825272  0.15430685  0.15430685
          2 situps  52.9032972  0.24741231  0.24741231
          3  jumps  48.7029572  0.05041493  0.05041493
```

Cumulative Fraction of Variance Explained: 0.930148

Parameter Matrix Beta[1]

	chins	situps	jumps
1 weight	-0.08200096	-1.22893624	-0.45465556
2 waist	-0.00930350	-0.13943019	-0.05158340
3 pulse	0.00945183	0.14165311	0.05240579

Predicted Values (Training): 1 Factor

Obs	chins	situps	jumps
1	8.36994987	129.363449	64.3116458
2	8.54355194	131.965197	65.2741851
3	8.26295555	127.759941	63.7184142
4	10.8707031	166.841867	78.1771097
5	8.50544799	131.394139	65.0629171
6	9.16466945	141.273793	68.7179805
7	6.76803463	105.355782	55.4298026
8	10.4510981	160.553310	75.8506038
9	9.87332557	151.894318	72.6471350
10	11.4886068	176.102299	81.6030862
11	10.1925780	156.678906	74.4172348
12	10.4667880	160.788451	75.9375964
13	11.5549179	177.096093	81.9707491
14	3.68486113	59.1487175	38.3351007
15	8.16814065	126.338964	63.1927115
16	7.57205772	117.405557	59.8877207
17	9.62846806	148.224674	71.2895189
18	11.2141001	171.988308	80.0810798
19	11.3057013	173.361120	80.5889636
20	12.9140441	197.465116	89.5064446

Training Run 2: Number Factors: 2

Response	RMS	R2	VarExp
1 chins	4.73822952	0.15431513	0.15431513
2 situps	52.8645640	0.24851392	0.24851392
3 jumps	48.6153642	0.05382755	0.05382755

Cumulative Fraction of Variance Explained: 0.996364

Parameter Matrix Beta[1]

```
               chins      situps       jumps
1 weight -0.08226345 -1.19310728 -0.50633858
2 waist  -0.00929180 -0.14102761 -0.04927912
3 pulse   0.00718609  0.45092109 -0.39371116
```

Predicted Values (Training): 2 Factor

Obs	chins	situps	jumps
1	8.38052302	127.920234	66.3934723
2	8.55013030	131.067264	66.5694480
3	8.25490127	128.859333	62.1325476
4	10.8616879	168.072426	76.4020362
5	8.52559735	128.643792	69.0302732
6	9.16401059	141.363727	68.5882525
7	6.75978705	106.481560	53.8058755
8	10.4452903	161.346075	74.7070447
9	9.83339991	157.344088	64.7858786
10	11.4952625	175.193813	82.9135700
11	10.2089024	154.450650	77.6314792
12	10.4793568	159.072842	78.4123517
13	11.5434594	178.660154	79.7146016
14	3.68085204	59.6959505	37.5457208
15	8.18725176	123.730336	66.9556398
16	7.55256642	120.066080	56.0499349
17	9.63392729	147.479500	72.3644271
18	11.2290196	169.951835	83.0186780
19	11.3163634	171.905757	82.6883152
20	12.8977107	199.694583	86.2904532

Training Run 3: Number Factors: 3

Response	RMS	R2	VarExp
1 chins	4.18720966	0.33957153	0.33957153
2 situps	45.7777205	0.43649217	0.43649217
3 jumps	48.6134272	0.05390294	0.05390294

Cumulative Fraction of Variance Explained: 1

Parameter Matrix Beta[1]

```
               chins      situps       jumps
1 weight  0.07884384  0.72765998 -0.53786495
2 waist  -1.45584256 -17.3872206  0.23378999
3 pulse  -0.01895002  0.13931888 -0.38859670
```

```
Predicted Values (Training): 3 Factor
-------------------------------------

 Obs       chins      situps       jumps
   1  9.66975320  143.290806  66.1411885
   2  8.01832292  124.726904  66.6735150
   3  6.76415560  111.086236  62.4242650
   4  8.61172416  141.247714  76.8423216
   5  11.0437082  158.665431  68.5375152
   6  8.84645852  137.577780  68.6503928
   7  8.22124476  123.905478  53.5198893
   8  10.4996860  161.994597  74.6964003
   9  15.3115079  222.655662  63.7138919
  10  11.0063587  169.364962  83.0092414
  11  10.8468738  162.056728  77.5066374
  12  12.0282848  177.539606  78.1092488
  13  9.39891596  153.092292  80.1342578
  14 -0.47341736  10.1675596  38.3586514
  15  9.90324096  144.188851  66.6198454
  16  8.85379264  135.579672  55.7953037
  17  6.95545296  115.545962  72.8885659
  18  12.7745328  188.377887  82.7162434
  19  11.2019464  170.541644  82.7107049
  20  9.51745700  159.394229  86.9519201
```

The following summary tables are printed if `optn[1]` is greater than zero. They are especially useful for large problems with a larger number of factors:

```
          Summary Table: Variance Explained
           ---------------------------------

Factor  Predictors       chins      situps       jumps
     1  0.93014832  0.15430685  0.24741231  0.05041493
     2  0.99636394  0.15431513  0.24851392  0.05382755
     3  1.00000000  0.33957153  0.43649217  0.05390294


                Summary Table: RMS
                ------------------

Factor       chins      situps       jumps
     1  4.73825272  52.9032972  48.7029572
     2  4.73822952  52.8645640  48.6153642
     3  4.18720966  45.7777205  48.6134272


            Summary Table: Rsquared
            -----------------------

Factor       chins      situps       jumps
     1  0.15430685  0.24741231  0.05041493
```

```
  2  0.15431513  0.24851392  0.05382755
  3  0.33957153  0.43649217  0.05390294
```

The following kind of tables are printed whenever `optn[1]` is greater than 1:

```
   ----------------------------------
   K=10-fold Cross Validation (Block)
   ----------------------------------

Cross Validat. Run 1: Number Factors: 1
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.39722983  1.23734312  0.00514788
2 situps  56.4852091  12.9573007  0.15171970
3  jumps  53.0236913  12.1644680  0.00631660


Cross Validat. Run 2: Number Factors: 2
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.89346569  1.35203921  0.01334978
2 situps  61.1210500  14.0220377  0.06584834
3  jumps  55.5670906  12.7432450  0.04910736


Cross Validat. Run 3: Number Factors: 3
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.22761380  1.19924932  0.11916464
2 situps  61.1556647  14.0280868  0.23756635
3  jumps  61.3835002  14.0459374  0.06953033

Best Number of Factors wrt. Cross Validation: 3
```

The following summary tables are printed if `optn[1]` is greater than zero:

```
                Summary Table: RMS
                ------------------

Factor       chins      situps       jumps
     1  5.39722983  56.4852091  53.0236913
     2  5.89346569  61.1210500  55.5670906
     3  5.22761380  61.1556647  61.3835002


            Summary Table: RMS_Stdev
            ------------------------
```

```
Factor       chins      situps       jumps
     1  1.23734312  12.9573007  12.1644680
     2  1.35203921  14.0220377  12.7432450
     3  1.19924932  14.0280868  14.0459374


          Summary Table: Rsquared
          -----------------------

Factor       chins      situps       jumps
     1  0.00514788  0.15171970  0.00631660
     2  0.01334978  0.06584834  0.04910736
     3  0.11916464  0.23756635  0.06953033


           Time for Training: 0
          Time for Validation: 0
               Total Time: 0
```

Here, three SIMPLS analyses for one, two, and three factors are computed and 10-fold cross classification is done:

```
/*--- (Y4,Y5,Y6) = f(X1,X2,X3) ---*/
xind = [ 1:3 ]; yind = [ 4:6 ];
cmp  = [ 1:3 ];
optn = cons(7,1,.);
optn[1] = 3;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 0;     /* vers=SIMPLS */
optn[5] = 2;     /* block  CRV */
optn[7] = 10;    /* CRV fold */
< rms,bmat,yprd > = pls(fit,yind,xind,cmp,optn);


