The PINGO[1] package

The documention of a software package of the DKRZ (Bundesstraße 55, 20146 Hamburg, Germany) which allows for data processing on workstations with SIMPLE, EXTRA, SERVICE, LOLA, and GRIB files.

by Jürgen Waszkewitz, Peter Lenzen, and Nathan Gillet

1. (Procedural INterface for GRIB formatted Objects)

DISCLAIMER OF WARRANTY

TABLE OF CONTENTS:

1. Introduction 7

2. Getting started 8

- 2.1 File Formats 8
- 2.2 The calling sequence 12
- 2.3 Missing values 14
- 2.4 The “Filling up” of input files and “enlarging” of input records 16

3. Advanced features 18

- 3.1 Protocol file 18
- 3.2 Combining different functions 19
- 3.3 Advanced standard input 21
- 3.4 Not required output files 22
- 3.5 The use of named pipes 23
- 3.6 Grid description files 24
- 3.7 The concept of area weights 27
- 3.8 How to convert files into SIMPLE, EXTRA, SERVICE, LOLA, or GRIB? 30
- 3.9 How to transfer files to other computers? 32
- 3.10 Computations for each level separately 34
- 3.11 About everlasting zombies, parent killing children, and other unimportant things 35

4. The functions 38

- 4.1 Information 38
- 4.2 Formatted input and output 44
- 4.3 Converting the format 48
- 4.4 Generation of files 49
- 4.5 Manipulating the header 52
- 4.6 Manipulating the field 54
- 4.7 Manipulating the sequence of records 59
- 4.8 Selection 62
- 4.9 Missing values 68
- 4.10 Sorting and ranking 71
- 4.11 Arithmetic 72
- 4.12 Maximum and Minimum 76
- 4.13 Mathematical functions 80
- 4.14 Comparisons and Logic 84
- 4.15 Conditions 89
- 4.16 Geometry 91
- 4.17 Means and averages 96
- 4.18 Variances, correlations, and co. 109
- 4.19 Regression 120
- 4.20 Tests, confidence intervals, and co. 123
- 4.21 Empirical Orthogonal Functions 152
- 4.22 Fourier and spectra 157
- 4.23 Interpolation 161
- 4.24 Classes 165

5. References 171

6. List of Functions 172

7. Index of Functions 178

1. Introduction

The PINGO[1] package provides many functions for standard post-processing of climate data sets on workstations. A large range of computations from basic arithmetic over statistics to spectral analysis and empirical orthogonal functions can very simply be done by this package. This package, which is a totally new development of the DKRZ in Hamburg (Germany), is designed for the operating system UNIX.

Contrary to many other existing tools which are related to only one file format this package accept 5 different formats, namely SIMPLE, EXTRA, SERVICE, LOLA, and GRIB. These formats are described in Section 2.1 “File Formats” on page 8. However, the user has not to care about the different input file formats, the program is able to decide alone which format is on. The user has only to decide on the output file format.

Other advantages of the PINGO package are:

- Missing values are supported. If a value other than the default is desired, the user just has to set an environment variable.
- This program can write a complete protocol of all calls.
- A data set can easily be processed by several functions, without storing the interim results in files. (No disk I/O.)
- Clear error messages. If file I/O leads to an error, the reason is printed, e.g. “No space left on device”.
- No “silent waiting”. If a function needs information from standard input, the program always asks the user to type in the desired information, which is then acknowledged with an “OK” message.
- No “seeks” on the input and output files, so they can be named pipes.
- It is very easy for the author to add new functions. An increasingly large range of functions will hence become available to the user.

1. (Procedural INterface for GRIB formatted Objects)

2. Getting started

2.1 File Formats

Every input and output file is made up of a series of records. Each record of an input file must be in SIMPLE, EXTRA, SERVICE, LOLA, or GRIB format. The formats are described further down. Every format is subdivided into different accuracies: 4 bytes or 8 bytes per number in the SIMPLE, EXTRA, SERVICE, and LOLA format and 1, 2, 3, 4, 5, 6, 7, or 8 bytes per number in the GRIB format. (Constant GRIB fields always have an accuracy of 4 bytes.)

GRIB is a WMO standard for meteorological data sets and is described in [1]. Even though no details of the GRIB format will be discussed here, two points should be mentioned: Firstly GRIB is completely self-describing. This means that every GRIB record contains not only information of the date, time, code, level, centre-ID, experiment-ID, etc., but also of the assigned grid, especially of its longitudes and latitudes. Secondly it compresses the data set (with lost of information), which is normally done by choosing an accuracy of 2 bytes per number.

The disadvantage of the GRIB format is its complexity: Because all meteorological data sets can be described by GRIB, it is very difficult to read and to write it. So if the user wants to write an own FORTRAN program to process the records, the GRIB format should be changed into a very simply readable format. For this purpose two formats were developed by the Meteorologisches Institut der Universität Hamburg. The first format developed there was the EXTRA format which is described later on. Its advantage: It can very easily be read and written by a FORTAN program. But caused by its simplicity it contains only a few describing variables: The date, code, level, and field size. It does not contain any grid description.

Because it was felt by the scientists of the Meteorologisches Institut der Universität Hamburg that this is too less, they created a new format called SERVICE. This format contains beside the time variable a little description of the grid: The number of longitudes and of latitudes.

Because the author of this package felt that in some situations, especially when processing regions instead of the whole globe, it would be nice to have an easy format containing also a complete grid description, he created the LOLA format. Since there is a wide range of possible grids he decided to restrict the format to longitude/latitude grids. The name LOLA stands for LOngitude/LAtitude.

The author also created the SIMPLE format which is free of any self describing. Its purpose is the “misuse” of this package for other, not meteorological, purposes.

Every SIMPLE, EXTRA, SERVICE, or LOLA record is either real or complex (whereas a GRIB record is always real). They all consist of a header containing the description part and of a field.

A SIMPLE record can be read in a fortran program by

```
READ(10) NSIZE
READ(10) (FIELD(ISIZE),ISIZE=1,NSIZE)
```

An EXTRA record can be read in a fortran program by

```
READ(10) IDATE,ICODE,ILEVEL,NSIZE
READ(10) (FIELD(ISIZE),ISIZE=1,NSIZE)
```

A SERVICE record can be read by

```
READ(10) ICODE,ILEVEL,IDATE,ITIME,NLON,NLAT,IDISP1,IDISP2
READ(10) ((FIELD(ILON,ILAT),ILON=1,NLON),ILAT=1,NLAT)
```

A LOLA record can be read by

```
READ(10) IDATE,ITIME,ICODE,ILEVEL,NLON,NLAT,LLON,LLAT,IDISP1,IDISP2
READ(10) (XLON(ILON),ILON=1,LLON)
READ(10) (XLAT(ILAT),ILAT=1,LLAT)
READ(10) ((FIELD(ILON,ILAT),ILON=1,NLON),ILAT=1,NLAT)
```

If the accuracy is 4 bytes per number, the variables should be declared respectively as

```
INTEGER*4 NSIZE
REAL*4 FIELD(MSIZE)
```

and

```
INTEGER*4 IDATE,ICODE,ILEVEL,NSIZE
REAL*4 FIELD(MSIZE)
```

and

```
INTEGER*4 ICODE,ILEVEL,IDATE,ITIME,NLON,NLAT,IDISP1,IDISP2
REAL*4 FIELD(MSIZE)
```

and

```
INTEGER*4 IDATE,ITIME,ICODE,ILEVEL,NLON,NLAT,LLON,LLAT,IDISP1,IDISP2
REAL*4 XLON(MLON), XLAT(MLAT), FIELD(MSIZE)
```

with `NSIZE ≤ MSIZE`, `NLON ≤ MLON`, `NLAT ≤ MLAT`. If the record is complex, then the variables must be declared as `COMPLEX*8` instead of `REAL*4`.

In general, on the SUN stations `INTEGER` is the same as `INTEGER*4`, `REAL` is the same as `REAL*4`, and `COMPLEX` is the same as `COMPLEX*8`.

If the accuracy is 8 bytes per number, the variables should be declared as `INTEGER*8` instead of `INTEGER*4`, `REAL*8` instead of `REAL*4`, and `COMPLEX*16` instead of `COMPLEX*8`. Normally on a sun station a `DOUBLE PRECISION` is the same size as a `REAL*8`, and a `DOUBLE COMPLEX` is the same size as a `COMPLEX*16`. (If this is the case, `DOUBLE PRECISION` and `DOUBLE COMPLEX` should be used to let the compiler chose the optimal routine in the case of generic functions.)

The meaning of the variables are:

IDATE	The date as YYYYMMDD with Y=year, M=month or season, D=Day. The months and seasons are coded as 01=January, 02=February, ... , 12=December, 13=season DJF (December,January,February), 14=season MAM (March,April,May), 15=season JJA (June,July,August), 15=season SON (September,October,November)). e.g. 1491230 for the 30th of December of year 149, 1501300 for the northern hemispheric winter beginning December of year 149 and ending February of year 150. (EXTRA, SERVICE, LOLA)
ITIME	The time as HHMM with H=hour and M=minute. (SERVICE, LOLA)
ICODE	The code as described in the DKRZ Technical Report No. 6. (EXTRA, SERVICE, LOLA)
ILEVEL	The level. (This can be model-level (hybrid-level) or pressure level.) (EXTRA, SERVICE, LOLA)
NSIZE	The size of the data file. (SIMPLE,EXTRA)
NLON	The number of longitudes. (SERVICE,LOLA)
NLAT	The number of latitudes. (SERVICE,LOLA)

LLON	(=”listed longitudes”) The size of the longitude description record. Up to three different values are possible. `LLON=NLON` means all single longitudes are given, `LLON=2` means longitudes are equidistantly distributed and only the first two ones are given, `LLON=0` means longitudes are the same as in the record before. If a program needs the longitudes, in all three cases the code
	`IF( LLON.EQ.2 )THEN`
	  `DO I = 3,NLON`
	    `XLON(I) = XLON(1) + (I-1) * (XLON(2)-XLON(1))`
	  `ENDDO`
	`ENDIF`
	constructs the array `XLON`. (LOLA)

`LLAT`	(“listed latitudes”) The size of the latitude description record. Up to three different values are possible. `LLAT=NLAT` means all single latitudes are given, `LLAT=2` means latitudes are equidistantly distributed and only the first two ones are given, `LLAT=0` means latitudes are the same as in the record before. If a program needs the latitudes, use program code analogous to the one given at the explanation of `LLON` to construct the array `XLAT`. (LOLA)
`XLON`	The array of longitudes. (LOLA)
`XLAT`	The array of latitudes. (LOLA)
`FIELD`	The field. (EXTRA, SERVICE, LOLA)
`IDISP1`	(“Dispo #1”) For the users disposal. Can be used for example to save a centre ID or what ever the user likes. (SERVICE,LOLA)
`IDISP2`	(“Dispo #2”) For the users disposal. Can be used to save an experiment number or what ever the user likes. (SERVICE,LOLA)

Functions which require the fields demand the same field size of all simultaneously read input records. Many of these functions allow fields of different sizes in series in one file, e.g. if they represent different regions of the globe. However, some functions cannot work with such input files: They “think” of an input file as containing different realisations of the same spatial field. An example of such a function is the one which computes the mean over all time steps. These functions require the same field size over all records, and have no meaning when applied to files whose records represent several different regions of the globe. A call of one of these functions in such cases would result in an error message.

The field size 1 is an exception and will be discussed in Section 2.4 “The “Filling up” of input files and “enlarging” of input records” on page 16.

There is no rule governing the sequence of records in a file, but it is convention to order the records by date. Records for the same date are then ordered in such a way that all records of the same code are immediately adjacent, and all records of the same date and code are ordered by ascending or descending level.

2.2 The calling sequence

The program can be called by 24 different names. The name used determines the output file format and the accuracy:

Used name	**Output file format**

`pure`
`pure4`
`pure8`
	The output format is PURE (IEEE format)

`smp`
`smp4`
`smp8`
	The output format is SIMPLE

`ext`
`ext4`
`ext8`
	The output format is EXTRA

`srv`
`srv4`
`srv8`
	The output format is SERVICE

`lola`
`lola4`
`lola8`
	The output format is LOLA

`grb`
`grb1`
`grb2`
`grb3`
`grb4`
`grb5`
`grb6`
`grb7`
`grb8`
	The output format is GRIB

The PURE format has no headers and no FORTRAN blocking. PURE files are just a series of `REAL` (or `COMPLEX`) numbers. They cannot be used as input files for this program, but are useful for plotting software, e.g. `grads`, `pvwave`, etc.

If the last character of the program name is a number, it determines the accuracy of the output file in bytes. If no accuracy is explicitly demanded by the user, the program chooses for every record the greatest accuracy of all corresponding input records. (If no input file is given, the smallest accuracy available for this format is used.)

The first argument after the program name must be a function name. The different functions are introduced later.

After the function name all input file names must be given, followed by all output file names.

Example: To add the two files `ifile1` and `ifile2` and store the result in `ofile` just type:

```
ext add ifile1 ifile2 ofile
```

The program then does the addition, and after finishing prints something like

```
ext add: Processed 4 records.
```

Example: To get information on the contents of file `ifile` just type:

```
ext info ifile
```

The program now prints something like

```
  REC :    DATE COD LEVEL      SIZE      MINIMUM       MEAN    MAXIMUM :   MISS
    1 :    1300   1     1         3 :  1.000e+00  2.667e+00  4.000e+00 :      0
    2 :    1400   1     1         3 :  3.000e+00  3.667e+00  4.000e+00 :      0
    3 :    1500   1     1         3 :  4.000e+00  5.000e+00  6.000e+00 :      0
    4 :    1600   1     1         3 :  7.000e+00  8.000e+00  9.000e+00 :      0
ext info: Processed 4 records.
```

Because the function `info` has no output file, the name of the program is ignored, i.e. `srv info` or `grb info` are absolutely identical to `ext info`.

Example: To subtract the constant `273.15` from file `ifile` and store the result in `ofile` just type:

```
ext subc ifile ofile
```

Then the program prints (to standard error):

```
ext subc: Enter constant!
ext subc>
```

and the user has to type in the constant and press return. Then the program prints (to standard error)

```
OK
```

does the calculations and prints (to standard error)

```
ext: Processed 4 records.
```

or whatever the number of records in the input file was.

Some remarks to GRIB records: There are three different kinds of functions: Those with the lowest requirement do not need the array but only the header, examples are `selcode` for selecting a code or `chdate` for changing the date. If no change in the output format is desired, i.e. the program is called by the name “grb”, then these functions (should) except all GRIB records and do not decompress the field. Functions of the second kind need the field, like `add` to add two record, or `mean` to compute the mean over all records. They cannot read all GRIB records, especially not records containing spherical harmonics and some kinds of data packing. And the most demanding functions need not only the field but also the grid information, like `meanr` to compute area weighted means of the records. They can read a smaller subset of all possible GRIB records only compared to the previous functions.

2.3 Missing values

In GRIB records the missing values are stored via a bit map in the GRIB header, so no special value acts as a missing value.

In SIMPLE, EXTRA, SERVICE, and LOLA records there are two different missing values: The value 9.E9 is used for records in 4-byte type, and 9.E99 is used for records in 8-byte type. To change these values, just set the environment variables `MISSVAL4` or `MISSVAL8.` If you do not know which type you are using, you can set both values via the environment variable `MISSVAL`. (But remember, if your missing value is too big to be stored in a `REAL*4`, e.g. 9.E99, only the value for the 8-byte type is changed.) For example:

```
MISSVAL=9999
export
```

in a Bourne-Shell and

```
setenv MISSVAL 9999
```

in a C-Shell.

The use of the missing value is shown in the following tables, where for each operation one table is printed. The operations are applied to arbitrarily number *a*, *b*, the special case *0*, and the missing value *miss*. Grey fields are of particular interest. For example the table named “addition” shows that the sum of an arbitrarily number *a* and the missing value is the missing value, and the table named “multiplication” shows that 0 multiplied by the missing value results in 0.

addition	b	miss
a	a+b	miss
miss	miss	miss

subtraction	b	miss
a	a-b	miss
miss	miss	miss

multiplication	b	0	miss
a	a*b	0	miss
0	0	0	0
miss	miss	0	miss

division	b,b≠0	0	miss
a	a/b	miss	miss
0	0	miss	miss
miss	miss	miss	miss

maximum	b	miss
a	max(a,b)	a
miss	b	miss

minimum	b	miss
a	min(a,b)	a
miss	b	miss

The handling of missing values by the operations “minimum” and “maximum” may be surprising, but it turned out that the definition given here is more related to what is expected in practice. Mathematical functions (e.g. *log*, *sqrt*, etc.) return the missing value if

- An argument is the missing value

  or
- An argument is out of range

All statistics functions ignore missing values, treating them as not belonging to the sample, with the side-effect of a reduced sample size. An artificial distinction is made between the notions mean and average. The mean is regarded as a statistics function, whereas the average is found simply by adding the sample members and dividing the result by the sample size. For example, the mean of 1, 2, miss, and 3 is (1+2+3)/3=2, whereas the average is (1+2+miss+3)/4=miss/4=miss. If there are no missing values in the sample, the average and mean are identical.

2.4 The “Filling up” of input files and “enlarging” of input records

If the called function needs more records from an input file than are available, the function uses a copy of the last record read as often as necessary. This “filling up” stops when all the input files are at end of file. The advantage of this “filling up” is that a file containing a single record behaves like a constant array. Let, for example, two input files, `ifile1` and `ifile2`, be given. Let `ifile1` consists of 12 records and `ifile2` of 1 record. Let us think of `ifile1` as monthly means and `ifile2` as the annual mean. To subtract the annual mean from every monthly mean and to store the result in `ofile`, only the following must be typed:

```
ext sub ifile1 ifile2 ofile
```

That’s all! The program prints on standard error the message:

```
ext sub: Filling up file ifile2 by copying the last record.
```

This last record is the only record, in this case. A graphical demonstration can be found in Figure 1 on page 17 (Advanced users can read the hints for function `null` (page 50) to suppress “filling up”.).

The other automatic adaptation of input records is for fields of size 1 if the chosen function requires the field. These records are “enlarged” to the size of the other input fields which are not 1. Remember: They must all have the same size. Let, for example, the two input files `ifile1` and `ifile2` be given. Let `ifile1` consist of records of field size 2048 and `ifile2` of records of field size 1. Let us think of `ifile1` as a global field and of `ifile2` as its global average. To subtract the global average from the spatial field and to store the result in `ofile`, only the following must be typed:

```
ext sub ifile1 ifile2 ofile
```

The program prints on standard error the message:

```
ext sub: Enlarging record(s) in file ifile2 by copying the one existing
element.
```

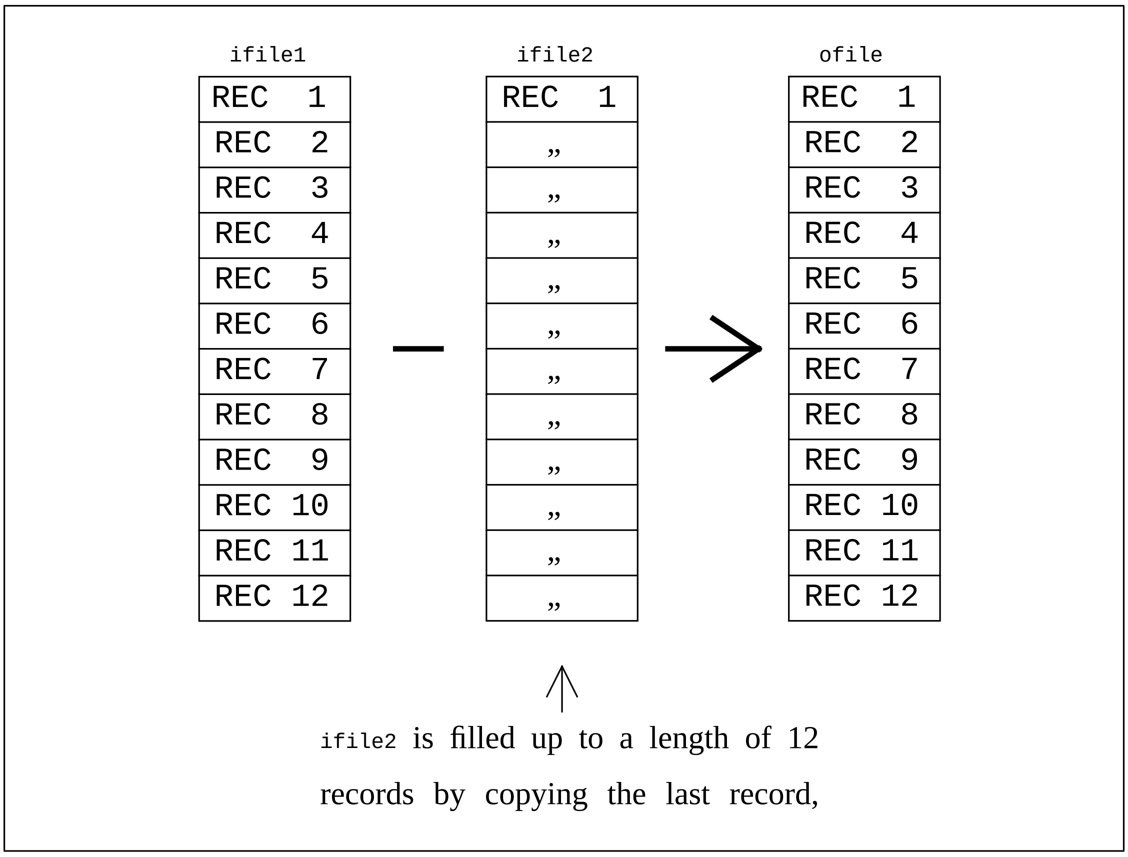


FIGURE 1. "Filling up" a file by copying the last record.

Even though this "enlarging" is done again for each record, the warning is only printed once, to prevent an overfull screen. A graphical demonstration can be found in Figure 2 on page 17.

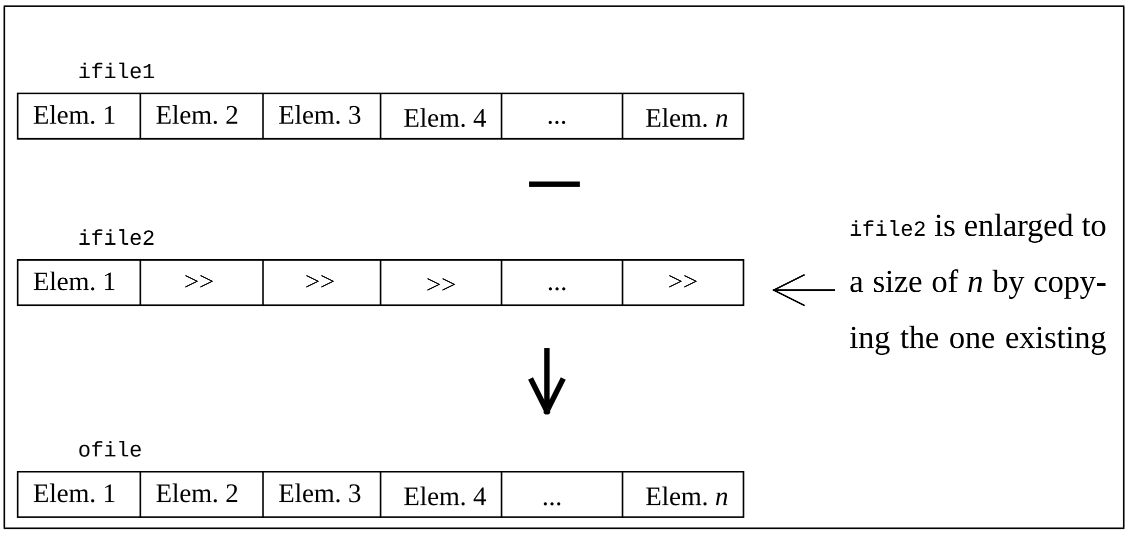


FIGURE 2. "Enlarging" a record by copying the one existing element.

3. Advanced features

3.1 Protocol file

It is possible to make the program write a protocol. To do this, the environment variable `PROTOCOL` must be set to a file name. For example:

```
PROTOCOL=/mf/k/k204099/protocol
export PROTOCOL
```

in a Bourne-Shell and

```
setenv PROTOCOL /mf/k/k204099/protocol
```

in a C-Shell. Every time the program is called and executed (assuming the syntax is correct, and the function exists) the protocol is taken and appended to this file. Always give the complete file name, including the path, not just a file name like “protocol”, otherwise the protocol will always be written in the current directory.

The protocol is an ascii file and can be revisited when the user needs to know of what he actually computed.

An entry in the protocol file looks like this:

```
    (20931) 95-12-06 15:37:35 started in /mf/k/k204099/datas:
ext add ifile1 ifile2 ofile
    (20931) 95-12-06 15:37:36 Processed 4 records.
    (20940) 95-12-06 15:43:32 started in /mf/k/k204099/datas:
ext info ifile
    (20940) 95-12-06 15:43:33 Processed 4 records.
    (21720) 95-12-06 15:53:25 started in /mf/k/k204099/datas:
ext subc ifile ofile
    (21720) 95-12-06 15:55:16 input values: 273.15
    (21720) 95-12-06 15:55:16 Processed 4 records.
    (22003) 95-12-06 16:02:35 started in /mf/k/k204099/datas:
pure copy ifile ofile
    (22003) 95-12-06 16:02:36 Processed 4 records.
```

The numbers in the brackets are the process ID’s. They are of interest in situations where the user is doing several calculations at the same time, so that it is clear which message belongs to which program call. The process ID is followed by the date, time and action that is being recorded.

If values other than the default missing values are used, this is also written into the protocol file.

A user who wants to kill a process by using the unix command `kill` should not use the `-9` option to enable the process to write its termination note into the protocol file.

3.2 Combining different functions

If a computation must be done in several steps, it is not necessary to save all the interim results on disk between the different calls. Furthermore, on a multiple processor machine it is possible to do the computations in parallel and allow the processes to communicate by internal pipes. The calling sequence is straight forward.

First Example: To compute mean after selecting code 167 of `ifile` and store the result in `ofile`, the user could type

```
ext selcode ifile temp
ext mean temp ofile
rm temp
```

BAD EXAMPLE !

The way to combine the functions `selcode`, for selecting a code, and `yearmeans`, for computing the annual means, is as follows:

```
ext mean -selcode ifile ofile
```

PREFERRED SOLUTION!

The program would write

```
ext mean: Started child process "ext(2) selcode ifile (pipe 2.1)"
ext(2) selcode: Enter code!
ext(2) selcode>
```

The user has now to type in the code number `167` and the computer prints

```
OK
ext(2) selcode: Processed 1000 records.
ext mean: Processed 100 records.
```

What happened? The program created a child process which behaves like the command `ext selcode ifile (pipe 2.1)`. The `(2)` in `ext(2)` which was printed on the screen is the position of the `-selcode` in the argument list of the calling sequence and is needed to easily identify the child in situations where there is more than one child. `(pipe 2.1)` is the output file of the `selcode` function and is really an internal pipe. The other end of this pipe acts as the input file of the function `mean`.

The general rule for combining functions is as follows: If, instead of an input file name, there is something beginning with “-“, the program assumes that this “-“ is immediately followed by a function name, which itself is followed by its input file names, all separated by blanks. This function is started as a child process, and every output file of this child is substituted for an input file of the parent function.

