CMAT© Newsletter: December 2014

Wolfgang M. Hartmann

December 2014

Contents

1 General Remarks

Most of my time I spent on a working version of the `LPASL1` algorithm for LP which was developed many years ago by Mustafa Pinar, Niels Brun, and Kaj Madsen. Especially the setup of the standard LP in form of $\mathbf{A}x = b$ subject to $0 \leq x \leq u$ had to be worked over and I rewrote some preprocessor similar to that used by the `pcx` program. Why did I work on that old algorithm? The older Fortran version showed for a small number of the Netlib examples very good results, but was not working on most of the examples. I was curious to find out what was happening and rewrote the algorithm in C language making the testing easier for me.

Inbetween the development of a general LP driver I have been working designing an interface to the huge *Clp* (Coin LP) and *Cbc* (Coin branch and cut) software designed by John Forrest in the *Coin* package for solving LPs.

The new `mpsread()` and `mpswrite()` functions permit the transformation of the common LP matrix model into an MPS file and vice versa. For those who don't know about MPS files, there are pretty nice descriptions on the internet. The LP function can now be called with an input .mps file specifying the optimization model.

Aside from my work on LP algorithms I implemented some algorithms for Fisher's exact statistical test of count data. The implementation of functions is similar of those of the CRAN functions `fisher.test`, `chisq.test`, `binom.test`, and `poisson.test` as well as the packages `exact2x2` and `exactci` by Michael P. Fay.

Additionally I added the new function `xctlog()` for exact logistic regression which is a modification of the main function of the `elrm` package by Zamar, Graham, & McNency (2013). Luckily I was able to enhance the implementation and hope that my version of the algorithm is able to solve larger problems than the `elrm` function in R.

A number of bugs were fixed. Among others a serious one when transposing sparse matrices with row and column names.

1.1 New Functions

contsim() constructs random two-way contingency tables with given row and column sums.

filestat() returns a vector of file statistics

mpsread() read an MPS file and create the matrix form of the LP model (note, the `lp()` function can use either of the two inputs and should yield the same result)

mpswrite() for a given matrix specification of an LP model the function writes a MPS file which can be used as input for `lp()` and `pcx()`.

pritfile() list text file linewise in .lst output and/or return as a vector of strings

xctbinom() computes estimates, probabilities, and confidence intervals for the *exact* Binomial test

xctbip1() computes $P(alt) = p1$ of Binomial test for given $p0$, power, and sample size

xctbipow() computes power of Binomial test for given $p0$, $p1$, and sample size

xctbissz() computes sample size of Binomial test for given $p0$, $p1$, and power

xctfishr() computes estimates, probabilities, and confidence intervals for the *exact* Fisher test of 2×2 contingency tables

xctfipow() computes the power of the *exact* Fisher test or the (paired) McNemar test when

- $p0$ true event rate in control group
- $p1$ true event rate in treatment group
- $n0$ sample size in control group
- $n1$ sample size in treatment group

are specified. My implementation is based on an algorithm by Fay (2014) in R package `exact2x2`.

xctfissz() computes the necessary sample size for specified power of the *exact* Fisher test or the (paired) McNemar test when

- $p0$ true event rate in control group
- $p1$ true event rate in treatment group
- *power* required power
- $n1\_over\_n0$ ratio of sample sizes (treatment over control group)

are specified. My implementation is based on an algorithm by Fay (2014) in R package `exact2x2`.

xcthybr() computes Fisher's exact test probability for (small) $m \times n$ contingency tables using the hybrid method by Mehta & Patel (1986).

xctlog() solves the exact logistic regression model for small data sets. My implementation is based on an algorithm by Zamar et al (2013).

xctmcnem() computes estimates, probabilities, and confidence intervals for the *exact* McNemar test of 2×2 contingency tables

xctpoiss() computes estimates, probabilities, and confidence intervals for the *exact* Poisson test

xctsimu() computes Fisher's exact test probability for (small) $m \times n$ contingency tables by MC simulation, in addition the probability of the χ^2 values of simulated tables and the common asymptotic probability of χ^2 test can be computed.

2 Extensions to Various Functions

2.1 Extensions of `round()` Function

The `round()` function was extended from one to two input arguments. Now, an optional second input argument can be used for specifying the integer decimal digit number of the value of the first argument, where rounding has to be applied.

i = round(z<,ndig>)

Purpose: Rounds its (first) argument toward the nearest integer. If a second input argument is specified, the rounding is applied at a specific decimal digit number. If z is `complex`, the argument is first replaced by its magnitude, that means, `round(abs(z))` is computed.

Input:
1. The argument z must be numeric scalar, vector, or matrix. These are the values that should be rounded.
2. An optional second integer argument is specified for indicating the decimal digit number where the values of the first arguments are rounded.

Output:
1. A missing value is returned if **z** contains any string data or missing values.
2. The data type of the second input argument must be integer.
3. If the first input argument is vector or matrix, the function is computed elementwise.

Relationships: ceil(), fix(), floor()

Examples:
1. Only one argument: For

```
x = [ -6.1 -1.9 2.2 5.9 ];
xx = x |> ceil(x) |> fix(x) |> floor(x) |> round(x);
rnam = [ "x", "ceil", "fix", "floor", "round"];
print poptn rowname=rnam : xx;
```

we obtain:

function	-6.1	-1.9	2.2	5.9	function	-6.1	-1.9	2.2	5.9
ceil(x)	-6.	-1.	3.	6.	floor(x)	-7.	-2.	2.	5.
fix(x)	-6.	-1.	2.	5.	round(x)	-6.	-2.	2.	6.

2. Two arguments: for input

```
x =  6.111;
xrnd = round(x, 0) -> round(x, 1) -> round(x, 2) -> round(x,-1);
print "Rund x=",x," is ",xrnd;
```

```
x =  6.555;
xrnd = round(x, 0) -> round(x, 1) -> round(x, 2) -> round(x,-1);
print "Rund x=",x," is ",xrnd;

x = -6.111;
xrnd = round(x, 0) -> round(x, 1) -> round(x, 2) -> round(x,-1);
print "Rund x=",x," is ",xrnd;

x = -6.555;
xrnd = round(x, 0) -> round(x, 1) -> round(x, 2) -> round(x,-1);
print "Rund x=",x," is ",xrnd;
```

we obtain:

```
Rund x= 6.1110 is
 R |         1         2         3         4
-------------------------------------------
 1 |    6.0000    6.1000    6.1100   10.0000
Rund x= 6.5550 is
 R |         1         2         3         4
-------------------------------------------
 1 |    7.0000    6.6000    6.5500   10.0000
Rund x=-6.1110 is
 R |         1         2         3         4
-------------------------------------------
 1 |   -6.0000   -6.1000   -6.1100  -10.0000
Rund x=-6.5550 is
 R |         1         2         3         4
-------------------------------------------
 1 |   -7.0000   -6.6000   -6.5500  -10.0000
```

3. Two arguments: for input

```
x = [ -6.11 -1.99 2.22 5.99 ];
print "Round(x,1): with x=", x, " is y =", round(x,1);
```

```
 is y =
 R |         1         2         3         4
-------------------------------------------
 1 |   -6.1000   -2.0000    2.2000    6.0000
```

3 New Developments

3.1 The `contsim()` Function

< cont,ntot > = contsim(rsum<,csum<,ntab>>)

Purpose: The function `contsim()` constructs a number of $n \times m$ random two-way contingency tables with specified row and column sums. The algorithm is almost the same as that of the Fortran program `RCOND` by M. Patefield. Some of the tables generated may be the same. The `srand()` function should be used for setting the seed of the random generator whenever a reproducable set of tables should be generated. Note, that it is possible to specify row and column sums so that no corresponding table exists.

Input: **rsum** a vector of n integers specifying the row sums.

csum a vector of m integers specifying the column sums. If this vector is not specified (missing), the column sums default to the row sums.

K an integer specifying the number of tables to be returned (if possible), default=1.

Note, that the sums or row sums and column sums must be the same.

Output: **cont** For $K = 1$ returns a single contingency table, for $K > 1$ returns a list of K contingency tables with the specified row and column sums.

ntot

Restrictions:

1. The specified row and column sum vectors should not contain any missing values or string data.
2. The sums of row and column sums must match.

Relationships: xctfishr()

Examples: Create five 3×3 tables:

```
rsum = [ 3 2 5 ];
csum = [ 1 3 6 ];
< cont,ntot > = contsim(rsum,csum,5);
```

```
**************************
cont (List with 5 Entries)
**************************

     cont[1]:
     *********

  Dense Matrix (3 by 3)

  |       1      2      3
-------------------------
1 |       0      0      3
2 |       0      1      1
3 |       1      2      2


     cont[2]:
     *********

  Dense Matrix (3 by 3)

  |       1      2      3
-------------------------
1 |       0      0      3
2 |       0      0      2
3 |       1      3      1


     cont[3]:
     *********

  Dense Matrix (3 by 3)

  |       1      2      3
-------------------------
1 |       0      1      2
2 |       0      0      2
3 |       1      2      2


     cont[4]:
     *********

  Dense Matrix (3 by 3)

  |       1      2      3
-------------------------
1 |       0      2      1
2 |       0      0      2
3 |       1      1      3


     cont[5]:
     *********

  Dense Matrix (3 by 3)

  |       1      2      3
-------------------------
1 |       1      1      1
2 |       0      0      2
3 |       0      2      3
```

3.2 The `filestat()` Function

stat = filestat("fpath")

Purpose: The function `filestat()` returns a vector of file statistics.

Input: The only input argument argument is a string `"fpath"` specifying the location of a readable file.