              ****************************
              SIMPLS Partial Least Squares
              ****************************

X Dimension . . . . . .        3
Y Dimension . . . . . .        3
Nobs Training Data. . . .     20
Number Analyses . . . . .      3
Max Latent Variables. . .      3
Cross Validation. . . . . .  Block
Cross Validation Fold . . .   10
Cross Validation Size . . .    2


               -------------------------
               Analysis of Training Data
               -------------------------
```

```
   Training Run 1: Number Factors: 1
   ---------------------------------

  Response         RMS          R2      VarExp
1  chins  4.73543718  0.15531160  0.15531160
2 situps  52.8558059  0.24876290  0.24876290
3  jumps  48.7045814  0.05035159  0.05035159

Cumulative Fraction of Variance Explained: 0.930137

         Parameter Matrix Beta[1]
         ------------------------

                chins      situps       jumps
1 weight -0.08175662 -1.22463350 -0.45154824
2 waist  -0.01329081 -0.19908318 -0.07340617
3 pulse   0.01001515  0.15001705  0.05531445

   Training Run 2: Number Factors: 2
   ---------------------------------

  Response         RMS          R2      VarExp
1  chins  4.60683948  0.20056615  0.20056615
2 situps  50.6973943  0.30886497  0.30886497
3  jumps  48.6253195  0.05344000  0.05344000

Cumulative Fraction of Variance Explained: 0.980532

         Parameter Matrix Beta[1]
         ------------------------

                chins      situps       jumps
1 weight -0.02479822 -0.44773604 -0.59588273
2 waist  -0.37359647 -5.11355687  0.83962043
3 pulse   0.14825728  2.03560311 -0.29499574

   Training Run 3: Number Factors: 3
   ---------------------------------

  Response         RMS          R2      VarExp
1  chins  4.18720966  0.33957153  0.33957153
2 situps  45.7777205  0.43649217  0.43649217
3  jumps  48.6134272  0.05390294  0.05390294

Cumulative Fraction of Variance Explained: 1

         Parameter Matrix Beta[1]
```

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

                 chins      situps       jumps
1 weight  0.07884384  0.72765998 -0.53786495
2 waist  -1.45584256 -17.3872206  0.23378999
3 pulse  -0.01895002  0.13931888 -0.38859670


   ----------------------------------
   K=10-fold Cross Validation (Block)
   ----------------------------------

Cross Validat. Run 1: Number Factors: 1
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.40880661  1.24001767  0.00483632
2 situps  56.7249597  13.0124935  0.14693190
3  jumps  53.0917782  12.1800881  0.00670106

Cross Validat. Run 2: Number Factors: 2
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.78899458  1.32555253  0.00407966
2 situps  64.1020836  14.6346871  0.09155188
3  jumps  55.1550994  12.6500543  0.03883131

Cross Validat. Run 3: Number Factors: 3
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.22761380  1.19924932  0.11916464
2 situps  61.1556647  14.0280868  0.23756635
3  jumps  61.3835002  14.0459374  0.06953033

Best Number of Factors wrt. Cross Validation: 3
```

Here, three Kernel PLS analyses for one, two, and three factors are computed and 10-fold cross classification is done:

```
xind = [ 1:3 ]; yind = [ 4:6 ];
cmp  = [ 1:3 ];
optn = cons(7,1,.);
optn[1] = 3;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 1;     /* vers=KERNPLS */
optn[7] = 10;    /* CRV fold */
< rms,bmat,yprd > = pls(fit,yind,xind,cmp,optn);
```

```
            ****************************
            Kernel Partial Least Squares
            ****************************

X Dimension . . . . . .                    3
Y Dimension . . . . . .                    3
Nobs Training Data. . . . . .             20
Number Analyses . . . . . .                3
Max Latent Variables. . . . . .            3
Cross Validation. . . . . .            Block
Cross Validation Fold . . . . . .         10
Cross Validation Size . . . . . .          2

            -------------------------
            Analysis of Training Data
            -------------------------

          Training Run 1: Number Factors: 1
          ---------------------------------

       Response         RMS          R2      VarExp
       1  chins  4.73543718  0.15531160  0.15531160
       2 situps  52.8558059  0.24876290  0.24876290
       3  jumps  48.7045814  0.05035159  0.05035159

  Cumulative Fraction of Variance Explained: 0.930137

           Parameter Matrix Beta[1]
           ------------------------

                  chins      situps       jumps
       1 weight -0.08175662 -1.22463350 -0.45154824
       2 waist  -0.01329081 -0.19908318 -0.07340617
       3 pulse   0.01001515  0.15001705  0.05531445

          Training Run 2: Number Factors: 2
          ---------------------------------

       Response         RMS          R2      VarExp
       1  chins  4.60646866  0.20069485  0.20069485
       2 situps  50.6921828  0.30900705  0.30900705
       3  jumps  48.6253694  0.05343805  0.05343805

  Cumulative Fraction of Variance Explained: 0.980488

           Parameter Matrix Beta[1]
           ------------------------
```

```
               chins      situps       jumps
1 weight -0.02466928 -0.44616366 -0.59595879
2 waist  -0.37460975 -5.12619904  0.84060168
3 pulse   0.14837619  2.03677355 -0.29468954


   Training Run 3: Number Factors: 3
   ---------------------------------

Response         RMS          R2      VarExp
1  chins  4.18720966  0.33957153  0.33957153
2 situps  45.7777205  0.43649217  0.43649217
3  jumps  48.6134272  0.05390294  0.05390294

Cumulative Fraction of Variance Explained: 1


         Parameter Matrix Beta[1]
         ------------------------

               chins      situps       jumps
1 weight  0.07884384  0.72765998 -0.53786495
2 waist  -1.45584256 -17.3872206  0.23378999
3 pulse  -0.01895002  0.13931888 -0.38859670


   ----------------------------------
   K=10-fold Cross Validation (Block)
   ----------------------------------

Cross Validat. Run 1: Number Factors: 1
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.40880661  1.24001767  0.00483632
2 situps  56.7249597  13.0124935  0.14693190
3  jumps  53.0917782  12.1800881  0.00670106


Cross Validat. Run 2: Number Factors: 2
---------------------------------------

Response         RMS       RMStd          R2
1  chins  5.79216208  1.32610164  0.00457861
2 situps  64.2758241  14.6707481  0.09188230
3  jumps  55.1611657  12.6515671  0.03840684


Cross Validat. Run 3: Number Factors: 3
---------------------------------------
```

```
Response         RMS       RMStd          R2
1  chins  5.22761380  1.19924932  0.11916464
2 situps  61.1556647  14.0280868  0.23756635
3  jumps  61.3835002  14.0459374  0.06953033

Best Number of Factors wrt. Cross Validation: 3
```

2. The NIR Spectra data consists of 21 observations in the training data and 7 observations in the test data set. Note, this data set has 268 predictor (x) variables and one response (y).

```
  print "NIR Spectra data set: train: nr=21, test: nr=7";
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  nrtrn = nrow(xtrn); nctrn = ncol(xtrn);
  print "nrtrn,nctrn=",nrtrn,nctrn;
  xytrn = xtrn -> ytrn‘; /* attrib(xytrn); */
  nrtst = nrow(xtst); nctst = ncol(xtst);
  print "nrtst,nctst=",nrtst,nctst;
  xytst = xtst -> ytst‘; /* attrib(xytst); */
  /* model: */
  xind = [ 1:268 ]; yind = 269;
  cmp  = [ 1:10 ];


    optn = cons(7,1,.);
    optn[1] = 1;     /* ipri     */
    optn[2] = 0;     /* imet=PCR */
    < rms,bmat,yprd > = pls(xytrn,yind,xind,cmp,optn,xytst);


                  ******************************
                  Principal Components Regression
                  ******************************

    X Dimension . . . . . .      268
    Y Dimension . . . . . .        1
    Nobs Training Data. . . .       21
    Number Analyses . . . . .       10
    Maximum Number Factors. . .     10
    Nobs Test Data. . . . . .        7
    Nullity of X Matrix . . . .      1


                    -------------------------
                    Analysis of Training Data
                    -------------------------

                     Summary Table: Training
                     -----------------------
```