It is possible to combine functions to an arbitrary depth of levels: Let us assume that in the first example only the mean of the years 1950 to 1979 is to be subtracted, not the mean over all records of the desired code. Before computing the mean, the records containing a year between 1950 and 1979 must be selected by the function `selyear`. The combined call could be

```
ext mean -selyear -selcode ifile ofile
```

The program then prints something like

```
ext mean: Started child process “ext(2) selyear -selcode ifile (pipe 2.1)”.
ext(2) selyear: Started child process “ext(3) selcode ifile (pipe 3.1)”.
ext(3) selcode>
```

The user has now to type in the code number `167` and the computer prints

```
OK
ext(2) selyear: Enter start year and end year!
ext(2) selyear>
```

The user has now to type in `1950 1979` and the computer prints

```
ext(3) selcode: Processed 1000 records.
ext(2) selyear: Processed 100 records.
ext mean: Processed 30 records.
```

The last example shows how complex formulae can be calculated in one call. As a further example, let us assume that the correlation of two time series is to be estimated without assuming normally distributed samples. The formula is

$$\frac{1}{(n^3 - n)/6} \sum_{i=1}^{n} (R_i^{(1)} - R_i^{(2)})^2 \qquad (1)$$

where n is the length of the time series and $R_i^{(j)}$ is the range of the i-th observation of time series $j = 1, 2$, for example for observations $(4.4, 5.1, 5.6, 2.3, 5.1, 7.8)$ the assigned ranks are $(2, 3.5, 5, 1, 3.5, 6)$, because 2.3 is the greatest number, 4.4 the

2nd greatest, 5.1 the 3rd and 4th greatest number, etc. If the time series are stored in the files `ifile1` and `ifile2` and if the result should be stored in `ofile`, then the call of the functions could be

```
ext divc -sum -sqr -sub -rang ifile1 -rang ifile2 ofile
```

(This kind of notation is similar to what is called “Polish notation”.) The function `divc` (division by a constant) will ask for a constant and the value for $(n^3 - n)/6$ should be typed in. The user then has to type in the value of $(n^3 - n)/6$. Five child processes are started, the dependencies of them are indicated in the following table.

Pos:	1	2	3	4	5	6	7	8	9
ext	divc	-sum	-sqr	-sub	-rang	ifile 1	-rang	ifile 2	ofile
					-rang	ifile 1	-rang	ifile 2	
					(pipe 5.1)		(pipe 7.1)		
				-sub	(pipe 5.1)		(pipe 7.1)		
				(pipe 4.1)					
			-sqr	(pipe 4.1)					
			(pipe 3.1)						
		-sum	(pipe 3.1)						
		(pipe 2.1)							
	divc	(pipe 2.1)							ofile

3.3 Advanced standard input

In some situations, the user might wish to type the standard input in at the command line, before calling the function. This can usually be done quite easily, by using “<<“ to redirect the standard input to the shell

```
ext info -subc ifile <<eoi
273.15
eoi
```

Or the user can redirect the standard input from a file, which is sensible for functions requiring a lot of standard input. This is done using the “<“ sign, as is shown below:

```
ext info -subc ifile < input
```

The file `input` must then contain the numbers. N.B. Anything written after an initial number or any line which does not start with a number is treated as a comment.

However, if several processes are started from within one command line, this procedure does not work: It is not clear which of the processes involved is picking up the redirected standard input. The following two command lines are examples of such situations:

```
ext info -subc ifile | more
```

BAD EXAMPLE !

Both processes, `ext` and `more`, are picking up data from the standard input channel.

```
ext sub -selrec ifile1 -selrec ifile2 ofile
```

In this example, where selected records of `ifile1` and `ifile2` are to be subtracted, it is unpredictable which child process (`ext(2) selrec ifile1 (pipe 2.1)` or `ext(4) selrec ifile2 (pipe 4.1)`) will be the first to ask for the start and end record numbers.

Hence, a better tool than “<<“ is needed for redirecting the standard input. This tool exists, and is very simple to use: **Just type the numbers required by each function directly after the function name, dividing them by commas**, e.g.

```
ext info -subc,273.15 ifile | more
```

PREFERRED SOLUTION!

and

```
ext sub -selrec,1,12 ifile1 -selrec,13,24 ifile2 ofile
```

PREFERRED SOLUTION!

If too few numbers are given, the function switches from reading the command line numbers to reading the standard input. Excess given numbers are ignored.

3.4 Not required output files

Some functions produce more than one output file. If the user is not interested in all of them, he might type “-” instead of an output file name. The “-” has the same effect as the file name `/dev/null`, except that it saves some computation time.

For example, if the user wants to split the file `ifile` seasonally and store the result in the files `ofile.djf.ext`, `ofile.mam.ext`, `ofile.jja.ext`, `ofile.son.ext` he can use the function `splitseas`:

```
ext splitseas ifile ofile.djf.ext ofile.mam.ext ofile.jja.ext ofile.son.ext
```

But if he is only interested in winter and summer, he could type

```
ext splitseas ofile.dfj.ext - ofile.jja.ext -
```

If not all of the output files of a function which is started as a child process should be used by the parent function (see Section 3.2 "Combining different functions" on page 19), put the functions `selfile<m>of<n>` (page 67) resp. `selfile<l>and<m>of<n>` (page 67) resp. `selfile<k>and<l>and<m>of<n>` (page 68) between the pipes.

3.5 The use of named pipes

In some situations, the user may wish to have named pipes as input or output files. If, for example, he wants to create two pipes with the names `pipe1` and `pipe2`, he just has to type

```
mknod pipe1 p
mknod pipe2 p
```

`mknod` is a unix command to create special files like pipes. If the user now prints a "long" file list by typing `ls -l`, he sees the following entries in the list of files:

```
prw-rw-rw-  1 k204099         0 Jan 10 15:25 pipe1
prw-rw-rw-  1 k204099         0 Jan 10 15:25 pipe2
```

The "`p`" at the left indicates a pipe.

To prevent disturbances by other users, he should take away the read and write permission of the group and others with the unix command `chmod go-rw pipe[12]`.

Once the pipes are created they can be used as often as desired, **BUT BEFORE USING A PIPE A SECOND TIME, IT SHOULD BE ENSURED THAT THERE ARE NO PROCESSES STILL WRITING TO OR READING FROM IT!** Because at the same time there should at most one process read from the pipe and at most one process write on the pipe. Otherwise one get typically "ERROR! File <pipename> has wrong format!" The easiest way to kill all processes writing to or reading from a pipe is to remove the pipe and create it again.

The reading process and the writing process are synchronizing themselves so that the reading process reads from the pipe with the same speed than the writing process writes on it. But if no process is writing on the pipe, a reading process would sleep (but not stop!), and if no process is reading from the pipe, a writing process would sleep (and again: not stop!).

Imagine now that the user has to compute the monthly means of the daily minimum temperature, given a number of 12-hour data sets. Imagine further that the input data file is too large to be stored on his machine, but is available on another machine called `schauer`, and that the result is to be stored on a machine called `local`. Provided the user has a `.netrc` file in his home-directory with the login and password of the machine `schauer`, he could type

```
ftp schauer <<eoi &
get ifile pipe1
quit
eoi
ext monmeans -daymins pipe1 pipe2 &
rcp pipe2 local:ofile &
wait
```

It is important to let all the processes involved run concurrently, but the calling sequence is unimportant.

And again, because it is a typical source of error:

EXCEPT FOR THE TWO DESIRED PROCESSES THERE MUST BE NO OTHER SLEEPING OR ACTIVE PROCESSES

The easiest way to ensure this is to remove the pipe and create it again.

3.6 Grid description files

In some situations it is necessary to give a description of a grid. These situations are

- the change of a format from the SIMPLE, EXTRA, or SERVICE format to the LOLA format as described in Section 3.8 “How to convert files into SIMPLE, EXTRA, SERVICE, LOLA, or GRIB?” on page 30

- functions that need the grid information, for example `grads` (page 40) to produce a `grads` description file, `sellonlatbox` (page 67) to select a box in longitude and latitude coordinates, or `weight0` (page 49) to construct a weight files as described in Section 3.7 “The concept of area weights” on page 27
- the function `interpolate` (page 161) which interpolate fields from one grid to another.

A grid description file is an ASCII formatted file. The first and second entries in these files are the numbers of longitudes and latitudes of the grid, followed by a description of the longitudes and then followed by a description of the latitudes. Hence only longitude/latitude grids can be described in a grid description file. The description of the longitudes (latitudes) starts with the number of “listed longitudes”, that is the number 2 if the longitudes (latitudes) are equidistant, or again the total number of longitudes (latitudes). In the case of equidistance the first 2 longitudes (latitudes) are listed, in the other case all longitudes (latitudes) are listed. Longitudes are always listed from west to east, latitudes may be in descending or ascending order.

For some standard resolutions there are existing “ready to use” grid description files. They can be found in the directory `grids`, a subdirectory of the directory containing the executable program. Before continuing, it is best to set an environment variable like `GRIDS` with this directory name. Use the command

```
which ext
```

to find the directory of the executable program.

In this directory, henceforth referred to as `$GRIDS`, are some useful files. All their names begin with a grid name. The grid names are `t21`, `t42`, `t106`, `r72x36`, `r72x37`, `r144x73`, `r180x90`, `r180x91`, `r360x180`, `r360x181`, `r720x360`, `r720x361`, where `t<n>` indicates a Gaussian grid and `r<m>x<n>` a regular grid with <*m*> longitudes and <*n*> latitudes. There are 2 files per grid; for example, for the T21 grid there are the files

```
t21.grid.asc
t21.weights.lola
```

The first one is the grid description file of a T21 grid. The second is a weight file. Weight files are explained in the Section 3.7 “The concept of area weights” on page 27.

The contents of `t21.grid.asc` are

```
Grid Description File
(Comments start at non digit characters and end at end of line)
First part: The dimensions.
64 32 = Number of longitudes and latitudes
Second part: The listed longitudes.
2 means equidistant longitudes
0.000000 5.625000 = Most western and second most western longitude
Third part: The listed latitudes.
32 means all 32 latitudes are given in the following list:
 85.761  80.269  74.745  69.213  63.679  58.143  52.607  47.070  41.532  35.995
 30.458  24.920  19.382  13.844   8.307   2.769  -2.769  -8.307 -13.844 -19.382
-24.920 -30.458 -35.995 -41.532 -47.070 -52.607 -58.143 -63.679 -69.213 -74.745
-80.269 -85.761
```

The function `griddes` (page 40) can be used to print the grid description file of LOLA and GRIB files.

In the “ready to use” grid description files the latitudes are given from north to south. The regular grids with an odd number of latitudes and the Gaussian grids have the first longitude exactly at 0˚. The grids r<*m*>x<*n*> with an even number <*n*> of latitudes have the first longitude at (180/<*n*>)˚. (The idea is that values on regular grids with an even number of latitudes are the means of boxes. The western borders of the boxes of longitudes of index 1 are therefore 0˚.) The longitudes are equally distributed from west to east. The following figure illustrates the situation for a regular grid with an odd number of latitudes on the left, and for an even number of latitudes on the right.

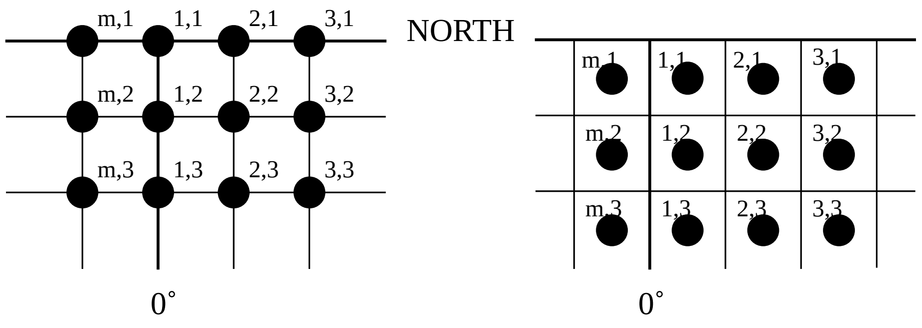


FIGURE 3. The location of the grid points of regular grids. On the left, the circles represent the grid points for a grid consisting of an odd number of latitudes, and on the right, for an even number of latitudes. The number pairs beside the circles are the longitude and latitude indexes.

The use of grid description files is as follows: Let the user for example wishes to select a box given in longitude and latitude coordinates. The function `sellonlatbox` (page 67), which is doing this job, asks for a complete grid description if the input file does not contain the grid information, that is it is SIMPLE, EXTRA, or SERVICE. Assuming the input file `ifile.ext` is in T21 the user could type

```
cat $GRIDS/t21.grid.asc - | lola sellonlatbox ifile.ext ofile.lola
```

and press `control-D` after finishing the input of the box coordinates.

3.7 The concept of area weights

What are area weights? - Typically in a climate data set of the globe or a part of the globe the desired physical quantity is given on a grid: One single point represents a mean of a box around it. These boxes are typically of different sizes, because the grid points are of varying distance. Even if the grid points have the same longitudinal and latitudinal distance the real distances between them is varying: The longitudes are very close together near the poles and widely spaced at the equator. For example the distance between two longitudes at 60˚N is only half as large as at the equator, because the cosine of 60˚ is 0.5.

While computing the global mean in this example a value at the equator must be weighted twice as much as a value at 60˚N. The idea is now to assign every field element of a record an area weight, which is the size of the area represented by this element. If for every field position i the area weight is denoted by w_i, then the area mean is computed as $\left(\sum_i w_i\right)^{-1}\sum_i w_i x_i$, where x_i is the value of the field at position i.

Many of the functions ending with the letter “r”, which means “for each record”, like `meanr`, which computes the mean of a record, need area weights.

Whenever a function needs area weights and the input record is a GRIB or LOLA record, or, in the case of functions with more than one input file, if one of the input records is a GRIB or LOLA record, the grid information of this GRIB resp. LOLA record is used to compute the area weights. But if the involved records only have SIMPLE, EXTRA, or SERVICE format a warning is printed and all grid points are weighted the same.

If this is not desired or if the grid used in a GRIB record is not supported (“`GRIB ERROR: Weights cannot be calculated - map type <n> not currently supported!`”), or other weights than that suggested by the stored grid should be used, e.g. because the grid information in the GRIB record is wrong (the user can use function `griddes` (page 40) to print the assigned grid description as explained in Section 3.6 “Grid description files” on page 24), then the area weights must supplied separately by using special weight files. Weight files are files of record length 1 corresponding to a particular grid. They contain the area weights of that grid as field elements.

To use the weight files the user has to call another function which has the same name but with an additional letter “w” at the end, e.g. `meanrw`, which means “(using a weight file)”. These functions have exactly one more input file, namely the weight file.

For some standard resolutions there are existing “ready to use” weight files. They can be found in the directory `grids`, a subdirectory of the directory containing the executable program. Before continuing, it is best to set an environment variable like `GRIDS` with this directory name. Use the command

```
which ext
```

to find the directory of the executable program.

In this directory, henceforth referred to as `$GRIDS`, are some useful files. All their names begin with a grid name. The grid names are `t21`, `t42`, `t106`, `r72x36`, `r72x37`, `r144x73`, `r180x90`, `r180x91`, `r360x180`, `r360x181`, `r720x360`, `r720x361`, where `t<n>` indicates a Gaussian grid and `r<m>x<n>` a regular grid with <*m*> longitudes and <*n*> latitudes. There are 2 files per grid; for example, for the T21 grid there are the files

```
t21.grid.asc
t21.weights.lola
```

The file `<`*`grid`*`>.grid.asc` is a grid description file as explained in Section 3.6 “Grid description files” on page 24. The file `<`*`grid`*`>.weights.lola` is the area weight file. The weights are normalised to the sum of 1, but there are no functions that require this. Weight files are used in the following way: Let us assume that the user wants to compute the global means of T21 fields stored in the file `ifile`, and wants to store the result in `ofile`. The user just has to type

```
ext meanrw ifile $GRIDS/t21.weights.lola ofile
```

Considering the fact that there is no grid information stored in SIMPLE, EXTRA or SERVICE files, it can be seen that weighted means are still easy to calculate. (The function name `meanrw` (page 97) stands for “mean for each record (using a weight file)”).

How can the mean over a particular area (which is assumed to be a rectangular box) be computed? There are two ways: The direct way uses the function `selindexbox` (page 66) or `sellonlatbox` (page 67) to select the box:

```
ext meanrw -selindexbox ifile -selindexbox $GRIDS/t21.weights.lola ofile
```

where the coordinates of the box must be given twice and must be the same. (The selected box of the weight file is not normalised to the sum of 1, but as already mentioned, this does not matter.)

To avoid (twice) typing the coordinates of the box every time the mean over this area is computed, it is also possible to construct a special weight file for the desired area, using the function `maskindexbox` (page 55) or `masklonlatbox` (page 55). This function does not actually remove the data points outside the desired box, but rather it sets all these entries to the missing value. Such a weight file could hence be constructed by typing

```
lola masklonlatbox $GRIDS/t21.weights.lola t21.weights.my_area.lola
```

and, once generated, used with

```
ext meanrw ifile t21.weights.my_area.lola ofile
```

If the user wishes to work with a longitude/latitude grid other than those with pre-prepared weight files, the best method is first to construct a grid description file, and then construct the weight file, or, much easier, if there is a GRIB file available containing a description of this grid, use the function `weight1` (page 50) to construct the weight file of this grid.

To construct the weight file for a given grid description file, use the function `weight0` (page 49). e.g.

```
lola weight0 r72x36.weights.lola < r72x36.grid.asc
```

If weight files are to be constructed for only land or sea, the user first has to find a sea/land mask. This is a task the user has to do for himself. Once he has such a mask, he can easily construct the desired weight files using the functions `ifthen` (page 89) and `ifnotthen` (page 90). For example, if the sea/land mask is called `r72x36.slmask.ext`, he can type:

```
lola ifthen r72x36.slmask.ext r72x36.weights.lola r72x36.weights.land.lola
lola ifnotthen r72x36.slmask.ext r72x36.weights.lola r72x36.weights.sea.lola
```

The new weight files can be normalised by using the function `normalize` (page 56) function, although this is not necessary.

```
lola normalize -ifthen r72x36.slmask.ext r72x36.weights.lola \
  r72x36.weights.land.lola
lola normalize -ifnotthen r72x36.slmask.ext r72x36.weights.lola \
  r72x36.weights.sea.lola
```

If the user wishes to work with a grid which is not a longitude/latitude grid, he must write his own FORTRAN program to produce the weight file.

In some situations it is defensible to use constant weights to get a first impression. Then function `const` (page 50) applied as `-const,1` can be used instead of the weight file:

```
ext meanrw ifile -const,1 ofile
```

3.8 How to convert files into SIMPLE, EXTRA, SERVICE, LOLA, or GRIB?

We consider two possibilities: In the first case the user has already a SIMPLE, EXTRA, SERVICE, LOLA, or GRIB file and wants to change it into another of one of these five formats, and in the second case the user has an ascii file. In all other cases, the file should first be converted to ascii format.

As regards the first case: The user can use the function `copy` (page 48) for converting to SIMPLE, EXTRA, SERVICE, or LOLA and the function `copy2` (page 48) for converting to GRIB. These functions are copying the contents of the input file to the output file and are changing the format due to the given program name. Remember: The name how the program was called determines the output format.

First example: To change the GRIB file `file.grb` to the EXTRA format, just type

```
ext copy file.grb file.ext
```

Second example: To upgrade the format of the SERVICE file `file.srv` to the LOLA format, assuming that the grid is T21, just type

```
lola copy file.srv file.lola < $GRIDS/t21.lola.asc
```

`$GRIDS/t21.lola.asc` is a grid description file which is explained in Section 3.6 “Grid description files” on page 24 and `GRIDS` is an environment variable which should be set as described in that section.

Even though some information of the grid is known, namely the number of longitudes and latitudes, the program asks for the complete description of the grid to facilitate the use of a grid description files.

Third example: Let us assume that the EXTRA file `file.ext` contains data belonging to a Gaussian T21 grid and that `any_t21_data_set.grb` is a GRIB file containing a T21 data set. To change now the EXTRA file `file.ext` to GRIB format containing the T21 grid description of `any_t21_data_set.grb`, the user has to type

```
grb2 copy2 file.ext any_t21_data_set.grb file.grb
```

(The trick is that all GRIB records that are written by this program get the description part of the last read in GRIB record which is in this case from `any_t21_data_set.grb`.)

If the program would be called as `grb` instead of `grb2`, the accuracy of the GRIB record of `file.grb` would be the greatest accuracy of both input files. This is normally not desired, so **CALL `grb2` INSTEAD OF `grb`**.

As regards the second case: Imagine that the user has been given an ascii format file called `file.asc`, and wants to make an EXTRA file called `file.ext`, or a SERVICE file called `file.srv`, or a LOLA file called `file.lola` out of it. Let us assume that the file `file.asc` is of an arbitrary format, but with at least one blank, tab, or new line between two numbers. (If this is not the case the user could use the unix commands `cut`, `paste`, or `vi` to insert blanks between the numbers.) Also important: **The exponential ascii representation must not contain a `D` or `d`, it must always be `E` or `e`, so change `1.00D-04` to `1.00E-04` for example.**

To start with, the user must create an EXTRA (or SERVICE or LOLA) file by using the function `input` (page 44). If the size of the field of each record is 2592, if the dates of the records are to be numbered from 18530100 in monthly steps, if the time is 0 and must not be increased, if the code is 167, and if there is only 1 level per date which is 0, he could type:

```
ext4 input,2592,18530100,100,0,0,167,1,0 file.ext < file.asc
```

Or, if the user wants to create a SERVICE file called `file.srv` with 72 longitudes, 36 latitudes, and the same header, he could type:

```
srv4 input,72,36,18530100,100,0,0,167,1,0 file.srv < file.asc
```

Or, if the user wants to create a LOLA file called `file.lola` with the same header, he could type

```
echo 18530100 100 0 0 167 1 0 72 36 | cat $GRID/r72x36.grid.asc - file.asc |\
  lola4 input file.lola
```

where `$GRID/r72x36.grid.asc` is a grid description file, see Section 3.6 "Grid description files" on page 24 for details.

After doing this, the user can check the contents of `file.ext` (or `file.srv` or `file.lola`) with the function `info` (page 38), or with `longinfo` (page 39).

It may then be necessary to shift the field to the left or to the right, or swap left with right or the top with the bottom to give consistency with other existing files. For this, the user can use the function `shiftleft` (page 56), `shiftright` (page 56), `swapleftright` (page 56), or `swaptopbottom` (page 57).

Lastly, it may be necessary to set some numbers to the missing value. For example, the missing value in the data could be -9999, and let us assume that the user uses the default missing value, so that -9999 appears in the data as -9999 and not as a missing value. To change a constant into the missing value, the user should call the function `setctomiss` (page 68).

The other way around, converting SIMPLE, EXTRA, SERVICE, LOLA, or GRIB files to ascii, is just to call one of the functions `output` (page 46), `outputsmp` (page 46), `outputext` (page 46), `outputsrv` (page 47), or `outputlola` (page 47).

3.9 How to transfer files to other computers?

Because GRIB is a standard, GRIB files can be transferred from one computer to another without taking care about internal number representations.

So it remains the question of how transfer EXTRA, SERVICE, or LOLA files between computers with different internal number representations. Two ways are possible:

The first is to change the format of the files into GRIB, transfer the GRIB file, and change the format back. (For details see Section 3.8 “How to convert files into SIMPLE, EXTRA, SERVICE, LOLA, or GRIB?” on page 30.) But in GRIB the numbers are compressed, so that after changing the format back to EXTRA, SERVICE, or LOLA they might have been changed a little.

The second way is to convert the file to ascii format, and then compress it if the data transfer is slow. After transfer, the file has to be uncompressed if necessary, and then converted back into EXTRA, SERVICE, or LOLA format. (The size of the compressed ascii file is of the same magnitude as that of the original file).

Let, for example, the SIMPLE file `ifile` be given. The commands at the sender side are

```
smp outputsmp ifile > ifile.asc
compress ifile.asc
```

The `compress` command generates a file named `ifile.asc.Z` which can now be sent. The receiving user has to type

```
uncompress ifile.asc.Z
smp inputsmp ifile < ifile.asc
```

If `ifile` is an EXTRA file, the sender should type

```
ext outputext ifile > ifile.asc
compress ifile.asc
```

and the receiving user should type

```
uncompress ifile.asc.Z
ext inputext ifile < ifile.asc
```

If `ifile` is a SERVICE file, the sender should type

```
srv outputsrv ifile > ifile.asc
compress ifile.asc
```

and the receiving user should type

```
uncompress ifile.asc.Z
srv inputsrv ifile < ifile.asc
```

If `ifile` is a LOLA file, the sender should type

```
lola outputlola ifile > ifile.asc
compress ifile.asc
```

and the receiving user should type

```
uncompress ifile.asc.Z
lola inputlola ifile < ifile.asc
```

3.10 Computations for each level separately

Sometimes it is desirable that computations should be performed for each level separately. For example, imagine that the monthly averages are to be computed for an input file of temperature at 15 levels. The simple call

```
ext monavg ifile ofile
```

does not work correctly, because the function `monavg` computes the average of all fields for each month, with no regard to level. If, for one month, the 15 levels are stored consecutively, then the averages could be correctly calculated by typing:

```
ext split15 ifile o01 o02 o03 o04 o05 o06 o07 o08 o09 o10 o11 o12 o13 o14 o15
ext monavg o01 m01
ext monavg o02 m02
ext monavg o03 m03
ext monavg o04 m04
ext monavg o05 m05
ext monavg o06 m06
ext monavg o07 m07
ext monavg o08 m08
ext monavg o09 m09
ext monavg o10 m10
ext monavg o11 m11
ext monavg o12 m12
ext monavg o13 m13
ext monavg o14 m14
ext monavg o15 m15
ext merge15 m01 m02 m03 m04 m05 m06 m07 m08 m09 m10 m11 m12 m13 m14 m15 ofile
rm o?? m??
```

BAD EXAMPLE !

In the first step `ifile` was divided by `split15` (`split<n>` (page 60)) into 15 files. Every record of `ifile` at the first level is now stored in `o01`, every record at the second level in `o02`, etc. The computation of the monthly averages is now done separately for each level in the next 15 steps. Afterwards, all the levels are merged together by the function `merge15` (`merge<n>` (page 60)).

A short-cut exists which summarizes all these steps. Just type

```
onlevels ext monavg ifile ofile
```

PREFERRED SOLUTION!

The job `onlevels` needs a program name as its first argument, a function name as its second argument, then an arbitrary number of input file names followed by one output file name. Input files may consist of only one level. These files are treated as if they had the same field on all levels, thus, for example, the correlation between the single level precipitation stored in `ifile1` and the multi level temperature fields stored in `ifile2` can be calculated with the following call

```
onlevels ext cor ifile1 ifile2 ofile
```

This job creates temporary files in the directory `$TMPDIR`. If not enough disk space is available in this directory, change it to another one.

3.11 About everlasting zombies, parent killing children, and other unimportant things

This chapter can be skipped, as knowledge of its contents is not necessary for use of the program.

Irrespective of the program name, the output format of a child process is an internal pipe format. It is GRIB like, if one of the input files is GRIB, otherwise it is LOLA like if the grid is known, otherwise it is SERVICE like, if the number of latitudes is known, otherwise it EXTRA like. It is never PURE or SIMPLE. If a GRIB input field was decompressed, then no compression is made for output.

If a process receives the `CONT`-signal, it prints after a while a status message. This can be used for functions which need very long time for computation to see what they are just doing. The function `status` (page 44) can be used to find out this signal number. Let for example this signal number be 19. Then the unix command `kill -19 <process ID>` sends this signal to the process which status is to be asked. It does not kill it! C-shell users can also stop this process by pressing `CONTROL-Z` and starting it again by the shell command `%`. This starting makes the C-shell sending the `CONT`-signal to the process.

