

# CDO User Guide

---

Climate Data Operators  
Version 1.9.0  
August 2017

Uwe Schulzweida – *MPI for Meteorology*

---

# Contents

<b>1. Introduction</b>	<b>6</b>
1.1. Building from sources	6
1.1.1. Compilation	7
1.1.2. Installation	7
1.2. Usage	7
1.2.1. Options	8
1.2.2. Environment variables	9
1.2.3. Operators	9
1.2.4. Operator chaining	9
1.2.5. Parallelized operators	10
1.2.6. Operator parameter	10
1.3. Horizontal grids	10
1.3.1. Grid area weights	11
1.3.2. Grid description	11
1.4. Z-axis description	14
1.5. Time axis	15
1.5.1. Absolute time	15
1.5.2. Relative time	15
1.5.3. Conversion of the time	15
1.6. Parameter table	15
1.7. Missing values	16
1.7.1. Mean and average	16
1.8. Percentile	17
1.8.1. Percentile over timesteps	17
<b>2. Reference manual</b>	<b>18</b>
2.1. Information	19
2.1.1. INFO - Information and simple statistics	20
2.1.2. SINFO - Short information	21
2.1.3. DIFF - Compare two datasets field by field	22
2.1.4. NINFO - Print the number of parameters, levels or times	23
2.1.5. SHOWINFO - Show variables, levels or times	24
2.1.6. FILEDES - Dataset description	25
2.2. File operations	26
2.2.1. COPY - Copy datasets	27
2.2.2. REPLACE - Replace variables	28
2.2.3. DUPLICATE - Duplicates a dataset	28
2.2.4. MERGEGRID - Merge grid	28
2.2.5. MERGE - Merge datasets	29
2.2.6. SPLIT - Split a dataset	30
2.2.7. SPLITTIME - Split timesteps of a dataset	32
2.2.8. SPLITSEL - Split selected timesteps	33
2.2.9. DISTGRID - Distribute horizontal grid	34
2.2.10. COLLGRID - Collect horizontal grid	35
2.3. Selection	36
2.3.1. SELECT - Select fields	37
2.3.2. SELMULTI - Select multiple fields via GRIB1 parameters	39
2.3.3. SELVAR - Select fields	40
2.3.4. SELTIME - Select timesteps	42
2.3.5. SELBOX - Select a box of a field	44

2.3.6.	SELGRIDCELL - Select grid cells	45
2.3.7.	SAMPLEGRID - Resample grid	45
2.4.	Conditional selection	46
2.4.1.	COND - Conditional select one field	47
2.4.2.	COND2 - Conditional select two fields	47
2.4.3.	CONDC - Conditional select a constant	48
2.4.4.	MAPREDUCE - Reduce fields to user-defined mask	49
2.5.	Comparison	50
2.5.1.	COMP - Comparison of two fields	51
2.5.2.	COMPC - Comparison of a field with a constant	52
2.6.	Modification	53
2.6.1.	SETATTRIBUTE - Set attributes	55
2.6.2.	SETPARTAB - Set parameter table	56
2.6.3.	SET - Set field info	58
2.6.4.	SETTIME - Set time	59
2.6.5.	CHANGE - Change field header	61
2.6.6.	SETGRID - Set grid information	62
2.6.7.	SETZAXIS - Set z-axis information	63
2.6.8.	INVERT - Invert latitudes	64
2.6.9.	INVERTLEV - Invert levels	64
2.6.10.	SHIFTXY - Shift field	65
2.6.11.	MASKREGION - Mask regions	66
2.6.12.	MASKBOX - Mask a box	67
2.6.13.	SETBOX - Set a box to constant	68
2.6.14.	ENLARGE - Enlarge fields	69
2.6.15.	SETMISS - Set missing value	70
2.7.	Arithmetic	72
2.7.1.	EXPR - Evaluate expressions	74
2.7.2.	MATH - Mathematical functions	77
2.7.3.	ARITHC - Arithmetic with a constant	78
2.7.4.	ARITH - Arithmetic on two datasets	79
2.7.5.	MONARITH - Monthly arithmetic	80
2.7.6.	YHOURLARITH - Multi-year hourly arithmetic	81
2.7.7.	YDAYARITH - Multi-year daily arithmetic	82
2.7.8.	YMONARITH - Multi-year monthly arithmetic	83
2.7.9.	YSEASARITH - Multi-year seasonal arithmetic	84
2.7.10.	ARITHDAYS - Arithmetic with days	84
2.8.	Statistical values	85
2.8.1.	TIMCUMSUM - Cumulative sum over all timesteps	91
2.8.2.	CONSECSTAT - Consecutive timestep periods	91
2.8.3.	ENSSTAT - Statistical values over an ensemble	92
2.8.4.	ENSSTAT2 - Statistical values over an ensemble	94
2.8.5.	ENSVAL - Ensemble validation tools	95
2.8.6.	FLDSTAT - Statistical values over a field	97
2.8.7.	ZONSTAT - Zonal statistical values	99
2.8.8.	MERSTAT - Meridional statistical values	100
2.8.9.	GRIDBOXSTAT - Statistical values over grid boxes	102
2.8.10.	VERTSTAT - Vertical statistical values	103
2.8.11.	TIMSELSTAT - Time range statistical values	104
2.8.12.	TIMSELPCTL - Time range percentile values	105
2.8.13.	RUNSTAT - Running statistical values	106
2.8.14.	RUNPCTL - Running percentile values	107
2.8.15.	TIMSTAT - Statistical values over all timesteps	108
2.8.16.	TIMPCTL - Percentile values over all timesteps	109
2.8.17.	HOURLSTAT - Hourly statistical values	110
2.8.18.	HOURLPCTL - Hourly percentile values	111
2.8.19.	DAYSTAT - Daily statistical values	112
2.8.20.	DAYPCTL - Daily percentile values	113

2.8.21. MONSTAT - Monthly statistical values	114
2.8.22. MONPCTL - Monthly percentile values	115
2.8.23. YEARMONSTAT - Yearly mean from monthly data	116
2.8.24. YEARSTAT - Yearly statistical values	117
2.8.25. YEARPCTL - Yearly percentile values	118
2.8.26. SEASSTAT - Seasonal statistical values	119
2.8.27. SEASPCTL - Seasonal percentile values	120
2.8.28. YHOURLSTAT - Multi-year hourly statistical values	121
2.8.29. YDAYSTAT - Multi-year daily statistical values	123
2.8.30. YDAYPCTL - Multi-year daily percentile values	125
2.8.31. YMONSTAT - Multi-year monthly statistical values	126
2.8.32. YMONPCTL - Multi-year monthly percentile values	128
2.8.33. YSEASSTAT - Multi-year seasonal statistical values	129
2.8.34. YSEASPCTL - Multi-year seasonal percentile values	131
2.8.35. YDRUNSTAT - Multi-year daily running statistical values	132
2.8.36. YDRUNPCTL - Multi-year daily running percentile values	134
2.9. Correlation and co.	135
2.9.1. FLDCOR - Correlation in grid space	136
2.9.2. TIMCOR - Correlation over time	136
2.9.3. FLDCOVAR - Covariance in grid space	137
2.9.4. TIMCOVAR - Covariance over time	137
2.10. Regression	138
2.10.1. REGRES - Regression	139
2.10.2. DETREND - Detrend time series	139
2.10.3. TREND - Trend of time series	140
2.10.4. SUBTREND - Subtract a trend	140
2.11. EOFs	141
2.11.1. EOFS - Empirical Orthogonal Functions	142
2.11.2. EOFCOEFF - Principal coefficients of EOFs	144
2.12. Interpolation	145
2.12.1. REMAPBIL - Bilinear interpolation	146
2.12.2. REMAPBIC - Bicubic interpolation	147
2.12.3. REMAPNN - Nearest neighbor remapping	148
2.12.4. REMAPDIS - Distance-weighted average remapping	149
2.12.5. REMAPYCON - First order conservative remapping	150
2.12.6. REMAPCON - First order conservative remapping	152
2.12.7. REMAPCON2 - Second order conservative remapping	154
2.12.8. REMAPLAF - Largest area fraction remapping	156
2.12.9. REMAP - Grid remapping	157
2.12.10. REMAPETA - Remap vertical hybrid level	158
2.12.11. VERTINTML - Vertical interpolation	160
2.12.12. VERTINTAP - Vertical interpolation	160
2.12.13. INTLEVEL - Linear level interpolation	161
2.12.14. INTLEVEL3D - Linear level interpolation from/to 3d vertical coordinates	163
2.12.15. INTTIME - Time interpolation	164
2.12.16. INTYEAR - Year interpolation	165
2.13. Transformation	166
2.13.1. SPECTRAL - Spectral transformation	167
2.13.2. WIND - Wind transformation	168
2.14. Import/Export	169
2.14.1. IMPORTBINARY - Import binary data sets	170
2.14.2. IMPORTCMSAF - Import CM-SAF HDF5 files	171
2.14.3. IMPORTAMSR - Import AMSR binary files	172
2.14.4. INPUT - Formatted input	173
2.14.5. OUTPUT - Formatted output	174
2.14.6. OUTPUTTAB - Table output	175
2.14.7. OUTPUTGMT - GMT output	176

2.15. Miscellaneous	178
2.15.1. GRADSDES - GrADS data descriptor file	180
2.15.2. AFTERBURNER - ECHAM standard post processor	181
2.15.3. FILTER - Time series filtering	183
2.15.4. GRIDCELL - Grid cell quantities	184
2.15.5. SMOOTH - Smooth grid points	185
2.15.6. REPLACEVALUES - Replace variable values	186
2.15.7. TIMSORT - Timsort	187
2.15.8. VARGEN - Generate a field	187
2.15.9. WINDTRANS - Wind Transformation	189
2.15.10. ROTUVB - Rotation	190
2.15.11. MASTRFU - Mass stream function	190
2.15.12. DERIVEPAR - Sea level pressure	190
2.15.13. ADISIT - Potential temperature to in-situ temperature and vice versa	191
2.15.14. RHOPOT - Calculates potential density	191
2.15.15. HISTOGRAM - Histogram	192
2.15.16. SETHALO - Set the left and right bounds of a field	192
2.15.17. WCT - Windchill temperature	193
2.15.18. FDNS - Frost days where no snow index per time period	193
2.15.19. STRWIN - Strong wind days index per time period	193
2.15.20. STRBRE - Strong breeze days index per time period	194
2.15.21. STRGAL - Strong gale days index per time period	194
2.15.22. HURR - Hurricane days index per time period	194
2.15.23. CMORLITE - CMOR lite	195
<b>3. Contributors</b>	<b>197</b>
3.1. History	197
3.2. External sources	197
3.3. Contributors	197
<b>A. Environment Variables</b>	<b>201</b>
<b>B. Parallelized operators</b>	<b>202</b>
<b>C. Standard name table</b>	<b>203</b>
<b>D. Grid description examples</b>	<b>204</b>
D.1. Example of a curvilinear grid description	204
D.2. Example description for an unstructured grid	205
<b>Operator index</b>	<b>206</b>

# 1. Introduction

The Climate Data Operators (**CDO**) software is a collection of many operators for standard processing of climate and forecast model data. The operators include simple statistical and arithmetic functions, data selection and subsampling tools, and spatial interpolation. **CDO** was developed to have the same set of processing functions for GRIB [GRIB] and NetCDF [NetCDF] datasets in one package.

The Climate Data Interface [CDI] is used for the fast and file format independent access to GRIB and NetCDF datasets. The local MPI-MET data formats SERVICE, EXTRA and IEG are also supported.

There are some limitations for GRIB and NetCDF datasets. A GRIB dataset has to be consistent, similar to NetCDF. That means all time steps need to have the same variables, and within a time step each variable may occur only once. NetCDF datasets are only supported for the classic data model and arrays up to 4 dimensions. These dimensions should only be used by the horizontal and vertical grid and the time. The NetCDF attributes should follow the GDT, COARDS or CF Conventions.

The user interface and some operators are similar to the PINGO [PINGO] package.

The main **CDO** features are:

- More than 700 operators available
- Modular design and easily extendable with new operators
- Very simple UNIX command line interface
- A dataset can be processed by several operators, without storing the interim results in files
- Most operators handle datasets with missing values
- Fast processing of large datasets
- Support of many different grid types
- Tested on many UNIX/Linux systems, Cygwin, and MacOS-X

## 1.1. Building from sources

This section describes how to build **CDO** from the sources on a UNIX system. **CDO** uses the GNU configure and build system for compilation. The only requirement is a working ISO C++11 and ANSI C99 compiler.

First go to the [download](https://code.zmaw.de/projects/cdo) page (<https://code.zmaw.de/projects/cdo>) to get the latest distribution, if you do not have it yet.

To take full advantage of **CDO** features the following additional libraries should be installed:

- Unidata [NetCDF](https://www.unidata.ucar.edu/software/netcdf) library (<https://www.unidata.ucar.edu/software/netcdf>) version 3 or higher. This library is needed to process NetCDF [NetCDF] files with **CDO**.
- ECMWF [ecCodes](https://software.ecmwf.int/wiki/display/ECC/ecCodes+Home) library (<https://software.ecmwf.int/wiki/display/ECC/ecCodes+Home>) version 2.3.0 or higher. This library is needed to process GRIB2 files with **CDO**.
- HDF5 [szip](http://www.hdfgroup.org/doc_resource/SZIP) library ([http://www.hdfgroup.org/doc\\_resource/SZIP](http://www.hdfgroup.org/doc_resource/SZIP)) version 2.1 or higher. This library is needed to process szip compressed GRIB [GRIB] files with **CDO**.
- [HDF5](http://www.hdfgroup.org/HDF5) library (<http://www.hdfgroup.org/HDF5>) version 1.6 or higher. This library is needed to import CM-SAF [CM-SAF] HDF5 files with the **CDO** operator `import_cmsaf`.

- **PROJ.4** library (<http://trac.osgeo.org/proj>) version 4.6 or higher.  
This library is needed to convert Sinusoidal and Lambert Azimuthal Equal Area coordinates to geographic coordinates, for e.g. remapping.
- **Magics** library (<https://software.ecmwf.int/wiki/display/MAGP/Magics>) version 2.18 or higher.  
This library is needed to create contour, vector and graph plots with **CDO**.

**CDO** is a multi-threaded application. Therefor all the above libraries should be compiled thread safe. Using non-threadsafe libraries could cause unexpected errors!

### 1.1.1. Compilation

Compilation is done by performing the following steps:

1. Unpack the archive, if you haven't done that yet:

```
gunzip cdo-$VERSION.tar.gz      # uncompress the archive
tar xf cdo-$VERSION.tar         # unpack it
cd cdo-$VERSION
```

2. Run the configure script:

```
./configure
```

- Optionally with NetCDF [\[NetCDF\]](#) support:

```
./configure --with-netcdf=<NetCDF root directory>
```

- and with ecCodes:

```
./configure --with-ecCodes=<ecCodes root directory>
```

For an overview of other configuration options use

```
./configure --help
```

3. Compile the program by running make:

```
make
```

The program should compile without problems and the binary (`cdo`) should be available in the `src` directory of the distribution.

### 1.1.2. Installation

After the compilation of the source code do a `make install`, possibly as root if the destination permissions require that.

```
make install
```

The binary is installed into the directory `<prefix>/bin`. `<prefix>` defaults to `/usr/local` but can be changed with the `-prefix` option of the configure script.

Alternatively, you can also copy the binary from the `src` directory manually to some `bin` directory in your search path.

## 1.2. Usage

This section describes how to use **CDO**. The syntax is:

```
cdo [ Options ] Operator1 [ -Operator2 [ -OperatorN ] ]
```

### 1.2.1. Options

All options have to be placed before the first operator. The following options are available for all operators:

- a Generate an absolute time axis.  
 -b <nbits> Set the number of bits for the output precision. The valid precisions depend on the file format:

<format>	<nbits>
grb1, grb2	P1 - P24
nc1, nc2, nc4, nc4c	I8/I16/I32/F32/F64
grb2, srv, ext, ieg	F32/F64

For srv, ext and ieg format the letter L or B can be added to set the byteorder to Little or Big endian.

- cmor CMOR conform NetCDF output.  
 -C, --color Colorized output messages.  
 -f <format> Set the output file format. The valid file formats are:

File format	<format>
GRIB version 1	grb1/grb
GRIB version 2	grb2
NetCDF	nc1
NetCDF version 2 (64-bit)	nc2/nc
NetCDF-4 (HDF5)	nc4
NetCDF-4 classic	nc4c
SERVICE	srv
EXTRA	ext
IEG	ieg

GRIB2 is only available if **CDO** was compiled with GRIB\_API support and all NetCDF file types are only available if **CDO** was compiled with NetCDF support!

- g <grid> Define the default grid description by name or from file (see chapter 1.3 on page 11). Available grid names are: r<NX>x<NY>, lon=<LON>/lat=<LAT>, n<N>, gme<NI>  
 -h, --help Help information for the operators.  
 --no\_history Do not append to NetCDF *history* global attribute.  
 --netcdf\_hdr\_pad, --hdr\_pad, --header\_pad <nbr> Pad NetCDF output header with *nbr* bytes.  
 -k <chunktype> NetCDF4 chunk type: auto, grid or lines.  
 -L Lock I/O (sequential access).  
 -M Switch to indicate that the I/O streams have missing values.  
 -m <missval> Set the missing value of non NetCDF files (default: -9e+33).  
 -O Overwrite existing output file, if checked.  
 Existing output file is checked only for: ens<STAT>, merge, mergetime  
 --operators List of all operators.  
 -P <nthreads> Set number of OpenMP threads (Only available if OpenMP support was compiled in).  
 --percentile <method> Percentile method: nrank nist numpy numpy\_lower numpy\_higher numpy\_nearest  
 --reduce\_dim Reduce NetCDF dimensions (module: TIMSTAT, FLDSTAT).  
 -R, --regular Convert GRIB1 data from reduced to regular grid (only with cgribex lib).  
 -r Generate a relative time axis.  
 -S Create an extra output stream for the module TIMSTAT. This stream contains the number of non missing values for each output period.  
 -s, --silent Silent mode.  
 --sort Alphanumeric sorting of NetCDF parameter names.  
 -t <partab> Set the GRIB1 (cgribex) default parameter table name or file (see chapter 1.6 on page 15).  
 Predefined tables are: echam4 echam5 echam6 mpiom1 ecmwf remo  
 --timestat\_date <srcdate> Target timestamp (time statistics): first, middle, midhigh or last source timestep.



<code>-V, --version</code>	Print the version number.
<code>-v, --verbose</code>	Print extra details for some operators.
<code>-W</code>	Print extra warning messages.
<code>-z szip</code>	SZIP compression of GRIB1 records.
<code>jpeg</code>	JPEG compression of GRIB2 records.
<code>zip[_1-9]</code>	Deflate compression of NetCDF4 variables.

### 1.2.2. Environment variables

There are some environment variables which influence the behavior of **CDO**. An incomplete list can be found in [Appendix A](#).

Here is an example to set the environment variable `CDO_RESET_HISTORY` for different shells:

```
Bourne shell (sh):  CDO_RESET_HISTORY=1 ; export CDO_RESET_HISTORY
Korn shell (ksh):   export CDO_RESET_HISTORY=1
C shell (csh):      setenv CDO_RESET_HISTORY 1
```

### 1.2.3. Operators

There are more than 700 operators available. A detailed description of all operators can be found in the [Reference Manual](#) section.

### 1.2.4. Operator chaining

All operators with a fixed number of input streams and one output stream can pipe the result directly to another operator. The operator must begin with "-", in order to combine it with others. This can improve the performance by:

- reducing unnecessary disk I/O
- parallel processing

Use

```
cdo sub -dayavg infile2 -timavg infile1 outfile
```

instead of

```
cdo timavg infile1 tmp1
cdo dayavg infile2 tmp2
cdo sub tmp2 tmp1 outfile
rm tmp1 tmp2
```

**Note:** Operator chaining is implemented over POSIX Threads (pthreads). Therefore this **CDO** feature is not available on operating systems without POSIX Threads support!

All operators with an arbitrary number of input streams (infiles) can't be combined with other operators if these operators are used with more than one input stream. Here is an incomplete list of these operators: [copy](#), [cat](#), [merge](#), [mergetime](#), [select](#), [ens<STAT>](#)

Use single quotes if the input stream names are generated with wildcards. In this case CDO will do the pattern matching and the output can be combined with other operators. Here is an example for this feature:

```
cdo timavg -select,name=temperature 'infile?' outfile
```

The CDO internal wildcard expansion is using the *glob()* function. Therefore internal wildcard expansion is not available on operating systems without the *glob()* function!

All operators with one input stream will process only one input stream! You need to take care when mixing those operators with operator with an arbitrary number of input streams. The following examples illustrate this problem.

1. `cdo info -timavg infile?`
2. `cdo info -timavg infile1 infile2`
3. `cdo timavg infile1 tmpfile`  
`cdo info tmpfile infile2`

All three examples produce identical results. The time average will be computed only on the first input file.

### 1.2.5. Parallelized operators

Some of the **CDO** operators are shared memory parallelized with OpenMP. An OpenMP-enabled C compiler is needed to use this feature. Users may request a specific number of OpenMP threads `nthreads` with the `'-P'` switch.

Here is an example to distribute the bilinear interpolation on 8 OpenMP threads:

```
cdo -P 8 remapbil,targetgrid infile outfile
```

Many **CDO** operators are I/O-bound. This means most of the time is spent in reading and writing the data. Only compute intensive **CDO** operators are parallelized. An incomplete list of OpenMP parallelized operators can be found in [Appendix B](#).

### 1.2.6. Operator parameter

Some operators need one or more parameter. A list of parameter is indicated by the separator `','`.

- **STRING**

Unquoted characters without blanks and tabs. The following command select variables with the name `pressure` and `tsurf`:

```
cdo selvar,pressure,tsurf infile outfile
```

- **FLOAT**

Floating point number in any representation. The following command sets the range between 0 and 273.15 of all fields to missing value:

```
cdo setrtomiss,0,273.15 infile outfile
```

- **INTEGER**

A range of integer parameter can be specified by *first/last[/inc]*. To select the days 5, 6, 7, 8 and 9 use:

```
cdo selday,5/9 infile outfile
```

The result is the same as:

```
cdo selday,5,6,7,8,9 infile outfile
```

## 1.3. Horizontal grids

Physical quantities of climate models are typically stored on a horizontal grid. The maximum number of supported grid cells is 2147483647 (`INT_MAX`). This corresponds to a global regular lon/lat grid with 65455x32727 grid cells and a global resolution of 0.0055 degree.

### 1.3.1. Grid area weights

One single point of a horizontal grid represents the mean of a grid cell. These grid cells are typically of different sizes, because the grid points are of varying distance.

Area weights are individual weights for each grid cell. They are needed to compute the area weighted mean or variance of a set of grid cells (e.g. [fldmean](#) - the mean value of all grid cells). In **CDO** the area weights are derived from the grid cell area. If the cell area is not available then it will be computed from the geographical coordinates via spherical triangles. This is only possible if the geographical coordinates of the grid cell corners are available or derivable. Otherwise **CDO** gives a warning message and uses constant area weights for all grid cells.

The cell area is read automatically from a NetCDF input file if a variable has the corresponding “cell\_measures” attribute, e.g.:

```
var:cell_measures = "area: cell_area" ;
```

If the computed cell area is not desired then the **CDO** operator [setgridarea](#) can be used to set or overwrite the grid cell area.

### 1.3.2. Grid description

In the following situations it is necessary to give a description of a horizontal grid:

- Changing the grid description (operator: [setgrid](#))
- Horizontal interpolation (all remapping operators)
- Generating of variables (operator: [const](#), [random](#))

As now described, there are several possibilities to define a horizontal grid.

#### 1.3.2.1. Predefined grids

Predefined grids are available for global regular, gaussian or icosahedral-hexagonal GME grids.

##### Global regular grid: `global_<DXY>`

`global_<DXY>` defines a global regular lon/lat grid. The grid increment `<DXY>` can be selected at will. The longitudes start at  $\langle DXY \rangle / 2 - 180^\circ$  and the latitudes start at  $\langle DXY \rangle / 2 - 90^\circ$ .

##### Global regular grid: `r<NX>x<NY>`

`r<NX>x<NY>` defines a global regular lon/lat grid. The number of the longitudes `<NX>` and the latitudes `<NY>` can be selected at will. The longitudes start at  $0^\circ$  with an increment of  $(360/\langle NX \rangle)^\circ$ . The latitudes go from south to north with an increment of  $(180/\langle NY \rangle)^\circ$ .