Factor	X_VarExp	Y_VarExp	RMS	R2
1	0.52053381	0.05173290	29.0387409	0.05173290
2	0.98780610	0.98206743	3.99331666	0.98206743
3	0.99512361	0.99465300	2.18055777	0.99465300
4	0.99736034	0.99772736	1.42160205	0.99772736
5	0.99892390	0.99951328	0.65788870	0.99951328
6	0.99977932	0.99989283	0.30871516	0.99989283
7	0.99986612	0.99990603	0.28907223	0.99990603
8	0.99991609	0.99998409	0.11895786	0.99998409
9	0.99993773	0.99998452	0.11732912	0.99998452
10	0.99995687	0.99998877	0.09992306	0.99998877

Analysis of Test Data

Summary Table: Test

Factor	RMS	R2
1	11.2130768	0.09541370
2	2.25304513	0.99759918
3	1.36236296	0.99686036
4	1.31020239	0.99138633
5	0.58783543	0.99897365
6	0.23412796	0.99971658
7	0.22129888	0.99975403
8	0.12103762	0.99991186
9	0.12516400	0.99990802
10	0.12329079	0.99990222

Time for Training: 0
Time for Test Scoring: 0
Total Time: 0

```
optn = cons(7,1,.);
optn[1] = 1;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 0;     /* vers=SIMPLS */
< rms,bmat,yprd > = pls(xytrn,yind,xind,cmp,optn,xytst);
```

****
SIMPLS Partial Least Squares
****

```
X Dimension . . . . . .      268
Y Dimension . . . . . .        1
Nobs Training Data. . . .       21
```

```
Number Analyses . . . . . .       10
Max Latent Variables. . . . . .       10
Nobs Test Data. . . . . .        7
```

Analysis of Training Data

Summary Table: Training

Factor	X_VarExp	Y_VarExp	RMS	R2
1	0.48011844	0.85125566	11.5009222	0.85125566
2	0.59857223	0.87561693	10.5170293	0.87561693
3	0.99369190	0.98847697	3.20107392	0.98847697
4	0.99655974	0.99545686	2.00997362	0.99545686
5	0.99794582	0.99937630	0.74473538	0.99937630
6	0.99875007	0.99937778	0.74385128	0.99937778
7	0.99981720	0.99994341	0.22432455	0.99994341
8	0.99984401	0.99996071	0.18692264	0.99996071
9	0.99988575	0.99996434	0.17808331	0.99996434
10	0.99994267	0.99998420	0.11851991	0.99998420

Summary Table: Test

Factor	RMS	R2
1	5.67164054	0.80686105
2	7.35858007	0.78729865
3	1.00093684	0.99455505
4	1.91764379	0.99496049
5	0.93329894	0.99761332
6	0.96104434	0.99756934
7	0.19970928	0.99991874
8	0.22702995	0.99980339
9	0.48869405	0.99944091
10	0.17409334	0.99987039

```
Time for Training: 0
Time for Test Scoring: 0
      Total Time: 0
```

```
optn = cons(7,1,.);
optn[1] = 1;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 1;     /* vers=KERNPLS */
< rms,bmat,yprd > = pls(xytrn,yind,xind,cmp,optn,xytst);
```

****
Kernel Partial Least Squares
****

X Dimension	268
Y Dimension	1
Nobs Training Data.	21
Number Analyses	10
Max Latent Variables.	10
Nobs Test Data.	7

Analysis of Training Data

Summary Table: Training

Factor	X_VarExp	Y_VarExp	RMS	R2
1	0.47070892	0.98186968	4.01527338	0.98186968
2	0.98577164	0.98293655	3.89534434	0.98293655
3	0.99496572	0.99713269	1.59679791	0.99713269
4	0.99715384	0.99973258	0.48765359	0.99973258
5	0.99871432	0.99988962	0.31329364	0.99988962
6	0.99976525	0.99991628	0.27284392	0.99991628
7	0.99983982	0.99998616	0.11092559	0.99998616
8	0.99991276	0.99999195	0.08458470	0.99999195
9	0.99993031	0.99999712	0.05059942	0.99999712
10	0.99994235	0.99999898	0.03013548	0.99999898

Analysis of Test Data

Summary Table: Test

Factor	RMS	R2
1	2.14266930	0.99616637
2	2.18111393	0.99798733
3	1.28918385	0.99739380
4	0.43425423	0.99914984
5	0.28600057	0.99961430
6	0.18771032	0.99979889
7	0.11053647	0.99992017
8	0.11597007	0.99991728
9	0.09526278	0.99993568
10	0.07784058	0.99995618

```
  Time for Training: 0
Time for Test Scoring: 0
      Total Time: 0
```

3. The following data are the Affymetrix microchip data which have been analyzed here before using the SVM feature selection method. The data set has $N = 36$ observations and $p = 22283$ predictor variables which are the expression rates of genes with a binary response.

```
resp = [ "2.000000" ,
         "3.000000" ];
optn = cons(8,1,.);
optn[1] = 1;   /* itra  */
optn[2] = 2;   /* iresp */
optn[3] = 1;   /* irnam */
optn[4] = 1;   /* icnam */
affym = generead("..\\tdata\\it_vsn.csv",resp,";",optn);
print "nc=", nc = ncol(affym),
      "nr=", nr = nrow(affym);


sopt = [ "ari" "std" "med" ];
mom = univar(affym,sopt);
print "Moments First Var and Response", mom[,1] -> mom[,22284];
xind = [ 1:22283 ]; yind = 22284;
cmp  = [ 1:10 ];


optn = cons(7,1,.);
optn[1] = 1;     /* ipri     */
optn[2] = 0;     /* imet=PCR */
< rms,bmat,yprd > = pls(affym,yind,xind,cmp,optn);
```

For so many variables, PCR does not seem to be the right method:

```
              ******************************
              Principal Components Regression
              ******************************

X Dimension . . . . . . . . . . . . . . . . . .    22283
Y Dimension . . . . . . . . . . . . . . . . . .        1
Nobs Training Data. . . . . . . . . . . . . . .       36
Number Analyses . . . . . . . . . . . . . . . .       10
Maximum Number Factors. . . . . . . . . . . . .       10
Nullity of X Matrix . . . . . . . . . . . . . .        1


                  -------------------------
                  Analysis of Training Data
                  -------------------------

                   Summary Table: Training
```

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

Factor    X_VarExp    Y_VarExp         RMS          R2
     1  0.19820307  1.836e-004  0.49995410  1.836e-004
     2  0.36074379  0.06111601  0.48448013  0.06111601
     3  0.42441135  0.08507732  0.47825795  0.08507732
     4  0.48269463  0.08755827  0.47760908  0.08755827
     5  0.52288901  0.12876049  0.46670106  0.12876049
     6  0.56035112  0.23025267  0.43867623  0.23025267
     7  0.59078340  0.23575957  0.43710423  0.23575957
     8  0.61932870  0.25391582  0.43188082  0.25391582
     9  0.64679220  0.25441412  0.43173658  0.25441412
    10  0.67253257  0.33519443  0.40767805  0.33519443
```

But computing the complete SVD for the 36×22283 matrix is not too time consuming. However, if K=10-fold cross validation would be included the time would increase by about a factor of 10.

