CMAT© Newsletter: June 2012

Wolfgang M. Hartmann

June 2012

Contents

1 General Remarks

The most important work was that the string functions `str...()`, were extended to permit data objects like vectors, matrices, and tensors. But even with string arguments, a number of bugs have been found and fixed.
A number of changes were made to `encrypt()` and `decrypt()` functions. In addition, two more functions `encryp2()` and `decryp2()` for the encryption and decryption of scalar or vector string objects were implemented.
The function `mixregv()` for mixed effects location scale regression agrees widely with the functionality of Don Hedeker's (2012) Fortran90 program `MIXREGLS`. It is inteded for a faster solution of a special class of problems where usually SAS PROC NLMIXED is used for.
The function `recupar()` is very much related to the SAS TREEDISC Macro by Warren Sarle (see also Michael Friendly's website of SAS software).
The functions `insert()` and `remove()` are implemented so that they widely agree with those in SAS/IML, which explains their (somewhat awful) argument designs. They were intended for easier translation of SAS/IML programs into the CMAT language. As in SAS/IML these functions are designed for vectors and matrices, but later, we may extend these functions to tensors.

1.1 New Functions

csvread read "comma-separated-values" (CSV) files

recupar recursive partitioning algorithm for tree regression

encryp2 encryption of string objects

decryp2 decryption of string objects

mixregv mixed-effects location scale analysis

insert inserting subobjects into matrices and vectors

remove removing entries from matrices and vectors

1.2 Fixed Bugs

A number of bugs were fixed. Among others some crash of the `quantile()` function was fixed. Also some bugs concerning tensors with string entries.

2 Modifications of Features

2.1 Modifications to `mvntest()` Functions

The computation of the p values for the three Mardia tests is for the most important applications (for data sets the number observations $10 \leq N \leq 200$ and the number of variables $2 \leq n \leq 10$) and the $W_{min}(5)$ test by Wang and Hwang (2011) now based on tables of critical values. We now obtain slightly different p values which make more sense in some applications. The new results (for $\alpha = 0.05$) for the first $N = 50$ observations of the Iris data are the following:

```
Multivariate Normality Test: Mardia MV Kurtosis Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 accepted: Probability=0.0543032 Statistics=26.5377


Multivariate Normality Test: Mardia MV Skewness Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
     H0 accepted: Probability=0.127934 Statistics=3.07972


Multivariate Normality Test: Mardia-Foster Omnibus Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
     H0 accepted: Probability=0.154825 Statistics=3.46194


Multivariate Normality Test: Wang-Hwang Wmin(5) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 accepted: Probability=0.0746763 Statistics=0.941232
```

The new Szekely-Rizzo test (2005) was added.

```
Multivariate Normality Test: Szekely-Rizzo (2005) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
     H0 rejected: Probability=0.0288179 Statistics=1.2034
```

3 Extensions to the Language

4 Extensions to Various Functions

4.1 Extensions of `atof()` and `atoi()` Functions

The `atof()` and `atoi()` functions have been extended for vector, matrix, and tensor arguments.

---

d = atof(s) | i = atoi(s)

Purpose: atof(s) converts the string `s` to a double `d`.

atoi(s) converts the string `s` to an integer `i`.

The function was extended to vector, matrix, and tensor argument. The result has the same dimension as the input argument.

Input: The only argument must be either a scalar string `s` describing a number, or a vector, matrix, or tensor of strings.

Output: The only result is either a scalar integer or real number, or a vector, matrix, or tensor of values. The result has the same dimension as the input argument.

Restrictions: 1. A missing value for v is returned if the input s contains other than string argument.

Relationships: byte()

Examples: 1. Scalar string input:

```
d1 = atof("3.1415926535");
print "ATOF1=", d1;
d2 = atof(" . ");
print "ATOF2=", d2;
```

```
ATOF1= 3.1416
ATOF2= .
```

```
i1 = atoi("12321");
print "ATOI1=", i1;
i2 = atoi("12.21");
print "ATOI2=", i2;
```

```
ATOI1= 12321
ATOI2= 12
```

2. Vector string input:

```
vec1 = [ "123" . "456" 7. " 8.9" ];
print "ATOIvec=",atoi(vec1);
vec2 = [ "3.1415" . "3.15e-01" 100. "21" ];
print "ATOFvec=",atof(vec2);
```

```
ATOIvec=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    123.00      .       456.00      .       8.0000

ATOFvec=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    3.1415     .       0.31500     .        21.000
```

3. Matrix string input:

```
mat1 = vec1 |> vec1;
print "ATOImat=",atoi(mat1);
mat2 = vec2 |> vec2;
print "ATOFmat=",atof(mat2);
```

```
ATOImat=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    123.00      .       456.00      .       8.0000
 2 |    123.00      .       456.00      .       8.0000

ATOFmat=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    3.1415     .       0.31500     .        21.000
 2 |    3.1415     .       0.31500     .        21.000
```

4.2 Extensions of `str...` Functions

The following functions with string arguments have been extended for vector, matrix, and tensor arguments.

1. One-argument functions:

   The input argument s can now be scalar, vector, matrix or tensor, have normally string entries, and the result has the same dimension as the input argument:

$i =$ **strlen**(s)

$str =$ **strrev**(s)

$str =$ **strlwr**(s)

$str =$ **strupr**(s)

The function names correspond to those in the C language. If the data object has other than string data (CMAT, other than e.g. SAS/IML, permits mixed data types in objects), the behavior of the `strlen()` is slightly different from the other functions:

`strlen()` For data entries that are missing the corresponding entries in the result are missing. For data entries which are numeric, the result is zero.

other functions For data entries that are missing or which are numeric, the corresponding entries of the result are missing.

2. Two-argument functions:

   The two input arguments have usually string data and

   (a) both arguments can be scalars: the result is scalar

   (b) one of the arguments is scalar and the other is a data object: the result has the dimension of the data object; the operations are performed pairwise between the scalar and each entry of the data object.

   (c) both arguments can be data objects of the same size: the result has the same dimension as the two input arguments, the operations are performed pairwise between corresponding entries of the two data objects.

   $i =$ **strspn**(s, set)

   $i =$ **strcspn**(s, set)

   $i =$ **strpos**(s, c)

   $i =$ **strrpos**(s, c)

   $str =$ **strchr**(s, c)

   $str =$ **strrchr**(s, c)

   The function names correspond to those in the C language. For data entries that are missing or which are numeric, the corresponding entries of the result are missing.

3. Three-argument functions:

   Since the first two arguments are commonly string and the third (integer) argument is optional, the usage is the same as that for the two-argument functions, i.e. the arguments $s1$ and $s2$ either must have the same size, or one of the arguments is scalar.

str = **strcat**($s1, s2 <, n >$)

i = **strcmp**($s1, s2 <, n >$)

str = **strcpy**($s1, s2 <, n >$)

If n is not specified the function names correspond to those in the C language. If $n \geq 0$ is specified the function names correspond to `strncat(), strncmp(), strncpy()`. Specifying $n < 0$ results in an error.

---

str = strcat(s1,s2<,n>)

Purpose: The `strcat` function appends the string $s2$ to the end of string $s1$ and the new string is returned. The termninating null character at the end of string $s1$ is overwritten. If n is specified as an integer, only the first n characters of $s2$ are appended to $s1$. If n is not specified the function name corresponds to that in the C language. If $n \geq 0$ is specified the function name corresponds to `strncat()`.

Input: **s1** string scalar or object with string data

s2 string scalar or object with string data

n optional argument: if specified must be nonnegative.

Output: The return object has the same dimension as the larger of the two input arguments and normally contains string data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions:
1. A missing value is returned if the input arguments $s1$ and $s2$ is a missing value or does not contain string data.
2. Input argument n must be either missing value or nonnegative integer.

Relationships:

Examples: 1. Two scalar arguments:

```
sa = "Hello "; sb = "world"; sc = "**********";
print "STRCAT 0: ", s1 = strcat(sa,sb);
print "STRCAT 1: ", s2 = strcat(sa,sb,8);
print "STRCAT 2: ", s3 = strcat(s2,sc,4);
print "STRCAT 3: ", s4 = strcat(s3,sc,0);
```

```
STRCAT 0: Hello world
STRCAT 1: Hello world
STRCAT 2: Hello world****
STRCAT 3: Hello world****
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRCAT 4: ", s5 = strcat(vec,sb);
print "STRCAT 5: ", s6 = strcat(mat,sb);
print "STRCAT 6: ", s7 = strcat(mat2,sb);
print "STRCAT 7: ", s8 = strcat(vec,vec2);
print "STRCAT 8: ", s9 = strcat(mat,mat2);
```

```
STRCAT 4:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
            aBccd     .          cdeFdcd     .          gHIJkcd


STRCAT 5:
                Mixed Type Symmetric Matrix (3 by 3)

                                    1       2     3
                     1       abccd       0     0
                     2       .      ghijkcd     0
                     3      cdefcd  lmnocd xyzcd


STRCAT 6:
 S |         1         2         3
-----------------------------------
 1 |     abccd
 2 |     defcd   ghijkcd
 3 |    cdefcd    lmnocd     xyzcd


STRCAT 7:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
```

```
                aBcax     .       cdeFdcdef     .       gHIJkHIjk
```

```
STRCAT 8:
                Mixed Type Symmetric Matrix (3 by 3)

                             1          2      3
                1      abcabc          0      0
                2      .      ghijkghijk      0
                3    cdefcdef   lmnolmno xyzxyz
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRCAT 1: ", s1 = strcat(ten1,sb);
print "STRCAT 2: ", s2 = strcat(ten2,sb);
print "STRCAT 3: ", s3 = strcat(ten1,ten2);
```

---

str = strchr(s,c)

Purpose: The function `strchr` searches the null terminating string *s* for the first occurence of the character *c*. If the character *c* is found, the rest of the string is returned. If the character *c* is not found, a missing value is returned.

Input: **s** string scalar or object with string data

c string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains string data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments s and c is a missing value or does not contain string data.

Relationships: strrchr()

Examples: 1. Two scalar arguments:

```
sa = "video x-rays";
print "STRCHR: ", sr = strchr(sa,"x");
```

```
STRCHR: x-rays
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRCHR 2: ", s2 = strchr(vec,sb);
print "STRCHR 3: ", s3 = strchr(mat,sb);
print "STRCHR 4: ", s4 = strchr(mat2,sb);
print "STRCHR 5: ", s5 = strchr(vec,vec2);
print "STRCHR 6: ", s6 = strchr(mat,mat2);
```

```
STRCHR 2:
                  Mixed Type Row Vector (ncol=5)

     R |          1          2          3          4          5
               c     .           cdeFd     .
```

```
STRCHR 3:
               Mixed Type Symmetric Matrix (3 by 3)

                           1          2    3
                1          c      .      cdef
                2      .
                3        cdef
```

```
STRCHR 4:
 S |      1      2      3
-------------------------
 1 |      c
 2 |    def
 3 |   cdef


STRCHR 5:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               aBc     .           cdeFd     .           HIJk


STRCHR 6:
               Mixed Type Symmetric Matrix (3 by 3)

                                1          2    3
                    1         abc      .      cdef
                    2       .           ghijk lmno
                    3         cdef       lmno  xyz
```

3. Tensor arguments:

   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;

mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRCHR 1: ", s1 = strchr(ten1,sb);
print "STRCHR 2: ", s2 = strchr(ten2,sb);
print "STRCHR 3: ", s3 = strchr(ten1,ten2);
```

int = strcmp(s1,s2<,n>)

Purpose: The function `strcmp` lexicographically compares the contents of string $s1$ with string $s2$. The integer returned is less than zero if $s1$ is less $s2$, it is equal zero if $s1$ is equal $s2$ and it is greater than zero if $s1$ is greater than $s2$. If the third input argument n is specified, only the first n characters of the input string $s1$ are being used. If n is not specified the function name corresponds to that in the C language. If $n \geq 0$ is specified the function name corresponds to `strncmp()`.

Input: **s1** string scalar or object with string data

s2 string scalar or object with string data

n optional argument: if specified must be nonnegative.

Output: The return object has the same dimension as the larger of the two input arguments and normally contains integer data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions:
1. A missing value is returned if the input arguments $s1$ and $s2$ is a missing value or does not contain string data.
2. Input argument n must be either missing value or nonnegative integer.

Relationships:

Examples: 1. Two scalar arguments:

```
sa = "abcdef"; sb = "abcdef";
print "STRCMP 1: ", i1 = strcmp(sa,sb);

sa = "abcdef"; sb = "abc";
print "STRCMP 2: ", i2 = strcmp(sa,sb);
sa = "abc"; sb = "abcdef";
print "STRCMP 3: ", i3 = strcmp(sa,sb);
sa = "abcdef"; sb = "mnopqr";
print "STRCMP 4: ", i4 = strcmp(sa,sb);
sa = "mnopqr"; sb = "abcdef";
print "STRCMP 5: ", i5 = strcmp(sa,sb);
```

```
STRCMP 1:  0
STRCMP 2:  1
STRCMP 3: -1
STRCMP 4: -1
STRCMP 5:  1
```

```
sa = "abcdef"; sb = "abcDEF";
print "STRCMP 1: ", i1 = strcmp(sa,sb,10);
print "STRCMP 2: ", i2 = strcmp(sa,sb,6);
print "STRCMP 3: ", i3 = strcmp(sa,sb,3);
print "STRCMP 4: ", i4 = strcmp(sa,sb,0);
```

```
STRCMP 1:  32
STRCMP 2:  32
STRCMP 3:  0
STRCMP 4:  0
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRCMP 5: ", i5 = strcmp(vec,sb);
print "STRCMP 6: ", i6 = strcmp(mat,sb);
print "STRCMP 7: ", i7 = strcmp(mat2,sb);
print "STRCMP 8: ", i8 = strcmp(vec,vec2);
print "STRCMP 9: ", i9 = strcmp(mat,mat2);
```

```
STRCMP 5:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |  -1.00000    .        1.00000    .        1.00000


STRCMP 6:
 S |         1         2         3
-----------------------------------
 1 |  -1.00000
 2 |    .        1.00000
 3 |   1.00000   1.00000   1.00000


STRCMP 7:
```

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


STRCMP 8:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |  -1.00000    .       -1.00000    .        1.00000


STRCMP 9:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |     .       0.00000
 3 |   0.00000   0.00000   0.00000
```

3. Tensor arguments:

   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRCMP 1: ", i1 = strcmp(ten1,sb);
print "STRCMP 2: ", i2 = strcmp(ten2,sb);
print "STRCMP 3: ", i3 = strcmp(ten1,ten2);
```

---

str = strcpy(s1,s2<,n>)

Purpose: The function `strcpy` is useful when string $s1$ is longer than string $s1$ since it copies the contents of string $s2$ to the string $s1$, overwriting the old content of the first part of $s1$. If the third input argument n is specified as an integer, only the first n characters of string $s2$ are copied to $s1$. If n is not specified the function name corresponds to that in the C language. If $n \geq 0$ is specified the function name corresponds to `strncpy()`.

Input: **s1** string scalar or object with string data

s2 string scalar or object with string data

n optional argument: if specified must be nonnegative.

Output: The return object has the same dimension as the larger of the two input arguments and normally contains string data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions:

1. A missing value is returned if the input arguments $s1$ and $s2$ is a missing value or does not contain string data.
2. Input argument n must be either missing value or nonnegative integer.

Relationships:

Examples:

1. Two scalar arguments:

```
sa = "abcdefg"; sb = "1234567"; sc = "*******";
print "STRCPY 1: ", s1 = strcpy(sa,sb,6);
print "STRCPY 2: ", s2 = strcpy(s1,sa,3);
print "STRCPY 3: ", s3 = strcpy(s2,sc,0);
```

```
STRCPY 1: 123456g
STRCPY 2: abc456g
STRCPY 3: abc456g
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
```

```
   print "STRCPY 5: ", s5 = strcpy(vec,sb);
   print "STRCPY 6: ", s6 = strcpy(mat,sb);
   print "STRCPY 7: ", s7 = strcpy(mat2,sb);
   print "STRCPY 8: ", s8 = strcpy(vec,vec2);
   print "STRCPY 9: ", s9 = strcpy(mat,mat2);
```

```
STRCPY 5:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               cdc     .           cdeFd     .           cdIJk
```

```
STRCPY 6:
               Mixed Type Symmetric Matrix (3 by 3)

                                  1          2    3
                   1         cdc      .      cdef
                   2      .            cdijk cdno
                   3        cdef        cdno  cdz
```

```
STRCPY 7:
 S |       1       2       3
---------------------------
 1 |     cdc
 2 |     cdf   cdijk
 3 |    cdef    cdno     cdz
```

```
STRCPY 8:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               axc     .           cdefd     .           HIjkk
```

```
STRCPY 9:
               Mixed Type Symmetric Matrix (3 by 3)

                                  1          2    3
                   1         abc      .      cdef
                   2      .            ghijk lmno
                   3        cdef        lmno  xyz
```

3. Tensor arguments:

Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRCPY 1 ", s1= strcpy(ten1,sb);
print "STRCPY 2 ", s2= strcpy(ten2,sb);
print "STRCPY 3 ", s3= strcpy(ten1,ten2);
```

---

int = strcspn(s,set)

Purpose: The function `strcspn` searches the string *s* for the first occurence of a character that is included in the string *set*, skipping over characters that are not in *set*. The second argument *set* is regarded as a collection of characters; the order of the characters or whether there are duplications does not matter. The value returned is the length of the longest initial segment of *s* that consists of characters not found in *set*. If no character of *s* appears in *set* the length of *s* is returned.

Input: **s** string scalar or object with string data

set string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains integer data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments *s* and *set* is a missing value or does not contain string data.

Relationships: strspn()

Examples: 1. Two scalar arguments:

```
sa = "abcbcadef"; sb = "xxxbcadef";
sc = "123456789"; sd = "cba";
print "STRCSPN 1: ", i1 = strcspn(sa,sd);
print "STRCSPN 2: ", i2 = strcspn(sb,sd);
print "STRCSPN 3: ", i3 = strcspn(sc,sd);
```

```
STRCSPN 1:  0
STRCSPN 2:  3
STRCSPN 3:  9
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRCSPN 3: ", i4 = strcspn(vec,sb);
print "STRCSPN 4: ", i5 = strcspn(mat,sb);
print "STRCSPN 5: ", i6 = strcspn(mat2,sb);
print "STRCSPN 6: ", i7 = strcspn(vec,vec2);
print "STRCSPN 7: ", i8 = strcspn(mat,mat2);
```

```
STRCSPN 3:
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    2.0000    .        0.00000    .         5.0000


STRCSPN 4:
 S |         1         2         3
-----------------------------------
 1 |    2.0000
 2 |    .         5.0000
 3 |   0.00000    4.0000    3.0000


STRCSPN 5:
 S |   1   2   3
```

```
---------------
 1 |   2
 2 |   0   5
 3 |   0   4   3


STRCSPN 6:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |    .         .        0.00000   0.00000   1.00000


STRCSPN 7:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |    .        0.00000
 3 |   0.00000   0.00000   0.00000
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRCSPN 1: ", i1 = strcspn(ten1,sb);
print "STRCSPN 2: ", i2 = strcspn(ten2,sb);
print "STRCSPN 3: ", i3 = strcspn(ten1,ten2);
```

---

int = strlen(s)

Purpose: The function `strlen` returns the number of characters in *s* preceding the terminating null character. The length of an empty string is zero.

Input: The only input argument is either string scalar or object with string data.

Output: The return object has the same dimension as the input argument and normally contains integer data. If the argument is missing, a missing value is returned and if the argument is numeric, zero is returned.

Restrictions: 1. A missing value is returned if the input argument *s* is a missing value.

2. A zero is returned if the input argument *s* is numeric.

Relationships:

Examples: 1. Two scalar arguments:

```
sa = "Howdy"; sb = "Hello world\n"; sc = " ";
print "STRLEN 1: ", i1 = strlen(sa);
print "STRLEN 2: ", i1 = strlen(sb);
print "STRLEN 3: ", i1 = strlen(sc);
```

```
STRLEN 1:  5
STRLEN 2:  12
STRLEN 3:  1
```

2. Vector and matrix arguments:

```
vec = [ "aBc" . "CdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRLEN 4: ", i1 = strlen(vec);
print "STRLEN 5: ", i2 = strlen(vec2);
print "STRLEN 6: ", i3 = strlen(mat);
print "STRLEN 7: ", i4 = strlen(mat2);
```

```
STRLEN 4:
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    3.0000    .         5.0000   0.00000    5.0000
```

```
STRLEN 5:
   |   1   2   3   4   5
-----------------------
 1 |   2   4   4   0   4


STRLEN 6:
   |         1         2         3
-----------------------------------
 1 |    3.0000   0.00000    4.0000
 2 |    .         5.0000    4.0000
 3 |    4.0000    4.0000    3.0000


STRLEN 7:
 S |   1   2   3
---------------
 1 |   3
 2 |   3   5
 3 |   4   4   3
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


i1 = strlen(ten1);
print "STRLEN 1: ", i1;
print "STRLEN 2: ", i2 = strlen(ten2);
```

---

str = strlwr(s)

Purpose: The returned string *str* agrees with the input argument *s* except that all upper case alphabetic characters in *s* are changed to lower case.

Input: The only input argument is either string scalar or object with string data.

Output: The return object has the same dimension as the input argument and normally contains string data. If the argument is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input argument *s* does not contain string data.

Relationships: strupr()

Examples: 1. Two scalar arguments:

```
ss = "A mixed-case STRING";
print "STRLWR: ", sr = strlwr(ss);
```

```
STRLWR: a mixed-case string
```

2. Vector and matrix arguments:

```
vec = [ "aBc" . "CdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "ABC"    50.   "cdef" ,
          .    "ghijk" "LMNO" ,
        "Cdef" "LMNO"  "xYz" ];
mat2 = [ "ABC"   "def" "CDEF" ,
         "def" "ghijk" "LMNO" ,
         "CDEF" "LMNO"  "xYz" ];


print "STRLWR 2: ", s2 = strlwr(vec);
print "STRLWR 3: ", s3 = strlwr(vec2);
print "STRLWR 4: ", s4 = strlwr(mat);
print "STRLWR 5: ", s5 = strlwr(mat2);
```

```
STRLWR 2:
                  Mixed Type Row Vector (ncol=5)

     R |          1         2         3         4         5
               abc     .          cdefd     .          ghijk


STRLWR 3:
                  Mixed Type Row Vector (ncol=5)

     R |          1         2         3         4         5
```

```
                  ax       pqrs       cdef     .            hijk
```

```
STRLWR 4:
               Mixed Type Symmetric Matrix (3 by 3)

                                 1           2    3
                    1          abc       .       cdef
                    2       .             ghijk lmno
                    3         cdef         lmno  xyz
```

```
STRLWR 5:
 S |       1       2       3
---------------------------
 1 |     abc
 2 |     def   ghijk
 3 |    cdef    lmno     xyz
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "aBc"    50.   "cDEf" ,
            .    "gHIJk" "lMNo" ,
          "cDEf" "lMNo"  "xYz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "aBc"   "dEf" "cDEf" ,
         "dEf" "ghijk" "lMNo" ,
         "cDEf" "lMNo"  "xYz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRLWR 1: ", s1 = strlwr(ten1);
print "STRLWR 2: ", s2 = strlwr(ten2);
```

---

int = strpos(s,c)

Purpose: The function `strpos` searches the string s for the first occurence of the character c. If the character c is found in s, the position of the first occurence is returned. If the character is not found, the value -1 is returned.

Input: **s** string scalar or object with string data

c string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains integer data. If one of the two arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments *s* and *c* is a missing value or does not contain string data.