Output: The only output argument is a vector of integer file statistics:

1. bit mask for file mode information
2. size of file in bytes
3. hard links
4. user ID: identifier of user that owns file (is zero on Windows)
5. groupID: identifier of group that owns file (is zero on Windows)
6. date of last access to file ((year*100 + month) * 100 + day)
7. time of last access to file ((hour*60 + min) * 60 + sec)
8. date of last modification of file
9. time of last modification of file
10. date of creation of file
11. time of creation of file
12. drive number of disk containing the file ('A'= 0)

Restrictions: 1. The `"fpath"` argument must point to a readable file

Relationships:

Examples: 1. See tnlp/tlp.inp: Example by Madsen and Pinar:

```
cx = cons(5,1,0.);
cx[3] = 6.;
lau = [ -8.  -6. -1. -6.  2.  0.  -8. ,
        -5.   0.  1. -6.  0.  2.  -5. ];
lubc = cons(5,2,0.);
lubc[,2] = 10.;
print "Transform matrix notation into MPS file";
prob = mpswrite("pinar1.mps",cx,lau,lubc);


stat = filestat("pinar1.mps");
print "Stat=", stat;
```

```
Stat=
              |          1
--------------------------
     FileMode |      33206
     FileSize |        625
    HardLinks |          1
       UserID |          0
      GroupID |          0
  LastAccDate |   20140626
  LastAccTime |      47329
  LastModDate |   20140627
```

```
LastModTime |      84236
CreationDat |   20140626
CreationTim |      47329
DriveNumber |          2
```

3.3 The `mpsread()` Function

< c,lau,lubc > = mpsread("fpath"<,optn>)

Purpose: The function `mpsread()` reads a MPS (Mathematical Programming System) file and returns the matrix notation as input for the `lp()` function for specifying a LP model. See Murtagh(1981), Nazareth (1987) or other standard literature on Linear Programming for more about the syntax of MPS files.

This function is useful for for specifying the model for a LP optimization. The `lp()` and `pcx()` functions are trying to minimize or maximize the linear objective (cost) function $c^T x = \sum c_j x_j$ in n variables x_j subject to a set of m general linear inequality (or equality) constraints

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad , i = 1, \ldots, m$$

and/or simple boundary constraints $l_j \leq x_j \leq u_j$.

Linear constraints in `lau` maybe reordered from the order of rows in the .mps file (see options below).

- empty rows in the .mps file are not stored in `lau`
- rows (constraints) with only one nonzero coefficient are reformulated in the bounds `lubc` return.
- duplicate rows with more than one nonzero coefficient in the .mps file are treated as one linear constraint in `lau`, where the max of lower and the min of upper values b is used. This is also done for rows which are different only by a positive or negative factor.
- depending on `optn[3]` linearly dependent equality constraints can be removed.

Input: "fpath" the first input argument argument is a string specifying the location of the input MPS file.

optn must be either a missing value, an int scalar, or a numeric vector of runtime options:

1. nonnegative int: amount of printed output where 0 is the default for no printed output

2. unequal zero specifies that the linear constraints in matrix `lau` are ordered so that equality constraints are stored in the first rows and inequality constraints in the last. Default is 0, meaning that the order of linear constraints in `lau` is close to the order of rows in the .mps file.
3. unequal zero specifies that all equality constraints are checked for linear depndencies. A computationally fast QR decomposition is used for detecting the rank of **A** and linearly dependent equality constraints are removed. Default is 0, meaning that the test is not computed.

Output: **c** is an n vector of the coefficients of the objective function.

lau is an $m \times (n+2)$ (frequently sparse) matrix of general linear constraints, the first column contains the lower range and the last column contains the upper range. The constraint is an equality constraint when lower and upper range are equal and an inequality constraint when the lower is smaller than then upper range. Missing values for the ranges stand for either $-\infty$ or $+\infty$.

lubc is an $n \times 2$ matrix of lower (in first column) and upper (in second column) boundary constraints.

Restrictions: 1.

Relationships: pcx(), lp()

Examples: The following examples use both, the `mpsread()` and the `mpswrite()` function.

1. 1. Example by Madsen and Pinar:
   - 1.1 Transform from matrix notation to MPS file:

```
print "*** 1. Example by Madsen and Pinar ***";
cx = cons(5,1,0.);
cx[3] = 6.;
lau = [ -8.  -6. -1. -6.  2.  0.  -8. ,
        -5.   0.  1. -6.  0.  2.  -5. ];
lubc = cons(5,2,0.);
lubc[,2] = 10.;
< prob,row,col,rhs,rng,bnd > = mpswrite("pinar1.mps",cx,lau,lubc);
```

The following is a listing of the output file `pinar1.mps`:

```
NAME          pinar1
ROWS
 E  _r1
 E  _r2
 N  COST
COLUMNS
```

```
    _x1       _r1       -6.
    _x2       _r1       -1.            _r2       1.
    _x3       _r1       -6.            _r2       -6.
    _x3       COST      6.
    _x4       _r1       2.
    _x5       _r2       2.
RHS
    RHS       _r1       -8.            _r2       -5.

RANGES
BOUNDS
 UP BOUND     _x1       10.
 UP BOUND     _x2       10.
 UP BOUND     _x3       10.
 UP BOUND     _x4       10.
 UP BOUND     _x5       10.
ENDATA
```

- 1.2 Transform from MPS file to matrix notation :

```
< c2,lau2,lbc2 > = mpsread("pinar1.mps");
print "C2=", c2;
print "Lau2=", lau2;
print "LBC2=", lbc2;
```

```
C2=
   |   _x1        _x2        _x3        _x4        _x5
-------------------------------------------------------------
 1 |    0.00000    0.00000     6.0000    0.00000    0.00000

Lau2=
           |   LowerRHS   _x1        _x2        _x3
-----------------------------------------------------------
  _r1      |    -8.0000    -6.0000   -1.00000    -6.0000
  _r2      |    -5.0000    0.00000    1.00000    -6.0000

           |   _x4        _x5        UpperRHS
-------------------------------------------
  _r1      |     2.0000    0.00000    -8.0000
  _r2      |    0.00000     2.0000    -5.0000

LBC2=
           |         1         2
--------------------------------
  _x1      |   0.00000   10.0000
```

```
_x2      |   0.00000   10.0000
_x3      |   0.00000   10.0000
_x4      |   0.00000   10.0000
_x5      |   0.00000   10.0000
```

2. 2. Example by Madsen and Pinar

- 2.1 Transform from matrix notation to MPS file:

```
print "*** 2. Example by Madsen and Pinar ***";
cx = cons(4,1,0.);
cx[1] = cx[4] = 1.;
lau = [   .   2.  2.  1.  0.   7. ,
          .   2.  1.  2.  0.   4. ,
          .   0. -1.  0. -1.  -1. ,
         3.   0.  1.  1.  1.   3. ];
lubc = cons(4,2,0.);
lubc[,2] = 2.;
< prob,row,col,rhs,rng,bnd > = mpswrite("pinar2.mps",cx,lau,lubc);
```

The following is a listing of the output file `pinar2.mps`:

```
NAME          pinar2
ROWS
 E  _r1
 L  _r2
 L  _r3
 L  _r4
 N  COST
COLUMNS
    _x1       _r2       2.             _r3       2.
    _x1       COST      1.
    _x2       _r1       1.             _r2       2.
    _x2       _r3       1.             _r4       -1.
    _x3       _r1       1.             _r2       1.
    _x3       _r3       2.
    _x4       _r1       1.             _r4       -1.
    _x4       COST      1.
RHS
    RHS       _r1       3.             _r2       7.
    RHS       _r3       4.             _r4       -1.

RANGES
BOUNDS
 UP BOUND     _x1       2.
 UP BOUND     _x2       2.
 UP BOUND     _x3       2.
```

```
 UP BOUND     _x4       2.
ENDATA
```

- 2.2 Transform from MPS file to matrix notation :

```
< c2,lau2,lbc2 > = mpsread("pinar2.mps");
print "C2=", c2;
print "Lau2=", lau2;
print "LBC2=", lbc2;
```

```
C2=
   |   _x1        _x2        _x3        _x4
-----------------------------------------------
 1 |    1.00000    0.00000    0.00000    1.00000
```

The order of the m general linear constraints is changed in such a way, that equality constraints are listed first and the inequality constraints last:

```
Lau2=
           |   LowerRHS   _x1        _x2
-------------------------------------------
  _r1      |     3.0000    0.00000    1.00000
  _r2      |     .          2.0000     2.0000
  _r3      |     .          2.0000     1.00000
  _r4      |     .         0.00000   -1.00000

           |   _x3        _x4        UpperRHS
-------------------------------------------
  _r1      |    1.00000    1.00000     3.0000
  _r2      |    1.00000    0.00000     7.0000
  _r3      |     2.0000    0.00000     4.0000
  _r4      |    0.00000   -1.00000   -1.00000

LBC2=
           |         1         2
------------------------------
  _x1      |   0.00000    2.0000
  _x2      |   0.00000    2.0000
  _x3      |   0.00000    2.0000
  _x4      |   0.00000    2.0000
```

3.4 The `mpswrite()` Function

< prob,row,col,rhs,rng,bnd > = mpswrite(strv,c<,lau<,lubc>>)

Purpose: The function `mpswrite()` writes a MPS file for a specified matrix model which can be used e.g. as input for the `pcx()` function for specifying a LP model. See Murtagh(1981), Nazareth (1987) or other standard literature on Linear Programming for more about the syntax of MPS files.

This function is useful for specifying the model for a LP optimization. The `lp()` and `pcx()` functions are trying to minimize or maximize the linear objective (cost) function $c^T x = \sum c_j x_j$ in n variables x_j subject to a set of m general linear inequality (or equality) constraints

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad , i = 1, \ldots, m$$

and/or simple boundary constraints $l_j \leq x_j \leq u_j$.

Note, similar code is being used for writing a temporary .mps file for interfacing CMAT's `lp()` function with DLL's of Coin, Clp, and Cbc.

Input: **strvec** must be either a string scalar `"fpath"` or a vector of strings specifying:

1. string specifying the path and name of the MPS file, should end with `.mps`
2. string specifying the problem name which defines the first line of the MPS file (maybe truncated to 15 chars), default is the specified file name without extension
3. string specifying the name of the objective function used in the ROWS and COLUMNS section of the file, default is "COST"
4. string specifying the name of the RHS section, default is "RHS",
5. string specifying the name of the RANGE section, default is "RANGE",
6. string specifying the name of the BOUNDS section, default is "BOUND".

c is the name of a n vector of real values specifying the linear objective function $c^T x$ which is to be minimized or maximized.

lau is the name of a $n_{lc} \times n + 2$ or $n_{lc} \times n + 1$ matrix with lower bounds (first column), coefficients, and upper bounds (last column) for n_{lc} general linear constraints. The specification of missing values for lower or upper bounds means that the corresponding side constraint is not imposed. If `lau` contains only $n+1$ columns, where n is defined by the dimension of the first argument `c`, then no upper bounds are imposed. This argument may be specified as a missing value if only boundary constraints are imposed with the last input argument.

lubc (optional) is the name of a $n \times 2$ matrix with lower (first column) and upper (second column) simple boundary constraints.