##### One grid point: `lon=<LON>/lat=<LAT>`

`lon=<LON>/lat=<LAT>` defines a lon/lat grid with only one grid point.

##### Global Gaussian grid: `n<N>`

`n<N>` defines a global Gaussian grid. `N` specifies the number of latitudes lines between the Pole and the Equator. The longitudes start at  $0^\circ$  with an increment of  $(360/nlon)^\circ$ . The gaussian latitudes go from north to south.

### Global icosahedral-hexagonal GME grid: `gme<NI>`

`gme<NI>` defines a global icosahedral-hexagonal GME grid. `NI` specifies the number of intervals on a main triangle side.

#### 1.3.2.2. Grids from data files

You can use the grid description from an other datafile. The format of the datafile and the grid of the data field must be supported by **CDO**. Use the operator '[sinfo](#)' to get short informations about your variables and the grids. If there are more then one grid in the datafile the grid description of the first variable will be used.

#### 1.3.2.3. SCRIP grids

SCRIP (Spherical Coordinate Remapping and Interpolation Package) uses a common grid description for curvilinear and unstructured grids. For more information about the convention see [[SCRIP](#)]. This grid description is stored in NetCDF. Therefor it is only available if **CDO** was compiled with NetCDF support!

SCRIP grid description example of a curvilinear MPIOM [[MPIOM](#)] GROB3 grid (only the NetCDF header):

```
netcdf grob3s {
  dimensions:
    grid_size = 12120 ;
    grid_xsize = 120 ;
    grid_ysize = 101 ;
    grid_corners = 4 ;
    grid_rank = 2 ;
  variables:
    int grid_dims(grid_rank) ;
    float grid_center_lat(grid_ysize, grid_xsize) ;
      grid_center_lat:units = "degrees" ;
      grid_center_lat:bounds = "grid_corner_lat" ;
    float grid_center_lon(grid_ysize, grid_xsize) ;
      grid_center_lon:units = "degrees" ;
      grid_center_lon:bounds = "grid_corner_lon" ;
    int grid_imask(grid_ysize, grid_xsize) ;
      grid_imask:units = "unitless" ;
      grid_imask:coordinates = "grid_center_lon grid_center_lat" ;
    float grid_corner_lat(grid_ysize, grid_xsize, grid_corners) ;
      grid_corner_lat:units = "degrees" ;
    float grid_corner_lon(grid_ysize, grid_xsize, grid_corners) ;
      grid_corner_lon:units = "degrees" ;

    // global attributes:
      :title = "grob3s" ;
}
```

#### 1.3.2.4. CDO grids

All supported grids can also be described with the **CDO** grid description. The following keywords can be used to describe a grid:

Keyword	Datatype	Description
<b>gridtype</b>	STRING	Type of the grid (gaussian, lonlat, curvilinear, unstructured).
<b>gridsize</b>	INTEGER	Size of the grid.
<b>xsize</b>	INTEGER	Size in x direction (number of longitudes).
<b>ysize</b>	INTEGER	Size in y direction (number of latitudes).
<b>xvals</b>	FLOAT ARRAY	X values of the grid cell center.
<b>yvals</b>	FLOAT ARRAY	Y values of the grid cell center.
<b>nvertex</b>	INTEGER	Number of the vertices for all grid cells.
<b>xbounds</b>	FLOAT ARRAY	X bounds of each gridbox.
<b>ybounds</b>	FLOAT ARRAY	Y bounds of each gridbox.
<b>xfirst, xinc</b>	FLOAT, FLOAT	Macros to define xvals with a constant increment, xfirst is the x value of the first grid cell center.
<b>yfirst, yinc</b>	FLOAT, FLOAT	Macros to define yvals with a constant increment, yfirst is the y value of the first grid cell center.
<b>xunits</b>	STRING	units of the x axis
<b>yunits</b>	STRING	units of the y axis

Which keywords are necessary depends on the gridtype. The following table gives an overview of the default values or the size with respect to the different grid types.

gridtype	lonlat	gaussian	projection	curvilinear	unstructured
gridsize	xsize*ysize	xsize*ysize	xsize*ysize	xsize*ysize	<b>ncell</b>
xsize	<b>nlon</b>	<b>nlon</b>	<b>nx</b>	<b>nlon</b>	gridsize
ysize	<b>nlat</b>	<b>nlat</b>	<b>ny</b>	<b>nlat</b>	gridsize
xvals	xsize	xsize	xsize	gridsize	gridsize
yvals	ysize	ysize	ysize	gridsize	gridsize
nvertex	2	2	2	4	<b>nv</b>
xbounds	2*xsize	2*xsize	2*xsize	4*gridsize	nv*gridsize
ybounds	2*ysize	2*ysize	2*xsize	4*gridsize	nv*gridsize
xunits	degrees	degrees	m	degrees	degrees
yunits	degrees	degrees	m	degrees	degrees

The keywords nvertex, xbounds and ybounds are optional if area weights are not needed. The grid cell corners xbounds and ybounds have to rotate counterclockwise.

**CDO** grid description example of a T21 gaussian grid:

```

gridtype = gaussian
xsize    = 64
ysize    = 32
xfirst   = 0
xinc     = 5.625
yvals    = 85.76  80.27  74.75  69.21  63.68  58.14  52.61  47.07
           41.53  36.00  30.46  24.92  19.38  13.84  8.31  2.77
           -2.77 -8.31 -13.84 -19.38 -24.92 -30.46 -36.00 -41.53
           -47.07 -52.61 -58.14 -63.68 -69.21 -74.75 -80.27 -85.76

```

**CDO** grid description example of a global regular grid with 60x30 points:

```

gridtype = lonlat
xsize    = 60
ysize    = 30
xfirst   = -177
xinc     = 6
yfirst   = -87
yinc     = 6

```

**CDO** grid description example of a regional rotated lon/lat grid:

```

gridtype = projection
xsize    = 81
ysize    = 91
xunits    = "degrees"
yunits    = "degrees"
xfirst    = -19.5
xinc      = 0.5
yfirst    = -25.0
yinc      = 0.5
grid_mapping_name = rotated_latitude_longitude
grid_north_pole_longitude = -170
grid_north_pole_latitude = 32.5

```

Example **CDO** descriptions of a curvilinear and an unstructured grid can be found in [Appendix C](#).

## 1.4. Z-axis description

Sometimes it is necessary to change the description of a z-axis. This can be done with the operator [setzaxis](#). This operator needs an ASCII formatted file with the description of the z-axis. The following keywords can be used to describe a z-axis:

Keyword	Datatype	Description
<b>zaxistype</b>	STRING	type of the z-axis
<b>size</b>	INTEGER	number of levels
<b>levels</b>	FLOAT ARRAY	values of the levels
<b>lbounds</b>	FLOAT ARRAY	lower level bounds
<b>ubounds</b>	FLOAT ARRAY	upper level bounds
<b>vctsize</b>	INTEGER	number of vertical coordinate parameters
<b>vct</b>	FLOAT ARRAY	vertical coordinate table

The keywords **lbounds** and **ubounds** are optional. **vctsize** and **vct** are only necessary to define hybrid model levels.

Available z-axis types:

Z-axis type	Description	Units
<b>surface</b>	Surface	
<b>pressure</b>	Pressure level	pascal
<b>hybrid</b>	Hybrid model level	
<b>height</b>	Height above ground	meter
<b>depth_below_sea</b>	Depth below sea level	meter
<b>depth_below_land</b>	Depth below land surface	centimeter
<b>isentropic</b>	Isentropic (theta) level	kelvin

Z-axis description example for pressure levels 100, 200, 500, 850 and 1000 hPa:

```

zaxistype = pressure
size      = 5
levels    = 10000 20000 50000 85000 100000

```

Z-axis description example for ECHAM5 L19 hybrid model levels:

```

zaxistype = hybrid
size      = 19
levels    = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
vctsize   = 40
vct       = 0 2000 4000 6046.10938 8267.92578 10609.5117 12851.1016 14698.5
           15861.125 16116.2383 15356.9258 13621.4609 11101.5625 8127.14453
           5125.14062 2549.96875 783.195068 0 0 0

```

```

0 0 0 0.000338993268 0.00335718691 0.0130700432 0.0340771675
0.0706498027 0.12591666 0.201195419 0.295519829 0.405408859
0.524931908 0.646107674 0.759697914 0.856437683 0.928747177
0.972985268 0.992281914 1

```

Note that the `vctsize` is twice the number of levels plus two and the vertical coordinate table must be specified for the level interfaces.

## 1.5. Time axis

A time axis describes the time for every timestep. Two time axis types are available: absolute time and relative time axis. **CDO** tries to maintain the actual type of the time axis for all operators.

### 1.5.1. Absolute time

An absolute time axis has the current time to each time step. It can be used without knowledge of the calendar. This is preferably used by climate models. In NetCDF files the absolute time axis is represented by the unit of the time: "day as %Y%m%d.%f".

### 1.5.2. Relative time

A relative time is the time relative to a fixed reference time. The current time results from the reference time and the elapsed interval. The result depends on the calendar used. **CDO** supports the standard Gregorian, proleptic Gregorian, 360 days, 365 days and 366 days calendars. The relative time axis is preferably used by numerical weather prediction models. In NetCDF files the relative time axis is represented by the unit of the time: "*time-units* since *reference-time*", e.g "days since 1989-6-15 12:00".

### 1.5.3. Conversion of the time

Some programs which work with NetCDF data can only process relative time axes. Therefore it may be necessary to convert from an absolute into a relative time axis. This conversion can be done for each operator with the **CDO** option '`-r`'. To convert a relative into an absolute time axis use the **CDO** option '`-a`'.

## 1.6. Parameter table

A parameter table is an ASCII formatted file to convert code numbers to variable names. Each variable has one line with its code number, name and a description with optional units in a blank separated list. It can only be used for GRIB, SERVICE, EXTRA and IEG formatted files. The **CDO** option '`-t <partab>`' sets the default parameter table for all input files. Use the operator '`setpartab`' to set the parameter table for a specific file.

Example of a **CDO** parameter table:

```

134  aps      surface pressure [Pa]
141  sn       snow depth [m]
147  ahfl     latent heat flux [W/m**2]
172  slm      land sea mask
175  albedo   surface albedo
211  siced    ice depth [m]

```

## 1.7. Missing values

Most operators can handle missing values. The default missing value for GRIB, SERVICE, EXTRA and IEG files is  $-9.e^{33}$ . The **CDO** option '-m <missval>' overwrites the default missing value. In NetCDF files the variable attribute '\_FillValue' is used as a missing value. The operator '[setmissval](#)' can be used to set a new missing value.

The **CDO** use of the missing value is shown in the following tables, where one table is printed for each operation. The operations are applied to arbitrary numbers  $a$ ,  $b$ , the special case 0, and the missing value *miss*. For example the table named "addition" shows that the sum of an arbitrary number  $a$  and the missing value is the missing value, and the table named "multiplication" shows that 0 multiplied by missing value results in 0.

<b>addition</b>	b		miss
a	$a + b$		<i>miss</i>
miss	<i>miss</i>		<i>miss</i>
<b>subtraction</b>	b		miss
a	$a - b$		<i>miss</i>
miss	<i>miss</i>		<i>miss</i>
<b>multiplication</b>	b	0	miss
a	$a * b$	0	<i>miss</i>
0	0	0	0
miss	<i>miss</i>	0	<i>miss</i>
<b>division</b>	b	0	miss
a	$a/b$	<i>miss</i>	<i>miss</i>
0	0	<i>miss</i>	<i>miss</i>
miss	<i>miss</i>	<i>miss</i>	<i>miss</i>
<b>maximum</b>	b		miss
a	$\max(a, b)$		<i>a</i>
miss	<i>b</i>		<i>miss</i>
<b>minimum</b>	b		miss
a	$\min(a, b)$		<i>a</i>
miss	<i>b</i>		<i>miss</i>
<b>sum</b>	b		miss
a	$a + b$		<i>a</i>
miss	<i>b</i>		<i>miss</i>

The handling of missing values by the operations "minimum" and "maximum" may be surprising, but the definition given here is more consistent with that 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 statistical functions ignore missing values, treating them as not belonging to the sample, with the side-effect of a reduced sample size.

### 1.7.1. Mean and average

An artificial distinction is made between the notions mean and average. The mean is regarded as a statistical 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  $(1 + 2 + \text{miss} + 3)/4 = \text{miss}/4 = \text{miss}$ . If there are no missing values in the sample, the average and mean are identical.



## 1.8. Percentile

There is no standard definition of percentile. All definitions yield to similar results when the number of values is very large. The following percentile methods are available in **CDO**:

Percentile method	Description
nrank	Nearest Rank method, the default method used in <b>CDO</b>
nist	The primary method recommended by NIST
numpy	numpy.percentile with the option interpolation set to 'linear'
numpy_lower	numpy.percentile with the option interpolation set to 'lower'
numpy_higher	numpy.percentile with the option interpolation set to 'higher'
numpy_nearest	numpy.percentile with the option interpolation set to 'nearest'

The percentile method can be selected with the **CDO** option `--percentile`. The Nearest Rank method is the default percentile method in **CDO**.

The different percentile methods can lead to different results, especially for small number of data values. Consider the ordered list {15, 20, 35, 40, 50, 55}, which contains six data values. Here is the result for the 30th, 40th, 50th, 75th and 100th percentiles of this list using the different percentile methods:

Percentile P	nrank	nist	numpy	numpy lower	numpy higher	numpy nearest
30th	20	21.5	27.5	20	35	35
40th	35	32	35	35	35	35
50th	35	37.5	37.5	35	40	40
75th	50	51.25	47.5	40	50	50
100th	55	55	55	55	55	55

### 1.8.1. Percentile over timesteps

The amount of data for time series can be very large. All data values need to be held in memory to calculate the percentile. The percentile over timesteps uses a histogram algorithm, to limit the amount of required memory. The default number of histogram bins is 101. That means the histogram algorithm is used, when the dataset has more than 101 time steps. The default can be overridden by setting the environment variable `CDO_PCTL_NBINS` to a different value. The histogram algorithm is implemented only for the Nearest Rank method.

## 2. Reference manual

This section gives a description of all operators. Related operators are grouped to modules. For easier description all single input files are named `infile` or `infile1`, `infile2`, etc., and an arbitrary number of input files are named `infile`s. All output files are named `outfile` or `outfile1`, `outfile2`, etc. Further the following notion is introduced:

$i(t)$       Timestep  $t$  of `infile`

$i(t, x)$     Element number  $x$  of the field at timestep  $t$  of `infile`

$o(t)$       Timestep  $t$  of `outfile`

$o(t, x)$     Element number  $x$  of the field at timestep  $t$  of `outfile`

## 2.1. Information

This section contains modules to print information about datasets. All operators print there results to standard output.

Here is a short overview of all operators in this section:

<b>info</b>	Dataset information listed by parameter identifier
<b>infor</b>	Dataset information listed by parameter name
<b>map</b>	Dataset information and simple map
<b>sinfor</b>	Short information listed by parameter identifier
<b>sinfor</b>	Short information listed by parameter name
<b>diff</b>	Compare two datasets listed by parameter id
<b>diffn</b>	Compare two datasets listed by parameter name
<b>npar</b>	Number of parameters
<b>nlevel</b>	Number of levels
<b>nyear</b>	Number of years
<b>nmon</b>	Number of months
<b>ndate</b>	Number of dates
<b>ntime</b>	Number of timesteps
<b>ngridpoints</b>	Number of gridpoints
<b>ngrids</b>	Number of horizontal grids
<b>showformat</b>	Show file format
<b>showcode</b>	Show code numbers
<b>showname</b>	Show variable names
<b>showstdname</b>	Show standard names
<b>showlevel</b>	Show levels
<b>showltype</b>	Show GRIB level types
<b>showyear</b>	Show years
<b>showmon</b>	Show months
<b>showdate</b>	Show date information
<b>showtime</b>	Show time information
<b>showtimestamp</b>	Show timestamp
<b>partab</b>	Parameter table
<b>codetab</b>	Parameter code table
<b>griddes</b>	Grid description
<b>zaxisdes</b>	Z-axis description
<b>vct</b>	Vertical coordinate table

### 2.1.1. INFO - Information and simple statistics

#### Synopsis

```
<operator> infile
```

#### Description

This module writes information about the structure and contents of all input files to standard output. All input files need to have the same structure with the same variables on different timesteps. The information displayed depends on the chosen operator.

#### Operators

- info** Dataset information listed by parameter identifier  
Prints information and simple statistics for each field of all input datasets. For each field the operator prints one line with the following elements:
- Date and Time
  - Level, Gridsize and number of Missing values
  - Minimum, Mean and Maximum  
The mean value is computed without the use of area weights!
  - Parameter identifier
- infor** Dataset information listed by parameter name  
The same as operator [info](#) but using the name instead of the identifier to label the parameter.
- map** Dataset information and simple map  
Prints information, simple statistics and a map for each field of all input datasets. The map will be printed only for fields on a regular lon/lat grid.

#### Example

To print information and simple statistics for each field of a dataset use:

```
cdo infor infile
```

This is an example result of a dataset with one 2D parameter over 12 timesteps:

-1 :	Date	Time	Level	Size	Miss	:	Minimum	Mean	Maximum	:	Name
1 :	1987-01-31	12:00:00	0	2048	1361	:	232.77	266.65	305.31	:	SST
2 :	1987-02-28	12:00:00	0	2048	1361	:	233.64	267.11	307.15	:	SST
3 :	1987-03-31	12:00:00	0	2048	1361	:	225.31	267.52	307.67	:	SST
4 :	1987-04-30	12:00:00	0	2048	1361	:	215.68	268.65	310.47	:	SST
5 :	1987-05-31	12:00:00	0	2048	1361	:	215.78	271.53	312.49	:	SST
6 :	1987-06-30	12:00:00	0	2048	1361	:	212.89	272.80	314.18	:	SST
7 :	1987-07-31	12:00:00	0	2048	1361	:	209.52	274.29	316.34	:	SST
8 :	1987-08-31	12:00:00	0	2048	1361	:	210.48	274.41	315.83	:	SST
9 :	1987-09-30	12:00:00	0	2048	1361	:	210.48	272.37	312.86	:	SST
10 :	1987-10-31	12:00:00	0	2048	1361	:	219.46	270.53	309.51	:	SST
11 :	1987-11-30	12:00:00	0	2048	1361	:	230.98	269.85	308.61	:	SST
12 :	1987-12-31	12:00:00	0	2048	1361	:	241.25	269.94	309.27	:	SST

## 2.1.2. SINFO - Short information

### Synopsis

```
<operator> infile
```

### Description

This module writes information about the structure of infiles to standard output. `infile` is an arbitrary number of input files. All input files need to have the same structure with the same variables on different timesteps. The information displayed depends on the chosen operator.

### Operators

- sinfo** Short information listed by parameter identifier  
Prints short information of a dataset. The information is divided into 4 sections. Section 1 prints one line per parameter with the following information:
- institute and source
  - timestep type
  - number of levels and z-axis number
  - horizontal grid size and number
  - data type
  - parameter identifier
- Section 2 and 3 gives a short overview of all grid and vertical coordinates. And the last section contains short information of the time coordinate.
- sinfon** Short information listed by parameter name  
The same as operator [sinfo](#) but using the name instead of the identifier to label the parameter.

### Example

To print short information of a dataset use:

```
cdo sinfon infile
```

This is the result of an ECHAM5 dataset with 3 parameter over 12 timesteps:

```
-1 : Institut Source Steptype Levels Num Points Num Dtype : Name
 1 : MPIMET ECHAM5 constant 1 1 2048 1 F32 : GEOSP
 2 : MPIMET ECHAM5 instant 4 2 2048 1 F32 : T
 3 : MPIMET ECHAM5 instant 1 1 2048 1 F32 : TSURF
Grid coordinates :
 1 : gaussian : points=2048 (64x32) np=16
               longitude : 0 to 354.375 by 5.625 degrees_east circular
               latitude : 85.7606 to -85.7606 degrees_north
Vertical coordinates :
 1 : surface : levels=1
 2 : pressure : levels=4
               level : 92500 to 20000 Pa
Time coordinate : 12 steps
YYYY-MM-DD hh:mm:ss YYYY-MM-DD hh:mm:ss YYYY-MM-DD hh:mm:ss YYYY-MM-DD hh:mm:ss
1987-01-31 12:00:00 1987-02-28 12:00:00 1987-03-31 12:00:00 1987-04-30 12:00:00
1987-05-31 12:00:00 1987-06-30 12:00:00 1987-07-31 12:00:00 1987-08-31 12:00:00
1987-09-30 12:00:00 1987-10-31 12:00:00 1987-11-30 12:00:00 1987-12-31 12:00:00
```

### 2.1.3. DIFF - Compare two datasets field by field

#### Synopsis

```
<operator> infile1 infile2
```

#### Description

Compares the contents of two datasets field by field. The input datasets need to have the same structure and its fields need to have the same header information and dimensions.

#### Operators

**diff** Compare two datasets listed by parameter id  
Provides statistics on differences between two datasets. For each pair of fields the operator prints one line with the following information:

- Date and Time
- Level, Gridsize and number of Missing values
- Number of different values
- Occurrence of coefficient pairs with different signs (S)
- Occurrence of zero values (Z)
- Maxima of absolute difference of coefficient pairs
- Maxima of relative difference of non-zero coefficient pairs with equal signs
- Parameter identifier

$$Absdiff(t, x) = |i_1(t, x) - i_2(t, x)|$$

$$Reldiff(t, x) = \frac{|i_1(t, x) - i_2(t, x)|}{\max(|i_1(t, x)|, |i_2(t, x)|)}$$

**diffn** Compare two datasets listed by parameter name  
The same as operator [diff](#). Using the name instead of the identifier to label the parameter.

#### Example

To print the difference for each field of two datasets use:

```
cdo diffn infile1 infile2
```

This is an example result of two datasets with one 2D parameter over 12 timesteps:

	Date	Time	Level	Size	Miss	Diff	:	S	Z	Max_Absdiff	Max_Reldiff	:	Name
1	: 1987-01-31	12:00:00	0	2048	1361	273	:	F	F	0.00010681	4.1660e-07	:	SST
2	: 1987-02-28	12:00:00	0	2048	1361	309	:	F	F	6.1035e-05	2.3742e-07	:	SST
3	: 1987-03-31	12:00:00	0	2048	1361	292	:	F	F	7.6294e-05	3.3784e-07	:	SST
4	: 1987-04-30	12:00:00	0	2048	1361	183	:	F	F	7.6294e-05	3.5117e-07	:	SST
5	: 1987-05-31	12:00:00	0	2048	1361	207	:	F	F	0.00010681	4.0307e-07	:	SST
7	: 1987-07-31	12:00:00	0	2048	1361	317	:	F	F	9.1553e-05	3.5634e-07	:	SST
8	: 1987-08-31	12:00:00	0	2048	1361	219	:	F	F	7.6294e-05	2.8849e-07	:	SST
9	: 1987-09-30	12:00:00	0	2048	1361	188	:	F	F	7.6294e-05	3.6168e-07	:	SST
10	: 1987-10-31	12:00:00	0	2048	1361	297	:	F	F	9.1553e-05	3.5001e-07	:	SST
11	: 1987-11-30	12:00:00	0	2048	1361	234	:	F	F	6.1035e-05	2.3839e-07	:	SST
12	: 1987-12-31	12:00:00	0	2048	1361	267	:	F	F	9.3553e-05	3.7624e-07	:	SST
11 of 12 records differ													

## 2.1.4. NINFO - Print the number of parameters, levels or times

### Synopsis

`<operator> infile`

### Description

This module prints the number of variables, levels or times of the input dataset.

### Operators

<b>npar</b>	Number of parameters Prints the number of parameters (variables).
<b>nlevel</b>	Number of levels Prints the number of levels for each variable.
<b>nyear</b>	Number of years Prints the number of different years.
<b>nmon</b>	Number of months Prints the number of different combinations of years and months.
<b>ndate</b>	Number of dates Prints the number of different dates.
<b>ntime</b>	Number of timesteps Prints the number of timesteps.
<b>ngridpoints</b>	Number of gridpoints Prints the number of gridpoints for each variable.
<b>ngrids</b>	Number of horizontal grids Prints the number of horizontal grids.