```
Time for Training: 51
   Total Time: 51
```

Interesting is, that also the SIMPLS method does not show very great results, but is much faster than PCR:

```
optn = cons(7,1,.);
optn[1] = 1;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 0;     /* vers=SIMPLS */
< rms,bmat,yprd > = pls(affym,yind,xind,cmp,optn);


              ***************************
              SIMPLS Partial Least Squares
              ***************************

X Dimension . . . . . .    22283
Y Dimension . . . . . .        1
Nobs Training Data. . . .     36
Number Analyses . . . . .     10
Maximum Number Factors. . .   10


               -------------------------
               Analysis of Training Data
               -------------------------

                Summary Table: Training
                -----------------------

Factor    X_VarExp    Y_VarExp         RMS          R2
     1  0.11507203  0.02844825  0.49283662  0.02844825
     2  0.19630890  0.04510635  0.48859330  0.04510635
```

```
 3  0.31600470  0.05431764  0.48623100  0.05431764
 4  0.42526394  0.11638961  0.47000276  0.11638961
 5  0.46187095  0.12733664  0.46708226  0.12733664
 6  0.49212236  0.13028685  0.46629206  0.13028685
 7  0.52803950  0.13123804  0.46603700  0.13123804
 8  0.57244211  0.16824757  0.45600231  0.16824757
 9  0.59453153  0.17398956  0.45442558  0.17398956
10  0.62442787  0.19935324  0.44739433  0.19935324


                Time for Training: 9
                    Total Time: 9
```

The kernel PLS method, however, gives an excellent result and is also fast. As told above, including 10-fold cross validation would increase the computer time by a factor of about 10, which would still be acceptable:

```
optn = cons(7,1,.);
optn[1] = 1;     /* ipri     */
optn[2] = 1;     /* imet=PLS */
optn[3] = 1;     /* vers=KERNPLS */
< rms,bmat,yprd > = pls(affym,yind,xind,cmp,optn);


              ****************************
              Kernel Partial Least Squares
              ****************************

X Dimension . . . . . .    22283
Y Dimension . . . . . .        1
Nobs Training Data. . . .     36
Number Analyses . . . . .     10
Maximum Number Factors. . .   10


               -------------------------
               Analysis of Training Data
               -------------------------

                Summary Table: Training
                -----------------------

Factor    X_VarExp    Y_VarExp         RMS          R2
     1  0.13332634  0.36722129  0.39773695  0.36722129
     2  0.19488413  0.78580665  0.23140514  0.78580665
     3  0.25252961  0.92718459  0.13492165  0.92718459
     4  0.41268553  0.95876691  0.10152967  0.95876691
     5  0.46989021  0.98903746  0.05235108  0.98903746
     6  0.49607198  0.99758092  0.02459205  0.99758092
     7  0.52633515  0.99939524  0.01229594  0.99939524
     8  0.56722694  0.99987326  0.00562905  0.99987326
```

```
 9  0.59551017  0.99996958  0.00275784  0.99996958
10  0.62090648  0.99999442  0.00118098  0.99999442


Time for Training: 10
   Total Time: 10
```

Using only 3 factors explains almost 93 % of the variance, using 5 factors explains 99 % of the variance.

3.5 Function `plslv`

Not done yet.

3.6 Function `svdtrip`

< s,v,u > = svdtrip(a,"meth"<,optn>)

< s,v,u > = svdtrip(funa,"meth"<,optn<,funata>>)

Purpose: This function is used to compute the singular triplets of largest singular values s_i of a $m \times n$ matrix $\mathbf{A}$ or an operator function simulating such a matrix. The following methods are based on Michael Berry's (1992) SVDPACK implementation which is available on *Netlib*:

"bls" Block Lanczos method

"las" Single step Lanczos method

"sis" Rutishauser-Ritz iteration method

"tms" Trace minimimization method

In the following we assume $m \geq n$, otherwise the matrix should be transposed and the u and v singular vectors are swapped. There are two versions of each of these methods:

1. the first version is based on the (large) $m \times n$ matrix $\mathbf{A}$: it is time and memory consuming but should be more precise for matrices with bad condition
2. the second version is based on the (small) $n \times n$ matrix $\mathbf{A}^T\mathbf{A}$: which can save memory and computer time, especially for $n \ll m$.

Input: There are two different input forms:

1. < s,u,v > = svdtrip(a,"meth"<,optn>) specifying a real $m \times n$ matrix $\mathbf{A}$
2. < s,u,v > = svdtrip(funa,"meth"<,optn<,funata>>) specifying an operator function `y=funa(itra,x)` for performing the matrix vector multiplications

$$y = \begin{cases} \mathbf{A}x & \text{for} itra = 0 \\ \mathbf{A}^T x & \text{otherwise} \end{cases}$$

and eventually the operator function `funata` for performing the matrix vector multiplication

$$y = \mathbf{A}^T\mathbf{A}x$$

The first input argument should be the name of a matrix $\mathbf{A}$ or a user specified operator function. The second argument is a string `"bls"`, `"las"`, `"sis"`, or `"tms"` specifying the method. The third input argument is an option vector specifying:

1 parameter for printed output (=0: no printed output, =1: only summary output, =2: additional output of singular values, =3: additional iteration history)

2 wanted number of triplets p

3 temporarily unused

4 specifying version of method:
=0: based on $m \times n$ matrix $\mathbf{A}$
=1: based on $n \times n$ matrix $\mathbf{A}^T\mathbf{A}$

5 BLS: maximum size of subspace (should be multiple of block size)
LAS: upper limit of desired number of Lanczos steps (should be larger than desired number of triplets `optn[6]`)
SIS: number of additional vectors to carry
TMS: initial subspace dimension (should be larger than number of triplets `optn[2]`)

6 BLS: initial block size (should be larger than 1 and smaller than dimension of subspace in `optn[5]`)
LAS: upper limit of desired number of singular triplets (should be larger than p)
TMS: shift method (=0: no shift, =1: Ritz shift acceleration (is default for $m \times n$ version), =2: Ritz and Chebyshev polynomial shifting, only for $\mathbf{A}^T\mathbf{A}$ version)

7 maximum number of iterations

8 precision tolerance for terminating iteration (default=1.e-6)

9 temporarily unused

10 TMS: residual norm reduction factor for initiating shifting (default = 1)

11 LAS: left limit of interval containing unwanted eigenvalues SIS, TMS: shift parameter α

12 LAS: right limit of interval containing unwanted eigenvalues

13 number of rows of $\mathbf{A}$ (only needed for operator function)

14 number of columns of $\mathbf{A}$ (only needed for operator function)

15 temporarily unused

If an operator function `funa` is specified with the first input argument and the version based on $\mathbf{A}^T\mathbf{A}$ is being used, then the last input argument can be the name of a user specified function `funata` performing the multiplication $y = \mathbf{A}^T\mathbf{A}x$ for given x. In general this specification of `funata` is not necessary, but can save considerable computer time since if it is not specified, the function `funa` is called twice for each multiplication $y = \mathbf{A}^T\mathbf{A}x$. Note, if you are using an operator function `funa`, the number of rows and columns of $\mathbf{A}$ must be specified with the `optn` vector.

Output: Returned is a number of singular triplets (s_i, v_i, u_i) where s_i are the largest singular values, the v_i are right vectors and the u_i are left vectors.

Restrictions: 1. Matrix $\mathbf{A}$ cannot have missing values and must be real.

2. No complex valued version is implemented yet.

Relationships: arpack(), svd(), eig()

Examples: The following is one of the sparse example matrices which comes with the SVDPACK on *Netlib*: BELLCORE ADI Matrix with 373 rows, 82 columns, and 1343 nonzero entries:

```
print "BLI1: BELLADI Problem (M. Berry): Raw version";
print "BELLCORE ADI TERM-DOCUMENT MATRIX (BELLADI, transposed)";
nr = 374; nc = 82; nzer = 1343;
```