If a child process is not needed any longer it will be killed. For example, if the user wants to select only the first record of `ifile` after subtracting 273.15 from it, he types:

```
ext selfirstrec -subc ifile ofile
```

The computer writes

```
ext selfirstrec: Started child process: “ext(2) subc ifile (pipe 2.1)”
ext(2) subc: Enter constant!
ext(2) subc>
```

The user types `273.15` and gets

```
OK
ext selfirstrec: Processed 1 record.
```

After the function `selfirstrec` selected the first record, there was no need to evaluate the function `subc` any longer, thus an interrupt signal was sent from the `selfirstrec`-function to the `subc`-function. A look at the protocol file confirms this:

```
    (  831) 96-01-10 13:08:53 started in /mf/k/k204099/datas:
ext selfirstrec -subc ifile ofile
    (  833) 96-01-10 13:08:53 started child of 831:
  ext(2) subc infile (pipe 2.1)
    (  833) 96-01-10 13:08:58 input values: 273.15
    (  833) 96-01-10 13:08:58 received signal 2 (Interrupt)!
    (  831) 96-01-10 13:08:58 Processed 1 records.
```

A remark concerning parallel computations: If several processes need to print on standard output or standard error at the same time, this does not lead to a chaotic screen, because there is synchronisation of all the processes involved. Only one process at a time can print to one of both, the standard output and the standard error. If a process needs standard input, it locks the screen from the moment it prints the request for input, until after the input has been given. This behaviour ensures that the correct process receives the input data.

If a process encounters an error or receives an error signal, it kills all its children by sending the signal “Broken Pipe” and, if it is not the root process, it also kills the root process, again by sending it the “Broken Pipe” signal. Thereafter this process pauses if it is not the root process. It does not close the pipes until this pause. If the root process is killed by a signal, it returns a nonzero exit code. This behaviour ensures that the user gets a nonzero exit code if one of the children or one of their descendants encounters an error, and it prevents the occurrence of everlasting “zombies”. (If a process has died, its “soul” (which is its exit code) must be “rescued” by its parent process, but if this parent is already dead, an everlasting “zombie” would arise. The user can recognise “zombies” by the status “`Z`” in a process list created by the unix command “`ps`“, but it is not possible to kill them, since they are already dead. A process will kill all its children and its grandparent to prevent itself becoming an everlasting zombie. What a cruel world!)

A last topic: In the directory of the executable program there is a directory called `stat`. This directory contains one file named `statistic`. After every successful start the program tries to write one line of statistical information into this file to enable the authors to separate the functions into important and less important ones. The version number, the date, a hash of the user name, the name how the program was started, and all involved functions are written into this file which is readable to everyone. To avoid disadvantages in speed the statistic is written by a child process which is destroying itself at the latest after 60 seconds.

4. The functions

This section gives a description of the function. For easier description all input files are named `ifile` or `ifile1`, `ifile2`, etc. and all output files are named `ofile` or `ofile1`, `ofile2`, etc. Further the following notion is introduced:

$i(t)$	Record number t of `ifile`.
$i(t, x)$	Element number x of the field of record number t of `ifile`.
$i_j(t, x)$	Element number x of the field of record number t of `ifile<j>`.
$o(t)$	Record number t of `ofile`.
$o(t, x)$	Element number x of the field of record number t of `ofile`.
$o_j(t, x)$	Element number x of the field of record number t of `ofile<j>`.

4.1 Information

`shortinfo`

```
ifile
```

(“short information”) Prints short information of the records of `ifile`. That is the format, date, time, code, level, and size. If the field is complex, this is also printed. If there is information in dispo #1 or dispo #2, this is also printed.

For GRIB records the accuracy is not printed. This is due to the fact that this function avoids reading the field for speed reasons, but without reading the field it is not possible for the program to determine the accuracy of a GRIB record. To print the accuracy of a GRIB record, use function `formatinfo` (page 39) instead.

`info`

```
ifile
```

(“information”) Prints information of the records of `ifile`. That is the date, code, level, size, and the minimum, mean, and maximum value, and the number of missing values. The mean value is computed without the use of area weights!

`longinfo`

`ifile`

("long information") Prints long informations of the records of `ifile`. That is the complete header and the complete field.

`formatinfo`

`ifile`

("Format information") Identical to function `shortinfo` (page 38) with the difference that also the field is read. Thus this function is slowlier but can also print the accuracy of GRIB records.

`pipeinfo`

`ifile ofile`

("Pipe information") Prints the same than function `info` (page 38) and copies `ifile` to `ofile`. The sense is to get information of what is flowing through an pipe. For example the command

```
ext monmeans -daymins pipe1 pipe2 &
```

in the example on page 24 could be replaced by

```
ext pipeinfo -monmeans -daymins pipe1 pipe2 &
```

`pipeshortinfo`

`ifile ofile`

("Pipe short information") Prints the same than function `shortinfo` (page 38) and copies `ifile` to `ofile`. The sense is to get information of what is flowing through an pipe. For example the command

```
ext monmeans -daymins pipe1 pipe2 &
```

in the example on page 24 could be replaced by

```
ext pipeshortinfo -monmeans -daymins pipe1 pipe2 &
```

gribinfo

`ifile`

("GRIB information") Prints for GRIB records all information stored in the description part. Can be used to see entries for the used grid, the centre ID, etc.

griddes

`ifile`

("grid description") The grid description of the first record is printed. See Section 3.6 "Grid description files" on page 24 for details. Typically the standard output is redirected in a file, for example

```
ext griddes t21.grb > t21.grid.asc
```

grads

`ifile ofile`

("grads") Prints a description file of `ifile` on standard output for the plotting software `grads`. Typically the standard output is redirected in a file and the program is called as `pure4` or `pure8`, depending on what is the natural size of a float. Example:

```
pure4 grads file.grb file.grads > file.des
grads -l
```

After starting `grads`, the user can open the file by the command

```
open file.des
```

and can display for example code 167 by

```
display c167
```

nrec

`ifile`

("number of records") Prints the number of records of `ifile`.

`nyear`

`ifile`

(“number of years”) Prints the number of different years. This functions assumes that the records for the same year are immediately adjacent. See also function `showyear` (page 42).

`nmon`

`ifile`

(“number of months”) Prints the number of different combinations of years and months. This functions assumes that the records for the same year and month are immediately adjacent. See also function `showmon` (page 42).

`ndate`

`ifile`

(“number of dates”) Prints the number of different dates. This functions assumes that the records for the same date are immediately adjacent. See also function `showdate` (page 42).

`ntime`

`ifile`

(“number of times”) Prints the number of different combinations of date and time. This functions assumes that the records for the same date and time are immediately adjacent. See also function `showtime` (page 43).

`ncode`

`ifile`

(“number of codes”) Prints per date the number of different codes. This functions assumes that the records for the same date are immediately adjacent and that the records for the same date and code are ordered in such a way that all records of the same code are immediately adjacent. This function can well be used in conjunction with function `selfirstdate` (page 63) as

```
ext ncode -selfirstdate ifile
```

See also function `showcode` (page 43).

nlevel

`ifile`

("number of levels") Prints per date and code the number of levels. This functions assumes that the records for the same date and code are ordered in such a way that all records of the same code are immediately adjacent. This function can well be used in conjunction with function `selfirstcode` (page 65) as

```
ext nlevel -selfirstdate ifile
```

See also function `showlevel` (page 43).

showyear

`ifile`

("show year") Prints all different years. This functions assumes that the records for the same year are immediately adjacent.

showmon

`ifile`

("show month") Prints all different combinations of years and months. This functions assumes that the records for the same year and month are immediately adjacent.

showdate

`ifile`

("show date") Prints all different dates. This functions assumes that the records for the same date are immediately adjacent.

`showtime`

`ifile`

("show time") Prints all different combinations of dates and times. This functions assumes that the records for the same date and time are immediately adjacent.

`showcode`

`ifile`

("which codes") Prints per date all different codes. This functions assumes that the records for the same date are immediately adjacent and that the records for the same date and code are ordered in such a way that all records of the same code are immediately adjacent. This function can well be used in conjunction with function `selfirstdate` (page 63) as

```
ext showcode -selfirstdate ifile
```

`showlevel`

`ifile`

("show level") Prints per date and code all different levels. This functions assumes that the records for the same date and code are ordered in such a way that all records of the same code are immediately adjacent. This function can well be used in conjunction with function `selfirstcode` (page 65) as

```
ext showlevel -selfirstcode ifile
```

`countc`

`ifile ofile`

("count constant")

$$o(1, x) = \#\{i(t', x'), x' = x, i(t', x') \neq \text{miss} \wedge i(t', x') = c\}$$

countcr

`ifile ofile`

("count constant for each record")

$$o(t,1) = \#\{i(t',x'), t' = t, i(t',x') \neq \text{miss} \wedge i(t',x') = c\}$$

status

("status") Prints the signal number of the CONT-signal. This can be used for other functions which need very long time for computation to see what they are just doing. Let for example this signal number be 19. Then the unix command `kill -19 <process ID>` sends this signal to the process which status is to be asked. It does not kill it! (C-shell users can also stop this process by pressing `CONTROL-Z` and starting it again by the shell command `%`. This starting makes the C-shell sending the CONT-signal to the process).

4.2 Formatted input and output

input

`ofile`

("input ascii") This function reads ascii numbers from standard input and stores them as field elements of `ofile`. Read Section 3.8 "How to convert files into SIMPLE, EXTRA, SERVICE, LOLA, or GRIB?" on page 30 for details of use. The first input numbers concerning the headers should be given as advanced input as described in Section 3.3 "Advanced standard input" on page 21. The input of the fields should be redirected to a file.

inputsmp

`ofile`

("input ascii SIMPLE likely") This function reads ascii numbers from standard input which are in an SIMPLE likely sequence and stores them in `ofile`. Read

Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard input is redirected to a file.

The numbers that are read are exactly that ones which are written out by `outputsmp` (page 46). For that reason `inputsmp` could be understood as the opposite of `outputsmp`.

`inputext`

`ofile`

(“input ascii EXTRA likely”) This function reads ascii numbers from standard input which are in an EXTRA likely sequence and stores them in `ofile`. Read Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard input is redirected to a file.

The numbers that are read are exactly that ones which are written out by `outputext` (page 46). For that reason `inputext` could be understood as the opposite of `outputext`.

`inputsrv`

`ofile`

(“input ascii SERVICE likely”) This function reads ascii numbers from standard input which are in an SERVICE likely sequence and stores them in `ofile`. Read Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard input is redirected to a file.

The numbers that are read in are exactly that ones which are written out by `outputsrv` (page 47). For that reason `inputsrv` could be understood as the opposite of `outputsrv`.

`inputlola`

`ofile`

(“input ascii LOLA likely”) This function reads ascii numbers from standard input which are in a LOLA likely sequence and stores them in `ofile`. Read

Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard input is redirected to a file.

The numbers that are read in are exactly that ones which are written out by `outputlola` (page 47). For that reason `inputlola` could be understood as the opposite of `outputlola`.

`output`

`ifile`

(“output”) Prints all values to standard output.

`outputint`

`ifile`

(“output integer”) Prints all values rounded to the nearest integers to standard output, each record in one line.

`outputsmp`

`ifile`

(“output ascii SIMPLE likely”) This function writes all headers and fields of `ofile` as ascii numbers to standard output in an SIMPLE likely sequence. Read Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard output is redirected to a file.

The numbers that are written out are exactly that ones which are read in by `inputsmp` (page 44). For that reason `inputext` could be understood as the opposite of `outputext`.

`outputext`

`ifile`

(“output ascii EXTRA likely”) This function writes all headers and fields of `ofile` as ascii numbers to standard output in an EXTRA likely sequence. Read

Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard output is redirected to a file.

The numbers that are written out are exactly that ones which are read in by `inputext` (page 45). For that reason `inputext` could be understood as the opposite of `outputext`.

`outputsrv`

`ifile`

(“output ascii SERVICE likely”) This function writes all headers and fields of `ofile` as ascii numbers to standard output in an SERVICE likely sequence. Read Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard output is redirected to a file.

The numbers that are written out are exactly that ones which are read in by `inputsrv` (page 45). For that reason `inputsrv` could be understood as the opposite of `outputsrv`.

`outputlola`

`ifile`

(“output ascii LOLA likely”) This function writes all headers and fields of `ofile` as ascii numbers to standard output in a LOLA likely sequence. Read Section 3.9 “How to transfer files to other computers?” on page 32 for details of use. Normally the standard output is redirected to a file.

The numbers that are written out are exactly that ones which are read in by `inputlola` (page 45). For that reason `inputsrv` could be understood as the opposite of `outputsrv`.

4.3 Converting the format

`copy`

`ifile ofile`

(“copy”) Copies each record of `ifile` to `ofile`. This function is normally used to change the format of `ifile` to EXTRA or SERVICE. For example: To change the GRIB file `file.grb` to the EXTRA format, just type

```
ext copy file.grb file.ext
```

Remember: The name of the program, in this example it is “ext”, determines the output format. See also function `copy2` (page 48).

`copy2`

`ifile1 ifile2 ofile`

(“copy”) Copies each record of `ifile1` to `ofile`. The headers and fields of `ifile2` are ignored. This function is normally used to change a SIMPLE, EXTRA, SERVICE, or LOLA file to GRIB format. In this case the description parts of the GRIB records of `ifile2` are used for the description parts of the records of `ofile`. (The trick is that all GRIB records that are written by this program get the description part of the last read in GRIB record which are in this case from `ifile2`.) For example: Let us assume that the EXTRA file `file.ext` contains data belonging to a Gaussian T21 grid and that `any_t21_data_set.grb` is a GRIB file containing a T21 data set. To change now the EXTRA file `file.ext` to GRIB format containing the T21 grid description of `any_t21_data_set.grb`, the user has to type

```
grb2 copy2 file.ext any_t21_data_set.grb file.grb
```

Remember: The name of the program, in the example it is “grb2”, determines the output format. See also function `copy` (page 48).

If the program would be called as `grb` instead of `grb2`, the accuracy of the GRIB record of `file.grb` would be the greatest accuracy of both input files. This is normally not desired, so **CALL `grb2` INSTEAD OF `grb`**.

4.4 Generation of files

`weight0`

`ofile`

(“weight”) An area weight file is generated, see Section 3.7 “The concept of area weights” on page 27 for more information. The requested input is exactly what is written in a grid description file, see Section 3.6 “Grid description files” on page 24. The area weights are the sizes of the areas which are bounded by lines exactly between the neighboured grid points.

The rules for the most western points are: The western boundary line for the areas lies between the these most western points and the most eastern points, but the longitudinal distance to west is never greater than to the east.

For example: If the two western longitudes are 30˚E and 40˚E and the most eastern one is 100˚E, then the areas of grid points at longitude 40˚E are bordered by 35˚E in the West and 45˚E in the East, and the areas of the grid points at longitude 30˚E are bordered by lines at 25˚E in the East and 35˚E in the West. If the most eastern longitude would be 28˚E instead of 100˚E, then the areas at grid points at longitude 30˚E would be bordered by lines at 29˚E in the West. A corresponding rule holds for the most eastern points.

The rules for the most northern points are: The northern boundary line for the areas has the same latitudinal distance to the north than to the south, but it is never greater than 90˚N. A corresponding rule holds for the most southern points.

For example: If the three most northern latitudes are at 85˚N, 80˚N, and at 75˚N, then the areas of the grid points at latitude 80˚N are bordered by lines at 77.5˚N in the South and at 82.5˚N in the North, and the areas of the grid points at latitude 85˚N are bordered by lines at 82.5˚N in the South and 87.5˚N in the North. If there would be additional grid points at 90˚N, then the corresponding areas are bordered by lines at 87.5˚N in the South and 90˚N in the North.

weight1

`ifile ofile`

("weight") Writes the area weights of the first record of `ifile` to `ofile`. This function can be used to create a weight file, see Section 3.7 "The concept of area weights" on page 27 for more information.

null

`ofile`

("null") Writes one record into `ofile` which consists of a field of size 1 with 0 as element. If for example in a command the "filling up" of file `ifile` (See Section 2.4 "The "Filling up" of input files and "enlarging" of input records" on page 16) should be done with a null field and not with a copy of the last record, it could be typed
`-cat2 ifile -null` in the command line instead of `ifile`.

const

`ofile`

("constant") Write one record into `ofile` which consists of a field of size 1 with a user given number as element. Such a file behaves in formulas like a constant, imagine the example `ext div -const,100 ifile 100_div_by_ifile` where 100 is divided by `ifile`.

consts

`ofile`

("constant series") Writes in every record of ofile a field of size 1 with for every record another user given number as element. In the example `ext add ifile -consts ifile_plus_constants` the user can give for every record another constant which should be added to the fields of `ifile`.

pi

`ofile`

(“π”) Write one record into `ofile` which consists of a field of size 1 with the mathematical constant π as element. Such a file behaves in formulas like a constant, imagine the example `ext sin -divc,180 -mul -pi ifile ofile`.

e

`ofile`

(“e”) Write one record into `ofile` which consists of a field of size 1 with the mathematical constant e as element. Such a file behaves in formulas like a constant, imagine the example `ext mul ifile -e e_times_ifile` where `ifile` is multiplied by e.

for

`ofile`

(“for number = a to b step c”) For generation of records with field size 1 and field elements beginning with a start value in record 1 which is increased from one record to the next. For example the command

```
ext info -for,1,2,0.25
```

writes to standard output

```
  REC :    DATE COD LEVEL      SIZE      MINIMUM       MEAN    MAXIMUM :   MISS
    1 :       0   0     0         1 :  1.000e+00  1.000e+00  1.000e+00 :      0
    2 :       0   0     0         1 :  1.250e+00  1.250e+00  1.250e+00 :      0
    3 :       0   0     0         1 :  1.500e+00  1.500e+00  1.500e+00 :      0
    4 :       0   0     0         1 :  1.750e+00  1.750e+00  1.750e+00 :      0
    5 :       0   0     0         1 :  2.000e+00  2.000e+00  2.000e+00 :      0
```

This function can be used for the creation of tables, try for fun

```
ext output -for,-5,5,0.1 > gauss_x.asc
ext output -div -exp -mulc,-0.5 -sqr -gensteps,101,-5,0.1 -sqrt -mulc,2 -pi \
  > gauss_y.asc
paste gauss_x.asc gauss_y.asc > gauss.asc
xvgr gauss.asc
```

`random`

`ofile`

("random") Generates a file of rectangularly distributed random numbers in the interval [0,1). (The internal start value of the random generator depends on the system time and the process identity number.)

`randomnormal`

`ofile`

("random normally distributed") Generates a file of normally distributed random numbers. (The internal start value of the random generator depends on the system time and the process identity number.)

4.5 Manipulating the header

`chdate`

`ifile ofile`

("change date") Changes the date. A start date and a date step must be given in the YYYYMMDD format. The dates of the records of `ifile` is changed into a new date, which is at the beginning the start date and which is increased by a date step whenever a record has the same time, code, and level as the first record.

`chtime`

`ifile ofile`

("change time") Changes the date and time. A start date and date step and a start time and time step must be given. The dates must be given in the YYYYMMDD format, the times in the HHMM format. The dates and times of the records of `ifile` are changed into a new date and time, which is at the beginning the start date and start time and which is increased by a date step and time step whenever a record has the same code and level as the first record.

chyear

```
ifile ofile
```

("change year") Changes the year in every record of `ifile` to the same given value.

chmon

```
ifile ofile
```

("change month") Changes the month in every record of `ifile` to the same given value.

chday

```
ifile ofile
```

("change day") Changes the day in every record of `ifile` to the same given value.

chcode

```
ifile ofile
```

("change code") Changes the code in every record of `ifile` to the same given value.

chcodes

```
ifile ofile
```

("change codes") Changes some user given codes of `ifile` to new user given values. First the user has to give the number of different codes to be changed. Afterwards he has to type in pairs of old and new codes.

chlevel

```
ifile ofile
```

("change level") Changes the level in every record of `ifile`. The user gives a list of levels which is used cyclically to change the levels of the records. This list is read from the beginning every time when a new code or a new date is read from `ifile`. If this is not desired, the dates can be set to 0 by using `chdate,0,0` (`chdate` (page 52)).

`chdispo1`

```
ifile ofile
```

("change dispo #1") Changes the dispo #1 entry in every record of `ifile`. Because only SERVICE and LOLA format have a dispo #1 entry, this function is only useful if the output format is SERVICE or LOLA. This value can be used to store a centre ID or what ever the user wants. The functions `longinfo` (page 39) and `shortinfo` (page 38) shows the dispo entries if they are different to zero.

`chdispo2`

```
ifile ofile
```

("change dispo #2") Changes the dispo #2 entry in every record of `ifile`. Because only SERVICE and LOLA format have a dispo #2 entry, this function is only useful if the output format is SERVICE or LOLA. This value can be used to store an experiment number or what ever the user wants. The functions `longinfo` (page 39) and `shortinfo` (page 38) shows the dispo entries if they are different to zero.

4.6 Manipulating the field

`chsize`

```
ifile ofile
```

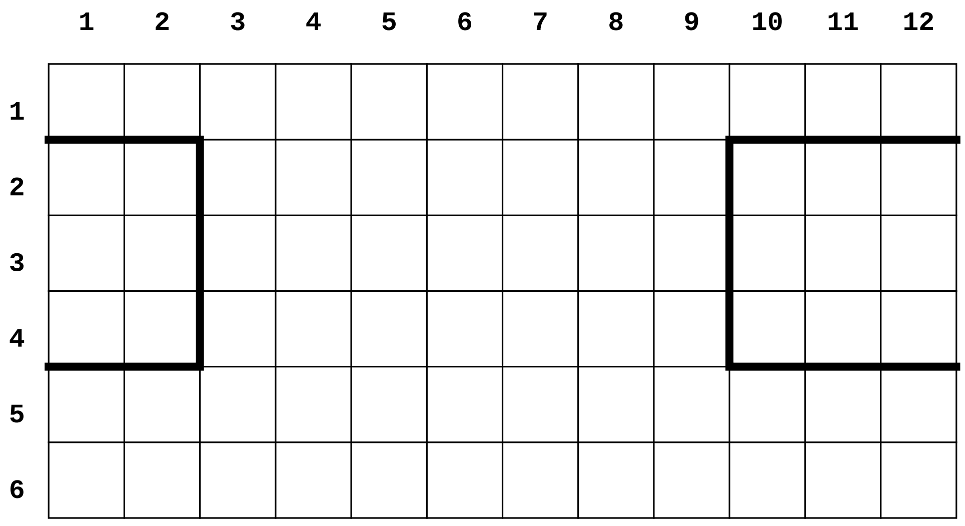
("change size") Changes the size of the field. If the new size is smaller than the original one, then the field is cut. If the new size is greater than the original one, then the field is filled up with missing values. See also function `enlarge` (page 58).

maskindexbox

`ifile ofile`

("mask index box") Masks a box of the rectangularly understood fields, i.e. the elements inside the box are untouched, the elements outside are set to the missing value. The field size of the records of `ofile` is therefore the same than that of `ifile`. The user has to give the indexes of the edges of the box. The index of the left edge may be greater than that of the right edge. See also function `selindexbox` (page 66).

The next figure demonstrate the numbering: To mask the bold marked box, the user has to type in 10 for index as the left and 2 for the index as the right column and 2 as the index of lower and 4 as the index of the upper row. (The numbers at the top are the indexes of the columns, that on the left handed side are that of the rows.)



masklonlatbox

`ifile ofile`

("mask longitude/latitude box") Masks a box of the rectangularly understood fields, i.e. the elements inside the box are untouched, the elements outside are set to the missing value. The user has to give the longitudes and latitudes of the edges of the box. The field size of the records of `ofile` is therefore the same than that of `ifile`. See also function `sellonlatbox` (page 67).

normalize

```
ifile ofile
```

(“normalize”)

$$o(t, x) = \frac{i(t, x)}{\sum_{x',\, i(t, x') \neq \mathrm{miss}} i(t, x')}$$

shiftleft

```
ifile ofile
```

(“shift left”) The rectangularly understood fields are cyclically shifted to the left.

shiftright

```
ifile ofile
```

(“shift right”) The rectangularly understood fields are cyclically shifted to the right.

swapleftright

```
ifile ofile
```

(“swap left with right”) The rectangularly understood fields are left right swapped, t.i. the first column is swapped with the last one, the second is swapped with the second last, etc.

swaprightleft

```
ifile ofile
```

(“sap right with left”) The same as `swapleftright` (page 56).

swaptopbottom

`ifile ofile`

("swap top with bottom") The rectangularly understood fields are top bottom swapped, t.i. the first row is swapped with the last one, the second is swapped with the second last, etc.

swapbottomtop

`ifile ofile`

("swap bottom with top") The same as `swaptopbottom` (page 57).

break

`ifile ofile`

("break") Breaks every record of `ifile` in a user given number of pieces of equal field size which are stored adjacent in `ofile`. This could be useful for two reasons: If otherwise computations could not be done for storage reasons or a field should be divided into different levels. `break` is the opposite of `melt` (page 57).

break<n>

`ifile ofile1 ... ofile<n>`

("break") Breaks every record of `ifile` in <n> pieces of equal field size which are stored in `ofile1` ... `ofile<n>`. This could be useful for two reasons: If otherwise computations could not be done for storage reasons or a field should be divided into different levels. `break<n>` is the opposite of `melt<n>` (page 58).

melt

`ifile ofile`

("melt") Melts the fields of every sequence of a user given length together to one record, t.i. if the length of the sequence is L, the field of the first record of `ofile` is melted of the fields of record 1 to L of `ifile`, the second record of `ofile` is melted of the fields of record L+1 to 2L of `ifile`, etc. `melt` is the opposite of `break`.

meltall

    ifile ofile

("melt all") Melts the fields of all records of `ifile` together to one field which is stored in `ofile`.

melt<n>

    ifile1 ... ifile<n> ofile

("melt") The field of record R of `ofile` is melted by the fields of record R of `ifile1` ... `ifile<n>`. `melt<n>` is the opposite of `break<n>`. If the records have different sizes, use the function `chsize` (page 54).

enlarge

    ifile ofile

("enlarge") Enlarge the field of `ifile` by a user given factor. If the field size is n and this factor is K then it is

$$o(t,x) = o(t,x+n) = o(t,x+2n) = \dots = o(t,x+Kn) = i(t,x)$$

thinout

    ifile ofile

("thin out") Thins out `ifile` by missing out a user given number of records while copying `ifile` to `ofile`. If only every s-th record should be taken, then it is

$$o(t,x) = i(1+(t-1)s, x)$$

thinoutr

    ifile ofile

("thin out for each record") Thins out the rectangularly understood fields of `ifile`. A user gives the number of longitudes and latitudes to be skipped while copying `ifile` to `ofile`. If the number of longitudes of `ifile` is n_{lon} and the number of latitudes is n_{lat} and if only every s_{lon}-th longitude and only every s_{lat}-th latitude should be taken, then the number of longitudes of `ofile` is

$$m_{lon} = \lfloor 1 + (n_{lon} - 1) / s_{lon} \rfloor$$

that of latitudes is

$$m_{lat} = \lfloor 1 + (n_{lat} - 1) / s_{lat} \rfloor$$

and it is

$$o(t, (x_{lon}, x_{lat})) = i(t, (1 + (x_{lon} - 1) s_{lon}, 1 + (x_{lat} - 1) s_{lat}))$$

4.7 Manipulating the sequence of records

`reverse`

```
ifile ofile
```

("reverse") Reverses the sequence of the records in `ifile`. If N is the number of records of `ifile`, then it is

$$o(t, x) = i(N - t + 1, x)$$

`reverser`

```
ifile ofile
```

("reverse for each record") Reverses the sequence of field elements in `ifile`. If $N(t)$ is the number of field elements of record number t of `ifile`, then it is

$$o(t, x) = i(t, N(t) - x + 1)$$

`transpose`

```
ifile ofile
```

("transpose") `ifile` is understood as a matrix with each record as a row. This matrix is transposed:

$o(t, x) = i(x, t)$

`transposer`

```
ifile ofile
```

("transpose for each record") Each record of `ifile` is understood as a matrix. These matrixes are transposed: If $n_{lon}(t)$ is the number of longitudes and $n_{lat}(t)$ is the number of latitudes of record t, then it is

$o(t, i_{lon} + n_{lon}(t)\ (i_{lat} - 1)) = o(t, i_{lat} + n_{lat}(t)\ (i_{lon} - 1))$

`split<n>`

```
ifile ofile1 ... ofile<n>
```

("split") Splits `ifile` into the <n> output files record by record. The first record of `ifile` becomes the first record of `ofile1`, the second record of `ifile` becomes the first record of `ofile2`, ... , and the <n>th record of `ifile` becomes the first record of `ofile<n>`. The (<n>+1)st record of `ifile` becomes the second record of `ofile1`, the (<n>+2)nd record of `ifile` becomes the second record of `ofile2`, ... , and the (2<n>)th record of `ifile` becomes the second record of `ofile<n>`, the (2<n>+1)st record of `ifile` becomes the third record of `ofile1` etc. `split<n>` is the opposite of `merge<n>`.