Output: If an error occurs this function only returns a single missing value. Otherwise it returns six vector or matrix arguments corresponding to each of the sections of the written MPS file.

prob section one: contaqining the string NAME and the specified problem name (if necessary truncated to 15 chars)

row section two: containing $n+1$ rows, each specifying the name of an x variable plus the specified name of the objective (by default=COST)

col section three: specifying the (sparse) matrix **A** of general linear constraints,

rhs section four: specifying the right hand sides `b` of the general linear constraints

rng section five: specifying the lower and upper ranges `b` of the general linear constraints

bnd section six: specifying lower and upper bounds for (some of) the variables.

Restrictions:
1. The specified input arguments `c` and `lau` may not contain missing values what does not mean that `lau` or `lubc` may be entirely missing.
2. The dimensions of `c`, `lau`, and `lubc` must be compatible.

Relationships: pcx(), lp()

Examples: The following examples use both, the `mpsread()` and the `mpswrite()` function.

1. 1. Example by Madsen and Pinar:
   - 1.1 Transform from matrix notation to MPS file:

```
print "*** 1. Example by Madsen and Pinar ***";
cx = cons(5,1,0.);
cx[3] = 6.;
lau = [ -8.  -6. -1. -6.  2.  0.  -8. ,
        -5.   0.  1. -6.  0.  2.  -5. ];
lubc = cons(5,2,0.);
lubc[,2] = 10.;
< prob,row,col,rhs,rng,bnd > = mpswrite("pinar1.mps",cx,lau,lubc);
```

The following is a listing of the output file `pinar1.mps`:

```
NAME          pinar1
ROWS
 E  _r1
 E  _r2
 N  COST
```

```
COLUMNS
    _x1       _r1       -6.
    _x2       _r1       -1.            _r2       1.
    _x3       _r1       -6.            _r2       -6.
    _x3       COST      6.
    _x4       _r1       2.
    _x5       _r2       2.
RHS
    RHS       _r1       -8.            _r2       -5.

RANGES
BOUNDS
 UP BOUND     _x1       10.
 UP BOUND     _x2       10.
 UP BOUND     _x3       10.
 UP BOUND     _x4       10.
 UP BOUND     _x5       10.
ENDATA
```

- 1.2 Transform from MPS file to matrix notation :

```
< c2,lau2,lbc2 > = mpsread("pinar1.mps");
print "C2=", c2;
print "Lau2=", lau2;
print "LBC2=", lbc2;
```

```
C2=
   |   _x1        _x2        _x3        _x4        _x5
-------------------------------------------------------------
 1 |    0.00000    0.00000     6.0000    0.00000    0.00000

Lau2=
           |   LowerRHS   _x1        _x2        _x3
-----------------------------------------------------------
  _r1      |    -8.0000    -6.0000   -1.00000    -6.0000
  _r2      |    -5.0000    0.00000    1.00000    -6.0000

           |   _x4        _x5        UpperRHS
-------------------------------------------
  _r1      |     2.0000    0.00000    -8.0000
  _r2      |    0.00000     2.0000    -5.0000

LBC2=
           |         1         2
------------------------------
```

```
_x1      |   0.00000   10.0000
_x2      |   0.00000   10.0000
_x3      |   0.00000   10.0000
_x4      |   0.00000   10.0000
_x5      |   0.00000   10.0000
```

2. 2. Example by Madsen and Pinar

- 2.1 Transform from matrix notation to MPS file:

```
print "*** 2. Example by Madsen and Pinar ***";
cx = cons(4,1,0.);
cx[1] = cx[4] = 1.;
lau = [   .   2.  2.  1.  0.   7. ,
          .   2.  1.  2.  0.   4. ,
          .   0. -1.  0. -1.  -1. ,
         3.   0.  1.  1.  1.   3. ];
lubc = cons(4,2,0.);
lubc[,2] = 2.;
< prob,row,col,rhs,rng,bnd > = mpswrite("pinar2.mps",cx,lau,lubc);
```

The following is a listing of the output file `pinar2.mps`:

```
NAME          pinar2
ROWS
 E  _r1
 L  _r2
 L  _r3
 L  _r4
 N  COST
COLUMNS
    _x1       _r2       2.             _r3       2.
    _x1       COST      1.
    _x2       _r1       1.             _r2       2.
    _x2       _r3       1.             _r4       -1.
    _x3       _r1       1.             _r2       1.
    _x3       _r3       2.
    _x4       _r1       1.             _r4       -1.
    _x4       COST      1.
RHS
    RHS       _r1       3.             _r2       7.
    RHS       _r3       4.             _r4       -1.

RANGES
BOUNDS
 UP BOUND     _x1       2.
 UP BOUND     _x2       2.
```

```
 UP BOUND     _x3       2.
 UP BOUND     _x4       2.
ENDATA
```

- 2.2 Transform from MPS file to matrix notation :

```
< c2,lau2,lbc2 > = mpsread("pinar2.mps");
print "C2=", c2;
print "Lau2=", lau2;
print "LBC2=", lbc2;
```

```
C2=
   |   _x1        _x2        _x3        _x4
-------------------------------------------------
 1 |    1.00000    0.00000    0.00000    1.00000
```

The order of the m general linear constraints is changed in such a way, that equality constraints are listed first and the inequality constraints last:

```
Lau2=
           |   LowerRHS   _x1        _x2
-----------------------------------------------
  _r1      |     3.0000    0.00000    1.00000
  _r2      |     .          2.0000     2.0000
  _r3      |     .          2.0000    1.00000
  _r4      |     .         0.00000   -1.00000

           |   _x3        _x4        UpperRHS
-----------------------------------------------
  _r1      |    1.00000    1.00000     3.0000
  _r2      |    1.00000    0.00000     7.0000
  _r3      |     2.0000    0.00000     4.0000
  _r4      |    0.00000   -1.00000   -1.00000

LBC2=
           |         1         2
--------------------------------
  _x1      |   0.00000    2.0000
  _x2      |   0.00000    2.0000
  _x3      |   0.00000    2.0000
  _x4      |   0.00000    2.0000
```

3.5 The `pritfile()` Function

s = pritfile("fpath"<,optn>)

Purpose: The function `pritfile()`

- prints a text file into the .lst file
- returns a vector of strings containing the text of each line.

Input: **"fpath"** string specifying the path and name of the input (text) file

optn a vector of integer values specifying runtime options:

1. a nonzero integer prevents the output of the file into the .lst file, default 0 will print the file
2. a nonzero integer will not return a vector of strings with the contents of the lines of the file, default 0 will return the vector of strings
3. an integer m restricting the output into the .lst file to the first m characters of each line, default 0 prints the full lines without length restriction

Output: There is only one return argument `s`:

- if optn[2] == 0: the return `s` is a string vector, each entry containing the text of a line of the input file,
- if optn[2] != 0: the return `s` is an integer giving the total length of the input file as number of characters.

In addition the function may output the content of the text file into the .lst file (if optn[1]=0).

Restrictions:
1. The path must point to a valid (text) file that can be opened for reading.
2. The content of the file should be common text.

Relationships: printf(), print()

Examples: 1. :

```
cx = cons(5,1,0.);
cx[3] = 6.;
lau = [ -8.  -6. -1. -6.  2.  0.  -8. ,
        -5.   0.  1. -6.  0.  2.  -5. ];
lubc = cons(5,2,0.);
lubc[,2] = 10.;
print "Transform matrix notation into MPS file";
< prob,row,col,rhs,rng,bnd > = mpswrite("pinar1.mps",cx,lau,lubc);
```

```
   print "[1] Print to list file and return string vector";
   optn = .;
   vecstr = pritfile("pinar1.mps",optn);
   print "VECSTR=", vecstr;
```

The following is the output into the .lst file:

```
Linewise Listing of File=pinar1.mps (Lenth=625)
************************************************
NAME          pinar1
ROWS
 E  _r1
 E  _r2
 N  COST
COLUMNS
    _x1       _r1       -6.
    _x2       _r1       -1.            _r2       1.
    _x3       _r1       -6.            _r2       -6.
    _x3       COST      6.
    _x4       _r1       2.
    _x5       _r2       2.
RHS
    RHS       _r1       -8.            _r2       -5.
RANGES
BOUNDS
 UP BOUND     _x1       10.
 UP BOUND     _x2       10.
 UP BOUND     _x3       10.
 UP BOUND     _x4       10.
 UP BOUND     _x5       10.
ENDATA
************************************************
```

```
   print "[2] Return string vector";
   optn = 1;
   vecstr = pritfile("pinar1.mps",optn);

   print "[2] Return file length ";
   optn = [ 1 1 ];
   flength = pritfile("pinar1.mps",optn);
   print "Flength=",flength;
```

```
Flength= 625
```

3.6 The `xctbinom()` Function

< est,ci > = xctbinom(x,n,p<,optn>)

Purpose: The function `xctbinom()` computes p values and confidence intervals for testing the null hypothesis of the probability of success in a Bernoulli experiment. The algorithm is very similar to that of function `binom.exact` in CRAN package `exactci` by M. P. Fay.

Input: **x** The first input argument is either a scalar or m vector of real values with the number of successes.

n The second input argument is is either a scalar or m vector of real values with the number of trials.

p The third input argument is is either a scalar or m vector of real values of the hypothesized probability of success.

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
`"alpha"`	real	the significance level for confidence intervals; default is $\alpha = 0.05$
`"alt"`	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
`"maxfun"`	int	maximum number of function calls for zero finder, default=10000,
`"maxref"`	int	maximum number of grid refinements, default=100,
`"meth"`	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
`"midp"`		for midpoint confidence limits
`"print"`	int	for the amount of printed output, default is 0, that means no printed output
`"prngl"`	real	lower range for probability, default=10^{-10}
`"prngu"`	real	upper range for probability, default=$1 - 10^{-10}$
`"relerr"`	real	for testing relative error, default is $1 + 10^{-7}$
`"tol"`	real	tolerance for zero finding, default 10^{-5}

Output: **est** The first output argument is a $m \times 2$ matrix containing the p values in its first column and the estimated probabililty of success in its second column.

ci The second output argument is a $m \times 2$ matrix containing the lower confidence limits in its first column and the upper confidence limits in its second column.

Restrictions: 1. The first three input arguments should be compatible in its size, i.e. should be either scalars or vectors of the same size m.