```
  options NOECHO;
#include "..\\tdata\\svd\_belladi.dat"
  options ECHO;
  cind = colini[2:83] - 1;
  a = spmat(nr,nc,rind,cind,valu,"color");
  nr = nrow(a); nc = ncol(a);
  print "BELLADI: nr,nc=",nr,nc;
```

1. Block Lanczos Method:

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  0;      /* iata: no A'A */
optn[ 5] = 50;      /* nsub: maximum subspace dim */
optn[ 6] = 10;      /* nblk: initial block size */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"bls",optn);
```

```
      ************************************************
      Singular Triplets by Block Lanczos Method (Cyclic)
      ************************************************

  No. of Rows of A. . . . . .          374
  No. of Columns of A . . . . . .       82
  Tolerance . . . . . .  1.000e-006
  Maximum No. of Iterations . . . . . .       400
  No. of Iterations Taken . . . . . .          5
  Number of Triplets Sought . . . . . .        10
  Number of Triplets Found. . . . . .          10
  Initial Block Size. . . . . .                10
  Final Block Size. . . . . .                   1
  Maximum Subspace Bound. . . . . .            41
  Final Subspace Bound. . . . . .              50
  Number Multiplications by A . . . . . .     247
  Number Multiplications by A^T . . . . . .   206
  Total Matrix-Vector Multipl.. . . . . .     453
  Int Memory Needed (4b). . . . . .            22
  Dbl Memory Needed (8b). . . . . .         65575
  Time for SVD. . . . . .                       2
```

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 50;      /* nsub: maximum subspace dim */
optn[ 6] = 10;      /* nblk: initial block size */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"bls",optn);
```

```
      ************************************************
      Singular Triplets by Block Lanczos Method (A^TA)
      ************************************************

 No. of Rows of A. . . . . .          374
 No. of Columns of A . . . . . .       82
 Tolerance . . . . . .          1.000e-006
 Maximum No. of Iterations . . . . . .  400
 No. of Iterations Taken . . . . . .     5
 Number of Triplets Sought . . . . . .  10
 Number of Triplets Found. . . . . .    10
 Initial Block Size. . . . . .          10
 Final Block Size. . . . . .             0
 Maximum Subspace Bound. . . . . .      40
 Final Subspace Bound. . . . . .        50
 Number Multiplications by A . . . . . .  299
 Number Multiplications by A^T . . . . . .  289
 Total Matrix-Vector Multipl.. . . . . .  588
 Int Memory Needed (4b). . . . . .       22
 Dbl Memory Needed (8b). . . . . .    20871
 Time for SVD. . . . . .                  1
```

2. Single Vector Lanczos Method:

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  0;      /* iata */
optn[ 5] = 82;      /* lanmax */
optn[ 6] = 10;      /* maxprs */
optn[ 8] = 1.e-6;   /* kappa: tolerance */
optn[11] = -1.e-30; /* endl */
optn[12] =  1.e-30; /* endr */
< d,v,u > = svdtrip(a,"las",optn);
```

```
   ********************************************************
   Singular Values by Single Vector Lanczos Method (Cyclic)
   ********************************************************

 No. of Rows of A. . . . . .          374
 No. of Columns of A . . . . . .       82
 No. of Equations. . . . . .          456
 Limit of Lanczos Steps. . . . . .     82
 Limit of Eigenpairs . . . . . .       10
 Tolerance . . . . . .          1.000e-006
 Left end of Interval. . . . . .  -1.000e-030
 Right end of Interval . . . . . .  1.000e-030
 Stabilized Ritz Values. . . . . .        22
```

```
  Number Multiplications by A . . . . . .          88
  Number Multiplications by A^T . . . . . .        88
  Total Matrix-Vector Multipl.. . . . . .         176
  Int Memory Needed (4b). . . . . .                 0
  Dbl Memory Needed (8b). . . . . .             50829
  Time for SVD. . . . . .                           1
```

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 82;      /* lanmax */
optn[ 6] = 10;      /* maxprs */
optn[ 8] = 1.e-6;   /* kappa: tolerance */
optn[11] = -1.e-30; /* endl */
optn[12] =  1.e-30; /* endr */
< d,v,u > = svdtrip(a,"las",optn);
```

```
    ******************************************************
    Singular Values by Single Vector Lanczos Method (A^TA)
    ******************************************************

  No. of Rows of A. . . . . .                     374
  No. of Columns of A . . . . . .                  82
  No. of Equations. . . . . .                      82
  Limit of Lanczos Steps. . . . . .                82
  Limit of Eigenpairs . . . . . .                  10
  Tolerance . . . . . .                    1.000e-006
  Left end of Interval. . . . . .         -1.000e-030
  Right end of Interval . . . . . .        1.000e-030
  Stabilized Ritz Values. . . . . .                17
  Number Multiplications by A . . . . . .          68
  Number Multiplications by A^T . . . . . .        58
  Total Matrix-Vector Multipl.. . . . . .         126
  Int Memory Needed (4b). . . . . .                 0
  Dbl Memory Needed (8b). . . . . .             16050
  Time for SVD. . . . . .                           0
```

3. Rutishauser Ritz Subspace Iteration:

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  0;      /* iata */
optn[ 5] = 10;      /* nxtr: extra vectors to carry */
optn[ 7] = 500;     /* itmx: max number iterations */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"sis",optn);
```

```
        ******************************************
        Singular Values by Subspace Iteration (Cyclic)
        ******************************************

  No. of Rows of A. . . . . .          374
  No. of Columns of A . . . . . .       82
  No. of Equations. . . . . .          456
  Shift Parameter . . . . . .           78
  No. Desired Eigenpairs. . . . . .     10
  Number of Eigenpairs Found. . . . . .  10
  Initial Subspace Dim. . . . . .       20
  Maximum No. of Iterations . . . . . .  500
  No. of Iterations Taken . . . . . .   152
  Tolerance . . . . . .          1.000e-006
  Number Multiplications by A . . . . . .  2611
  Number Multiplications by A^T . . . . . .  2611
  Total Matrix-Vector Multipl.. . . . . .  5222
  Int Memory Needed (4b). . . . . .        0
  Dbl Memory Needed (8b). . . . . .    19632
  Time for SVD. . . . . .               2


optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 10;      /* nxtr: extra vectors to carry */
optn[ 7] = 500;     /* itmx: max number iterations */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"sis",optn);


        ******************************************
        Singular Values by Subspace Iteration (A^TA)
        ******************************************

  No. of Rows of A. . . . . .          374
  No. of Columns of A . . . . . .       82
  No. of Equations. . . . . .           82
  Shift Parameter . . . . . .            .
  No. Desired Eigenpairs. . . . . .     10
  Number of Eigenpairs Found. . . . . .  10
  Initial Subspace Dim. . . . . .       20
  Maximum No. of Iterations . . . . . .  500
  No. of Iterations Taken . . . . . .    35
  Tolerance . . . . . .          1.000e-006
  Number Multiplications by A . . . . . .  619
  Number Multiplications by A^T . . . . . .  609
  Total Matrix-Vector Multipl.. . . . . .  1228
  Int Memory Needed (4b). . . . . .        0
  Dbl Memory Needed (8b). . . . . .     4590
```

```
  Time for SVD. . . . . .          1
```

4. Trace Minimization Method:

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  0;      /* iata */
optn[ 5] = 12;      /* nsub: initial subspace dim */
optn[ 6] =  1;      /* job: acceleration method: Ritz */
optm[ 7] = 400;     /* itmx: max number iterations */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
optn[ 9] = 1.0;     /* cred: residual norm reduction factor */
< d,v,u > = svdtrip(a,"tms",optn);
```

```
    ********************************************************
    Singular Triplets by Trace Minimization Method (Cyclic)
    ********************************************************