### Example

To print the number of parameters (variables) in a dataset use:

```
cdo npar infile
```

To print the number of months in a dataset use:

```
cdo nmon infile
```

## 2.1.5. SHOWINFO - Show variables, levels or times

### Synopsis

```
<operator> infile
```

### Description

This module prints the format, variables, levels or times of the input dataset.

### Operators

<b>showformat</b>	Show file format Prints the file format of the input dataset.
<b>showcode</b>	Show code numbers Prints the code number of all variables.
<b>showname</b>	Show variable names Prints the name of all variables.
<b>showstdname</b>	Show standard names Prints the standard name of all variables.
<b>showlevel</b>	Show levels Prints all levels for each variable.
<b>showltype</b>	Show GRIB level types Prints the GRIB level type for all z-axes.
<b>showyear</b>	Show years Prints all years.
<b>showmon</b>	Show months Prints all months.
<b>showdate</b>	Show date information Prints date information of all timesteps (format YYYY-MM-DD).
<b>showtime</b>	Show time information Prints time information of all timesteps (format hh:mm:ss).
<b>showtimestamp</b>	Show timestamp Prints timestamp of all timesteps (format YYYY-MM-DDThh:mm:ss).

### Example

To print the code number of all variables in a dataset use:

```
cdo showcode infile
```

This is an example result of a dataset with three variables:

```
129 130 139
```

To print all months in a dataset use:

```
cdo showmon infile
```

This is an examples result of a dataset with an annual cycle:

```
1 2 3 4 5 6 7 8 9 10 11 12
```



## 2.1.6. FILEDES - Dataset description

### Synopsis

```
<operator> infile
```

### Description

This module provides operators to print meta information about a dataset. The printed meta-data depends on the chosen operator.

### Operators

<b>partab</b>	Parameter table Prints all available meta information of the variables.
<b>codetab</b>	Parameter code table Prints a code table with a description of all variables. For each variable the operator prints one line listing the code, name, description and units.
<b>griddes</b>	Grid description Prints the description of all grids.
<b>zaxisdes</b>	Z-axis description Prints the description of all z-axes.
<b>vct</b>	Vertical coordinate table Prints the vertical coordinate table.

### Example

Assume all variables of the dataset are on a Gaussssian N16 grid. To print the grid description of this dataset use:

```
cdo griddes infile
```

Result:

```
gridtype  : gaussian
gridsize  : 2048
xname     : lon
xlongname  : longitude
xunits    : degrees_east
yname     : lat
ylongname  : latitude
yunits    : degrees_north
xsize     : 64
ysize     : 32
xfirst    : 0
xinc      : 5.625
yvals     : 85.76058 80.26877 74.74454 69.21297 63.67863 58.1429 52.6065
            47.06964 41.53246 35.99507 30.4575 24.91992 19.38223 13.84448
            8.306702 2.768903 -2.768903 -8.306702 -13.84448 -19.38223
            -24.91992 -30.4575 -35.99507 -41.53246 -47.06964 -52.6065
            -58.1429 -63.67863 -69.21297 -74.74454 -80.26877 -85.76058
```

## 2.2. File operations

This section contains modules to perform operations on files.

Here is a short overview of all operators in this section:

<b>copy</b>	Copy datasets
<b>cat</b>	Concatenate datasets
<b>replace</b>	Replace variables
<b>duplicate</b>	Duplicates a dataset
<b>mergegrid</b>	Merge grid
<b>merge</b>	Merge datasets with different fields
<b>mergetime</b>	Merge datasets sorted by date and time
<b>splitcode</b>	Split code numbers
<b>splitparam</b>	Split parameter identifiers
<b>splitname</b>	Split variable names
<b>splitlevel</b>	Split levels
<b>splitgrid</b>	Split grids
<b>splitzaxis</b>	Split z-axes
<b>splittabnum</b>	Split parameter table numbers
<b>splithour</b>	Split hours
<b>splitday</b>	Split days
<b>splitseas</b>	Split seasons
<b>splityear</b>	Split years
<b>splityearmon</b>	Split in years and months
<b>splitmon</b>	Split months
<b>splitsel</b>	Split time selection
<b>distgrid</b>	Distribute horizontal grid
<b>collgrid</b>	Collect horizontal grid

### 2.2.1. COPY - Copy datasets

#### Synopsis

```
<operator> infile outfile
```

#### Description

This module contains operators to copy or concatenate datasets. `infile` is an arbitrary number of input files. All input files need to have the same structure with the same variables on different timesteps.

#### Operators

- |             |   |
|-------------|---|
| <b>copy</b> | Copy datasets<br>Copies all input datasets to outfile.  |
| <b>cat</b>  | Concatenate datasets<br>Concatenates all input datasets and appends the result to the end of outfile. If outfile does not exist it will be created. |

#### Example

To change the format of a dataset to NetCDF use:

```
cdo -f nc copy infile outfile.nc
```

Add the option `'-r'` to create a relative time axis, as is required for proper recognition by GrADS or Ferret:

```
cdo -r -f nc copy infile outfile.nc
```

To concatenate 3 datasets with different timesteps of the same variables use:

```
cdo copy infile1 infile2 infile3 outfile
```

If the output dataset already exists and you wish to extend it with more timesteps use:

```
cdo cat infile1 infile2 infile3 outfile
```

### 2.2.2. REPLACE - Replace variables

#### Synopsis

```
replace infile1 infile2 outfile
```

#### Description

The replace operator replaces variables in `infile1` by variables from `infile2` and write the result to `outfile`. Both input datasets need to have the same number of timesteps.

#### Example

Assume the first input dataset `infile1` has three variables with the names `geosp`, `t` and `tslm1` and the second input dataset `infile2` has only the variable `tslm1`. To replace the variable `tslm1` in `infile1` by `tslm1` from `infile2` use:

```
cdo replace infile1 infile2 outfile
```

### 2.2.3. DUPLICATE - Duplicates a dataset

#### Synopsis

```
duplicate[,ndup] infile outfile
```

#### Description

This operator duplicates the contents of `infile` and writes the result to `outfile`. The optional parameter sets the number of duplicates, the default is 2.

#### Parameter

`ndup`     INTEGER     Number of duplicates, default is 2.

### 2.2.4. MERGEGRID - Merge grid

#### Synopsis

```
mergegrid infile1 infile2 outfile
```

#### Description

Merges grid points of all variables from `infile2` to `infile1` and write the result to `outfile`. Only the non missing values of `infile2` will be used. The horizontal grid of `infile2` should be smaller or equal to the grid of `infile1` and the resolution must be the same. Only rectilinear grids are supported. Both input files need to have the same variables and the same number of timesteps.

## 2.2.5. MERGE - Merge datasets

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module reads datasets from several input files, merges them and writes the resulting dataset to outfile.

### Operators

<b>merge</b>	<p>Merge datasets with different fields</p> <p>Merges time series of different fields from several input datasets. The number of fields per timestep written to outfile is the sum of the field numbers per timestep in all input datasets. The time series on all input datasets are required to have different fields and the same number of timesteps. The fields in each different input file either have to be different variables or different levels of the same variable. A mixture of different variables on different levels in different input files is not allowed.</p>
<b>mergetime</b>	<p>Merge datasets sorted by date and time</p> <p>Merges all timesteps of all input files sorted by date and time. All input files need to have the same structure with the same variables on different timesteps. After this operation every input timestep is in outfile and all timesteps are sorted by date and time.</p>

### Environment

SKIP_SAME_TIME	If set to 1, skips all consecutive timesteps with a double entry of the same timestamp.
----------------	---

### Note

The operators in this module need to open all input files simultaneously. The maximum number of open files depends on the operating system!

### Example

Assume three datasets with the same number of timesteps and different variables in each dataset. To merge these datasets to a new dataset use:

```
cdo merge infile1 infile2 infile3 outfile
```

Assume you split a 6 hourly dataset with [splithour](#). This produces four datasets, one for each hour. The following command merges them together:

```
cdo mergetime infile1 infile2 infile3 infile4 outfile
```

## 2.2.6. SPLIT - Split a dataset

### Synopsis

```
<operator>[,params] infile obase
```

### Description

This module splits `infile` into pieces. The output files will be named `<obase><xxx><suffix>` where `suffix` is the filename extension derived from the file format. `xxx` and the contents of the output files depends on the chosen operator. `params` is a comma separated list of processing parameters.

### Operators

<b>splitcode</b>	Split code numbers Splits a dataset into pieces, one for each different code number. <code>xxx</code> will have three digits with the code number.
<b>splitparam</b>	Split parameter identifiers Splits a dataset into pieces, one for each different parameter identifier. <code>xxx</code> will be a string with the parameter identifier.
<b>splitname</b>	Split variable names Splits a dataset into pieces, one for each variable name. <code>xxx</code> will be a string with the variable name.
<b>splitlevel</b>	Split levels Splits a dataset into pieces, one for each different level. <code>xxx</code> will have six digits with the level.
<b>splitgrid</b>	Split grids Splits a dataset into pieces, one for each different grid. <code>xxx</code> will have two digits with the grid number.
<b>splitzaxis</b>	Split z-axes Splits a dataset into pieces, one for each different z-axis. <code>xxx</code> will have two digits with the z-axis number.
<b>splittabnum</b>	Split parameter table numbers Splits a dataset into pieces, one for each GRIB1 parameter table number. <code>xxx</code> will have three digits with the GRIB1 parameter table number.

### Parameter

<code>swap</code>	STRING	Swap the position of <code>obase</code> and <code>xxx</code> in the output filename
<code>uuid=&lt;attname&gt;</code>	STRING	Add a UUID as global attribute <code>&lt;attname&gt;</code> to each output file

### Environment

<code>CDO_FILE_SUFFIX</code>	Set the default file suffix. This suffix will be added to the output file names instead of the filename extension derived from the file format. Set this variable to NULL to disable the adding of a file suffix.
------------------------------	---

### Note

The operators in this module need to open all output files simultaneously. The maximum number of open files depends on the operating system!

## Example

Assume an input GRIB1 dataset with three variables, e.g. code number 129, 130 and 139. To split this dataset into three pieces, one for each code number use:

```
cdo splitcode infile code
```

Result of 'dir code\*':

```
code129.grb code130.grb code139.grb
```

## 2.2.7. SPLITTIME - Split timesteps of a dataset

### Synopsis

```
<operator> infile obase
```

```
splitmon[,format] infile obase
```

### Description

This module splits `infile` into timesteps pieces. The output files will be named `<obase><xxx><suffix>` where `suffix` is the filename extension derived from the file format. `xxx` and the contents of the output files depends on the chosen operator.

### Operators

<b>splithour</b>	Split hours Splits a file into pieces, one for each different hour. <code>xxx</code> will have two digits with the hour.
<b>splitday</b>	Split days Splits a file into pieces, one for each different day. <code>xxx</code> will have two digits with the day.
<b>splitseas</b>	Split seasons Splits a file into pieces, one for each different season. <code>xxx</code> will have three characters with the season.
<b>splityear</b>	Split years Splits a file into pieces, one for each different year. <code>xxx</code> will have four digits with the year (YYYY).
<b>splityearmon</b>	Split in years and months Splits a file into pieces, one for each different year and month. <code>xxx</code> will have six digits with the year and month (YYYYMM).
<b>splitmon</b>	Split months Splits a file into pieces, one for each different month. <code>xxx</code> will have two digits with the month.

### Parameter

<i>format</i>	STRING	C-style format for <code>strftime()</code> (e.g. <code>%B</code> for the full month name)
---------------	--------	---

### Environment

<code>CDO_FILE_SUFFIX</code>	Set the default file suffix. This suffix will be added to the output file names instead of the filename extension derived from the file format. Set this variable to <code>NULL</code> to disable the adding of a file suffix.
------------------------------	--

### Note

The operators in this module need to open all output files simultaneously. The maximum number of open files depends on the operating system!



## Example

Assume the input GRIB1 dataset has timesteps from January to December. To split each month with all variables into one separate file use:

```
cdo splitmon infile mon
```

Result of 'dir mon\*':

```
mon01.grb  mon02.grb  mon03.grb  mon04.grb  mon05.grb  mon06.grb
mon07.grb  mon08.grb  mon09.grb  mon10.grb  mon11.grb  mon12.grb
```

## 2.2.8. SPLITSEL - Split selected timesteps

### Synopsis

```
splitsel,nsets[,noffset[,nskip]] infile obase
```

### Description

This operator splits *infile* into pieces, one for each adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same selected time range. The output files will be named `<obase><nnnnnn><suffix>` where *nnnnnn* is the sequence number and *suffix* is the filename extension derived from the file format.

### Parameter

<i>nsets</i>	INTEGER	Number of input timesteps for each output file
<i>noffset</i>	INTEGER	Number of input timesteps skipped before the first timestep range (optional)
<i>nskip</i>	INTEGER	Number of input timesteps skipped between timestep ranges (optional)

### Environment

CDO_FILE_SUFFIX	Set the default file suffix. This suffix will be added to the output file names instead of the filename extension derived from the file format. Set this variable to NULL to disable the adding of a file suffix.
-----------------	---

## 2.2.9. DISTGRID - Distribute horizontal grid

### Synopsis

```
distgrid,nx[,ny] infile obase
```

### Description

This operator distributes a dataset into smaller pieces. Each output file contains a different region of the horizontal source grid. A target grid region contains a structured longitude/latitude box of the source grid. Only rectilinear and curvilinear source grids are supported by this operator. The number of different regions can be specified with the parameter *nx* and *ny*. The output files will be named `<obase><xxx><suffix>` where suffix is the filename extension derived from the file format. xxx will have five digits with the number of the target region.

### Parameter

<i>nx</i>	INTEGER	Number of regions in x direction
<i>ny</i>	INTEGER	Number of regions in y direction [default: 1]

### Note

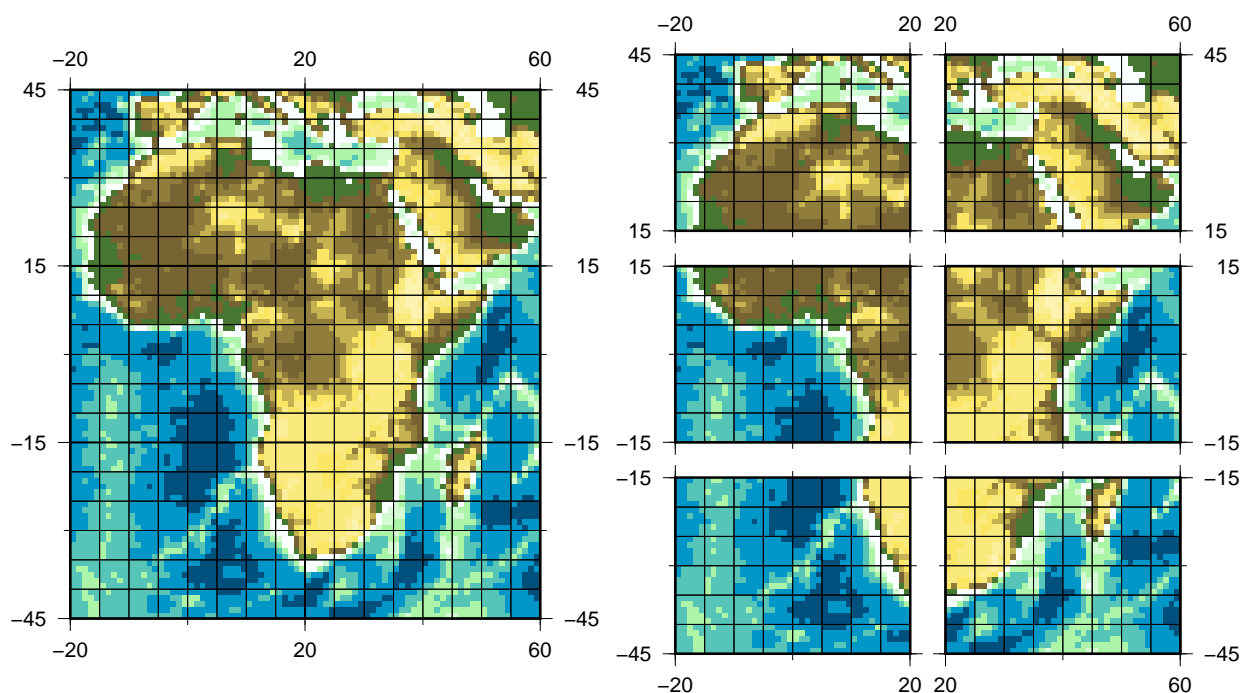
This operator needs to open all output files simultaneously. The maximum number of open files depends on the operating system!

### Example

Distribute a file into 6 smaller files, each output file receives one half of x and a third of y of the source grid:

```
cdo distgrid,2,3 infile.nc obase
```

Below is a schematic illustration of this example:



On the left side is the data of the input file and on the right side is the data of the six output files.

## 2.2.10. COLLGRID - Collect horizontal grid

### Synopsis

```
collgrid[,nx[,names]] infile outfile
```

### Description

This operator collects the data of the input files to one output file. All input files need to have the same variables and the same number of timesteps on a different horizontal grid region. A source region must be a structured longitude/latitude grid box. The parameter *nx* needs to be specified only for non regular lon/lat grids.

### Parameter

<i>nx</i>	INTEGER	Number of regions in x direction [default: number of input files]
<i>names</i>	STRING	Comma separated list of variable names [default: all variables]

### Note

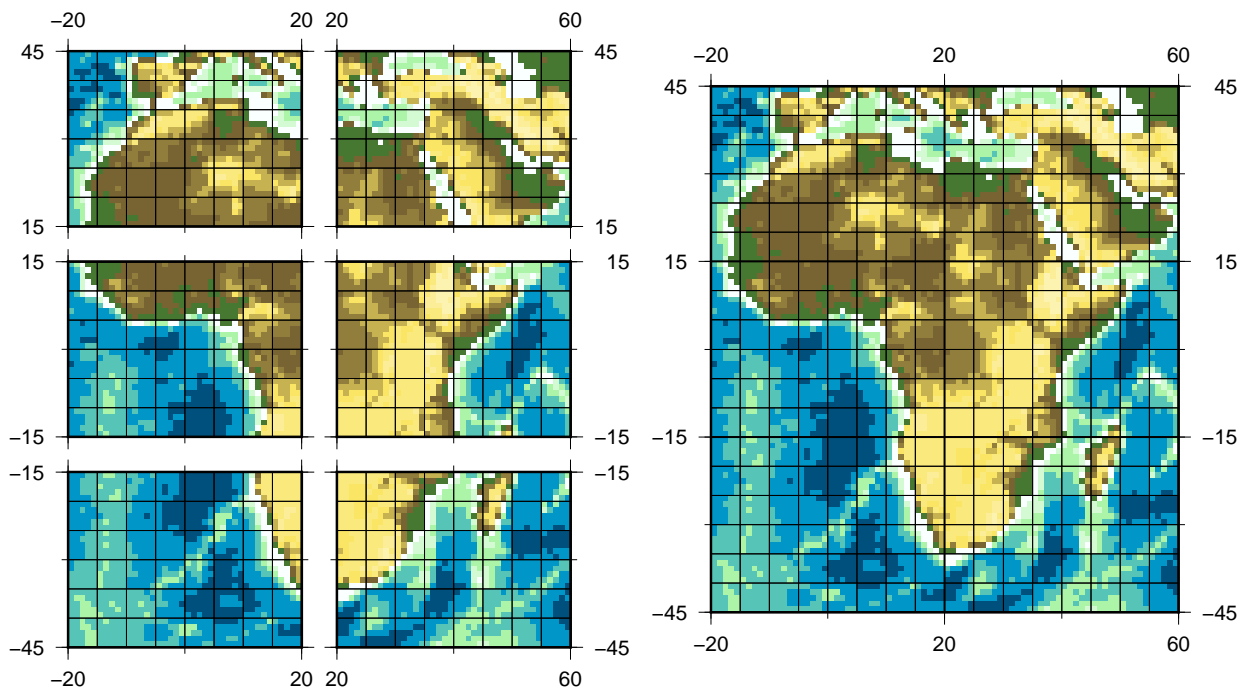
This operator needs to open all input files simultaneously. The maximum number of open files depends on the operating system!

### Example

Collect the horizontal grid of 6 input files. Each input file contains a lon/lat region of the target grid:

```
cdo collgrid infile[1-6] outfile
```

Below is a schematic illustration of this example:



On the left side is the data of the six input files and on the right side is the collected data of the output file.

## 2.3. Selection

This section contains modules to select time steps, fields or a part of a field from a dataset.

Here is a short overview of all operators in this section:

<b>select</b>	Select fields
<b>delete</b>	Delete fields
<b>selmulti</b>	Select multiple fields
<b>delmulti</b>	Delete multiple fields
<b>changemulti</b>	Change identification of multiple fields
<b>selparam</b>	Select parameters by identifier
<b>delparam</b>	Delete parameters by identifier
<b>selcode</b>	Select parameters by code number
<b>delcode</b>	Delete parameters by code number
<b>selname</b>	Select parameters by name
<b>delname</b>	Delete parameters by name
<b>selstdname</b>	Select parameters by standard name
<b>sellevel</b>	Select levels
<b>sellevidx</b>	Select levels by index
<b>selgrid</b>	Select grids
<b>selzaxis</b>	Select z-axes
<b>selzaxisname</b>	Select z-axes by name
<b>selltype</b>	Select GRIB level types
<b>seltabnum</b>	Select parameter table numbers
<b>sel timestep</b>	Select timesteps
<b>seltime</b>	Select times
<b>selhour</b>	Select hours
<b>selday</b>	Select days
<b>selmonth</b>	Select months
<b>selyear</b>	Select years
<b>selseason</b>	Select seasons
<b>seldate</b>	Select dates
<b>selsmon</b>	Select single month
<b>sellonlatbox</b>	Select a longitude/latitude box
<b>selindexbox</b>	Select an index box
<b>selgridcell</b>	Select grid cells
<b>delgridcell</b>	Delete grid cells
<b>samplegrid</b>	Resample grid

### 2.3.1. SELECT - Select fields

#### Synopsis

```
<operator> ,params infiles outfile
```

#### Description

This module selects some fields from `infiles` and writes them to `outfile`. `infiles` is an arbitrary number of input files. All input files need to have the same structure with the same variables on different timesteps. The fields selected depends on the chosen parameters. Parameter is a comma separated list of "key=value" pairs. Wildcards can be used for string parameter.

#### Operators

<b>select</b>	Select fields Selects all fields with parameters in a user given list.
<b>delete</b>	Delete fields Deletes all fields with parameters in a user given list.

#### Parameter

<i>name</i>	STRING	Comma separated list of variable names.
<i>param</i>	STRING	Comma separated list of parameter identifiers.
<i>code</i>	INTEGER	Comma separated list of code numbers.
<i>level</i>	FLOAT	Comma separated list of vertical levels.
<i>levidx</i>	INTEGER	Comma separated list of index of levels.
<i>zaxisname</i>	STRING	Comma separated list of zaxis names.
<i>zaxisnum</i>	INTEGER	Comma separated list of zaxis numbers.
<i>ltype</i>	INTEGER	Comma separated list of GRIB level types.
<i>gridname</i>	STRING	Comma separated list of grid names.
<i>gridnum</i>	INTEGER	Comma separated list of grid numbers.
<i>steptype</i>	STRING	Comma separated list of timestep types.
<i>date</i>	STRING	Comma separated list of dates (format YYYY-MM-DDThh:mm:ss).
<i>startdate</i>	STRING	Start date (format YYYY-MM-DDThh:mm:ss).
<i>enddate</i>	STRING	End date (format YYYY-MM-DDThh:mm:ss).
<i>minute</i>	INTEGER	Comma separated list of minutes.
<i>hour</i>	INTEGER	Comma separated list of hours.
<i>day</i>	INTEGER	Comma separated list of days.
<i>month</i>	INTEGER	Comma separated list of months.
<i>season</i>	STRING	Comma separated list of seasons (substring of DJFMAMJJASOND or ANN).
<i>year</i>	INTEGER	Comma separated list of years.
<i>timestep</i>	INTEGER	Comma separated list of timesteps. Negative values selects timesteps from the end (NetCDF only).
<i>timestep_of_year</i>	INTEGER	Comma separated list of timesteps of year.
<i>timestepmask</i>	STRING	Read timesteps from a mask file.