Relationships: strrpos()

Examples: 1. Two scalar arguments:

```
sa = "video x-rays and x movies";
print "STRPOS: ", where = strpos(sa,"x");
```

```
STRPOS:  6
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRPOS 2: ", i2 = strpos(vec,sb);
print "STRPOS 3: ", i3 = strpos(mat,sb);
print "STRPOS 4: ", i4 = strpos(mat2,sb);
print "STRPOS 5: ", i5 = strpos(vec,vec2);
print "STRPOS 6: ", i6 = strpos(mat,mat2);
```

```
STRPOS 2:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |    2.0000    .        0.00000    .         5.0000


STRPOS 3:
 S |         1         2         3
```

```
---------------------------------
 1 |    2.0000
 2 |    .         5.0000
 3 |   0.00000    4.0000    3.0000


STRPOS 4:
 S |   1   2   3
---------------
 1 |   2
 2 |   0   5
 3 |   0   4   3


STRPOS 5:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |    .         .        0.00000   0.00000   1.00000


STRPOS 6:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |    .        0.00000
 3 |   0.00000   0.00000   0.00000
```

3. Tensor arguments:

   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRPOS 1: ", i1 = strpos(ten1,sb);
print "STRPOS 2: ", i2 = strpos(ten2,sb);
```

```
print "STRPOS 3: ", i3 = strpos(ten1,ten2);
```

---

str = strrchr(s,c)

Purpose: The function `strrchr` searches the null terminating string *s* for the last occurence of the character *c*. If the character *c* is found, the rest of the string is returned. If the character *c* is not found, a missing value is returned.

Input: **s** string scalar or object with string data

c string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains string data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments *s* and *c* is a missing value or does not contain string data.

Relationships: strchr()

Examples: 1. Two scalar arguments:

```
sa = "abcdeabcde"; sb = "c"; sc = "x";
print "STRRCHR 1: ", s1 = strchr(sa,sb);
print "STRRCHR 2: ", s2 = strchr(sa,sc);
```

```
STRRCHR 1: cdeabcde
STRRCHR 2:
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
```

```
   print "STRRCHR 3: ", s3 = strrchr(vec,sb);
   print "STRRCHR 4: ", s4 = strrchr(mat,sb);
   print "STRRCHR 5: ", s5 = strrchr(mat2,sb);
   print "STRRCHR 6: ", s6 = strrchr(vec,vec2);
   print "STRRCHR 7: ", s7 = strrchr(mat,mat2);
```

```
STRRCHR 3:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               cBa     .           dFedc     .
```

```
STRRCHR 4:
               Mixed Type Symmetric Matrix (3 by 3)

                                 1          2   3
                    1         cba      .       dc
                    2      .
                    3          dc
```

```
STRRCHR 5:
 S |     1     2     3
-----------------------
 1 |   cba
 2 |     d
 3 |    dc
```

```
STRRCHR 6:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               a     .           dFedc     .          kJIHg
```

```
STRRCHR 7:
               Mixed Type Symmetric Matrix (3 by 3)

                                   1          2    3
                    1         cba      .      fedc
                    2      .           kjihg onml
                    3        fedc        onml  zyx
```

3. Tensor arguments:

Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRRCHR 1: ", s1 = strrchr(ten1,sb);
print "STRRCHR 2: ", s2 = strrchr(ten2,sb);
print "STRRCHR 3: ", s3 = strrchr(ten1,ten2);
```

---

str = strrev(s)

Purpose: The returned string *str* is the same as the input argument *s*, however, the characters are arranged in reverse order.

Input: The only input argument is either string scalar or object with string data.

Output: The return object has the same dimension as the input argument and normally contains string data. If the argument is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input argument *s* does not contain string data.

Relationships:

Examples: 1. Two scalar arguments:

```
sa = "A sample STRING";
print "STRREV 1: ", sb = strrev(sa);
print "STRREV 2: ", sc = strrev(sb);


STRREV 1: GNIRTS elpmas A
STRREV 2: A sample STRING
```

2. Vector and matrix arguments:

```
vec = [ "aBc" . "CdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
print "STRREV 3: ", i1 = strrev(vec);
print "STRREV 4: ", i2 = strrev(vec2);
print "STRREV 4: ", i3 = strrev(mat);
print "STRREV 5: ", i4 = strrev(mat2);
```

```
STRREV 3:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
               cBa     .           dFedC     .           kJIHg

STRREV 4:
                  Mixed Type Row Vector (ncol=5)

     R |           1          2          3          4          5
                xa       srQP       fedc     .           kjIH

STRREV 4:
               Mixed Type Symmetric Matrix (3 by 3)

                             1          2    3
                  1        cba      .      fedc
                  2      .           kjihg onml
                  3       fedc        onml  zyx

STRREV 5:
 S |       1       2       3
---------------------------
 1 |     cba
 2 |     fed   kjihg
 3 |    fedc    onml     zyx
```

3. Tensor arguments:

Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRREV 1: ", i1 = strrev(ten1);
print "STRREV 2: ", i2 = strrev(ten2);
```

---

int = strrpos(s,c)

Purpose: The function `strpos` searches the string *s* for the last occurence of the character *c*. If the character *c* is found in *s*, the position of the last occurence is returned. If the character is not found, the value -1 is returned.

Input: **s** string scalar or object with string data

c string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains integer data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments *s* and *c* is a missing value or does not contain string data.

Relationships: strpos()

Examples: 1. Two scalar arguments:

```
sa = "video x-rays and x movies";
print "STRRPOS: ", where = strrpos(sa,"x");


STRRPOS:  7
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRRPOS 2: ", i2 = strrpos(vec,sb);
print "STRRPOS 3: ", i3 = strrpos(mat,sb);
print "STRRPOS 4: ", i4 = strrpos(mat2,sb);
print "STRRPOS 5: ", i5 = strrpos(vec,vec2);
print "STRRPOS 6: ", i6 = strrpos(mat,mat2);
```

```
STRRPOS 2:
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00000    .        0.00000    .         5.0000


STRRPOS 3:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |    .         5.0000
 3 |    2.0000    4.0000    3.0000


STRRPOS 4:
 S |   1   2   3
----------------
 1 |   0
 2 |   2   5
 3 |   2   4   3


STRRPOS 5:
   |         1         2         3         4         5
---------------------------------------------------------
 1 |    2.0000    .         .        0.00000   0.00000
```

```
STRRPOS 6:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |    .        0.00000
 3 |   0.00000   0.00000   0.00000
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRRPOS 1: ", i1 = strrpos(ten1,sb);
print "STRRPOS 2: ", i2 = strrpos(ten2,sb);
print "STRRPOS 3: ", i3 = strrpos(ten1,ten2);
```

---

int = strspn(s,set)

Purpose: The function `strspn()` searches the string *s* for the first occurence of a character that is **not** included in the string *set*, skipping over characters that are in *set*. The second argument *set* is regarded as a collection of characters; the order of the characters or whether there are duplications does not matter. The value returned is the length of the longest initial segment of *s* that consists of characters found in *set*. If every character of *s* appears in *set* the length of *s* is returned.

Input: **s** string scalar or object with string data

set string scalar or object with string data

Output: The return object has the same dimension as the larger of the two input arguments and normally contains integer data. If one of the two input arguments is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input arguments *s* and *set* is a missing value or does not contain string data.

Relationships: strcspn()

Examples: 1. Two scalar arguments:

```
sa = "out to lunch"; sb = "aeiou"; sc = "xyz";
print "STRSPN 1: ", i1 = strspn(sa,sb);
print "STRSPN 2: ", i2 = strspn(sa,sc);
```

```
STRSPN 1:  2
STRSPN 2:  0
```

2. Vector and matrix arguments:

```
sb = "cd";
vec = [ "aBc" . "cdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "abc"    50.   "cdef" ,
          .    "ghijk" "lmno" ,
        "cdef" "lmno"  "xyz" ];
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];


print "STRSPN 3: ", i3 = strspn(vec,sb);
print "STRSPN 4: ", i4 = strspn(mat,sb);
print "STRSPN 5: ", i5 = strspn(mat2,sb);
print "STRSPN 6: ", i6 = strspn(vec,vec2);
print "STRSPN 7: ", i7 = strspn(mat,mat2);
```

```
STRSPN 3:
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00000    .         2.0000    .         0.00000


STRSPN 4:
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |    .         0.00000
 3 |    2.0000    0.00000   0.00000
```

```
STRSPN 5:
 S |   1   2   3
---------------
 1 |   0
 2 |   1   0
 3 |   2   0   0


STRSPN 6:
   |         1         2         3         4         5
-------------------------------------------------------
 1 |   1.00000    .         3.0000    .        0.00000


STRSPN 7:
 S |         1         2         3
-----------------------------------
 1 |    3.0000
 2 |    .         5.0000
 3 |    4.0000    4.0000    3.0000
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
sb = "cd";
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "abc"    50.   "cdef" ,
           .    "ghijk" "lmno" ,
          "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "abc"   "def" "cdef" ,
         "def" "ghijk" "lmno" ,
         "cdef" "lmno"  "xyz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRSPN 1: ", i1 = strspn(ten1,sb);
print "STRSPN 2: ", i2 = strspn(ten2,sb);
print "STRSPN 3: ", i3 = strspn(ten1,ten2);
```

---

str = strupr(s)

Purpose: The returned string *str* agrees with the input argument *s* except that all lower case alphabetic characters in *s* are changed to upper case.

Input: The only input argument is either string scalar or object with string data.

Output: The return object has the same dimension as the input argument and normally contains string data. If the argument is missing or numeric, a missing value is returned.

Restrictions: 1. A missing value is returned if the input argument *s* does not contain string data.

Relationships:

Examples: 1. Two scalar arguments:

```
ss = "A mixed-case STRING";
print "STRUPR: ", sr = strupr(ss);
```

```
STRUPR: A MIXED-CASE STRING
```

2. Vector and matrix arguments:

```
vec = [ "aBc" . "CdeFd" 100. "gHIJk" ];
vec2 = [ "ax" "PQrs" "cdef" 3 "HIjk" ];
mat = [ "ABC"    50.   "cdef" ,
          .    "ghijk" "LMNO" ,
        "Cdef" "LMNO"  "xYz" ];
mat2 = [ "ABC"   "def" "CDEF" ,
         "def" "ghijk" "LMNO" ,
         "CDEF" "LMNO"  "xYz" ];


print "STRUPR 2: ", s2 = strupr(vec);
print "STRUPR 3: ", s3 = strupr(vec2);
print "STRUPR 4: ", s4 = strupr(mat);
print "STRUPR 5: ", s5 = strupr(mat2);
```

```
STRUPR 2:
                  Mixed Type Row Vector (ncol=5)

     R |          1          2          3          4          5
               ABC     .          CDEFD     .          GHIJK
```

```
STRUPR 3:
     R |           1          2          3          4          5
               AX       PQRS       CDEF     .            HIJK


STRUPR 4:
               Mixed Type Symmetric Matrix (3 by 3)

                                 1          2    3
                    1          ABC      .      CDEF
                    2      .             GHIJK LMNO
                    3        CDEF         LMNO  XYZ


STRUPR 5:
 S |       1       2       3
---------------------------
 1 |     ABC
 2 |     DEF   GHIJK
 3 |    CDEF    LMNO     XYZ
```

3. Tensor arguments:
   Only the input is shown (for space reasons). The results can be looked up in the CMAT test directory and are similar to those of the matrix arguments since each slide of the input tensor is a similar matrix:

```
real ten1[3,3,3],ten2[3,3,3];
mat1 = [ "aBc"    50.   "cDEf" ,
           .    "gHIJk" "lMNo" ,
         "cDEf" "lMNo"  "xYz" ];
for (i = 1; i <= 3; i++) ten1[i,,] = mat1;
mat2 = [ "aBc"   "dEf" "cDEf" ,
         "dEf" "ghijk" "lMNo" ,
         "cDEf" "lMNo"  "xYz" ];
for (i = 1; i <= 3; i++) ten2[i,,] = mat2;


print "STRUPR 1: ", s1 = strupr(ten1);
print "STRUPR 2: ", s2 = strupr(ten2);
```

4.3 Extensions of `decrypt()` Functions

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

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 (which has been used with the encryption of the file) is still available. There are two ways to specify this password:

- as the same string with the third input argument `pwd` that was used at the encryption.
- if the third input argument is not specified (specified as a missing value), then the output password file from the encrypting run must be available for reading from the `cmat/save/encrypt` directory. The path of the password file can be
  - either specified using the first entry of the `optn` input argument
  - or can be derived from the `ofil` or `ifil` arguments by default:

    encrypt The `encrypt()` function derives the name of the password file from the name specified with the `ofile` string.

    decrypt The `decrypt()` function derives the name of the password file from the name specified with the `ifile` string.

  The name of the password file **must be** specified when the `ifil` and `ofil` arguments are vectors and not single file or directory names. For the default version of derived password file names the files are always assumed written to or ead from the `cmat/save/encrypt/` directory. To avoid overwriting and loosing password files it is recommended to specify the path name of the password file over the default derivation.

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.

The function `decrypt()` is useful only for input files which were generated by function `encrypt()`. Note, ciphertext created using the hash algorithms SHA-2 and SHA-3 with functions `encrypt()` and `encryp2()` cannot be decrypted.

Input: ofil This argument can be missing or must be a string or a vector of n strings or missing values each specifying the path

- either for an **output directory** when `ifil` is a directory name which must end either with a slash / or a backslash \.
- or for an **output file** when `ifil` is a single file name

which will contain the *plaintext*. An output directory name should end with a slash / or backslash \, otherwise it will be assumed to be a file name. If a missing value is specified, the `ofil` string is created from the `ifil` string extended by `_enc` suffix.

ifil This must be either one string or a vector of n strings each specifying the path

- either for an **input directory** (a name which must end either with a slash / or a backslash \)
- or for a single **input file**

which must contain the *ciphertext*. An input directory name should end with a slash / or backslash \, otherwise it will be assumed to be a file name.

pwd If this argument is specified it must be a string which is the same as that which has been used to obtain the encrypted file `ifil`. If this argument is not specified, it is assumed that a 256 character password file is available either at a specified location (first entry of the `optn` input argument) or at the `cmat/save/encrypt/` directory.

optn This argument maybe specified as a single string or a vector of strings. For the meaning of the vector entries see table below. Note, that the options used for decryption must be compatible with those used for encryption.

Entry	Option	Meaning
1	string	path name to password file
2	"own"	own encryption method
	"aes"	Advanced Encryption Standard (Rijndael; Hellekalek & Wegenkittel, 2003))
	"anu"	Anubis (Barreto & Rijmen, 2000)
	"cam"	Camellia (Ichikawa, Matsui, Nakajima, & Tokita, Mitsubishi, 2001)
	"cle"	Clefia (Sony, 2007)
	"hr3"	Hierocrypt-3 (Myasnikov)
	"sc2"	sc2000 (Myasnikov)
	"spe"	SPEED (Zheng)
	"2fi"	Twofish (Gladman, 1999)
	"unc"	CipherUnicorn (Myasnikov)
3	"abo"	abort encryption if password file with same name exists (for savety reason this is the default)
	"wri"	overwrite existing password file
4	"sub"	process files in all subdirectories (the default)
	"nos"	do not process any subdirectories

Note, that the hash algorithms SHA-2 and SHA-3 can only be used for encryption but not for decryption.

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

Restrictions:

1. The number n of entries for the `ofil` and `ifil` vector arguments must be the same.
2. To differentiate between file and directory names the function assumes that directory names end either with a slash / or a backslash \.
3. The first argument cannot specify the path to a file (directory) if the second argument specifies a path to a directory (file). The two arguments must be insync.
4. Pathnames to files and directories must not contain any white space.
5. If the password (file) is lost, an encrypted file cannot be decrypted anymore.
6. The password and the encrypted files are written in binary mode.
7. Note, that the `optn` input argument used for decryption must be compatible with that used for encryption.

Relationships: encrypt(), decryp2(), encryp2()

Examples: See the `encrypt()` function (encrypt()) for a set of examples.

4.4 Extensions of the `encrypt()` Function

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

Purpose: The `encrypt()` function can be used for the save encryption of the content of files. This function implements a number of algorithms for encrypting the content of files. Similar, the `encryp2()` function can be used for encrypting scalars and vectors of string data. Most of the algorithms implemented create ciphertext which can be decrypted again using the `decrypt()` and `decryp2()` functions. However, this function also provides a number of hash algorithms like SHA-2 and SDHA-3 which create ciphertext that cannot be decrypted again, but are very useful especially for password encryption. Note, no passwords are used for the encryption with the hash algorithms SHA-2 and SHA-3.

If the output name `ofil` is not specified (the first argument is missing) the encrypted output files have the same name as the plaintext input files but with an extension `.enc` added to their names. The second input argument `ifil` must always be specified as a string or a vector of strings. The strings must be specifying the path name referring either to a directory or to a file. Assuming for simplicity that the second argument `ifil` is a single string specifying the path to a directory or file, then the first argument `ofil` could be either a missing value or must specify the path to a file or directory like the second argument does.

There are two ways to specify this password:

- the third input argument `pwd` is a string no more than 256 characters
- if the third input argument is not specified (specified as a missing value), then the password is a randomly generated password string of 264 characters and a password file is written. The path of the password file can be
  - either specified using the first entry of the `optn` input argument
  - or can be derived from the `ofil` or `ifil` arguments by default:

    encrypt The `encrypt()` function derives the name of the password file from the name specified with the `ofile` string.

    decrypt The `decrypt()` function derives the name of the password file from the name specified with the `ifile` string.

  The name of the password file **must be** specified when the `ifil` and `ofil` arguments are vectors and not single file or directory names. For the default version of derived password file names the files are always assumed written to or ead from the `cmat/save/encrypt/` directory. To avoid overwriting and loosing password files it is recommended to specify the path name of the password file over the default derivation.

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.

Input: ofil This argument can be missing or must be a string or a vector of n strings or missing values each specifying the path

- either for an **output directory** when `ifil` is a directory name which must end either with a slash / or a backslash \.
- or for an **output file** when `ifil` is a single file name

which will contain the *ciphertext*. An output directory name should end with a slash / or backslash \, otherwise it will be assumed to be a file name. If a missing value is specified, the `ofil` string is created from the `ifil` string extended by `_enc` suffix.

ifil This must be either one string or a vector of n strings each specifying the path

- either for an **input directory** (a name which must end either with a slash / or a backslash \)
- or for a single **input file**

which must contain the *plaintext*. An input directory name should end with a slash / or backslash \, otherwise it will be assumed to be a file name.

pwd If this argument is specified it must be a string, no more than 256 characters are used. If this argument is not specified, a 256 character password is machine generated. If no password string is specified a password file is written either to a specified location (first entry of the `optn` input argument) or to the `cmat/save/encrypt/` directory. This file must be available for the decryption.

optn This argument maybe specified as a single string or a vector of strings. The meaning of the vector entries is as follows:

Entry	Option	Meaning
1	string	path name to password file
2	"own"	own encryption method
	"aes"	Advanced Encryption Standard (Rijndael; Hellekalek & Wegenkittel, 2003))
	"anu"	Anubis (Barreto & Rijmen, 2000)
	"cam"	Camellia (Ichikawa, Matsui, Nakajima, & Tokita, Mitsubishi, 2001)
	"cle"	Clefia (Sony, 2007)
	"hr3"	Hierocrypt-3 (Myasnikov)
	"sc2"	sc2000 (Myasnikov)
	"spe"	SPEED (Zheng)
	"2fi"	Twofish (Gladman, 1999)
	"unc"	CipherUnicorn (Myasnikov)
	"sh2"	SHA-2 secure hash algorithm (Christophe Devine)
	"sh3"	SHA-3 SKEIN hash algorithm (Doug Whiting)
3	"abo"	abort encryption if password file with same name exists (for savety reason this is the default)
	"wri"	overwrite existing password file
4	"sub"	process files in all subdirectories (default)
	"nos"	do not process any subdirectories
5	"noh"	hash: output in string form (length=32)
	"hex"	hash: hexadecimal output (length=64) (default)

Output: 1. The path name for the encrypted output file is specified as the first input argument.

2. If no password is specified with input argument `pwd` a randonly generated password file is written either to a location specified with the first entry of the `optn` input argument or to the default location `cmat/save/encrypt/` with a file name derived from the `ofil` input argument.

Restrictions: 1. The number n of entries for the `ofil` and `ifil` vector arguments must be the same.

2. To differentiate between file and directory names the function assumes that directory names end either with a slash / or a backslash \.

3. Pathnames to files and directories must not contain any white space.

4. The first argument cannot specify the path to a file (directory) if the second argument specifies a path to a directory (file). The two arguments must be insync.

5. If the password (file) is lost, an encrypted file cannot be decrypted anymore.

6. The password and the encrypted files are written in binary mode.

7. Note, that the `optn` input argument used for decryption must be compatible with that used for encryption.

Relationships: decrypt(), encryp2(), decryp2()

Examples: 1. No password specified:

(a) File Encryption:
The plaintext file `cmat/test/multcomp/multcomp.txt` is encrypted and written to the file encrypted file is written to `cmat/test/multcomp.txt.enc` and the 256 byte password file `multcomp.txt.enc.pwd` is written to the `cmat/save/encrypt/` directory.

```
print "No password specified";
ipath = "..\\..\\cmat\\test\\multcomp\\multcomp.txt";
opath = "multcomp.txt.enc";


print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "wri" ];
encrypt(opath,ipath,.,optn);
```

The password file `multcomp.txt.enc.pwd` written during the encryption is picked up from the `cmat/save/encrypt/` directory and the `multcomp.txt.enc` file is decrypted to the `multcomp.txt.dec` file:

```
print "No password specified";
ipath = "multcomp.txt.enc";
opath = "multcomp.txt.dec";
decrypt(opath,ipath);
```

(b) Directory Encryption including subdirectories:
Specifying the `"sub"` option (which is default) the entire `cmat/test/multcomp/` directory, including subdirectories is to be encrypted and the encrypted files of the directory tree are written to the `cmat/test/multcomp_enc/` directory. The password file `multcomp_enc.pwd` is written to the `cmat/save/encrypt/` directory.

```
print "No password specified";
ipath = "..\\..\\cmat\\test\\multcomp\\";
opath = "multcomp_enc\\";


print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "wri", "sub" ];
encrypt(opath,ipath,.,optn);
```

The password file `multcomp_enc.pwd` is picked up from the `cmat/save/encrypt/` directory

```
print "No password specified";
ipath = "..\\..\\cmat\\test\\multcomp_enc\\";
opath = "multcomp_dec\\";
decrypt(opath,ipath,.,optn);
```

(c) Directory Encryption skipping subdirectories:
Specifying the `"nos"` option (which is not default) the files of the `cmat/test/multcomp/` directory, without subdirectories are to be encrypted and the encrypted files of the directory tree are written to the `cmat/test/multcmp_enc/` directory. The password file `multcmp_enc.pwd` is written to the `cmat/save/encrypt/` directory.

```
print "No password specified";
ipath = "..\\..\\cmat\\test\\multcomp\\";
opath = "multcmp_enc\\";

print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "wri", "nos" ];
encrypt(opath,ipath,.,optn);
```

The password file `multcmp_enc.pwd` is picked up from the `cmat/save/` directory

```
print "No password specified";
ipath = "..\\..\\cmat\\test\\multcmp_enc\\";
opath = "multcmp_dec\\";
decrypt(opath,ipath,.,optn);
```

2. Password is specified:

(a) File Encryption:
When specifying a password, no password file is written. The plaintext file is `cmat/test/multcomp/multcomp.txt` and the encrypted file is `cmat/test/multcomp2.txt.enc`.