  Ritz Shift Acceleration . . . . . .          78
  No. of Rows of A. . . . . .                 374
  No. of Columns of A . . . . . .              82
  No. of Equations. . . . . .                 456
  No. Desired Eigenpairs. . . . . .            10
  Initial Subspace Dim. . . . . .              12
  Maximum No. of Iterations . . . . . .       400
  No. of Iterations Taken . . . . . .          24
  Final Res. Tolerance. . . . . .    1.000e-006
  Residual Reduction Tol. . . . . .  1.000e+000
  Number Multiplications by A . . . . . .    1179
  Number Multiplications by A^T . . . . . .  1427
  Total Matrix-Vector Multipl.. . . . . .    2606
  Int Memory Needed (4b). . . . . .            36
  Dbl Memory Needed (8b). . . . . .         20734
  Time for SVD. . . . . .           3.00e+000
  Time for Gram-Schmidt . . . . . .          0   0%
  Time for Spectral Dec.. . . . . .          0   0%
  Time for Converg. Crit. . . . . .          0   0%
  Time for Conjug. Grad.. . . . . .          2 100%
```

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 12;      /* nsub: initial subspace dim */
optn[ 6] =  1;      /* job: acceleration method: Shift */
optm[ 7] = 400;     /* itmx: max number iterations */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
optn[ 9] = 1.0;     /* cred: residual norm reduction factor */
```

```
< d,v,u > = svdtrip(a,"tms",optn);


     ****************************************************
     Singular Triplets by Trace Minimization Method (A^TA)
     ****************************************************

  Ritz Shift Acceleration . . . . . .          78
  No. of Rows of A. . . . . . . . . . .       374
  No. of Columns of A . . . . . . . . .        82
  No. of Equations. . . . . . . . . . .        82
  No. Desired Eigenpairs. . . . . . . .        10
  Initial Subspace Dim. . . . . . . . .        12
  Maximum No. of Iterations . . . . . .       400
  No. of Iterations Taken . . . . . .          17
  Final Res. Tolerance. . . . . . . .  1.000e-006
  Residual Reduction Tol. . . . . . .  1.000e+000
  Number Multiplications by A . . . . .       932
  Number Multiplications by A^T . . . . .     922
  Total Matrix-Vector Multipl.. . . . .      1854
  Int Memory Needed (4b). . . . . . . .        61
  Dbl Memory Needed (8b). . . . . . . .      4278
  Time for SVD. . . . . . . . . . . . .  0.00e+000
  Time for Gram-Schmidt . . . . . .          0   0%
  Time for Spectral Dec.. . . . . .          0   0%
  Time for Converg. Crit. . . . . .          0   0%
  Time for Conjug. Grad.. . . . . .          0   0%


optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] = 10;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 12;      /* nsub: initial subspace dim */
optn[ 6] =  2;      /* job: acceleration method: Cheby */
optm[ 7] = 400;     /* itmx: max number iterations */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
optn[ 9] = 1.0;     /* cred: residual norm reduction factor */
< d,v,u > = svdtrip(a,"tms",optn);


     ****************************************************
     Singular Triplets by Trace Minimization Method (A^TA)
     ****************************************************

  Chebyshev Polynomial. . . . . .              78
  No. of Rows of A. . . . . . . . . . .       374
  No. of Columns of A . . . . . . . . .        82
  No. of Equations. . . . . . . . . . .        82
  No. Desired Eigenpairs. . . . . . . .        10
  Initial Subspace Dim. . . . . . . . .        12
  Maximum No. of Iterations . . . . . .       400
```

```
No. of Iterations Taken . . . . . .          18
Final Res. Tolerance. . . . . .    1.000e-006
Residual Reduction Tol. . . . . .  1.000e+000
Number Multiplications by A . . . . . .     907
Number Multiplications by A^T . . . . . .   897
Total Matrix-Vector Multipl.. . . . . .    1804
Int Memory Needed (4b). . . . . .            61
Dbl Memory Needed (8b). . . . . .          4278
Time for SVD. . . . . .           0.00e+000
Time for Gram-Schmidt . . . . . .        0   0%
Time for Spectral Dec.. . . . . .        0   0%
Time for Converg. Crit. . . . . .        0   0%
Time for Conjug. Grad.. . . . . .        0   0%
```

5. Use the Arpack Function: The amount of memory needed for the `arpack` function includes the data matrix, different from the `svdtrip` function which lists only the needed work space.

```
optn = [ "which"    "lm",
         "print"      1 ];
< val,vec,uvec,res,scl > = arpack(a,"svd",10,20,optn);


      Standard Singular Value Problem (input matrix)
      **********************************************
  Size of Matrix . . . . . .                82
  Number of Ritz Values Requested. . . . . .  10
  Number of Arnoldi Vectors Generated. . . . .  20
  What Portion of the Spectrum . . . . . .    LM
  Number of Singular Values Requested. . . . . .  10
  Number of Implicit Arnoldi Update Iterations .   9
  Number of OP*x . . . . . .                 67
  Number of B*x. . . . . .                    0
  Number of Reorthogonalizations . . . . . .  67
  Convergence Criterion. . . . . .  1.1102e-016
  Computer Time (in seconds) . . . . . .      1
  Dbl Memory Needed (4b) . . . . . .       8399
  Int Memory Needed (8b) . . . . . .       1777


  dtim = time("clock");
  < d,v,u > = svd(a,"eco");
  dtim = time("clock") - dtim;
  print "Time for SVD=",dtim;
  print "Singular Values", dia2vec(d);


  Time for SVD= 1.4840
```

From the results we see that the methods using the "small" $n \times n$ matrix $\mathbf{A}^T\mathbf{A}$ need much less memory than those which work on the large $m \times n$ matrix $\mathbf{A}$. Among those methods the block and single step Lanczos methods need more memory and less matrix multiplications than the Rutishauser-Ritz and Trace

minimization methods. For much larger matrices we observed that the block and single step Lanczos methods can be much faster than the Rutishauser-Ritz and Trace minimization methods.

Experiments with much larger matrices show, that the methods in the `svdtrip()` functions compare better with the `arpack()` method if many (in the 100's and 1000's) are wanted. For only a few singular triplets the `arpack` method seems to be comparable with the `"las"` method but much faster than the other three methods.

4 Illustration

4.1 Some `svdtrip` and `arpack` Benchmarks

1. Bellcore ADI Matrix with 373 rows, 82 columns, and 1343 nonzero entries: The summary results are shown in the doc of the function.
2. Seismic Tomography Jacobian (AMOCO) with 1436 rows, 330 columns, and 35210 nonzero entries: This is the matrix with most columns and most nonzero entries. The computer time will probably be largest due to the more time consuming matrix vector multiplications.
3. Information Retrieval (APPLE1) with 3206 rows, 44 columns, and 7722 nonzero entries: Even though there is a large number of rows, the computer time and memory need will be relatively small for the $\mathbf{A}^T\mathbf{A}$ algorithms.
4. Information Retrieval (APPLE2): with 1472 rows, 294 columns, and 13442 nonzero entries.
5. Large 10000×100 and 40000×100 dense randomly generated matrix.

The following four tables show the amounts of computer time and storage used for computing the 10 largest singular values and corresponding left and right vectors of the four sparse data sets by the eight methods of the `svdtrip` function and the Arnoldi method used by the `arpack` function. The listed amount of memory needed for the `arpack` function includes the data matrix, different from the `svdtrip` function which lists only the needed work space.