2. The first three input arguments should have not any missing or string values.

Relationships: xctmcnem(), xctbipow(), xctbissz(), xctpoiss(), xctfishr()

Examples: 1. Simple example from CRAN package `exactci`:

```
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "minl" ];
p = .05;
< pval, ci > = xctbinom(4,20,p,optn);


         Exact two-sided Binomial test (MinLike Method)
              Testing True Probability of Success

            Probability= 0.0159 Prob. of Success=0.2
     Min. Likelihood Method 95 Percent CI: [0.0714,0.4236]


optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "blak" ];
p = .05;
< pval, ci > = xctbinom(4,20,p,optn);


         Exact two-sided Binomial test (Blaker Method)
              Testing True Probability of Success

            Probability= 0.0159 Prob. of Success=0.2
     Blaker 95 Percent Confidence Interval: [0.0714,0.4219]


optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "cent" ];
p = .05;
```

```
< pval, ci > = xctbinom(4,20,p,optn);


        Exact two-sided Binomial test (Central Method)
             Testing True Probability of Success

          Probability= 0.0318 Prob. of Success=0.2
  Central 95 Percent Confidence Interval: [0.057334,0.436614]
```

3.7 The `xctbip1()` Function

p1 = xctbip1(powr,ssiz,p0<,optn>)

Purpose: The function `xctbip1()` computes $p1$, the probability of success under the alternative for given power, sample size, and $p0$. The three functions `xctbipow()`, `xctbissz()`, and `xctbip1()` all deal with the parameters `powr`, `ssiz`, `p0`, and `p1`, where three of them must be specified input and one is computed,

xctbipow input: `ssiz`, `p0`, `p1`, result: `powr`

xctbissz input: `powr`, `p0`, `p1`, result: `ssiz`

xctbip1 input: `powr`, `ssiz`, `p0`, result: `p1`

The implementation is based on similar algorithms as used by M. Fay in function `powerBinom.R` of the package `exactci`.

Input: The first three arguments must be real scalars or m vectors of real values.

powr the power must be inside $(0, 1)$

ssiz the sample size must be positive integer

p0 the probability of the null hypothesis (default=.5)

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"alt"	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
"maxfun"	int	maximum number of function calls for zero finder, default=10000,
"maxref"	int	maximum number of grid refinements, default=100,
"meth"	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
"midp"		for midpoint confidence limits
"print"	int	for the amount of printed output, default is 0, that means no printed output
"prngl"	real	lower range for probability, default=10^{-10}
"prngu"	real	upper range for probability, default=$1 - 10^{-10}$
"relerr"	real	for testing relative error, default is $1 + 10^{-7}$
"strict"		strict interpretation of the two-sided alternative, by default two-sided is changed to one-sided for count of rejections
"tol"	real	tolerance for zero finding, default 10^{-5}

Output: The only result is an $m \times 4$ matrix containing the three input arguments *powr*, *ssiz*, $p0$ and the computed value of $p1$, the probability of success under the alternative, in its columns.

Restrictions: 1. The first three input arguments should be compatible in its size, i.e. should be either scalars or vectors of the same size m.

2. The first three input arguments should have not any missing or string values.

Relationships: xctbinom(), xctbipow(), xctbissz()

Examples: 1. Example from CRAN package `exactci`:

```
powr = .8; ssiz = 90; p0 = .5;
optn = [ "print"         2 ,
         "alpha"      0.05 ,
         "alt"      "less" ];
p1 = xctbip1(powr,ssiz,p0,optn);
print "P_success = P(alternative)=", p1;
```

```
Probability of success for power and sample size (alpha= 0.0500)
```

```
Power=0.8 SampleSize=90 p0=0.5 p1=0.363038
```

3.8 The `xctbipow()` Function

powr = xctbipow(ssiz,p0,p1<,optn>)

Purpose: The function `xctbipow()` computes the power for given values of sample size, $p0$, and $p1$. The three functions `xctbipow()`, `xctbissz()`, and `xctbip1()` all deal with the parameters `powr`, `ssiz`, `p0`, and `p1`, where three of them must be specified input and one is computed,

xctbipow input: `ssiz`, `p0`, `p1`, result: `powr`

xctbissz input: `powr`, `p0`, `p1`, result: `ssiz`

xctbip1 input: `powr`, `ssiz`, `p0`, result: `p1`

The implementation is based on similar algorithms as used by M. Fay in function `powerBinom.R` of the package `exactci`.

Input: The first three arguments must be real scalars or m vectors of real values.

ssiz the sample size must be positive integer

p0 the probability of the null hypothesis (default=.5)

p1 the probability of success under the alternative

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"alt"	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
"maxfun"	int	maximum number of function calls for zero finder, default=10000,
"maxref"	int	maximum number of grid refinements, default=100,
"meth"	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
"midp"		for midpoint confidence limits
"print"	int	for the amount of printed output, default is 0, that means no printed output
"prngl"	real	lower range for probability, default=10^{-10}
"prngu"	real	upper range for probability, default=$1 - 10^{-10}$
"relerr"	real	for testing relative error, default is $1 + 10^{-7}$
"strict"		strict interpretation of the two-sided alternative, by default two-sided is changed to one-sided for count of rejections
"tol"	real	tolerance for zero finding, default 10^{-5}

Output: The only result is an $m \times 4$ matrix containing the three input arguments *ssiz*, $p0$, $p1$, and the computed value of *powr*, the power, which is areal value in $(0, 1)$, in its columns.

Restrictions:
1. The first three input arguments should be compatible in its size, i.e. should be either scalars or vectors of the same size m.
2. The first three input arguments should have not any missing or string values.

Relationships: xctbinom(), xctbissz(), xctbip1()

Examples: 1. Example from CRAN package `exactci`:

```
nsiz = 90; p0 = .5; p1 = .7;
optn = [ "print"          2 ,
         "type"     "stand" ,
         "alpha"       0.05 ,
         "alt"       "twos" ];
powr = xctbipow(nsiz,p0,p1,optn);
print "Power=", powr;
```

```
Power for single Binomial Response (alpha= 0.0250)
    (Use rejections in correct direction only)

     SampleSize=90 p0=0.5 p1=0.7 Power=0.972555
```

3.9 The `xctbissz()` Function

ssiz = xctbissz(powr,p0,p1<,optn>)

Purpose: The function `xctbissz()` computes the sample size for given values of power, $p0$, and $p1$. The three functions `xctbipow()`, `xctbissz()`, and `xctbip1()` all deal with the parameters `powr`, `ssiz`, `p0`, and `p1`, where three of them must be specified input and one is computed,

xctbipow input: `ssiz`, `p0`, `p1`, result: `powr`

xctbissz input: `powr`, `p0`, `p1`, result: `ssiz`

xctbip1 input: `powr`, `ssiz`, `p0`, result: `p1`

The implementation is based on similar algorithms as used by M. Fay in function `powerBinom.R` of the package `exactci`.

Input: The first three arguments must be real scalars or m vectors of real values.

powr the power must be inside $(0, 1)$

p0 the probability of the null hypothesis (default=.5)

p1 the probability of success under the alternative

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"alt"	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
"maxfun"	int	maximum number of function calls for zero finder, default=10000,
"maxref"	int	maximum number of grid refinements, default=100,
"meth"	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
"midp"		for midpoint confidence limits
"print"	int	for the amount of printed output, default is 0, that means no printed output
"prngl"	real	lower range for probability, default=10^{-10}
"prngu"	real	upper range for probability, default=$1 - 10^{-10}$
"relerr"	real	for testing relative error, default is $1 + 10^{-7}$
"strict"		strict interpretation of the two-sided alternative, by default two-sided is changed to one-sided for count of rejections
"tol"	real	tolerance for zero finding, default 10^{-5}
"type"	string	type of estimation: "stand" for standard estimation, "cilen" computing sample size based on specified CI length input (and not power), "obs1or" for computing sample size related to probability to observe at least one success

Output: The only result is an $m \times 4$ matrix containing the three input arguments *powr*, $p0$, $p1$, and the computed value of *ssiz*, the integer value of the sample size, in its columns. Since the value of the sample size is an integer, the output of the value for the power may slightly differ from its input value (which corresponds to a real value of the sample size).

Restrictions:
1. The first three input arguments should be compatible in its size, i.e. should be either scalars or vectors of the same size m.
2. The first three input arguments should have not any missing or string values.

Relationships: xctbinom(), xctbipow(), xctbip1()

Examples: 1. Example from CRAN package `exactci`: Type = `"stand"`

```
powr = .8; p0 = .7; p1 = .5;
alt = (p1 > p0) ? "grea" : "less";
```

```
optn = [ "print"         2 ,
         "type"    "stand" ,
         "alpha"     0.025 ,
         "alt"         alt ];
ssiz = xctbissz(powr,p0,p1,optn);
print "SampleSize=", ssiz;
```

Since the sample size is an integer, the corresponding power is slightly modified from the input:

```
    Sample size for single Binomial Response (alpha= 0.0250)
           (Use rejections in correct direction only)

            Power=0.809153 p0=0.7 p1=0.5 SampleSize=47
```

2. Example from CRAN package `exactci`: Type = `"cilen"`

```
cilen = .3; p0 = p1 = .5;
optn = [ "print"          2 ,
         "type"     "cilen" ,
         "alpha"      0.05 ,
         "alt"       "twos" ];
ssiz = xctbissz(cilen,p0,p1,optn);
print "SampleSize=", ssiz;
```

```
Compute minimum sample size for specified CI length (alpha= 0.0500)
              (sample size is maximized when p1=0.5)

             CI_length=0.3 p0=0.5 p1=0.5 SampleSize=45
```

3. Example from CRAN package `exactci`: Type = `"obs1or"`

```
powr = .9; p0 = .; p1 = .01;
optn = [ "print"        2 ,
         "type"  "obs1or" ,
         "alpha"     0.05 ];
ssiz = xctbissz(powr,p0,p1,optn);
print "SampleSize=", ssiz;
```

```
  Sample size related to the probability to observe at least
                one success (alpha= 0.0500)

           Power=0.9 p0=0.5 p1=0.01 SampleSize=230
```

```
SampleSize=
 R |        Power     ProbNull      ProbAlt   SampleSize
-------------------------------------------------------
 1 |      0.90000      0.50000        0.010       230.00
```

3.10 The `xctfipow()` Function

pow = xctfipow(x,r<,optn>)

Purpose: The function `xctfipow()` computes the power of Fisher's or McNemar's exact test for specified sample sizes.

Input: **x** is a vector of four nonnegative real values:

- $p0$ true event rate in control group
- $p1$ true event rate in treatment group
- $n0$ sample size in control group
- $n1$ sample size in treatment group (def=$n0$)

or The second input argument is either a scalar or m vector of real values of the hypothesized odds ratio. Default=1.