```
print "Short Password specified: Length=5";
ipath = "..\\test\\multcomp\\multcomp.txt";
opath = "multcomp2.txt.enc";
passw = "bully";
encrypt(opath,ipath,passw);
```

For decryption the same password must be available:

```
ipath = "multcomp2.txt.enc";
opath = "multcomp2.txt.dec";
decrypt(opath,ipath,passw);
```

For the next example we specify a password of 128 characters and obtain the ciphertext file `cmat/test/multcomp3.txt.enc`:

```
print "Long Password specified: Length=128";
ipath = "..\\test\\multcomp\\multcomp.txt";
opath = "multcomp3.txt.enc";
pass1 = "bullybummybullybummybullybummybullybummybullybummy";
pass2 = "bullybummybullybummybullybummybullybummybullybummy";
pass3 = "bullybummybullybummybullybum";
passw = strcat(pass1,strcat(pass2,pass3));
encrypt(opath,ipath,passw);
```

For decryption the same password must be available:

```
ipath = "multcomp3.txt.enc";
opath = "multcomp3.txt.dec";
decrypt(opath,ipath,passw);
```

Changing the password only slightly at the end the decryption creates an output file `cmat/test/multcomp4.txt.dec` which is quite different from `cmat/test/multcomp3.txt.dec`:

```
/* Read with very similar password */
/* pass1 = "bullybummyblulybummybullybummybullybummybullybummy"; */
/* pass2 = "bullybummybullybummybullybummybullybummybullybummy"; */
pass3 = "bullybummybullybummybullibmu";
passw = strcat(pass1,strcat(pass2,pass3));
ipath = "multcomp3.txt.enc";
opath = "multcomp4.txt.dec";
decrypt(opath,ipath,passw);
```

(b) Directory Encryption including subdirectories:

```
print "Password specified: with Subdirectories";
passw = "bullybummy";
ipath = "..\\..\\cmat\\test\\multcomp\\";
opath = "multcpw1_enc\\";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "wri", "sub" ];
encrypt(opath,ipath,passw,optn);

print "Password specified: with Subdirectories";
ipath = "..\\..\\cmat\\test\\multcpw1_enc\\";
opath = "multcpw1_dec\\";
decrypt(opath,ipath,passw,optn);
```

(c) Directory Encryption skipping subdirectories:

```
print "Password specified: without subdirectories";
```

```
passw = "bullybummy";
ipath = "..\\..\\cmat\\test\\multcomp\\";
opath = "multcpw2_enc\\";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "wri", "nos" ];
encrypt(opath,ipath,passw,optn);

print "Password specified: without subdirectories";
ipath = "..\\..\\cmat\\test\\multcpw2_enc\\";
opath = "multcpw2_dec\\";
decrypt(opath,ipath,passw,optn);
```

3. Script of files and directories specified (vector input):
   Here the pathname of the password file must be specified (since a default name cannot be easily derived).

```
print "No password specified";
ipath = [ "multcomp\\multcomp.txt" ,
          "multcomp\\tt1.inp"      ,
          "multcomp\\tt2.c"        ,
          "multcomp\\"             ,
          "multcomp\\tt.c"         ];
opath = [ "multcpw3_enc\\multcomp.txt.enc" ,
          "multcpw3_enc\\tt1.inp.enc"      ,
          "multcpw3_enc\\tt2.c.enc"        ,
          "multcpw4_enc\\"                 ,
          "multcpw3_enc\\tt.c.enc"         ];
print "For vector input and unspeified password:\n",
      "path for password file must be specified";
pwfil= "../save/multvect.pwd";
optn = [ pwfil, "own", "wri", "sub" ];
encrypt(opath,ipath,.,optn);
```

   The specification for the decryption must be compatible with that used for the encryption:

```
ipath = [ "multcpw3_enc\\multcomp.txt.enc" ,
          "multcpw3_enc\\tt1.inp.enc"      ,
          "multcpw3_enc\\tt2.c.enc"        ,
          "multcpw4_enc\\"                 ,
          "multcpw3_enc\\tt.c.enc"         ];
opath = [ "multcpw3_dec\\multcomp.txt.dec" ,
          "multcpw3_dec\\tt1.inp.dec"      ,
          "multcpw3_dec\\tt2.c.dec"        ,
          "multcpw4_dec\\"                 ,
```

```
          "multcpw3_dec\\tt.c.dec"         ];
print "For vector input and unspecified password:\n",
      "path for password file must be specified";
decrypt(opath,ipath,.,optn);
```

4. Different Methods:

   (a) "aes": Advanced Encryption Standard: Hellakalek & Wegenkittel (2003)

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_aes.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "aes" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_aes.txt.enc";
opath = "multcmp2_dec\\multc_aes.txt.dec";
decrypt(opath,ipath,.,optn);
```

   (b) "anu": Anubis: Rijmen & Barreto (2000)

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_anu.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "anu" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_anu.txt.enc";
opath = "multcmp2_dec\\multc_anu.txt.dec";
decrypt(opath,ipath,.,optn);
```

   (c) "cam": Camellia: Aoki, Kanda, & Moriai: *Nippon Telegraph and Telephone Corporation*, Ichikawa, Matsui, Nakajima, & Tokita: *Mitsubishi Electric Corporation*:

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_cam.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "cam" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_cam.txt.enc";
opath = "multcmp2_dec\\multc_cam.txt.dec";
decrypt(opath,ipath,.,optn);
```

   (d) "cle": Clefia: *Sony*

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_cle.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "cle" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_cle.txt.enc";
opath = "multcmp2_dec\\multc_cle.txt.dec";
decrypt(opath,ipath,.,optn);
```

(e) "hcr": HIEROCRYPT-3: Myasnikow: www.darksoftware.narod.ru

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_hcr.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "hcr" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_hcr.txt.enc";
opath = "multcmp2_dec\\multc_hcr.txt.dec";
decrypt(opath,ipath,.,optn);
```

(f) "sc2": SC-2000: Myasnikow: www.darksoftware.narod.ru

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_sc2.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "sc2" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_sc2.txt.enc";
opath = "multcmp2_dec\\multc_sc2.txt.dec";
decrypt(opath,ipath,.,optn);
```

(g) "spe": SPEED (Zheng)

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_spe.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "spe" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_spe.txt.enc";
opath = "multcmp2_dec\\multc_spe.txt.dec";
decrypt(opath,ipath,.,optn);
```

(h) "2fi": Twofish (Schneier & Colleagues):

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_2fi.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "2fi" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_2fi.txt.enc";
opath = "multcmp2_dec\\multc_2fi.txt.dec";
decrypt(opath,ipath,.,optn);
```

(i) "uni": CIPHERUNICORN Myasnikow: www.darksoftware.narod.ru

```
ipath = "multcomp\\multcomp.txt";
opath = "multcmp2_enc\\multc_uni.txt.enc";
print "The 'wri' option for overwriting password file is dangerous";
optn = [ ., "uni" , "wri" ];
encrypt(opath,ipath,.,optn);

ipath = "multcmp2_enc\\multc_uni.txt.enc";
opath = "multcmp2_dec\\multc_uni.txt.dec";
decrypt(opath,ipath,.,optn);
```

5 New Developments

5.1 The `csvread()` Function

a = csvread(fpath<,optn>)

Purpose: The `csvread()` function can be used for reading the values (and maybe row or column names) from a file referred to by its path into an object. The data may contain string or numeric values including missing values. The end of the line in the input file indicates the end of the row in the output object. If rows have different numbers of comma separators, they are filled up at the end by missing values. Strings may (but not necessarily must) be in double quotes. No values between the comma separators or strings of zero length are coded into missing values.

Input: **fpath** Must be a string (in double quotes) to a valid file with comma separated values.

optn Can be missing or a vector of no more than four entries with the following:

1. =0: do not transform `a`, =1: do transform `a`
2. separator other than `","`
3. i = 0: file does not contain any row names, i ¿ 0: column i contains row names
4. j = 0: file does not contain any row names, j ¿ 0: row j contains row names

Output: The only return argument `a` is either a vector or a matrix with the values from the input data file. Depending on the entries `[3]` and `[4]` and the contant of the data set `fpath` of the option vector, row and/or column names may be assigned to the object `a`.

Restrictions:

Relationships: genereadO

Examples: 1. Example 1: all rows have same length:

The data set `csvread.dat` is the following:

```
Name, Party, Age, Weight
"Phil", 0, 55,180
"Jack", 0, 29,200
"Wolfgang", 1, 60, 175
"Billy", 0, 18,
"Bob",0, ,160
" ", 1, 42, 145
```

```
path = "..\\tdata\\csvread.dat";
optn = cons(8,1,.);
optn[1] = 0;   /* itra  */
optn[3] = 0;   /* irnam */
optn[4] = 1;   /* icnam */
test = csvread(path,optn);
print "Result=",test;
```

```
Result=
                                test
                                ****

                        Mixed Type Matrix test

                            Name Party      Age    Weight
                  1         Phil     0  55.0000  180.0000
                  2         Jack     0  29.0000  200.0000
                  3     Wolfgang     1  60.0000  175.0000
                  4        Billy     0  18.0000         .
                  5          Bob     0     .     160.0000
                  6            .     1  42.0000  145.0000
```

2. Example 2: row for "Wolfgang" is shorter than others:
   The data set `csvread2.dat` is the following:

```
Name, Party, Age, Weight
"Phil", 0, 55,180
"Jack", 0, 29,200
"Wolfgang", 1
"Billy", 0, 18,
"Bob",0, ,160
" ", 1, 42, 145
```

```
path = "..\\tdata\\csvread2.dat";
optn = cons(8,1,.);
optn[1] = 0;   /* itra  */
optn[3] = 0;   /* irnam */
optn[4] = 1;   /* icnam */
test2 = csvread(path,optn);
print "Result=",test2;
```

```
Result=
```

```
                    test2
                    *****

          Mixed Type Matrix test2

          Name Party      Age    Weight
1         Phil     0  55.0000  180.0000
2         Jack     0  29.0000  200.0000
3    Wolfgang     1    .         .
4        Billy     0  18.0000     .
5          Bob     0    .      160.0000
6            .     1  42.0000  145.0000
```

3. Example 2: semicolon separator:
   The data set `csvread3.dat` is the following:

```
Name; Party; Age; Weight
"Phil"; 0; 55;180
"Jack"; 0; 29;200
"Wolfgang"; 1
"Billy"; 0; 18;
"Bob";0; ;160
" "; 1; 42; 145
```

```
path = "..\\tdata\\csvread3.dat";
optn = cons(8,1,.);
optn[1] = 0;     /* itra  */
optn[2] = ";";   /* semicolon is separator */
optn[3] = 0;     /* irnam */
optn[4] = 1;     /* icnam */
test3 = csvread(path,optn);
print "Result=",test3;
```

```
                    test3
                    *****

          Mixed Type Matrix test3

          Name Party      Age    Weight
1         Phil     0  55.0000  180.0000
2         Jack     0  29.0000  200.0000
3    Wolfgang     1    .         .
4        Billy     0  18.0000     .
5          Bob     0    .      160.0000
```

```
     6       .          1  42.0000  145.0000
```

5.2 The `recupar()` Function

< gof,tree,cltrn,prtrn,cltst,prtst > = recupar(trn,modl,optn,ord<,nom<,flt<,test> . >)

Purpose: The `recupar()` function can be used for tree analysis by recursive partitioning (Chaid, C4.5). The predictive analysis is for a nominal scaled response and ordinal or nominal predictors. The data set is stepwise partitioned into two or more subsets and forming a tree with branches corresponding to the subsets. The function is a close equivalent to the SAS Macro `TREEDISC` by W. Sarle (with some modifications of M. Friendly).

Input: **trn** must be some $N \times n$ matrix of training data for which a regression tree is to be computed by merging and splitting sets of categories.

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

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

ord an index vector specifiying those terms of the model which are scaled ordinally (sets of categories must contain adjacent levels)

nom an index vector specifiying those terms of the model which are scaled nominally (all combinations of levels may be contained in merged and splittet level sets)

flt an index vector specifiying those terms of the model which are scaled ordinally with a floating category (all categories are ordered except the first one floating one which is treated separately which is not related in size to the others and refers commonly to a missing value category).

test must be some $N_t \times n$ matrix of test data which should be scored (obtain predicted values and classification table).

Note, ordinal scaled variables which have missing values, should be specified with `flt` and not `ord`. For nominal scaled variables missing values just define an additional category.

Options Matrix Argument: Most options correspond to similar options in the SAS `treedisc` macro. The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
"alpha"	real	Numeric value in the range (0,1), adjusted significance level for using a predictor in the tree. If the significance level for the most significant (optimally merged) predictor exceeds this value, then the node is subdivided according to the predictor's merged categories. The default is 0.1.
"amerge"	real	Numeric value in the range (0,1), raw significance level for combining two categories into a compound category. The default is 0.0001.
"asplit"	real	Numeric value in the range (0,1), raw significance level for dividing one compound category that contains at least three of the original categories into its most significant binary split. The default is 0.049.
"bonf"	int	Bonferroni adjustment must be in [0,3], def=1 $p = \min(\text{Bonf.mult.} * \text{Prob}(\chi^2(r-1, k-1) > X), \text{Prob}(\chi^2(r-1, c-1) > X))$
"bran"	int	positive integer specifying the minimum number of observations in a node to qualify it for further subdivision, def=2*leaf
"chaid"	int	Use the original CHAID merging and splitting algorithm. See the remarks about cycling in the comments part of the SAS Macro TREEDISC.
"freq"	int	column number of Frequency variable in data
"fuzzy"	real	required change of the p value to avoid cycling, def=10^{-35}
"gabr"		Gabriel's adjustment to increase the power for multiple comparisons in a contingency table as suggested by Hawkins & Kass (1982)
"leaf"	int	positive integer specifying the minimum number of observations allowed in a leaf; any partition that would result in a leaf with fewer observations is rejected, def=1.
"limit"	int	maximum number of iterations, def=10
"maxdep"	int	Maximum depth (number of nodes from root to leaf) allowed in the tree, def=100

Option Name	Second Column	Meaning
"nolog"		supress some warnings in the log output
"nomord"		treat binary nominal variable as ordinal
"nomsp"	int	positive integer specifying the maximum number of categories in a compound category of a nominal variable for which splitting will be attempted, def=10
"ordsp"	int	positive integer specifying the maximum number of categories in a compound category of an ordinal variable for which splitting will be attempted, def=100
"pobs"		print table of predicted values and residials
"phis"	int	print history of tree splitting
"pinit"	int	print information of input values
"print"	int	amount of printed output, def=0

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

tree description of tree in matrix form for scoring

cltrn classification table training data

prtrn predicted values and residuals for training data

cltst classification table test data

prtst predicted values and residuals for test data

Restrictions: 1. Nominal predictors may have a category for missing values.

2. Ordinal predictors with missing values should be specified with the `flt` argument.

Relationships: split()

Examples: 1. Iris example for SAS macro `treedisc`: Four predictor variables are treated as ordinal

```
  options ls=68 ps=2000;
  options NOECHO;
#include "..\\tdata\\iris.dat"
  options ECHO;


  print "For testing with SAS macro: switch columns";
  ind = [ 3 4 1 2 5 ];
  iris = iris[,ind];


  /* there are 5 columns: col 1 until 4 with length,
     column 5 with kind of iris: values 1,2,3 */
  nr = nrow(iris); nc = ncol(iris); print "nr,nc=", nr,nc;
```

```
  print "Iris Data=",iris[1:10,];
  print "Make Test data from Training data";
  irtst = iris[26:125,];
```

```
Iris Data=
   |   PETALLEN   PETALWID   SEPALLEN   SEPALWID    SPEC_NO
-------------------------------------------------------------
 1 |         14          2         50         33          1
 2 |         56         22         64         28          3
 3 |         46         15         65         28          2
 4 |         56         24         67         31          3
 5 |         51         15         63         28          3
 6 |         14          3         46         34          1
 7 |         51         23         69         31          3
 8 |         45         15         62         22          2
 9 |         48         18         59         32          2
10 |         10          2         46         36          1
```

```
  print "Example of TREEDISC Macro: only Training data";
  modl = "5 = 1:4";
  ord = [ 1 : 4 ];
  optn = [ "print"       5 ,
           "pinit"       3 ,
           "pobs"          ,
           "phist"       2 ];
  < gof,tree,clastrn,predtrn,clastst,predtst >
         = recupar(iris,modl,optn,ord,.,.,irtst);
```

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

                  Number Valid Observations  150
                  Observations Test Data     100
                  Response Variable          Y[5]
                  N Independent Variables      4
                  N Ordinal Predictors         4
                  N Nominal Predictors         0
                  N OrdFloat Predictors        0
                  Max # Cats for Ordinal     100
                  Max # Cats for Nominal      10
                  Significance Level   0.1000000
                  Maximum Depth of Tree      100
                  Number Passes               10
```

```
Min # Obs for Split          2
Min # Obs for Leaf           1
Signif. for Merging  1.00e-004
Signif. for Splits   0.0490000


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

C1 + C2 + C3 + C4


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

Class  Level    Value

 Y[5]      3        1       2       3
 C[1]     43       10      11      12      13      14
                   15      16      17      19      30
 ..................................................


           Neff   Predictor     Type  Levels Missing

              1    PETALLEN  Ordinal      43       0
              2    PETALWID  Ordinal      22       0
              3    SEPALLEN  Ordinal      35       0
              4    SEPALWID  Ordinal      23       0


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

           Variable       Value    Nobs  Proportion
           Y[5]               1      50   33.333333
                              2      50   33.333333
                              3      50   33.333333

           C[1]              10       1    0.666667
                             11       1    0.666667
           ....................................


       Input Predictors and Response SPEC_NO (Training)
```

```
  PETALLEN PETALWID SEPALLEN SEPALWID  SPEC_NO
1  4.00000  1.00000  7.00000  12.0000        0
2  32.0000  18.0000  21.0000  7.00000  2.00000
3  22.0000  11.0000  22.0000  7.00000  1.00000
4  32.0000  20.0000  24.0000  10.0000  2.00000
5  27.0000  11.0000  20.0000  7.00000  2.00000
.........................................
```

Input Predictors and Response SPEC_NO (Test)

```
  PETALLEN PETALWID SEPALLEN SEPALWID  SPEC_NO
1  3.00000  1.00000  12.0000  14.0000        0
2  28.0000  19.0000  24.0000  9.00000  2.00000
3  23.0000  10.0000  27.0000  11.0000  1.00000
4  21.0000  11.0000  21.0000  11.0000  1.00000
5  16.0000  9.00000  18.0000  7.00000  1.00000
.........................................

   ***********************************
   Iteration 1: Best 4 Splits (Depth=1)
   ***********************************

Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
 PETALWID  Ordinal      22       3  266.9082126  3.1075e-054
 PETALLEN  Ordinal      43       3  262.0606061  1.4123e-052
 SEPALLEN  Ordinal      35       4  135.3615385  5.7061e-023
 SEPALWID  Ordinal      23       3  72.68304206  1.4219e-012

 Best Split PETALWID with prob=3.1075e-054 Chi^2=266.908

   ***********************************
   Iteration 3: Best 4 Splits (Depth=2)
   ***********************************

Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
 PETALLEN  Ordinal      21       2  26.47653061  5.3352e-006
 PETALWID  Ordinal       8       2  5.934065934  0.103956565
 SEPALLEN  Ordinal      22       3  14.44321729  0.153431493
 SEPALWID  Ordinal      14       2  1.487837962  0.999958840

Best Split PETALLEN with prob=5.33519e-006 Chi^2=26.4765

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

```
                Iteration 4: Best 4 Splits (Depth=3)
                ************************************

    Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
     PETALWID  Ordinal       8       2  48.00000000  2.9835e-011
     SEPALLEN  Ordinal      21       2  23.48936170  2.5121e-005
     SEPALWID  Ordinal      14       3  8.782978723  0.789129482
     PETALLEN  Ordinal      17       3  5.106382979  0.995198705

       Best Split PETALWID with prob=2.98353e-011 Chi^2=48


                ************************************
                Iteration 7: Best 4 Splits (Depth=3)
                ************************************

    Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
     PETALWID  Ordinal       4       2  3.000000000  0.249793550
     SEPALLEN  Ordinal       5       3  2.625000000  0.622400931
     PETALLEN  Ordinal       4       2  1.500000000  0.662014086
     SEPALWID  Ordinal       5       2  1.500000000  0.826641467

         Best Split PETALWID with prob=0.249794 Chi^2=3


                ************************************
                Iteration 8: Best 4 Splits (Depth=2)
                ************************************

    Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
     SEPALLEN  Ordinal      20       3  22.48888889  0.002236639
     PETALLEN  Ordinal      19       2  14.65185185  0.002327536
     PETALWID  Ordinal       8       2  2.896296296  0.621483327
     SEPALWID  Ordinal      12       3  6.814814815  0.813875729

      Best Split SEPALLEN with prob=0.00223664 Chi^2=22.4889


[warning] Cannot split on PETALWID only one nonzero category.
[warning] Cannot split on SEPALLEN only one nonzero category.


               ************************************
               Iteration 10: Best 2 Splits (Depth=3)
               ************************************

    Predictor     Type   Xcats  Levels        Chi^2     Adj.Prob
```

```
      PETALLEN  Ordinal       2       2  2.000000000  0.157299207
      SEPALWID  Ordinal       2       2  2.000000000  0.157299207

          Best Split PETALLEN with prob=0.157299 Chi^2=2


           ********************************************
           Tree Analysis: 11 Nodes Dependent Var: SPEC_NO
           ********************************************

Node[0]=1 Depth=1 Order=3 Tie=4 Into=Post=.
SPEC_NO value(s): 01 02 03
DV Counts: 50 50 50 Best p-value(s):  0.0000  0.0000

  Node[1]=2 Depth=2 Order=4 Tie=0 Into=1 Post=0
  PETALWID value(s): 01 02 03 04 05 06
  DV Counts: 50 0 0

  Node[2]=3 Depth=2 Order=6 Tie=0 Into=2 Post=0
  PETALWID value(s): 10 11 12 13 14 15 16 17
  DV Counts: 0 49 5 Best p-value(s):  0.0000  0.1040

    Node[3]=5 Depth=3 Order=8 Tie=0 Into=2 Post=0
    PETALLEN value(s): 30 33 35 36 37 38 39 40 41 42 43 44 45 46
                       47 48 49
    DV Counts: 0 47 1 Best p-value(s):  0.0000  0.0000

      Node[4]=7 Depth=4 Order=9 Tie=0 Into=2 Post=0
      PETALWID value(s): 10 11 12 13 14 15 16
      DV Counts: 0 47 0

      Node[5]=8 Depth=4 Order=10 Tie=0 Into=3 Post=0
      PETALWID value(s): 17
      DV Counts: 0 0 1

    Node[6]=6 Depth=3 Order=11 Tie=0 Into=3 Post=0
    PETALLEN value(s): 50 51 56 58
    DV Counts: 0 2 4 Best p-value(s):  0.2498  0.6224

  Node[7]=4 Depth=2 Order=14 Tie=0 Into=3 Post=0
  PETALWID value(s): 18 19 20 21 22 23 24 25
  DV Counts: 0 1 45 Best p-value(s):  0.0022  0.0023

    Node[8]=9 Depth=3 Order=15 Tie=0 Into=3 Post=0
    SEPALLEN value(s): 56 57 58
    DV Counts: 0 0 5
```

```
Node[9]=10 Depth=3 Order=16 Tie=3 Into=Post=.
SEPALLEN value(s): 59
DV Counts: 0 1 1 Best p-value(s):  0.1573  0.1573

Node[10]=11 Depth=3 Order=17 Tie=0 Into=3 Post=0
SEPALLEN value(s): 60 61 62 63 64 65 67 68 69 71 72 73 74 76
                   77 79
DV Counts: 0 0 39
```

Goodness of Fit

Dense Column Vector (nrow=15)

C		ErrorCode	CompTime	NumberNodes	MaxValues	ClassMissTr
		0	0	11.000000	17.000000	2.0000000

C		MissClassTr	AccuracyTr	ClassMissTt	MissClassTt	AccuracyTt
		2.0000000	98.648649	1.0000000	0	100.00000

C		unused	unused	unused	unused	unused
		.	.	.	.	.

OUTTREE Data Set for Scoring

Dense Matrix (11 by 30)

	Node	From	Variable	If/Else	Depth
Node01	0	1.0000000	5.0000000	0	0
Node02	1.0000000	2.0000000	2.0000000	1.0000000	2.0000000
Node03	2.0000000	3.0000000	2.0000000	2.0000000	2.0000000
Node04	3.0000000	5.0000000	1.0000000	1.0000000	3.0000000
Node05	4.0000000	7.0000000	2.0000000	1.0000000	4.0000000
Node06	5.0000000	8.0000000	2.0000000	2.0000000	4.0000000
Node07	6.0000000	6.0000000	1.0000000	3.0000000	3.0000000
Node08	7.0000000	4.0000000	2.0000000	3.0000000	2.0000000
Node09	8.0000000	9.0000000	3.0000000	1.0000000	3.0000000
Node10	9.0000000	10.000000	3.0000000	2.0000000	3.0000000
Node11	10.000000	11.000000	3.0000000	2.0000000	3.0000000

	Ties	Into	Post	Prob1	Prob2

Node01	3.0000000	.	.	0	0
Node02	0	1.0000000	1.0000000	.	.
Node03	0	2.0000000	0.9074074	5.34e-006	0.1039566
Node04	0	2.0000000	0.9791667	2.98e-011	2.51e-005
Node05	0	2.0000000	1.0000000	.	.
Node06	0	3.0000000	1.0000000	.	.
Node07	0	3.0000000	0.6666667	0.2497935	0.6224009
Node08	0	3.0000000	0.9782609	0.0022366	0.0023275
Node09	0	3.0000000	1.0000000	.	.
Node10	2.0000000	.	.	0.1572992	0.1572992
Node11	0	3.0000000	1.0000000	.	.

	Count1	Count2	Count3	Value01	Value02
Node01	50.000000	50.000000	50.000000	0	1.0000000
Node02	50.000000	0	0	0	1.0000000
Node03	0	49.000000	5.0000000	6.0000000	7.0000000
Node04	0	47.000000	1.0000000	9.0000000	10.000000
Node05	0	47.000000	0	6.0000000	7.0000000
Node06	0	0	1.0000000	13.000000	.
Node07	0	2.0000000	4.0000000	26.000000	27.000000
Node08	0	1.0000000	45.000000	14.000000	15.000000
Node09	0	0	5.0000000	13.000000	14.000000
Node10	0	1.0000000	1.0000000	16.000000	.
Node11	0	0	39.000000	17.000000	18.000000

	Value03	Value04	Value05	Value06	Value07
Node01	2.0000000	.	.	.	.
Node02	2.0000000	3.0000000	4.0000000	5.0000000	.
Node03	8.0000000	9.0000000	10.000000	11.000000	12.000000
Node04	11.000000	12.000000	13.000000	14.000000	15.000000
Node05	8.0000000	9.0000000	10.000000	11.000000	12.000000
Node06	.	.	.	.	.
Node07	32.000000	34.000000	.	.	.
Node08	16.000000	17.000000	18.000000	19.000000	20.000000
Node09	15.000000	.	.	.	.
Node10	.	.	.	.	.
Node11	19.000000	20.000000	21.000000	22.000000	24.000000

	Value08	Value09	Value10	Value11	Value12
Node01	.	.	.	.	.