**Example**

Assume you have 3 inputfiles. Each inputfile contains the same variables for a different time period. To select the variable T,U and V on the levels 200, 500 and 850 from all 3 input files, use:

```
cdo select,name=T,U,V,level=200,500,850 infile1 infile2 infile3 outfile
```

## 2.3.2. SELMULTI - Select multiple fields via GRIB1 parameters

### Synopsis

`<operator>,selection-specification infile outfile`

### Description

This module selects multiple fields from infile and writes them to outfile. selection-specification is a filename or in-place string with the selection specification. Each selection-specification has the following compact notation format:

```
<type>(parameters; leveltype(s); levels)
```

type            sel for select or del for delete (optional)

parameters    GRIB1 parameter code number

leveltype     GRIB1 level type

levels        value of each level

Examples:

```
(1; 103; 0)
(33,34; 105; 10)
(11,17; 105; 2)
(71,73,74,75,61,62,65,117,67,122,121,11,131,66,84,111,112; 105; 0)
```

The following descriptive notation can also be used for selection specification from a file:

```
SELECT/DELETE, PARAMETER=parameters, LEVTYPE=leveltype(s), LEVEL=levels
```

Examples:

```
SELECT, PARAMETER=1, LEVTYPE=103, LEVEL=0
SELECT, PARAMETER=33/34, LEVTYPE=105, LEVEL=10
SELECT, PARAMETER=11/17, LEVTYPE=105, LEVEL=2
SELECT, PARAMETER=71/73/74/75/61/62/65/117/67/122, LEVTYPE=105, LEVEL=0
DELETE, PARAMETER=128, LEVTYPE=109, LEVEL=*
```

The following will convert Pressure from Pa into hPa; Temp from Kelvin to Celsius:

```
SELECT, PARAMETER=1, LEVTYPE= 103, LEVEL=0, SCALE=0.01
SELECT, PARAMETER=11, LEVTYPE=105, LEVEL=2, OFFSET=273.15
```

If SCALE and/or OFFSET are defined, then the data values are scaled as  $SCALE*(VALUE-OFFSET)$ .

### Operators

**selmulti**        Select multiple fields

**delmulti**        Delete multiple fields

**changemulti**    Change identification of multiple fields

### Example

Change ECMWF GRIB code of surface pressure to Hirlam notation:

```
cdo changemulti,'{(134;1;*|1;105;*)}' infile outfile
```

### 2.3.3. SELVAR - Select fields

#### Synopsis

```

<operator> ,params infile outfile
selcode, codes infile outfile
delcode, codes infile outfile
selname, names infile outfile
delname, names infile outfile
selstdname, stdnames infile outfile
sellevel, levels infile outfile
sellevidx, levidx infile outfile
selgrid, grids infile outfile
selzaxis, zaxes infile outfile
selzaxisname, zaxisnames infile outfile
selltype, ltypes infile outfile
seltabnum, tabnums infile outfile

```

#### Description

This module selects some fields from infile and writes them to outfile. The fields selected depends on the chosen operator and the parameters.

#### Operators

<b>selparam</b>	Select parameters by identifier Selects all fields with parameter identifiers in a user given list.
<b>delparam</b>	Delete parameters by identifier Deletes all fields with parameter identifiers in a user given list.
<b>selcode</b>	Select parameters by code number Selects all fields with code numbers in a user given list.
<b>delcode</b>	Delete parameters by code number Deletes all fields with code numbers in a user given list.
<b>selname</b>	Select parameters by name Selects all fields with parameter names in a user given list.
<b>delname</b>	Delete parameters by name Deletes all fields with parameter names in a user given list.
<b>selstdname</b>	Select parameters by standard name Selects all fields with standard names in a user given list.
<b>sellevel</b>	Select levels Selects all fields with levels in a user given list.
<b>sellevidx</b>	Select levels by index Selects all fields with index of levels in a user given list.
<b>selgrid</b>	Select grids Selects all fields with grids in a user given list.



<b>selzaxis</b>	Select z-axes Selects all fields with z-axes in a user given list.
<b>selzaxisname</b>	Select z-axes by name Selects all fields with z-axis names in a user given list.
<b>selltype</b>	Select GRIB level types Selects all fields with GRIB level type in a user given list.
<b>seltabnum</b>	Select parameter table numbers Selects all fields with parameter table numbers in a user given list.

## Parameter

<i>params</i>	INTEGER	Comma separated list of parameter identifiers
<i>codes</i>	INTEGER	Comma separated list of code numbers
<i>names</i>	STRING	Comma separated list of variable names
<i>stdnames</i>	STRING	Comma separated list of standard names
<i>levels</i>	FLOAT	Comma separated list of vertical levels
<i>levidx</i>	INTEGER	Comma separated list of index of levels
<i>ltypes</i>	INTEGER	Comma separated list of GRIB level types
<i>grids</i>	STRING	Comma separated list of grid names or numbers
<i>zaxes</i>	STRING	Comma separated list of z-axis types or numbers
<i>zaxisnames</i>	STRING	Comma separated list of z-axis names
<i>tabnums</i>	INTEGER	Comma separated list of parameter table numbers

## Example

Assume an input dataset has three variables with the code numbers 129, 130 and 139. To select the variables with the code number 129 and 139 use:

```
cdo selcode,129,139 infile outfile
```

You can also select the code number 129 and 139 by deleting the code number 130 with:

```
cdo delcode,130 infile outfile
```

### 2.3.4. SELTIME - Select timesteps

#### Synopsis

```

se timestep,timesteps infile outfile
seltime,times infile outfile
selhour,hours infile outfile
selday,days infile outfile
selmonth,months infile outfile
selyear,years infile outfile
selseason,seasons infile outfile
seldate,date1[,date2] infile outfile
selsmon,month[,nts1[,nts2]] infile outfile

```

#### Description

This module selects user specified timesteps from *infile* and writes them to *outfile*. The timesteps selected depends on the chosen operator and the parameters.

#### Operators

<b>se timestep</b>	Select timesteps Selects all timesteps with a timestep in a user given list.
<b>seltime</b>	Select times Selects all timesteps with a time in a user given list.
<b>selhour</b>	Select hours Selects all timesteps with a hour in a user given list.
<b>selday</b>	Select days Selects all timesteps with a day in a user given list.
<b>selmonth</b>	Select months Selects all timesteps with a month in a user given list.
<b>selyear</b>	Select years Selects all timesteps with a year in a user given list.
<b>selseason</b>	Select seasons Selects all timesteps with a month of a season in a user given list.
<b>seldate</b>	Select dates Selects all timesteps with a date in a user given range.
<b>selsmon</b>	Select single month Selects a month and optional an arbitrary number of timesteps before and after this month.

#### Parameter

<i>timesteps</i>	INTEGER	Comma separated list of timesteps. Negative values selects timesteps from the end (NetCDF only).
<i>times</i>	STRING	Comma separated list of times (format hh:mm:ss).
<i>hours</i>	INTEGER	Comma separated list of hours.

---

<i>days</i>	INTEGER	Comma separated list of days.
<i>months</i>	INTEGER	Comma separated list of months.
<i>years</i>	INTEGER	Comma separated list of years.
<i>seasons</i>	STRING	Comma separated list of seasons (substring of DJFMAMJJASOND or ANN).
<i>date1</i>	STRING	Start date (format YYYY-MM-DDThh:mm:ss).
<i>date2</i>	STRING	End date (format YYYY-MM-DDThh:mm:ss) [default: date1].
<i>nts1</i>	INTEGER	Number of timesteps before the selected month [default: 0].
<i>nts2</i>	INTEGER	Number of timesteps after the selected month [default: nts1].

### 2.3.5. SELBOX - Select a box of a field

#### Synopsis

```
sellonlatbox,lon1,lon2,lat1,lat2 infile outfile
selindexbox,idx1,idx2,idy1,idy2 infile outfile
```

#### Description

Selects a box of the rectangularly understood field.

#### Operators

<b>sellonlatbox</b>	Select a longitude/latitude box Selects a regular longitude/latitude box. The user has to give the longitudes and latitudes of the edges of the box. Considered are only those grid cells with the grid center inside the lon/lat box. For rotated lon/lat grids the parameter needs to be rotated coordinates.
<b>selindexbox</b>	Select an index box Selects an index box. The user has to give the indexes of the edges of the box. The index of the left edge may be greater then that of the right edge.

#### Parameter

<i>lon1</i>	FLOAT	Western longitude
<i>lon2</i>	FLOAT	Eastern longitude
<i>lat1</i>	FLOAT	Southern or northern latitude
<i>lat2</i>	FLOAT	Northern or southern latitude
<i>idx1</i>	INTEGER	Index of first longitude (1 - nlon)
<i>idx2</i>	INTEGER	Index of last longitude (1 - nlon)
<i>idy1</i>	INTEGER	Index of first latitude (1 - nlat)
<i>idy2</i>	INTEGER	Index of last latitude (1 - nlat)

#### Example

To select the region with the longitudes from 30W to 60E and latitudes from 30N to 80N from all input fields use:

```
cdo sellonlatbox,-30,60,30,80 infile outfile
```

If the input dataset has fields on a Gaussian N16 grid, the same box can be selected with [selindexbox](#) by:

```
cdo selindexbox,60,11,3,11 infile outfile
```

### 2.3.6. SELGRIDCELL - Select grid cells

#### Synopsis

```
<operator> ,indexes infile outfile
```

#### Description

Selects grid cells of all fields from `infile`. The user has to give the indexes of each grid cell. The resulting grid in `outfile` is unstructured.

#### Operators

`selgridcell`     Select grid cells

`delgridcell`     Delete grid cells

#### Parameter

*indexes*     INTEGER     Comma separated list of indexes

### 2.3.7. SAMPLEGRID - Resample grid

#### Synopsis

```
samplegrid ,factor infile outfile
```

#### Description

This is a special operator for resampling the horizontal grid. No interpolation takes place. Resample factor=2 means every second grid point is removed. Only rectilinear and curvilinear source grids are supported by this operator.

#### Parameter

*factor*     INTEGER     Resample factor, typically 2, which will half the resolution

## 2.4. Conditional selection

This section contains modules to conditional select field elements. The fields in the first input file are handled as a mask. A value not equal to zero is treated as "true", zero is treated as "false".

Here is a short overview of all operators in this section:

<b>ifthen</b>	If then
<b>ifnotthen</b>	If not then
<b>ifthenelse</b>	If then else
<b>ifthenc</b>	If then constant
<b>ifnotthenc</b>	If not then constant
<b>reducegrid</b>	Reduce input file variables to locations, where mask is non-zero.

### 2.4.1. COND - Conditional select one field

#### Synopsis

```
<operator> infile1 infile2 outfile
```

#### Description

This module selects field elements from `infile2` with respect to `infile1` and writes them to `outfile`. The fields in `infile1` are handled as a mask. A value not equal to zero is treated as "true", zero is treated as "false". The number of fields in `infile1` has either to be the same as in `infile2` or the same as in one timestep of `infile2` or only one. The fields in `outfile` inherit the meta data from `infile2`.

#### Operators

<b>ifthen</b>	If then
	$o(t, x) = \begin{cases} i_2(t, x) & \text{if } i_1([t, ]x) \neq 0 \quad \wedge \quad i_1([t, ]x) \neq \text{miss} \\ \text{miss} & \text{if } i_1([t, ]x) = 0 \quad \vee \quad i_1([t, ]x) = \text{miss} \end{cases}$
<b>ifnotthen</b>	If not then
	$o(t, x) = \begin{cases} i_2(t, x) & \text{if } i_1([t, ]x) = 0 \quad \wedge \quad i_1([t, ]x) \neq \text{miss} \\ \text{miss} & \text{if } i_1([t, ]x) \neq 0 \quad \vee \quad i_1([t, ]x) = \text{miss} \end{cases}$

#### Example

To select all field elements of `infile2` if the corresponding field element of `infile1` is greater than 0 use:

```
cdo ifthen infile1 infile2 outfile
```

### 2.4.2. COND2 - Conditional select two fields

#### Synopsis

```
ifthenelse infile1 infile2 infile3 outfile
```

#### Description

This operator selects field elements from `infile2` or `infile3` with respect to `infile1` and writes them to `outfile`. The fields in `infile1` are handled as a mask. A value not equal to zero is treated as "true", zero is treated as "false". The number of fields in `infile1` has either to be the same as in `infile2` or the same as in one timestep of `infile2` or only one. `infile2` and `infile3` need to have the same number of fields. The fields in `outfile` inherit the meta data from `infile2`.

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

#### Example

To select all field elements of `infile2` if the corresponding field element of `infile1` is greater than 0 and from `infile3` otherwise use:

```
cdo ifthenelse infile1 infile2 infile3 outfile
```

### 2.4.3. CONDC - Conditional select a constant

#### Synopsis

```
<operator>,c infile outfile
```

#### Description

This module creates fields with a constant value or missing value. The fields in `infile` are handled as a mask. A value not equal to zero is treated as "true", zero is treated as "false".

#### Operators

<b>ifthenc</b>	If then constant
	$o(t, x) = \begin{cases} c & \text{if } i(t, x) \neq 0 \wedge i(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = 0 \vee i(t, x) = \text{miss} \end{cases}$
<b>ifnotthenc</b>	If not then constant
	$o(t, x) = \begin{cases} c & \text{if } i(t, x) = 0 \wedge i(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) \neq 0 \vee i(t, x) = \text{miss} \end{cases}$

#### Parameter

<code>c</code>	Float	Constant
----------------	-------	----------

#### Example

To create fields with the constant value 7 if the corresponding field element of `infile` is greater than 0 use:

```
cdo ifthenc,7 infile outfile
```



## 2.4.4. MAPREDUCE - Reduce fields to user-defined mask

### Synopsis

```
reducegrid,mask[,limitCoordsOutput] infile outfile
```

### Description

This module holds an operator for data reduction based on a user defined mask. The output grid is unstructured and includes coordinate bounds. Bounds can be avoided by using the additional 'nobounds' keyword. With 'nocoords' given, coordinates are completely suppressed.

### Parameter

<i>mask</i>	STRING	file which holds the mask field
<i>limitCoordsOutput</i>	STRING	optional parameter to limit coordinates output: 'nobounds' disables coordinate bounds, 'nocoords' avoids all coordinate information

### Example

To limit data fields to land values, a mask has to be created first with

```
cdo -gtc,0 -topo,ni96 lsm_gme96.grb
```

Here a GME grid is used. Say temp\_gme96.grb contains a global temperature field. The following command limits the global grid to landpoints.

```
cdo -f nc reduce,lsm_gme96.grb temp_gme96.grb tempOnLand_gme96.nc
```

Note that output file type is NetCDF, because unstructured grids cannot be stored in GRIB format.

## 2.5. Comparison

This section contains modules to compare datasets. The resulting field is a mask containing 1 if the comparison is true and 0 if not.

Here is a short overview of all operators in this section:

<code>eq</code>	Equal
<code>ne</code>	Not equal
<code>le</code>	Less equal
<code>lt</code>	Less than
<code>ge</code>	Greater equal
<code>gt</code>	Greater than
<code>eqc</code>	Equal constant
<code>nec</code>	Not equal constant
<code>lec</code>	Less equal constant
<code>ltc</code>	Less than constant
<code>gec</code>	Greater equal constant
<code>gtc</code>	Greater than constant

## 2.5.1. COMP - Comparison of two fields

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module compares two datasets field by field. The resulting field is a mask containing 1 if the comparison is true and 0 if not. The number of fields in `infile1` should be the same as in `infile2`. One of the input files can contain only one timestep or one field. The fields in `outfile` inherit the meta data from `infile1` or `infile2`. The type of comparison depends on the chosen operator.

### Operators

<b>eq</b>	Equal	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) = i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) \neq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$
<b>ne</b>	Not equal	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) \neq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) = i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$
<b>le</b>	Less equal	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) \leq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) > i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$
<b>lt</b>	Less than	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) < i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) \geq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$
<b>ge</b>	Greater equal	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) \geq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) < i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$
<b>gt</b>	Greater than	$o(t, x) = \begin{cases} 1 & \text{if } i_1(t, x) > i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ 0 & \text{if } i_1(t, x) \leq i_2(t, x) \quad \wedge \quad i_1(t, x), i_2(t, x) \neq \text{miss} \\ \text{miss} & \text{if } i_1(t, x) = \text{miss} \quad \vee \quad i_2(t, x) = \text{miss} \end{cases}$

### Example

To create a mask containing 1 if the elements of two fields are the same and 0 if the elements are different use:

```
cdo eq infile1 infile2 outfile
```

## 2.5.2. COMPC - Comparison of a field with a constant

### Synopsis

```
<operator>,c infile outfile
```

### Description

This module compares all fields of a dataset with a constant. The resulting field is a mask containing 1 if the comparison is true and 0 if not. The type of comparison depends on the chosen operator.

### Operators

<b>eqc</b>	Equal constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) = c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) \neq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	
<b>nec</b>	Not equal constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) \neq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) = c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	
<b>lec</b>	Less equal constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) \leq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) > c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	
<b>ltc</b>	Less than constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) < c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) \geq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	
<b>gec</b>	Greater equal constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) \geq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) < c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	
<b>gtc</b>	Greater than constant
$o(t, x) = \begin{cases} 1 & \text{if } i(t, x) > c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ 0 & \text{if } i(t, x) \leq c \quad \wedge \quad i(t, x), c \neq \text{miss} \\ \text{miss} & \text{if } i(t, x) = \text{miss} \quad \vee \quad c = \text{miss} \end{cases}$	

### Parameter

*c*      FLOAT      Constant

### Example

To create a mask containing 1 if the field element is greater than 273.15 and 0 if not use:

```
cdo gtc,273.15 infile outfile
```

## 2.6. Modification

This section contains modules to modify the metadata, fields or part of a field in a dataset.

Here is a short overview of all operators in this section:

<b>setattribute</b>	Set attributes
<b>setpartabp</b>	Set parameter table
<b>setpartabn</b>	Set parameter table
<b>setcodetab</b>	Set parameter code table
<b>setcode</b>	Set code number
<b>setparam</b>	Set parameter identifier
<b>setname</b>	Set variable name
<b>setunit</b>	Set variable unit
<b>setlevel</b>	Set level
<b>setltype</b>	Set GRIB level type
<b>setdate</b>	Set date
<b>settime</b>	Set time of the day
<b>setday</b>	Set day
<b>setmon</b>	Set month
<b>setyear</b>	Set year
<b>settunits</b>	Set time units
<b>settaxis</b>	Set time axis
<b>settbounds</b>	Set time bounds
<b>setreftime</b>	Set reference time
<b>setcalendar</b>	Set calendar
<b>shifttime</b>	Shift timesteps
<b>chcode</b>	Change code number
<b>chparam</b>	Change parameter identifier
<b>chname</b>	Change variable name
<b>chunit</b>	Change variable unit
<b>chlevel</b>	Change level
<b>chlevelc</b>	Change level of one code
<b>chlevelv</b>	Change level of one variable
<b>setgrid</b>	Set grid
<b>setgridtype</b>	Set grid type
<b>setgridarea</b>	Set grid cell area
<b>setzaxis</b>	Set z-axis
<b>genlevelbounds</b>	Generate level bounds
<b>invertlat</b>	Invert latitudes
<b>invertlev</b>	Invert levels
<b>shiftx</b>	Shift x
<b>shifty</b>	Shift y
<b>maskregion</b>	Mask regions
<b>masklonlatbox</b>	Mask a longitude/latitude box
<b>maskindexbox</b>	Mask an index box

---

<b>setclonlatbox</b>	Set a longitude/latitude box to constant
<b>setcindexbox</b>	Set an index box to constant
<b>enlarge</b>	Enlarge fields
<b>setmissval</b>	Set a new missing value
<b>setctomiss</b>	Set constant to missing value
<b>setmisstoc</b>	Set missing value to constant
<b>setrtomiss</b>	Set range to missing value
<b>setvrangle</b>	Set valid range
<b>setmisstonn</b>	Set missing value to nearest neighbor
<b>setmisstodis</b>	Set missing value to distance-weighted average

## 2.6.1. SETATTRIBUTE - Set attributes

### Synopsis

```
setattribute,attributes infile outfile
```

### Description

This operator sets attributes of a dataset. Each attribute has the following structure:

```
[var_nm@]att_nm=att_val
```

**var\_nm** Variable name (optional). Example: pressure

**att\_nm** Attribute name. Example: units

**att\_val** Comma separated list of attribute values. Example: pascal

The value of **var\_nm** is the name of the variable containing the attribute (named **att\_nm**) that you want to set. Use wildcards to set the attribute **att\_nm** to more than one variable. A value of **var\_nm** of '\*' will set the attribute **att\_nm** to all data variables. If **var\_nm** is missing then **att\_nm** refers to a global attribute.

The value of **att\_nm** is the name of the attribute you want to set.

The value of **att\_val** is the contents of the attribute **att\_nm**. **att\_val** may be a single value or one-dimensional array of elements. The type of the attribute value will be detected automatically from the contents of the value.

A special meaning has the attribute name **FILE**. If this is the 1st attribute then all attributes are read from a file specified in the value of **att\_val**.

### Parameter

**attributes**      STRING      Comma separated list of attributes.

### Example

To set the units of the variable pressure to pascal use:

```
cdo setattribute,pressure@units=pascal infile outfile
```

To set the global text attribute "my\_att" to "my contents", use:

```
cdo setattribute,my_att="my contents" infile outfile
```

Result of 'ncdump -h outfile':

```
netcdf outfile {
  dimensions: ...

  variables: ...

  // global attributes:
      :my_att = "my contents" ;
}
```

## 2.6.2. SETPARTAB - Set parameter table

### Synopsis

```
<operator>[,table[,convert]] infile outfile
```

### Description

This module transforms data and metadata of infile via a parameter table and writes the result to outfile. A parameter table is an ASCII formatted file with a set of parameter entries for each variable. Each new set have to start with "&parameter" and to end with "/".

The following parameter table entries are supported:

Entry	Type	Description
name	WORD	Name of the variable
out_name	WORD	New name of the variable
param	WORD	Parameter identifier (GRIB1: code[.tabnum]; GRIB2: num[.cat[.dis]])
out_param	WORD	New parameter identifier
type	WORD	Data type (real or double)
standard_name	WORD	As defined in the CF standard name table
long_name	STRING	Describing the variable
units	STRING	Specifying the units for the variable
comment	STRING	Information concerning the variable
cell_methods	STRING	Information concerning calculation of means or climatologies
cell_measures	STRING	Indicates the names of the variables containing cell areas and volumes
missing_value	FLOAT	Specifying how missing data will be identified
valid_min	FLOAT	Minimum valid value
valid_max	FLOAT	Maximum valid value
ok_min_mean_abs	FLOAT	Minimum absolute mean
ok_max_mean_abs	FLOAT	Maximum absolute mean
factor	FLOAT	Scale factor
delete	INTEGER	Set to 1 to delete variable
convert	INTEGER	Set to 1 to convert the unit if necessary

Unsupported parameter table entries are stored as variable attributes. The search key for the variable depends on the operator. Use [setpartabn](#) to search variables by the name. This is typically used for NetCDF datasets. The operator [setpartabp](#) searches variables by the parameter ID.

### Operators

**setpartabp**      Set parameter table  
                     Search variables by the parameter identifier.

**setpartabn**      Set parameter table  
                     Search variables by name.

### Parameter

*table*            STRING      Parameter table file or name

*convert*        STRING      Converts the units if necessary



## Example

Here is an example of a parameter table for one variable:

```
prompt> cat mypartab
&parameter
  name          = t
  out_name       = ta
  standard_name  = air_temperature
  units          = "K"
  missing_value  = 1e+20
  valid_min      = 157.1
  valid_max      = 336.3
/
```

To apply this parameter table to a dataset use:

```
cdo setpartabn,mypartab,convert infile outfile
```

This command renames the variable **t** to **ta**. The standard name of this variable is set to **air\_temperature** and the unit is set to **[K]** (converts the unit if necessary). The missing value will be set to **1e+20**. In addition it will be checked whether the values of the variable are in the range of **157.1** to **336.3**.

### 2.6.3. SET - Set field info

#### Synopsis

```

setcodetab,table infile outfile
setcode,code infile outfile
setparam,param infile outfile
setname,name infile outfile
setunit,unit infile outfile
setlevel,level infile outfile
setltype,ltype infile outfile

```

#### Description

This module sets some field information. Depending on the chosen operator the parameter table, code number, parameter identifier, variable name or level is set.