1. Bellcore ADI Matrix with 373 rows, 82 columns, and 1343 nonzero entries,

Method	Memory	Multiplications	Time
Arpack	8399	67	1
BLS1	65575	453	2
LAS1	50829	176	1
SIS1	19632	5222	2
TMS1	20734	2606	3
BLS2	20871	588	1
LAS2	16050	126	0
SIS2	4590	1228	1
TMS2	4278	1854	1

2. Seismic Tomography Jacobian (AMOCO) with 1436 rows, 330 columns, and 35210 nonzero entries:

Method	Memory	Multiplications	Time
Arpack	60962	77	2
BLS1	234235	619	11
LAS1	172725	176	4
SIS1	74652	5692	48
TMS1	122364	3163	42
BLS2	56583	678	6
LAS2	42052	138	1
SIS2	16882	1228	10
TMS2	24716	2946	32

3. Information Retrieval (APPLE1) with 3206 rows, 44 columns, and 7722 nonzero entries:

Method	Memory	Multiplications	Time
Arpack	47850	75	1
BLS1	424921	363	7
LAS1	190613	110	1
SIS1	136980	8708	26
TMS1	224760	3777	46
BLS2	14251	416	1
LAS2	14286	120	0
SIS2	8696	1450	3
TMS2	6752	1812	3

4. Information Retrieval (APPLE2): with 1472 rows, 294 columns, and 13442 nonzero entries.

Method	Memory	Multiplications	Time
Arpack	38762	91	1
BLS1	236755	623	8
LAS1	172725	176	1
SIS1	74652	5592	17
TMS1	122364	4109	32
BLS2	51399	680	3
LAS2	38776	150	1
SIS2	15478	1666	4
TMS2	22268	2608	9

5. Dense 10000×100 randomly generated matrix: We require only 3 singular values and run only the following three methods: `svdtrip()` with "bls", and "las" with the $\mathbf{A}^T\mathbf{A}$ version and `arpack`:

```
srand(123);
nr = 10000; nc = 100;
a = rand(nr,nc);
```

Calling `arpack`:

```
optn = [ "which"    "lm",
         "print"      1 ];
< val,vec,uvec,res,scl > = arpack(a,"svd",3,20,optn);
```

```
        Standard Singular Value Problem (input matrix)
        **********************************************
    Size of Matrix . . . . . .      100
    Number of Ritz Values Requested. . . . . .        3
    Number of Arnoldi Vectors Generated. . . . . .       20
    What Portion of the Spectrum . . . . . .       LM
    Number of Singular Values Requested. . . . . .      3
    Number of Implicit Arnoldi Update Iterations .        6
    Number of OP*x . . . . . .       99
    Number of B*x. . . . . .        0
    Number of Reorthogonalizations . . . . . .       98
    Convergence Criterion. . . . . .  1.1102e-016
    Computer Time (in seconds) . . . . . .       31
    Dbl Memory Needed (4b) . . . . . .  1053000
    Int Memory Needed (8b) . . . . . .       60

              Singular Values and Direct Residuals
              ************************************

                    N SingularValue    Residual
                    1  31.622762560  7.333e-014
                    2  31.712483012  3.950e-015
                    3  500.48733627  6.749e-014

optn = cons(12,1,.);
optn[ 1] =  3;      /* ipri */
optn[ 2] =  3;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 40;      /* nsub: maximum subspace dim */
optn[ 6] = 10;      /* nblk: initial block size */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"bls",optn);

        ***********************************************
        Singular Triplets by Block Lanczos Method (A^TA)
        ***********************************************

  No. of Rows of A. . . . . .      10000
  No. of Columns of A . . . . . .        100
  Tolerance . . . . . .   1.000e-006
  Maximum No. of Iterations . . . . . .        400
  No. of Iterations Taken . . . . . .          9
  Number of Triplets Sought . . . . . .          3
  Number of Triplets Found. . . . . .          3
  Initial Block Size. . . . . .         10
  Final Block Size. . . . . .          7
  Maximum Subspace Bound. . . . . .         37
  Final Subspace Bound. . . . . .         40
  Number Multiplications by A . . . . . .        374
  Number Multiplications by A^T . . . . . .        371
```

```
  Total Matrix-Vector Multipl.. . . . . .        745
  Int Memory Needed (4b). . . . . .               22
  Dbl Memory Needed (8b). . . . . .            25433
  Time for SVD. . . . . .                    1.2e+002

               Singular Values and Residual Norms
               **********************************

                   1 5.00487336e+002   1.6376e-012
                   2 3.17124830e+001   2.0179e-009
                   3 3.16227626e+001   9.6398e-010

optn = cons(12,1,.);
optn[ 1] =  3;      /* ipri */
optn[ 2] =  3;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 40;      /* lanmax */
optn[ 6] = 10;      /* maxprs */
optn[ 8] = 1.e-6;   /* kappa: tolerance */
optn[11] = -1.e-30; /* endl */
optn[12] =  1.e-30; /* endr */
< d,v,u > = svdtrip(a,"las",optn);

     *****************************************************
     Singular Values by Single Vector Lanczos Method (A^TA)
     *****************************************************

  No. of Rows of A. . . . . .                    10000
  No. of Columns of A . . . . . .                  100
  No. of Equations. . . . . .                      100
  Limit of Lanczos Steps. . . . . .                 40
  Limit of Eigenpairs . . . . . .                   10
  Tolerance . . . . . .                      1.000e-006
  Left end of Interval. . . . . .           -1.000e-030
  Right end of Interval . . . . . .          1.000e-030
  Stabilized Ritz Values. . . . . .                  6
  Number Multiplications by A . . . . . .           47
  Number Multiplications by A^T . . . . . .         44
  Total Matrix-Vector Multipl.. . . . . .           91
  Int Memory Needed (4b). . . . . .                  0
  Dbl Memory Needed (8b). . . . . .              37164
  Time for SVD. . . . . .                           17

               Singular Values and Residual Norms
               **********************************

                   1 5.00487336e+002   2.0447e-012
                   2 3.17124830e+001   9.5038e-007
                   3 3.16227626e+001   7.4765e-007
```

6. Dense 40000×100 randomly generated matrix: We require only one singular value and run only the following three methods: `svdtrip()` with "bls", and "las" with the $\mathbf{A}^T\mathbf{A}$ version and `arpack`:

```
srand(123);
nr = 40000; nc = 100;
a = rand(nr,nc);
```

Calling `arpack`:

```
optn = [ "which"    "lm",
         "print"      1 ];
< val,vec,uvec,res,scl > = arpack(a,"svd",1,10,optn);

         Standard Singular Value Problem (input matrix)
         **********************************************
    Size of Matrix . . . . . .      100
    Number of Ritz Values Requested. . . . . .        1
    Number of Arnoldi Vectors Generated. . . . . .       10
    What Portion of the Spectrum . . . . . .       LM
    Number of Singular Values Requested. . . . . .      1
    Number of Implicit Arnoldi Update Iterations .        1
    Number of OP*x . . . . . .       10
    Number of B*x. . . . . .        0
    Number of Reorthogonalizations . . . . . .        9
    Convergence Criterion. . . . . .  1.1102e-016
    Computer Time (in seconds) . . . . . .       14
    Dbl Memory Needed (4b) . . . . . .  4121600
    Int Memory Needed (8b) . . . . . .       30

              Singular Values and Direct Residuals
              ************************************

                    N SingularValue    Residual
                    1  1001.8844924  2.405e-012

optn = cons(12,1,.);
optn[ 1] =  3;      /* ipri */
optn[ 2] =  1;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 10;      /* nsub: maximum subspace dim */
optn[ 6] =  5;      /* nblk: initial block size */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
< d,v,u > = svdtrip(a,"bls",optn);

        ***********************************************
        Singular Triplets by Block Lanczos Method (A^TA)
        ***********************************************

  No. of Rows of A. . . . . .      40000
  No. of Columns of A . . . . . .        100
```

```
  Tolerance . . . . . .   1.000e-006
  Maximum No. of Iterations . . . . . .        400
  No. of Iterations Taken . . . . . .            4
  Number of Triplets Sought . . . . . .          1
  Number of Triplets Found. . . . . .            1
  Initial Block Size. . . . . .                  5
  Final Block Size. . . . . .                    4
  Maximum Subspace Bound. . . . . .              9
  Final Subspace Bound. . . . . .               10
  Number Multiplications by A . . . . . .       52
  Number Multiplications by A^T . . . . . .     51
  Total Matrix-Vector Multipl.. . . . . .      103
  Int Memory Needed (4b). . . . . .             17
  Dbl Memory Needed (8b). . . . . .          43963
  Time for SVD. . . . . .                        72

               Singular Values and Residual Norms
               **********************************

                   1 1.00188449e+003   2.0412e-012

optn = cons(12,1,.);
optn[ 1] =  3;      /* ipri */
optn[ 2] =  3;      /* ntri: number values */
optn[ 4] =  1;      /* iata */
optn[ 5] = 10;      /* lanmax */
optn[ 6] =  4;      /* maxprs */
optn[ 8] = 1.e-6;   /* kappa: tolerance */
optn[11] = -1.e-30; /* endl */
optn[12] =  1.e-30; /* endr */
< d,v,u > = svdtrip(a,"las",optn);

     ******************************************************
     Singular Values by Single Vector Lanczos Method (A^TA)
     ******************************************************

  No. of Rows of A. . . . . .          40000
  No. of Columns of A . . . . . .        100
  No. of Equations. . . . . .            100
  Limit of Lanczos Steps. . . . . .       10
  Limit of Eigenpairs . . . . . .          4
  Tolerance . . . . . .   1.000e-006
  Left end of Interval. . . . . .  -1.000e-030
  Right end of Interval . . . . . .   1.000e-030
  Stabilized Ritz Values. . . . . .        1
  Number Multiplications by A . . . . . .       13
  Number Multiplications by A^T . . . . . .     12
  Total Matrix-Vector Multipl.. . . . . .       25
  Int Memory Needed (4b). . . . . .              0
  Dbl Memory Needed (8b). . . . . .         122544
  Time for SVD. . . . . .                       18
```

```
Singular Values and Residual Norms
**********************************

    1 1.00188449e+003   7.4378e-012
```