```
Node02 |      .          .          .          .          .
Node03 |   13.000000     .          .          .          .
Node04 |   16.000000  17.000000  18.000000  19.000000  20.000000
Node05 |      .          .          .          .          .
Node06 |      .          .          .          .          .
Node07 |      .          .          .          .          .
Node08 |   21.000000     .          .          .          .
Node09 |      .          .          .          .          .
Node10 |      .          .          .          .          .
Node11 |   25.000000  26.000000  28.000000  29.000000  30.000000


       |     Value13    Value14    Value15    Value16    Value17
     ------------------------------------------------------------
Node01 |      .          .          .          .          .
Node02 |      .          .          .          .          .
Node03 |      .          .          .          .          .
Node04 |   21.000000  22.000000  23.000000  24.000000  25.000000
Node05 |      .          .          .          .          .
Node06 |      .          .          .          .          .
Node07 |      .          .          .          .          .
Node08 |      .          .          .          .          .
Node09 |      .          .          .          .          .
Node10 |      .          .          .          .          .
Node11 |   31.000000  32.000000  33.000000  34.000000     .

          Legend for if/else: 1: if, 2: else, 3: end


 Classification Table (Training): Accuracy=98.6486 %
 ***************************************************

              Dense Matrix (4 by 4)

       |     Pred_01    Pred_02    Pred_03      Total
     ---------------------------------------------------
Data_01 |   50.000000          0          0  50.000000
Data_02 |          0  47.000000  2.0000000  49.000000
Data_03 |          0          0  49.000000  49.000000
  Total |   50.000000  47.000000  51.000000  148.00000

             Frequency of Misclassification: 2
            Frequency of Classified as Missing: 2


              Predicted Values of Training Data
              *********************************
```

Dense Matrix (150 by 3)

	Ypred	Ydata	Residual
1	0	0	0
2	2.0000000	2.0000000	0
3	1.0000000	1.0000000	0
4	2.0000000	2.0000000	0
5	2.0000000	2.0000000	0
6	0	0	0
7	2.0000000	2.0000000	0
8	1.0000000	1.0000000	0
9	.	1.0000000	.
10	0	0	0
11	1.0000000	1.0000000	0
12	2.0000000	1.0000000	-1.0000000
13	2.0000000	2.0000000	0
14	1.0000000	1.0000000	0
15	2.0000000	2.0000000	0

.....

Classification Table (Test): Accuracy=100 %

****

Dense Matrix (4 by 4)

	Pred_01	Pred_02	Pred_03	Total
Data_01	36.000000	0	0	36.000000
Data_02	0	34.000000	0	34.000000
Data_03	0	0	29.000000	29.000000
Total	36.000000	34.000000	29.000000	99.000000

Frequency of Misclassification: 0

Frequency of Classified as Missing: 1

Predicted Values of Test Data

****

Dense Matrix (100 by 3)

	Ypred	Ydata	Residual
1	0	0	0

```
 2 |   2.0000000  2.0000000          0
 3 |   1.0000000  1.0000000          0
 4 |   1.0000000  1.0000000          0
 5 |   1.0000000  1.0000000          0
 6 |          0          0          0
 7 |      .       2.0000000     .
 8 |   1.0000000  1.0000000          0
 9 |   2.0000000  2.0000000          0
10 |   2.0000000  2.0000000          0
11 |          0          0          0
12 |          0          0          0
13 |   1.0000000  1.0000000          0
14 |   2.0000000  2.0000000          0
15 |          0          0          0
.....................................
```

2. Iris example for SAS macro `treedisc`: Four predictor variables are treated as *float ordinal*

```
modl = "5 = 1:4";
flt = [ 1 : 4 ];
optn = [ "print"        5 ,
         "pinit"        3 ,
         "pobs"           ,
         "phist"        2 ];
< gof,tree,clastrn,predtrn > = recupar(iris,modl,optn,.,.,flt);
```

We only report part of the results:

```
*******************************************
Tree Analysis: 11 Nodes Dependent Var: SPEC_NO
*******************************************

Node[0]=1 Depth=1 Order=3 Tie=4 Into=Post=.
SPEC_NO value(s): 01 02 03
DV Counts: 50 50 50 Best p-value(s):  0.0000  0.0000

  Node[1]=2 Depth=2 Order=4 Tie=0 Into=1 Post=0
  PETALWID value(s): 01 02 03 04 05 06
  DV Counts: 50 0 0

  Node[2]=3 Depth=2 Order=6 Tie=0 Into=2 Post=0
  PETALWID value(s): 10 11 12 13 14 15 16 17
  DV Counts: 0 49 5 Best p-value(s):  0.0000  0.1931
```

```
  Node[3]=5 Depth=3 Order=8 Tie=0 Into=2 Post=0
  PETALLEN value(s): 30 33 35 36 37 38 39 40 41 42 43 44 45 46
                     47 48 49
  DV Counts: 0 47 1 Best p-value(s):  0.0000  0.2654

    Node[4]=7 Depth=4 Order=9 Tie=0 Into=2 Post=0
    PETALWID value(s): 10 11 12 13 14 15 16
    DV Counts: 0 47 0

    Node[5]=8 Depth=4 Order=10 Tie=0 Into=3 Post=0
    PETALWID value(s): 17
    DV Counts: 0 0 1

  Node[6]=6 Depth=3 Order=11 Tie=0 Into=3 Post=0
  PETALLEN value(s): 50 51 56 58
  DV Counts: 0 2 4 Best p-value(s):  0.3916  0.6823

Node[7]=4 Depth=2 Order=14 Tie=0 Into=3 Post=0
PETALWID value(s): 18 19 20 21 22 23 24 25
DV Counts: 0 1 45 Best p-value(s):  0.0062  0.6857

  Node[8]=9 Depth=3 Order=15 Tie=0 Into=3 Post=0
  SEPALLEN value(s): 56 57 58
  DV Counts: 0 0 5

  Node[9]=10 Depth=3 Order=16 Tie=3 Into=Post=.
  SEPALLEN value(s): 59
  DV Counts: 0 1 1 Best p-value(s):  0.1573  0.1573

  Node[10]=11 Depth=3 Order=17 Tie=0 Into=3 Post=0
  SEPALLEN value(s): 60 61 62 63 64 65 67 68 69 71 72 73 74 76
                     77 79
  DV Counts: 0 0 39


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

                 Dense Column Vector (nrow=15)

C |    ErrorCode    CompTime NumberNodes   MaxValues ClassMissTr
                0           0   11.000000   17.000000   2.0000000

C |  MissClassTr  AccuracyTr ClassMissTt MissClassTt  AccuracyTt
       2.0000000   98.648649      .           .           .
```

```
C |       unused      unused      unused      unused      unused
          .           .           .           .           .


 Classification Table (Training): Accuracy=98.6486 %
 ***************************************************

               Dense Matrix (4 by 4)

        |     Pred_01    Pred_02    Pred_03      Total
      ------------------------------------------------
Data_01 |   50.000000          0          0  50.000000
Data_02 |          0  47.000000  2.0000000  49.000000
Data_03 |          0          0  49.000000  49.000000
  Total |   50.000000  47.000000  51.000000  148.00000

              Frequency of Misclassification: 2
            Frequency of Classified as Missing: 2
```

3. Iris example for SAS macro `treedisc`: Four predictor variables are treated as nominal

```
print "Example of TREEDISC Macro: only Training data: Nominal Vars";
modl = "5 = 1:4";
nom = [ 1 : 4 ];
optn = [ "print"        5 ,
         "pinit"        3 ,
         "pobs"           ,
         "phist"        2 ];
< gof,tree,clastrn,predtrn > = recupar(iris,modl,optn,.,nom);
```

We only report part of the results:

```
          *********************************************
          Tree Analysis: 10 Nodes Dependent Var: SPEC_NO
          *********************************************

Node[0]=1 Depth=1 Order=3 Tie=4 Into=Post=.
SPEC_NO value(s): 01 02 03
DV Counts: 50 50 50 Best p-value(s):  0.0000  0.0000

  Node[1]=2 Depth=2 Order=4 Tie=0 Into=1 Post=0
  PETALWID value(s): 01 02 03 04 05 06
  DV Counts: 50 0 0

  Node[2]=3 Depth=2 Order=5 Tie=0 Into=3 Post=0
```

```
PETALWID value(s): 18 19 20 21 22 23 24 25
DV Counts: 0 1 45 Best p-value(s):  0.2606  0.6857

Node[3]=4 Depth=2 Order=9 Tie=0 Into=2 Post=0
PETALWID value(s): 10 11 12 13 14 15 16 17
DV Counts: 0 49 5 Best p-value(s):  0.0468  0.5475

  Node[4]=5 Depth=3 Order=11 Tie=0 Into=2 Post=0
  PETALLEN value(s): 45
  DV Counts: 0 7 1 Best p-value(s):  0.0327  0.2381

    Node[5]=9 Depth=4 Order=12 Tie=0 Into=3 Post=0
    PETALWID value(s): 17
    DV Counts: 0 0 1

    Node[6]=10 Depth=4 Order=13 Tie=0 Into=2 Post=0
    PETALWID value(s): 13 15 16
    DV Counts: 0 7 0

  Node[7]=6 Depth=3 Order=14 Tie=0 Into=3 Post=0
  PETALLEN value(s): 56 58
  DV Counts: 0 0 2

  Node[8]=7 Depth=3 Order=15 Tie=0 Into=2 Post=0
  PETALLEN value(s): 30 33 35 36 37 38 39 40 41 42 43 44 46 47
                     48 49
  DV Counts: 0 40 0

  Node[9]=8 Depth=3 Order=16 Tie=3 Into=Post=.
  PETALLEN value(s): 50 51
  DV Counts: 0 2 2 Best p-value(s):  0.1353  0.2615


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

                Dense Column Vector (nrow=15)

C |    ErrorCode    CompTime NumberNodes   MaxValues ClassMissTr
               0   1.0000000   10.000000   16.000000   4.0000000

C |  MissClassTr  AccuracyTr ClassMissTt MissClassTt  AccuracyTt
       1.0000000   99.315068           .           .           .

C |       unused      unused      unused      unused      unused
          .           .           .           .           .
```

```
 Classification Table (Training): Accuracy=99.3151 %
 ***************************************************

                Dense Matrix (4 by 4)

        |     Pred_01    Pred_02    Pred_03      Total
      ------------------------------------------------
 Data_01 |   50.000000          0          0  50.000000
 Data_02 |          0  47.000000  1.0000000  48.000000
 Data_03 |          0          0  48.000000  48.000000
   Total |   50.000000  47.000000  49.000000  146.00000

               Frequency of Misclassification: 1
             Frequency of Classified as Missing: 4
```

4. Iris example for SAS macro `treedisc`: Three of the four predictor variables have missing values and are treated as *float ordinal*:

```
  options ls=68 ps=2000;
  options NOECHO;
#include "..\\tdata\\iris.dat"
  options ECHO;

  /* there are 5 columns: col 1 until 4 with length,
     column 5 with kind of iris: values 1,2,3 */
  nr = nrow(iris); nc = ncol(iris); print "nr,nc=", nr,nc;


  /* set to missing values */
  iris[ 5,1] =  iris[ 6,1] =  iris[ 7,1] =  iris[ 8,1] = .;
  iris[13,2] =  iris[14,2] =  iris[15,2] =  iris[16,2] = .;
  iris[21,3] =  iris[22,3] =  iris[23,3] =  iris[24,3] = .;
  iris[29,1] =  iris[30,1] =  iris[31,1] =  iris[32,1] = .;
  iris[37,2] =  iris[38,2] =  iris[39,2] =  iris[40,2] = .;
  iris[45,3] =  iris[46,3] =  iris[47,3] =  iris[48,3] = .;


  /* for testing with SAS macro: switch columns */
  ind = [ 3 4 1 2 5 ];
  iris = iris[,ind];
  print "Iris Data=",iris[1:10,];

  print "Example of TREEDISC Macro: only Training data: Float Order";
  modl = "5 = 1:4";
  flt = [ 1 : 4 ];
```

```
optn = [ "print"        5 ,
         "pinit"        3 ,
         "pobs"           ,
         "phist"        2 ];
< gof,tree,clastrn,predtrn > = recupar(iris,modl,optn,.,.,flt);
```

With most functions in CMAT the missing value categories of class variables are located at the end, here we move them to the front.

```
Missing value category moved to first for the following effects:

                PETALLEN  SEPALLEN  SEPALWID


            Neff   Predictor     Type  Levels Missing

               1    PETALLEN OrdFloat      42       8
               2    PETALWID OrdFloat      22       0
               3    SEPALLEN OrdFloat      36       8
               4    SEPALWID OrdFloat      24       8
```

We only report the tree and classification results:

```
          ********************************************
          Tree Analysis: 11 Nodes Dependent Var: SPEC_NO
          ********************************************

Node[0]=1 Depth=1 Order=3 Tie=4 Into=Post=.
SPEC_NO value(s): 01  02  03
DV Counts: 50 50 50 Best p-value(s):  0.0000  0.0000

  Node[1]=2 Depth=2 Order=4 Tie=0 Into=1 Post=0
  PETALWID value(s): 01  02  03  04  05  06
  DV Counts: 50 0 0

  Node[2]=3 Depth=2 Order=6 Tie=0 Into=2 Post=0
  PETALWID value(s): 10  11  12  13  14  15  16  17
  DV Counts: 0 49 5 Best p-value(s):  0.0000  0.0679

    Node[3]=5 Depth=3 Order=8 Tie=0 Into=2 Post=0
    PETALLEN value(s):  .  30  33  35  36  37  38  39  40  41  42
                       43  44  45  46  47  48  49
    DV Counts: 0 47 1 Best p-value(s):  0.0000  0.0001

      Node[4]=7 Depth=4 Order=9 Tie=0 Into=2 Post=0
      PETALWID value(s): 10  11  12  13  14  15  16
```

```
    DV Counts: 0 47 0

    Node[5]=8 Depth=4 Order=10 Tie=0 Into=3 Post=0
    PETALWID value(s): 17
    DV Counts: 0 0 1

  Node[6]=6 Depth=3 Order=11 Tie=0 Into=3 Post=0
  PETALLEN value(s): 50  51  56  58
  DV Counts: 0 2 4 Best p-value(s):  0.3916  0.6224

Node[7]=4 Depth=2 Order=14 Tie=0 Into=3 Post=0
PETALWID value(s): 18  19  20  21  22  23  24  25
DV Counts: 0 1 45 Best p-value(s):  0.0000  0.0043

  Node[8]=9 Depth=3 Order=15 Tie=0 Into=3 Post=0
  SEPALLEN value(s):  .  56  57  58
  DV Counts: 0 0 7

  Node[9]=10 Depth=3 Order=16 Tie=0 Into=2 Post=0
  SEPALLEN value(s): 59
  DV Counts: 0 1 0

  Node[10]=11 Depth=3 Order=17 Tie=0 Into=3 Post=0
  SEPALLEN value(s): 60  61  62  63  64  65  67  68  69  71  72
                     73  74  76  77  79
  DV Counts: 0 0 38


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

                Dense Column Vector (nrow=15)

C |    ErrorCode    CompTime NumberNodes   MaxValues ClassMissTr
               0           0   11.000000   18.000000           0

C |  MissClassTr  AccuracyTr ClassMissTt MissClassTt  AccuracyTt
       2.0000000   98.666667      .           .           .

C |       unused      unused      unused      unused      unused
          .           .           .           .           .


 Classification Table (Training): Accuracy=98.6667 %
 ***************************************************

                Dense Matrix (4 by 4)
```

	Pred_01	Pred_02	Pred_03	Total
Data_01	50.000000	0	0	50.000000
Data_02	0	48.000000	2.0000000	50.000000
Data_03	0	0	50.000000	50.000000
Total	50.000000	48.000000	52.000000	150.00000

Frequency of Misclassification: 2

5.3 The `insert()` Function

d = insert(a,b,r<,c>)

Purpose: The `insert()` function can be used for inserting rows or columns `b` into a vector or matrix `a`. The location for the insert is specified either by row `r` or column `c`. If `a` is an $n \times m$ matrix, `r` must be integer in $[0, n+1]$ and `c` must be integer in $[0, m+1]$. Either `r` or `c` must be zero, indicating that the insertion concerns the other argument. The column argument is optional and defaults to zero.

Input: **a** $r_a \times c_a$ target object `a` into which $r_b \times c_b$ object `b` is to be inserted and `a` can be a matrix or a row or column vector.

b $r_b \times c_b$ object `b` which is to be inserted into $r_a \times c_a$ object `a` and `b` can be a matrix or a row or column vector.

r specifies row number of `a` at which `b` is to be inserted, or $r = 0$ for columnwise insertion at $c > 0$

c specifies column number of `a` at which `b` is to be inserted, or $c = 0$ for rowwise insertion at $r > 0$ (defaults to $c = 0$)

Output: The only return object `d` contains either `a_1 -> b -> a_2` or `a_1 |> b |> a_2`.

Restrictions: 1. The specified integer arguments r and c must be valid row or column numbers of `a`.

2. Either r or c must be zero.

Relationships:

Examples: 1. Example in SAS/IML Manual for INSERT(), p. 352:

```
a = [ 1 2 , 3 4 ];
b = [ 5 6 , 7 8 ];
print "C=",c= insert(a,b,2,0);
print "D=",d= insert(a,b,0,3);
```

```
C=
   |   1   2
-----------
 1 |   1   2
 2 |   5   6
 3 |   7   8
 4 |   3   4
```

```
D=
   |   1   2   3   4
--------------------
 1 |   1   2   5   6
 2 |   3   4   7   8
```

```
print "Vector Example";
a = [ 1 2 3 4 ]‘;
b = 5;
print "C=",c= insert(a,b,2,0);
```

```
Vector Example

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

2. First argument `a` can be scalar:

```
a = 5.;
b = [ 1 2 3 4 ];
print "C3=",c3 = insert(a,b,0,1);
print "C4=",c4 = insert(a,b,0,2);
```

```
C3=
   |         1         2         3         4         5
-----------------------------------------------------------
 1 |   1.00000    2.0000    3.0000    4.0000    5.0000

C4=
   |         1         2         3         4         5
-----------------------------------------------------------
 1 |    5.0000   1.00000    2.0000    3.0000    4.0000
```

5.4 The `remove()` Function

d = remove(a,indv)

Purpose: The `remove()` function can be used to remove entries from a matrix or vector `a`. After calling the `remove()` function usually a call of `shape()` is necessary for reshaping an output row vector `d`.

Input: **a** $r_a \times c_a$ object `a` from which entries are removed.

indv index vector of nonmissing integer values specifying the locations of entries that are to be removed from `a`. The entries of `indv` must be in the range of $[1, N]$ where $N = r_a * c_a$ is the number of all entries of input object `a`.

Output: d is either a row vector or scalar or missing value. If all entries are removed from the input object, a missing value d is returned.

Restrictions: 1. The index vector `indv` must contain nonmissing integer values in the range of $[1, N]$ where $N = r_a * c_a$ is the number of all entries of input object `a`.