#### Operators

<b>setcodetab</b>	Set parameter code table Sets the parameter code table for all variables.
<b>setcode</b>	Set code number Sets the code number for all variables to the same given value.
<b>setparam</b>	Set parameter identifier Sets the parameter identifier of the first variable.
<b>setname</b>	Set variable name Sets the name of the first variable.
<b>setunit</b>	Set variable unit Sets the unit of the first variable.
<b>setlevel</b>	Set level Sets the first level of all variables.
<b>setltype</b>	Set GRIB level type Sets the GRIB level type of all variables.

#### Parameter

<i>table</i>	STRING	Parameter table file or name
<i>code</i>	INTEGER	Code number
<i>param</i>	STRING	Parameter identifier (GRIB1: code[.tabnum]; GRIB2: num[.cat[.dis]])
<i>name</i>	STRING	Variable name
<i>level</i>	FLOAT	New level
<i>ltype</i>	INTEGER	GRIB level type

## 2.6.4. SETTIME - Set time

### Synopsis

```

setdate,date infile outfile
settime,time infile outfile
setday,day infile outfile
setmon,month infile outfile
setyear,year infile outfile
settunits,units infile outfile
settaxis,date,time[,inc] infile outfile
settbounds,frequency infile outfile
setreftime,date,time[,units] infile outfile
setcalendar,calendar infile outfile
shifttime,sval infile outfile

```

### Description

This module sets the time axis or part of the time axis. Which part of the time axis is overwritten/created depends on the chosen operator.

### Operators

<b>setdate</b>	Set date Sets the date in every timestep to the same given value.
<b>settime</b>	Set time of the day Sets the time in every timestep to the same given value.
<b>setday</b>	Set day Sets the day in every timestep to the same given value.
<b>setmon</b>	Set month Sets the month in every timestep to the same given value.
<b>setyear</b>	Set year Sets the year in every timestep to the same given value.
<b>settunits</b>	Set time units Sets the base units of a relative time axis.
<b>settaxis</b>	Set time axis Sets the time axis.
<b>settbounds</b>	Set time bounds Sets the time bounds.
<b>setreftime</b>	Set reference time Sets the reference time of a relative time axis.
<b>setcalendar</b>	Set calendar Sets the calendar of a relative time axis.
<b>shifttime</b>	Shift timesteps Shifts all timesteps by the parameter sval.

## Parameter

<i>day</i>	INTEGER	Value of the new day
<i>month</i>	INTEGER	Value of the new month
<i>year</i>	INTEGER	Value of the new year
<i>units</i>	STRING	Base units of the time axis (seconds, minutes, hours, days, months, years)
<i>date</i>	STRING	Date (format: YYYY-MM-DD)
<i>time</i>	STRING	Time (format: hh:mm:ss)
<i>inc</i>	STRING	Optional increment (seconds, minutes, hours, days, months, years) [default: 1hour]
<i>frequency</i>	STRING	Frequency of the time series (hour, day, month, year)
<i>calendar</i>	STRING	Calendar (standard, proleptic_gregorian, 360_day, 365_day, 366_day)
<i>sval</i>	STRING	Shift value (e.g. -3hour)

## Example

To set the time axis to 1987-01-16 12:00:00 with an increment of one month for each timestep use:

```
cdo settaxis,1987-01-16,12:00:00,1mon infile outfile
```

Result of 'cdo showdate outfile' for a dataset with 12 timesteps:

```
1987-01-16 1987-02-16 1987-03-16 1987-04-16 1987-05-16 1987-06-16 \
1987-07-16 1987-08-16 1987-09-16 1987-10-16 1987-11-16 1987-12-16
```

To shift this time axis by -15 days use:

```
cdo shifttime,-15days infile outfile
```

Result of 'cdo showdate outfile':

```
1987-01-01 1987-02-01 1987-03-01 1987-04-01 1987-05-01 1987-06-01 \
1987-07-01 1987-08-01 1987-09-01 1987-10-01 1987-11-01 1987-12-01
```

## 2.6.5. CHANGE - Change field header

### Synopsis

```

chcode,oldcode,newcode[,...]  infile outfile
chparam,oldparam,newparam,...  infile outfile
chname,oldname,newname,...  infile outfile
chunit,oldunit,newunit,...  infile outfile
chlevel,oldlev,newlev,...  infile outfile
chlevelc,code,oldlev,newlev  infile outfile
chlevelv,name,oldlev,newlev  infile outfile

```

### Description

This module reads fields from *infile*, changes some header values and writes the results to *outfile*. The kind of changes depends on the chosen operator.

### Operators

<b>chcode</b>	Change code number Changes some user given code numbers to new user given values.
<b>chparam</b>	Change parameter identifier Changes some user given parameter identifiers to new user given values.
<b>chname</b>	Change variable name Changes some user given variable names to new user given names.
<b>chunit</b>	Change variable unit Changes some user given variable units to new user given units.
<b>chlevel</b>	Change level Changes some user given levels to new user given values.
<b>chlevelc</b>	Change level of one code Changes one level of a user given code number.
<b>chlevelv</b>	Change level of one variable Changes one level of a user given variable name.

### Parameter

<i>code</i>	INTEGER	Code number
<i>oldcode</i> , <i>newcode</i> ,...	INTEGER	Pairs of old and new code numbers
<i>oldparam</i> , <i>newparam</i> ,...	STRING	Pairs of old and new parameter identifiers
<i>name</i>	STRING	Variable name
<i>oldname</i> , <i>newname</i> ,...	STRING	Pairs of old and new variable names
<i>oldlev</i>	FLOAT	Old level
<i>newlev</i>	FLOAT	New level
<i>oldlev</i> , <i>newlev</i> ,...	FLOAT	Pairs of old and new levels

### Example

To change the code number 98 to 179 and 99 to 211 use:

```
cdo chcode,98,179,99,211 infile outfile
```

## 2.6.6. SETGRID - Set grid information

### Synopsis

```
setgrid,grid infile outfile
setgridtype,gridtype infile outfile
setgridarea,gridarea infile outfile
```

### Description

This module modifies the metadata of the horizontal grid. Depending on the chosen operator a new grid description is set, the coordinates are converted or the grid cell area is added.

### Operators

<b>setgrid</b>	Set grid Sets a new grid description. The input fields need to have the same grid size as the size of the target grid description.												
<b>setgridtype</b>	Set grid type Sets the grid type of all input fields. The following grid types are available: <table border="0"> <tr> <td>curvilinear</td><td>Converts a regular grid to a curvilinear grid</td></tr> <tr> <td>unstructured</td><td>Converts a regular or curvilinear grid to an unstructured grid</td></tr> <tr> <td>dereference</td><td>Dereference a reference to a grid</td></tr> <tr> <td>regular</td><td>Linear interpolation of a reduced Gaussian grid to a regular Gaussian grid</td></tr> <tr> <td>regularnn</td><td>Nearest neighbor interpolation of a reduced Gaussian grid to a regular Gaussian grid</td></tr> <tr> <td>lonlat</td><td>Converts a regular lonlat grid stored as a curvilinear grid back to a lonlat grid</td></tr> </table>	curvilinear	Converts a regular grid to a curvilinear grid	unstructured	Converts a regular or curvilinear grid to an unstructured grid	dereference	Dereference a reference to a grid	regular	Linear interpolation of a reduced Gaussian grid to a regular Gaussian grid	regularnn	Nearest neighbor interpolation of a reduced Gaussian grid to a regular Gaussian grid	lonlat	Converts a regular lonlat grid stored as a curvilinear grid back to a lonlat grid
curvilinear	Converts a regular grid to a curvilinear grid												
unstructured	Converts a regular or curvilinear grid to an unstructured grid												
dereference	Dereference a reference to a grid												
regular	Linear interpolation of a reduced Gaussian grid to a regular Gaussian grid												
regularnn	Nearest neighbor interpolation of a reduced Gaussian grid to a regular Gaussian grid												
lonlat	Converts a regular lonlat grid stored as a curvilinear grid back to a lonlat grid												
<b>setgridarea</b>	Set grid cell area Sets the grid cell area. The parameter <i>gridarea</i> is the path to a data file, the first field is used as grid cell area. The input fields need to have the same grid size as the grid cell area. The grid cell area is used to compute the weights of each grid cell if needed by an operator, e.g. for <a href="#">fldmean</a> .												

### Parameter

<i>grid</i>	STRING	Grid description file or name
<i>gridtype</i>	STRING	Grid type (curvilinear, unstructured, regular, lonlat or dereference)
<i>gridarea</i>	STRING	Data file, the first field is used as grid cell area

### Example

Assuming a dataset has fields on a grid with 2048 elements without or with wrong grid description. To set the grid description of all input fields to a Gaussian N32 grid (8192 gridpoints) use:

```
cdo setgrid,n32 infile outfile
```

## 2.6.7. SETZAXIS - Set z-axis information

### Synopsis

```
setzaxis,zaxis infile outfile
```

```
genlevelbounds[,zbot[,ztop]] infile outfile
```

### Description

This module modifies the metadata of the vertical grid.

### Operators

<b>setzaxis</b>	Set z-axis This operator sets the z-axis description of all variables with the same number of level as the new z-axis.
<b>genlevelbounds</b>	Generate level bounds Generates the layer bounds of the z-axis.

### Parameter

<i>zaxis</i>	STRING	Z-axis description file or name of the target z-axis
<i>zbot</i>	FLOAT z-axis.	Specifying the bottom of the vertical column. Must have the same units as
<i>ztop</i>	FLOAT	Specifying the top of the vertical column. Must have the same units as z-axis.

### 2.6.8. INVERT - Invert latitudes

#### Synopsis

```
invertlat infile outfile
```

#### Description

This operator inverts the latitudes of all fields on a rectilinear grid.

#### Example

To invert the latitudes of a 2D field from N->S to S->N use:

```
cdo invertlat infile outfile
```

### 2.6.9. INVERTLEV - Invert levels

#### Synopsis

```
invertlev infile outfile
```

#### Description

This operator inverts the levels of all 3D variables.



## 2.6.10. SHIFTXY - Shift field

### Synopsis

```
<operator>,<nshift>,<cyclic>,<coord> infile outfile
```

### Description

This module contains operators to shift all fields in x or y direction. All fields need to have the same horizontal rectilinear or curvilinear grid.

### Operators

<b>shiftx</b>	Shift x Shifts all fields in x direction.
<b>shifty</b>	Shift y Shifts all fields in y direction.

### Parameter

<i>nshift</i>	INTEGER	Number of grid cells to shift (default: 1)
<i>cyclic</i>	STRING	If set, cells are filled up cyclic (default: missing value)
<i>coord</i>	STRING	If set, coordinates are also shifted

### Example

To shift all input fields in the x direction by +1 cells and fill the new cells with missing value, use:

```
cdo shiftx infile outfile
```

To shift all input fields in the x direction by +1 cells and fill the new cells cyclic, use:

```
cdo shiftx,1,cyclic infile outfile
```

## 2.6.11. MASKREGION - Mask regions

### Synopsis

```
maskregion,regions infile outfile
```

### Description

Masks different regions of fields with a regular lon/lat grid. The elements inside a region are untouched, the elements outside are set to missing value. Considered are only those grid cells with the grid center inside the regions. All input fields must have the same horizontal grid. The user has to give ASCII formatted files with different regions. A region is defined by a polygon. Each line of a polygon description file contains the longitude and latitude of one point. Each polygon description file can contain one or more polygons separated by a line with the character &.

### Parameter

*regions*      STRING      Comma separated list of ASCII formatted files with different regions

### Example

To mask the region with the longitudes from 120E to 90W and latitudes from 20N to 20S on all input fields use:

```
cdo maskregion,myregion infile outfile
```

For this example the polygon description file myregion should contain the following four coordinates:

```
120  20
120 -20
270 -20
270  20
```

## 2.6.12. MASKBOX - Mask a box

### Synopsis

```
masklonlatbox,lon1,lon2,lat1,lat2 infile outfile
```

```
maskindexbox,idx1,idx2,idy1,idy2 infile outfile
```

### Description

Masked a box of the rectangularly understood field. The elements inside the box are untouched, the elements outside are set to missing value. All input fields need to have the same horizontal grid. Use [sellonlatbox](#) or [selindexbox](#) if only the data inside the box are needed.

### Operators

<b>masklonlatbox</b>	Mask a longitude/latitude box Masked a regular longitude/latitude box. The user has to give the longitudes and latitudes of the edges of the box. Considered are only those grid cells with the grid center inside the lon/lat box.
<b>maskindexbox</b>	Mask an index box Masked an index box. The user has to give the indexes of the edges of the box. The index of the left edge can be greater then the one of the right edge.

### Parameter

<i>lon1</i>	FLOAT	Western longitude
<i>lon2</i>	FLOAT	Eastern longitude
<i>lat1</i>	FLOAT	Southern or northern latitude
<i>lat2</i>	FLOAT	Northern or southern latitude
<i>idx1</i>	INTEGER	Index of first longitude
<i>idx2</i>	INTEGER	Index of last longitude
<i>idy1</i>	INTEGER	Index of first latitude
<i>idy2</i>	INTEGER	Index of last latitude

### Example

To mask the region with the longitudes from 120E to 90W and latitudes from 20N to 20S on all input fields use:

```
cdo masklonlatbox,120,-90,20,-20 infile outfile
```

If the input dataset has fields on a Gaussian N16 grid, the same box can be masked with [maskindexbox](#) by:

```
cdo maskindexbox,23,48,13,20 infile outfile
```

## 2.6.13. SETBOX - Set a box to constant

### Synopsis

```
setclonlatbox,c,lon1,lon2,lat1,lat2 infile outfile
```

```
setcindexbox,c,idx1,idx2,idy1,idy2 infile outfile
```

### Description

Sets a box of the rectangularly understood field to a constant value. The elements outside the box are untouched, the elements inside are set to the given constant. All input fields need to have the same horizontal grid.

### Operators

<b>setclonlatbox</b>	Set a longitude/latitude box to constant Sets the values of a longitude/latitude box to a constant value. The user has to give the longitudes and latitudes of the edges of the box.
<b>setcindexbox</b>	Set an index box to constant Sets the values of an index box to a constant value. The user has to give the indexes of the edges of the box. The index of the left edge can be greater than the one of the right edge.

### Parameter

<i>c</i>	FLOAT	Constant
<i>lon1</i>	FLOAT	Western longitude
<i>lon2</i>	FLOAT	Eastern longitude
<i>lat1</i>	FLOAT	Southern or northern latitude
<i>lat2</i>	FLOAT	Northern or southern latitude
<i>idx1</i>	INTEGER	Index of first longitude
<i>idx2</i>	INTEGER	Index of last longitude
<i>idy1</i>	INTEGER	Index of first latitude
<i>idy2</i>	INTEGER	Index of last latitude

### Example

To set all values in the region with the longitudes from 120E to 90W and latitudes from 20N to 20S to the constant value -1.23 use:

```
cdo setclonlatbox,-1.23,120,-90,20,-20 infile outfile
```

If the input dataset has fields on a Gaussian N16 grid, the same box can be set with [setcindexbox](#) by:

```
cdo setcindexbox,-1.23,23,48,13,20 infile outfile
```

## 2.6.14. ENLARGE - Enlarge fields

### Synopsis

```
enlarge,grid infile outfile
```

### Description

Enlarge all fields of *infile* to a user given horizontal grid. Normally only the last field element is used for the enlargement. If however the input and output grid are regular lon/lat grids, a zonal or meridional enlargement is possible. Zonal enlargement takes place, if the *xsize* of the input field is 1 and the *ysize* of both grids are the same. For meridional enlargement the *ysize* have to be 1 and the *xsize* of both grids should have the same size.

### Parameter

*grid*      STRING      Target grid description file or name

### Example

Assumed you want to add two datasets. The first dataset is a field on a global grid (n field elements) and the second dataset is a global mean (1 field element). Before you can add these two datasets the second dataset have to be enlarged to the grid size of the first dataset:

```
cdo enlarge,infile1 infile2 tmpfile
cdo add infile1 tmpfile outfile
```

Or shorter using operator piping:

```
cdo add infile1 -enlarge,infile1 infile2 outfile
```

## 2.6.15. SETMISS - Set missing value

### Synopsis

```

setmissval,newmiss infile outfile
setctomiss,c infile outfile
setmisstoc,c infile outfile
setrtomiss,rmin,rmax infile outfile
setvrange,rmin,rmax infile outfile
setmisstonn infile outfile
setmisstodis[,neighbors] infile outfile

```

### Description

This module sets part of a field to missing value or missing values to a constant value. Which part of the field is set depends on the chosen operator.

### Operators

<b>setmissval</b>	Set a new missing value $o(t, x) = \begin{cases} \text{newmiss} & \text{if } i(t, x) = \text{miss} \\ i(t, x) & \text{if } i(t, x) \neq \text{miss} \end{cases}$
<b>setctomiss</b>	Set constant to missing value $o(t, x) = \begin{cases} \text{miss} & \text{if } i(t, x) = c \\ i(t, x) & \text{if } i(t, x) \neq c \end{cases}$
<b>setmisstoc</b>	Set missing value to constant $o(t, x) = \begin{cases} c & \text{if } i(t, x) = \text{miss} \\ i(t, x) & \text{if } i(t, x) \neq \text{miss} \end{cases}$
<b>setrtomiss</b>	Set range to missing value $o(t, x) = \begin{cases} \text{miss} & \text{if } i(t, x) \geq rmin \wedge i(t, x) \leq rmax \\ i(t, x) & \text{if } i(t, x) < rmin \vee i(t, x) > rmax \end{cases}$
<b>setvrange</b>	Set valid range $o(t, x) = \begin{cases} \text{miss} & \text{if } i(t, x) < rmin \vee i(t, x) > rmax \\ i(t, x) & \text{if } i(t, x) \geq rmin \wedge i(t, x) \leq rmax \end{cases}$
<b>setmisstonn</b>	Set missing value to nearest neighbor Set all missing values to the nearest non missing value. $o(t, x) = \begin{cases} i(t, y) & \text{if } i(t, x) = \text{miss} \wedge i(t, y) \neq \text{miss} \\ i(t, x) & \text{if } i(t, x) \neq \text{miss} \end{cases}$
<b>setmisstodis</b>	Set missing value to distance-weighted average Set all missing values to the distance-weighted average of the nearest non missing values. The default number of nearest neighbors is 4.

### Parameter

<i>neighbors</i>	INTEGER	Number of nearest neighbors
<i>newmiss</i>	FLOAT	New missing value
<i>c</i>	FLOAT	Constant
<i>rmin</i>	FLOAT	Lower bound
<i>rmax</i>	FLOAT	Upper bound

## Example

### setrtomiss

Assume an input dataset has one field with temperatures in the range from 246 to 304 Kelvin. To set all values below 273.15 Kelvin to missing value use:

```
cdo setrtomiss,0,273.15 infile outfile
```

Result of 'cdo info infile':

-1 :	Date	Time	Code	Level	Size	Miss :	Minimum	Mean	Maximum
1 :	1987-12-31	12:00:00	139	0	2048	0 :	246.27	276.75	303.71

Result of 'cdo info outfile':

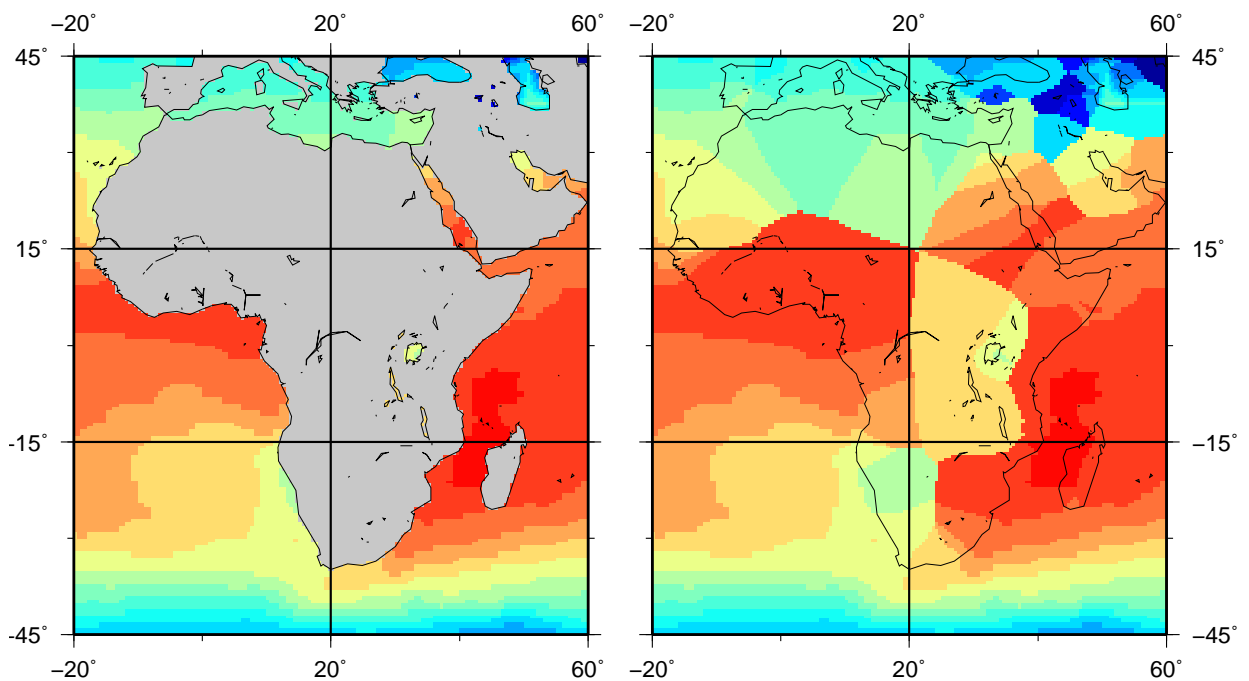
-1 :	Date	Time	Code	Level	Size	Miss :	Minimum	Mean	Maximum
1 :	1987-12-31	12:00:00	139	0	2048	871 :	273.16	287.08	303.71

### setmisstonn

Set all missing values to the nearest non missing value:

```
cdo setmisstonn infile outfile
```

Below is a schematic illustration of this example:



On the left side is input data with missing values in grey and on the right side the result with the filled missing values.

## 2.7. Arithmetic

This section contains modules to arithmetically process datasets.

Here is a short overview of all operators in this section:

<b>expr</b>	Evaluate expressions
<b>exprf</b>	Evaluate expressions script
<b>aexpr</b>	Evaluate expressions and append results
<b>aexprf</b>	Evaluate expression script and append results
<b>abs</b>	Absolute value
<b>int</b>	Integer value
<b>nint</b>	Nearest integer value
<b>pow</b>	Power
<b>sqr</b>	Square
<b>sqrt</b>	Square root
<b>exp</b>	Exponential
<b>ln</b>	Natural logarithm
<b>log10</b>	Base 10 logarithm
<b>sin</b>	Sine
<b>cos</b>	Cosine
<b>tan</b>	Tangent
<b>asin</b>	Arc sine
<b>acos</b>	Arc cosine
<b>atan</b>	Arc tangent
<b>reci</b>	Reciprocal value
<b>adde</b>	Add a constant
<b>subc</b>	Subtract a constant
<b>mulc</b>	Multiply with a constant
<b>divc</b>	Divide by a constant
<b>add</b>	Add two fields
<b>sub</b>	Subtract two fields
<b>mul</b>	Multiply two fields
<b>div</b>	Divide two fields
<b>min</b>	Minimum of two fields
<b>max</b>	Maximum of two fields
<b>atan2</b>	Arc tangent of two fields
<b>monadd</b>	Add monthly time series
<b>monsub</b>	Subtract monthly time series
<b>monmul</b>	Multiply monthly time series
<b>monddiv</b>	Divide monthly time series
<b>yhouradd</b>	Add multi-year hourly time series
<b>yhoursub</b>	Subtract multi-year hourly time series
<b>yhourmul</b>	Multiply multi-year hourly time series
<b>yhourdiv</b>	Divide multi-year hourly time series
<b>ydayadd</b>	Add multi-year daily time series
<b>ydaysub</b>	Subtract multi-year daily time series
<b>ydaymul</b>	Multiply multi-year daily time series
<b>ydaydiv</b>	Divide multi-year daily time series



---

<b>ymonadd</b>	Add multi-year monthly time series
<b>ymonsub</b>	Subtract multi-year monthly time series
<b>ymonmul</b>	Multiply multi-year monthly time series
<b>ymonddiv</b>	Divide multi-year monthly time series
<b>yseasadd</b>	Add multi-year seasonal time series
<b>yseassub</b>	Subtract multi-year seasonal time series
<b>yseasmul</b>	Multiply multi-year seasonal time series
<b>yseasdiv</b>	Divide multi-year seasonal time series
<b>muldpm</b>	Multiply with days per month
<b>divdpm</b>	Divide by days per month
<b>muldpy</b>	Multiply with days per year
<b>divdpy</b>	Divide by days per year

## 2.7.1. EXPR - Evaluate expressions

### Synopsis

```

expr,instr infile outfile
exprf,filename infile outfile
aexpr,instr infile outfile
aexprf,filename infile outfile

```

### Description

This module arithmetically processes every timestep of the input dataset. Each individual assignment statement have to end with a semi-colon. Unlike regular variables, temporary variables are never written to the output stream. To define a temporary variable simply prefix the variable name with an underscore (e.g. `_varname`) when the variable is declared.