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"alt"	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
"apprx"		use fast but approximate method by Fleiss (1981)
"maxfun"	int	maximum number of function calls for zero finder, default=10000,
"maxref"	int	maximum number of grid refinements, default=100,
"mcnem"		use McNemar's test instead of Fisher's test (note, that there must be also $n0 = n1$),
"meth"	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
"midp"		for midpoint confidence limits
"ngrid"	int	start number of grid points, default=100,
"orngl"	real	lower range for odds ratio, default=10^{-10}
"orngu"	real	upper range for odds ratio, default=10^{+10}
"print"	int	for the amount of printed output, default is 0, that means no printed output
"relerr"	real	for testing relative error, default is $1 + 10^{-7}$
"tol"	real	tolerance for zero finding, default 10^{-5}

Output: The only return argument is a real scalar or an m vector of real values for the power of Fisher's or McNemar's test.

Restrictions: 1. The values of $p0$ and $p1$ must be in $[0, 1]$, and $n0$ and $n1$ must be both positive.

Relationships: xctfishr(), xctmcnem(), xctfissz()

Examples: 1. Example of R code in package `exact2x2`:

```
print "Example from power2x2 code in R";
tab = [ .3 .8 12 . ];
cnam = [" p0 p1 n0 n1 "];


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "minl" ];
powr = xctfipow(tab,oddr,optn);
print "Power=", powr;
```

```
Power= 0.6098
```

3.11 The `xctfishr()` Function

< est,ci > = xctfishr(x,or<,optn>)

Purpose: The function `xctfishr()` can be used for testing the independence of rows and columns in a 2×2 contingency table with fixed rowe and column sums. The algorithm is very similar to that of function `exact2x2` in CRAN package `exact2x2` by M. P. Fay.

Input: **x** The first input argument must be a 2×2 matrix of count data (contingency table).

or The second input argument is either a scalar or m vector of real values of the hypothesized odds ratio.

optn The fourth input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
`"alpha"`	real	the significance level for confidence intervals; default is $\alpha = 0.05$
`"alt"`	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
`"maxfun"`	int	maximum number of function calls for zero finder, default=10000,
`"maxref"`	int	maximum number of grid refinements, default=100,
`"meth"`	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
`"midp"`		for midpoint confidence limits
`"ngrid"`	int	start number of grid points, default=100,
`"orngl"`	real	lower range for odds ratio, default=10^{-10}
`"orngu"`	real	upper range for odds ratio, default=10^{+10}
`"print"`	int	for the amount of printed output, default is 0, that means no printed output
`"relerr"`	real	for testing relative error, default is $1 + 10^{-7}$
`"tol"`	real	tolerance for zero finding, default 10^{-5}

Output: **est** The first output argument is a $m \times 2$ matrix containing the p values in its first column and the estimated odds ratio(s) in its second

column.

ci The second output argument is a $m \times 2$ matrix containing the lower confidence limits in its first column and the upper confidence limits in its second column.

Restrictions: 1. The first input argument must contain integer values.

2. The first two input arguments should have not any missing or string values.

Relationships: xctmcnem(), xctbinom(), xctpoiss(), xctfipow()

Examples: 1. The classic Fisher's Tea Drinker Example, see documentation of `fisher.test` in CRAN: A British woman claimed to be able to distinguish whether milk or tea was added to the cup first. To test, she was given 8 cups of tea, in four of which milk was added first. The null hypothesis is that there is no association between the true order of pouring and the woman's guess, the alternative that there is a positive association (that the odds ratio is greater than 1). See Agresti (2002, p. 91).

```
print "Fisher\'s Tea Drinker: with alternative=greater";
print " => p = 0.2429, association could not be established";
tab = [ 3 1 , 1 3 ];
rnam = [ "Guess\_Milk", "Guess\_Tea" ];
cnam = [ "Truth\_Milk", "Truth\_Tea" ];
tab = cname(tab,cnam); tab = rname(tab,rnam);
print "Tab=", tab;
```

```
Tab=
         SYM |   Truth_Milk    Truth_Tea
-----------------------------------------
  Guess_Milk |            3
   Guess_Tea |            1            3
```

```
oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "grea" ,
         "meth"   "minl" ];
< pval, ci > = xctfishr(tab,oddr,optn);
```

```
          Exact one-sided Fisher's test (MinLike Method)
                    Testing True Odds Ratio

              Probability= 0.2429 Odds Ratio=6.40832
```

```
Min. Likelihood Method 95 Percent CI: [0.313574,1.79769e+308]
```

2. Example in CRAN package `exact2x2`:

```
oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "minl" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
< pval, ci > = xctfishr(x,oddr,optn);


         Exact two-sided Fisher's test (MinLike Method)
                   Testing True Odds Ratio

             Probability= 0.1007 Odds Ratio=1.62024
     Min. Likelihood Method 95 Percent CI: [0.8962,2.9237]


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "cent" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
< pval, ci > = xctfishr(x,oddr,optn);


         Exact two-sided Fisher's test (Central Method)
                   Testing True Odds Ratio

             Probability= 0.1373 Odds Ratio=1.62024
   Central 95 Percent Confidence Interval: [0.860585,2.91139]


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "blak" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
< pval, ci > = xctfishr(x,oddr,optn);
```

```
Exact two-sided Fisher's test (Blaker Method)
Testing True Odds Ratio

Probability= 0.1007 Odds Ratio=1.62024
Blaker 95 Percent Confidence Interval: [0.8805,2.91]
```

3.12 The `xctfissz()` Function

pow = xctfissz(x,r<,optn>)

Purpose: The function `xctfissz()` computes the necessary sample size for specified power of the *exact* Fisher test or the (paired) McNemar test. My implementation is based on an algorithm by Fay (2014) in R package `exact2x2`.

Input: **x** is a vector of four nonnegative real values:

- $p0$ true event rate in control group
- $p1$ true event rate in treatment group
- *power* required power, def=0.8
- $n1\_over\_n0$ ratio of sample sizes (treatment over control group), def=1.

or The second input argument is either a scalar or m vector of real values of the hypothesized odds ratio. Default=1.

optn The third input argument specifies a number of options in form of one 2-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.

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"apprx"		use fast but approximate method by Fleiss (1981)
"alt"	string	specifying the test alternative: "twos" for two-sided, "less", and "grea" for one-sided tests, default is two-sided
"maxfun"	int	maximum number of function calls for zero finder, default=10000,
"maxref"	int	maximum number of grid refinements, default=100,
"mcnem"		use McNemar's test instead of Fisher's test (note, that there must be also $n1\_over\_n0 = 1$),
"meth"	string	specifying the method for computing confidence limits: "minl" for minimum-likelihood, "blak" for Blaker's, "cent" for the central method, default is minimum-likelihood
"midp"		for midpoint confidence limits
"ngrid"	int	start number of grid points, default=100,
"orngl"	real	lower range for odds ratio, default=10^{-10}
"orngu"	real	upper range for odds ratio, default=10^{+10}
"print"	int	for the amount of printed output, default is 0, that means no printed output
"relerr"	real	for testing relative error, default is $1 + 10^{-7}$
"tol"	real	tolerance for zero finding, default 10^{-5}

Output: The only return argument is an $m \times 2$ vector of real values for $n0$ in column 1 and $n1$ in column 2 of Fisher's or McNemar's test.

Restrictions: 1. The values of $p0$, $p1$, $n1\_over\_n0$ must be in $[0, 1]$.

Relationships: xctfishr(), xctmcnem(), xctfipow()

Examples: 1. Example of R code in package `exact2x2`:

```
tab = [ .5 .99 .8 . ];
cnam = [" p0 p1 powr n1ovrn0 "];


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "minl" ];
size = xctfissz(tab,oddr,optn);
print "SampleSize=", size;
```

```
SampleSize=
 R |         1         2
-------------------------
 1 |    13.000    13.000
```

3.13 The `xcthybr()` Function

pval = xcthybr(tab<,optn>)

Purpose: The function `xcthybr()` computes the p values of the exact Fisher test using the hybrid method of Mehta & Patel (1986), e.g. subroutine FEXACT, algorithm 643 from ACM TOMS. Note, that this method is VERY memory consuming. In addition, the common p value based on asymptotic chi^2 distribution may be computed.

Input: **tab** should be an $m \times n$ table of integer values. It the values are real they are rounded to the next integer.

optn is a numeric vector specifying some runtime options. The following is mainly from the header of subroutine FEXACT:

[**1**] amount of printed output, default=0

[**2**] amount of workspace allocated, default=200,000 byte For many problems one megabyte or more of workspace can be required. If the environment supports it, the user should begin by increasing the workspace used to 200,000 units.

[**3**] if not zero the p value of χ^2 is simulated too and in addition the asymptotic p is computed, default=0

[**4**] if not zero and input table is $n = m = 2$, Yates correction is applied to asymptotic p value, default=0

[**5**] argument `expect` in FEXACT, default=5. : expected value used in the hybrid algorithm for deciding when to use asymptotic theory probabilities. If EXPECT ¡= 0.0 then asymptotic theory probabilities are not used and Fisher exact test probabilities are computed. Otherwise, if PERCNT or more of the cells in the remaining table have estimated expected values of EXPECT or more, with no remaining cell having expected value less than EMIN, then asymptotic chi-squared probabilities are used. See the algorithm section of the manual document for details.

[**6**] argument `percnt` in FEXACT, default=80. : PERCNT - Percentage of remaining cells that must have estimated expected values greater than EXPECT before asymptotic probabilities can be used.(Input) See argument EXPECT for more details.

[**7**] argument `emin` in FEXACT, default=1. : EMIN - Minimum cell estimated expected value allowed for asymptotic chi-squared

probabilities to be used.(Input) See argument EXPECT for more details.

[**8**] argument `mult` in FEXACT, default=30.

Note, the default options for (opt[5,...,8]) are those of the Cochran choice: $(exp, perc, emin, mult) = (5, 80, 1, 30)$. To obtain the exact Fisher probabilities use: $(exp, perc, emin, mult) = (0, 0, 0, 30)$.

Output: The only output argument `pval` is either a scalar (for optn[3]=0) or a numeric vector (for optn[3]=1) containing the p values.

Restrictions: 1. The input argument `tab` may not have missing values or string data.