`merge<n>`

```
ifile1 ... ifile<n> ofile
```

("merge") Merges the <n> input files into ofile record by record. The first record of `ofile` was the first record of `ifile1`, the second record of `ofile` was the first record of `ifile2`, ... , and the <n>th record of `ofile` was the first record of `ifile<n>`. The (<n>+1)st record of `ofile` was the second record of `ifile1`, the (<n>+2)nd record of `ofile` was the second record of `ifile2`, ... , and the (2<n>)th record of `ofile` was the second record of `ifile<n>`, the (2<n>+1)st record of `ofile` was the third record of `ifile1` etc. `merge<n>` is the opposite of `split<n>`. If the records have different sizes, use the function `chsize` (page 54).

`mergedate2`

```
ifile1 ifile2 ofile
```

(“merge sorted by dates”) Merges the records of `ifile1` and `ifile2` sorted by date, t.i. every record of `ifile1` and every record of `ifile2` is in ofile, and all records in `ofile` are sorted by date. THIS FUNCTION ASSUMES THAT THE RECORDS ARE SORTED BY DATE!

`replace`

```
ifile1 ifile2 ofile
```

(“replace”) This function can be used to replace some records of `ifile1` by records of `ifile2`. The records of `ifile1` are copied to `ofile` as long as the header of the record does not mach the header of the first record of `ifile2`. If they match, the first record of `ifile2` is copied to `ofile` and the record of `ifile1` is skipped. Afterwards again the records of `ifile1` are copied as long as the header of the record does not mach the header of the second record of `ifile2`. Afterwards the same proceeding is done with the third record of `ifile2`, etc. If there are no records left in `ifile2` then all the remaining records of `ifile1` are copied to `ofile`.

`cat<n>`

```
ifile1 ... ifile<n>
```

(“concaternate”) Concaternates the contents of the files, similar to the unix command `cat`. The sense of using this function instead of the unix command `cat` is to let it write the protocol file (See Section 3.1 “Protocol file” on page 18) and to start it as a child process.

4.8 Selection

`sel`

```
ifile ofile
```

("select") For selection of records. This function is good for stepping through `ifile` and deciding manually for each record if it should be copied to `ofile` or not.

Some information is displayed about the first record. The user has now to decide, weather this and the following records should be copied to `ofile` or skipped. The user has to type in two numbers. The first gives a selection and the second is 0 for selection should not be copied to `ofile` or a number not equal to 0, for example 1, if it should be copied to `ofile`. The first number, which defines the selection, could positive, zero, or negative. A positive number `<`*n*`>` is a relative selection, t.i. the current record and the following `<`*n*`>`-1 records are selected. A zero selects all records from the current one to the last one. A negative number `-<`*n*`>` is an absolute selection. It selects all records from the current one to the record number `<`*n*`>`. The selection is then copied or skipped, depending on the second given number, and if not the end of `ifile` is reached, again information about the current record is displayed and again the user have to select, weather this and the following records should be copied to `ofile` or skipped.

`selrec`

```
ifile ofile
```

("select record") Selects all records of `ifile` with the record number in a given range and copies them to `ofile`. If 0 is given as the last record number, then all records from the start record to the end of `ifile` are copied.

`selfirstrec`

```
ifile ofile
```

("select first record") Selects the first record of `ifile` and copies it to `ofile`.

selfirstmidlastrec

`ifile ofile1 ofile2 ofile3`

("select first middle last record") The first record of `ifile` is copied to `ofile1`, the last record of `ifile` is copied to `ofile3`, and all other records of `ifile`, t.i. from then second record to the second last record, are copied to `ofile2`. This function is thought to be used in connection with function `seasmeans` and `seasavgs`.

seldate

`ifile ofile`

("select date") Selects all records of `ifile` with the date in a given range and copies them to `ofile`.

selfirstdate

`ifile ofile`

("select first date") Selects all records of `ifile` until a record is found with another date than the first record. This function can well be used in conjunction with function `ncode` (page 41) or function `showcode` (page 43) as

```
ext ncode -selfirstdate ifile
```

respectively as

```
ext showcode -selfirstdate ifile
```

selfirsttime

`ifile ofile`

("select first time") Selects all records of `ifile` until a record is found with another combination of date and time than the first record. This function can well be used in conjunction with function `ncode` (page 41) or function `showcode` (page 43) as

```
ext ncode -selfirsttime ifile
```

respectively as

```
ext showcode -selfirsttime ifile
```

selyear

```
ifile ofile
```

("select year") Selects all records of `ifile` with the year in a given range and copies them to `ofile`.

selmon

```
ifile ofile
```

("select month") Selects all records of `ifile` with the month in a given range and copies them to `ofile`.

selseas

```
ifile ofile
```

("select season") Selects all records of `ifile` with a given season and copies them to `ofile`.

selday

```
ifile ofile
```

("select day") Selects all records of `ifile` with the day in a given range and copies them to `ofile`.

selcode

```
ifile ofile
```

("select code") Selects all records of `ifile` with a user given code and copies them to `ofile`. For selection of different codes simultaneously, see function `selcode<n>` (page 65).

selfirstcode

`ifile ofile`

("select first date") Selects all records of `ifile` until a record is found with another code or another date then the first record. This function can well be used in conjunction with function `nlevel` (page 42) or function `showlevel` (page 43) as

```
ext nlevel -selfirstcode ifile
```

respectively as

```
ext showlevel -selfirstcode ifile
```

selcode<n>

`ifile ofile1 ... ofile<n>`

("select code") For each output file the user must give a code. For every output file this function then selects all records of `ifile` with this code and copies them into this output file.

sellevel

`ifile ofile`

("select level") Selects all records of `ifile` with the level in a given range and copies them to `ofile`.

seldispo1

`ifile ofile`

("select dispo #1") Selects all records of `ifile` with a user given dispo #1 entry and copies them to `ofile`. Because only SERVICE and LOLA format have a dispo #1 entry, this function is only useful if the input format is SERVICE or LOLA. The functions `longinfo` (page 39) and `shortinfo` (page 38) shows the dispo entries if they are different to zero.

seldispo2

`ifile ofile`

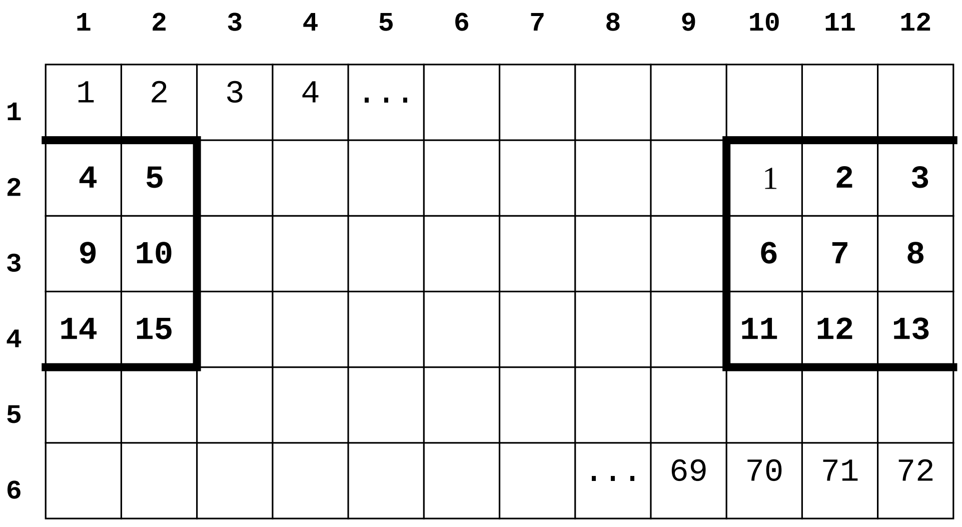
("select dispo #1") Selects all records of `ifile` with a user given dispo #2 entry and copies them to `ofile`. Because only SERVICE and LOLA format have a dispo #2 entry, this function is only useful if the input format is SERVICE or LOLA. The functions `longinfo` (page 39) and `shortinfo` (page 38) shows the dispo entries if they are different to zero.

selindexbox

`ifile ofile`

("select index box") Selects a box of the rectangularly understood fields. The field size of the records of `ofile` is therefore generally smaller than that of `ifile`. The user has to give the indexes of the edges of the box. (Smallest index is 1.) The index of the left edge may be greater than that of the right edge.

The next figure demonstrate the numbering: To select the bold marked box, the user has to type in 10 for index as the left and 2 for the index as the right column and 2 as the index of lower and 4 as the index of the upper row. In `ifile` the field elements are numbered as indicated by the thin numbers, in `ofile` as indicated by the thick numbers. (The numbers at the top are the indexes of the columns, that on the left handed side are that of the rows.)



(If `ofile` should be in GRIB format, better use function `maskindexbox` (page 55), which preserves the grib information, which is necessary for computing the area weights. The GRIB formatted output file of `maskindexbox` is only slightly greater than that of `selindexbox`.)

sellonlatbox

```
ifile ofile
```

("select longitude/latitude box") Selects a box of the rectangularly understood fields. The field size of the records of `ofile` is therefore generally smaller than that of `ifile`. The user has to give the longitudes and latitudes of the edges of the box. It is a good idea to choose LOLA as the output format.

(If `ofile` should be in GRIB format, better use function `masklonlatbox` (page 55), which preserves the grib information, which is necessary for computing the area weights. The GRIB formatted output file of `masklonlatbox` is only slightly greater than that of `sellonlatbox`.)

selfile<m>of<n>

```
ifile1 ... ifile<n> ofile
```

("select file") `ifile<m>` is copied to `ofile`. See Section 3.4 "Not required output files" on page 22 for the sense of it.

selfile<l>and<m>of<n>

```
ifile1 ... ifile<n> ofile1 ofile2
```

("select file") `ifile<l>` is copied to `ofile1` and `ifile<m>` is copied to `ofile2`. See Section 3.4 "Not required output files" on page 22 for the sense of it.

`selfile<k>and<l>and<m>of<n>`

```
ifile1 ... ifile<n> ofile1 ofile2 ofile3
```

("select file") `ifile<k>` is copied to `ofile1` and `ifile<l>` is copied to `ofile2` and `ifile<m>` is copied to `ofile3`. See Section 3.4 "Not required output files" on page 22 for the sense of it.

4.9 Missing values

`setctomiss`

```
ifile ofile
```

("set constant to missing value")

$$o(t,x) = \begin{cases} i(t,x) & \text{if } i(t,x) \neq c \\ \text{miss} & \text{if } i(t,x) = c \end{cases}$$

`setmiss`

```
ifile1 ifile2 ofile
```

("set missing value")

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) = \text{miss} \\ i_1(t,x) & \text{if } i_1(t,x) \neq \text{miss} \end{cases}$$

`setmissc`

```
ifile ofile
```

("set missing value by constant")

$$o(t,x) = \begin{cases} c & \text{if } i_1(t,x) = \text{miss} \\ i_1(t,x) & \text{if } i_1(t,x) \neq \text{miss} \end{cases}$$

setnotmiss

```
ifile1 ifile2 ofile
```

(“set not missing value”)

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \end{cases}$$

setnotmissc

```
ifile ofile
```

(“set not missing value by constant”)

$$o(t,x) = \begin{cases} c & \text{if } i_1(t,x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \end{cases}$$

countmiss

```
ifile ofile
```

(“count missing values”)

$$o(1,x) = \#\{i(t',x'), x' = x, i(t',x') = \text{miss}\}$$

countmissr

```
ifile ofile
```

(“count missing values for each record”)

$$o(t,1) = \#\{i(t',x'), t' = t, i(t',x') = \text{miss}\}$$

countnotmiss

```
ifile ofile
```

(“count not missing values”)

$$o(1, x) = \#\{i(t', x'), x' = x, i(t', x') \neq \text{miss}\}$$

countnotmissr

`ifile ofile`

(“count not missing values for each record”)

$$o(t, 1) = \#\{i(t', x'), t' = t, i(t', x') \neq \text{miss}\}$$

packr

`ifile1 ifile2 ofile`

(“pack for each record”) Every field of `ofile` is built only of that elements of `ifile1` where the corresponding element in the field of `ifile2` is not the missing value. It is possible to think of `ifile2` as a mask. If for example the fields of `ifile1` are only defined on land points with missing values at sea and only at sea, `ifile1` could be packed by typing `ext packr ifile ifile ofile`. If the field in `ifile` has also missing values at land, then another file should be used as the second argument, for example the land weight file, which has missing values at and only at sea points.

unpackr

`ifile1 ifile2 ofile`

(“unpack for each record”) The field of `ifile2` is scanned. Whenever a not missing value is found, an element of `ifile1` is stored in the field of `ofile`. The elements of the fields of `ifile1` are read one after another. Maybe not all of them are used. It is possible to think of `ifile2` as a mask. If `packed_ifile` was built by the command

```
ext packr ifile ifile packed_ifile
```

then `ofile`, constructed by the command

```
ext size ifile | ext unpackr -chsize -packed_ifile ifile ofile
```

is equal to `ifile`.

4.10 Sorting and ranking

`sort`

```
ifile ofile
```

("sort") Sorts for every field position the elements in ascending order. Missing values are put to the end. After sorting it is

$$o(t_1, x) < o(t_2, x) \qquad \forall (t_1 < t_2), x$$

`sortr`

```
ifile ofile
```

(sort for each record) Sorts the elements of each record in ascending order. Missing values are put to the end. After sorting it is

$$o(t, x_1) < o(t, x_2) \qquad \forall t, \ (x_1 < x_2)$$

`rank`

```
ifile ofile
```

("rank") $o(t, x)$ is the rank of $i(t, x)$ in $\{i(t', x'), x' = x\}$. Equal elements gets the same rank. For example: The ranks of $(4.4, 5.1, 5.6, 2.3, 5.1, 7.8)$ are $(2, 3.5, 5, 1, 3.5, 6)$, because 2.3 is the greatest number, 4.4 the 2nd greatest, 5.1 the 3rd and 4th greatest number, etc.

`rankr`

```
ifile ofile
```

("rank for each record") $o(t, x)$ is the rank of $i(t, x)$ in $\{i(t', x'), t' = t\}$. Equal elements gets the same rank. For example: The ranks of $(4.4, 5.1, 5.6, 2.3, 5.1, 7.8)$ are $(2, 3.5, 5, 1, 3.5, 6)$, because 2.3 is the greatest number, 4.4 the 2nd greatest, 5.1 the 3rd and 4th greatest number, etc.

4.11 Arithmetic

sum

```
ifile ofile
```

(“sum”)

$$o(1, x) = \sum_{t} i(t, x)$$

sum<n>

```
ifile1 ... ifile<n> ofile
```

(“sum”)

$$o(t, x) = \sum_{j=1}^{n} i_j(t, x)$$

sumr

```
ifile ofile
```

(“sum for each record”)

$$o(t, 1) = \sum_{x} i(t, x)$$

add

```
ifile1 ifile2 ofile
```

(“add”). Identical with `sum2` (`sum<n>` (page 72)).

$$o(t, x) = i_1(t, x) + i_2(t, x)$$

addc

```
ifile ofile
```

(“add constant”)

$o(t, x) = i(t, x) + c$

addcc

```
ifile ofile
```

("add complex constant")

$o(t, x) = i(t, x) + c$

sub

```
ifile1 ifile2 ofile
```

("subtract").

$o(t, x) = i_1(t, x) - i_2(t, x)$

subc

```
ifile ofile
```

("subtract constant")

$o(t, x) = i(t, x) - c$

subfromc

```
ifile ofile
```

("subtract from constant")

$o(t, x) = c - i(t, x)$

subcc

```
ifile ofile
```

("subtract complex constant")

$o(t, x) = i(t, x) - c$

subfromcc

```
ifile ofile
```

("subtract from complex constant")

$o(t,x) = i(t,x) - c$

minus

```
ifile ofile
```

("minus")

$o(t,x) = -i(t,x)$

mul

```
ifile1 ifile2 ofile
```

("multiply")

$o(t,x) = i_1(t,x) \cdot i_2(t,x)$

mulc

```
ifile ofile
```

("multiply by constant")

$o(t,x) = i(t,x) \cdot c$

mulcc

```
ifile ofile
```

("multiply by complex constant")

$o(t,x) = i(t,x) \cdot c$

div

```
ifile1 ifile2 ofile
```

("divide")

$$o(t,x) = \frac{i_1(t,x)}{i_2(t,x)}$$

divc

```
ifile ofile
```

("divide by constant")

$$o(t,x) = \frac{i(t,x)}{c}$$

divcc

```
ifile ofile
```

("divide by complex constant")

$$o(t,x) = \frac{i(t,x)}{c}$$

inverse

```
ifile ofile
```

("inverse")

$$o(t,x) = \frac{1}{i(t,x)}$$

mod

```
ifile1 ifile2 ofile
```

("modulus")

$$o(t,x) = \begin{cases} \left\lfloor \dfrac{i_1(t,x)}{i_2(t,x)} \right\rfloor i_2(t,x) & \text{if } \begin{array}{l} i_1(t,x) \neq \text{miss} \wedge \\ i_2(t,x) \neq \text{miss} \wedge i_2(t,x) \neq 0 \end{array} \\ \text{miss} & \text{if } \begin{array}{l} i_1(t,x) = \text{miss} \vee \\ i_2(t,x) = \text{miss} \vee i_2(t,x) \neq 0 \end{array} \end{cases}$$

$i_2(t,x)$ has not to be an integer number.

modc

```
ifile ofile
```

("modulus constant")

$$o(t,x) = \begin{cases} \left\lfloor \dfrac{i_1(t,x)}{c} \right\rfloor c & \text{if } i(t,x) \neq \text{miss} \wedge c \neq \text{miss} \wedge c \neq 0 \\ \text{miss} & \text{if } i(t,x) = \text{miss} \vee c = 0 \vee c = 0 \end{cases}$$

c has not to be an integer number.

power

```
ifile1 ifile2 ofile
```

("power")

$$o(t,x) = \begin{cases} (i_1(t,x))^{i_2(t,x)} & \text{if } i_1(t,x) > 0 \wedge i_2(t,x) \geq 0 \\ \text{miss} & \text{if } i_1(t,x) \leq 0 \vee i_2(t,x) < 0 \end{cases}$$

4.12 Maximum and Minimum

max

```
ifile ofile
```

("maximum")

$$o(1,x) = \max\{i(t',x'), x' = x\}$$

max<n>

```
ifile1... ifile<n> ofile
```

(“maximum”)

$$o(t, x) = \max\{i_1(t, x), \ldots, i_n(t, x)\}$$

maxr

```
ifile ofile
```

(“maximum for each record”)

$$o(t, 1) = \max\{i(t', x'), t' = t\}$$

maxabsdiffr

```
ifile1 ifile2 ofile
```

(“maximum of absolute differences for each record”)

$$o(t, 1) = \max\{\left|i_1(t', x') - i_2(t', x')\right|, t' = t\}$$

This function can be used for comparison of the records of `ifile1` and `ifile2`.

yearmaxs

```
ifile ofile
```

(“yearly maximum series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year it is

$$o(t, x) = \max\{i(t', x), t_1 < t' \le t_n\}$$

monmaxs

```
ifile ofile
```

("monthly maximum series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and month it is

$$o(t, x) = \max\{i(t', x), t_1 < t' \le t_n\}$$

daymaxs

```
ifile ofile
```

("daily maximum series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and day it is

$$o(t, x) = \max\{i(t', x), t_1 < t' \le t_n\}$$

min

```
ifile ofile
```

("minimum")

$$o(1, x) = \min\{i(t', x'), x' = x\}$$

min<n>

```
ifile1... ifile<n> ofile
```

("minimum")

$$o(t, x) = \min\{i_1(t, x), \ldots, i_n(t, x)\}$$

`minr`

`ifile ofile`

(“minimum for each record”)

$$o(t, 1) = \min\{i(t', x'), t' = t\}$$

`yearmins`

`ifile ofile`

(“yearly minimum series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year it is

$$o(t, x) = \min\{i(t', x), t_1 < t' \le t_n\}$$

`monmins`

`ifile ofile`

(“monthly minimum series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and month it is

$$o(t, x) = \min\{i(t', x), t_1 < t' \le t_n\}$$

`daymins`

`ifile ofile`

(“daily minimum series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and day it is

$$o(t, x) = \min\{i(t', x), t_1 < t' \le t_n\}$$

4.13 Mathematical functions

`sign`

```
ifile ofile
```

("sign")

$$o(t,x) = \begin{cases} 1 & \text{if } i(t,x) > 0 \\ 0 & \text{if } i(t,x) = 0 \\ -1 & \text{if } i(t,x) < 0 \\ \text{miss} & \text{if } i(t,x) = \text{miss} \end{cases}$$

`exp`

```
ifile ofile
```

("exp")

$$o(t,x) = e^{i(t,x)}$$

`log`

```
ifile ofile
```

("log")

$$o(t,x) = \log(i(t,x))$$

`log10`

```
ifile ofile
```

("log base 10")

$$o(t,x) = \log_{10}(i(t,x))$$

`sin`

`ifile ofile`

(“sin”)

$o\,(t, x) \;=\; \mathrm{sin}\,(i\,(t, x)\,)$

`cos`

`ifile ofile`

(“cos”)

$o\,(t, x) \;=\; \mathrm{cos}\,(i\,(t, x)\,)$

`tan`

`ifile ofile`

(“tan”)

$o\,(t, x) \;=\; \mathrm{tan}\,(i\,(t, x)\,)$

`asin`

`ifile ofile`

(“asin”)

$o\,(t, x) \;=\; \mathrm{asin}\,(i\,(t, x)\,)$

`acos`

`ifile ofile`

(“acos”)

$o\,(t, x) \;=\; \mathrm{acos}\,(i\,(t, x)\,)$

atan

```
ifile ofile
```

(“atan”)

$o(t, x) = \text{atan}(i(t, x))$

atan2

```
ifile1 ifile2 ofile
```

(“atan2”)

$o(t, x) = \text{atan2}(i_1(t, x), i_2(t, x))$

with $\text{atan2}(0, 0) = 0$ or $\text{atan2}(0, 0) = \pm\pi$

conj

```
ifile ofile
```

(“complex conjugate”)

$o(t, x) = i^*(t, x)$

re

```
ifile ofile
```

(“real part”)

$o(t, x) = \text{Re}(i(t, x))$

im

```
ifile ofile
```

(“imaginary part”)

$o(t, x) = \text{Im}(i(t, x))$

arg

```
ifile ofile
```

("argument") Computes the argument or phase of the complex numbers.

$$o(t,x) = \arg(i(t,x))$$

retocomplex

```
ifile ofile
```

("real to complex")

$$o(t,x) = i(t,x)$$

imtocomplex

```
ifile ofile
```

("imaginary to complex")

$$o(t,x) = i \cdot i(t,x)$$

recttocomplex

```
ifile1 ifile2 ofile
```

("rectangular to complex")

$$o(t,x) = i(t,x) + i \cdot i_2(t,x)$$

complextorect

```
ifile ofile1 ofile2
```

("complex to rectangular")

$$o_1(t,x) = \mathrm{Re}(i(t,x))$$
$$o_2(t,x) = \mathrm{Im}(i(t,x))$$

`poltocomplex`

    ifile1 ifile2 ofile

(“polar to complex”)

$$o(t,x) = i_1(t,x)\, e^{i \cdot i_2(t,x)}$$

`complextopol`

    ifile ofile1 ofile2

(“complex to polar”)

$$o_1(t,x) = |i(t,x)|$$
$$o_2(t,x) = \arg(i(t,x))$$

4.14 Comparisons and Logic

A value not equal to zero is treated as “true”, zero is treated as “false”.

`eq`

    ifile1 ifile2 ofile

(“equal”)

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) = i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \neq i_2(t,x) \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

`eqc`

    ifile ofile

(“equal constant”)

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) = c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \neq c \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

neq

```
ifile1 ifile2 ofile
```

("not equal")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \neq i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) = i_2(t,x) \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

nec

```
ifile ofile
```

("not equal constant")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \neq c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) = c \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

le

```
ifile1 ifile2 ofile
```

("lower equal")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \leq i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) > i_2(t,x) \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

lec

```
ifile ofile
```

("lower equal constant")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \leq c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) > c \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

lt

```
ifile1 ifile2 ofile
```

("lower than")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) < i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \geq i_2(t,x) \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

ltc

```
ifile ofile
```

("lower than constant")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) < c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \geq c \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

ge

```
ifile1 ifile2 ofile
```

("greater equal")

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \geq i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) < i_2(t,x) \\ \text{miss} & \text{if } \quad i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

gec

```
ifile ofile
```

(“equal constant”)

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \geq c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) < c \\ \text{miss} & \text{if } \quad i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

gt

```
ifile1 ifile2 ofile
```

(“greater than”)

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) > i_2(t,x) \\ 0 & \text{if } i_1(t,x), i_2(t,x) \neq \text{miss} \wedge i_1(t,x) \leq i_2(t,x) \\ \text{miss} & \text{if } \quad i_1(t,x) = \text{miss} \vee i_2(t,x) = \text{miss} \end{cases}$$

gtc

```
ifile ofile
```

(“greater than constant”)

$$o(t,x) = \begin{cases} 1 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) > c \\ 0 & \text{if } i_1(t,x), c \neq \text{miss} \wedge i_1(t,x) \leq c \\ \text{miss} & \text{if } \quad i_1(t,x) = \text{miss} \vee c = \text{miss} \end{cases}$$

and

`ifile1 ifile2 ofile`

("and")

$$o(t,x) = \begin{cases} 0 \quad \text{if} & i_1(t,x) = 0 \vee i_2(t,x) = 0 \\ 1 \quad \text{if} & \begin{cases} i_1(t,x) \neq 0 \wedge i_1(t,x) \neq \text{miss} \\ \wedge\, i_2(t,x) \neq 0 \wedge i_2(t,x) \neq \text{miss} \end{cases} \\ \text{miss if} & \begin{cases} i_1(t,x) \neq 0 \wedge i_2(t,x) = \text{miss} \\ \vee\, i_1(t,x) = \text{miss} \wedge i_2(t,x) \neq 0 \end{cases} \end{cases}$$

The following table, which must be read as described in Section 2.3 "Missing values" on page 14, describes the use of the missing value:

and	0	b,b≠0	miss
0	0	0	0
a,a≠0	0	1	miss
miss	0	miss	miss

(The result is undefined if the missing value is set to 0.)

or

`ifile1 ifile2 ofile`

("or")

$$o(t,x) = \begin{cases} 0 \quad \text{if} & i_1(t,x) = 0 \wedge i_2(t,x) = 0 \\ 1 \quad \text{if} & \begin{cases} i_1(t,x) \neq 0 \wedge i_1(t,x) \neq \text{miss} \\ \vee\, i_2(t,x) \neq 0 \wedge i_2(t,x) \neq \text{miss} \end{cases} \\ \text{miss if} & \begin{cases} i_1(t,x) = 0 \wedge i_2(t,x) = \text{miss} \\ \vee\, i_1(t,x) = \text{miss} \wedge i_2(t,x) = 0 \\ \vee\, i_1(t,x) = \text{miss} \wedge i_2(t,x) = \text{miss} \end{cases} \end{cases}$$

The following table, which must be read as described in Section 2.3 “Missing values” on page 14, describes the use of the missing value:

or	0	b,b≠0	miss
0	0	1	miss
a,a≠0	1	1	1
miss	miss	1	miss

(The result is undefined if the missing value is set to 0.)