The following operators are supported:

Operator	Meaning	Example	Result
=	assignment	<code>x = y</code>	Assigns y to x
+	addition	<code>x + y</code>	Sum of x and y
-	subtraction	<code>x - y</code>	Difference of x and y
*	multiplication	<code>x * y</code>	Product of x and y
/	division	<code>x / y</code>	Quotient of x and y
^	exponentiation	<code>x ^ y</code>	Exponentiates x with y
==	equal to	<code>x == y</code>	1, if x equal to y; else 0
!=	not equal to	<code>x != y</code>	1, if x not equal to y; else 0
>	greater than	<code>x &gt; y</code>	1, if x greater than y; else 0
<	less than	<code>x &lt; y</code>	1, if x less than y; else 0
>=	greater equal	<code>x &gt;= y</code>	1, if x greater equal y; else 0
<=	less equal	<code>x &lt;= y</code>	1, if x less equal y; else 0
<=>	less equal greater	<code>x &lt;=&gt; y</code>	-1, if x less y; 1, if x greater y; else 0
&&	logical AND	<code>x &amp;&amp; y</code>	1, if x and y not equal 0; else 0
	logical OR	<code>x    y</code>	1, if x or y not equal 0; else 0
?:	ternary conditional	<code>x ? y : z</code>	y, if x not equal 0, else z

The following functions are supported:

Math intrinsics:

```

abs(x)      Absolute value of x
floor(x)    Round to largest integral value not greater than x
ceil(x)     Round to smallest integral value not less than x
int(x)      Integer value of x
nint(x)     Nearest integer value of x
sqr(x)      Square of x
sqrt(x)     Square Root of x
exp(x)      Exponential of x
ln(x)       Natural logarithm of x
log10(x)    Base 10 logarithm of x

```

<code>sin(x)</code>	Sine of x, where x is specified in radians
<code>cos(x)</code>	Cosine of x, where x is specified in radians
<code>tan(x)</code>	Tangent of x, where x is specified in radians
<code>asin(x)</code>	Arc-sine of x, where x is specified in radians
<code>acos(x)</code>	Arc-cosine of x, where x is specified in radians
<code>atan(x)</code>	Arc-tangent of x, where x is specified in radians
<code>rad(x)</code>	Convert x from degrees to radians
<code>deg(x)</code>	Convert x from radians to degrees

## Coordinates:

<code>clon(x)</code>	Longitude coordinate of x (available only if x has geographical coordinates)
<code>clat(x)</code>	Latitude coordinate of x (available only if x has geographical coordinates)
<code>gridarea(x)</code>	Grid cell area of x (available only if x has geographical coordinates)
<code>clev(x)</code>	Level coordinate of x (0, if x is a 2D surface variable)

## Constants:

<code>ngp(x)</code>	Number of horizontal grid points
<code>nlev(x)</code>	Number of vertical levels
<code>size(x)</code>	Total number of elements ( <code>ngp(x)*nlev(x)</code> )
<code>missval(x)</code>	Returns the missing value of variable x

## Statistical values over a field:

`fldmin(x)`, `fldmax(x)`, `fldsum(x)`, `fldmean(x)`, `fldavg(x)`, `fldstd(x)`, `fldstd1(x)`, `fldvar(x)`, `fldvar1(x)`

## Vertical statistical values:

`vertmin(x)`, `vertmax(x)`, `vertsum(x)`, `vertmean(x)`, `vertavg(x)`, `vertstd(x)`, `vertstd1(x)`, `vertvar(x)`, `vertvar1(x)`

## Miscellaneous:

<code>sellevel(x,k)</code>	Select level k of variable x
<code>sellevidx(x,k)</code>	Select level index k of variable x
<code>remove(x)</code>	Remove variable x from output stream

## Operators

<b><code>expr</code></b>	Evaluate expressions The processing instructions are read from the parameter.
<b><code>exprf</code></b>	Evaluate expressions script Contrary to <code>expr</code> the processing instructions are read from a file.
<b><code>aexpr</code></b>	Evaluate expressions and append results Same as <code>expr</code> , but keep input variables and append results
<b><code>aexprf</code></b>	Evaluate expression script and append results Same as <code>exprf</code> , but keep input variables and append results

## Parameter

<i>instr</i>	STRING	Processing instructions (need to be 'quoted' in most cases)
<i>filename</i>	STRING	File with processing instructions

## Note

The `expr` commands `sellevel(x,k)` and `sellevidx(x,k)` are only available with [exprf/aexprf](#). If the input stream contains duplicate entries of the same variable name then the last one is used.

## Example

Assume an input dataset contains at least the variables 'aprl', 'aprc' and 'ts'. To create a new variable 'var1' with the sum of 'aprl' and 'aprc' and a variable 'var2' which convert the temperature 'ts' from Kelvin to Celsius use:

```
cdo expr,'var1=aprl+aprc;var2=ts-273.15;' infile outfile
```

The same example, but the instructions are read from a file:

```
cdo exprf,myexpr infile outfile
```

The file `myexpr` contains:

```
var1 = aprl + aprc;  
var2 = ts - 273.15;
```

## 2.7.2. MATH - Mathematical functions

### Synopsis

```
<operator> infile outfile
```

### Description

This module contains some standard mathematical functions. All trigonometric functions calculate with radians.

### Operators

<b>abs</b>	Absolute value $o(t, x) = \text{abs}(i(t, x))$
<b>int</b>	Integer value $o(t, x) = \text{int}(i(t, x))$
<b>nint</b>	Nearest integer value $o(t, x) = \text{nint}(i(t, x))$
<b>pow</b>	Power $o(t, x) = i(t, x)^y$
<b>sqr</b>	Square $o(t, x) = i(t, x)^2$
<b>sqrt</b>	Square root $o(t, x) = \sqrt{i(t, x)}$
<b>exp</b>	Exponential $o(t, x) = e^{i(t, x)}$
<b>ln</b>	Natural logarithm $o(t, x) = \ln(i(t, x))$
<b>log10</b>	Base 10 logarithm $o(t, x) = \log_{10}(i(t, x))$
<b>sin</b>	Sine $o(t, x) = \sin(i(t, x))$
<b>cos</b>	Cosine $o(t, x) = \cos(i(t, x))$
<b>tan</b>	Tangent $o(t, x) = \tan(i(t, x))$
<b>asin</b>	Arc sine $o(t, x) = \arcsin(i(t, x))$
<b>acos</b>	Arc cosine $o(t, x) = \arccos(i(t, x))$
<b>atan</b>	Arc tangent $o(t, x) = \arctan(i(t, x))$
<b>reci</b>	Reciprocal value $o(t, x) = 1/i(t, x)$

### Example

To calculate the square root for all field elements use:

```
cdo sqrt infile outfile
```

### 2.7.3. ARITHC - Arithmetic with a constant

#### Synopsis

```
<operator>,c infile outfile
```

#### Description

This module performs simple arithmetic with all field elements of a dataset and a constant. The fields in outfile inherit the meta data from infile.

#### Operators

<b>addc</b>	Add a constant $o(t, x) = i(t, x) + c$
<b>subc</b>	Subtract a constant $o(t, x) = i(t, x) - c$
<b>mulc</b>	Multiply with a constant $o(t, x) = i(t, x) * c$
<b>divc</b>	Divide by a constant $o(t, x) = i(t, x) / c$

#### Parameter

<i>c</i>	FLOAT	Constant
----------	-------	----------

#### Example

To sum all input fields with the constant -273.15 use:

```
cdo addc,-273.15 infile outfile
```

## 2.7.4. ARITH - Arithmetic on two datasets

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module performs simple arithmetic of two datasets. The number of fields in `infile1` should be the same as in `infile2`. The fields in `outfile` inherit the meta data from `infile1`. One of the input files can contain only one timestep or one variable.

### Operators

<b>add</b>	Add two fields $o(t, x) = i_1(t, x) + i_2(t, x)$
<b>sub</b>	Subtract two fields $o(t, x) = i_1(t, x) - i_2(t, x)$
<b>mul</b>	Multiply two fields $o(t, x) = i_1(t, x) * i_2(t, x)$
<b>div</b>	Divide two fields $o(t, x) = i_1(t, x) / i_2(t, x)$
<b>min</b>	Minimum of two fields $o(t, x) = \min(i_1(t, x), i_2(t, x))$
<b>max</b>	Maximum of two fields $o(t, x) = \max(i_1(t, x), i_2(t, x))$
<b>atan2</b>	Arc tangent of two fields The <i>atan2</i> operator calculates the arc tangent of two fields. The result is in radians, which is between -PI and PI (inclusive). $o(t, x) = \text{atan2}(i_1(t, x), i_2(t, x))$

### Example

To sum all fields of the first input file with the corresponding fields of the second input file use:

```
cdo add infile1 infile2 outfile
```

## 2.7.5. MONARITH - Monthly arithmetic

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module performs simple arithmetic of a time series and one timestep with the same month and year. For each field in `infile1` the corresponding field of the timestep in `infile2` with the same month and year is used. The header information in `infile1` have to be the same as in `infile2`. Usually `infile2` is generated by an operator of the module [MONSTAT](#).

### Operators

<b>monadd</b>	Add monthly time series Adds a time series and a monthly time series.
<b>monsub</b>	Subtract monthly time series Subtracts a time series and a monthly time series.
<b>monmul</b>	Multiply monthly time series Multiplies a time series and a monthly time series.
<b>monddiv</b>	Divide monthly time series Divides a time series and a monthly time series.

### Example

To subtract a monthly time average from a time series use:

```
cdo monsub infile -monavg infile outfile
```



## 2.7.6. YHOURARITH - Multi-year hourly arithmetic

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module performs simple arithmetic of a time series and one timestep with the same hour and day of year. For each field in `infile1` the corresponding field of the timestep in `infile2` with the same hour and day of year is used. The header information in `infile1` have to be the same as in `infile2`. Usually `infile2` is generated by an operator of the module [YHOURSTAT](#).

### Operators

<b>yhouradd</b>	Add multi-year hourly time series Adds a time series and a multi-year hourly time series.
<b>yhoursub</b>	Subtract multi-year hourly time series Subtracts a time series and a multi-year hourly time series.
<b>yhourmul</b>	Multiply multi-year hourly time series Multiplies a time series and a multi-year hourly time series.
<b>yhourdiv</b>	Divide multi-year hourly time series Divides a time series and a multi-year hourly time series.

### Example

To subtract a multi-year hourly time average from a time series use:

```
cdo yhoursub infile -yhouravg infile outfile
```

## 2.7.7. YDAYARITH - Multi-year daily arithmetic

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module performs simple arithmetic of a time series and one timestep with the same day of year. For each field in `infile1` the corresponding field of the timestep in `infile2` with the same day of year is used. The header information in `infile1` have to be the same as in `infile2`. Usually `infile2` is generated by an operator of the module [YDAYSTAT](#).

### Operators

<b>ydayadd</b>	Add multi-year daily time series Adds a time series and a multi-year daily time series.
<b>ydaysub</b>	Subtract multi-year daily time series Subtracts a time series and a multi-year daily time series.
<b>ydaymul</b>	Multiply multi-year daily time series Multiplies a time series and a multi-year daily time series.
<b>ydaydiv</b>	Divide multi-year daily time series Divides a time series and a multi-year daily time series.

### Example

To subtract a multi-year daily time average from a time series use:

```
cdo ydaysub infile -ydayavg infile outfile
```

## 2.7.8. YMONARITH - Multi-year monthly arithmetic

### Synopsis

```
<operator> infile1 infile2 outfile
```

### Description

This module performs simple arithmetic of a time series and one timestep with the same month of year. For each field in `infile1` the corresponding field of the timestep in `infile2` with the same month of year is used. The header information in `infile1` have to be the same as in `infile2`. Usually `infile2` is generated by an operator of the module [YMONSTAT](#).

### Operators

<b>ymonadd</b>	Add multi-year monthly time series Adds a time series and a multi-year monthly time series.
<b>ymonsub</b>	Subtract multi-year monthly time series Subtracts a time series and a multi-year monthly time series.
<b>ymonmul</b>	Multiply multi-year monthly time series Multiplies a time series and a multi-year monthly time series.
<b>ymonddiv</b>	Divide multi-year monthly time series Divides a time series and a multi-year monthly time series.

### Example

To subtract a multi-year monthly time average from a time series use:

```
cdo ymonsub infile -ymonavg infile outfile
```

### 2.7.9. YSEASARITH - Multi-year seasonal arithmetic

#### Synopsis

```
<operator> infile1 infile2 outfile
```

#### Description

This module performs simple arithmetic of a time series and one timestep with the same season. For each field in `infile1` the corresponding field of the timestep in `infile2` with the same season is used. The header information in `infile1` have to be the same as in `infile2`. Usually `infile2` is generated by an operator of the module [YSEASSTAT](#).

#### Operators

<b>yseasadd</b>	Add multi-year seasonal time series Adds a time series and a multi-year seasonal time series.
<b>yseassub</b>	Subtract multi-year seasonal time series Subtracts a time series and a multi-year seasonal time series.
<b>yseasmul</b>	Multiply multi-year seasonal time series Multiplies a time series and a multi-year seasonal time series.
<b>yseasdiv</b>	Divide multi-year seasonal time series Divides a time series and a multi-year seasonal time series.

#### Example

To subtract a multi-year seasonal time average from a time series use:

```
cdo yseassub infile -yseasavg infile outfile
```

### 2.7.10. ARITHDDAYS - Arithmetic with days

#### Synopsis

```
<operator> infile outfile
```

#### Description

This module multiplies or divides each timestep of a dataset with the corresponding days per month or days per year. The result of these functions depends on the used calendar of the input data.

#### Operators

<b>muldpm</b>	Multiply with days per month $o(t, x) = i(t, x) * days\_per\_month$
<b>divdpm</b>	Divide by days per month $o(t, x) = i(t, x) / days\_per\_month$
<b>muldpy</b>	Multiply with days per year $o(t, x) = i(t, x) * days\_per\_year$
<b>divdpy</b>	Divide by days per year $o(t, x) = i(t, x) / days\_per\_year$

## 2.8. Statistical values

This section contains modules to compute statistical values of datasets. 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.

This program is using the verification time to identify the time range for time-statistics. The time bounds are never used!

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

<b>sum</b>	$\sum_{i=1}^n x_i$
<b>mean</b> resp. <b>avg</b>	$n^{-1} \sum_{i=1}^n x_i$
<b>mean</b> resp. <b>avg</b> weighted by $\{w_i, i = 1, \dots, n\}$	$\left( \sum_{j=1}^n w_j \right)^{-1} \sum_{i=1}^n w_i x_i$
Variance <b>var</b>	$n^{-1} \sum_{i=1}^n (x_i - \bar{x})^2$
<b>var1</b>	$(n-1)^{-1} \sum_{i=1}^n (x_i - \bar{x})^2$
<b>var</b> weighted by $\{w_i, i = 1, \dots, n\}$	$\left( \sum_{j=1}^n w_j \right)^{-1} \sum_{i=1}^n w_i \left( x_i - \left( \sum_{j=1}^n w_j \right)^{-1} \sum_{j=1}^n w_j x_j \right)^2$
Standard deviation <b>std</b>	$\sqrt{n^{-1} \sum_{i=1}^n (x_i - \bar{x})^2}$
<b>std1</b>	$\sqrt{(n-1)^{-1} \sum_{i=1}^n (x_i - \bar{x})^2}$
<b>std</b> weighted by $\{w_i, i = 1, \dots, n\}$	$\sqrt{\left( \sum_{j=1}^n w_j \right)^{-1} \sum_{i=1}^n w_i \left( x_i - \left( \sum_{j=1}^n w_j \right)^{-1} \sum_{j=1}^n w_j x_j \right)^2}$
Cumulative Ranked Probability Score <b>crps</b>	$\int_{-\infty}^{\infty} [H(x_1) - cdf(\{x_2 \dots x_n\}) _r]^2 dr$
with $cdf(X) _r$ being the cumulative distribution function of $\{x_i, i = 2 \dots n\}$ at $r$	
and $H(x)$ the Heavyside function jumping at $x$ .	

Here is a short overview of all operators in this section:

<b>timcumsum</b>	Cumulative sum over all timesteps
<b>consecsum</b>	Consecutive Sum
<b>consects</b>	Consecutive Timesteps
<b>ensmin</b>	Ensemble minimum
<b>ensmax</b>	Ensemble maximum
<b>ensrange</b>	Ensemble range
<b>enssum</b>	Ensemble sum
<b>ensmean</b>	Ensemble mean
<b>ensavg</b>	Ensemble average
<b>ensstd</b>	Ensemble standard deviation
<b>ensstd1</b>	Ensemble standard deviation (n-1)
<b>ensvar</b>	Ensemble variance
<b>ensvar1</b>	Ensemble variance (n-1)
<b>enspctl</b>	Ensemble percentiles
<b>ensrkhistspace</b>	Ranked Histogram averaged over time
<b>ensrkhisttime</b>	Ranked Histogram averaged over space
<b>ensroc</b>	Ensemble Receiver Operating characteristics
<b>enscrps</b>	Ensemble CRPS and decomposition
<b>ensbrs</b>	Ensemble Brier score
<b>fldmin</b>	Field minimum
<b>fldmax</b>	Field maximum
<b>fldrange</b>	Field range
<b>fldsum</b>	Field sum
<b>fldmean</b>	Field mean
<b>fldavg</b>	Field average
<b>fldstd</b>	Field standard deviation
<b>fldstd1</b>	Field standard deviation (n-1)
<b>fldvar</b>	Field variance
<b>fldvar1</b>	Field variance (n-1)
<b>fldpctl</b>	Field percentiles
<b>zonmin</b>	Zonal minimum
<b>zonmax</b>	Zonal maximum
<b>zonrange</b>	Zonal range
<b>zonsum</b>	Zonal sum
<b>zonmean</b>	Zonal mean
<b>zonavg</b>	Zonal average
<b>zonstd</b>	Zonal standard deviation
<b>zonstd1</b>	Zonal standard deviation (n-1)
<b>zonvar</b>	Zonal variance
<b>zonvar1</b>	Zonal variance (n-1)
<b>zonpctl</b>	Zonal percentiles

<b>mermin</b>	Meridional minimum
<b>mermax</b>	Meridional maximum
<b>merrange</b>	Meridional range
<b>mersum</b>	Meridional sum
<b>mermean</b>	Meridional mean
<b>meravg</b>	Meridional average
<b>merstd</b>	Meridional standard deviation
<b>merstd1</b>	Meridional standard deviation (n-1)
<b>mervar</b>	Meridional variance
<b>mervar1</b>	Meridional variance (n-1)
<b>merpctl</b>	Meridional percentiles
<b>gridboxmin</b>	Gridbox minimum
<b>gridboxmax</b>	Gridbox maximum
<b>gridboxrange</b>	Gridbox range
<b>gridboxsum</b>	Gridbox sum
<b>gridboxmean</b>	Gridbox mean
<b>gridboxavg</b>	Gridbox average
<b>gridboxstd</b>	Gridbox standard deviation
<b>gridboxstd1</b>	Gridbox standard deviation (n-1)
<b>gridboxvar</b>	Gridbox variance
<b>gridboxvar1</b>	Gridbox variance (n-1)
<b>vertmin</b>	Vertical minimum
<b>vertmax</b>	Vertical maximum
<b>vertrange</b>	Vertical range
<b>vertsum</b>	Vertical sum
<b>vertmean</b>	Vertical mean
<b>vertavg</b>	Vertical average
<b>vertstd</b>	Vertical standard deviation
<b>vertstd1</b>	Vertical standard deviation (n-1)
<b>vertvar</b>	Vertical variance
<b>vertvar1</b>	Vertical variance (n-1)
<b>timselmin</b>	Time selection minimum
<b>timselmax</b>	Time selection maximum
<b>timselrange</b>	Time selection range
<b>timselsum</b>	Time selection sum
<b>timselmean</b>	Time selection mean
<b>timselavg</b>	Time selection average
<b>timselstd</b>	Time selection standard deviation
<b>timselstd1</b>	Time selection standard deviation (n-1)
<b>timselvar</b>	Time selection variance
<b>timselvar1</b>	Time selection variance (n-1)
<b>timselpctl</b>	Time range percentiles

<b>runmin</b>	Running minimum
<b>runmax</b>	Running maximum
<b>runrange</b>	Running range
<b>runsum</b>	Running sum
<b>runmean</b>	Running mean
<b>runavg</b>	Running average
<b>runstd</b>	Running standard deviation
<b>runstd1</b>	Running standard deviation (n-1)
<b>runvar</b>	Running variance
<b>runvar1</b>	Running variance (n-1)
<b>runpctl</b>	Running percentiles
<b>timmin</b>	Time minimum
<b>timmax</b>	Time maximum
<b>timrange</b>	Time range
<b>timsun</b>	Time sum
<b>timmean</b>	Time mean
<b>timavg</b>	Time average
<b>timstd</b>	Time standard deviation
<b>timstd1</b>	Time standard deviation (n-1)
<b>timvar</b>	Time variance
<b>timvar1</b>	Time variance (n-1)
<b>timpctl</b>	Time percentiles
<b>hourmin</b>	Hourly minimum
<b>hourmax</b>	Hourly maximum
<b>hourrange</b>	Hourly range
<b>hoursum</b>	Hourly sum
<b>hourmean</b>	Hourly mean
<b>houravg</b>	Hourly average
<b>hourstd</b>	Hourly standard deviation
<b>hourstd1</b>	Hourly standard deviation (n-1)
<b>hourvar</b>	Hourly variance
<b>hourvar1</b>	Hourly variance (n-1)
<b>hourpctl</b>	Hourly percentiles
<b>daymin</b>	Daily minimum
<b>daymax</b>	Daily maximum
<b>dayrange</b>	Daily range
<b>daysum</b>	Daily sum
<b>daymean</b>	Daily mean
<b>dayavg</b>	Daily average
<b>daystd</b>	Daily standard deviation
<b>daystd1</b>	Daily standard deviation (n-1)
<b>dayvar</b>	Daily variance
<b>dayvar1</b>	Daily variance (n-1)
<b>daypctl</b>	Daily percentiles



<b>monmin</b>	Monthly minimum
<b>monmax</b>	Monthly maximum
<b>monrange</b>	Monthly range
<b>monsum</b>	Monthly sum
<b>monmean</b>	Monthly mean
<b>monavg</b>	Monthly average
<b>monstd</b>	Monthly standard deviation
<b>monstd1</b>	Monthly standard deviation (n-1)
<b>monvar</b>	Monthly variance
<b>monvar1</b>	Monthly variance (n-1)
<b>monpctl</b>	Monthly percentiles
<b>yearmonmean</b>	Yearly mean from monthly data
<b>yearmin</b>	Yearly minimum
<b>yearmax</b>	Yearly maximum
<b>yearrange</b>	Yearly range
<b>yearsum</b>	Yearly sum
<b>yearmean</b>	Yearly mean
<b>yearavg</b>	Yearly average
<b>yearstd</b>	Yearly standard deviation
<b>yearstd1</b>	Yearly standard deviation (n-1)
<b>yearvar</b>	Yearly variance
<b>yearvar1</b>	Yearly variance (n-1)
<b>yearpctl</b>	Yearly percentiles
<b>seasmin</b>	Seasonal minimum
<b>seasmax</b>	Seasonal maximum
<b>seasrange</b>	Seasonal range
<b>seassum</b>	Seasonal sum
<b>seasmean</b>	Seasonal mean
<b>seasavg</b>	Seasonal average
<b>seasstd</b>	Seasonal standard deviation
<b>seasstd1</b>	Seasonal standard deviation (n-1)
<b>seasvar</b>	Seasonal variance
<b>seasvar1</b>	Seasonal variance (n-1)
<b>seaspctl</b>	Seasonal percentiles
<b>yhourmin</b>	Multi-year hourly minimum
<b>yhourmax</b>	Multi-year hourly maximum
<b>yhourrange</b>	Multi-year hourly range
<b>yhoursum</b>	Multi-year hourly sum
<b>yhourmean</b>	Multi-year hourly mean
<b>yhouravg</b>	Multi-year hourly average
<b>yhourstd</b>	Multi-year hourly standard deviation
<b>yhourstd1</b>	Multi-year hourly standard deviation (n-1)
<b>yhourvar</b>	Multi-year hourly variance
<b>yhourvar1</b>	Multi-year hourly variance (n-1)