2. Yates correction of the common asymptotic p value is valid only for 2×2 tables.

Relationships: xctfishr(), conting(), xctsimu()

Examples: 1. Fisher (1962, 1970): Criminal convictions of like-sex twins:

```
tab = [ 2  15 , 10  3 ];
rnam = [ "Dizygotic" "Monozygotic" ];
cnam = [ "Convicted", "Not convicted" ];
tab = cname(tab,cnam); tab = rname(tab,rnam);
print "Tab=", tab;
```

```
Tab=
              |       Convicted   Not convicted
-----------------------------------------------
    Dizygotic |               2              15
  Monozygotic |              10               3
```

```
/* cochran setup */
opt = [      2 ,   /* ipri */
        300000 ,   /* nwsp */
             1 ,   /* chi too */
             1 ,   /* yates */
             5.,   /* exp */
            80.,   /* percent */
             1.,   /* emin */
            30 ];  /* mult */
pval = xcthybr(tab,opt);
```

```
    Probability of simulated exact Fisher test: 0.000536724
   Yates corr. Chisquare=10.4581 df=1 Probability=0.0012211
```

```
Pval=
 R |   P(Fisher)   P(Pextrm)   P(AsyChi)
---------------------------------------
 1 |     0.00045     0.00054     0.00122
```

```
  /* exact setup */
  opt = [      2 ,   /* ipri */
          300000 ,   /* nwsp */
               1 ,   /* chi too */
               1 ,   /* yates */
              -1.,   /* exp */
               0. ,  /* percent */
               0.,   /* emin */
              30 ];  /* mult */
  pval = xcthybr(tab,opt);
```

```
      Probability of simulated exact Fisher test: 0.000536724
     Yates corr. Chisquare=10.4581 df=1 Probability=0.0012211

Pval=
 R |   P(Fisher)   P(Pextrm)   P(AsyChi)
---------------------------------------
 1 |     0.00045     0.00054     0.00122
```

2. Agresti (2002), p.57: Job Satisfaction:

```
  tab = [ 1  3  10   6 ,
          2  3  10   7 ,
          1  6  14  12 ,
          0  1   9  11 ];
  rnam = [ "< 15k", "15-25k", "25-40k", "> 40k" ];
  cnam = [ "VeryD", "LittleD", "ModerateS", "VeryS" ];
  tab = cname(tab,cnam); tab = rname(tab,rnam);
  print "Tab=", tab;
```

```
Tab=
         |       VeryD     LittleD   ModerateS       VeryS
-------------------------------------------------------------
   < 15k |           1           3          10           6
  15-25k |           2           3          10           7
  25-40k |           1           6          14          12
   > 40k |           0           1           9          11
```

```
  /* cochran setup */
  opt = [      2 ,   /* ipri */
          300000 ,   /* nwsp */
               1 ,   /* chi too */
               0 ,   /* yates */
               5.,   /* exp */
              80.,   /* percent */
               1.,   /* emin */
              30 ];  /* mult */
  pval = xcthybr(tab,opt);


     Probability of simulated exact Fisher test: 0.782683
           Chisquare=5.96551 df=9 Probability=0.743365

Pval=
 R |   P(Fisher)   P(Pextrm)   P(AsyChi)
---------------------------------------
 1 |     0.00000     0.78268     0.74336


  /* exact setup */
  opt = [      2 ,   /* ipri */
          300000 ,   /* nwsp */
               1 ,   /* chi too */
               0 ,   /* yates */
              -1.,   /* exp */
               0. ,  /* percent */
               0.,   /* emin */
              30 ];  /* mult */
  pval = xcthybr(tab,opt);


     Probability of simulated exact Fisher test: 0.782685
           Chisquare=5.96551 df=9 Probability=0.743365

Pval=
 R |   P(Fisher)   P(Pextrm)   P(AsyChi)
---------------------------------------
 1 |     0.00000     0.78268     0.74336
```

3.14 The `xctlog()` Function

< gof,est,conf > = xctlog(data,model,exct<,optn<class>>)

Purpose: The function `xctlog()` implements an MCMC algorithm developed by Zamar et al. (2013) for the solution of the exact logistic model. The results of the R version and this differ since the seed for the random generator is set by the actual computer time. Here the seed maybe specified guaranteeing the same result at different runs. Another difference between the two implementations is the option `"minsmp"` which can be used to specify the minimum number of valid samples used for the inference computations. In `elrm` that is always set to 1000, whereas here that is only the default, but it maybe specified to larger values. The algorithm will always use that value as a lower bound, even when the specified number of iterations would be too small. That makes the update function available for `elrm` here not necessary. Also, the input data here must not necessarily be in the form of *events / trial* response specification. If the column number of the trial variable is not specified, the input response y must be $(0, 1)$ binary. As in `elrm`, the intercept is always part of the model, which means, there is no `"noint"` model specification.

Input: **data** must be an m *timesn* data matrix with columns referred to by the model specification.

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

exct is either an integer scalar ($nz = 1$) or a nz vector of integers specifying the numbers of effects in the model specification which are of interest for inference.

optn : The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. Some of the options are similar to other functions. See table below for content.

class : This optional argument should be an integer scalar or vector of integer scalars naming the number of columns which are considered categorical (nominal scaled) variables.

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

Option Name	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"burnin"	int	number of burnin calls of the random number generator, def=0
"freq"	int	column number of frequency variable
"iter"	int	proposed number of samples generated, def=1000
"minsmp"	int	minimum number of valid samples for inference, def=1000
"print"	int	amount of printed output (=0: no printed output, =2: default)
"pall"		is equivalent to "print" 10
"noprint"		there will be no printed output
"rmc"	int	MCMC parameter $R \leq Nobs$, must be even, default=4
"rand"	int	random generator: =0: default, =1: MS VisualC/C++, = CRAN
"seed"	int	seed for random generator
"trial"	int	column number of trial variable
"weight"	int	column number of weight variable

Output: **gof** : vector of goodness of fit indices

est : 5-vector or $nz + 1 \times 5$ matrix of parameter estimates, p values, asymptotic standard errors of p values, number of valid samples, and number of unique patterns of the sampled y

conf : $nz \times 2$ matrix of lower and upper confidence levels

Restrictions: 1. The data set and the model formula must be conform, and the integers in "exct" must refer to the numbers of effects as they appear in the model formula.

2. If the column number of the trial variable is not specified, the input response y must be $(0, 1)$ binary.

Relationships: xctfishr(), glim()

Examples: 1. Example taken from R Manual:

```
print "Example from {\tt elrm} document on Internet";
data = [ 1   8   0   0  ,
         1   6   0   1  ,
         7  10   1   0  ,
         6   6   1   1  ] ;
cnam = [" admit trials female apcalc "];
data = cname(data,cnam);


model = "1/2 = 4";
exct = 1;   /* points to the effect number in model statement */
optn = [ "print"          5 ,
         "rmc"            4 ,
         "burnin"         0 ,
         "iter"        1000 ,
```

```
         "alpha"         .05 ,
         "trial"           2 ];
< gof,parm,conf > = xctlog(data,model,exct,optn);


   Effect=1: Nsamples=1000 Nunique=10 Index=1 CritFrq=0.218
 Effect=1: BatchSize=44 NB=22 Mean=0.1403 Variance=0.000773194


       ************************************************
       Exact Logistic Regression (D. Zumar, 2013, R_MC=4)
       ************************************************

        Effect    Estimate P-value P_AsyStdErr Nsample Nunique
        apcalc  0.50650995  0.4880  0.00592834    1000      10


        Confidence Intervals for Estimates (alpha=0.05)
                   Effect        Lower        Upper
                   apcalc -1.075177571  2.472632132
```

2. Example which comes with `elrm` package:

```
print "Drug Data comes with elrm package";
data = [ 16  27  1  1 ,
         10  19  0  1 ,
         13  32  1  0 ,
          7  21  0  0 ];
cnam = [" recover nt sex treat "];
data = cname(data,cnam);


model = "1/2 = 3 4";
exct = [ 1 2 ];   /* points to the effect number in model statement */
optn = [ "print"           5 ,
         "rmc"             4 ,
         "burnin"       1000 ,
         "iter"        50000 ,
         "alpha"         .05 ,
         "trial"           2 ];
< gof,parm,conf > = xctlog(data,model,exct,optn);


    Effect=1: Nsamples=1535 Nunique=16 Index=1 CritFrq=0.12899
Effect=1: BatchSize=101 NB=15 Mean=0.0939061 Variance=3.12062e-005
   Effect=2: Nsamples=6712 Nunique=19 Index=1 CritFrq=0.0294994
Effect=2: BatchSize=519 NB=12 Mean=0.0190532 Variance=1.45524e-006
```

```
    Joint: Nsamples=50000 Nunique=283 Index=98 CritFrq=0.00394
JointASE: BatchSize=3099 NB=16 Mean=0.00207198 Variance=5.1679e-009


          ************************************************
          Exact Logistic Regression (D. Zumar, 2013, R_MC=4)
          ************************************************

           Effect    Estimate P-value P_AsyStdErr Nsample Nunique
              sex  0.28706262  0.6638  0.00144236    1535      16
            treat  0.84700861  0.0730  3.482e-004    6712      19
            Joint       .      0.1473  1.797e-005   50000     283


           Confidence Intervals for Estimates (alpha=0.05)
                     Effect        Lower        Upper
                        sex -0.619734185  1.155103745
                      treat -0.128576371  2.030446480
```

3. Large example from publication in JSS (Zamar, et al., 2007):

```
  print "Simulated Diabetes Dataset: see peper in JSS";
  options NOECHO;
#include "..\\tdata\\diabet.dat"
  options ECHO;

  cnam = ["  n IA2A gender age nDQ2 nDQ8 nDQ62 "];
  diabet = cname(diabet,cnam);


  model = "2/1 = 3 4 5 6 7 4*5 4*6 4*7";
  exct = [ 5 8 ];    /* point to the effect number in model statement */
  optn = [ "print"           5 ,
           "rand"            1 ,
           "noimp"             ,
           "rmc"             4 ,
           "burnin"        500 ,
           "iter"       100000 ,
           "alpha"         .05 ,
           "trial"           1 ];
  < gof,parm,conf > = xctlog(diabet,model,exct,optn);
  print "GOF=", gof;
  print "Parm=", parm;
  print "CI=", conf;


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

```
Events/Trials Response Variable
*******************************

 Value         Nobs  Proportion
 Event          288   43.049327
 NonEvent       381   56.950673