`not`

`ifile ofile`

(“not”)

$$o(t,x) = \begin{cases} 0 & \text{if } i(t,x) \neq 0 \wedge i(t,x) \neq \text{miss} \\ 1 & \text{if } i(t,x) = 0 \\ \text{miss} & \text{if } i(t,x) = \text{miss} \end{cases}$$

(The result is undefined if the missing value is set to 0.)

4.15 Conditions

A value not equal to zero is treated as “true”, zero is treated as “false”. Typically the condition, which is in all functions of this section the first input file, is substituted by an internal pipe. For example, a command to store the differences of the two files `ifile1` and `ifile2` in `ofile`, but only where the differences are significant, could look like

```
ext ifthen -diff1test ifile1 ifile2 -sub -ifile1 -ifile2 ofile
```

`ifthen`

`ifile1 ifile2 ofile`

(“if `ifile1` then `ifile2`”)

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_1(t,x) = 0 \end{cases}$$

`ifthenc`

```
ifile ofile
```

("if `ifile` then constant")

$$o(t,x) = \begin{cases} c & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_1(t,x) = 0 \end{cases}$$

`ifnotthen`

```
ifile1 ifile2 ofile
```

("if not `ifile1` then `ifile2`")

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_1(t,x) \neq 0 \end{cases}$$

`ifnotthenc`

```
ifile ofile
```

("if not `ifile` then constant")

$$o(t,x) = \begin{cases} c & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \vee i_1(t,x) \neq 0 \end{cases}$$

`ifthenelse`

```
ifile1 ifile2 ifile3 ofile
```

("if `ifile1` then `ifile2` else `ifile3`")

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ i_3(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } \quad i_1(t,x) = \text{miss} \end{cases}$$

ifthenelsec

```
ifile1 ifile2 ofile
```

("if `ifile1` then `ifile2` else constant")

$$o(t,x) = \begin{cases} i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ c & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \end{cases}$$

ifthencelse

```
ifile1 ifile2 ofile
```

("if `ifile1` then constant else `ifile2`")

$$o(t,x) = \begin{cases} c & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ i_2(t,x) & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \end{cases}$$

ifthencelsec

```
ifile1 ofile
```

("if `ifile` then constant #1 else constant #2")

$$o(t,x) = \begin{cases} c_1 & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) \neq 0 \\ c_2 & \text{if } i_1(t,x) \neq \text{miss} \wedge i_1(t,x) = 0 \\ \text{miss} & \text{if } i_1(t,x) = \text{miss} \end{cases}$$

4.16 Geometry

dotprod

```
ifile1 ifile2 ofile
```

("dotproduct")

$$o(1, x) = \sum_t i_1(t, x)\, i_2(t, x)$$

dotprodr

```
ifile1 ifile2 ofile
```

(“dotproduct for each record”)

$$o(t, 1) = \left(\sum_{x'} w(t, x')\right)^{-1} \sum_x w(t, x)\, i_1(t, x)\, i_2(t, x)$$

where $\{w(t', x'), t' = t\}$ are the area weights obtained by the input records as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

dotprodrw

```
ifile1 ifile2 ifile3 ofile
```

(“dotproduct for each record (using a weight file)”)

$$o(t, 1) = \left(\sum_{x,\, i_3(x, t) \neq \mathrm{miss}} i_3(x, t)\right)^{-1} \sum_{x,\, i_3(x, t) \neq \mathrm{miss}} i_3(x, t)\, i_1(t, x)\, i_2(t, x)$$

`ifile3` can be understood as a weight file, see Section 3.7 “The concept of area weights” on page 27.

norm

```
ifile ofile
```

(“norm”)

$$o(1, x) = \sqrt{\sum_t i(x, t)^2}$$

normr

```
ifile ofile
```

("norm for each record")

$$o(t,1) = \left(\sum_{x'} w(t,x')\right)^{-1} \sqrt{\sum_{x} w(t,x)\, i(x,t)^2}$$

where $\{w(t',x'),\, t' = t\}$ are the area weights obtained by the input records as described in the first paragraph of Section 3.7 "The concept of area weights" on page 27.

normrw

```
ifile1 ifile2 ofile
```

("norm for each record (using a weight file)")

$$o(t,1) = \left(\sum_{x,\, i_2(x,t) \neq \mathrm{miss}} i_2(x,t)\right)^{-1} \sqrt{\sum_{x,\, i_2(x,t) \neq \mathrm{miss}} i_2(x,t)\, i_1(x,t)^2}$$

`ifile2` can be understood as a weight file, see Section 3.7 "The concept of area weights" on page 27.

dist

```
ifile1 ifile2 ofile
```

("distance")

$$o(1,x) = \sqrt{\sum_{t} (i_2(t,x) - i_1(t,x))^2}$$

distr

```
ifile1 ifile2 ofile
```

("distance for each record")

$$o(t,1) = \left(\sum_{x'} w(t,x')\right)^{-1} \sqrt{\sum_{x} w(t,x)\, (i_2(t,x) - i_1(t,x))^2}$$

where $\{w(t', x'), t' = t\}$ are the area weights obtained by the input records as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

distrw

```
ifile1 ifile2 ifile3 ofile
```

(“distance for each record (using a weight file)”)

$$o(t, 1) = \left(\sum_{x,\, i_3(x,t) \neq \text{miss}} i_3(x, t)\right)^{-1} \sqrt{\sum_{x,\, i_3(x,t) \neq \text{miss}} i_3(t, x)\,(i_2(t, x) - i_1(t, x))^2}$$

`ifile3` can be understood as a weight file, see Section 3.7 “The concept of area weights” on page 27.

rms

```
ifile1 ifile2 ofile
```

(“root mean square”) Identical to function `dist` (page 93).

rmsr

```
ifile1 ifile2 ofile
```

(“root mean square for each record”) Identical to function `distr` (page 93).

rmsrw

```
ifile1 ifile2 ifile3 ofile
```

(“root mean square for each record (using a weight file)”) Identical to function `distrw` (page 94).

normdotprod

```
ifile1 ifile2 ofile
```

(“normalized dotproduct”) The normalized dotproduct is the arcus cosine of the angle between `ifile1` and `ifile2`. With

$$S(x) = \{t, i_1(t,x) \neq \text{miss} \wedge i_2(t,x) \neq \text{miss}\}$$

it is

$$o(1,x) = \frac{\sum_{t \in S(x)} i_1(t,x)\, i_2(t,x)}{\sqrt{\sum_{t \in S(x)} i_1(t,x)^2 \sum_{t \in S(x)} i_2(t,x)^2}}$$

normdotprodr

```
ifile1 ifile2 ofile
```

("normalized dotproduct for each record") The normalized dotproduct is the arcus cosine of the angle between `ifile1` and `ifile2`. With

$$S(t) = \{x, i_1(t,x) \neq \text{miss} \wedge i_2(t,x) \neq \text{miss}\}$$

it is

$$o(t,1) = \frac{\sum_{x \in S(t)} w(t,x)\, i_1(t,x)\, i_2(t,x)}{\sqrt{\sum_{x \in S(t)} w(t,x)\, i_1(t,x)^2 \sum_{x \in S(t)} w(t,x)\, i_2(t,x)^2}}$$

where $\{w(t', x'), t' = t\}$ are the area weights obtained by the input records as described in the first paragraph of Section 3.7 "The concept of area weights" on page 27.

normdotprodrw

```
ifile1 ifile2 ifile3 ofile
```

("normalized dotproduct for each record (using a weight file)") The normalized dotproduct is the arcus cosine of the angle between `ifile1` and `ifile2`. With

$$S(t) = \{x, i_1(t,x) \neq \text{miss} \wedge i_2(t,x) \neq \text{miss} \wedge i_3(t,x) \neq \text{miss}\}$$

it is

$$o(t,1) = \frac{\sum_{x \in S(t)} i_3(t,x)\, i_1(t,x)\, i_2(t,x)}{\sqrt{\sum_{x \in S(t)} i_3(t,x)\, i_1(t,x)^2 \sum_{x \in S(t)} i_3(t,x)\, i_2(t,x)^2}}$$

`ifile3` can be understood as a weight file, see Section 3.7 “The concept of area weights” on page 27.

4.17 Means and averages

In this program there is the different notion of “mean” and “average” to distinguish two different kinds of treatment of missing values: While computing the mean, only the not missing values are considered to belong to the sample with the side effect of a probably reduced sample size. Computing the average is just adding the sample members and divide the result by the sample size. For example, the mean of 1, 2, miss, and 3 is (1+2+3)/3=2, whereas the average is (1+2+miss+3)/4=miss/4=miss. If there are no missing values in the sample, the average and the mean are identical.

In this chapter the abbreviations as in the following table are used:

mean resp. avg	$n^{-1}\sum_{i=1}^{n} x_i$
mean resp. avg weighted by $\{w_i, i = 1, \ldots, n\}$	$\left(\sum_{j=1}^{n} w_j\right)^{-1} \sum_{i=1}^{n} w_i x_i$

`mean`

```
ifile ofile
```

(“mean”)

$o(1,x) = \text{mean}\{i(t',x'), x' = x\}$

avg

```
ifile ofile
```

(“average”)

$o(1, x) = \text{avg}\{i(t', x'), x' = x\}$

mean<n>

```
ifile1 ... ifile<n> ofile
```

(“mean”)

$o(t, x) = \text{mean}\{i_1(t, x), \ldots, i_n(t, x)\}$

avg<n>

```
ifile1 ... ifile<n> ofile
```

(“average”)

$o(t, x) = \text{avg}\{i_1(t, x), \ldots, i_n(t, x)\}$

meanr

```
ifile ofile
```

(“mean for each record”)

$o(t, 1) = \text{mean}\{i(t', x'), t' = t\}$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

meanrw

```
ifile1 ifile2 ofile
```

(“mean for each record (using a weight file)”)

$$o(t,1) = \text{mean}\{i_1(t', x'), t' = t\}$$

weighted by $\{i_2(t', x'), t' = t\}$.

If `ifile1` is a GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

avgr

```
ifile ofile
```

(“average for each record”)

$$o(t,1) = \text{avg}\{i(t', x'), t' = t\}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

avgrw

```
ifile1 ifile2 ofile
```

(“average for each record (using a weight file)”)

$$o(t,1) = \text{avg}\{i_1(t', x'), t' = t\}$$

weighted by $\{i_2(t', x'), t' = t\}$.

Missing values in the weight file do not lead automatically to a missing value average. Only those points are averaged, where the weight is not the missing value.

If `ifile1` is a GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and

Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

anom

`ifile ofile`

(“anomaly”)

$o(t,x) = i(t,x) - \text{mean}\{i(t',x'), x' = x\}$

This function has to keep the fields of all records concurrently in the memory. If not enough memory is available, the user should use the functions `mean` (page 96) and `sub` (page 73).

anomr

`ifile ofile`

(“anomaly for each record”)

$o(t,x) = i(t,x) - \text{mean}\{i(t',x'), x' = x\}$

where the mean is weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

anomrw

`ifile ofile`

(“anomaly for each record (using a weight file)”)

$o(t,x) = i(t,x) - \text{mean}\{i(t',x'), t' = t\}$

where the mean is weighted by $\{i_2(t',x'), t' = t\}$.

zonmean

```
ifile ofile
```

("zonal mean") For every latitude the mean over all longitudes is computed.

zonavg

```
ifile ofile
```

("zonal average") For every latitude the average over all longitudes is computed.

mermean

```
ifile ofile
```

("meridial mean") For every longitude the mean over all latitudes is computed.

meravg

```
ifile ofile
```

("meridial average") For every longitude the average over all latitudes is computed.

runmeans

```
ifile ofile
```

("running mean series")

$o(t, x) = \text{mean}\{i(t, x), i(t+1, x), \ldots, i(t+c-1, x)\}$

runavgs

```
ifile ofile
```

("running average series")

$o(t, x) = \text{avg}\{i(t, x), i(t+1, x), \ldots, i(t+c-1, x)\}$

daymeans

```
ifile ofile
```

("daily mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and day it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

dayavgs

```
ifile ofile
```

("daily average series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and day it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \le t_n\}$$

5daymeans

```
ifile ofile
```

("5-day-interval mean series") The days of a month are thought as grouped to 5-day-intervals. These intervals are day 01 to day 05, day 06 to day 10, day 11 to day 15, etc. For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and 5-day-interval it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

5dayavgs

```
ifile ofile
```

("5-day-interval average series") The days of a month are thought as grouped to 5-day-intervals. These intervals are day 01 to day 05, day 06 to day 10, day 11 to day 15, etc. For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and 5-day-interval it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

10daymeans

```
ifile ofile
```

("10-day mean series") The days of a month are thought as grouped to 10-day-intervals. These intervals are day 01 to day 10, day 11 to day 20, and day 21 to day 30, etc. For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and 10-day-interval it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \leq t_n\}$$

10dayavgs

```
ifile ofile
```

("10-day average series") The days of a month are thought as grouped to 10-day-intervals. These intervals are day 01 to day 10, day 11 to day 20, and day 21 to day 30, etc. For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, month, and 10-day-interval it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

monmean

```
ifile ofile
```

("monthly mean")

$$\begin{aligned} o(01, x) &= \text{mean}\{i(t, x), \text{month}(i(t)) = 01\} \\ &\ldots \\ o(12, x) &= \text{mean}\{i(t, x), \text{month}(i(t)) = 12\} \end{aligned}$$

`ofile` consists always of exactly 12 records.

`monavg`

`ifile ofile`

(“monthly average”)

$$o(01, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 01\}$$
$$\ldots$$
$$o(12, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 12\}$$

`ofile` consists always of exactly 12 records.

`monmeans`

`ifile ofile`

(“monthly mean series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and month it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \leq t_n\}$$

`monavgs`

`ifile ofile`

(“monthly average series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and month it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

`seasmean`

`ifile ofile`

(“seasonal mean”)

$$o(01, x) = \text{mean}\{i(t, x), \text{season}(i(t)) = 13\}$$
$$\ldots$$
$$o(04, x) = \text{mean}\{i(t, x), \text{season}(i(t)) = 16\}$$

`ofile` consists always of exactly 4 records.

seasavg

```
ifile ofile
```

(“seasonal average”)

$$\begin{aligned} o(01, x) &= \text{avg}\{i(t, x), \text{season}(i(t)) = 13\} \\ &\dots \\ o(04, x) &= \text{avg}\{i(t, x), \text{season}(i(t)) = 16\} \end{aligned}$$

`ofile` consists always of exactly 4 records.

seasmeans

```
ifile ofile
```

(“seasonal mean series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

Be careful about the first and the last record, they may be incorrect DJF means. See function `selfirstmidlastrec` (page 63) for selecting the second to the last second record.

See also functions `rseasmeans` (page 104), `cseasmeans` (page 105), `rcseasmeans` (page 106).

rseasmeans

```
ifile ofile
```

(“robust seasonal mean series”) For every adjacent sequence $t_1, \ldots, t_n$ of records of the same season, it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

This function is robust in the sense that it ignores the year. If for example month 12 of year 255 is followed by month 1 of year 0, then both month are considered as belonging to the same season.

Be careful about the first and the last record, they may be incorrect DJF means. See function `selfirstmidlastrec` (page 63) for selecting the second to the last second record.

See also functions `seasmeans` (page 104), `cseasmeans` (page 105), `rcseasmeans` (page 106).

cseasmeans

`ifile ofile`

("controlled seasonal mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

This function is controlled in the sense, that only those seasonal means are written out, which have the same number of records for every month of the season. If for example the first three records of a `ifile` are monthly means of January, February, and March, then the mean of January and February is not written out, because there is no record for December.

See also functions `seasmeans` (page 104), `rseasmeans` (page 104), `crseasmeans` (page 105).

crseasmeans

`ifile ofile`

("controlled robust seasonal mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t,x) = \mathrm{mean}\{i(t',x), t_1 < t' \leq t_n\}$$

This function is controlled in the sense, that only those seasonal means are written out, which have the same number of records for every month of the season. If for example the first three records of a `ifile` are monthly means of January, February, and March, then the mean of January and February is not written out, because there is no record for December.

This function is robust in the sense that it ignores the year. If for example month 12 of year 255 is followed by month 1 of year 0, then both month are considered as belonging to the same season.

See also function `seasmeans` (page 104), `rseasmeans` (page 104), `cseasmeans` (page 105).

rcseasmeans

```
ifile ofile
```

("robust controlled seasonal mean series") This function is identical to function `crseasmeans` (page 105).

seasavgs

```
ifile ofile
```

("seasonal average series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t,x) = \mathrm{avg}\{i(t',x), t_1 < t' \leq t_n\}$$

Be careful about the first and the last record, they may be incorrect DJF averages. See function `selfirstmidlastrec` (page 63) for selecting the second to the last second record.

See also functions `rseasavgs` (page 107), `cseasavgs` (page 107), `crseasavgs` (page 108).

rseasavgs

`ifile ofile`

("robust seasonal mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same season, it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

This function is robust in the sense that it ignores the year. If for example month 12 of year 255 is followed by month 1 of year 0, then both month are considered as belonging to the same season.

Be careful about the first and the last record, they may be incorrect DJF averages. See function `selfirstmidlastrec` (page 63) for selecting the second to the last second record.

See also functions `seasavgs` (page 106), `cseasavgs` (page 107), `crseasavgs` (page 108).

cseasavgs

`ifile ofile`

("controlled seasonal mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

This function is controlled in the sense, that only those seasonal averages are written out, which have the same number of records for every month of the season. If for example the first three records of a `ifile` are monthly means of January, February, and March, then the average of January and February is not written out, because there is no record for December.

See also functions `seasavgs` (page 106), `rseasavgs` (page 107), `crseasavgs` (page 108)).

crseasavgs

```
ifile ofile
```

("controlled robust seasonal mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year and season, where december belongs to the northern hemispheric winter of the next year, it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$$

This function is controlled in the sense, that only those seasonal averages are written out, which have the same number of records for every month of the season. If for example the first three records of a `ifile` are monthly means of January, February, and March, then the average of January and February is not written out, because there is no record for December.

This function is robust in the sense that it ignores the year. If for example month 12 of year 255 is followed by month 1 of year 0, then both month are considered as belonging to the same season.

See also function `seasavgs` (page 106), `rseasavgs` (page 107), `cseasavgs` (page 107).

rcseasavgs

```
ifile ofile
```

("robust controlled seasonal mean series") This function is identical to function `crseasavgs` (page 108).

yearmean

    ifile ofile

("yearly mean") Identical to function `mean` (page 96).

yearavg

    ifile ofile

("yearly average") Identical to function `avg` (page 97).

yearmeans

    ifile ofile

("yearly mean series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, it is

$$o(t, x) = \text{mean}\{i(t', x), t_1 < t' \le t_n\}$$

yearavgs

    ifile ofile

("yearly average series") For every adjacent sequence $t_1, \ldots, t_n$ of records of the same year, it is

$$o(t, x) = \text{avg}\{i(t', x), t_1 < t' \le t_n\}$$

4.18 Variances, correlations, and co.

In this chapter the abbreviations as in the following table are used:

Var_0	$n^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
Var_1	$(n-1)^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
Covar_0	$n^{-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$

Covar_1	$(n-1)^{-1}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})$
Var_0 weighted by $\{w_i, i=1,\ldots,n\}$	$\frac{1}{\sum_{j=1}^{n} w_j}\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)^2$
Var_1 weighted by $\{w_i, i=1,\ldots,n\}$	$\frac{\sum_{j=1}^{n} w_j}{\left(\sum_{j=1}^{n} w_j\right)^2-\sum_{j=1}^{n} w_j^2}\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)^2$
Covar_0 weighted by $\{w_i, i=1,\ldots,n\}$	$\frac{1}{\sum_{j=1}^{n} w_j}\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)\left(y_i-\frac{\sum_{j=1}^{n} w_j y_j}{\sum_{j=1}^{n} w_j}\right)$
Covar_1 weighted by $\{w_i, i=1,\ldots,n\}$	$\frac{\sum_{j=1}^{n} w_j}{\left(\sum_{j=1}^{n} w_j\right)^2-\sum_{j=1}^{n} w_j^2}\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)\left(y_i-\frac{\sum_{j=1}^{n} w_j y_j}{\sum_{j=1}^{n} w_j}\right)$
Cor weighted by $\{w_i, i=1,\ldots,n\}$	$\frac{\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)\left(y_i-\frac{\sum_{j=1}^{n} w_j y_j}{\sum_{j=1}^{n} w_j}\right)}{\sqrt{\sum_{i=1}^{n} w_i\left(x_i-\frac{\sum_{j=1}^{n} w_j x_j}{\sum_{j=1}^{n} w_j}\right)^2\sum_{i=1}^{n} w_i\left(y_i-\frac{\sum_{j=1}^{n} w_j y_j}{\sum_{j=1}^{n} w_j}\right)^2}}$

Only those values resp. pairs of values belong to the sample which are not the missing value resp. which are both not the missing value.

var0

```
ifile ofile
```

(“variance [divisor was n-0]”)

$$o(1, x) = \mathrm{Var}_0\{i(t', x'), x' = x\}$$

`var1`

`ifile ofile`

(“variance [divisor was n-1]”)

$$o(1, x) = \mathrm{Var}_1\{i(t', x'), x' = x\}$$

`var0r`

`ifile ofile`

(“variance [divisor was n-0] for each record”)

$$o(t, 1) = \mathrm{Var}_0\{i(t', x'), t' = t\}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

`var0rw`

`ifile1 ifile2 ofile`

(“variance [divisor was n-0] for each record (using a weight file)”)

$$o(t, 1) = \mathrm{Var}_0\{i_1(t', x'), t' = t\}$$

weighted by $\{i_2(t', x'), t' = t\}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

var1r

`ifile ofile`

("variance [divisor was n-1] for each record")

$$o(t,1) = \mathrm{Var}_0\{i(t',x'), t' = t\}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 "The concept of area weights" on page 27.

var1rw

`ifile1 ifile2 ofile`

("variance [divisor was n-1] for each record (using a weight file)")

$$o(t,1) = \mathrm{Var}_1\{i_1(t',x'), t' = t\}$$

weighted by $\{i_2(t',x'), t' = t\}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 "Combining different functions" on page 19 and Section 3.3 "Advanced standard input" on page 21.)

pooledvar<n>

`ifile1 ... ifile<n> ofile`

("pooled variance") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function computes the pooled variance σ^2. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \mathrm{miss}$. It is

$$o(1,x) = \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x) \right)^2}$$

stddev0

`ifile ofile`

(“standard deviation [divisor was n-0]”)

$$o(1, x) = \sqrt{\mathrm{Var}_0\{i(t', x'), x' = x\}}$$

stddev1

`ifile ofile`

(“standard deviation [divisor was n-1]”)

$$o(1, x) = \sqrt{\mathrm{Var}_1\{i(t', x'), x' = x\}}$$

stddev0r

`ifile ofile`

(“standard deviation [divisor was n-0] for each record”)

$$o(t, 1) = \sqrt{\mathrm{Var}_0\{i(t', x'), t' = t\}}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

stddev0rw

`ifile1 ifile2 ofile`

(“standard deviation [divisor was n-0] for each record (using a weight file)”)

$$o(t, 1) = \sqrt{\mathrm{Var}_0\{i_1(t', x'), t' = t\}}$$

weighted by $\{i_2(t', x'), t' = t\}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

stddev1r

```
ifile ofile
```

(“standard deviation [divisor was n-1] for each record”)

$$o(t, 1) = \sqrt{\mathrm{Var}_1 \{ i(t', x'), t' = t \}}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

stddev1rw

```
ifile1 ifile2 ofile
```

(“standard deviation [divisor was n-1] for each record (using a weight file)”)

$$o(t, 1) = \sqrt{\mathrm{Var}_1 \{ i_1(t', x'), t' = t \}}$$

weighted by $\{ i_2(t', x'), t' = t \}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

2stddev0

```
ifile ofile
```

(“2 times standard deviation [divisor was n-0]”)

$$o(1, x) = 2\sqrt{\mathrm{Var}_0 \{ i(t', x'), x' = x \}}$$

2stddev1

`ifile ofile`

(“2 times standard deviation [divisor was n-1]”)

$$o(1, x) = 2\sqrt{\mathrm{Var}_1\{i(t', x'), x' = x\}}$$

2stddev0r

`ifile ofile`

(“2 times standard deviation [divisor was n-0] for each record”)

$$o(t, 1) = 2\sqrt{\mathrm{Var}_0\{i(t', x'), t' = t\}}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

2stddev0rw

`ifile1 ifile2 ofile`

(“2 times standard deviation [divisor was n-0] for each record (using a weight file)”)

$$o(t, 1) = 2\sqrt{\mathrm{Var}_0\{i_1(t', x'), t' = t\}}$$

weighted by $\{i_2(t', x'), t' = t\}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

2stddev1r

```
ifile ofile
```

("2 times standard deviation [divisor was n-1] for each record")

$$o(t,1) = 2\sqrt{\mathrm{Var}_1\{i_1(t',x'), t' = t\}}$$

weighted by area weights obtained by the input record as described in the first paragraph of Section 3.7 "The concept of area weights" on page 27.

2stddev1rw

```
ifile1 ifile2 ofile
```

("2 times standard deviation [divisor was n-1] for each record (using a weight file)")

$$o(t,1) = 2\sqrt{\mathrm{Var}_1\{i_1(t',x'), t' = t\}}$$

weighted by $\{i_2(t',x'), t' = t\}$.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 "Combining different functions" on page 19 and Section 3.3 "Advanced standard input" on page 21.)

covar0

```
ifile1 ifile2 ofile
```

("covariance [divisor was n-0]")

$$o(1,x) = \mathrm{Covar}_0\{(i_1(t',x'), i_2(t',x')), x' = x\}$$

covar1

```
ifile1 ifile2 ofile
```

(“covariance [divisor was n-1]”)

$$o(1, x) = \mathrm{Covar}_1 \{ (i_1(t', x'), i_2(t', x')), x' = x\}$$

covar0r

```
ifile1 ifile2 ofile
```

(“covariance [divisor was n-0] for each record”)

$$o(t, 1) = \mathrm{Covar}_0 \{ (i_1(t', x'), i_2(t', x')), t' = t\}$$

weighted by area weights obtained by the input records as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

covar0rw

```
ifile1 ifile2 ifile3 ofile
```

(“covariance [divisor was n-0] for each record (using a weight file)”)

$$o(t, 1) = \mathrm{Covar}_0 \{ (i_1(t', x'), i_2(t', x')), t' = t\}$$

weighted by $\{i_3(t', x'), t' = t\}$.

If `ifile1` or `ifile2` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile3`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

covar1r

```
ifile1 ifile2 ofile
```

(“covariance [divisor was n-1] for each record”)

$$o(t, 1) = \mathrm{Covar}_1 \{ (i_1(t', x'), i_2(t', x')), t' = t \}$$

weighted by area weights obtained by the input records as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

covar1rw

```
ifile1 ifile2 ifile3 ofile
```

(“covariance [divisor was n-1] for each record (using a weight file)”)

$$o(t, 1) = \mathrm{Covar}_1 \{ (i_1(t', x'), i_2(t', x')), t' = t \}$$

weighted by $\{ i_3(t', x'), t' = t \}$.

If `ifile1` or `ifile2` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile3`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

cor

```
ifile1 ifile2 ofile
```

(“correlation”)

$$o(1, x) = \mathrm{Cor} \{ (i_1(t', x'), i_2(t', x')), x' = x \}$$

For correlation without subtracting the mean see function `normdotprod` (page 94).

corr

```
ifile1 ifile2 ofile
```

(“correlation for each record”)

For every record t only those field elements x belong to the sample, which have $i_1(t, x) \neq \text{miss}$ and $i_2(t, x) \neq \text{miss}$. It is

$$o(t, 1) = \text{Cor}\{(i_1(t', x'), i_2(t', x')), t' = t\}$$

weighted by area weights obtained by the input records as described in the first paragraph of Section 3.7 “The concept of area weights” on page 27.