<b>ydaymin</b>	Multi-year daily minimum
<b>ydaymax</b>	Multi-year daily maximum
<b>ydayrange</b>	Multi-year daily range
<b>ydaysum</b>	Multi-year daily sum
<b>ydaymean</b>	Multi-year daily mean
<b>ydayavg</b>	Multi-year daily average
<b>ydaystd</b>	Multi-year daily standard deviation
<b>ydaystd1</b>	Multi-year daily standard deviation (n-1)
<b>ydayvar</b>	Multi-year daily variance
<b>ydayvar1</b>	Multi-year daily variance (n-1)
<b>ydaypctl</b>	Multi-year daily percentiles
<b>ymonmin</b>	Multi-year monthly minimum
<b>ymonmax</b>	Multi-year monthly maximum
<b>ymonrange</b>	Multi-year monthly range
<b>ymonsum</b>	Multi-year monthly sum
<b>ymonmean</b>	Multi-year monthly mean
<b>ymonavg</b>	Multi-year monthly average
<b>ymonstd</b>	Multi-year monthly standard deviation
<b>ymonstd1</b>	Multi-year monthly standard deviation (n-1)
<b>ymonvar</b>	Multi-year monthly variance
<b>ymonvar1</b>	Multi-year monthly variance (n-1)
<b>ymonpctl</b>	Multi-year monthly percentiles
<b>yseasmin</b>	Multi-year seasonal minimum
<b>yseasmax</b>	Multi-year seasonal maximum
<b>yseasrange</b>	Multi-year seasonal range
<b>yseassum</b>	Multi-year seasonal sum
<b>yseasmean</b>	Multi-year seasonal mean
<b>yseasavg</b>	Multi-year seasonal average
<b>yseasstd</b>	Multi-year seasonal standard deviation
<b>yseasstd1</b>	Multi-year seasonal standard deviation (n-1)
<b>yseasvar</b>	Multi-year seasonal variance
<b>yseasvar1</b>	Multi-year seasonal variance (n-1)
<b>yseaspctl</b>	Multi-year seasonal percentiles
<b>ydrunmin</b>	Multi-year daily running minimum
<b>ydrunmax</b>	Multi-year daily running maximum
<b>ydrunsum</b>	Multi-year daily running sum
<b>ydrunmean</b>	Multi-year daily running mean
<b>ydrunavg</b>	Multi-year daily running average
<b>ydrunstd</b>	Multi-year daily running standard deviation
<b>ydrunstd1</b>	Multi-year daily running standard deviation (n-1)
<b>ydrunvar</b>	Multi-year daily running variance
<b>ydrunvar1</b>	Multi-year daily running variance (n-1)
<b>ydrunpctl</b>	Multi-year daily running percentiles

### 2.8.1. TIMCUMSUM - Cumulative sum over all timesteps

#### Synopsis

```
timcumsum infile outfile
```

#### Description

The timcumsum operator calculates the cumulative sum over all timesteps. Missing values are treated as numeric zero when summing.

$$o(t, x) = \text{sum}\{i(t', x), 0 < t' \leq t\}$$

### 2.8.2. CONSECSTAT - Consecutive timestep periods

#### Synopsis

```
<operator> infile outfile
```

#### Description

This module computes periods over all timesteps in `infile` where a certain property is valid. The property can be chosen by creating a mask from the original data, which is the expected input format for operators of this module. Depending on the operator full information about each period or just its length and ending date are computed.

#### Operators

<b>consecsum</b>	<p>Consecutive Sum</p> <p>This operator computes periods of consecutive timesteps similar to a <a href="#">runsum</a>, but periods are finished, when the mask value is 0. That way multiple periods can be found. Timesteps from the input are preserved. Missing values are handled like 0, i.e. finish periods of consecutive timesteps.</p>
<b>consects</b>	<p>Consecutive Timesteps</p> <p>In contrast to the operator above <code>consects</code> only computes the length of each period together with its last timestep. To be able to perform statistical analysis like min, max or mean, everything else is set to missing value.</p>

#### Example

For a given time series of daily temperatures, the periods of summer days can be calculated with inplace masking the input field:

```
cdo consects -gtc,20.0 infile1 outfile
```

### 2.8.3. ENSSTAT - Statistical values over an ensemble

#### Synopsis

`<operator> infiles outfile`

`enspctl,p infiles outfile`

#### Description

This module computes statistical values over an ensemble of input files. Depending on the chosen operator the minimum, maximum, range, sum, average, variance, standard deviation or a certain percentile over all input files is written to outfile. All input files need to have the same structure with the same variables. The date information of a timestep in outfile is the date of the first input file.

#### Operators

<b>ensmin</b>	Ensemble minimum $o(t, x) = \mathbf{min}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensmax</b>	Ensemble maximum $o(t, x) = \mathbf{max}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensrange</b>	Ensemble range $o(t, x) = \mathbf{range}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>enssum</b>	Ensemble sum $o(t, x) = \mathbf{sum}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensmean</b>	Ensemble mean $o(t, x) = \mathbf{mean}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensavg</b>	Ensemble average $o(t, x) = \mathbf{avg}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensstd</b>	Ensemble standard deviation Normalize by n. $o(t, x) = \mathbf{std}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensstd1</b>	Ensemble standard deviation (n-1) Normalize by (n-1). $o(t, x) = \mathbf{std1}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensvar</b>	Ensemble variance Normalize by n. $o(t, x) = \mathbf{var}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>ensvar1</b>	Ensemble variance (n-1) Normalize by (n-1). $o(t, x) = \mathbf{var1}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$
<b>enspctl</b>	Ensemble percentiles $o(t, x) = \mathbf{pth\ percentile}\{i_1(t, x), i_2(t, x), \dots, i_n(t, x)\}$

#### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

## Note

This operator needs to open all input files simultaneously. The maximum number of open files depends on the operating system!

## Example

To compute the ensemble mean over 6 input files use:

```
cdo ensmean infile1 infile2 infile3 infile4 infile5 infile6 outfile
```

Or shorter with filename substitution:

```
cdo ensmean infile[1-6] outfile
```

To compute the 50th percentile (median) over 6 input files use:

```
cdo enspctl,50 infile1 infile2 infile3 infile4 infile5 infile6 outfile
```

## 2.8.4. ENSSTAT2 - Statistical values over an ensemble

### Synopsis

```
<operator> obsfile ensfiles outfile
```

### Description

This module computes statistical values over the ensemble of `ensfiles` using `obsfile` as a reference. Depending on the operator a ranked Histogram or a roc-curve over all Ensembles `ensfiles` with reference to `obsfile` is written to `outfile`. The date and grid information of a timestep in `outfile` is the date of the first input file. Thus all input files are required to have the same structure in terms of the gridsize, variable definitions and number of timesteps.

All Operators in this module use `obsfile` as the reference (for instance an observation) whereas `ensfiles` are understood as an ensemble consisting of `n` (where `n` is the number of `ensfiles`) members.

The operators `ensrkhistspace` and `ensrkhisttime` compute Ranked Histograms. Therefor the vertical axis is utilized as the Histogram axis, which prohibits the use of files containing more than one level. The histogram axis has `nensfiles+1` bins with level 0 containing for each grid point the number of observations being smaller as all ensembles and level `nensfiles+1` indicating the number of observations being larger than all ensembles.

`ensrkhistspace` computes a ranked histogram at each timestep reducing each horizontal grid to a 1x1 grid and keeping the time axis as in `obsfile`. Contrary `ensrkhisttime` computes a histogram at each grid point keeping the horizontal grid for each variable and reducing the time-axis. The time information is that from the last timestep in `obsfile`.

### Operators

<b>ensrkhistspace</b>	Ranked Histogram averaged over time
<b>ensrkhisttime</b>	Ranked Histogram averaged over space
<b>ensroc</b>	Ensemble Receiver Operating characteristics

### Example

To compute a rank histogram over 5 input files `ensfile1-ensfile5` given an observation in `obsfile` use:

```
cdo ensrkhisttime obsfile ensfile1 ensfile2 ensfile3 ensfile4 ensfile5 outfile
```

Or shorter with filename substitution:

```
cdo ensrkhisttime obsfile ensfile[1-5] outfile
```

## 2.8.5. ENSVAL - Ensemble validation tools

### Synopsis

```
enscrps rfile infiles outfilebase
ensbrs,x rfile infiles outfilebase
```

### Description

This module computes ensemble validation scores and their decomposition such as the Brier and cumulative ranked probability score (CRPS). The first file is used as a reference it can be a climatology, observation or reanalysis against which the skill of the ensembles given in infiles is measured. Depending on the operator a number of output files is generated each containing the skill score and its decomposition corresponding to the operator. The output is averaged over horizontal fields using appropriate weights for each level and timestep in rfile.

All input files need to have the same structure with the same variables. The date information of a timestep in outfile is the date of the first input file. The output files are named as `<outfilebase>.<type>.<filesuffix>` where `<type>` depends on the operator and `<filesuffix>` is determined from the output file type. There are three output files for operator `enscrps` and four output files for operator `ensbrs`.

The CRPS and its decomposition into Reliability and the potential CRPS are calculated by an appropriate averaging over the field members (note, that the CRPS does *\*not\** average linearly). In the three output files `<type>` has the following meaning: `crps` for the CRPS, `reli` for the reliability and `crpspot` for the potential crps. The relation  $CRPS = CRPS_{pot} + RELI$

holds.

The Brier score of the Ensemble given by infiles with respect to the reference given in rfile and the threshold `x` is calculated. In the four output files `<type>` has the following meaning: `brs` for the Brier score wrt threshold `x`; `brsreli` for the Brier score reliability wrt threshold `x`; `brsreso` for the Brier score resolution wrt threshold `x`; `brsunct` for the Brier score uncertainty wrt threshold `x`. In analogy to the CRPS the following relation holds:  $BRS(x) = RELI(x) - RESO(x) + UNCT(x)$ .

The implementation of the decomposition of the CRPS and Brier Score follows Hans Hersbach (2000): Decomposition of the Continuous Ranked Probability Score for Ensemble Prediction Systems, in: Weather and Forecasting (15) pp. 559-570.

The CRPS code decomposition has been verified against the CRAN - ensemble validation package from R. Differences occur when grid-cell area is not uniform as the implementation in R does not account for that.

### Operators

<b>enscrps</b>	Ensemble CRPS and decomposition
<b>ensbrs</b>	Ensemble Brier score Ensemble Brier Score and Decomposition

### Example

To compute the field averaged Brier score at `x=5` over an ensemble with 5 members `ensfile1-5` w.r.t. the reference `rfile` and write the results to files `obase.brs.<suff>`, `obase.brsreli<suff>`, `obase.brsreso<suff>`, `obase.brsunct<suff>` where `<suff>` is determined from the output file type, use

```
cdo ensbrs,5 rfile ensfile1 ensfile2 ensfile3 ensfile4 ensfile5 obase
```

or shorter using file name substitution:

```
cdo ensbrs,5 rfile ensfile[1-5] obase
```



## 2.8.6. FLDSTAT - Statistical values over a field

### Synopsis

```
<operator> infile outfile
```

```
fldpctl,p infile outfile
```

### Description

This module computes statistical values of the input fields. According to the chosen operator the field minimum, maximum, range, sum, average, variance, standard deviation or a certain percentile is written to outfile.

### Operators

<b>fldmin</b>	Field minimum For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{min}\{i(t, x'), x_1 < x' \leq x_n\}$
<b>fldmax</b>	Field maximum For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{max}\{i(t, x'), x_1 < x' \leq x_n\}$
<b>fldrange</b>	Field range For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{range}\{i(t, x'), x_1 < x' \leq x_n\}$
<b>fldsum</b>	Field sum For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{sum}\{i(t, x'), x_1 < x' \leq x_n\}$
<b>fldmean</b>	Field mean For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{mean}\{i(t, x'), x_1 < x' \leq x_n\}$  weighted by area weights obtained by the input field.
<b>fldavg</b>	Field average For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{avg}\{i(t, x'), x_1 < x' \leq x_n\}$  weighted by area weights obtained by the input field.
<b>fldstd</b>	Field standard deviation Normalize by n. For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{std}\{i(t, x'), x_1 < x' \leq x_n\}$  weighted by area weights obtained by the input field.
<b>fldstd1</b>	Field standard deviation (n-1) Normalize by (n-1). For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{std1}\{i(t, x'), x_1 < x' \leq x_n\}$  weighted by area weights obtained by the input field.
<b>fldvar</b>	Field variance Normalize by n. For every gridpoint $x_1, \dots, x_n$ of the same field it is: $o(t, 1) = \mathbf{var}\{i(t, x'), x_1 < x' \leq x_n\}$  weighted by area weights obtained by the input field.

- fldvar1** Field variance (n-1)  
 Normalize by (n-1). For every gridpoint  $x_1, \dots, x_n$  of the same field it is:  

$$o(t, 1) = \mathbf{var1}\{i(t, x'), x_1 < x' \leq x_n\}$$
  
 weighted by area weights obtained by the input field.
- fldpctl** Field percentiles  
 For every gridpoint  $x_1, \dots, x_n$  of the same field it is:  

$$o(t, 1) = \mathbf{pth\ percentile}\{i(t, x'), x_1 < x' \leq x_n\}$$

## Parameter

$p$       FLOAT      Percentile number in 0, ..., 100

## Example

To compute the field mean of all input fields use:

```
cdo fldmean infile outfile
```

To compute the 90th percentile of all input fields use:

```
cdo fldpctl,90 infile outfile
```

## 2.8.7. ZONSTAT - Zonal statistical values

### Synopsis

```
<operator> infile outfile
zonpctl,p infile outfile
```

### Description

This module computes zonal statistical values of the input fields. According to the chosen operator the zonal minimum, maximum, range, sum, average, variance, standard deviation or a certain percentile is written to outfile. This operator requires all variables on the same regular lon/lat grid.

### Operators

<b>zonmin</b>	Zonal minimum For every latitude the minimum over all longitudes is computed.
<b>zonmax</b>	Zonal maximum For every latitude the maximum over all longitudes is computed.
<b>zonrange</b>	Zonal range For every latitude the range over all longitudes is computed.
<b>zonsum</b>	Zonal sum For every latitude the sum over all longitudes is computed.
<b>zonmean</b>	Zonal mean For every latitude the mean over all longitudes is computed.
<b>zonavg</b>	Zonal average For every latitude the average over all longitudes is computed.
<b>zonstd</b>	Zonal standard deviation For every latitude the standard deviation over all longitudes is computed. Normalize by n.
<b>zonstd1</b>	Zonal standard deviation (n-1) For every latitude the standard deviation over all longitudes is computed. Normalize by (n-1).
<b>zonvar</b>	Zonal variance For every latitude the variance over all longitudes is computed. Normalize by n.
<b>zonvar1</b>	Zonal variance (n-1) For every latitude the variance over all longitudes is computed. Normalize by (n-1).
<b>zonpctl</b>	Zonal percentiles For every latitude the pth percentile over all longitudes is computed.

### Parameter

*p*      FLOAT      Percentile number in 0, ..., 100

### Example

To compute the zonal mean of all input fields use:

```
cdo zonmean infile outfile
```

To compute the 50th meridional percentile (median) of all input fields use:

```
cdo zonpctl,50 infile outfile
```

## 2.8.8. MERSTAT - Meridional statistical values

### Synopsis

```
<operator> infile outfile
```

```
merpctl,p infile outfile
```

### Description

This module computes meridional statistical values of the input fields. According to the chosen operator the meridional minimum, maximum, range, sum, average, variance, standard deviation or a certain percentile is written to outfile. This operator requires all variables on the same regular lon/lat grid.

### Operators

<b>mermin</b>	Meridional minimum For every longitude the minimum over all latitudes is computed.
<b>mermax</b>	Meridional maximum For every longitude the maximum over all latitudes is computed.
<b>merrange</b>	Meridional range For every longitude the range over all latitudes is computed.
<b>mersum</b>	Meridional sum For every longitude the sum over all latitudes is computed.
<b>mermean</b>	Meridional mean For every longitude the area weighted mean over all latitudes is computed.
<b>meravg</b>	Meridional average For every longitude the area weighted average over all latitudes is computed.
<b>merstd</b>	Meridional standard deviation For every longitude the standard deviation over all latitudes is computed. Normalize by n.
<b>merstd1</b>	Meridional standard deviation (n-1) For every longitude the standard deviation over all latitudes is computed. Normalize by (n-1).
<b>mervar</b>	Meridional variance For every longitude the variance over all latitudes is computed. Normalize by n.
<b>mervar1</b>	Meridional variance (n-1) For every longitude the variance over all latitudes is computed. Normalize by (n-1).
<b>merpctl</b>	Meridional percentiles For every longitude the pth percentile over all latitudes is computed.

### Parameter

<i>p</i>	FLOAT	Percentile number in 0, ..., 100
----------	-------	----------------------------------

**Example**

To compute the meridional mean of all input fields use:

```
cdo mermean infile outfile
```

To compute the 50th meridional percentile (median) of all input fields use:

```
cdo merpctl,50 infile outfile
```

## 2.8.9. GRIDBOXSTAT - Statistical values over grid boxes

### Synopsis

```
<operator> ,nx,ny infile outfile
```

### Description

This module computes statistical values over surrounding grid boxes. According to the chosen operator the minimum, maximum, range, sum, average, variance, or standard deviation of the neighboring grid boxes is written to outfile. All gridbox operators only works on quadrilateral curvilinear grids.

### Operators

<b>gridboxmin</b>	Gridbox minimum Minimum value of the selected grid boxes.
<b>gridboxmax</b>	Gridbox maximum Maximum value of the selected grid boxes.
<b>gridboxrange</b>	Gridbox range Range (max-min value) of the selected grid boxes.
<b>gridboxsum</b>	Gridbox sum Sum of the selected grid boxes.
<b>gridboxmean</b>	Gridbox mean Mean of the selected grid boxes.
<b>gridboxavg</b>	Gridbox average Average of the selected grid boxes.
<b>gridboxstd</b>	Gridbox standard deviation Standard deviation of the selected grid boxes. Normalize by n.
<b>gridboxstd1</b>	Gridbox standard deviation (n-1) Standard deviation of the selected grid boxes. Normalize by (n-1).
<b>gridboxvar</b>	Gridbox variance Variance of the selected grid boxes. Normalize by n.
<b>gridboxvar1</b>	Gridbox variance (n-1) Variance of the selected grid boxes. Normalize by (n-1).

### Parameter

<i>nx</i>	INTEGER	Number of grid boxes in x direction
<i>ny</i>	INTEGER	Number of grid boxes in y direction

### Example

To compute the mean over 10x10 grid boxes of the input field use:

```
cdo gridboxmean,10,10 infile outfile
```

## 2.8.10. VERTSTAT - Vertical statistical values

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over all levels of the input variables. According to chosen operator the vertical minimum, maximum, range, sum, average, variance or standard deviation is written to outfile.

### Operators

<b>vertmin</b>	Vertical minimum For every gridpoint the minimum over all levels is computed.
<b>vertmax</b>	Vertical maximum For every gridpoint the maximum over all levels is computed.
<b>vertrange</b>	Vertical range For every gridpoint the range over all levels is computed.
<b>vertsum</b>	Vertical sum For every gridpoint the sum over all levels is computed.
<b>vertmean</b>	Vertical mean For every gridpoint the layer weighted mean over all levels is computed.
<b>vertavg</b>	Vertical average For every gridpoint the layer weighted average over all levels is computed.
<b>vertstd</b>	Vertical standard deviation For every gridpoint the standard deviation over all levels is computed. Normalize by n.
<b>vertstd1</b>	Vertical standard deviation (n-1) For every gridpoint the standard deviation over all levels is computed. Normalize by (n-1).
<b>vertvar</b>	Vertical variance For every gridpoint the variance over all levels is computed. Normalize by n.
<b>vertvar1</b>	Vertical variance (n-1) For every gridpoint the variance over all levels is computed. Normalize by (n-1).

### Example

To compute the vertical sum of all input variables use:

```
cdo vertsum infile outfile
```

## 2.8.11. TIMSELSTAT - Time range statistical values

### Synopsis

`<operator>,nsets[,noffset[,nskip]] infile outfile`

### Description

This module computes statistical values for a selected number of timesteps. According to the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of the selected timesteps is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

### Operators

<b>timselmin</b>	Time selection minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \min\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselmax</b>	Time selection maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \max\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselrange</b>	Time selection range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselsum</b>	Time selection sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselmean</b>	Time selection mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselavg</b>	Time selection average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselstd</b>	Time selection standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselstd1</b>	Time selection standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timselvar</b>	Time selection variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same selected time range it is: $o(t, x) = \text{var}\{i(t', x), t_1 < t' \leq t_n\}$



**timselvar1** Time selection variance (n-1)  
 Normalize by (n-1). For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same selected time range it is:  

$$o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$$

### Parameter

<i>nsets</i>	INTEGER	Number of input timesteps for each output timestep
<i>noffset</i>	INTEGER	Number of input timesteps skipped before the first timestep range (optional)
<i>nskip</i>	INTEGER	Number of input timesteps skipped between timestep ranges (optional)

### Example

Assume an input dataset has monthly means over several years. To compute seasonal means from monthly means the first two month have to be skipped:

```
cdo timselmean,3,2 infile outfile
```

## 2.8.12. TIMSELPCTL - Time range percentile values

### Synopsis

```
timselpctl,p,nsets[,noffset[,nskip]] infile1 infile2 infile3 outfile
```

### Description

This operator computes percentile values over a selected number of timesteps in *infile1*. The algorithm uses histograms with minimum and maximum bounds given in *infile2* and *infile3*, respectively. The default number of histogram bins is 101. The default can be overridden by setting the environment variable CDO\_PCTL\_NBINS to a different value. The files *infile2* and *infile3* should be the result of corresponding [timselmin](#) and [timselmax](#) operations, respectively. The time of *outfile* is determined by the time in the middle of all contributing timesteps of *infile1*.

For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same selected time range it is:

$$o(t, x) = \mathbf{pth\ percentile}\{i(t', x), t_1 < t' \leq t_n\}$$

### Parameter

<i>p</i>	FLOAT	Percentile number in 0, ..., 100
<i>nsets</i>	INTEGER	Number of input timesteps for each output timestep
<i>noffset</i>	INTEGER	Number of input timesteps skipped before the first timestep range (optional)
<i>nskip</i>	INTEGER	Number of input timesteps skipped between timestep ranges (optional)

### Environment

CDO\_PCTL\_NBINS Sets the number of histogram bins. The default number is 101.

## 2.8.13. RUNSTAT - Running statistical values

### Synopsis

`<operator>,nts infile outfile`

### Description

This module computes running statistical values over a selected number of timesteps. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of a selected number of consecutive timesteps read from `infile` is written to `outfile`. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile`.