  Progress: 5 % N Samples: 5000 Time: 9 Seconds
 Progress: 10 % N Samples: 10000 Time: 19 Seconds
 Progress: 15 % N Samples: 15000 Time: 28 Seconds
 Progress: 20 % N Samples: 20000 Time: 37 Seconds
 Progress: 25 % N Samples: 25000 Time: 46 Seconds
 Progress: 30 % N Samples: 30000 Time: 56 Seconds
 Progress: 35 % N Samples: 35000 Time: 65 Seconds
 Progress: 40 % N Samples: 40000 Time: 74 Seconds
 Progress: 45 % N Samples: 45000 Time: 83 Seconds
 Progress: 50 % N Samples: 50000 Time: 93 Seconds
 Progress: 55 % N Samples: 55000 Time: 102 Seconds
 Progress: 60 % N Samples: 60000 Time: 111 Seconds
 Progress: 65 % N Samples: 65000 Time: 121 Seconds
 Progress: 70 % N Samples: 70000 Time: 130 Seconds
 Progress: 75 % N Samples: 75000 Time: 139 Seconds
 Progress: 80 % N Samples: 80000 Time: 148 Seconds
 Progress: 85 % N Samples: 85000 Time: 158 Seconds
 Progress: 90 % N Samples: 90000 Time: 167 Seconds
 Progress: 95 % N Samples: 95000 Time: 176 Seconds
Progress: 100 % N Samples: 100000 Time: 186 Seconds


  Effect=5: Nsamples=8942 Nunique=1 Index=1 CritFrq=1
  Effect=8: Nsamples=8942 Nunique=1 Index=1 CritFrq=1
Joint: Nsamples=100000 Nunique=57 Index=2 CritFrq=0.08941
JointASE: BatchSize=7961 NB=12 Mean=0.0368405 Variance=4.17978e-005
```

It occurs quite frequently that the parameterwise inference computations are not feasible, especially if there is more than one effect defined for the "exact" analysis. However, the global result by CMAT is very similar of that shown at the publication:

```
************************************************
Exact Logistic Regression (D. Zamar, 2013, R_MC=4)
************************************************

 Effect    Estimate P-value P_AsyStdErr Nsample Nunique
  nDQ62       .        .          .            0       0
```

```
age*nDQ62       .         .           .            0       0
     Joint      .       0.7672  0.00186632  100000      57

    Confidence Intervals for Estimates (alpha=0.05)
              Effect        Lower        Upper
               nDQ62      .            .
          age*nDQ62       .            .
```

And the total computer time is much less than that of the R code which is reported as more than one hour:

```
GOF=
           COL |          1
---------------------------
   1 RetCode   |   0.000000
   2 C_Time    |    186.000
   3 N_obs     |    229.000
   4 N_eff     |    9.00000
   5 N_exct    |    2.00000
   6 R_MCMC    |    4.00000
   7 InNiter   |   100000.0
   8 MinSamp   |   1000.000
   9 Burnin    |    500.000
  10 unused    |         .
```

3.15 The `xctmcnem()` Function

< est,ci > = xctmcnem(x,or<,optn>)

Purpose: The function `xctmcnem()` tests the difference of the two off-diagonal entries in a 2×2 contingency table when there is some *pairing* of the data, e.g. when the data are binary responses of two groups A and B (control and treatment) or pre- and posttest responses. The method is basically applying binomial testing using function `xctbinom()` with one of the offdiagonal entries (number of successes x) and the sum of the two offdiagonal entries (number of trials n). The resulting p values and confidence limits are then converted to odds ratios. The method does not depend on the values of the diagonal entries of the 2×2 contingency table. The algorithm is very similar to that of function `exact2x2` in CRAN package `exact2x2` by M. P. Fay.

Input: x The first input argument must be a 2×2 matrix of count data (contingency table). The result only depends on the two off-diagonal entries and does not depend on the values of the diagonal entries.

or The second input argument is either a scalar or m vector of real values of the hypothesized odds ratio.

optn The fourth input argument specifies a number of options in form of one 2-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 the function `xctfishr()` for a table of options.

Output: **est** The first output argument is a $m \times 2$ matrix containing the p values in its first column and the estimated odds ratio in its second column.

ci The second output argument is a $m \times 2$ matrix containing the lower confidence limits in its first column and the upper confidence limits in its second column.

Restrictions: 1. The first input argument must contain integer values.

2. The first two input arguments should have not any missing or string values.

Relationships: xctfishr(), xctbinom(), xctpoiss(), xctfipow()

Examples: 1. :

```
oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"     "twos" ,
         "meth"    "minl" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
< pval, ci > = xctmcnem(x,oddr,optn);


        Exact two-sided Mc Nemar's test (MinLike Method)
                    Testing True Odds Ratio

              Probability= 0.0000 Odds Ratio=0.507937
   Min. Likelihood Method 95 Percent CI: [0.373061,0.688334]


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"     "twos" ,
         "meth"    "cent" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
```

```
< pval, ci > = xctmcnem(x,oddr,optn);


        Exact two-sided Mc Nemar's test (Central Method)
                    Testing True Odds Ratio

              Probability= 0.0000 Odds Ratio=0.507937
   Central 95 Percent Confidence Interval: [0.369933,0.69156]


oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "blak" ];
x = [ 17 64 , 126 769 ];
orng = [ 1.e-3, 1.e3 ];
< pval, ci > = xctmcnem(x,oddr,optn);


        Exact two-sided Mc Nemar's test (Blaker Method)
                    Testing True Odds Ratio

              Probability= 0.0000 Odds Ratio=0.507937
   Blaker 95 Percent Confidence Interval: [0.373061,0.687764]
```

3.16 The `xctpoiss()` Function

< est,ci > = xctpoiss(x,T,r<,optn>)

Purpose: The function `xctpoiss()` computes the estimate of the difference between two exact Poisson rates, the corresponding p value, and its confidence limits. The algorithm is very similar to that of function `poisson.exact` in CRAN package `exactci` by M. P. Fay.

Input: **x** The first input argument is either a scalar, a n vector, or an $m \times 2$ matrix of real values with the number of events.

T The second input argument is either a scalar, a n vector, or an $m \times 2$ matrix of real values with the time base for the event count.

r The third input argument is is either a scalar or m vector of real values with the hypothesized rate or rate ratio.

optn The fourth input argument specifies a number of options in form of one 2-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 the function `xctbinom()` for a table of options. However, as in CRAN's `exactci`, default is here the central method.

The first two arguments x and T can be considered either as single or pairs depending whether only the rate or the rate ratio is estimated. The following input forms are permitted:

1. Both x and T are scalars: estimate one rate.
2. Both x and T are 2-vectors: estimate one rate ratio.
3. One of the two is scalar and the other is n vector: estimate n rates.
4. Both x and T are n vectors with $n \neq 2$: estimate n rates.
5. One of the two is 2-vector and the other is $m \times 2$ matrix: estimate m rate ratios.
6. Both are $m \times 2$ matrices: estimate m rate ratios by treating the input pairwise corresponding.

Output: **est** The first output argument is a $m \times 2$ matrix containing the p values in its first column and the estimated rate or rate ratio in its second column.

ci The second output argument is a $m \times 2$ matrix containing the lower confidence limits in its first column and the upper confidence limits in its second column.

Restrictions: 1. The first three input arguments should be compatible in its size.

2. The first three input arguments should have not any missing or string values.

Relationships: xctbinom(), xctfishr(), xctmcnem()

Examples: 1. Simple example from CRAN package `exactci`:

```
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ];
x = [ 5 4 ]; T = [ 132412, 311312 ]; r = 1.;
< pval, ci > = xctpoiss(x,T,r,optn);


        Exact two-sided Poisson test (Central Method)
                  Testing True Rate Ratio

           Probability= 0.1937 Rate Ratio=2.93886
  Central 95 Percent Confidence Interval: [0.632558,14.8107]
```

```
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"     "twos" ,
         "meth"    "minl" ];
x = [ 5 4 ]; T = [ 132412, 311312 ]; r = 1.;
< pval, ci > = xctpoiss(x,T,r,optn);


         Exact two-sided Poisson test (MinLike Method)
                   Testing True Rate Ratio

             Probability= 0.1381 Rate Ratio=2.93886
    Min. Likelihood Method 95 Percent CI: [0.789558,11.5771]


optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"     "twos" ,
         "meth"    "blake" ];
x = [ 5 4 ]; T = [ 132412, 311312 ]; r = 1.;
< pval, ci > = xctpoiss(x,T,r,optn);


          Exact two-sided Poisson test (Blaker Method)
                   Testing True Rate Ratio

             Probability= 0.1381 Rate Ratio=2.93886
   Blaker 95 Percent Confidence Interval: [0.789558,11.5771]
```

3.17 The `xctsimu()` Function

pval = xctsimu(tab<,optn>)

Purpose: The function `xctsimu()` computes the p values of the exact Fisher test and optional the χ^2 test of a $m \times n$ contingency table by simulation. In addition the common p value based on asymptotic chi^2 distribution is computed. Mike Patefield's `rcont` algorithm is being used for generating contingency tables with specified row and column totals. The function is similar to CRANs `chisq.test` and `fisher.test` functions.

Input: **tab** should be an $m \times n$ table of integer values. It the values are real they are rounded to the next integer.

optn is a numeric vector specifying some runtime options:

[**1**] amount of printed output, default=0

[**2**] number of simulations, default=2000

[**3**] if not zero the p value of χ^2 is simulated too and in addition the asymptotic p is computed, default=0

[**4**] if not zero and input table is $n = m = 2$, Yates correction is applied to asymptotic p value, default=0

Output: The only output argument `pval` is either a scalar (for optn[3]=0) or a numeric vector (for optn[3]=1) containing the p values.

Restrictions: 1. The input argument `tab` may not have missing values or string data.

2. Yates correction of the common asymptotic p value is valid only for 2×2 tables.

Relationships: xctfishr(), conting(), xcthybr()

Examples: 1. Fisher (1962, 1970): Criminal convictions of like-sex twins:

```
tab = [ 2  15 , 10  3 ];
rnam = [ "Dizygotic" "Monozygotic" ];
cnam = [ "Convicted", "Not convicted" ];
tab = cname(tab,cnam); tab = rname(tab,rnam);
print "Tab=", tab;
```

```
Tab=
              |       Convicted   Not convicted
----------------------------------------------
    Dizygotic |               2              15
  Monozygotic |              10               3
```

```
opt = [     2 ,  /* ipri */
         3000 ,  /* nsim */
            1 ,  /* chi too */
            1 ]; /* yates corr */
pval = xctsimu(tab,opt);
```

```
    Probability of simulated exact Fisher test: 0.000333222
     Probability of simulated Chisquared test: 0.00166611
   Yates corr. Chisquare=10.4581 df=1 Probability=0.0012211
```

2. Agresti (2002), p.57: Job Satisfaction:

```
tab = [ 1  3  10   6 ,
        2  3  10   7 ,
        1  6  14  12 ,
```

```
        0  1   9  11 ];
  rnam = [ "< 15k", "15-25k", "25-40k", "> 40k" ];
  cnam = [ "VeryD", "LittleD", "ModerateS", "VeryS" ];
  tab = cname(tab,cnam); tab = rname(tab,rnam);
  print "Tab=", tab;
```

```
Tab=
         |       VeryD     LittleD   ModerateS       VeryS
-------------------------------------------------------------
   < 15k |           1           3          10           6
  15-25k |           2           3          10           7
  25-40k |           1           6          14          12
   > 40k |           0           1           9          11
```

```
  opt = [     2 ,  /* ipri */
            3000 ,  /* nsim */
               1 ]; /* chi too */
  pval = xctsimu(tab,opt);
  print "Pval=", pval;
```

```
      Probability of simulated exact Fisher test: 0.776408
       Probability of simulated Chisquared test: 0.748417
           Chisquare=5.96551 df=9 Probability=0.743365
```

4 Illustrations

4.1 Comparing p Values for Exact Methods in SAS and CMAT

The reason for writing this small report was to find out what could be behind the fact that I get rather different p values for β parameters in exact logistic regression by SAS PROC LOGISTIC and the `elrm` algorithm in CRAN which I have added as function `xctlog` into CMAT.
To keep things simple I think that the one parameter exact logistic regression is very similar to Fisher's exact test. The first table shows the setup of the exact logistic regression problem in CMAT (left column of table) and in SAS (right column of the table).

```
data = [ 17  36  0  ,
         3   30  1  ];
cnam = [" cancer n treat "];
data = cname(data,cnam);

model = "1/2 = 3";
exct = 1;
optn = [ "print"          2 ,
         "rmc"            2 ,
         "burnin"         0 ,
         "iter"       50000 ,
         "trial"          2 ];
< gof,parm,conf > = xctlog(data,
                    model,exct,optn);
```

```
data exctlog1;
  input cancer treat num;
  datalines;
             1   0   17
             0   0   19
             1   1    3
             0   1   27
             ;
run;

proc logistic data=exctlog1 desc;
  freq num;
  model cancer = treat;
  exact treat / estimate=both;
run;
```

The code in CMAT obtains $p = 0.0013$

```
************************************************
Exact Logistic Regression (D. Zamar, 2013, R_MC=2)
************************************************

 Effect    Estimate P-value P_AsyStdErr Nsample Nunique
  treat -2.43127761  0.0013  7.292e-005   50000      15

 Confidence Intervals for Estimates (alpha=0.05)
           Effect        Lower        Upper
            treat -6.219380610 -0.610403011
```

and PROC LOGISTIC

```
                        Exact Parameter Estimates

                                        95% Confidence
          Parameter    Estimate           Limits          p-Value
```

```
treat          -2.0543     -3.8576     -0.6314     0.0019
```

The following table shows the setup of Fisher's exact test in CMAT (left column) and with PROC FREQ (right column):

```
oddr = 1.;
optn = [ "print"       2 ,
         "alpha"    0.05 ,
         "alt"    "twos" ,
         "meth"   "minl" ];
x = [ 17 19 , 3 27 ];
< pval, ci > = xctfishr(x,oddr,optn);
```

```
data;
do a = 1 to 2;
do b = 1 to 2;
  input wt1@@;
  output;
end; end;
cards;
  17  19  3  27
;

proc freq;
  weight wt1;
  tables a * b / exact;
run;
```

The resulting $p = 0.0012$ by CMAT

```
Exact two-sided Fisher's test (MinLike Method)
            Testing True Odds Ratio

        Probability= 0.0012 Odds Ratio=7.80127
Min. Likelihood Method 95 Percent CI: [0.4398,35.3363]
```

is the same as that obtained by PROC FREQ:

```
                Fisher's Exact Test

Cell (1,1) Frequency (F)        17
Left-sided Pr <= F          0.9999
Right-sided Pr >= F      9.621E-04

Table Probability (P)    8.585E-04
Two-sided Pr <= P           0.0012
```

The differences in the p values seem to be small considering the fact that the result of the elrm method in CMAT is based on an MCMC algorithm (however, with 50000 samples drawn). But it seems to be remarkable how close the `elrm` result is to Fisher's exact test.

Name	xctlog elrm	xctfishr Two-sided	SAS Logistic	SAS FREQ Two-sided	SAS FREQ 2*Right-sided
exctlog1: 1	0.0013	0.0012	0.0019	0.0012	0.0019
exctlog1: 2	0.0639	0.0626	0.0977	0.0626	0.0978
exctlog1: 3	0.0272	0.0280	0.0429	0.0280	0.0428
exctlog2: 1	0.1762	0.1773	0.2307	0.1773	0.2308
exctlog2: 2	0.1018	0.1027	0.1338	0.1027	0.1338
exctlog3: 1	0.0952	0.0954	0.1558	0.0954	0.1558
exctlog3: 2	0.0918	0.1016	0.2031	0.1016	0.2032
exctlog3: 3	0.0115	0.0111	0.0183	0.0111	0.0184
exctlog3: 4	1.0000	1.0000	1.0000	1.0000	1.0000
exctlog3: 5	1.0000	1.0000	1.0000	1.0000	1.0000
exctlog3: 6	0.0918	0.1016	0.2031	0.1016	0.2032
exctlog4: 1	0.0841	0.0852	0.1234	0.0852	0.1234
exctlog4: 2	0.0793	0.0869	0.1738	0.0869	0.1738
exctlog4: 3	0.0097	0.0092	0.0123	0.0092	0.0122
exctlog4: 4	1.0000	1.0000	1.0000	1.0000	1.0000
exctlog4: 5	1.0000	1.0000	1.0000	1.0000	1.0000
exctlog4: 6	0.0793	0.0869	0.1738	0.0869	0.1738
exctlog5: 1.1	0.0829	0.0834	0.1165	0.0834	0.1166
exctlog5: 1.2	0.1639	0.1659	0.2363	0.1659	0.2364
exctlog5: 1.3	0.0100	0.0095	0.0174	0.0095	0.0174
exctlog5: 2.1	0.0841	0.0852	0.1234	0.0852	0.1234
exctlog5: 2.2	0.0346	0.0372	0.0745	0.0372	0.0744
exctlog5: 2.3	0.0043	0.0042	0.0053	0.0042	0.0054

That simple model which reduces the general exact logistic regression problem to Fisher's exact test model illustrates that the R program `elrm` (and therefore the CMAT function `xctlog()`) and the SAS PROC LOGISTIC compute the p value differently:

- `elrm` computes the p value as in a two-sided test
- SAS PROC LOGISTIC computes the p value as twice the `Right-sided Pr >= F`.

In all our examples `elrm` never resulted in a p value larger than that which SAS PROC LOGISTIC obtained. Thanks to the developer Bob Derr, of SAS Institute, for pointing that difference out.

5 The Bibliography

References

[1] Agresti, A. (2002), *Categorical Data Analysis*, Second Edition, New York: John Wiley & Sons.

[2] Berkelaar, M., Eikland, K., & Notebaert, P. (2004), *lp_solve (alternatively lpsolve)*, Open source (Mixed-Integer) Linear Programming system, Version 5.5.

[3] Blaker, H. (2000), "Confidence curves and improved exact confidence intervals for discrete distributions"; *Canadian Journal of Statistics*, **28**, 783-798.

[4] Breslow, N. E. & Day, N. E. (1980), *Statistical Methods of Cancer Research; Vol. I: The analysis of Case-Control Studies*, IARC Scientific Publications, IARC Lyon.

[5] Czyzyk, J., Mehrotra, S., Wagner, M. & Wright, S.J. (1997) "PCx User Guide (Version 1.1)", Technical Report OTC 96/01, Office of Computational and Technology Research, US Dept. of Energy.

[6] Fay, M. P. (2010), "Two-sided exact tests and matching confidence intervals for discrete data", *R Journal*, **2**, 53-58.

[7] Fisher, R. A. (1935), "The logic of inductive inference", *Journal of the Royal Statistical Society, Series A*, 39-54.

[8] Fisher, R. A. (1962), "Confidence limits for a cross-product ratio", *Australian Journal of Statistics*, **4**, 41.

[9] Fisher, R. A. (1970), *Statistical Methods for Research Workers*, Oliver & Boyd.

[10] Forrest, J. J. & Lougee-Helmer, R. (2014), *Cbc*, `jjforre@us.ibm.com`, `robinlh@us.ibm.com`.

[11] Forrest, J. J., de la Nuez, D., & Lougee-Helmer, R. (2014), *Clp*, `http://www.tsp.gatech.edu/concorde`.

[12] Gupta, A. & Avron, H. (2000, 2013), *WSMP: Watson Sparse Matrix Package*, *Part I - direct solution of symmetric systems*, *Part II - direct solution of general systems*, *Part III - iterative solution of sparse systems*, version 13.11, IBM Research Division, 1101 Kitchawan Road, Yorktown Heights, NY 10598 `http://www.research.ibm.com/projects/wsmp`

[13] Hirjim, K.F. (2006), *Exact Analysis of Discrete Data*, New York: Chapman and Hall, CRC.

[14] Mehta, C. R. & Patel, N. R. (1986), " Algorithm 643: FEXACT: A Fortran subroutine for Fisher's exact test on unordered $r * c$ contingency tables", *ACM Transactions on mathematical Software*, **12**, 154-161.

[15] "MPS file format", `http://lpsolve.sourceforge.net/5.5/mps-format.htm`

[16] Murtagh, B. A. (1981), "Advanced Linear Programming: Computation and Practice", McGraw - Hill, New York.

[17] Nazareth, J. L. (1987), "Computer Solutions of Linear Programs", Oxford University Press, New York - Oxford.

[18] Patefield, W.M. (1981), "Algorithm AS 159. An efficient method for generating $r * c$ tables with given row and columns totals", *Applied Statistics*, **30**, 91-97.

[19] Powell, J.M.D. (2014), "On fast trust region methods for quadratic models with linear constraints", Report DAMTP 2014/NA02, University of Cambridge.

[20] Zamar, D., Graham, J., & McNency, B. (2007), "elrm: Software implementing exact-like inference for logistic regression models"; *JSS*, **21**.

[21] Zamar, D., Graham, J., & McNency, B. (2013), "Package `elrm`", in CRAN.