4.2 Comparing Anderson-Darling and Marsaglia-Tsang-Wang Version of K-S Test

For a small study we selected the following 4 uniform random generators in CMAT:

RANDUNI The RANDUNI option specifies that the uniform random generator developed by Moore from the RAND Corporation is used. The generator is described at Fishman, 1996, p.605. In SAS Language it is known as RANUNI. This is the default for uniform random numbers.

RANDKISS The RANDKISS option specifies that the uniform random generator KISS ("Keep It Simple Stupid") by Marsaglia & Tsang (2002) is used.

RANDLECU The RANDLECU option specifies that the uniform random generator by L'Ecuyer (1999) is used.

RANDXORWOW The uniform RNG XORWOW by Marsaglia (2003) is used. This is probably the best choice available. Period: $2^{192} - 2^{32}$

for generating 5 sets of uniformly generated random samples of size $n = 50, 100, 400, 1000, 5000$. For each of those samples we computed the p values using the Anderson-Darling and the Marsaglia-Tsang-Wang methods. For each sample we averaged the two p values over the 5 sets and computed the sum-of-squares difference.
The following is the CMAT input for sample size $nr = 100$. The $nr \times nc$ matrix `mu` contains the 5 sets in its columns. The options statement can be used to switch from one random generator to the other. The `kstest` has the data matrix in its first argument and now has two string arguments specifying the distribution and the version of the K-S test.

```
nr = 100; nc = 5;
print "\n *** nr,nc=",nr,nc," ***\n";
options RANDUNI;
mu1 = rand(nr,nc,'g',"duni");
p11 = kstest(mu1,"uni","and");
print "AND: Prob for RANDUNI:",p11;
p12 = kstest(mu1,"uni","mtw");
print "MTW: Prob for RANDUNI:",p12;
print "RANDUNI:", p11[+]/nc, p12[+]/nc, ssq(p11 - p12);
```

```
AND: Prob for RANDUNI:
   |         1
---------------
 1 |   0.33038
 2 |   0.42575
 3 |   0.26321
 4 |   0.51741
 5 |   0.77500
```

```
MTW: Prob for RANDUNI:
   |         1
---------------
 1 |   0.34837
 2 |   0.40739
 3 |   0.27564
 4 |   0.45955
 5 |   0.86244
```

```
RANDUNI: 0.4623 0.4707 0.01181
```

```
 options RANDKIS;
 mu2 = rand(nr,nc,'g',"duni");
 p21 = kstest(mu2,"uni","and");
 print "AND: Prob for RANDKIS:",p21;
 p22 = kstest(mu2,"uni","mtw");
 print "MTW: Prob for RANDKIS:",p22;
 print "RANDKIS:", p21[+]/nc, p22[+]/nc, ssq(p21 - p22);
```

```
AND: Prob for RANDKIS:                    MTW: Prob for RANDKIS:
   |         1                               |         1
-------------                             -------------
 1 |   0.82105                              1 |   0.85801
 2 |   0.84471                              2 |   0.86250
 3 |   0.03047                              3 |   0.02118
 4 |   0.49042                              4 |   0.64616
 5 |   0.59501                              5 |   0.79719
```

```
RANDKIS: 0.5563 0.6370 0.06690
```

```
 options RANDLECU;
 mu3 = rand(nr,nc,'g',"duni");
 p31 = kstest(mu3,"uni","and");
 print "AND: Prob for RANDLECU:",p31;
 p32 = kstest(mu3,"uni","mtw");
 print "MTW: Prob for RANDLECU:",p32;
 print "RANDLECU:", p31[+]/nc, p32[+]/nc, ssq(p31 - p32);
```

```
AND: Prob for RANDLECU:                   MTW: Prob for RANDLECU:
   |         1                               |         1
-------------                             -------------
 1 |   0.96864                              1 |   0.97155
 2 |   0.97750                              2 |   0.99204
 3 |   0.51448                              3 |   0.77799
 4 |   0.65809                              4 |   0.82707
 5 |   0.87049                              5 |   0.86039
```

```
RANDLECU: 0.7978 0.8858 0.09832
```

```
 options RANDXORWOW;
 mu4 = rand(nr,nc,'g',"duni");
 p41 = kstest(mu4,"uni","and");
 print "AND: Prob for RANDXORWOW:",p41;
 p42 = kstest(mu4,"uni","mtw");
 print "MTW: Prob for RANXORWOW:",p42;
```

```
 res = ssq(p41 - p42);
 print "RANDXORWOW:", p41[+]/nc, p42[+]/nc, ssq(p41 - p42);
 options RANDUNI;
```

```
AND: Prob for RANDXORWOW:
   |         1
-------------
 1 |   0.47622
 2 |   0.47891
 3 |   0.22874
 4 |   0.91175
 5 |   0.14746
```

```
MTW: Prob for RANXORWOW:
   |         1
-------------
 1 |   0.27661
 2 |   0.61490
 3 |   0.27463
 4 |   0.88084
 5 |   0.23088
```

```
RANDXORWOW: 0.4486 0.4556 0.06836
```

The following are the results for the sample with $nr = 400$:

```
RANDUNI: 0.5513 0.5477 0.02574
RANDKIS: 0.6030 0.6378 0.03898
RANDLECU: 0.4662 0.4979 0.06441
RANDXORWOW: 0.6620 0.6840 0.05758
```

The following are the results for the sample with $nr = 1000$:

```
RANDUNI: 0.7256 0.7084 0.04600
RANDKIS: 0.5107 0.5119 0.2571
RANDLECU: 0.4424 0.4483 0.1074
RANDXORWOW: 0.3586 0.3754 0.1832
```

The following are the results for the sample with $nr = 5000$:

```
RANDUNI: 0.7158 0.6947 0.05980
RANDKIS: 0.3819 0.3607 0.06410
RANDLECU: 0.5302 0.5942 0.06783
RANDXORWOW: 0.2517 0.3324 0.08743
```

We were surprised that there were such considerable differences between the p values of the two test versions.