For correlation without subtracting the mean see function `normdotprodr` (page 95).

corrw

```
ifile1 ifile2 ifile3 ofile
```

(“correlation for each record (using a weight file)”)

For every record t only those field elements x belong to the sample, which have $i_1(t, x) \neq \text{miss}$ and $i_2(t, x) \neq \text{miss}$ and $i_3(t, x) \neq \text{miss}$. Then it is

$$o(t, 1) = \text{Cor}\{(i_1(t', x'), i_2(t', x')), t' = t\}$$

weighted by $\{i_3(t', x'), t' = t\}$.

For correlation without subtracting the mean see function `normdotprodrw` (page 95).

If `ifile1` or `ifile2` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile3`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

4.19 Regression

`regres`

```
ifile ofile
```

("regression") The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function estimates the parameter b. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. It is

$$o(1, x) = \frac{\sum_{t \in S(x)} \left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

`detrend`

```
ifile ofile
```

("detrend") Every time series in `ifile` is linearly detrended. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. With

$$b(x) = \frac{\sum_{t \in S(x)} \left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

and

$$a(x) = \frac{1}{\#S(x)} \sum_{t \in S(x)} i(t, x) - b(x) \left(\frac{1}{\#S(x)} \sum_{t \in S(x)} t \right)$$

it is

$$o(t, x) = i(t, x) - (a + bt)$$

This function has to keep the fields of all records concurrently in the memory. If not enough memory is available, the user should use the functions `trend` (page 121) and `subtrend` (page 121).

trend

`ifile ofile1 ofile2`

("trend") The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function estimates the parameters a and b. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. It is

$$o_2(1, x) = \frac{\sum_{t \in S(x)} \left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

and

$$o_1(1, x) = \frac{1}{\#S(x)} \sum_{t \in S(x)} i(t, x) - o_2(1, x) \left(\frac{1}{\#S(x)} \sum_{t \in S(x)} t \right)$$

Thus the estimation for a is stored in `ofile1` and that for b is stored in `ofile2`. To subtract the trend from the data see function `subtrend` (page 121).

subtrend

`ifile1 ifile2 ifile3 ofile`

("subtract trend") This function is for subtracting a trend computed by the function `trend` (page 121). The typical function call for detrend the data in `ifile` and to store the detrended data in `ofile` is

```
ext subtrend ifile -trend ifile ofile
```

It is

$$o(t, x) = i_1(t, x) - (i_2(t, x) + i_3(t, x) \cdot t)$$

where t is the record number. (The first record has $t = 1$.)

anomcoupl

`ifile1 ifile2 ofile`

("anomaly coupling") If a forcing in stored in `ifile1` drives the quantity in `ifile2`, then with this function the linear regression between both quantities can be computed.

$$o(1,x) = \frac{\mathrm{Covar}_0\{(i_1(t',x'), i_2(t',x')), x' = x\}}{\mathrm{Var}_0\{i_1(t',x'), x' = x\}}$$

anomcouplr

`ifile1 ifile2 ofile`

("anomaly coupling for each record") If a forcing in stored in `ifile1` drives the quantity in `ifile2`, then with this function the linear regression between both quantities can be computed. For every record t only those field elements x belong to the sample, which have $i_1(t,x) \neq \mathrm{miss}$ and $i_2(t,x) \neq \mathrm{miss}$. It is

$$o(t,1) = \frac{\mathrm{Covar}_0\{(i_1(t',x'), i_2(t',x')), t' = t\}}{\mathrm{Var}_0\{i_1(t',x'), t' = t\}}$$

weighted by area weights obtained by the input records as described in the first paragraph of Section 3.7 "The concept of area weights" on page 27.

anomcouplrw

`ifile1 ifile2 ifile3 ofile`

("anomaly coupling for each record (using a weight file)") If a forcing in stored in `ifile1` drives the quantity in `ifile2`, then with this function the linear regression between both quantities can be computed. For every record t only those field elements x belong to the sample, which have $i_1(t,x) \neq \mathrm{miss}$ and $i_2(t,x) \neq \mathrm{miss}$ and $i_3(t,x) \neq \mathrm{miss}$. Then it is

$$o(t,1) = \frac{\mathrm{Covar}_0\{(i_1(t',x'), i_2(t',x')), t' = t\}}{\mathrm{Var}_0\{i_1(t',x'), t' = t\}}$$

weighted by $\{i_3(t',x'), t' = t\}$.

If `ifile1` or `ifile2` is a GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the GRIB record, use `-const,1` instead of `ifile3`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.

4.20 Tests, confidence intervals, and co.

In this chapter the following notions are used

Name of distribution	**Density**	**Distribution**
`Student-t`		t_n
χ `-square`	h_n	χ_n^2
`Beta`	$b_{p,q}$	$B(p,q)$

where n is the degree of freedom and p, q are constants.

It follows some well known statistical theory about optimal 2-sided tests. Let a set θ of parameters be given, let for every $\vartheta \in \theta$ a probability P_ϑ be given, and let the null hypothesis $\vartheta = \vartheta_0$ to be tested. Let for the sample $X = (X_1, \ldots, X_n)$ the statistic of a test problem be denoted by T_{ϑ_0}. Then the 2-sided test φ_{ϑ_0} at risk α has the form

$$\varphi_{\vartheta_0}(x) = \begin{cases} 0 \text{ if } T_{\vartheta_0}(x) \in [C_1, C_2] \\ 1 \text{ if } T_{\vartheta_0}(x) \notin [C_1, C_2] \end{cases} \tag{1}$$

where the two constants C_1, C_2 are chosen in a way that firstly, under the conditions of the null hypothesis, the probability of rejection of this null hypothesis is α, and secondly that this probability is greater under the conditions of the alternative.

As regards the first condition: The requested probability is

$$P_{\vartheta_0}(\varphi_{\vartheta_0}(X) = 1) = 1 - P_{\vartheta_0}(T_{\vartheta_0}(X) \in [C_1, C_2]) = 1 - \int_{C_1}^{C_2} f_{\vartheta_0}(t)\,dt \qquad (2)$$

where f_{ϑ_0} is the density function of $T_{\vartheta_0}(X)$ under the probability P_{ϑ_0}. This leads to the first equation for determining C_1, C_2:

$$\int_{C_1}^{C_2} f_{\vartheta_0}(t)\,dt = 1 - \alpha \qquad (3)$$

As regards to the second condition: This condition is equivalent to

$$P_{\vartheta_0}(\varphi_{\vartheta_0}(X) = 1) = \min_{\upsilon \in \theta} (P_{\vartheta}(\varphi_{\vartheta_0}(X) = 1) \qquad (4)$$

with

$$P_{\vartheta}(\varphi_{\vartheta_0}(X) = 1) = 1 - P_{\vartheta}(\varphi_{\vartheta_0}(X) = 0) = 1 - \int_{C_1}^{C_2} f_{\upsilon}(t)\,dt \qquad (5)$$

where f_{ϑ} is the density function of $T_{\vartheta_0}(X)$ under the probability P_{ϑ}. In nearly all practical applications (possibly after a suitable transformation of the parameters) f_{ϑ} can be written in the form

$$f_{\upsilon}(t) = (t \in A)\, g(\upsilon)\, h(t)\, e^{\vartheta t} \qquad (6)$$

with suitable defined functions g and h and set A. (Distribution families of this kind are called exponential families.) Now equation (4) can now be written as

$$\left. \frac{\partial}{\partial \upsilon} \int_{C_1}^{C_2} f_{\upsilon}(t)\,dt \right|_{\upsilon = \vartheta_0} = 0 \qquad (7)$$

It is

$$\frac{\partial}{\partial \vartheta} f_{\upsilon}(t) = (t \in A) \frac{\partial g}{\partial \vartheta}(\upsilon)\, h(t)\, e^{\vartheta t} + (t \in A)\, t g(\vartheta)\, h(t)\, e^{\vartheta t} \qquad (8)$$

and differentiating of $1 = \int f_\upsilon(t)\,dt$ (density functions are normalised to the overall probability of one) leads to

$$0 = \frac{\partial}{\partial\vartheta}\int f_\upsilon(t)\,dt = \frac{\frac{\partial g}{\partial\vartheta}(\upsilon)}{g(\vartheta)}\int f_\upsilon(t)\,dt + \int t f_\upsilon(t)\,dt = \frac{\frac{\partial g}{\partial\vartheta}(\upsilon)}{g(\vartheta)} + E_\vartheta \tag{9}$$

where E_ϑ is the expectation value of $T_{\vartheta_0}(X)$ under P_ϑ. Thus it is $\frac{\partial g}{\partial\vartheta}(\upsilon) = -g(\vartheta)E_\vartheta$ and equation (8) rewrites to

$$\frac{\partial}{\partial\vartheta}f_\upsilon(t) = -E_\vartheta d_\upsilon(t) + t f_\vartheta(t)$$

Equation (7) is now equivalent to

$$\int_{C_1}^{C_2} t f_{\upsilon_0}(t)\,dt = E_{\vartheta_0}\int_{C_1}^{C_2} f_{\upsilon_0}(t)\,dt$$

Using equation (3) one gets the second equation for determining C_1, C_2:

$$\int_{C_1}^{C_2} t f_{\upsilon_0}(t)\,dt = E_{\vartheta_0}(1-\alpha)$$

It follows some well known statistical theory about optimal confidence intervals. The optimal confidence interval $C(X)$ at risk α, that is the true parameter ϑ is covered by this interval at probability $1-\alpha$, in formula $P_\vartheta(\vartheta \in C(X)) = 1-\alpha$, is constructed as

$$\vartheta' \in C(x) \Leftrightarrow \varphi_{\vartheta'}(x) = 0 \tag{1}$$

this means that the parameter ϑ' belongs to the confidence interval if and only if the null hypothesis $\vartheta = \vartheta'$ can not be rejected at risk α.

More details can be found in [5]. Look for “unbiased unifomly most powerful tests”.

studentt

`ifile ofile`

("student-t") Computes the distribution function of the student-t distribution.

$$o(t,x) = t_{\text{degree\_of\_freedom}}(i(t,x))$$

studenttinv

`ofile1 ofile2 ofile3 ofile4`

("student-t inverse") This function computes for the student-t-distribution some significance areas. The test results in "significant lower" if the corresponding statistic is lower than the number in `ofile1`. It results in "significant greater" if the corresponding statistic is greater than the number in `ofile2`. And it results in "significant different" if the corresponding statistic is lower than the number in `ofile3` or greater than that in `ofile4`. It is

$$o_1(1,1) = t^{-1}_{\text{degree\_of\_freedom}}(\text{risk})$$
$$o_2(1,1) = t^{-1}_{\text{degree\_of\_freedom}}(1-\text{risk})$$
$$o_3(1,1) = t^{-1}_{\text{degree\_of\_freedom}}(\text{risk}/2)$$
$$o_4(1,1) = t^{-1}_{\text{degree\_of\_freedom}}(1-\text{risk}/2)$$

chisquare

`ifile ofile`

("χ^2") Computes the distribution function of the χ^2-distribution.

$$o(t,x) = \chi^2_{\text{degree\_of\_freedom}}(i(t,x))$$

chisquareinv

`ofile1 ofile2 ofile3 ofile4`

("χ^2 inverse") This function computes for the χ^2-distribution some significance areas. The test results in "significant lower" if the corresponding statistic is lower than the number in `ofile1`. It results in “significant greater” if the corresponding statistic is greater than the number in `ofile2`. And it results in “significant different” if the corresponding statistic is lower than the number in `ofile3` or greater than that in `ofile4`. It is

$$
\begin{aligned}
o_1(1,1) &= \left(\chi^2_{\text{degree\_of\_freedom}}\right)^{-1}(\text{risk}) \\
o_2(1,1) &= \left(\chi^2_{\text{degree\_of\_freedom}}\right)^{-1}(1-\text{risk}) \\
o_3(1,1) &= C_1 \\
o_4(1,1) &= C_2
\end{aligned}
$$

where C_1, C_2 is the solution of the following equation system:

$$
\begin{aligned}
\int_{C_1}^{C_2} h_{\text{degree\_of\_freedom}}(x)\,dx &= 1-\text{risk} \\
\int_{C_1}^{C_2} h_{\text{degree\_of\_freedom}+2}(x)\,dx &= 1-\text{risk}
\end{aligned}
$$

`beta`

`ifile ofile`

(“β”) Computes the distribution function of the β-distribution.

$$o(t,x) = B(p,q)\ (i(t,x))$$

`betainv`

`ofile1 ofile2 ofile3 ofile4`

(“β inverse”) This function computes for the β-distribution some not significance area. The test results in “significant lower” if the corresponding statistic is lower than the number in `ofile1`. It results in “significant greater” if the corresponding statistic is greater than the number in `ofile2`. And it results in “significant different” if the corresponding statistic is lower than the number in `ofile3` or greater than that in `ofile4`. It is

$$o_1(1,1) = (B(p,q))^{-1}(\text{risk})$$
$$o_2(1,1) = (B(p,q))^{-1}(1-\text{risk})$$
$$o_3(1,1) = C_1$$
$$o_4(1,1) = C_2$$

where C_1, C_2 is the solution of the following equation system:

$$\int_{C_1}^{C_2} b_{p,q}(x)\,dx = 1-\text{risk}$$
$$\int_{C_1}^{C_2} b_{p+1,q}(x)\,dx = 1-\text{risk}$$

meanstatist<n>

```
ifile1 ... ifile<n> ofile
```

("mean statistic") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function computes the statistic of $\sum_{j=1}^{n} r_j a_j - c$ with user given r_j and c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \text{miss}$. It is

$$o(1,x) = \frac{\left(\sum_{j=1}^{n} \frac{r_j^2\lambda_j}{\#S_j(x)}\right)^{-1/2}\left(\sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t\in S_j(x)} i_j(t,x) - c\right)}{\sqrt{\frac{1}{\sum_{j=1}^{n}\#S_j(x) - n}\sum_{j=1}^{n}\lambda_j^{-1}\sum_{t\in S_j(x)}\left(i_j(t,x) - \frac{1}{\#S_j(x)}\sum_{t'\in S_j(x)} i_j(t',x)\right)^2}}$$

meansignif<n>

```
ifile1 ... ifile<n> ofile
```

("mean significance level") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function computes the significance level of the 1-sided test $\sum_{j=1}^{n} r_j a_j \geq c$ with user given r_j and c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T(x) = \frac{\left(\sum_{j=1}^{n} \frac{r_j^2 \lambda_j}{\#S_j(x)}\right)^{-1/2} \left(\sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t \in S_j(x)} i_j(t, x) - c\right)}{\sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x)\right)^2}}$$

it is

$$o(1, x) = t_{\sum_{j=1}^{n} \#S_j(x) - n}(T(x))$$

mean1test<n>

```
ifile1 ... ifile<n> ofile
```

("mean 1-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j \sigma^2)$ with user given λ_j and unknown σ^2. This function tests if $\sum_{j=1}^{n} r_j a_j$ is significantly lower or significantly greater than c with user given r_j and c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T(x) = \frac{\left(\sum_{j=1}^{n} \frac{r_j^2 \lambda_j}{\#S_j(x)}\right)^{-1/2} \left(\sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t \in S_j(x)} i_j(t, x) - c\right)}{\sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x)\right)^2}}$$

it is

$$o(1, x) = \begin{cases} 1 & \text{if} \quad T(x) \geq t^{-1}_{\#S_1(x) + \ldots + \#S_n(x) - n}(1 - \text{risk}) \\ -1 & \text{if} \quad T(x) \leq t^{-1}_{\#S_1(x) + \ldots + \#S_n(x) - n}(\text{risk}) \\ 0 & \text{else} \end{cases}$$

Thus a 1 means “significant greater”, a -1 means “significant lower”, and a 0 means “no significance in the sign”.

mean2test<n>

```
ifile1 ... ifile<n> ofile
```

(“mean 2-sided test”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function tests if $\sum_{j=1}^{n} r_j a_j$ is significantly different to c with user given r_j and c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T(x) = \frac{\left(\sum_{j=1}^{n} \frac{r_j^2\lambda_j}{\#S_j(x)}\right)^{-1/2}\left(\sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t \in S_j(x)} i_j(t, x) - c\right)}{\sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x)\right)^2}}$$

it is

$$o(1, x) = \begin{cases} 1 \quad \text{if} \quad T(x) \geq t^{-1}_{\#S_1(x) + \ldots + \#S_n(x) - n}(1 - \text{risk}/2) \\ \qquad\qquad \vee\, T(x) \leq t^{-1}_{\#S_1(x) + \ldots + \#S_n(x) - n}(\text{risk}/2) \\ 0 \;\; \text{else} \end{cases}$$

Thus a 1 means “significant different” and a 0 means “not significant different”.

meanconfid<n>

```
ifile1 ... ifile<n> ofile1 ofile2
```

(“mean confidence interval”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function computes the confidence interval for $\sum_{j=1}^{n} r_j a_j$ with user given r_j. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$\Delta(x) = t_{\#S_1(x) + \ldots + \#S_n(x) - n, \alpha/2}\left(\sum_{j=1}^{n} \frac{r_j^2 \lambda_j}{\#S_j(x)}\right)^{1/2}$$

$$\cdot \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x)\right)^2}$$

it is

$$o(1,x) = \sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t \in S_j(x)} i_j(t,x) - \Delta(x)$$
$$o(1,x) = \sum_{j=1}^{n} r_j \frac{1}{\#S_j(x)} \sum_{t \in S_j(x)} i_j(t,x) + \Delta(x)$$

The lower boundary of the confidence interval is therefore $o_1(1,x)$, and the upper boundary is therefore $o_2(1,x)$.

meanstatist

`ifile ofile`

("mean statistic") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the statistic of $a - c$ with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t,x) \neq \text{miss}$. This function is a special case of the function `meanstatist1` (`meanstatist<n>` (page 128)) with $r_1 = 1$ and $\lambda_1 = 1$.

meansignif

`ifile ofile`

("mean significance level") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the significance level of the 1-sided test $a \geq c$ with user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t,x) \neq \text{miss}$. This function is a special case of the function `meansignif1` (`meansignif<n>` (page 128)) with $r_1 = 1$ and $\lambda_1 = 1$.

mean1test

`ifile ofile`

("mean 1-sided test") The values of the input file `ifile1` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function tests if a is significantly lower or significantly greater than c with user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `mean1test1` (`mean1test<n>` (page 129)) with $r_1 = 1$ and $\lambda_1 = 1$.

Thus in `ofile` a 1 means "significant greater", a -1 means "significant lower", and a 0 means "no significance in the sign".

mean2test

`ifile ofile`

("mean 2-sided test") The values of the input file `ifile1` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function tests if a is significantly different to c with user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `mean2test1` (`mean2test<n>` (page 130)) with $r_1 = 1$ and $\lambda_1 = 1$.

Thus in `ofile` a 1 means "significant different" and a 0 means "not significant different".

meanconfid

`ifile ofile1 ofile2`

("mean confidence interval") The values of the input file `ifile1` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the confidence interval for a. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `meanconfid1` (`meanconfid<n>` (page 130)) with $r_1 = 1$ and $\lambda_1 = 1$.

The lower boundaries are therefore stored in `ofile1` and the upper ones in `ofile2`.

meandiffstatist

```
ifile1 ifile2 ofile
```

("mean difference statistic") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the statistic of $a_1 - a_2 - c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meanstatist2` (`meanstatist<n>` (page 128)) with $r_1 = 1, r_2 = -1$ and $\lambda_1 = \lambda_2 = 1$.

meandiffsignif

```
ifile1 ifile2 ofile
```

("mean difference significance level") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the significance level of the 1-sided test $a_1 - a_2 \geq c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meansignif2` (`meansignif<n>` (page 128)) with $r_1 = 1, r_2 = -1$ and $\lambda_1 = \lambda_2 = 1$.

meandiff1test

```
ifile1 ifile2 ofile
```

("mean difference 1-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function tests if $a_1 - a_2$ is significantly lower or significantly greater than c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `mean1test2` (`mean1test<n>` (page 129)) with $r_1 = 1, r_2 = -1$ and $\lambda_1 = \lambda_2 = 1$.

Thus in `ofile` a 1 means "significant greater", a -1 means "significant lower", and a 0 means "no significance in the sign".

meandiff2test

`ifile1 ifile2 ofile`

("mean difference 2-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function tests if $a_1 - a_2$ is significantly different to c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `mean2test2` (`mean2test<n>` (page 130)) with $r_1 = 1, r_2 = -1$ and $\lambda_1 = \lambda_2 = 1$.

Thus in `ofile` a 1 means "significant different" and a 0 means "not significant different"

meandiffconfid

`ifile1 ifile2 ofile1 ofile2`

("mean difference confidence interval") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the confidence interval of $a_1 - a_2$. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meanconfid2` (`meanconfid<n>` (page 130)) with $r_1 = 1, r_2 = -1$ and $\lambda_1 = \lambda_2 = 1$.

The lower boundaries are therefore stored in `ofile1` and the upper ones in `ofile2`.

meandiffdiffstatist

`ifile1 ifile2 ifile3 ifile4 ofile`

("mean difference of difference statistic") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the statistic of $(a_1 - a_2) - (a_3 - a_4) - c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meanstatist4` (`mean-statist<n>` (page 128)) with $r_1 = 1, r_2 = -1, r_3 = -1, r_4 = 1$ and $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

meandiffdiffsignif

`ifile1 ifile2 ifile3 ifile4 ofile`

("mean difference of difference statistic") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the significance level of the 1-sided test $(a_1 - a_2) - (a_3 - a_4) \geq c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meansignif4` (`meansignif<n>` (page 128)) with
$r_1 = 1, r_2 = -1, r_3 = -1, r_4 = 1$ and $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

meandiffdiff1test

`ifile1 ifile2 ifile3 ifile4 ofile`

("mean difference of difference 1-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function tests if $(a_1 - a_2) - (a_3 - a_4)$ is significantly lower or significantly greater than c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `mean1test4` (`mean1test<n>` (page 129)) with
$r_1 = 1, r_2 = -1, r_3 = -1, r_4 = 1$ and $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

Thus in `ofile` a 1 means "significant greater", a -1 means "significant lower", and a 0 means "no significance in the sign".

meandiffdiff2test

`ifile1 ifile2 ifile3 ifile4 ofile`

("mean difference of difference 2-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function tests if $(a_1 - a_2) - (a_3 - a_4)$ is significantly different to c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `mean2test4` (`mean2test<n>` (page 130)) with $r_1 = 1, r_2 = -1, r_3 = -1, r_4 = 1$ and $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

Thus in `ofile` a 1 means “significant different” and a 0 means “not significant different”

meandiffdiffconfid

```
ifile1 ifile2 ifile3 ifile4 ofile1 ofile2
```

(“mean difference of difference confidence interval”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma^2)$ with unknown σ^2. This function computes the confidence interval for $(a_1 - a_2) - (a_3 - a_4)$. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. This function is a special case of the function `meanconfid4` (`meanconfid<n>` (page 130)) with $r_1 = 1, r_2 = -1, r_3 = -1, r_4 = 1$ and $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

The lower boundary of the confidence interval is therefore $o_2(1, x)$, and the upper boundary is therefore $o_1(1, x)$.

The lower boundaries are therefore stored in `ofile1` and the upper ones in `ofile2`.

pooledvarstatist<n>

```
ifile1 ... ifile<n> ofile
```

(“pooled variance statistic”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j \sigma^2)$ with user given λ_j and unknown σ^2. This function computes the statistic of σ^2 / c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. It is

$$o(1, x) = \frac{1}{c} \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x) \right)^2}$$

pooledvarsignif<n>

`ifile1 ... ifile<n> ofile`

("pooled variance significance level") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j \sigma^2)$ with user given λ_j and unknown σ^2. This function computes the significance level of the 1-sided test $\sigma^2 \geq c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T(x) = \frac{1}{c} \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x) \right)^2}$$

it is

$$o(1, x) = \chi^2_{\sum_{j=1}^{n} \#S_j(x) - n}(T(x))$$

pooledvar1test<n>

`ifile1 ... ifile<n> ofile`

("pooled variance 1-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j \sigma^2)$ with user given λ_j and unknown σ^2. This function tests if σ^2 is significantly lower or significantly greater than c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T(x) = \frac{1}{c} \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x) \right)^2}$$

it is

$$o(1,x) = \begin{cases} 1 \quad \text{if} \quad T(x) \geq \left(\chi^2_{\sum_{j=1}^{n} \#S_j(x) - n}\right)^{-1}(1 - \text{risk}) \\ -1 \quad \text{if} \quad T(x) \leq \left(\chi^2_{\sum_{j=1}^{n} \#S_j(x) - n}\right)^{-1}(\text{risk}) \\ 0 \quad \text{else} \end{cases}$$

Thus a 1 means “significant greater”, a -1 means “significant lower”, and a 0 means “no significance in the sign”.

pooledvar2test<n>

```
ifile1 ... ifile<n> ofile
```

(“pooled variance 2-sided test”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function tests if σ^2 is significantly different to c with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \text{miss}$. With

$$T(x) = \frac{1}{c}\sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x)\right)^2}$$

it is

$$o(1,x) = \begin{cases} 1 \text{ if } T(x) \notin [C_1, C_2] \\ 0 \text{ if } T(x) \in [C_1, C_2] \end{cases}$$

where C_1, C_2 is the solution of the following equation system:

$$\int_{C_1}^{C_2} h_{\sum_{j=1}^{n} \#S_j(x) - n}(x)\, dx = 1 - \text{risk}$$

$$\int_{C_1}^{C_2} h_{\sum_{j=1}^{n} \#S_j(x) - n + 2}(x)\, dx = 1 - \text{risk}$$

Thus in `ofile` a 1 means “significant different” and a 0 means “not significant different”

pooledvarconfid<n>

```
ifile1 ... ifile<n> ofile1 ofile2
```

(“pooled variance confidence interval”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \lambda_j\sigma^2)$ with user given λ_j and unknown σ^2. This function computes the confidence interval for σ^2. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \mathrm{miss}$. With

$$T(x) = \sqrt{\frac{1}{\sum_{j=1}^{n} \#S_j(x) - n} \sum_{j=1}^{n} \lambda_j^{-1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x) \right)^2}$$

it is

$$o_1(1, x) = \frac{1}{C_2} T(x)$$
$$o_2(1, x) = \frac{1}{C_1} T(x)$$

where C_1, C_2 is the solution of the following equation system:

$$\int_{C_1}^{C_2} h_{\sum_{j=1}^{n} \#S_j(x) - n}(x)\, dx = 1 - \mathrm{risk}$$
$$\int_{C_1}^{C_2} h_{\sum_{j=1}^{n} \#S_j(x) - n + 2}(x)\, dx = 1 - \mathrm{risk}$$

The lower boundary of the confidence interval is therefore $o_1(1, x)$, and the upper boundary is therefore $o_2(1, x)$.

varstatist

`ifile ofile`

("variance statistic") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the statistic of σ^2/c with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `pooledvarstatist1` (`pooledvarstatist<n>` (page 136)) with $\lambda_1 = 1$.

varsignif

`ifile ofile`

("variance significance level") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the significance of the 1-sided test $\sigma^2 \geq c$ with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `pooledvarsignif1` (`pooledvarsignif<n>` (page 137)) with $\lambda_1 = 1$.

var1test

`ifile ofile`

("variance 1-sided test") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function tests if σ^2 is significantly lower or significantly greater than c with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `pooledvar1test1` (`pooledvar1test<n>` (page 137)) with $\lambda_1 = 1$.

Thus in `ofile` a 1 means "significant greater", a -1 means "significant lower", and a 0 means "no significance in the sign".

var2test

`ifile ofile`

("variance 2-sided test") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function tests if σ^2 is significantly different to c with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `pooledvar2test1` (`pooledvar2test<n>` (page 138)) with $\lambda_1 = 1$.