Relationships:

Examples: 1. Example in SAS/IML Manual for REMOVE():

```
a = [ 5 6 , 7 8 ];
print "C=", c = remove(a,3);
```

```
C=
   |   1   2   3
---------------
 1 |   5   6   8
```

```
a = [ 5 6 , 7 8 ];
ind = [ 3 2 3 1 ];
print "D=", d = remove(a,ind);
```

```
D= 8
```

```
print "Vector Example";
a = [ 1 2 3 4 ]`;
print "C=",c= remove(a,2);
```

```
Vector Example

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

2. Remove all entries: Result is missing value

```
a = [ 1 2 3 4 ]‘;
ind = [ 1 2 3 4 ];
print "C3=",c3= remove(a,ind);


C3= .
```

5.5 The `encryp2()` Function

ostr = encryp2(istr<,pwd<,optn>>)

Purpose: The `encryp2()` function can be used for the save encryption of scalar or vector string data. For ways how to specify the password see the `encrypt()` function. Whereas the `encrypt()` function encrypts plaintext in files and directories the `encryp2()` function encrypts scalar or vector string data objects. Most of the algorithms implemented here create ciphertext which can be decrypted again using the `decryp2()` function. However, this function also provides a number of hash algorithms like SHA-2 and SHA-3 which create ciphertext that cannot be decrypted again, but are very useful especially for password encryption. Note, no passwords are used for the encryption with the hash algorithms SHA-2 and SHA-3.

Input: istr This must be either a string scalar or a vector of n plaintext string data which should be encrypted to ciphertext.

pwd If this argument is specified it must be a string, no more than 256 characters are used. If this argument is not specified, a 256 character password is machine generated. If no password string is specified a password file is written either to a specified location (first entry of the `optn` input argument) or to the `cmat/save/encrypt/` directory. This file must be available for the decryption.

optn This argument maybe specified as a single string or a vector of strings. The meaning of the vector entries is as follows:

Entry	Option	Meaning
1	string	path name to password file
2	"own"	own encryption method
	"aes"	Advanced Encryption Standard (Rijndael; Hellekalek & Wegenkittel, 2003))
	"anu"	Anubis (Barreto & Rijmen, 2000)
	"cam"	Camellia (Ichikawa, Matsui, Nakajima, & Tokita, Mitsubishi, 2001)
	"cle"	Clefia (Sony, 2007)
	"hr3"	Hierocrypt-3 (Myasnikov)
	"sc2"	sc2000 (Myasnikov)
	"spe"	SPEED (Zheng)
	"2fi"	Twofish (Gladman, 1999)
	"unc"	CipherUnicorn (Myasnikov)
	"sh2"	SHA-2 secure hash algorithm (Christophe Devine)
	"sh3"	SHA-3 SKEIN hash algorithm (Doug Whiting)
3	"abo"	abort encryption if password file with same name exists (for savety reason this is the default)
	"wri"	overwrite existing password file
4	"sub"	process files in all subdirectories (default)
	"nos"	do not process any subdirectories
5	"noh"	hash: output in string form (length=32)
	"hex"	hash: hexadecimal output (length=64) (default)
6	"nop"	hash: no printed output (default)
	"pri"	hash: printed output (default)

Output: The `ostr` return object contains the encrypted scalar or vector string data. If no password is specified with input argument `pwd` a randonly generated password file is written either to a location specified with the first entry of the `optn` input argument or to the default location `cmat/save/encrypt/` with a file name derived from the `ofil` input argument.

Restrictions:

1. Pathnames to password files must not contain any white space.
2. If the password (file) is lost, an encrypted file cannot be decrypted anymore.
3. The password file is written in binary mode.
4. Note, that the `optn` input argument used for decryption must be compatible with that used for encryption.

Relationships: decryp2(), encrypt(), decrypt()

Examples: 1. "Own" method:

```
passw = "bully";
istr = [ "abc",
```

```
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "own" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

2. Advanced Encryption Standard method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "aes" ];
ostr = encryp2(istr,passw,optn);
print "Length=",strlen(ostr[1]),strlen(ostr[2]);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

3. Anubis method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "anu" ];
ostr = encryp2(istr,passw,optn);
print "Length=",strlen(ostr[1]),strlen(ostr[2]);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

4. Camellia method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "cam" ];
```

```
ostr = encryp2(istr,passw,optn);
print "Length=",strlen(ostr[1]),strlen(ostr[2]);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

5. Clefia method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "cle" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

6. HieroCrypt method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "hcr" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

7. SC2000 method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "sc2" ];
ostr = encryp2(istr,passw,optn);
```

```
istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

8. SPEED method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "spe" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

9. Twofish method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "2fi" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

10. Unicorn method:

```
passw = "bully";
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "uni" ];
ostr = encryp2(istr,passw,optn);


istr = ostr;
ostr = decryp2(istr,passw,optn);
print "Decrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);
```

11. SHA-2 stack method (cannot be decrypted):

```
istr = [ "abc",
         "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" ];
optn = [ ., "sh2", ., ., ., "pri" ];
ostr = encryp2(istr,.,optn);


print "Encrypted: Ostr=",ostr;
print "Length=",strlen(ostr[1]),strlen(ostr[2]);


                  ************************
                  Hash in Hexadecimal Form
                  ************************

 1 ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

 2 248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1


Encrypted: Ostr=
   |                                                  1
-------------------------------------------------------
 1 |ba7816bf8f01cfea414140de5dae2223b00361a396177a9
   |cb410ff61f20015ad
 2 |248d6a61d20638b8e5c026930c3e6039a33ce45964ff216
   |7f6ecedd419db06c1
Length= 64 64
```

For that input SHA-2 must create the following hexadecimal output:

```
print "Results are hexadecimal:";
res = [ "ba7816bf8f01cfea414140de5dae2223" ,
        "b00361a396177a9cb410ff61f20015ad" ,
        "248d6a61d20638b8e5c026930c3e6039" ,
        "a33ce45964ff2167f6ecedd419db06c1" ];
```

5.6 The `decryp2()` Function

ostr = decryp2(istr<,pwd<,optn>>)

Purpose: The `decryp2()` function can be used to decrypt formerly encrypted string data. It is assumed that the correct password (which has been used with the encryption of the file) is still available. For ways how to specify the password see the `decrypt()` function. Whereas the `decrypt()` function decrypts ciphertext in files and directories the `decryp2()` function decrypts scalar or vector string data objects.

The function `decryp2()` is useful only for ciphertext which was generated by function `encryp2()`. Note, ciphertext created using the hash algorithms SHA-2 and SHA-3 with functions `encrypt()` and `encryp2()` cannot be decrypted.

Input: istr This must be either a string scalar or a vector of n string data consisting of formerly encrypted *ciphertext*.

pwd If this argument is specified it must be a string which is the same as that which has been used to obtain the encrypted file `ifil`. If this argument is not specified, it is assumed that a 256 character password file is available either at a specified location (first entry of the `optn` input argument) or at the `cmat/save/encrypt/` directory.

optn This argument maybe specified as a single string or a vector of strings. For the meaning of the vector entries see table below. Note, that the options used for decryption must be compatible with those used for encryption.

Entry	Option	Meaning
1	string	path name to password file
2	"own"	own encryption method
	"aes"	Advanced Encryption Standard (Rijndael; Hellekalek & Wegenkittel, 2003))
	"anu"	Anubis (Barreto & Rijmen, 2000)
	"cam"	Camellia (Ichikawa, Matsui, Nakajima, & Tokita, Mitsubishi, 2001)
	"cle"	Clefia (Sony, 2007)
	"hr3"	Hierocrypt-3 (Myasnikov)
	"sc2"	sc2000 (Myasnikov)
	"spe"	SPEED (Zheng)
	"2fi"	Twofish (Gladman, 1999)
	"unc"	CipherUnicorn (Myasnikov)
3	"abo"	abort encryption if password file with same name exists (for savety reason this is the default)
	"wri"	overwrite existing password file

Note, that the hash algorithms SHA-2 and SHA-3 can only be used for encryption but not for decryption.

Output: The `ostr` return object should contain the decrypted scalar or vector string data which were input to the former run of the `encryp2()` function.

Restrictions:
1. Pathnames to password files must not contain any white space.
2. If the password (file) is lost, an encrypted file cannot be decrypted anymore.
3. The password and the encrypted files are written in binary mode.
4. Note, that the `optn` input argument used for decryption must be compatible with that used for encryption.

Relationships: encryp2(), decrypt(), encrypt()

Examples: See the `encryp2()` function (encryp2()) for a set of examples.

5.7 The `mixregv()` Function

< gof,parm,cov,resi > = mixregv(data,modl,optn,covar<,rand<,errvar>>)

Purpose: The `mixregv()` function can be used for estimating optimal parameter estimates for mixed effects regression (MRMs, location scale models). The model is useful for longitudinal regression analysis (e.g. a group of smokers at time points) estimating the between subjects (BS) and within subjects (WS) variance.

The data are so-called *ecological momentary assessments* (EMA) data (x_{ij}, u_i, w_{ij}) where $i = 1, \ldots, N$ subjects are evaluated at $j = 1, \ldots, n_i$ occasions, where the $N \times n_x$ matrix contains the regressors usually (but not necessarily) including an intercept (a column of 1's), the $N \times n_u$ matrix contains the data for the "between subjects" and the $N \times n_w$ matrix the data for "within subjects" variance. The regression model is

$$y_{ij} = x'_{ij}\beta + v_i + \epsilon_{ij}$$

where β is the p vector of regression coefficients, the random subject effect v_i indicates the influence of subject i on his/her repeated assessments. The population distribution of the random effects v_i is usually assumed normal with zero mean and (between subjects, BS) variance σ_v^2

$$\sigma_{v_i}^2 = exp(u'_i\alpha)$$

If for a particular effect of u the coefficient $\alpha > 0$, the the BS variance increases as this efect of u increases, and vice versa when $\alpha < 0$.

The errors ϵ_{ij} are assmed to be normal distributed with zero mean and (within subjects, WS) variance σ_ϵ^2

$$\sigma_{\epsilon_{ij}}^2 = exp(w'_{ij}\tau)\tau$$

The $n_x + n_v + n_w$ parameter vector (β, α, τ) does not vary with i or j. Normally, but not necessarily, both the u_i and w_{ij} include the intercept (column of 1's) for the BS and WS variances.

If the model is modified to represent the random effects in standardized form, either $q = 2$ or $q = 3$ additional parameters are to be estimated. For more details see Hedeker (2012), Hedeker, Mermelstein, & Demirtas (2008).

The `mixregv()` function computes optimal parameter estimates for up to three models

1. Model Without Scale Parameters: $n_x + n_u + 1$ estimates
2. Model With Scale Parameters: $n_x + n_u + n_w$ estimates

3. Model With Random Scale: $n_x + n_u + n_w + q$ estimates, where $q = 1, 2, 3$ depends on the functional form of the relationship between the random location (mean) and the WS variance:

   q=1 none

   q=2 linear

   q=3 quadratic

The number s of computed solutions depends on the number n_w of error variances:

$$s = \begin{cases} 1 & n_w = 0 : \text{model 1 only} \\ 2 & n_w > 1 : \text{models 1 and 3} \\ 3 & n_w \geq 1 : \text{models 1,2,3} \end{cases}$$

This implementation is based on the Fortran90 program `MIXREGLS` by Don Hedeker (2012).

Input: data The data set must be an $N \times n$ matrix of integer or real data where the N rows relate to observations and the n columns to variables which are assumed to be interval scaled. The data matrix must contain a column specifying an ID variable for groups of observations (subjects). The model statement relates to the column numbers of this data set.

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

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content. The option statement is not optional, since at least the column number of the ID variable must be specified.

covar specifies the x effect numbers in the model statement which are estimated as fixed effects; a value of -1 indicates that an intercept variable should be added to the model; can be missing value if no fixed effects are estimated

rand specifies the u effect numbers in the model statement which are estimated as random effects; a value of -1 indicates that an intercept variable should be added to the model; can be missing value if no random effects are estimated

errvar specifies the w effect numbers in the model statement which are estimated as error variances; a value of -1 indicates that an intercept variable should be added to the model; can be missing value if no error variances are estimated in the model

Note that the index vectors for covariate, random effects, and the error covariances relate to the effect numbers.

Option	Second Column	Meaning
"adapt"		use adaptive quadrature formula
"alpha"	real	probability for confidence limits, default $\alpha = .05$
"baypri"		print empirical Bayes estimates
"conv"	real	convergence tolerance for Hedeker algorithm
"covrel"	string	relationship bewtween random location and WS variance: "no", "lin" (default), "qua"
"freq"	int	column number of frequency variable
"idvar"	int	column number of ID variable
"maxit"	int	maximum number of iterations, default=200
"nquad"	int	(small) number q of quadrature points, default $q = 10$
"pcov"		print cov and corr matrix of parameter estimates
"pobs"		print observationwise: predicted values and residuals
"popt"		print optimization history
"print"	int	amount of printed output, def=2
"nopr"		no printed output
"pres"		print predicted values and residuals
"scal"		standardize data ($\mu = 0, \sigma = 1$)
"sing"	real	singularity criterion, def=1.e-8
"wgt"	int	column number of weight variable

At least the column number of the ID variable must be specified.

Output: **gof** is a $10 \times s$ matrix of scalar results for the s solutions

parm is a $p \times s * 6$ matrix of parameter estimates, asymptotic standard errors and related measures:

1. parameter estimates
2. asymptotic standard errors
3. z values
4. p values
5. lower confidence limits
6. upper confidence limits

cov is

1. either for $s = 1$: a $p \times p$ symmetric matrix of the covariances of the p parameter estimates (inverse Hessian matrix of the Likelihood function)
2. or for $s > 1$: $\begin{pmatrix} p \\ 2 \end{pmatrix} \times s$ matrix with s symmetric matrices of the covariances of the p parameter estimates stored in its columns (compact storage of the lower triangle)

resi is a $N \times (1 + 2 * s)$ matrix containing the input data in its first column and s column tripels with predicted values, residuals, and standardized residuals

Restrictions: 1. Observations with missing values in the data are skipped.

2. The data may not have any complex values and no string values except for the ID variable.

3.

Different from Don Hedeker's program, the data must not be sorted w.r.t. the ID variable.

Relationships: glmixd(), reg(), glim()

Examples: 1. Example 1 by Hedeker (2012): Riesby Data $N = 396, n = 6$ with linear relationship between the random location (mean) and the WS variance. Deleting rows with missing values there are 375 observations without missing values.

```
  101 26  1  0  0  0
  101 22  1  1  0  0
  101 18  1  2  0  0
  ...................
```

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


riesby = replace(riesby,-9,.);
riesby = shape(riesby,.,6);
cnam = [" ID HamDep Intercept Week Endog EndWeek "];
riesby = cname(riesby,cnam);
nr = nrow(riesby); nc = ncol(riesby);
print "nr,nc=",nr,nc;


modl = "2 = 3:6";
/* the following index vectors relate to
   the effect number in the model statement */
xind = [ 1:4 ];
uind = [ 1 3 ];
wind = [ 1:3 ];
```

```
optn = [ "idvar"       1 ,
         "adapt"         ,
         "nquad"      11 ,
         "maxit"     200 ,
         "pobs"          ,
         "popt"        2 ,
         "conv"    1.e-5 ];
< gof,parm,cov,yhat > = mixregv(riesby,modl,optn,xind,uind,wind);
```

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

                Number Valid Observations  375
                Response Variable          Y[2]
                N Independent Variables      4
                Mean and WS Variance: Linear
                Number Level-2 Clusters     66
                Number Fixed Effects         4
                Number Random Effects        2
                Number Error Variances       3
                Quad Points per Dimension   11
                Adaptive Quadrature
                Skip Cases with Missing Values
                Subject Variable Column      1
                Significance Level:  0.0500000


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

                     X3 + X4 + X5 + X6


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

 Class  Level    Value

  I[1]     66      101     103     104     105     106
                   107     108     113     114     115
                   117     118     120     121     123
                   302     303     304     305     308
                   309     310     311     312     313
```

315	316	318	319	322
327	328	331	333	334
335	337	338	339	344
345	346	347	348	349
350	351	352	353	354
355	357	360	361	501
502	504	505	507	603
604	606	607	608	609
610				

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[2]	375	17.637333	7.1900625	-0.1138446	-0.2714409
X[3]	396	1.0000000	0	0	0
X[4]	396	2.5000000	1.7099856	0	-1.2694297
X[5]	396	0.5606061	0.4969412	-0.2451546	-1.9497723
X[6]	396	1.4040404	1.7827255	0.8722706	-0.7506594

Data are sorted in groups of subject ID variable.

	Variable	Mean	Min	Max	StdDev
Y	HamDep	17.637333	0	39.000000	7.1900625
		Mean Model Covariates X			
X	Intercept	1.0000000	1.0000000	1.0000000	0
X	Week	2.4800000	0	5.0000000	1.6831979
X	Endog	0.5466667	0	1.0000000	0.4984825
X	EndWeek	1.3520000	0	5.0000000	1.7455336
		BS Variance Model Covariates U			
U	Intercept	1.0000000	1.0000000	1.0000000	0
U	Endog	0.5466667	0	1.0000000	0.4984825
		WS Variance Model Covariates W			
W	Intercept	1.0000000	1.0000000	1.0000000	0
W	Week	2.4800000	0	5.0000000	1.6831979
W	Endog	0.5466667	0	1.0000000	0.4984825

The following are starting values for the first optimization:

****
Starting Values for Optimization
****

```
        BETA: mean model regression coefficients
                                   1
                Beta_1   22.44183873
                Beta_2  -2.351841893
                Beta_3   1.992471848
                Beta_4  -0.044175984


 ALPHA: BS variance log-linear model regression coefficients
                                   1
                Alpha_1   2.726978482
                Alpha_2   0.642391583


  TAU: WS variance log-linear model regression coefficients
                                  1
                 Tau_1   2.946263776
                 Tau_2   0.193253790
                 Tau_3   0.645302867


             Scale covariance and variance
                                   1
                 Spar_1  -0.200074845
                 Spar_2   0.950844227


             ******************************
             Model Without Scale Parameters
             ******************************

  Start of Optimization: Maximum Iterations=200 Xtol=1e-005

     Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

       1B -1142.27111  7.82538061  0.27721525           0
       2B -1140.62800  0.93418339  0.05481512           0
       3B -1140.59952  0.01310270  7.611e-004           0
       4B -1140.59951  2.039e-006  1.304e-007           0
       5B -1140.59951  6.423e-011  5.110e-012           0
   Succesful Optimization after 5 Iterations, crit=-1140.6


         Results of Hedeker Optimization Algorithm
Iterations . . . . .       5   Ridge . . . . . .    0.000000
LogLik. . . . . -1140.599511   Times-2 . . . . . 2281.199022
AIC . . . . . . -1147.599511   Times-2 . . . . . 2295.199022
```

```
SBC . . . . . -1155.263303   Times-2 . . . . . 2310.526605

    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
     Beta_1  22.4458169  0.87362697  25.6926786  1.411e-145
     Beta_2 -2.35330401  0.19797121 -11.8871021  1.381e-032
     Beta_3  1.98710420  1.24592367  1.59488438  0.11073809
     Beta_4 -0.04182137  0.27058310 -0.15456017  0.87716807
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
    Alpha_1  2.47223063  0.33480058  7.38418859  1.534e-013
    Alpha_2  0.42075266  0.43398742  0.96950427  0.33229365
  ------TAU (WS Variance Parameters: Log-linear model)------
      Tau_1  2.94603603  0.08042874  36.6291479  9.830e-294


    Variable    Estimate  AsStdError    Lower CL    Upper CL
  --------------BETA (regression coefficients)--------------
     Beta_1  22.4458169  0.87362697  20.7335395  24.1580942
     Beta_2 -2.35330401  0.19797121 -2.74132046 -1.96528757
     Beta_3  1.98710420  1.24592367 -0.45486132  4.42906972
     Beta_4 -0.04182137  0.27058310 -0.57215450  0.48851176
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
    Alpha_1  2.47223063  0.33480058  1.81603355  3.12842771
    Alpha_2  0.42075266  0.43398742 -0.42984706  1.27135238
  ------TAU (WS Variance Parameters: Log-linear model)------
      Tau_1  2.94603603  0.08042874  2.78839861  3.10367346


                ***************************
                Model With Scale Parameters
                ***************************

  Start of Optimization: Maximum Iterations=200 Xtol=1e-005

     Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

       1B -1180.05546  229.086599  0.60554096  0.10000000
       2B -1142.36144  141.557143  0.11151574  0.10000000
       3B -1135.42475  35.6875309  0.05825344  0.10000000
       4B -1134.77580  8.41361465  0.03097786  0.10000000
       5B -1134.64307  3.68826317  0.09732959           0
       6B -1134.49976  0.19515144  0.00157960           0
       7B -1134.49971  6.243e-005  1.513e-006           0
       8B -1134.49971  9.436e-011  7.099e-012           0
   Succesful Optimization after 8 Iterations, crit=-1134.5
```

```
                    ***************************
                    Model With Scale Parameters
                    ***************************

          Results of Hedeker Optimization Algorithm
Iterations . . . . .       8   Ridge . . . . . .    0.000000
LogLik. . . . . -1134.499708   Times-2 . . . . . 2268.999416
AIC . . . . . . -1143.499708   Times-2 . . . . . 2286.999416
SBC . . . . . . -1153.353154   Times-2 . . . . . 2306.706308


    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
      Beta_1  22.5565200  0.74425066  30.3076924  9.078e-202
      Beta_2 -2.39855570  0.18435148 -13.0107751  1.063e-038
      Beta_3  1.85334851  1.10623319  1.67536874  0.09386191
      Beta_4  0.01527996  0.26949546  0.05669839  0.95478545
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  2.25028583  0.34600423  6.50363674  7.840e-011
     Alpha_2  0.48166202  0.44626590  1.07931621  0.28044679
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  2.34613663  0.18330810  12.7988706  1.664e-037
       Tau_2  0.17670505  0.06077689  2.90743823  0.00364402
       Tau_3  0.27196762  0.16205598  1.67823257  0.09330171


    Variable    Estimate  AsStdError    Lower CL    Upper CL
  --------------BETA (regression coefficients)--------------
      Beta_1  22.5565200  0.74425066  21.0978155  24.0152245
      Beta_2 -2.39855570  0.18435148 -2.75987797 -2.03723343
      Beta_3  1.85334851  1.10623319 -0.31482870  4.02152571
      Beta_4  0.01527996  0.26949546 -0.51292144  0.54348136
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  2.25028583  0.34600423  1.57213000  2.92844166
     Alpha_2  0.48166202  0.44626590 -0.39300307  1.35632712
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  2.34613663  0.18330810  1.98685936  2.70541390
       Tau_2  0.17670505  0.06077689  0.05758454  0.29582557
       Tau_3  0.27196762  0.16205598 -0.04565626  0.58959150


                    ***********************
                    Model With Random Scale
                    ***********************

  Start of Optimization: Maximum Iterations=200 Xtol=1e-005
```

```
     Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

       1B -1129.55156  25.8812733  0.46136827  0.20000000
       2B -1124.37571  9.08412995  0.18153386  0.20000000
       3B -1123.16990  5.85935557  0.14724264  0.20000000
       4B -1122.69867  3.85304409  0.09381031  0.20000000
       5B -1122.50146  2.54578517  0.05792945  0.20000000
       6B -1122.41169  1.69092656  0.03223498  0.20000000
       7B -1122.33765  1.20994600  0.08163707           0
       8B -1122.29660  0.03106596  0.00151538           0
       9B -1122.29654  9.859e-004  2.126e-004           0
      10B -1122.29650  0.00103321  1.392e-004           0
      11B -1122.29650  3.727e-005  3.703e-006           0
      12B -1122.29651  8.543e-005  1.146e-005           0
  Succesful Optimization after 12 Iterations, crit=-1122.3


          Results of Hedeker Optimization Algorithm
Iterations . . . . .      12   Ridge . . . . . .    0.000000
LogLik. . . . . -1122.296505   Times-2 . . . . . 2244.593010
AIC . . . . . . -1133.296505   Times-2 . . . . . 2266.593010
SBC . . . . . . -1145.339606   Times-2 . . . . . 2290.679212


    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
      Beta_1  22.3783209  0.72337793  30.9358634  3.937e-210
      Beta_2 -2.29543135  0.18772989 -12.2273089  2.222e-034
      Beta_3  1.87941977  1.07656449  1.74575679  0.08085322
      Beta_4 -0.02861396  0.26772260 -0.10687914  0.91488487
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  2.19825311  0.35443307  6.20216700  5.569e-010
     Alpha_2  0.50681846  0.45811417  1.10631475  0.26859031
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  2.08768096  0.23637495  8.83207348  1.028e-018
       Tau_2  0.19234039  0.06282843  3.06135904  0.00220335
       Tau_3  0.28814845  0.24544345  1.17399119  0.24039856
  -Random Scale Parameters: Cholesky Factorization Elements-
      Spar_1  0.21326535  0.14559033  1.46483179  0.14296682
      Spar_2  0.65869504  0.13395145  4.91741618  8.769e-007


    Variable    Estimate  AsStdError    Lower CL    Upper CL
  --------------BETA (regression coefficients)--------------
      Beta_1  22.3783209  0.72337793  20.9605262  23.7961156
      Beta_2 -2.29543135  0.18772989 -2.66337518 -1.92748753
      Beta_3  1.87941977  1.07656449 -0.23060786  3.98944739
```

```
    Beta_4 -0.02861396  0.26772260 -0.55334061  0.49611269
-----ALPHA (BS Variance Parameters: Log-linear model)-----
   Alpha_1  2.19825311  0.35443307  1.50357705  2.89292917
   Alpha_2  0.50681846  0.45811417 -0.39106881  1.40470573
------TAU (WS Variance Parameters: Log-linear model)------
     Tau_1  2.08768096  0.23637495  1.62439457  2.55096736
     Tau_2  0.19234039  0.06282843  0.06919893  0.31548186
     Tau_3  0.28814845  0.24544345 -0.19291188  0.76920878
-Random Scale Parameters: Cholesky Factorization Elements-
    Spar_1  0.21326535  0.14559033 -0.07208646  0.49861716
    Spar_2  0.65869504  0.13395145  0.39615502  0.92123507
```

Predicted Values and Residuals

****

Dense Matrix (396 by 10)

	Ydata	Ypred_1	Yresid_1	StdRes_1	Ypred_2
1	26.000000	19.898432	6.1015683	1.3987288	21.133006
2	22.000000	17.545129	4.4548715	1.0212386	18.734450
3	18.000000	15.191825	2.8081747	0.6437484	16.335895
4	7.0000000	12.838522	-5.8385221	-1.3384278	13.937339
5	4.0000000	10.485219	-6.4852189	-1.4866772	11.538784
6	3.0000000	8.1319157	-5.1319157	-1.1764448	9.1402279
7	33.000000	25.683354	7.3166456	1.6772741	26.221202
8	24.000000	23.330051	0.6699488	0.1535796	23.822646
9	15.000000	20.976748	-5.9767480	-1.3701148	21.424090
10	24.000000	18.623445	5.3765552	1.2325261	19.025535

	Yresid_2	StdRes_2	Ypred_3	Yresid_3	StdRes_3
1	4.8669940	1.5059252	21.513429	4.4865713	1.1803817
2	3.2655496	0.9249707	19.216605	2.7833948	0.6651185
3	1.6641052	0.4315008	16.919782	1.0802184	0.2344510
4	-6.9373391	-1.6467319	14.622958	-7.6229581	-1.5027283
5	-7.5387835	-1.6381753	12.326135	-8.3261345	-1.4907908
6	-6.1402279	-1.2214417	10.029311	-7.0293110	-1.1431478
7	6.7787984	2.0974678	25.878942	7.1210577	1.8584564
8	0.1773540	0.0502357	23.582119	0.4178812	0.0990552
9	-6.4240904	-1.6657602	21.285295	-6.2852952	-1.3532133
10	4.9744653	1.1808001	18.988472	5.0115283	0.9800023

```
  print "GOF=",gof;
  print "Parm=",parm;
  print "Cov=",cov;
```

GOF= Page 95

	WithoutSc	WithScale	WithRandS
RetCode	0.00000	0.00000	0.00000
N_obs	375.00	375.00	375.00
N_par	11.000	11.000	11.000
OptTime	0.00000	1.00000	2.0000
MaxAbsG	6e-011	9e-011	9e-005
LogLik	-1140.6	-1134.5	-1122.3
AIC	-1147.6	-1143.5	-1133.3
SBC	-1155.3	-1153.4	-1145.3
unused	.	.	.
unused	.	.	.