Thus in `ofile` a 1 means "significant greater", a -1 means "significant lower", and a 0 means "no significance in the sign".

varconfid

`ifile ofile`

("variance confidence interval") The values of the input file `ifile` are assumed to be distributed as $N(a, \sigma^2)$ with unknown σ^2. This function computes the confidence interval for σ^2. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. This function is a special case of the function `pooledvarconfid1` (`pooledvar2test<n>` (page 138)) with $\lambda_1 = 1$.

The lower boundaries are therefore stored in `ofile1` and the upper ones in `ofile2`.

varquotstatist

`ifile1 ifile2 ofile`

("variance quotient statistic") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown σ_j^2. This function computes the statistic of $(\sigma_1^2 / \sigma_2^2) / c$ with a user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t, x) \neq \text{miss}$. With

$$T_j(x) = \sqrt{\frac{1}{\#S_j(x) - 1} \sum_{t \in S_j(x)} \left(i_j(t, x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t', x) \right)^2}$$

it is

$$o(1,x) = \frac{T_1(x)}{T_1(x) + cT_2(x)}$$

varquotsignif

```
ifile1 ifile2 ofile
```

("variance quotient significance level") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown σ_j^2. This function computes the significance level of the 1-sided test $\sigma_1^2/\sigma_2^2 \geq c$ with user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \text{miss}$. With

$$T_j(x) = \sqrt{\frac{1}{\#S_j(x) - 1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x) \right)^2}$$

it is

$$o(1,x) = B\left(\frac{\#S_1(x) - 1}{2}, \frac{\#S_2(x) - 1}{2}\right)\left(\frac{T_1(x)}{T_1(x) + cT_2(x)}\right)$$

varquot1test

ifile1 ifile2 ofile

("variance quotient 1-sided test") The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown σ_j^2. This function tests if σ_1^2/σ_2^2 is significantly lower or significantly greater than c with user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \text{miss}$. With

$$T_j(x) = \sqrt{\frac{1}{\#S_j(x) - 1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x) \right)^2}$$

and

$$T(x) = \frac{T_1(x)}{T_1(x) + cT_2(x)}$$

it is

$$o(1,x) = \begin{cases} 1 \quad \text{if} \quad T(x) \geq B\left(\frac{\#S_1(x)-1}{2}, \frac{\#S_2(x)-1}{2}\right)^{-1}(1-\text{risk}) \\ -1 \quad \text{if} \quad T(x) \leq B\left(\frac{\#S_1(x)-1}{2}, \frac{\#S_2(x)-1}{2}\right)^{-1}(\text{risk}) \\ 0 \quad \text{else} \end{cases}$$

Thus a 1 means “significant greater”, a -1 means “significant lower”, and a 0 means “no significance in the sign”.

varquot2test

ifile1 ifile2 ofile

(“variance quotient 2-sided test”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown σ_j^2. This function tests if σ_1^2/σ_2^2 is significantly different to c with user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \text{miss}$. With

$$T_j(x) = \sqrt{\frac{1}{\#S_j(x)-1} \sum_{t \in S_j(x)} \left(i_j(t,x) - \frac{1}{\#S_j(x)} \sum_{t' \in S_j(x)} i_j(t',x)\right)^2}$$

and

$$T(x) = \frac{T_1(x)}{T_1(x) + cT_2(x)}$$

it is

$$o(1,x) = \begin{cases} 1 \text{ if } T(x) \notin [C_1, C_2] \\ 0 \text{ if } T(x) \in [C_1, C_2] \end{cases}$$

where C_1, C_2 is the solution of the following equation system:

$$\int_{C_1}^{C_2} b_{\frac{\#S_1(x)-1}{2},\frac{\#S_2(x)-1}{2}}(x)\,dx = 1-\mathrm{risk}$$
$$\int_{C_1}^{C_2} b_{\frac{\#S_1(x)-1}{2}+1,\frac{\#S_2(x)-1}{2}}(x)\,dx = 1-\mathrm{risk}$$

Thus in `ofile` a 1 means “significant different” and a 0 means “not significant different”.

`varquotconfid`

ifile1 ifile2 ofile

(“variance quotient confidence interval”) The values of the input file `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown σ_j^2. This function tests if σ_1^2/σ_2^2 is significantly different to c with user given c. For every field element x only those records t belong to the sample $S_j(x)$, which have $i_j(t,x) \neq \mathrm{miss}$. With

$$T_j(x) = \sqrt{\frac{1}{\#S_j(x)-1}\sum_{t\in S_j(x)}\left(i_j(t,x)-\frac{1}{\#S_j(x)}\sum_{t'\in S_j(x)} i_j(t',x)\right)^2}$$

it is

$$o_1(1,x) = \frac{(1-C_2)\,T_1(x)}{C_2 T_2(x)}$$
$$o_2(1,x) = \frac{(1-C_1)\,T_1(x)}{C_1 T_2(x)}$$

where C_1, C_2 is the solution of the following equation system:

$$\int_{C_1}^{C_2} b_{\frac{\#S_1(x)-1}{2},\frac{\#S_2(x)-1}{2}}(x)\,dx = 1-\mathrm{risk}$$
$$\int_{C_1}^{C_2} b_{\frac{\#S_1(x)-1}{2}+1,\frac{\#S_2(x)-1}{2}}(x)\,dx = 1-\mathrm{risk}$$

The lower boundary of the confidence interval is therefore $o_1(1, x)$, and the upper boundary is therefore $o_2(1, x)$.

regresstatist

```
ifile ofile
```

("regression statistic") The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function computes the statistic of $b - c$ with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. With

$$\hat{b} = \frac{\sum_{t \in S(x)} \left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

it is

$$o(1, x) = \frac{\sqrt{\#S(x) - 2}\,(\hat{b} - c) \sqrt{t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t'}}{\sqrt{\left[\left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) - \hat{b}^2 \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right) \right]^2}}$$

regressions

```
ifile ofile
```

("regression significance level") The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function computes the significance level of the 1-sided test $b \geq c$ with a user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. With

$$\hat{b} = \frac{\sum_{t \in S(x)} \left(i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

and

$$T(x) = \frac{\sqrt{\#S(x) - 2}\,(\hat{b} - c)\sqrt{t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'}}{\sqrt{\left[\left(i(t,x) - \frac{1}{\#S(x)}\sum_{t' \in S(x)} i(t',x)\right) - \hat{b}^2\left(t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'\right)\right]^2}}$$

it is

$$o(1,x) = t_{\#S(x) - 2}(T(x))$$

regres1test

```
ifile ofile
```

("regression 1-sided test") The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function tests if a is significantly lower or significantly greater than c with user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t,x) \neq \text{miss}$. With

$$\hat{b} = \frac{\sum_{t \in S(x)}\left(i(t,x) - \frac{1}{\#S(x)}\sum_{t' \in S(x)} i(t',x)\right)\left(t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'\right)}{\sum_{t \in S(x)}\left(t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'\right)}$$

and

$$T(x) = \frac{\sqrt{\#S(x) - 2}\,(\hat{b} - c)\sqrt{t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'}}{\sqrt{\left[\left(i(t,x) - \frac{1}{\#S(x)}\sum_{t' \in S(x)} i(t',x)\right) - \hat{b}^2\left(t - \frac{1}{\#S(x)}\sum_{t' \in S(x)} t'\right)\right]^2}}$$

it is

$$o(1,x) = \begin{cases} 1 & \text{if} \quad T(x) \geq t^{-1}_{\#S(x) - 2}(1 - \text{risk}) \\ -1 & \text{if} \quad T(x) \leq t^{-1}_{\#S_1(x) - 2}(\text{risk}) \\ 0 & \text{else} \end{cases}$$

Thus a 1 means “significant greater”, a -1 means “significant lower”, and a 0 means “no significance in the sign”.

regres2test

```
ifile ofile
```

(“regression 2-sided test”) The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function tests if a is significantly different to c with user given c. For every field element x only those records t belong to the sample $S(x)$, which have $i(t, x) \neq \text{miss}$. With

$$\hat{b} = \frac{\sum_{t \in S(x)} \left(i(t,x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t',x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

and

$$T(x) = \frac{\sqrt{\#S(x) - 2}\,(\hat{b} - c)\sqrt{t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t'}}{\sqrt{\left[\left(i(t,x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t',x) \right) - \hat{b}^2 \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right) \right]^2}}$$

it is

$$o(1,x) = \begin{cases} 1 \quad \text{if} \quad T(x) \leq t^{-1}_{\#S_1(x) - 2}(\text{risk}/2) \vee T(x) \geq t^{-1}_{\#S(x) - 2}(1 - \text{risk}/2) \\ 0 \;\; \text{else} \end{cases}$$

Thus a 1 means “significant different” and a 0 means “not significant different”.

regresconfid

```
ifile ofile1 ofile2
```

(“regression confidence interval”) The values of the input file `ifile` are assumed to be distributed as $N(a + bt, \sigma^2)$ with unknown a, b, and σ^2 and with record number t. This function computes the confidence interval for b. For every field element x only those records t belong to the sample $S(x)$, which have $i(t,x) \neq \text{miss}$. With

$$\hat{b} = \frac{\sum_{t \in S(x)} \left(i(t,x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t',x) \right) \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}$$

and

$$\Delta(x) = t_{\#S(x) - 2, \alpha/2} \cdot \frac{\sqrt{\left[\left(i(t,x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t',x) \right) - \hat{b}^2 \left(t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right) \right]^2}}{\sqrt{\#S(x) - 2} \sqrt{t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t'}}$$

it is

$$o_1(1,x) = \hat{b} - \Delta(x)$$
$$o_2(1,x) = \hat{b} + \Delta(x)$$

The lower boundary of the confidence interval is therefore $o_1(1,x)$, and the upper boundary is therefore $o_2(1,x)$.

corstatist

```
ifile1 ifile2 ofile
```

("correlation statistic") The values of the input files `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown a_j and σ_j^2 and with an unknown correlation ρ. This function computes the statistic of $(\rho - \rho_0) / \sqrt{1 - 2\rho\rho_0 - \rho_0^2}$ with a user given ρ_0. For every field element x only those records t belong to the sample $S(x)$, which have $i_1(t,x) \neq \text{miss}$ and $i_2(t,x) \neq \text{miss}$. Let the sample correlation denoted by $r(x)$. With

$$r'(x) = \frac{(r(x) - \rho_0)}{\sqrt{1 - 2r(x)\rho_0 - \rho_0^2}}$$

it is

$$o(1,x) = \sqrt{\#S(x) - 2}\frac{r'(x)}{\sqrt{1 - r'(x)^2}}$$

For a mathematical explanation see function `cor1test` (page 149).

corsignif

```
ifile1 ifile2 ofile
```

("correlation significance level") The values of the input files `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown a_j and σ_j^2 and with an unknown correlation ρ. This function computes the significance level of the 1-sided test $\rho \geq \rho_0$ with a user given ρ_0. For every field element x only those records t belong to the sample $S(x)$, which have $i_1(t,x) \neq \text{miss}$ and $i_2(t,x) \neq \text{miss}$. Let the sample correlation denoted by $r(x)$. With

$$r'(x) = \frac{(r(x) - \rho_0)}{\sqrt{1 - 2r(x)\rho_0 - \rho_0^2}}$$

and

$$T(x) = \sqrt{\#S(x) - 2}\frac{r'(x)}{\sqrt{1 - r'(x)^2}}$$

it is

$$o(1,x) = t_{\#S(x) - 2}(T(x))$$

For a mathematical explanation see function `cor1test` (`cor1test` (page 149)).

cor1test

```
ifile1 ifile2 ofile
```

("correlation 1-sided test") The values of the input files `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown a_j and σ_j^2 and with an unknown correlation ρ. This function tests if ρ is significantly lower or significantly greater than ρ_0 with a user given ρ_0. For every field element x only those records t belong to the sample $S(x)$, which have $i_1(t,x) \neq \text{miss}$ and $i_2(t,x) \neq \text{miss}$.

Let for every field element x the sample of `ifile<j>` be denoted by X_j. The sample X_2 is then transformed to the sample

$$X_2' = X_2 - r_0 \frac{\sqrt{\mathrm{Var}(X_2)}}{\sqrt{\mathrm{Var}(X_1)}} X_1$$

Now it is $\mathrm{Cor}(X_1, X_2) = \rho_0$ equivalent to $\mathrm{Cor}(X_1, X_2') = 0$. And the latter one is tested. It is $\mathrm{Cor}(X_1, X_2') = (\rho - \rho_0) / \sqrt{1 - 2\rho\rho_0 - \rho_0^2}$, and an analogous formula holds for the sample correlation.

Let the sample correlation denoted by $r(x)$. With

$$r'(x) = \frac{(r(x) - \rho_0)}{\sqrt{1 - 2r(x)\rho_0 - \rho_0^2}}$$

and

$$T(x) = \sqrt{\#S(x) - 2}\frac{r'(x)}{\sqrt{1 - r'(x)^2}}$$

it is

$$o(1,x) = \begin{cases} 1 & \text{if} \quad T(x) \geq t^{-1}_{\#S(x)-2}(1 - \text{risk}) \\ -1 & \text{if} \quad T(x) \leq t^{-1}_{\#S_1(x)-2}(\text{risk}) \\ 0 & \text{else} \end{cases}$$

Thus a 1 means “significant greater”, a -1 means “significant lower”, and a 0 means “no significance in the sign”.

cor2test

```
ifile1 ifile2 ofile
```

(“correlation 2-sided test”) The values of the input files `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown a_j and σ_j^2 and with an unknown correlation ρ. This function tests if ρ is significantly different to ρ_0 with a user given ρ_0. For every field element x only those records t belong to the sample $S(x)$, which have $i_1(t, x) \neq \text{miss}$ and $i_2(t, x) \neq \text{miss}$.

Let for every field element x the sample of `ifile<j>` be denoted by X_j. The sample X_2 is then transformed to the sample

$$X_2' = X_2 - r_0 \frac{\sqrt{\text{Var}(X_2)}}{\sqrt{\text{Var}(X_1)}} X_1$$

Now it is $\text{Cor}(X_1, X_2) = \rho_0$ equivalent to $\text{Cor}(X_1, X_2') = 0$. And the latter one is tested. It is $\text{Cor}(X_1, X_2') = (\rho - \rho_0) / \sqrt{1 - 2\rho\rho_0 - \rho_0^2}$, and an analogous formula holds for the sample correlation.

Let the sample correlation denoted by $r(x)$. With

$$r'(x) = \frac{(r(x) - \rho_0)}{\sqrt{1 - 2r(x)\rho_0 - \rho_0^2}}$$

and

$$T(x) = \sqrt{\#S(x) - 2} \frac{r'(x)}{\sqrt{1 - r'(x)^2}}$$

it is

$$o(1, x) = \begin{cases} 1 \quad \text{if} \quad T(x) \leq t^{-1}_{\#S_1(x) - 2}(\text{risk}/2) \vee T(x) \geq t^{-1}_{\#S(x) - 2}(1 - \text{risk}/2) \\ 0 \ \text{else} \end{cases}$$

Thus a 1 means “significant different” and a 0 means “not significant different”.

corconfid

```
ifile1 ifile2 ofile1 ofile2
```

(“correlation confidence interval”) The values of the input files `ifile<j>` are assumed to be distributed as $N(a_j, \sigma_j^2)$ with unknown a_j and σ_j^2 and with an unknown correlation ρ. This function computes the confidence interval for ρ.

For every field element x only those records t belong to the sample $S(x)$, which have $i_1(t, x) \neq \text{miss}$ and $i_2(t, x) \neq \text{miss}$. Let the sample correlation denoted by $r(x)$. With

$$c = \frac{t_{\#S(x)-2, \alpha/2}}{\#S(x) - 2 + t^2_{\#S(x)-2, \alpha}}$$

and

$$\Delta(x) = \sqrt{\frac{c^2}{1-c^2}\left(1 - r(x)^2\right)}$$

it is

$$\begin{aligned} o_1(1, x) &= r(x) - \Delta(x) \\ o_2(1, x) &= r(x) + \Delta(x) \end{aligned}$$

The lower boundary of the confidence interval is therefore $o_1(1, x)$, and the upper boundary is therefore $o_2(1, x)$.

The basis of this confidence interval is the test described at the explanation of function `cor2test` (page 150). Every value ρ_0 belongs to the confidence interval for which the null hypotheses $\rho = \rho_0$ can not be rejected.

4.21 Empirical Orthogonal Functions

Some theory about EOF's can be found in [3]. The way how to take area weights into account is outlined in [4].

`eof`

```
ifile ofile1 ofile2
```

("empirical orthogonal functions") This functions computes the (area weighted) empirical orthogonal functions (EOF's). The eigenvalues are stored in `ofile1` and the corresponding eigenfunctions in `ofile2`. The eigenvalues are sorted in descending order.

To print the relative amount of the eigenvalues compared to the total variance, type after the computations of the EOF's

```
ext info -div ofile1 -sum ofile1
```

To compute the time series of the principal component number <*n*> and to store it in the file `pc.<n>.ext`, type

```
ext anom -dotprodr ifile -selrec,<n>,<n> ofile2 pc.<n>.ext
```

The number of EOF's is one less than the minimum of the number of records and the field size.

Let now n be the field size, m be the number of time steps, let the $m \times n$-matrix A of the anomalies be defined as

$$A = (a_{t,x})_{t,x} = \left(i(t,x) - \frac{1}{\#T} \sum_{t' \in T} i(t',x) \right)_{t,x}$$

and let the area weights of the elements on position i be denoted as w_i and let the $n \times n$-matrix W be defined by

$$W = \begin{bmatrix} w_1 & & \\ & \dots & \\ & & w_n \end{bmatrix}$$

(It is possible to suppress the subtraction of the mean, but in this case something else comparable to the mean should have been subtracted by the user before.)

The covariance matrix C is now defined by

$$C = \frac{1}{m-1} A^T A$$

where A^T is the transpose of A.

The EOF's $(e_j)_j$ to the eigenvalues $(\lambda_j)_j$ are defined as the eigenvectors of CW, i.e. it is $CWe_j = \lambda_j e_j$.

The eigenvectors are orthonormal, i.e. it is

$$\sum_{x=1}^{n} w_x e_j(x)\, e_{j'}(x) = \begin{cases} 1 \text{ if } j = j' \\ 0 \text{ if } j \neq j' \end{cases}$$

If $n \le m$ then the program computes the eigenvectors $(f_j)_j$ of $W^{1/2} C W^{1/2}$ and afterwards $e_j(x) = w_j^{-1/2} f_j(x)$. Because of $W^{1/2} C W^{1/2} = \frac{1}{m-1}\left(A W^{1/2}\right)^T \left(A W^{1/2}\right)$ the elements of $W^{1/2} C W^{1/2}$ could be understood as the covariances of the “adjusted” anomalies $\sqrt{w_x} a_{t,x}$.

If $n > m$ then the program computes the eigenvectors $(g_j)_j$ of $\frac{1}{m-1} A W A^T$ and afterwards $e_j = A^T g$ and normalizes e_j at the end. The elements of $A W A^T$ could be understood as area weighted dotproducts of the anomalies of each record.

This function stores simultaneously all fields of `ifile` in the memory and should therefore not be used for large files. For large files use functions `eofspatial` (page 155) resp. `eoftime` (page 155).

`eofw`

`ifile1 ifile2 ofile1 ofile2`

(“empirical orthogonal functions (using a weight file)”) The same as function `eof` (page 152), but with `ifile2` as the weights instead of the internal weight field. The weights in `ifile2` are normalized to the sum of 1 before used.

This function stores all fields of `ifile1` in the memory and should therefore not be used for large files. For large files use functions `eofspatialw` (page 155) resp. `eof-timew` (page 156).

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

eofspatial

`ifile ofile1 ofile2`

(“empirical orthogonal function computed in the spatial space”) The same as function `eof` (page 152), but two differences: It does not store all fields of `ifile` in the memory and it does the computation always in the spatial space, i.e. it always computes the eigensolution of $\frac{1}{m-1}\left(AW^{1/2}\right)^T\left(AW^{1/2}\right)$ in the notion of the description of `eof`. Thus, if the number time steps is notable smaller then the number of not missing field elements per record, function `eoftime` (page 155) should be used instead.

eofspatialw

`ifile1 ifile2 ofile1 ofile2`

(“empirical orthogonal function computed in the spatial space (using a weight file)”) The same as function `eofspatial` (page 155), but with `ifile2` as the weights instead of the internal weight field. The weights in `ifile2` are normalized to the sum of 1 before used.

If the number time steps is notable smaller then the number of not missing field elements per record, function `eoftimew` (page 156) should be used instead.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 “Combining different functions” on page 19 and Section 3.3 “Advanced standard input” on page 21.)

eoftime

`ifile ofile1 ofile2`

(“empirical orthogonal function computed in the time space”) The same as function `eof` (page 152), but two differences: It does not store all fields of `ifile` in the memory and it does the computation always in the time space, i.e. it always computes the eigensolution of $\frac{1}{m-1}AWA^T$ in the notion of the description of `eof`.

Thus, if the number of time steps is greater than the number of not missing field elements per record, function `eofspatial` (page 155) should be used instead.

This function is creating a temporary file which is normally removed at the end. But there are some cases in which this function has no chance to remove this file, the most common is the killing of the process by the unix command `kill -9`. So if this job should be killed, it should be done without the option `-9`. If this temporarily file is not removed, the user should do it by hand. Therefore the name of this file is printed to standard error.

The size of this file is approximately 2 times the number of records of `ifile1` multiplied by the number of field elements per record.

So in the case of a very big `ifile1` the user has to take care of the directory where this file is created. This directory is `$TMPDIR` if such an environment variable exists and its contains is really a directory. Otherwise it is `/tmp`.

eoftimew

`ifile1 ifile2 ofile1 ofile2`

("empirical orthogonal function computed in the time space (using a weight file)") The same as function `eoftime` (page 155), but with `ifile2` as the weights instead of the internal weight field. The weights in `ifile2` are normalized to the sum of 1 before used.

If the number of time steps is greater than the number of not missing field elements per record, function `eofspatialw` (page 155) should be used instead.

If `ifile1` is a LOLA or GRIB file and constant area weights should be used instead of the area weights which correspond to the grid described in the LOLA or GRIB record, use `-const,1` instead of `ifile2`. (For an explanation see function `const` (page 50) and Section 3.2 "Combining different functions" on page 19 and Section 3.3 "Advanced standard input" on page 21.)

This function is creating a temporary file which is normally removed at the end. But there are some cases in which this function has no chance to remove this file, the most common is the killing of the process by the unix command `kill -9`. So if this job should be killed, it should be done without the option `-9`. If this temporarily file is not removed, the user should do it by hand. Therefore the name of this file is printed to standard error.

The size of this file is approximately 2 times the number of records of `ifile1` multiplied by the number of field elements per record.

So in the case of a very big `ifile1` the user has to take care of the directory where this file is created. This directory is `$TMPDIR` if such an environment variable exists and its contains is really a directory. Otherwise it is `/tmp`.

4.22 Fourier and spectra

`fourier`

`ifile ofile`

("fourier") Performs the fourier transformation or the reverse fourier transformation. If the record number n is a power of 2 then the algorithm of the fast fourier transformation is used. It is

$$o(t+1,x) = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} i(j+1,x)\, e^{\varepsilon 2\pi ijt}$$

where a user given $\varepsilon = -1$ leads to the forward transformation and a user given $\varepsilon = 1$ leads to the backward transformation. If the file `ifile` consists only of complex records, then the fields of file `ofile`, computed by

```
ext fourier,1 -fourier,-1 ifile ofile
```

are the same than that of `ifile`. For real input files see function `retocomplex` (page 83).

`spectrum`

`ifile ofile`

("spectrum") This function is for doing a spectral analyses of `ifile`. The user has to give the chunk length, the number of segments, and the kind of data window.

The estimation of the frequencies is done for several segments separately and averaged afterwards. The length of such a segment is called chunk length. The chunk length, which is also the longest period which can be resolved, should be chosen smaller than the total length of `ifile` to get statistical certainty by the use of several segments.

It is the task of the user to find a sensible balance between the number N of segments and the chunk length L. It is sensible to use overlapping segments. If an integer value for the number of segments is suggested by this function and this value is chosen (which should be done), then the segments are overlapping by 50%. If n is the total length of `ifile`, then the function suggests

$$N \approx \frac{2n}{L} - 1$$

Be warned: If only one segment is chosen, the variance of the frequency estimator is equal to the mean itself! This variance is reduced approximately by a factor of $(9/11)\,N$ if the segments are overlapping by 50%.

If there is no overlapping then the variance reduces nearly by the same factor (namely N), but the maximal resolvable period length is shorter. This would be a vast of information. If there is too much overlapping then the statistically dependence between the frequency estimators of the different segments is too large. In the extreme case of 100% overlapping the variance is not reduced any more!

If L is a power of 2 then for the internal fourier transformation the algorithm for the fast fourier transformation is used.

There are 4 built in data windows. Why a data window? The base of the frequency estimation is a fourier transformation. But a fourier transformation “thinks” that the data are periodically, this means that the last record of `ifile` is “thought” to be followed by the first record of `ifile`. If the data are periodical, this is correct and no window should be used, but if the data are not periodical, the fourier transformation “concentrates” too much on the sharp jump from the last record to the first one. To avoid this the data should be windowed. There are

3 often used window functions, but the influence of the used window to the result should not be too strong. The Hann window is a often used one.

The window functions are:

Window	Window function
No	$w(k) = 1$ for $k = 0, \ldots, L-1$.
Hann	$w(k) = 1 - \cos\frac{2\pi(k+1)}{L+1}$ for $k = 0, \ldots, L-1$.
Bartlett	$w(k) = w(L-1-k) = k$ for $k = 0, \ldots, \frac{L}{2} - 1$.
Welch	$1 - \left[\frac{(k+1) - 0.5(L+1)}{0.5(L+1)}\right]^2$ for $k = 0, \ldots, L-1$.

The record number t of `ofile` is the estimation of the power spectrum density of frequencies around $(t-1)/L$ cycles per time step, which corresponds to a period of $L/(t-1)$.

If a window is used `ifile` should be detrended, because not the power at exact one frequency is estimated, but a weighted average of the powers of the frequency and some “frequency bins” around it. Thus if the original data have a big mean (frequency 0), then the second record of `ofile` is very strongly influenced by this mean. The input data can be detrended by the user or by this function.

The spectrum is normalized in a way that the integral of the power spectrum density over all frequencies is round about the sum of the squares of the original data (after subtracting the mean resp. after detrending). If the data were detrended or at least the mean was subtracted, then this sum of squares is the sample variance.

For even L this integral over all frequencies is defined as

$$\int_0^{1/2} spectrum(k,x)\,dk = \frac{1}{2L}o(0,x) + \frac{1}{L}\sum_{k=1}^{L/2-1} o(k,x) + \frac{1}{2L}o(L/2,x)$$

and for odd L it is defined as

$$\int_0^{1/2} spectrum(k,x)\,dk = \frac{1}{2L}o(0,x) + \frac{1}{L}\sum_{k=1}^{(L-1)/2} o(k,x)$$

Try

```
ext longinfo -spectrum -randomnormal,10000,6
```

to convince yourself. The function `randomnormal` (page 52) produces a white spectrum, so due to the normalization the spectrum has the constant value 2.

If the detrended input data are denoted by $x_0, \ldots, x_{L-1}$ and the window function is denoted by $w_0, \ldots, w_{L-1}$ and is normalized to $\sum_t^{L-1} w_t = 1$, then a fourier transformation

$$c_k = \sum_{t=0}^{L-1} w_t x_t e^{\varepsilon 2\pi ikt}$$

is done for all $k = 0, \ldots, L-1$ and the power spectrum density $psd(k)$ at frequency k is estimated as

$$psd(k) = \begin{cases} 2|c_0|^2 & \text{if} \quad k = 0 \\ |c_k|^2 + |c_{L-k}|^2 & \text{if} \quad k > 1 \wedge k < L/2 \\ 2|c_{L/2}|^2 & \text{if} \quad k = L/2 \end{cases}$$

Thus the integral of the power spectrum density is $\sum_{t=0}^{L-1} w_t x_t$, which is approximately equal to $L^{-1} \sum_{t=0}^{L-1} x_t$.