The three solutions are packed in the same matrix object:

Parm=

	Parm_1	ASE_1	ZValue_1	PValue_1	CI_LOW_1
Beta_1	22.446	0.87363	25.693	1e-145	20.734
Beta_2	-2.3533	0.19797	-11.887	1e-032	-2.7413
Beta_3	1.9871	1.2459	1.5949	0.11074	-0.45486
Beta_4	-0.0418	0.27058	-0.15456	0.87717	-0.57215
Alpha_1	2.4722	0.33480	7.3842	2e-013	1.8160
Alpha_2	0.42075	0.43399	0.96950	0.33229	-0.42985
Tau_1	2.9460	0.0804	36.629	0.00000	2.7884
Tau_2	.	.	.	.	.
Tau_3	.	.	.	.	.
Spar_1	.	.	.	.	.
Spar_2	.	.	.	.	.

	CI_UPP_1	Parm_2	ASE_2	ZValue_2	PValue_2
Beta_1	24.158	22.557	0.74425	30.308	0.00000
Beta_2	-1.9653	-2.3986	0.18435	-13.011	1e-038
Beta_3	4.4291	1.8533	1.1062	1.6754	0.0939
Beta_4	0.48851	0.0153	0.26950	0.0567	0.95479
Alpha_1	3.1284	2.2503	0.34600	6.5036	8e-011
Alpha_2	1.2714	0.48166	0.44627	1.0793	0.28045

```
    Tau_1 |     3.1037     2.3461    0.18331     12.799     2e-037
    Tau_2 |      .        0.17671     0.0608     2.9074     0.0036
    Tau_3 |      .        0.27197    0.16206     1.6782     0.0933
   Spar_1 |      .          .          .          .          .
   Spar_2 |      .          .          .          .          .

          |   CI_LOW_2   CI_UPP_2     Parm_3      ASE_3   ZValue_3
-------------------------------------------------------------------
   Beta_1 |     21.098     24.015     22.378    0.72338     30.936
   Beta_2 |    -2.7599    -2.0372    -2.2954    0.18773    -12.227
   Beta_3 |   -0.31483     4.0215     1.8794     1.0766     1.7458
   Beta_4 |   -0.51292    0.54348    -0.0286    0.26772   -0.10688
  Alpha_1 |     1.5721     2.9284     2.1983    0.35443     6.2022
  Alpha_2 |   -0.39300     1.3563    0.50682    0.45811     1.1063
    Tau_1 |     1.9869     2.7054     2.0877    0.23637     8.8321
    Tau_2 |     0.0576    0.29583    0.19234     0.0628     3.0614
    Tau_3 |    -0.0457    0.58959    0.28815    0.24544     1.1740
   Spar_1 |      .          .        0.21327    0.14559     1.4648
   Spar_2 |      .          .        0.65870    0.13395     4.9174

          |   PValue_3   CI_LOW_3   CI_UPP_3
-------------------------------------------
   Beta_1 |    0.00000     20.961     23.796
   Beta_2 |     2e-034    -2.6634    -1.9275
   Beta_3 |     0.0809   -0.23061     3.9894
   Beta_4 |    0.91488   -0.55334    0.49611
  Alpha_1 |     6e-010     1.5036     2.8929
  Alpha_2 |    0.26859   -0.39107     1.4047
    Tau_1 |     1e-018     1.6244     2.5510
    Tau_2 |     0.0022     0.0692    0.31548
    Tau_3 |    0.24040   -0.19291    0.76921
   Spar_1 |    0.14297    -0.0721    0.49862
   Spar_2 |     9e-007    0.39616    0.92124
```

2. Example 1 by Hedeker (2012): Alternative Model Specification
   If the data set does not contain an intercept variable (column of one's) as here where we delete that column from the data:

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

```
riesby = replace(riesby,-9,.);
riesby = shape(riesby,.,6);
ind = [ 1 2 4:6 ];
riesb2 = riesby[, ind];
cnam = [" ID HamDep Week Endog EndWeek "];
riesb2 = cname(riesb2,cnam);
nr = nrow(riesb2); nc = ncol(riesb2);
print "nr,nc=",nr,nc;
```

Now, the model only contains the effects without the intercept, but the intercept is automatically included by using the -1 in the `xind`, `uind`, and `wind` arguments.

```
modl = "2 = 3:5";
/* the following index vectors relate to
   the effect number in the model statement */
xind = [ -1 1:3 ];
uind = [ -1   2 ];
wind = [ -1 1 2 ];
```

The result are the same as shown.

3. Example 1 by Hedeker (2012): Compare optimization methods

   - Double Dogleg Optimization
     This optimization method does not use second order derivatives and usually takes more iterations and function calls than the others here. It seems to be inferior for models with few parameters (as here) but could be competitive for models with many parameters.

```
optn = [ "idvar"        1 ,
         "adapt"          ,
         "nquad"       11 ,
         "tech"      "dd" ,
         "maxit"      200 ,
         "popt"         2 ,
         "absgtol"  1.e-2 ,
         "gtol"     1.e-9 ];
< gof,parm,cov,yhat > = mixregv(riesby,modl,optn,xind,uind,wind);
```

```
GOF=
          |   WithoutSc   WithScale   WithRandS
-----------------------------------------------
  RetCode |      3.0000      6.0000      6.0000
    N_obs |      375.00      375.00      375.00
    N_par |      11.000      11.000      11.000
```

```
 OptTime |     0.00000     0.00000      5.0000
 MaxAbsG |        0.01      228.98        0.04
  LogLik |     -1140.6     -1180.1     -1122.3
     AIC |     -1147.6     -1189.1     -1133.3
     SBC |     -1155.3     -1198.9     -1145.3
  unused |          .           .           .
  unused |          .           .           .
```

- Newton-Raphson Ridge optimization:
  This method and the next make good use of second order derivatives (Hessian matrix) and have for that size of optimization problem about the same performance as Hedeker's Newton-Raphson ridge algorithm.

```
optn = [ "idvar"       1 ,
         "adapt"         ,
         "nquad"      11 ,
         "tech"    "nrr" ,
         "maxit"     200 ,
         "popt"        2 ,
         "gtol"    1.e-4 ];
< gof,parm,cov,yhat > = mixregv(riesby,modl,optn,xind,uind,wind);

GOF=
          |   WithoutSc   WithScale   WithRandS
-----------------------------------------------
 RetCode |      6.0000      6.0000      6.0000
   N_obs |      375.00      375.00      375.00
   N_par |      11.000      11.000      11.000
 OptTime |     0.00000     0.00000      3.0000
 MaxAbsG |        0.01        0.02     0.10669
  LogLik |     -1140.6     -1134.5     -1122.3
     AIC |     -1147.6     -1143.5     -1133.3
     SBC |     -1155.3     -1153.4     -1145.3
  unused |          .           .           .
  unused |          .           .           .
```

- Trust-Region optimization:

```
optn = [ "idvar"       1 ,
         "adapt"         ,
         "nquad"      11 ,
         "tech"    "tr" ,
         "maxit"     200 ,
         "popt"        2 ,
         "gtol"    1.e-5 ];
< gof,parm,cov,yhat > = mixregv(riesby,modl,optn,xind,uind,wind);
```

```
GOF=
          |   WithoutSc   WithScale   WithRandS
-----------------------------------------------
  RetCode |      6.0000      6.0000      6.0000
    N_obs |      375.00      375.00      375.00
    N_par |      11.000      11.000      11.000
  OptTime |     0.00000     0.00000      2.0000
  MaxAbsG |      2e-006        0.01        0.02
   LogLik |     -1140.6     -1134.5     -1122.3
      AIC |     -1147.6     -1143.5     -1133.3
      SBC |     -1155.3     -1153.4     -1145.3
   unused |         .           .           .
   unused |         .           .           .
```

4. Example 1 by Hedeker (2012): Riesby Data $N = 396, n = 6$ with quadratic relationship between the random location (mean) and the WS variance.

```
print "Tech=Hedeker NRR: COVREL=QUAD";
optn = [ "idvar"        1 ,
         "covrel"   "qua" ,
         "adapt"          ,
         "nquad"       11 ,
         "tech"     "hed" ,
         "maxit"      200 ,
         "pcov"           ,
         "popt"         2 ,
         "conv"     1.e-5 ];
< gof,parm,cov,yhat > = mixregv(riesby,modl,optn,xind,uind,wind);


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

                  Number Valid Observations  375
                  Response Variable          Y[2]
                  N Independent Variables      4
                  Mean and WS Variance: Quadratic
                  Number Level-2 Clusters     66
                  Number Fixed Effects         4
                  Number Random Effects        2
                  Number Error Variances       3
                  Quad Points per Dimension   11
                  Nonadaptive Quadrature
```

```
Skip Cases with Missing Values
Subject Variable Column      1
Significance Level:  0.0500000


*******************************
Starting Values for Optimization
*******************************

BETA: mean model regression coefficients
                              1
          Beta_1   22.44183873
          Beta_2  -2.351841893
          Beta_3   1.992471848
          Beta_4  -0.044175984


ALPHA: BS variance log-linear model regression coefficients
                              1
          Alpha_1   2.726978482
          Alpha_2   0.642391583


TAU: WS variance log-linear model regression coefficients
                              1
          Tau_1   2.946263776
          Tau_2   0.193253790
          Tau_3   0.645302867

Random linear location effects on WS variance. . . .   -0.200075
Random quadratic location effects on WS variance . .    0.050019
Scale standard deviation . . . . . .    0.950844


*****************************
Model Without Scale Parameters
*****************************

Start of Optimization: Maximum Iterations=200 Xtol=1e-005

    Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

      1B -1142.27111  7.82538061  0.27721525           0
      2B -1140.62800  0.93418339  0.05481512           0
      3B -1140.59952  0.01310270  7.611e-004           0
      4B -1140.59951  2.039e-006  1.304e-007           0
      5B -1140.59951  6.423e-011  5.110e-012           0
```

Succesful Optimization after 5 Iterations, crit=-1140.6

Results of Hedeker Optimization Algorithm

Iterations	5	Ridge	0.000000
CompTime.	0	MaxAbsGrad. . . .	0.000000
LogLik.	-1140.599511	Times-2	2281.199022
AIC	-1147.599511	Times-2	2295.199022
SBC	-1155.263303	Times-2	2310.526605

Variable	Estimate	AsStdError	Zvalue	p-Value
BETA (regression coefficients)				
Beta_1	22.4458169	0.87362697	25.6926786	1.411e-145
Beta_2	-2.35330401	0.19797121	-11.8871021	1.381e-032
Beta_3	1.98710420	1.24592367	1.59488438	0.11073809
Beta_4	-0.04182137	0.27058310	-0.15456017	0.87716807
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	2.47223063	0.33480058	7.38418859	1.534e-013
Alpha_2	0.42075266	0.43398742	0.96950427	0.33229365
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	2.94603603	0.08042874	36.6291479	9.830e-294

Variable	Estimate	AsStdError	Lower CL	Upper CL
BETA (regression coefficients)				
Beta_1	22.4458169	0.87362697	20.7335395	24.1580942
Beta_2	-2.35330401	0.19797121	-2.74132046	-1.96528757
Beta_3	1.98710420	1.24592367	-0.45486132	4.42906972
Beta_4	-0.04182137	0.27058310	-0.57215450	0.48851176
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	2.47223063	0.33480058	1.81603355	3.12842771
Alpha_2	0.42075266	0.43398742	-0.42984706	1.27135238
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	2.94603603	0.08042874	2.78839861	3.10367346

Estimated Covariance Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Beta_4
Beta_1	0.763224075	-0.097461654	-0.763222531	0.097460777
Beta_2	-0.097461654	0.039192601	0.097461115	-0.039192295
Beta_3	-0.763222531	0.097461115	1.552325790	-0.181520988
Beta_4	0.097460777	-0.039192295	-0.181520988	0.073215214
Alpha_1	-0.002005812	0.000699924	0.001987209	-0.000689354
Alpha_2	0.001978301	-0.000690324	-0.002660119	0.001077734
Tau_1	0.000144712	-5.0497e-005	-7.5710e-005	1.1290e-005

Estimated Covariance Matrix of Estimates

	Alpha_1	Alpha_2	Tau_1
Beta_1	-0.002005812	0.001978301	0.000144712
Beta_2	0.000699924	-0.000690324	-5.0497e-005
Beta_3	0.001987209	-0.002660119	-7.5710e-005
Beta_4	-0.000689354	0.001077734	1.1290e-005
Alpha_1	0.112091429	-0.111759860	-0.001744046
Alpha_2	-0.111759860	0.188345083	0.000514235
Tau_1	-0.001744046	0.000514235	0.006468781

Estimated Correlation Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Beta_4
Beta_1	1.000000000	-0.563515287	-0.701186774	0.412290350
Beta_2	-0.563515287	1.000000000	0.395128082	-0.731640916
Beta_3	-0.701186774	0.395128082	1.000000000	-0.538436812
Beta_4	0.412290350	-0.731640916	-0.538436812	1.000000000
Alpha_1	-0.006857694	0.010559972	0.004763936	-0.007609483
Alpha_2	0.005217820	-0.008034776	-0.004919630	0.009177699
Tau_1	0.002059530	-0.003171413	-0.000755532	0.000518782

Estimated Correlation Matrix of Estimates

	Alpha_1	Alpha_2	Tau_1
Beta_1	-0.006857694	0.005217820	0.002059530
Beta_2	0.010559972	-0.008034776	-0.003171413
Beta_3	0.004763936	-0.004919630	-0.000755532
Beta_4	-0.007609483	0.009177699	0.000518782
Alpha_1	1.000000000	-0.769170294	-0.064767986
Alpha_2	-0.769170294	1.000000000	0.014732389
Tau_1	-0.064767986	0.014732389	1.000000000

****
Model With Scale Parameters
****

Start of Optimization: Maximum Iterations=200 Xtol=1e-005

Iter	Loglike	MaxAbsGrad	MaxChange	Ridge
1B	-1180.05546	229.086599	0.60554096	0.10000000
2B	-1142.36144	141.557143	0.11151574	0.10000000
3B	-1135.42475	35.6875309	0.05825344	0.10000000

```
      4B -1134.77580  8.41361465  0.03097786  0.10000000
      5B -1134.64307  3.68826317  0.09732959           0
      6B -1134.49976  0.19515144  0.00157960           0
      7B -1134.49971  6.243e-005  1.513e-006           0
      8B -1134.49971  9.434e-011  7.110e-012           0
  Succesful Optimization after 8 Iterations, crit=-1134.5


         Results of Hedeker Optimization Algorithm
Iterations . . . . .       8   Ridge . . . . . .    0.000000
CompTime. . . .            1   MaxAbsGrad. . . .    0.000000
LogLik. . . . . -1134.499708   Times-2 . . . . . 2268.999416
AIC . . . . . . -1143.499708   Times-2 . . . . . 2286.999416
SBC . . . . . . -1153.353154   Times-2 . . . . . 2306.706308


   Variable    Estimate  AsStdError      Zvalue     p-Value
 --------------BETA (regression coefficients)--------------
     Beta_1  22.5565200  0.74425066  30.3076924  9.078e-202
     Beta_2 -2.39855570  0.18435148 -13.0107751  1.063e-038
     Beta_3  1.85334851  1.10623319  1.67536874  0.09386191
     Beta_4  0.01527996  0.26949546  0.05669839  0.95478545
 -----ALPHA (BS Variance Parameters: Log-linear model)-----
    Alpha_1  2.25028583  0.34600423  6.50363674  7.840e-011
    Alpha_2  0.48166202  0.44626590  1.07931621  0.28044679
 ------TAU (WS Variance Parameters: Log-linear model)------
      Tau_1  2.34613663  0.18330810  12.7988706  1.664e-037
      Tau_2  0.17670505  0.06077689  2.90743823  0.00364402
      Tau_3  0.27196762  0.16205598  1.67823257  0.09330171


   Variable    Estimate  AsStdError    Lower CL    Upper CL
 --------------BETA (regression coefficients)--------------
     Beta_1  22.5565200  0.74425066  21.0978155  24.0152245
     Beta_2 -2.39855570  0.18435148 -2.75987797 -2.03723343
     Beta_3  1.85334851  1.10623319 -0.31482870  4.02152571
     Beta_4  0.01527996  0.26949546 -0.51292144  0.54348136
 -----ALPHA (BS Variance Parameters: Log-linear model)-----
    Alpha_1  2.25028583  0.34600423  1.57213000  2.92844166
    Alpha_2  0.48166202  0.44626590 -0.39300307  1.35632712
 ------TAU (WS Variance Parameters: Log-linear model)------
      Tau_1  2.34613663  0.18330810  1.98685936  2.70541390
      Tau_2  0.17670505  0.06077689  0.05758454  0.29582557
      Tau_3  0.27196762  0.16205598 -0.04565626  0.58959150


         Estimated Covariance Matrix of Estimates
```

	Beta_1	Beta_2	Beta_3	Beta_4
Beta_1	0.553909039	-0.067764618	-0.554306311	0.067986867
Beta_2	-0.067764618	0.033985470	0.067942890	-0.034085201
Beta_3	-0.554306311	0.067942890	1.223751868	-0.145028102
Beta_4	0.067986867	-0.034085201	-0.145028102	0.072627803
Alpha_1	-0.004705681	0.001994306	0.005636906	-0.002515267
Alpha_2	0.003561127	-0.001480700	-0.005120569	0.002360701
Tau_1	-0.004258608	0.001929048	0.005939389	-0.002869340
Tau_2	0.002002110	-0.000898425	-0.002735066	0.001308468
Tau_3	-0.000787374	0.000335281	0.001132076	-0.000529412

Estimated Covariance Matrix of Estimates

	Alpha_1	Alpha_2	Tau_1	Tau_2
Beta_1	-0.004705681	0.003561127	-0.004258608	0.002002110
Beta_2	0.001994306	-0.001480700	0.001929048	-0.000898425
Beta_3	0.005636906	-0.005120569	0.005939389	-0.002735066
Beta_4	-0.002515267	0.002360701	-0.002869340	0.001308468
Alpha_1	0.119718928	-0.117036044	0.006839017	-0.004693033
Alpha_2	-0.117036044	0.199153256	-0.001996640	0.002581367
Tau_1	0.006839017	-0.001996640	0.033601859	-0.008470524
Tau_2	-0.004693033	0.002581367	-0.008470524	0.003693830
Tau_3	0.004988985	-0.007316842	-0.012253324	-0.000839155

Estimated Covariance Matrix of Estimates

	Tau_3
Beta_1	-0.000787374
Beta_2	0.000335281
Beta_3	0.001132076
Beta_4	-0.000529412
Alpha_1	0.004988985
Alpha_2	-0.007316842
Tau_1	-0.012253324
Tau_2	-0.000839155
Tau_3	0.026262140

Estimated Correlation Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Beta_4
Beta_1	1.000000000	-0.493897845	-0.673261707	0.338964603
Beta_2	-0.493897845	1.000000000	0.333158344	-0.686068963
Beta_3	-0.673261707	0.333158344	1.000000000	-0.486467715

```
Beta_4    0.338964603 -0.686068963 -0.486467715  1.000000000
Alpha_1  -0.018273506  0.031265378  0.014726946 -0.026974369
Alpha_2   0.010721970 -0.017998100 -0.010372367  0.019628895
Tau_1    -0.031215246  0.057084054  0.029289600 -0.058082977
Tau_2     0.044261914 -0.080185645 -0.040680157  0.079886435
Tau_3    -0.006528251  0.011222687  0.006314862 -0.012122074
```

Estimated Correlation Matrix of Estimates

```
             Alpha_1      Alpha_2        Tau_1        Tau_2
Beta_1   -0.018273506  0.010721970 -0.031215246  0.044261914
Beta_2    0.031265378 -0.017998100  0.057084054 -0.080185645
Beta_3    0.014726946 -0.010372367  0.029289600 -0.040680157
Beta_4   -0.026974369  0.019628895 -0.058082977  0.079886435
Alpha_1   1.000000000 -0.757956912  0.107827770 -0.223168917
Alpha_2  -0.757956912  1.000000000 -0.024407561  0.095173823
Tau_1     0.107827770 -0.024407561  1.000000000 -0.760309067
Tau_2    -0.223168917  0.095173823 -0.760309067  1.000000000
Tau_3     0.088974530 -0.101173074 -0.412484087 -0.085199842
```