To visualize a spectra using the plotting software `xvgr`, one can type

```
ext output spectrum.ext > y.asc
ext output -divc,<segment_length> -for,0,<half_segment_length>,1 > x.asc
paste x.asc y.asc > x_y.asc
xvgr x_y.asc
```

(See functions `output` (page 46), `divc` (page 75) and `for` (page 51) and Section 3.2 “Combining different functions” on page 19 for details.) If a logarithmic x-axis is desired, before calling `xvgr` the first line of the `x_y.asc` must be deleted, because it contains 0 as x-values. Chose menu `View` - `Graph` - `Set graph type` to chose logarithmic x-axis and menu `View - Ticks/tick labels -`
`Special ticks-tick label` to write period lengths instead of frequencies at the x-axis.

More information can be found in [2].

4.23 Interpolation

timeinterpolate

`ifile ofile`

("time interpolate") This function perform a linear interpolation between the different records. The user has to type in the number n of time steps from one record to the next. For t' with $t' = 0, \ldots, n$ it is

$$o(nt - t' + 1, x) = \frac{t'}{n} i(t, x) + \left(1 - \frac{t'}{n}\right) i(t + 1, x)$$

interpolate

`ifile ofile`

("interpolate") This function, which interpolates fields from one longitude/latitude grid to another, needs probably a description of the input grid and in any case a description of the output grid. To avoid typing the whole descriptions of the grids the user should use grib description files as in Section 3.6 "Grid description files" on page 24.

If `ifile` has SIMPLE, EXTRA, or SERVICE format, then both descriptions must be given, that of the input grid and that of the output grid. If `ifile` has LOLA or GRIB format, then only the description of the output grid must be given, since the input grid is known. It is a good idea to choose LOLA as output format.

First example: To interpolate the EXTRA file `ifile.ext` from a T21 grid to a regular grid with 72 longitude and 36 latitudes, just type

```
cat $GRIDS/t21.grid.asc $GRIDS/r72x36.grid.asc | lola interpolate ifile.ext \
  ofile.lola
```

The environment variable `GRIDS` should be set as described in Section 3.6 "Grid description files" on page 24.

Second example: To interpolate the GRIB file `ifile.grb` to the same output grid, just type

```
lola interpolate ifile.grb ofile.lola < $GRIDS/r72x36.grid.asc
```

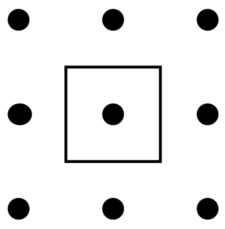
If only a box of `ofile` is needed, construct a grid description file of this box by using the functions `griddesindexbox0` (page 164), `griddesindexbox1` (page 164), `griddeslonlatbox0` (page 165), or `griddeslonlatbox1` (page 165).

The basis of the interpolation is an underlying continuous field which is constructed in the following way: For two neighboured longitudes x_1 and x_2 and two neighboured latitudes y_1 and y_2 of the input grid every point at longitude x and latitude y with $x_1 \le x \le x_2$ and $y_1 \le y \le y_2$ is assigned the value

$$\begin{aligned} a = a_{11} &+ (a_{12} - a_{11}) \frac{x - x_1}{x_2 - x_1} + (a_{22} - a_{11}) \frac{y - y_1}{y_2 - y_1} \\ &+ (a_{22} - a_{12} - a_{21} + a_{11}) \frac{(x - x_1)\,(y - y_1)}{(x_2 - x_1)\,(y_2 - y_1)} \end{aligned} \tag{1}$$

where a_{ij} is the value at longitude x_i and latitude y_j. If one of the four values $a_{11}, a_{12}, a_{21}, a_{22}$ is the missing value, then a is also the missing value. This means: Interpolation is done only between input grid points, there is never done some kind of extrapolation.

The value at a special output grid point is now computed as a weighted mean of this underlying continuous field over an area which is bounded by lines lying exactly between this grid point and the neighboured ones. In the following figure the black circles represent output grid points and the square shows the area which is used for computing the area weighted mean to determine the value of the output grid point in the middle.



The special cases of output grid points lying at the edge of the output grid is treated the same as discussed in the explanation of function `weight0` (page 49).

Parts of the area at which the underlying continuous field has the missing value are treated as not belonging to the area with the side effect of a probably reduced area size.

A field is losing contrast while interpolating, or more precise, the field is converging to a constant field if interpolation is done infinitely often. This can very easily be seen for the special case of an output grid which is identical to the input grid. The reason for losing contrast is as follows: The values at the output grid points are area weighted means of the underlying continuous field, but the values at the input grid points are not.

To get a contrast preserving interpolation one has to use another underlying continuous field: It must be determined by the condition that for every input grid point the mean of this field over the area around this point must be the value of the input field at this point.

To use such an underlying continuous field call function `contrast` (page 164) before the interpolation. Function `contrast` changes the input field to a new field in a way, that the continuous field which is constructed from the new field using equation (1) is the desired underlying continuous field.

This implies that if a coarse grid is interpolated contrast preserving to a fine grid and is afterwards interpolated back contrast preserving to the original grid, the field should be nearly unchanged. For the special case of an output grid which is equal to the input grid the contrast preserving interpolation let the field unchanged.

Why is function `contrast` not part of the function interpolate? There are two reasons: Firstly a user does probably not wish that kind of contrast preserving and secondly on a multi processor machine it is now possible to parallelize the computation.

Summarized: **If function `contrast` is not used then the interpolation is losing contrast and if it is used, then the interpolation is contrast preserving.**

contrast

`ifile ofile`

("contrast") This function is thought to be used before calling the function `interpolate` (page 161) to raise the "contrast" of the input field to make a contrast preserving interpolation. For reason see the explanation of function `interpolate`.

For example type

```
cat $GRIDS/t21.grid.asc $GRIDS/r72x36.grid.asc | lola interpolate -contrast \
  ifile.ext ofile.lola
```

or

```
lola interpolate -contrast ifile.grb ofile.lola < $GRIDS/r72x36.grid.asc
```

If the field of `ofile` is interpolated by function `interpolate` to the same grid, then the interpolated field is unchanged. Try

```
lola griddes ifile.lola | lola interpolate -contrast ifile.lola ofile.lola
lola info -sub ifile.lola ofile.lola
```

to convince yourself!

griddesindexbox0

("grid description of index box") This function can be used to construct a grid description file of a box.

For example:

```
cat $GRIDS/r72x36.grid.asc - | lola griddesindexbox0 > box.grid.asc
```

(Press `control-D` at end of standard input.)

griddesindexbox1

`ifile`

("grid description of index box") This function can be used to construct a grid description file of a box.

For example:

```
lola griddesindexbox1 any_file_with_correct_grid.lola > box.grid.asc
```

griddeslonlatbox0

("grid description of longitude/latitude box") This function can be used to construct a grid description file of a box.

For example:

```
cat $GRIDS/r72x36.grid.asc - | lola griddeslonlatbox0 > box.grid.asc
```

(Press `control-D` at end of standard input.)

griddeslonlatbox1

`ifile`

("grid description of longitude/latitude box") This function can be used to construct a grid description file of a box.

For example:

```
lola griddeslonlatbox1 any_file_with_correct_grid.lola > box.grid.asc
```

4.24 Classes

classes

`ifile ofile1 ofile2`

("classes") This function is for counting and for averaging data which lie in user given quantity classes. For every user given time period in `ofile1` is stored for every user given quantity class the mean of all values which lie in this classes and in `ofile2` the number of values which lie in this class.

(See Section 3.4 "Not required output files" on page 22 if only one of the both output files are required.)

At first the user has to give the number of quantity classes.

Then he has to give a factor b and a summand a which can be used for transforming the input data to more common units. If for example in `ifile` temperatures in Kelvin are stored, but the quantity classes should be given in degree Celsius, then a good choice is $b = 1$ and $a = -273.15$. If in `ifile` precipitations in meter per second are stored, but the quantity classes should be given in millimetre per day, then a good choice is $b = 8.64e7$ and $a = 0$. If no transformation on the input data should be done the user must set $b = 1$ and $a = 0$.

After giving b and a the user has to give for all classes a level number and the lower and the upper boundary of this class. A value y falls in a quantity classes with lower boundary c_1 and upper boundary c_2 if and only if

$$c_0 \leq a + by \wedge a + by < c_1$$

At the end the user has to give the length of the time periods which should be considered separately. The length 0 means to consider all records together.

This computations are done independent for every field element.

durations

```
ifile1 ifile2 ifile3 ofile1 ofile2 ofile3
```

("durations of events") This function is for analysing the durations of events which are defined by a value lying in a user given range. For example dry spells could be defined by precipitation less than 0.1 mm/day for more than a given number of days. Heat periods could be defined by a maximum temperature of more than 30˚C for a at least a given number of days.

This function is doing the computations separately for every user given time period and independent for every field element.

Every event belongs to an event spell, which is defined by the maximal interval around this event which consists only of other events. For example every dry day belongs to a dry spell (which naturally could have a length of 1).

For analysing the durations of these event spells the user can define several event spell duration classes. An event belongs to such an event spell duration class if the event spell around it has a length greater than or equal to a minimal and less than of equal to a maximal number of records. A record which is not an event belongs to no event spell duration class.

This function stores in `ofile1` for every event spell duration class the relative amount of records belonging to this class. In `ofile1` the user can find the numbers for statements like: “40% of all days are belonging to a dry spell longer than one month”.

It stores in `ofile2` for every event spell duration class the number of the occurrences of this event spell duration class. Here the user can find the values for statements like: “Every year there are at average 3.4 dry spells with a length of more than one month”. An event spell which overlap the time period which is considered separately is only counted relatively. If for example only 3 records of an event spell of the total length of 12 records are lying in the considered time interval, these event spell is counted only as 0.25.

And last not least this function stores in `ofile3` the average waiting time for the next record which is not an event. These numbers can be used for statements like “The average waiting time for the next precipitation on a randomly chosen day is 5.2 days”. The more exact formulation is: For every record in the considered time period which is not an event it is added 0 and for every one which is an event it is added the number of records to the next record which is not an event, even if this one is lying outside the time period. This sum is then divided by the length of the considered time period and stored in `ofile3`.

(See Section 3.4 “Not required output files” on page 22 if not all of the output files are required.)

The input file `ifile1` is scanned for the values.

Because an event spell could have begun before the first record and could go on behind the last one, there must be additional information about the beginning of the first and the ending of the last event spell. These information must be given in `ifile2` and `ifile3`.

From `ifile2` and `ifile3` only the first records are used. That of `ifile2` must give the number of records an event spell has occurred already before the first record of `ifile1` and that of `ifile3` must give the number of records an event spell is still going on after the last record of `ifile1`. Only the integer parts of the field elements of `ifile2` and `ifile3` are used.

The exact use is as follows: If the value of the first record of `ifile1` is an event, then (the integer part of) the number of `ifile2` is added to the length of the event spell which this first record belongs to. And if the value of the last record of `ifile1` is an event, then (the integer part of) the number of `ifile3` is added to the length of the event spell which this last record belongs to.

Normally the values needed for `ifile2` and `ifile3` are unknown, because there are no data before and after the used time period. So a more or less realistic estimation must be used. For getting such estimations see function `durations0` (page 169).

Use `-null` instead of `ifile2` and `ifile3` if all the numbers should be 0. (See description of function `null` (page 50).)

After calling the function the user has to give a factor b and a summand a which can be used for transforming the input data to more common units. If for example in `ifile1` temperatures in Kelvin are stored, but the quantity classes should be given in degree Celsius, then a good choice is $b = 1$ and $a = -273.15$. If in `ifile` precipitations in meter per second are stored, but the quantity classes should be given in millimetre per day, then a good choice is $b = 8.64e7$ and $a = 0$. If no transformation on the input data should be done the user must set $b = 1$ and $a = 0$.

Then the user has to give a lower boundary c_1 and an upper boundary c_2 to define an event. If for example dry spells are considered, the lower boundary could be 0 of less and the upper one could be 1 millimetre per day. If extreme heat events are investigated, the lower boundary could be 30 degree Celsius and the upper one could be 100 degree Celsius.

A value y is an event if and only if

$$c_0 \le a + by \wedge a + by < c_1$$

Afterwards the user has to give the code numbers which should be used while writing `ofile1`, `ofile2`, and `ofile3`.

Several duration classes could be analysed simultaneously by this function. Therefore the number of duration classes must be given now.

Then for every event spell duration class a level number (which is only used for writing the headers) and the minimal and the maximal length must be given. (A maximal length of 0 means no upper limit.)

At the end the user has to give the length of the time periods which should be considered separately. The length 0 means to consider all records together.

This function is creating a temporary file which is normally removed at the end. But there are some cases in which this function has no chance to remove this file, the most common is the killing of the process by the unix command `kill -9`. So if this job should be killed, it should be done without the option `-9`. If this temporarily file is not removed, the user should do it by hand. Therefore the name of this file is printed to standard error.

The size of this file is approximately 2 times the number of records of `ifile1` multiplied by the number of field elements per record.

So in the case of a very big `ifile1` the user has to take care of the directory where this file is created. This directory is `$TMPDIR` if such an environment variable exists and its contains is really a directory. Otherwise it is `/tmp`.

durations0

```
ifile ofile1 ofile2
```

("durations of events preparation") Before reading the explanations to this functions read that of function `durations` (page 166) first, please.

This function is doing the computations independent for every field element.

The estimation of the values for input file `ifile2` and `ifile3` of function `durations` can be done by thinking the data file `ifile1` as cyclic. Naturally this is only sensible if the date which would follow the date of the last record of `ifile1` is the same date than that of the first record of `ifile1`.

If the value of the first record of `ifile1` is not an event a 0 is written into `ofile1`, otherwise the length (in records) of the event spell around it is written into `ofile1`. If the value of the last record of `ifile1` is not an event a 0 is written into `ofile2`, otherwise the length (in records) of the event spell around it is written into `ofile2`.

The typically combination of this function together with function `durations` could be as follows:

```
ext durations0 ifile first.ext last.ext < durations.in
ext durations ifile last.ext first.ext ofile1 ofile2 ofile3 < durations.in
```

5. References

[1] WMO-Nr. 306 Manual on Codes, Volume 1, International Codes, Part B - Binary Codes.

[2] Press W. H., Saul A. Teukolsky, W. T, Vetterling, B. P. Flannery, 1992, Numerical Recipes in C, Cambridge University Press, ISBN 0-521-43108-5.

[3] R.W.Preisendorfer, 1988: “Principal Component Analysis in Meteorology and Oceanography”, Developments in Atmospheric Science,17.

[4] T.P.Buell, 1971: “Integral Equation Representation for Factor Analysis”, J.Atmos.Science,28,1502-1505.

[5] Lehmann, E.L., 1986, Testing Statistical Hypothesis, 2nd ed., John Wiley & Sons, New York, Chichester, Brisbane, Toronto, Singapore, ISBN
0-471-84083-1.

6. List of Functions

Information 38

shortinfo 38
info 38
longinfo 39
formatinfo 39
pipeinfo 39
pipeshortinfo 39
gribinfo 40
griddes 40
grads 40
nrec 40
nyear 41
nmon 41
ndate 41
ntime 41
ncode 41
nlevel 42
showyear 42
showmon 42
showdate 42
showtime 43
showcode 43
showlevel 43
countc 43
countcr 44
status 44

Formatted input and output 44

input 44
inputsmp 44
inputext 45
inputsrv 45
inputlola 45
output 46
outputint 46
outputsmp 46
outputext 46
outputsrv 47
outputlola 47

Converting the format 48

copy 48
copy2 48

Generation of files 49

weight0 49
weight1 50
null 50
const 50
consts 50
pi 51
e 51
for 51
random 52
randomnormal 52

Manipulating the header 52

chdate 52
chtime 52
chyear 53
chmon 53
chday 53
chcode 53
chcodes 53
chlevel 53
chdispo1 54
chdispo2 54

Manipulating the field 54

chsize 54
maskindexbox 55
masklonlatbox 55
normalize 56
shiftleft 56
shiftright 56
swapleftright 56
swaprightleft 56

swaptopbottom 57
swapbottomtop 57
break 57
break<n> 57
melt 57
meltall 58
melt<n> 58
enlarge 58
thinout 58
thinoutr 58

Manipulating the sequence of records 59

reverse 59
reverser 59
transpose 59
transposer 60
split<n> 60
merge<n> 60
mergedate2 61
replace 61
cat<n> 61

Selection 62

sel 62
selrec 62
selfirstrec 62
selfirstmidlastrec 63
seldate 63
selfirstdate 63
selfirsttime 63
selyear 64
selmon 64
selseas 64
selday 64
selcode 64
selfirstcode 65
selcode<n> 65
sellevel 65
seldispo1 65
seldispo2 66
selindexbox 66
sellonlatbox 67
selfile<m>of<n> 67
selfile<l>and<m>of<n> 67
selfile<k>and<l>and<m>of<n> 68

Missing values 68

setctomiss 68
setmiss 68
setmissc 68
setnotmiss 69
setnotmissc 69
countmiss 69
countmissr 69
countnotmiss 69
countnotmissr 70
packr 70
unpackr 70

Sorting and ranking 71

sort 71
sortr 71
rank 71
rankr 71

Arithmetic 72

sum 72
sum<n> 72
sumr 72
add 72
addc 72
addcc 73
sub 73
subc 73
subfromc 73
subcc 73
subfromcc 74
minus 74
mul 74
mulc 74
mulcc 74
div 75
divc 75
divcc 75
inverse 75
mod 75
modc 76
power 76

Maximum and Minimum 76
max 76
max<n> 77
maxr 77
maxabsdiffr 77
yearmaxs 77
monmaxs 78
daymaxs 78
min 78
min<n> 78
minr 79
yearmins 79
monmins 79
daymins 79

Mathematical functions 80
sign 80
exp 80
log 80
log10 80
sin 81
cos 81
tan 81
asin 81
acos 81
atan 82
atan2 82
conj 82
re 82
im 82
arg 83
retocomplex 83
imtocomplex 83
recttocomplex 83
complextorect 83
poltocomplex 84
complextopol 84

Comparisons and Logic 84
eq 84
eqc 84
neq 85
nec 85
le 85
lec 86
lt 86
ltc 86
ge 86
gec 87
gt 87
gtc 87
and 88
or 88
not 89

Conditions 89
ifthen 89
ifthenc 90
ifnotthen 90
ifnotthenc 90
ifthenelse 90
ifthenelsec 91
ifthencelse 91
ifthencelsec 91

Geometry 91
dotprod 91
dotprodr 92
dotprodrw 92
norm 92
normr 93
normrw 93
dist 93
distr 93
distrw 94
rms 94
rmsr 94
rmsrw 94
normdotprod 94
normdotprodr 95
normdotprodrw 95

Means and averages 96
mean 96
avg 97
mean<n> 97
avg<n> 97
meanr 97
meanrw 97

avgr 98
avgrw 98
anom 99
anomr 99
anomrw 99
zonmean 100
zonavg 100
mermean 100
meravg 100
runmeans 100
runavgs 100
daymeans 101
dayavgs 101
5daymeans 101
5dayavgs 101
10daymeans 102
10dayavgs 102
monmean 102
monavg 103
monmeans 103
monavgs 103
seasmean 103
seasavg 104
seasmeans 104
rseasmeans 104
cseasmeans 105
crseasmeans 105
rcseasmeans 106
seasavgs 106
rseasavgs 107
cseasavgs 107
crseasavgs 108
rcseasavgs 108
yearmean 109
yearavg 109
yearmeans 109
yearavgs 109

Variances, correlations, and co. 109

var0 110
var1 111
var0r 111
var0rw 111
var1r 112
var1rw 112
pooledvar<n> 112
stddev0 113
stddev1 113
stddev0r 113
stddev0rw 113
stddev1r 114
stddev1rw 114
2stddev0 114
2stddev1 115
2stddev0r 115
2stddev0rw 115
2stddev1r 116
2stddev1rw 116
covar0 116
covar1 117
covar0r 117
covar0rw 117
covar1r 118
covar1rw 118
cor 118
corr 119
corrw 119

Regression 120

regres 120
detrend 120
trend 121
subtrend 121
anomcoupl 122
anomcouplr 122
anomcouplrw 122

Tests, confidence intervals, and co. 123

studentt 126
studenttinv 126
chisquare 126
chisquareinv 126
beta 127
betainv 127
meanstatist<n> 128
meansignif<n> 128
mean1test<n> 129
mean2test<n> 130
meanconfid<n> 130
meanstatist 131
meansignif 131
mean1test 132
mean2test 132

meanconfid 132
meandiffstatist 133
meandiffsignif 133
meandiff1test 133
meandiff2test 134
meandiffconfid 134
meandiffdiffstatist 134
meandiffdiffsignif 135
meandiffdiff1test 135
meandiffdiff2test 135
meandiffdiffconfid 136
pooledvarstatist<n> 136
pooledvarsignif<n> 137
pooledvar1test<n> 137
pooledvar2test<n> 138
pooledvarconfid<n> 139
varstatist 140
varsignif 140
var1test 140
var2test 141
varconfid 141
varquotstatist 141
varquotsignif 142
varquot1test 142
varquot2test 143
varquotconfid 144
regresstatist 145
regressions 145
regres1test 146
regres2test 147
regresconfid 147
corstatist 148
corsignif 149
cor1test 149
cor2test 150
corconfid 151

Empirical Orthogonal Functions 152

eof 152
eofw 154
eofspatial 155
eofspatialw 155
eoftime 155
eoftimew 156

Fourier and spectra 157

fourier 157
spectrum 157

Interpolation 161

timeinterpolate 161
interpolate 161
contrast 164
griddesindexbox0 164
griddesindexbox1 164
griddeslonlatbox0 165
griddeslonlatbox1 165

Classes 165

classes 165
durations 166
durations0 169

7. Index of Functions

Numerics

10dayavgs 102
10daymeans 102
2stddev0 114
2stddev0r 115
2stddev0wr 115
2stddev1 115
2stddev1r 116
2stddev1rw 115, 116
5dayavgs 101
5daymeans 101

A

acos 81
add 72
addc 72
addcc 73
and 88
anom 99
anomcoupl 122
anomcouplr 122
anomcouplrw 122
anomr 99
anomrw 99
arg 83
asin 81
atan 82
atan2 82
avg 97
avg<n> 97
avgr 98
avgrw 98

B

beta 127
betainv 127
break 57
break<n> 57

C

cat<n> 61
chcode 53
chdate 52
chday 53
chdispo1 54
chdispo2 54
chisquare 126
chisquareinv 126
chmon 53
chsize 54
chtime 52
chyear 53
classes 165
complextopol 84
complextorect 83
conj 82
const 50
consts 50
contrast 164
copy 48
copy2 48
cor 118
cor1test 149
cor2test 150
corconfid 151
corr 119
corrw 119
corsignif 149
corstatist 148
cos 81
count 44
countc 43
countcr 44
countmiss 69
countmissr 69
countnotmiss 69
countnotmissr 70
covar0 116
covar0r 117
covar0rw 117
covar1 117
covar1r 118
covar1rw 118
crseasavgs 108
crseasmeans 105
cseasavgs 107
cseasmeans 105

D

datemerge2 61
dayavgs 101
daymaxs 78
daymeans 101

daymins 79
detrend 120
dist 93
distr 93
distrw 94
div 75
divc 75
divcc 75
dotprod 91
dotprodr 92
dotprodrw 92
durations 166
durations0 169

E

e 51
enlarge 58
eof 152
eofspatial 155
eofspatialw 155
eoftime 155
eoftimew 156
eofw 154
eq 84
eqc 84
exp 80

F

for 51
formatinfo 39
fourier 157

G

ge 86
gec 87
grads 40
gribinfo 40
griddes 40
griddesindexbox0 164
griddesindexbox1 164
griddeslonlatbox0 165
griddeslonlatbox1 165
gt 87
gtc 87

I

ifnotthen 90
ifnotthenc 90
ifthen 89
ifthenc 90
ifthencelse 91
ifthencelsec 91
ifthenelse 90
ifthenelsec 91
im 82
imtocomplex 83
info 38
input 44
inputext 45
inputsmp 44
inputsrv 45
interpolate 161
interpolatem 165
inverse 75

L

le 85
lec 86
log 80
log10 80
longinfo 39
lt 86
ltc 86

M

maskbox 55
maskindexbox 55
masklonlatbox 55
max 76
max<n> 77
maxr 77
mean 96
mean<n> 97
mean1test 132
mean1test<n> 129
mean2test 132
mean2test<n> 130
meanconfid 132
meanconfid<n> 130
meandiff1test 133
meandiff2test 134
meandiffconfid 134
meandiffdiff1test 135
meandiffdiff2test 135
meandiffdiffconfid 136
meandiffdiffsignif 135
meandiffdiffstatist 134
meandiffsignif 133
meandiffstatist 133
meanr 97
meanrw 97

meansignif 131
meansignif<n> 128
meanstatist 131
meanstatist<n> 128
meanwr 91
melt 57
melt<n> 58
meltall 58
meravg 100
merge<n> 60
mermean 100
min 78
min<n> 78
minr 79
minus 74
mod 75
modc 76
monavg 103
monavgs 103
monmaxs 78
monmean 102
monmeans 103
monmins 79
mul 74
mulc 74
mulcc 74

N

ncode 41
ndate 41
nec 85
neq 85
nlevel 42
nmon 41
norm 92
normalize 56, 84
normdotprod 94
normdotprodr 95
normdotprodrw 95
normr 93
normrw 93
not 89
nrec 40
ntime 41
null 50
nyear 41

O

or 88
output 46
outputext 46
outputint 46
outputintasc 46
outputlola 47
outputsmp 46
outputsrv 47

P

packr 70
pi 51
pipeinfo 39
pipeshortinfo 39
poltocomplex 84
pooledvar<n> 112
pooledvar1test<n> 137
pooledvar2test<n> 138
pooledvarconfid<n> 139
pooledvarsignif<n> 137
pooledvarstatist<n> 136
power 76

R

random 52
randomnormal 52
rank 71
rankr 71
rcseasavgs 108
rcseasmeans 106
re 82
recttocomplex 83
regres2test 147
regressignif 145
replace 61
retocomplex 83
reverser 59
rms 94
rmsr 94
rmsrw 94
rseasavgs 107
rseasmeans 104
runavgs 100
runmeans 100

S

seasavg 104
seasavgs 106
seasmean 103
seasmeans 104
sel 62
selcode 64
seldispo1 65
seldispo2 66

selfile<k>and<l>and<m>of<n> 68
selfile<l>and<m>of<n> 67
selfile<m>of<n> 67
selfirstcode 65
selfirstdate 63
selfirsttime 63
selindexbox 66
sellonlatbox 67
shortinfo 38
showcode 43
showdate 42
showlevel 43
showmon 42
showtime 43
showyear 42
sign 80
sin 81
sort 49, 71
sortr 59, 71
spectrum 157
split<n> 60
stddev0 113
stddev0r 113
stddev0rw 113
stddev1 113
stddev1r 114
stddev1rw 114
studentt 126
studenttinv 126
sub 73
subc 73
subcc 73
subfromc 73
subfromcc 74
subtrend 121
sum 72
sum<n> 72
sumr 72

T

tan 81
thinout 58
thinoutr 58
timeinterpolate 161
transposer 60
trend 121

U

unpackr 70

V

var0 110
var0r 111
var0rw 111
var1 111
var1r 112
var1rw 112
var1test 140
var2test 141
varconfid 141
varquot1test 142
varquot2test 143
varquotconfid 144
varquotsignif 142
varquotstatist 141
varsignif 140
varstatistic 140

W

weight0 49
weights 50

Y

yearavg 109
yearavgs 109
yearmaxs 77
yearmean 109
yearmeans 109
yearmins 79

Z

zonavg 100
zonmean 100