Estimated Correlation Matrix of Estimates

```
                    Tau_3
Beta_1   -0.006528251
Beta_2    0.011222687
Beta_3    0.006314862
Beta_4   -0.012122074
Alpha_1   0.088974530
Alpha_2  -0.101173074
Tau_1    -0.412484087
Tau_2    -0.085199842
Tau_3     1.000000000
```

```
**********************
Model With Random Scale
**********************

Start of Optimization: Maximum Iterations=200 Xtol=1e-005

    Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

      1A -1130.25167  32.0888228  0.58535562  0.20000000
      2B -1123.57245  8.27218117  0.22225972  0.20000000
      3B -1122.76819  6.19282522  0.14222092  0.20000000
```

```
      4B -1122.33680  5.04846402  0.09066719  0.20000000
      5B -1122.06710  4.37135546  0.06719363  0.20000000
      6B -1121.87147  3.92352489  0.05203982  0.20000000
      7B -1121.58030  3.19701644  0.54997039           0
      9B -1124.08506  20.0210881  0.16481344  0.10000000
     10B -1121.75604  6.29166993  0.08110725  0.10000000
     11B -1121.28778  2.28793968  0.04744670  0.10000000
     12B -1121.17237  1.03971253  0.03164354  0.10000000
     13B -1121.14046  0.54328152  0.02346401  0.10000000
     14B -1121.12970  0.32055282  0.01532685  0.10000000
     15B -1121.12588  0.19705632  0.00968081  0.10000000
     16B -1121.12448  0.12241339  0.00614401  0.10000000
     17B -1121.12400  0.07651122  0.00396001  0.10000000
     18B -1121.12385  0.04819068  0.00255917  0.10000000
     19B -1121.12383  0.03057382  0.00593673           0
     20B -1121.12380  0.00217206  2.121e-004           0
     21B -1121.12391  0.00393688  3.003e-004           0
     22B -1121.12392  3.540e-004  7.918e-005           0
     23B -1121.12392  2.841e-004  7.695e-005           0
     24B -1121.12393  1.002e-004  6.014e-006           0
     25B -1121.12393  9.069e-005  6.307e-006           0
  Succesful Optimization after 25 Iterations, crit=-1121.12


          Results of Hedeker Optimization Algorithm
Iterations . . . . .      25   Ridge . . . . . .    0.000000
CompTime. . . .            6   MaxAbsGrad. . . .    0.000091
LogLik. . . . . -1121.123929   Times-2 . . . . . 2242.247857
AIC . . . . . . -1133.123929   Times-2 . . . . . 2266.247857
SBC . . . . . . -1146.261857   Times-2 . . . . . 2292.523714


    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
      Beta_1  22.2985211  0.71853725  31.0332153  1.922e-211
      Beta_2 -2.26550301  0.19346236 -11.7103037  1.129e-031
      Beta_3  2.04268471  1.00519916  2.03211939  0.04214157
      Beta_4 -0.08632340  0.26318278 -0.32799792  0.74291324
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  2.19328838  0.33148567  6.61654052  3.677e-011
     Alpha_2  0.20353343  0.51201359  0.39751568  0.69098723
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  2.29066904  0.26264840  8.72142771  2.747e-018
       Tau_2  0.22320219  0.05951177  3.75055525  1.764e-004
       Tau_3  0.34183002  0.27292192  1.25248282  0.21039398
  -Random Scale Parameters: Cholesky Factorization Elements-
      Spar_1  0.32899726  0.16906217  1.94601344  0.05165312
```

```
    Spar_2 -0.30879887  0.17195914 -1.79576882  0.07253129
    Spar_3  0.64055835  0.13405702  4.77825307  1.768e-006


  Variable    Estimate  AsStdError    Lower CL    Upper CL
--------------BETA (regression coefficients)--------------
    Beta_1  22.2985211  0.71853725  20.8902140  23.7068283
    Beta_2 -2.26550301  0.19346236 -2.64468228 -1.88632375
    Beta_3  2.04268471  1.00519916  0.07253055  4.01283887
    Beta_4 -0.08632340  0.26318278 -0.60215218  0.42950537
-----ALPHA (BS Variance Parameters: Log-linear model)-----
   Alpha_1  2.19328838  0.33148567  1.54358840  2.84298835
   Alpha_2  0.20353343  0.51201359 -0.79999477  1.20706163
------TAU (WS Variance Parameters: Log-linear model)------
     Tau_1  2.29066904  0.26264840  1.77588764  2.80545045
     Tau_2  0.22320219  0.05951177  0.10656126  0.33984313
     Tau_3  0.34183002  0.27292192 -0.19308712  0.87674715
-Random Scale Parameters: Cholesky Factorization Elements-
    Spar_1  0.32899726  0.16906217 -0.00235851  0.66035303
    Spar_2 -0.30879887  0.17195914 -0.64583259  0.02823486
    Spar_3  0.64055835  0.13405702  0.37781143  0.90330528


          Estimated Covariance Matrix of Estimates

              Beta_1       Beta_2       Beta_3       Beta_4
Beta_1    0.516295779 -0.074693339 -0.498386190  0.075884414
Beta_2   -0.074693339  0.037427686  0.075435785 -0.035956564
Beta_3   -0.498386190  0.075435785  1.010425360 -0.147924212
Beta_4    0.075884414 -0.035956564 -0.147924212  0.069265177
Alpha_1   0.004265662 -9.4528e-005 -0.007565547 -0.001127909
Alpha_2  -0.011980451 -0.000966537  0.022401201  0.005888020
Tau_1     0.036949917 -0.002837619 -0.033678650  0.001136541
Tau_2    -0.000465344  0.000887513  5.7371e-005 -0.000627807
Tau_3    -0.030891162  0.003149964  0.056760207 -0.005617555
Spar_1   -0.012887002 -0.003815250 -0.009564947 -0.002033914
Spar_2    0.000254062 -0.002476524  0.003252406  0.004087615
Spar_3   -0.011124503  0.004368637  0.003534113 -0.001769863


          Estimated Covariance Matrix of Estimates

             Alpha_1      Alpha_2        Tau_1        Tau_2
Beta_1    0.004265662 -0.011980451  0.036949917 -0.000465344
Beta_2   -9.4528e-005 -0.000966537 -0.002837619  0.000887513
Beta_3   -0.007565547  0.022401201 -0.033678650  5.7371e-005
Beta_4   -0.001127909  0.005888020  0.001136541 -0.000627807
```

```
Alpha_1   0.109882750 -0.121397308 -0.001319966 -0.002114255
Alpha_2  -0.121397308  0.262157920 -0.011448552 -0.000378453
Tau_1    -0.001319966 -0.011448552  0.068984182 -0.007080977
Tau_2    -0.002114255 -0.000378453 -0.007080977  0.003541651
Tau_3     0.026649372 -0.059832141 -0.029549185  9.7198e-005
Spar_1    0.006027486 -0.007399542  0.002284728  0.000736226
Spar_2   -0.018419403  0.040091329 -0.018307146 -0.001567793
Spar_3   -0.001450604  0.012836443 -0.005979072 -0.001017716
```

Estimated Covariance Matrix of Estimates

```
              Tau_3       Spar_1       Spar_2       Spar_3
Beta_1   -0.030891162 -0.012887002  0.000254062 -0.011124503
Beta_2    0.003149964 -0.003815250 -0.002476524  0.004368637
Beta_3    0.056760207 -0.009564947  0.003252406  0.003534113
Beta_4   -0.005617555 -0.002033914  0.004087615 -0.001769863
Alpha_1   0.026649372  0.006027486 -0.018419403 -0.001450604
Alpha_2  -0.059832141 -0.007399542  0.040091329  0.012836443
Tau_1    -0.029549185  0.002284728 -0.018307146 -0.005979072
Tau_2     9.7198e-005  0.000736226 -0.001567793 -0.001017716
Tau_3     0.074486375  0.001612535 -0.010847321 -0.001581516
Spar_1    0.001612535  0.028582018 -0.009282572 -0.003682853
Spar_2   -0.010847321 -0.009282572  0.029569946  0.005805974
Spar_3   -0.001581516 -0.003682853  0.005805974  0.017971284
```

Estimated Correlation Matrix of Estimates

```
             Beta_1       Beta_2       Beta_3       Beta_4
Beta_1    1.000000000 -0.537323828 -0.690024631  0.401278439
Beta_2   -0.537323828  1.000000000  0.387908065 -0.706194316
Beta_3   -0.690024631  0.387908065  1.000000000 -0.559151727
Beta_4    0.401278439 -0.706194316 -0.559151727  1.000000000
Alpha_1   0.017909043 -0.001473999 -0.022705102 -0.012928605
Alpha_2  -0.032564349 -0.009757540  0.043524891  0.043694856
Tau_1     0.195789512 -0.055844816 -0.127563902  0.016441936
Tau_2    -0.010882322  0.077085949  0.000959048 -0.040083483
Tau_3    -0.157523931  0.059658277  0.206896639 -0.078208046
Spar_1   -0.106085533 -0.116648755 -0.056283880 -0.045711846
Spar_2    0.002056195 -0.074442475  0.018816001  0.090320698
Spar_3   -0.115489306  0.168445706  0.026226408 -0.050164066
```

Estimated Correlation Matrix of Estimates

```
          Alpha_1      Alpha_2        Tau_1        Tau_2
```

```
Beta_1    0.017909043 -0.032564349  0.195789512 -0.010882322
Beta_2   -0.001473999 -0.009757540 -0.055844816  0.077085949
Beta_3   -0.022705102  0.043524891 -0.127563902  0.000959048
Beta_4   -0.012928605  0.043694856  0.016441936 -0.040083483
Alpha_1   1.000000000 -0.715258135 -0.015160844 -0.107174057
Alpha_2  -0.715258135  1.000000000 -0.085132287 -0.012420183
Tau_1    -0.015160844 -0.085132287  1.000000000 -0.453018048
Tau_2    -0.107174057 -0.012420183 -0.453018048  1.000000000
Tau_3     0.294566806 -0.428168423 -0.412223105  0.005984315
Spar_1    0.107553617 -0.085482437  0.051453305  0.073174864
Spar_2   -0.323136123  0.455348275 -0.405341078 -0.153200623
Spar_3   -0.032643342  0.187013793 -0.169812439 -0.127565750


         Estimated Correlation Matrix of Estimates

                Tau_3       Spar_1       Spar_2       Spar_3
Beta_1   -0.157523931 -0.106085533  0.002056195 -0.115489306
Beta_2    0.059658277 -0.116648755 -0.074442475  0.168445706
Beta_3    0.206896639 -0.056283880  0.018816001  0.026226408
Beta_4   -0.078208046 -0.045711846  0.090320698 -0.050164066
Alpha_1   0.294566806  0.107553617 -0.323136123 -0.032643342
Alpha_2  -0.428168423 -0.085482437  0.455348275  0.187013793
Tau_1    -0.412223105  0.051453305 -0.405341078 -0.169812439
Tau_2     0.005984315  0.073174864 -0.153200623 -0.127565750
Tau_3     1.000000000  0.034948161 -0.231131328 -0.043226069
Spar_1    0.034948161  1.000000000 -0.319298320 -0.162498122
Spar_2   -0.231131328 -0.319298320  1.000000000  0.251860632
Spar_3   -0.043226069 -0.162498122  0.251860632  1.000000000
```

5. Example 2 by Hedeker (2012): Positive Mood Data $N = 17514, n = 5$: This data set has no missing values.

```
1  9.6667 1  0 1
1 10.0000 1  1 1
1  9.0000 1  0 1
.................
```

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


posmood = shape(posmood,.,5);
```

```
cnam = [" ID PosMood Intercept Alone GenderF "];
posmood = cname(posmood,cnam);
nr = nrow(posmood); nc = ncol(posmood);
print "nr,nc=",nr,nc;


modl = "2 = 3:5";
/* the following index vectors relate to
   the effect number in the model statement */
xind = [ 1:3 ];
uind = [ 1 3 ];
wind = [ 1:3 ];


optn = [ "idvar"       1 ,
         "adapt"         ,
         "nquad"      11 ,
         "maxit"     200 ,
         "popt"        2 ,
         "conv"    1.e-5 ];
< gof,parm,cov,yhat > = mixregv(posmood,modl,optn,xind,uind,wind);

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

                Number Valid Observations17514
                Response Variable          Y[2]
                N Independent Variables      3
                Mean and WS Variance: Linear
                Number Level-2 Clusters    515
                Number Fixed Effects         3
                Number Random Effects        2
                Number Error Variances       3
                Quad Points per Dimension   11
                Adaptive Quadrature
                Skip Cases with Missing Values
                Subject Variable Column      1
                Significance Level:  0.0500000


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

                        X3 + X4 + X5
```

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

Class  Level    Value

I[1]    515        1       2       3       4       5
                   6       7       8       9      10
 ..................................................
                  511     512     513     514     515


                    *****************
                    Simple Statistics
                    *****************

Column  Nobs        Mean     Std Dev    Skewness    Kurtosis
Y[2]   17514   6.7326523   2.1165274  -0.5428862  -0.2184033
X[3]   17514   1.0000000           0           0           0
X[4]   17514   0.4627155   0.4986222   0.1495670  -1.9778556
X[5]   17514   0.5842183   0.4928703  -0.3417855  -1.8833978

     Data are sorted in groups of subject ID variable.


    Variable       Mean        Min        Max     StdDev
  ----------------------------------------------------------
  Y    PosMood  6.7326523  1.0000000  10.000000  2.1165274
  ----------------Mean Model Covariates X-----------------
  X  Intercept  1.0000000  1.0000000  1.0000000          0
  X      Alone  0.4627155          0  1.0000000  0.4986222
  X    GenderF  0.5842183          0  1.0000000  0.4928703
  -------------BS Variance Model Covariates U-------------
  U  Intercept  1.0000000  1.0000000  1.0000000          0
  U    GenderF  0.5842183          0  1.0000000  0.4928703
  -------------WS Variance Model Covariates W-------------
  W  Intercept  1.0000000  1.0000000  1.0000000          0
  W      Alone  0.4627155          0  1.0000000  0.4986222
  W    GenderF  0.5842183          0  1.0000000  0.4928703
```

The following are starting values for the first optimization:

```
                    *******************************
                    Starting Values for Optimization
                    *******************************
```

```
          BETA: mean model regression coefficients
                                      1
                    Beta_1   7.052064259
                    Beta_2  -0.449182713
                    Beta_3  -0.193014768


 ALPHA: BS variance log-linear model regression coefficients
                                      1
                   Alpha_1   0.330756712
                   Alpha_2   0.069095230


  TAU: WS variance log-linear model regression coefficients
                                      1
                    Tau_1   1.101568073
                    Tau_2   0.037123289
                    Tau_3   0.069989704


               Scale covariance and variance
                                      1
                    Spar_1  -0.222043420
                    Spar_2   0.468909716


               ******************************
               Model Without Scale Parameters
               ******************************

  Start of Optimization: Maximum Iterations=200 Xtol=1e-005

      Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

        1B -35210.1857  11.3584891  0.12908420           0
        2B -35209.5910  0.46821097  0.00309441           0
        3B -35209.5902  5.657e-004  4.440e-006           0
        4B -35209.5902  8.186e-009  5.609e-011           0
  Succesful Optimization after 4 Iterations, crit=-35209.6


          Results of Hedeker Optimization Algorithm
Iterations . . . . .       4   Ridge . . . . . .    0.000000
LogLik. . . . . -35209.590201   Times-2 . . . . .70419.180402
AIC . . . . . . -35215.590201   Times-2 . . . . .70431.180402
SBC . . . . . . -35228.322702   Times-2 . . . . .70456.645403
```

```
  Variable    Estimate  AsStdError      Zvalue     p-Value
--------------BETA (regression coefficients)--------------
    Beta_1  7.04729903  0.08335523  84.5453700           0
    Beta_2 -0.44907586  0.02827235 -15.8839231  8.189e-057
    Beta_3 -0.18516736  0.10882435 -1.70152505  0.08884444
-----ALPHA (BS Variance Parameters: Log-linear model)-----
   Alpha_1  0.36208742  0.10039358  3.60667892  3.101e-004
   Alpha_2 -0.05689012  0.13462083 -0.42259521  0.67259062
------TAU (WS Variance Parameters: Log-linear model)------
     Tau_1  1.10156829  0.01084664  101.558448           0


  Variable    Estimate  AsStdError    Lower CL    Upper CL
--------------BETA (regression coefficients)--------------
    Beta_1  7.04729903  0.08335523  6.88392577  7.21067228
    Beta_2 -0.44907586  0.02827235 -0.50448865 -0.39366307
    Beta_3 -0.18516736  0.10882435 -0.39845916  0.02812445
-----ALPHA (BS Variance Parameters: Log-linear model)-----
   Alpha_1  0.36208742  0.10039358  0.16531961  0.55885523
   Alpha_2 -0.05689012  0.13462083 -0.32074210  0.20696186
------TAU (WS Variance Parameters: Log-linear model)------
     Tau_1  1.10156829  0.01084664  1.08030926  1.12282732


             ***************************
             Model With Scale Parameters
             ***************************

 Start of Optimization: Maximum Iterations=200 Xtol=1e-005

     Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

       1B -35200.3459  503.266565  0.03459346  0.10000000
       2B -35185.2511  232.916573  0.02125982  0.10000000
       3B -35179.6934  153.370620  0.01622187  0.10000000
       4B -35176.1998  118.366418  0.01334928  0.10000000
       5B -35173.8515  96.4029796  0.06227509           0
       6B -35168.9312  5.02009738  0.00131601           0
       7B -35168.9287  0.00255205  7.714e-007           0
       8B -35168.9287  1.464e-009  4.937e-012           0
 Succesful Optimization after 8 Iterations, crit=-35168.9


         Results of Hedeker Optimization Algorithm
Iterations . . . . .       8   Ridge . . . . . .    0.000000
LogLik. . . . . -35168.928668   Times-2 . . . . .70337.857336
```

```
AIC . . . . . -35176.928668   Times-2 . . . . .70353.857336
SBC . . . . . -35193.905336   Times-2 . . . . .70387.810672
```

Variable	Estimate	AsStdError	Zvalue	p-Value
BETA (regression coefficients)				
Beta_1	7.04310322	0.08324068	84.6113120	0
Beta_2	-0.44251872	0.02821811	-15.6820796	2.006e-055
Beta_3	-0.18312803	0.10876202	-1.68374980	0.09223003
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	0.36900397	0.09963038	3.70372941	2.125e-004
Alpha_2	-0.07019510	0.13438103	-0.52235866	0.60142063
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	0.95555142	0.02010575	47.5262771	0
Tau_2	0.06034293	0.02226427	2.71030375	0.00672216
Tau_3	0.19370818	0.02205255	8.78393519	1.579e-018

Variable	Estimate	AsStdError	Lower CL	Upper CL
BETA (regression coefficients)				
Beta_1	7.04310322	0.08324068	6.87995448	7.20625195
Beta_2	-0.44251872	0.02821811	-0.49782521	-0.38721223
Beta_3	-0.18312803	0.10876202	-0.39629766	0.03004161
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	0.36900397	0.09963038	0.17373202	0.56427593
Alpha_2	-0.07019510	0.13438103	-0.33357709	0.19318689
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	0.95555142	0.02010575	0.91614488	0.99495797
Tau_2	0.06034293	0.02226427	0.01670577	0.10398009
Tau_3	0.19370818	0.02205255	0.15048598	0.23693039

****
Model WITH Random Scale
****

Start of Optimization: Maximum Iterations=200 Xtol=1e-005

Iter	Loglike	MaxAbsGrad	MaxChange	Ridge
1A	-34198.7093	424.312180	0.08622514	0.20000000
2B	-34182.4210	358.378888	0.06359605	0.20000000
3B	-34165.6783	252.091636	0.05088140	0.20000000
4B	-34156.6046	176.109249	0.04078646	0.20000000
5B	-34151.7162	121.968566	0.03266688	0.20000000
6B	-34149.1112	83.7436738	0.02614068	0.20000000
7B	-34146.8306	32.4050655	0.04259342	0

```
       8B -34146.0881  2.30631130  0.00159641           0
       9B -34146.0861  0.01031675  1.138e-005           0
      10B -34146.0861  1.462e-004  3.243e-007           0
      11B -34146.0861  1.865e-006  1.288e-008           0
  Succesful Optimization after 11 Iterations, crit=-34146.1


          Results of Hedeker Optimization Algorithm
Iterations . . . . .      11   Ridge . . . . . .    0.000000
LogLik. . . . . -34146.086110   Times-2 . . . . .68292.172220
AIC . . . . . . -34156.086110   Times-2 . . . . .68312.172220
SBC . . . . . . -34177.306944   Times-2 . . . . .68354.613889


    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
      Beta_1  7.00260667  0.08216016  85.2311716           0
      Beta_2 -0.35319900  0.02420073 -14.5945604  3.042e-048
      Beta_3 -0.18508648  0.10928756 -1.69357316  0.09034642
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  0.34814162  0.09623484  3.61762580  2.973e-004
     Alpha_2  0.00251555  0.13059489  0.01926225  0.98463190
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  0.75750215  0.04736547  15.9927096  1.436e-057
       Tau_2  0.08634372  0.02467214  3.49964380  4.659e-004
       Tau_3  0.22262369  0.06117985  3.63883998  2.739e-004
  -Random Scale Parameters: Cholesky Factorization Elements-
      Spar_1 -0.21431775  0.03096406 -6.92149916  4.469e-012
      Spar_2  0.59692000  0.02259534  26.4178417  8.548e-154


    Variable    Estimate  AsStdError    Lower CL    Upper CL
  --------------BETA (regression coefficients)--------------
      Beta_1  7.00260667  0.08216016  6.84157571  7.16363762
      Beta_2 -0.35319900  0.02420073 -0.40063156 -0.30576644
      Beta_3 -0.18508648  0.10928756 -0.39928616  0.02911320
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  0.34814162  0.09623484  0.15952481  0.53675843
     Alpha_2  0.00251555  0.13059489 -0.25344574  0.25847684
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  0.75750215  0.04736547  0.66466754  0.85033676
       Tau_2  0.08634372  0.02467214  0.03798720  0.13470023
       Tau_3  0.22262369  0.06117985  0.10271338  0.34253400
  -Random Scale Parameters: Cholesky Factorization Elements-
      Spar_1 -0.21431775  0.03096406 -0.27500620 -0.15362930
      Spar_2  0.59692000  0.02259534  0.55263396  0.64120605
```

Predicted Values and Residuals
****

Dense Matrix (17514 by 10)

	Ydata	Ypred_1	Yresid_1	StdRes_1	Ypred_2
1	9.6667000	7.6567655	2.0099345	1.1587224	7.6501929
2	10.000000	7.2076896	2.7923104	1.6097602	7.2076742
3	9.0000000	7.6567655	1.3432345	0.7743714	7.6501929
4	7.3333000	7.6567655	-0.3234655	-0.1864771	7.6501929
5	8.6667000	7.6567655	1.0099345	0.5822248	7.6501929
6	10.000000	7.6567655	2.3432345	1.3508690	7.6501929
7	7.6667000	7.2076896	0.4590104	0.2646184	7.2076742
8	5.6667000	7.6567655	-1.9900655	-1.1472679	7.6501929
9	8.0000000	7.6567655	0.3432345	0.1978739	7.6501929
10	6.6667000	7.2076896	-0.5409896	-0.3118792	7.2076742

	Yresid_2	StdRes_2	Ypred_3	Yresid_3	StdRes_3
1	2.0165071	1.1351185	7.6211187	2.0455813	1.3241960
2	2.7923258	1.5251208	7.2679197	2.7320803	1.6938685
3	1.3498071	0.7598243	7.6211187	1.3788813	0.8926114
4	-0.3168929	-0.1783832	7.6211187	-0.2878187	-0.1863179
5	1.0165071	0.5722053	7.6211187	1.0455813	0.6768514
6	2.3498071	1.3227375	7.6211187	2.3788813	1.5399560
7	0.4590258	0.2507121	7.2679197	0.3987803	0.2472407
8	-1.9834929	-1.1165344	7.6211187	-1.9544187	-1.2651824
9	0.3498071	0.1969110	7.6211187	0.3788813	0.2452668
10	-0.5409742	-0.2954709	7.2679197	-0.6012197	-0.3727515

GOF=

	WithoutSc	WithScale	WithRandS
RetCode	0.000000	0.000000	0.000000
N_obs	17514.0	17514.0	17514.0
N_par	10.00000	10.00000	10.00000
OptTime	1.000000	3.00000	82.0000
MaxAbsG	8.2e-009	1.5e-009	1.9e-006
LogLik	-35209.6	-35168.9	-34146.1
AIC	-35215.6	-35176.9	-34156.1
SBC	-35228.3	-35193.9	-34177.3

```
unused |          .            .            .
unused |          .            .            .
```

6. Example 2 by Hedeker (2012): Compare optimization methods

   - Double Dogleg Optimization
     This optimization method does not use second order derivatives and usually takes more iterations and function calls than the others here. It seems to be inferior for models with few parameters (as here) but could be competitive for models with many parameters.

```
  optn = [ "idvar"        1 ,
           "adapt"          ,
           "nquad"       11 ,
           "tech"      "dd" ,
           "maxit"      200 ,
           "popt"         2 ,
           "absgtol"  1.e-2 ,
           "gtol"    1.e-10 ];
  < gof,parm,cov,yhat > = mixregv(posmood,modl,optn,xind,uind,wind);

  GOF=
           |   WithoutSc   WithScale   WithRandS
-------------------------------------------------
  RetCode |     6.00000     6.00000     6.00000
    N_obs |     17514.0     17514.0     17514.0
    N_par |    10.00000    10.00000    10.00000
  OptTime |    10.00000     6.00000     77.0000
  MaxAbsG |        0.01        0.01        0.01
   LogLik |    -35209.6    -35168.9    -34146.1
      AIC |    -35215.6    -35176.9    -34156.1
      SBC |    -35228.3    -35193.9    -34177.3
   unused |          .           .           .
   unused |          .           .           .
```

   - Newton-Raphson Ridge optimization:
     This method and the next make good use of second order derivatives (Hessian matrix) and are for that size of optimization problem slightly faster than Hedeker's Newton-Raphson ridge algorithm.

```
  optn = [ "idvar"       1 ,
           "adapt"         ,
           "nquad"      11 ,
           "tech"    "nrr" ,
           "maxit"     200 ,
```

```
          "popt"        2 ,
          "gtol"    1.e-6 ];
  < gof,parm,cov,yhat > = mixregv(posmood,modl,optn,xind,uind,wind);

GOF=
          |   WithoutSc   WithScale   WithRandS
-----------------------------------------------
  RetCode |     6.00000     6.00000     6.00000
    N_obs |     17514.0     17514.0     17514.0
    N_par |    10.00000    10.00000    10.00000
  OptTime |     3.00000     4.00000     62.0000
  MaxAbsG |    5.7e-004    3.7e-006    9.7e-004
   LogLik |    -35209.6    -35168.9    -34146.1
      AIC |    -35215.6    -35176.9    -34156.1
      SBC |    -35228.3    -35193.9    -34177.3
   unused |          .           .           .
   unused |          .           .           .
```

- Trust-Region optimization:

```
  optn = [ "idvar"       1 ,
           "adapt"         ,
           "nquad"      11 ,
           "tech"     "tr" ,
           "maxit"     200 ,
           "popt"        2 ,
           "gtol"    1.e-6 ];
  < gof,parm,cov,yhat > = mixregv(posmood,modl,optn,xind,uind,wind);

GOF=
          |   WithoutSc   WithScale   WithRandS
-----------------------------------------------
  RetCode |     6.00000     6.00000     6.00000
    N_obs |     17514.0     17514.0     17514.0
    N_par |    10.00000    10.00000    10.00000
  OptTime |     3.00000     4.00000     62.0000
  MaxAbsG |    5.7e-004    3.7e-006    9.7e-004
   LogLik |    -35209.6    -35168.9    -34146.1
      AIC |    -35215.6    -35176.9    -34156.1
      SBC |    -35228.3    -35193.9    -34177.3
   unused |          .           .           .
   unused |          .           .           .
```

7. Example 2 by Hedeker (2012): Positive Mood Data $N = 17514, n = 5$: quadratic relationship between the random location (mean) and the WS variance.

```
print "Tech=Hedeker NRR: COVREL=QUAD";
optn = [ "idvar"        1 ,
         "covrel"   "qua" ,
         "adapt"          ,
         "nquad"       11 ,
         "tech"     "hed" ,
         "maxit"      200 ,
         "pcov"           ,
         "popt"         2 ,
         "conv"     1.e-5 ];
< gof,parm,cov,yhat > = mixregv(posmood,modl,optn,xind,uind,wind);
```

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

                Number Valid Observations17514
                Response Variable          Y[2]
                N Independent Variables      3
                Mean and WS Variance: Quadratic
                Number Level-2 Clusters    515
                Number Fixed Effects         3
                Number Random Effects        2
                Number Error Variances       3
                Quad Points per Dimension   11
                Nonadaptive Quadrature
                Skip Cases with Missing Values
                Subject Variable Column      1
                Significance Level:  0.0500000


               *******************************
               Starting Values for Optimization
               *******************************

          BETA: mean model regression coefficients
                                     1
                    Beta_1   7.052064259
                    Beta_2  -0.449182713
                    Beta_3  -0.193014768


 ALPHA: BS variance log-linear model regression coefficients
                                     1
                   Alpha_1   0.330756712
                   Alpha_2   0.069095230
```

```
   TAU: WS variance log-linear model regression coefficients
                                      1
                        Tau_1   1.101568073
                        Tau_2   0.037123289
                        Tau_3   0.069989704


Random linear location effects on WS variance. . . .   -0.222043
Random quadratic location effects on WS variance . .    0.055511
Scale standard deviation . . . . . .                    0.468910

                    ******************************
                    Model Without Scale Parameters
                    ******************************

   Start of Optimization: Maximum Iterations=200 Xtol=1e-005

       Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

         1B -35210.1857  11.3584891  0.12908420           0
         2B -35209.5910  0.46821097  0.00309441           0
         3B -35209.5902  5.657e-004  4.440e-006           0
         4B -35209.5902  8.186e-009  5.609e-011           0
   Succesful Optimization after 4 Iterations, crit=-35209.6


            Results of Hedeker Optimization Algorithm
  Iterations . . . . .       4   Ridge . . . . . .    0.000000
  CompTime. . . .            2   MaxAbsGrad. . . .    0.000000
  LogLik. . . . . -35209.590201   Times-2 . . . . .70419.180402
  AIC . . . . . . -35215.590201   Times-2 . . . . .70431.180402
  SBC . . . . . . -35228.322702   Times-2 . . . . .70456.645403


     Variable    Estimate  AsStdError      Zvalue     p-Value
   --------------BETA (regression coefficients)--------------
       Beta_1  7.04729903  0.08335523  84.5453700           0
       Beta_2 -0.44907586  0.02827235 -15.8839231  8.189e-057
       Beta_3 -0.18516736  0.10882435 -1.70152505  0.08884444
   -----ALPHA (BS Variance Parameters: Log-linear model)-----
      Alpha_1  0.36208742  0.10039358  3.60667892  3.101e-004
      Alpha_2 -0.05689012  0.13462083 -0.42259521  0.67259062
   ------TAU (WS Variance Parameters: Log-linear model)------
        Tau_1  1.10156829  0.01084664  101.558448           0
```

Variable	Estimate	AsStdError	Lower CL	Upper CL
BETA (regression coefficients)				
Beta_1	7.04729903	0.08335523	6.88392577	7.21067228
Beta_2	-0.44907586	0.02827235	-0.50448865	-0.39366307
Beta_3	-0.18516736	0.10882435	-0.39845916	0.02812445
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	0.36208742	0.10039358	0.16531961	0.55885523
Alpha_2	-0.05689012	0.13462083	-0.32074210	0.20696186
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	1.10156829	0.01084664	1.08030926	1.12282732

Estimated Covariance Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	0.006948095	-0.000384316	-0.006775311	-2.7767e-005
Beta_2	-0.000384316	0.000799326	2.4949e-005	2.9322e-005
Beta_3	-0.006775311	2.4949e-005	0.011842739	1.4569e-005
Alpha_1	-2.7767e-005	2.9322e-005	1.4569e-005	0.010078872
Alpha_2	2.9348e-005	-3.2636e-005	-2.8689e-005	-0.010078378
Tau_1	3.2138e-007	-3.1357e-007	3.9823e-008	-8.2531e-006

Estimated Covariance Matrix of Estimates

	Alpha_2	Tau_1
Beta_1	2.9348e-005	3.2138e-007
Beta_2	-3.2636e-005	-3.1357e-007
Beta_3	-2.8689e-005	3.9823e-008
Alpha_1	-0.010078378	-8.2531e-006
Alpha_2	0.018122768	-5.2420e-007
Tau_1	-5.2420e-007	0.000117650

Estimated Correlation Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	1.000000000	-0.163077352	-0.746913472	-0.003318118
Beta_2	-0.163077352	1.000000000	0.008109085	0.010330445
Beta_3	-0.746913472	0.008109085	1.000000000	0.001333523
Alpha_1	-0.003318118	0.010330445	0.001333523	1.000000000
Alpha_2	0.002615377	-0.008574829	-0.001958261	-0.745714235
Tau_1	0.000355459	-0.001022520	3.3737e-005	-0.007579033

Estimated Correlation Matrix of Estimates

```
                 Alpha_2        Tau_1
Beta_1     0.002615377  0.000355459
Beta_2    -0.008574829 -0.001022520
Beta_3    -0.001958261  3.3737e-005
Alpha_1   -0.745714235 -0.007579033
Alpha_2    1.000000000 -0.000358998
Tau_1     -0.000358998  1.000000000


            ***************************
            Model With Scale Parameters
            ***************************

  Start of Optimization: Maximum Iterations=200 Xtol=1e-005

      Iter     Loglike  MaxAbsGrad   MaxChange       Ridge

        1B -35200.3459  503.266565  0.03459346  0.10000000
        2B -35185.2511  232.916573  0.02125982  0.10000000
        3B -35179.6934  153.370620  0.01622187  0.10000000
        4B -35176.1998  118.366418  0.01334928  0.10000000
        5B -35173.8515  96.4029796  0.06227509           0
        6B -35168.9312  5.02009738  0.00131601           0
        7B -35168.9287  0.00255205  7.714e-007           0
        8B -35168.9287  1.461e-009  4.936e-012           0
  Succesful Optimization after 8 Iterations, crit=-35168.9


           Results of Hedeker Optimization Algorithm
Iterations . . . . .       8   Ridge . . . . . .    0.000000
CompTime. . . .            5   MaxAbsGrad. . . .    0.000000
LogLik. . . . . -35168.928668   Times-2 . . . . .70337.857336
AIC . . . . . . -35176.928668   Times-2 . . . . .70353.857336
SBC . . . . . . -35193.905336   Times-2 . . . . .70387.810672


    Variable    Estimate  AsStdError      Zvalue     p-Value
  --------------BETA (regression coefficients)--------------
      Beta_1  7.04310322  0.08324068  84.6113120           0
      Beta_2 -0.44251872  0.02821811 -15.6820796  2.006e-055
      Beta_3 -0.18312803  0.10876202 -1.68374980  0.09223003
  -----ALPHA (BS Variance Parameters: Log-linear model)-----
     Alpha_1  0.36900397  0.09963038  3.70372941  2.125e-004
     Alpha_2 -0.07019510  0.13438103 -0.52235866  0.60142063
  ------TAU (WS Variance Parameters: Log-linear model)------
       Tau_1  0.95555142  0.02010575  47.5262771           0
       Tau_2  0.06034293  0.02226427  2.71030375  0.00672216
```

```
     Tau_3  0.19370818  0.02205255  8.78393519  1.579e-018
```

```
  Variable    Estimate  AsStdError    Lower CL    Upper CL
--------------BETA (regression coefficients)--------------
    Beta_1  7.04310322  0.08324068  6.87995448  7.20625195
    Beta_2 -0.44251872  0.02821811 -0.49782521 -0.38721223
    Beta_3 -0.18312803  0.10876202 -0.39629766  0.03004161
-----ALPHA (BS Variance Parameters: Log-linear model)-----
   Alpha_1  0.36900397  0.09963038  0.17373202  0.56427593
   Alpha_2 -0.07019510  0.13438103 -0.33357709  0.19318689
------TAU (WS Variance Parameters: Log-linear model)------
     Tau_1  0.95555142  0.02010575  0.91614488  0.99495797
     Tau_2  0.06034293  0.02226427  0.01670577  0.10398009
     Tau_3  0.19370818  0.02205255  0.15048598  0.23693039
```

Estimated Covariance Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	0.006929011	-0.000372639	-0.006766384	-2.6702e-005
Beta_2	-0.000372639	0.000796262	2.4970e-005	2.9466e-005
Beta_3	-0.006766384	2.4970e-005	0.011829177	1.3753e-005
Alpha_1	-2.6702e-005	2.9466e-005	1.3753e-005	0.009926213
Alpha_2	2.8648e-005	-3.3493e-005	-2.8364e-005	-0.009926296
Tau_1	8.8048e-006	-1.0363e-005	-6.9057e-006	-1.4290e-005
Tau_2	-7.4884e-006	5.7744e-007	1.2560e-005	-7.6903e-006
Tau_3	-8.6005e-006	1.6730e-005	1.9568e-006	1.7927e-005

Estimated Covariance Matrix of Estimates

	Alpha_2	Tau_1	Tau_2	Tau_3
Beta_1	2.8648e-005	8.8048e-006	-7.4884e-006	-8.6005e-006
Beta_2	-3.3493e-005	-1.0363e-005	5.7744e-007	1.6730e-005
Beta_3	-2.8364e-005	-6.9057e-006	1.2560e-005	1.9568e-006
Alpha_1	-0.009926296	-1.4290e-005	-7.6903e-006	1.7927e-005
Alpha_2	0.018058262	1.6440e-005	3.4332e-006	-3.4385e-005
Tau_1	1.6440e-005	0.000404241	-0.000244509	-0.000296479
Tau_2	3.4332e-006	-0.000244509	0.000495698	2.5869e-005
Tau_3	-3.4385e-005	-0.000296479	2.5869e-005	0.000486315

Estimated Correlation Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	1.000000000	-0.158644578	-0.747383967	-0.003219686

Beta_2	-0.158644578	1.000000000	0.008136204	0.010480935
Beta_3	-0.747383967	0.008136204	1.000000000	0.001269233
Alpha_1	-0.003219686	0.010480935	0.001269233	1.000000000
Alpha_2	0.002561089	-0.008832518	-0.001940687	-0.741408290
Tau_1	0.005260933	-0.018266298	-0.003158005	-0.007133599
Tau_2	-0.004040569	0.000919120	0.005186790	-0.003466906
Tau_3	-0.004685233	0.026885052	0.000815848	0.008159244

Estimated Correlation Matrix of Estimates

	Alpha_2	Tau_1	Tau_2	Tau_3
Beta_1	0.002561089	0.005260933	-0.004040569	-0.004685233
Beta_2	-0.008832518	-0.018266298	0.000919120	0.026885052
Beta_3	-0.001940687	-0.003158005	0.005186790	0.000815848
Alpha_1	-0.741408290	-0.007133599	-0.003466906	0.008159244
Alpha_2	1.000000000	0.006084896	0.001147513	-0.011602929
Tau_1	0.006084896	1.000000000	-0.546217233	-0.668673801
Tau_2	0.001147513	-0.546217233	1.000000000	0.052688752
Tau_3	-0.011602929	-0.668673801	0.052688752	1.000000000

****
Model With Random Scale
****

Start of Optimization: Maximum Iterations=200 Xtol=1e-005

Iter	Loglike	MaxAbsGrad	MaxChange	Ridge
1A	-34243.2338	907.502320	0.15178863	0.20000000
2B	-34214.7848	854.997947	0.06285516	0.20000000
3B	-34175.5780	624.035974	0.05150724	0.20000000
4B	-34154.0146	444.722156	0.04492957	0.20000000
5B	-34141.9097	312.402994	0.04057023	0.20000000
6B	-34135.1057	218.053337	0.03255922	0.20000000
7B	-34128.8211	92.5226742	0.03237981	0
8B	-34126.8281	9.97935786	0.01409645	0
9B	-34126.7895	0.33797150	2.369e-004	0
10B	-34126.7893	0.00223082	1.839e-005	0
11B	-34126.7893	8.843e-005	6.535e-007	0
12B	-34126.7893	4.554e-006	2.962e-008	0

Succesful Optimization after 12 Iterations, crit=-34126.8

Results of Hedeker Optimization Algorithm

Iterations . . . . . 12 Ridge 0.000000

```
CompTime. . . .          112   MaxAbsGrad. . . .    0.000005
LogLik. . . . . -34126.789338   Times-2 . . . . .68253.578676
AIC . . . . . . -34137.789338   Times-2 . . . . .68275.578676
SBC . . . . . . -34161.132256   Times-2 . . . . .68322.264512
```

Variable	Estimate	AsStdError	Zvalue	p-Value
BETA (regression coefficients)				
Beta_1	6.96177205	0.07898110	88.1447888	0
Beta_2	-0.35020557	0.02410949	-14.5256309	8.337e-048
Beta_3	-0.11450726	0.10290345	-1.11276406	0.26580977
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	0.31998809	0.09308268	3.43767580	5.867e-004
Alpha_2	0.02906407	0.12293506	0.23641803	0.81310830
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	0.89823431	0.05139644	17.4765847	2.161e-068
Tau_2	0.08690145	0.02465887	3.52414538	4.249e-004
Tau_3	0.19531957	0.06065448	3.22019998	0.00128101
Random Scale Parameters: Cholesky Factorization Elements				
Spar_1	-0.22475798	0.03183972	-7.05904384	1.677e-012
Spar_2	-0.12686175	0.02137412	-5.93529612	2.933e-009
Spar_3	0.57078808	0.02193140	26.0260708	2.511e-149

Variable	Estimate	AsStdError	Lower CL	Upper CL
BETA (regression coefficients)				
Beta_1	6.96177205	0.07898110	6.80697194	7.11657215
Beta_2	-0.35020557	0.02410949	-0.39745930	-0.30295183
Beta_3	-0.11450726	0.10290345	-0.31619432	0.08717980
ALPHA (BS Variance Parameters: Log-linear model)				
Alpha_1	0.31998809	0.09308268	0.13754938	0.50242679
Alpha_2	0.02906407	0.12293506	-0.21188423	0.27001236
TAU (WS Variance Parameters: Log-linear model)				
Tau_1	0.89823431	0.05139644	0.79749913	0.99896949
Tau_2	0.08690145	0.02465887	0.03857095	0.13523196
Tau_3	0.19531957	0.06065448	0.07643896	0.31420017
Random Scale Parameters: Cholesky Factorization Elements				
Spar_1	-0.22475798	0.03183972	-0.28716269	-0.16235328
Spar_2	-0.12686175	0.02137412	-0.16875427	-0.08496924
Spar_3	0.57078808	0.02193140	0.52780333	0.61377282

Estimated Covariance Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	0.006238014	-0.000273089	-0.005827419	-0.000626101
Beta_2	-0.000273089	0.000581268	2.4728e-005	1.9150e-005

Beta_3	-0.005827419	2.4728e-005	0.010589120	0.000861679
Alpha_1	-0.000626101	1.9150e-005	0.000861679	0.008664386
Alpha_2	0.000751457	-1.6586e-005	-0.001058050	-0.008064185
Tau_1	-0.001127100	1.2726e-005	0.001094592	-0.000511365
Tau_2	1.6909e-005	-1.8786e-005	6.8034e-006	-8.8466e-006
Tau_3	0.000964534	1.2668e-005	-0.001775437	0.000794422
Spar_1	-0.000544234	-1.0611e-005	-6.3762e-007	-0.000442620
Spar_2	7.2679e-005	-1.3283e-005	-0.000120978	-0.000435425
Spar_3	5.0843e-005	1.8268e-005	-8.0255e-005	7.8019e-005

Estimated Covariance Matrix of Estimates

	Alpha_2	Tau_1	Tau_2	Tau_3
Beta_1	0.000751457	-0.001127100	1.6909e-005	0.000964534
Beta_2	-1.6586e-005	1.2726e-005	-1.8786e-005	1.2668e-005
Beta_3	-0.001058050	0.001094592	6.8034e-006	-0.001775437
Alpha_1	-0.008064185	-0.000511365	-8.8466e-006	0.000794422
Alpha_2	0.015113029	0.000938675	3.0689e-006	-0.001567920
Tau_1	0.000938675	0.002641594	-0.000293252	-0.002151114
Tau_2	3.0689e-006	-0.000293252	0.000608060	2.2719e-005
Tau_3	-0.001567920	-0.002151114	2.2719e-005	0.003678966
Spar_1	-8.6877e-006	5.3149e-005	-3.3494e-006	5.0489e-005
Spar_2	-6.1962e-005	-0.000431220	-2.0288e-006	7.5148e-005
Spar_3	-0.000108631	-3.9262e-005	1.2535e-005	3.2028e-005

Estimated Covariance Matrix of Estimates

	Spar_1	Spar_2	Spar_3
Beta_1	-0.000544234	7.2679e-005	5.0843e-005
Beta_2	-1.0611e-005	-1.3283e-005	1.8268e-005
Beta_3	-6.3762e-007	-0.000120978	-8.0255e-005
Alpha_1	-0.000442620	-0.000435425	7.8019e-005
Alpha_2	-8.6877e-006	-6.1962e-005	-0.000108631
Tau_1	5.3149e-005	-0.000431220	-3.9262e-005
Tau_2	-3.3494e-006	-2.0288e-006	1.2535e-005
Tau_3	5.0489e-005	7.5148e-005	3.2028e-005
Spar_1	0.001013768	8.5385e-005	1.3208e-005
Spar_2	8.5385e-005	0.000456853	4.8032e-006
Spar_3	1.3208e-005	4.8032e-006	0.000480986

Estimated Correlation Matrix of Estimates

	Beta_1	Beta_2	Beta_3	Alpha_1
Beta_1	1.000000000	-0.143414253	-0.717006618	-0.085163220

Beta_2	-0.143414253	1.000000000	0.009967095	0.008533012
Beta_3	-0.717006618	0.009967095	1.000000000	0.089959427
Alpha_1	-0.085163220	0.008533012	0.089959427	1.000000000
Alpha_2	0.077393628	-0.005596155	-0.083637409	-0.704718790
Tau_1	-0.277655530	0.010270154	0.206961417	-0.106888067
Tau_2	0.008682059	-0.031598939	0.002681165	-0.003854218
Tau_3	0.201340701	0.008662488	-0.284454305	0.140708164
Spar_1	-0.216417802	-0.013822770	-0.000194608	-0.149345819
Spar_2	0.043052500	-0.025776588	-0.055003368	-0.218855070
Spar_3	0.029352150	0.034550014	-0.035561183	0.038217693

Estimated Correlation Matrix of Estimates

	Alpha_2	Tau_1	Tau_2	Tau_3
Beta_1	0.077393628	-0.277655530	0.008682059	0.201340701
Beta_2	-0.005596155	0.010270154	-0.031598939	0.008662488
Beta_3	-0.083637409	0.206961417	0.002681165	-0.284454305
Alpha_1	-0.704718790	-0.106888067	-0.003854218	0.140708164
Alpha_2	1.000000000	0.148561528	0.001012348	-0.210273821
Tau_1	0.148561528	1.000000000	-0.231384555	-0.690029037
Tau_2	0.001012348	-0.231384555	1.000000000	0.015190026
Tau_3	-0.210273821	-0.690029037	0.015190026	1.000000000
Spar_1	-0.002219535	0.032478353	-0.004266033	0.026143801
Spar_2	-0.023580791	-0.392533837	-0.003849312	0.057964692
Spar_3	-0.040291162	-0.034831170	0.023178245	0.024077146

Estimated Correlation Matrix of Estimates

	Spar_1	Spar_2	Spar_3
Beta_1	-0.216417802	0.043052500	0.029352150
Beta_2	-0.013822770	-0.025776588	0.034550014
Beta_3	-0.000194608	-0.055003368	-0.035561183
Alpha_1	-0.149345819	-0.218855070	0.038217693
Alpha_2	-0.002219535	-0.023580791	-0.040291162
Tau_1	0.032478353	-0.392533837	-0.034831170
Tau_2	-0.004266033	-0.003849312	0.023178245
Tau_3	0.026143801	0.057964692	0.024077146
Spar_1	1.000000000	0.125465464	0.018914349
Spar_2	0.125465464	1.000000000	0.010246600
Spar_3	0.018914349	0.010246600	1.000000000