### Operators

<b>runmin</b>	Running minimum $o(t + (nts - 1)/2, x) = \mathbf{min}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runmax</b>	Running maximum $o(t + (nts - 1)/2, x) = \mathbf{max}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runrange</b>	Running range $o(t + (nts - 1)/2, x) = \mathbf{range}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runsum</b>	Running sum $o(t + (nts - 1)/2, x) = \mathbf{sum}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runmean</b>	Running mean $o(t + (nts - 1)/2, x) = \mathbf{mean}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runavg</b>	Running average $o(t + (nts - 1)/2, x) = \mathbf{avg}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runstd</b>	Running standard deviation Normalize by n. $o(t + (nts - 1)/2, x) = \mathbf{std}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runstd1</b>	Running standard deviation (n-1) Normalize by (n-1). $o(t + (nts - 1)/2, x) = \mathbf{std1}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runvar</b>	Running variance Normalize by n. $o(t + (nts - 1)/2, x) = \mathbf{var}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$
<b>runvar1</b>	Running variance (n-1) Normalize by (n-1). $o(t + (nts - 1)/2, x) = \mathbf{var1}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$

### Parameter

`nts`    INTEGER    Number of timesteps

### Environment

`CDO_TIMESTAT_DATE`    Sets the time stamp in `outfile` to the "first", "middle" or "last" contributing timestep of `infile`.

**Example**

To compute the running mean over 9 timesteps use:

```
cdo runmean,9 infile outfile
```

**2.8.14. RUNPCTL - Running percentile values****Synopsis**

```
runpctl,p,nts infile outfile
```

**Description**

This module computes running percentiles over a selected number of timesteps in `infile`. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile`.

$$o(t + (nts - 1)/2, x) = \mathbf{pth\ percentile}\{i(t, x), i(t + 1, x), \dots, i(t + nts - 1, x)\}$$

**Parameter**

<i>p</i>	FLOAT	Percentile number in 0, ..., 100
<i>nts</i>	INTEGER	Number of timesteps

**Example**

To compute the running 50th percentile (median) over 9 timesteps use:

```
cdo runpctl,50,9 infile outfile
```

## 2.8.15. TIMSTAT - Statistical values over all timesteps

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over all timesteps in `infile`. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of all timesteps read from `infile` is written to `outfile`. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile`.

### Operators

<b>timmin</b>	Time minimum $o(1, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timmax</b>	Time maximum $o(1, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timrange</b>	Time range $o(1, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timsun</b>	Time sum $o(1, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timmean</b>	Time mean $o(1, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timavg</b>	Time average $o(1, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timstd</b>	Time standard deviation Normalize by n. $o(1, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timstd1</b>	Time standard deviation (n-1) Normalize by (n-1). $o(1, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timvar</b>	Time variance Normalize by n. $o(1, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>timvar1</b>	Time variance (n-1) Normalize by (n-1). $o(1, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$

### Example

To compute the mean over all input timesteps use:

```
cdo timmean infile outfile
```

## 2.8.16. TIMPCTL - Percentile values over all timesteps

### Synopsis

```
timctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps in `infile1`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding `timmin` and `timmax` operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`.

$$o(1, x) = \text{pth percentile}\{i(t', x), t_1 < t' \leq t_n\}$$

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

### Example

To compute the 90th percentile over all input timesteps use:

```
cdo timmin infile minfile
cdo timmax infile maxfile
cdo timctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo timctl,90 infile -timmin infile -timmax infile outfile
```

## 2.8.17. HOURSTAT - Hourly statistical values

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over timesteps of the same hour. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of timesteps of the same hour is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

### Operators

<b>hourmin</b>	Hourly minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourmax</b>	Hourly maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourrange</b>	Hourly range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hoursum</b>	Hourly sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourmean</b>	Hourly mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>houravg</b>	Hourly average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourstd</b>	Hourly standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourstd1</b>	Hourly standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourvar</b>	Hourly variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>hourvar1</b>	Hourly variance (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same hour it is: $o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$

## Example

To compute the hourly mean of a time series use:

```
cdo hourmean infile outfile
```

## 2.8.18. HOURPCTL - Hourly percentile values

### Synopsis

```
hourpctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps of the same hour in `infile1`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding [hourmin](#) and [hourmax](#) operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`.

For every adjacent sequence  $t_{-1}, \dots, t_n$  of timesteps of the same hour it is:

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

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

## Example

To compute the hourly 90th percentile of a time series use:

```
cdo hourmin infile minfile
cdo hourmax infile maxfile
cdo hourpctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo hourpctl,90 infile -hourmin infile -hourmax infile outfile
```

## 2.8.19. DAYSTAT - Daily statistical values

### Synopsis

`<operator> infile outfile`

### Description

This module computes statistical values over timesteps of the same day. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of timesteps of the same day is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

### Operators

<b>daymin</b>	Daily minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>daymax</b>	Daily maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>dayrange</b>	Daily range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>daysum</b>	Daily sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>daymean</b>	Daily mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>dayavg</b>	Daily average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>daystd</b>	Daily standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>daystd1</b>	Daily standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>dayvar</b>	Daily variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>dayvar1</b>	Daily variance (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same day it is: $o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$



## Example

To compute the daily mean of a time series use:

```
cdo daymean infile outfile
```

## 2.8.20. DAYPCTL - Daily percentile values

### Synopsis

```
daypctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps of the same day in `infile1`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding [daymin](#) and [daymax](#) operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`.

For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same day it is:

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

### Parameter

$p$	FLOAT	Percentile number in 0, ..., 100
-----	-------	----------------------------------

### Environment

<code>CDO_PCTL_NBINS</code>	Sets the number of histogram bins. The default number is 101.
-----------------------------	---

## Example

To compute the daily 90th percentile of a time series use:

```
cdo daymin infile minfile
cdo daymax infile maxfile
cdo daypctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo daypctl,90 infile -daymin infile -daymax infile outfile
```

## 2.8.21. MONSTAT - Monthly statistical values

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over timesteps of the same month. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of timesteps of the same month is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

### Operators

<b>monmin</b>	Monthly minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monmax</b>	Monthly maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monrange</b>	Monthly range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monsum</b>	Monthly sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monmean</b>	Monthly mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monavg</b>	Monthly average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monstd</b>	Monthly standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monstd1</b>	Monthly standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monvar</b>	Monthly variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>monvar1</b>	Monthly variance (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same month it is: $o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$

## Example

To compute the monthly mean of a time series use:

```
cdo monmean infile outfile
```

## 2.8.22. MONPCTL - Monthly percentile values

### Synopsis

```
monpctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps of the same month in `infile1`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding [monmin](#) and [monmax](#) operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`.

For every adjacent sequence  $t_{-1}, \dots, t_n$  of timesteps of the same month it is:

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

### Parameter

<code>p</code>	FLOAT	Percentile number in 0, ..., 100
----------------	-------	----------------------------------

### Environment

<code>CDO_PCTL_NBINS</code>	Sets the number of histogram bins. The default number is 101.
-----------------------------	---

## Example

To compute the monthly 90th percentile of a time series use:

```
cdo monmin infile minfile
cdo monmax infile maxfile
cdo monpctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo monpctl,90 infile -monmin infile -monmax infile outfile
```

## 2.8.23. YEARMONSTAT - Yearly mean from monthly data

### Synopsis

```
yearmonmean infile outfile
```

### Description

This operator computes the yearly mean of a monthly time series. Each month is weighted with the number of days per month. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same year it is:

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

### Environment

CDO\_TIMESTAT\_DATE      Sets the date information in outfile to the "first", "middle" or "last" contributing timestep of infile.

### Example

To compute the yearly mean of a monthly time series use:

```
cdo yearmonmean infile outfile
```

## 2.8.24. YEARSTAT - Yearly statistical values

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over timesteps of the same year. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of timesteps of the same year is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile.

### Operators

<b>yearmin</b>	Yearly minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearmax</b>	Yearly maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearrange</b>	Yearly range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearsum</b>	Yearly sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearmean</b>	Yearly mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearavg</b>	Yearly average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearstd</b>	Yearly standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearstd1</b>	Yearly standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearvar</b>	Yearly variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>yearvar1</b>	Yearly variance (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same year it is: $o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$

### Note

The operators yearmean and yearavg compute only arithmetical means!

## Example

To compute the yearly mean of a time series use:

```
cdo yearmean infile outfile
```

To compute the yearly mean from the correct weighted monthly mean use:

```
cdo yearmonmean infile outfile
```

## 2.8.25. YEARPCTL - Yearly percentile values

### Synopsis

```
yearpctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps of the same year in `infile1`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding [yearmin](#) and [yearmax](#) operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`.

For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same year it is:

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

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

## Example

To compute the yearly 90th percentile of a time series use:

```
cdo yearmin infile minfile
cdo yearmax infile maxfile
cdo yearpctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo yearpctl,90 infile -yearmin infile -yearmax infile outfile
```

## 2.8.26. SEASSTAT - Seasonal statistical values

### Synopsis

```
<operator> infile outfile
```

### Description

This module computes statistical values over timesteps of the same season. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of timesteps of the same season is written to outfile. The time of outfile is determined by the time in the middle of all contributing timesteps of infile. Be careful about the first and the last output timestep, they may be incorrect values if the seasons have incomplete timesteps.

### Operators

<b>seasmin</b>	Seasonal minimum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{min}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasmax</b>	Seasonal maximum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{max}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasrange</b>	Seasonal range For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{range}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seassum</b>	Seasonal sum For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{sum}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasmean</b>	Seasonal mean For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{mean}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasavg</b>	Seasonal average For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{avg}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasstd</b>	Seasonal standard deviation Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{std}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasstd1</b>	Seasonal standard deviation (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{std1}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasvar</b>	Seasonal variance Normalize by n. For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{var}\{i(t', x), t_1 < t' \leq t_n\}$
<b>seasvar1</b>	Seasonal variance (n-1) Normalize by (n-1). For every adjacent sequence $t_1, \dots, t_n$ of timesteps of the same season it is: $o(t, x) = \mathbf{var1}\{i(t', x), t_1 < t' \leq t_n\}$

## Example

To compute the seasonal mean of a time series use:

```
cdo seasmean infile outfile
```

## 2.8.27. SEASPCTL - Seasonal percentile values

### Synopsis

```
seaspctl,p infile1 infile2 infile3 outfile
```

### Description

This operator computes percentiles over all timesteps in `infile1` of the same season. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by defining the environment variable `CDO_PCTL_NBINS`. The files `infile2` and `infile3` should be the result of corresponding [seasmin](#) and [seasmax](#) operations, respectively. The time of `outfile` is determined by the time in the middle of all contributing timesteps of `infile1`. Be careful about the first and the last output timestep, they may be incorrect values if the seasons have incomplete timesteps.

For every adjacent sequence  $t_1, \dots, t_n$  of timesteps of the same season it is:

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

### Parameter

<code>p</code>	FLOAT	Percentile number in 0, ..., 100
----------------	-------	----------------------------------

### Environment

<code>CDO_PCTL_NBINS</code>	Sets the number of histogram bins. The default number is 101.
-----------------------------	---

## Example

To compute the seasonal 90th percentile of a time series use:

```
cdo seasmin infile minfile
cdo seasmax infile maxfile
cdo seaspctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo seaspctl,90 infile -seasmin infile -seasmax infile outfile
```



## 2.8.28. YHOURSTAT - Multi-year hourly statistical values

### Synopsis

`<operator> infile outfile`

### Description

This module computes statistical values of each hour and day of year. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of each hour and day of year in `infile` is written to `outfile`. The date information in an output field is the date of the last contributing input field.

### Operators

<b>yhourmin</b>	Multi-year hourly minimum $o(0001, x) = \min\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \min\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourmax</b>	Multi-year hourly maximum $o(0001, x) = \max\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \max\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourrange</b>	Multi-year hourly range $o(0001, x) = \text{range}\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \text{range}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhoursum</b>	Multi-year hourly sum $o(0001, x) = \text{sum}\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \text{sum}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourmean</b>	Multi-year hourly mean $o(0001, x) = \text{mean}\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \text{mean}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhouravg</b>	Multi-year hourly average $o(0001, x) = \text{avg}\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \text{avg}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourstd</b>	Multi-year hourly standard deviation Normalize by n. $o(0001, x) = \text{std}\{i(t, x), \text{day}(i(t)) = 0001\}$ $\vdots$ $o(8784, x) = \text{std}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourstd1</b>	Multi-year hourly standard deviation (n-1) Normalize by (n-1).

	$o(0001, x) = \mathbf{std1}\{i(t, x), \text{day}(i(t)) = 0001\}$
	$\vdots$
	$o(8784, x) = \mathbf{std1}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourvar</b>	Multi-year hourly variance Normalize by n.
	$o(0001, x) = \mathbf{var}\{i(t, x), \text{day}(i(t)) = 0001\}$
	$\vdots$
	$o(8784, x) = \mathbf{var}\{i(t, x), \text{day}(i(t)) = 8784\}$
<b>yhourvar1</b>	Multi-year hourly variance (n-1) Normalize by (n-1).
	$o(0001, x) = \mathbf{var1}\{i(t, x), \text{day}(i(t)) = 0001\}$
	$\vdots$
	$o(8784, x) = \mathbf{var1}\{i(t, x), \text{day}(i(t)) = 8784\}$

## 2.8.29. YDAYSTAT - Multi-year daily statistical values

### Synopsis

`<operator> infile outfile`

### Description

This module computes statistical values of each day of year. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of each day of year in infile is written to outfile. The date information in an output field is the date of the last contributing input field.

### Operators

<b>ydaymin</b>	Multi-year daily minimum $o(001, x) = \mathbf{min}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{min}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydaymax</b>	Multi-year daily maximum $o(001, x) = \mathbf{max}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{max}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydayrange</b>	Multi-year daily range $o(001, x) = \mathbf{range}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{range}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydaysum</b>	Multi-year daily sum $o(001, x) = \mathbf{sum}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{sum}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydaymean</b>	Multi-year daily mean $o(001, x) = \mathbf{mean}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{mean}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydayavg</b>	Multi-year daily average $o(001, x) = \mathbf{avg}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{avg}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydaystd</b>	Multi-year daily standard deviation Normalize by n. $o(001, x) = \mathbf{std}\{i(t, x), \text{day}(i(t)) = 001\}$ $\vdots$ $o(366, x) = \mathbf{std}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydaystd1</b>	Multi-year daily standard deviation (n-1) Normalize by (n-1).

	$o(001, x) = \mathbf{std1}\{i(t, x), \text{day}(i(t)) = 001\}$
	$\vdots$
	$o(366, x) = \mathbf{std1}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydayvar</b>	Multi-year daily variance Normalize by n.
	$o(001, x) = \mathbf{var}\{i(t, x), \text{day}(i(t)) = 001\}$
	$\vdots$
	$o(366, x) = \mathbf{var}\{i(t, x), \text{day}(i(t)) = 366\}$
<b>ydayvar1</b>	Multi-year daily variance (n-1) Normalize by (n-1).
	$o(001, x) = \mathbf{var1}\{i(t, x), \text{day}(i(t)) = 001\}$
	$\vdots$
	$o(366, x) = \mathbf{var1}\{i(t, x), \text{day}(i(t)) = 366\}$

### Example

To compute the daily mean over all input years use:

```
cdo ydaymean infile outfile
```

## 2.8.30. YDAYPCTL - Multi-year daily percentile values

### Synopsis

```
ydaypctl,p infile1 infile2 infile3 outfile
```

### Description

This operator writes a certain percentile of each day of year in `infile1` to `outfile`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by setting the environment variable `CDO_PCTL_NBINS` to a different value. The files `infile2` and `infile3` should be the result of corresponding `ydaymin` and `ydaymax` operations, respectively. The date information in an output field is the date of the last contributing input field.

$$\begin{aligned} o(001, x) &= \text{pth percentile}\{i(t, x), \text{day}(i(t)) = 001\} \\ &\vdots \\ o(366, x) &= \text{pth percentile}\{i(t, x), \text{day}(i(t)) = 366\} \end{aligned}$$

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

### Example

To compute the daily 90th percentile over all input years use:

```
cdo ydaymin infile minfile
cdo ydaymax infile maxfile
cdo ydaypctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo ydaypctl,90 infile -ydaymin infile -ydaymax infile outfile
```

## 2.8.31. YMONSTAT - Multi-year monthly statistical values

### Synopsis

`<operator> infile outfile`

### Description

This module computes statistical values of each month of year. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of each month of year in `infile` is written to `outfile`. The date information in an output field is the date of the last contributing input field.

### Operators

<b>ymonmin</b>	Multi-year monthly minimum $o(01, x) = \mathbf{min}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{min}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonmax</b>	Multi-year monthly maximum $o(01, x) = \mathbf{max}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{max}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonrange</b>	Multi-year monthly range $o(01, x) = \mathbf{range}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{range}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonsum</b>	Multi-year monthly sum $o(01, x) = \mathbf{sum}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{sum}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonmean</b>	Multi-year monthly mean $o(01, x) = \mathbf{mean}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{mean}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonavg</b>	Multi-year monthly average $o(01, x) = \mathbf{avg}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{avg}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonstd</b>	Multi-year monthly standard deviation Normalize by n. $o(01, x) = \mathbf{std}\{i(t, x), \text{month}(i(t)) = 01\}$ $\vdots$ $o(12, x) = \mathbf{std}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonstd1</b>	Multi-year monthly standard deviation (n-1) Normalize by (n-1).

	$o(01, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 01\}$
	$\vdots$
	$o(12, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonvar</b>	Multi-year monthly variance Normalize by n.
	$o(01, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 01\}$
	$\vdots$
	$o(12, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 12\}$
<b>ymonvar1</b>	Multi-year monthly variance (n-1) Normalize by (n-1).
	$o(01, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 01\}$
	$\vdots$
	$o(12, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 12\}$

## Example

To compute the monthly mean over all input years use:

```
cdo ymonmean infile outfile
```

## 2.8.32. YMONPCTL - Multi-year monthly percentile values

### Synopsis

```
ymonpctl,p infile1 infile2 infile3 outfile
```

### Description

This operator writes a certain percentile of each month of year in `infile1` to `outfile`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by setting the environment variable `CDO_PCTL_NBINS` to a different value. The files `infile2` and `infile3` should be the result of corresponding [ymonmin](#) and [ymonmax](#) operations, respectively. The date information in an output field is the date of the last contributing input field.

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

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

### Example

To compute the monthly 90th percentile over all input years use:

```
cdo ymonmin infile minfile
cdo ymonmax infile maxfile
cdo ymonpctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo ymonpctl,90 infile -ymonmin infile -ymonmax infile outfile
```



## 2.8.33. YSEASSTAT - Multi-year seasonal statistical values

### Synopsis

`<operator> infile outfile`

### Description

This module computes statistical values of each season. Depending on the chosen operator the minimum, maximum, range, sum, average, variance or standard deviation of each season in `infile` is written to `outfile`. The date information in an output field is the date of the last contributing input field.

### Operators

<b>yseasmin</b>	Multi-year seasonal minimum $o(1, x) = \min\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \min\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \min\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \min\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasmax</b>	Multi-year seasonal maximum $o(1, x) = \max\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \max\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \max\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \max\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasrange</b>	Multi-year seasonal range $o(1, x) = \text{range}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \text{range}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \text{range}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \text{range}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseassum</b>	Multi-year seasonal sum $o(1, x) = \text{sum}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \text{sum}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \text{sum}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \text{sum}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasmean</b>	Multi-year seasonal mean $o(1, x) = \text{mean}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \text{mean}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \text{mean}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \text{mean}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasavg</b>	Multi-year seasonal average $o(1, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \text{avg}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasstd</b>	Multi-year seasonal standard deviation $o(1, x) = \text{std}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \text{std}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \text{std}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \text{std}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$

<b>yseasstd1</b>	Multi-year seasonal standard deviation (n-1) $o(1, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \mathbf{std1}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasvar</b>	Multi-year seasonal variance $o(1, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \mathbf{var}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$
<b>yseasvar1</b>	Multi-year seasonal variance (n-1) $o(1, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\}$ $o(2, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\}$ $o(3, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\}$ $o(4, x) = \mathbf{var1}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\}$

## Example

To compute the seasonal mean over all input years use:

```
cdo yseasmean infile outfile
```

## 2.8.34. YSEASPCTL - Multi-year seasonal percentile values

### Synopsis

```
yseaspctl,p infile1 infile2 infile3 outfile
```

### Description

This operator writes a certain percentile of each season in `infile1` to `outfile`. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by setting the environment variable `CDO_PCTL_NBINS` to a different value. The files `infile2` and `infile3` should be the result of corresponding [yseasmin](#) and [yseasmax](#) operations, respectively. The date information in an output field is the date of the last contributing input field.

$$\begin{aligned} o(1, x) &= \text{pth percentile}\{i(t, x), \text{month}(i(t)) = 12, 01, 02\} \\ o(2, x) &= \text{pth percentile}\{i(t, x), \text{month}(i(t)) = 03, 04, 05\} \\ o(3, x) &= \text{pth percentile}\{i(t, x), \text{month}(i(t)) = 06, 07, 08\} \\ o(4, x) &= \text{pth percentile}\{i(t, x), \text{month}(i(t)) = 09, 10, 11\} \end{aligned}$$

### Parameter

`p`      FLOAT      Percentile number in 0, ..., 100

### Environment

`CDO_PCTL_NBINS`      Sets the number of histogram bins. The default number is 101.

### Example

To compute the seasonal 90th percentile over all input years use:

```
cdo yseasmin infile minfile
cdo yseasmax infile maxfile
cdo yseaspctl,90 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo yseaspctl,90 infile -yseasmin infile -yseasmax infile outfile
```

## 2.8.35. YDRUNSTAT - Multi-year daily running statistical values

### Synopsis

`<operator>,nts infile outfile`

### Description

This module writes running statistical values for each day of year in `infile` to `outfile`. Depending on the chosen operator, the minimum, maximum, sum, average, variance or standard deviation of all timesteps in running windows of which the medium timestep corresponds to a certain day of year is computed. The date information in an output field is the date of the timestep in the middle of the last contributing running window. Note that the operator have to be applied to a continuous time series of daily measurements in order to yield physically meaningful results. Also note that the output time series begins  $(nts-1)/2$  timesteps after the first timestep of the input time series and ends  $(nts-1)/2$  timesteps before the last one. For input data which are complete but not continuous, such as time series of daily measurements for the same month or season within different years, the operator yields physically meaningful results only if the input time series does include the  $(nts-1)/2$  days before and after each period of interest.

### Operators

<b>ydrunmin</b>	Multi-year daily running minimum $o(001, x) = \min\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \min\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunmax</b>	Multi-year daily running maximum $o(001, x) = \max\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \max\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunsum</b>	Multi-year daily running sum $o(001, x) = \sum\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \sum\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunmean</b>	Multi-year daily running mean $o(001, x) = \text{mean}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \text{mean}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunavg</b>	Multi-year daily running average $o(001, x) = \text{avg}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \text{avg}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunstd</b>	Multi-year daily running standard deviation Normalize by n. $o(001, x) = \text{std}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \text{std}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$

<b>ydrunstd1</b>	Multi-year daily running standard deviation (n-1) Normalize by (n-1). $o(001, x) = \mathbf{std1}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \mathbf{std1}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunvar</b>	Multi-year daily running variance Normalize by n. $o(001, x) = \mathbf{var}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \mathbf{var}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$
<b>ydrunvar1</b>	Multi-year daily running variance (n-1) Normalize by (n-1). $o(001, x) = \mathbf{var1}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\}$ $\vdots$ $o(366, x) = \mathbf{var1}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\}$

## Parameter

*nts*    INTEGER    Number of timesteps

## Example

Assume the input data provide a continuous time series of daily measurements. To compute the running multi-year daily mean over all input timesteps for a running window of five days use:

```
cdo ydrunmean,5 infile outfile
```

Note that except for the standard deviation the results of the operators in this module are equivalent to a composition of corresponding operators from the [YDAYSTAT](#) and [RUNSTAT](#) modules. For instance, the above command yields the same result as:

```
cdo ydaymean -runmean,5 infile outfile
```

## 2.8.36. YDRUNPCTL - Multi-year daily running percentile values

### Synopsis

```
ydrunpctl,p,nts infile1 infile2 infile3 outfile
```

### Description

This operator writes running percentile values for each day of year in `infile1` to `outfile`. A certain percentile is computed for all timesteps in running windows of which the medium timestep corresponds to a certain day of year. The algorithm uses histograms with minimum and maximum bounds given in `infile2` and `infile3`, respectively. The default number of histogram bins is 101. The default can be overridden by setting the environment variable `CDO_PCTL_NBINS` to a different value. The files `infile2` and `infile3` should be the result of corresponding `ydrunmin` and `ydrunmax` operations, respectively. The date information in an output field is the date of the timestep in the middle of the last contributing running window. Note that the operator have to be applied to a continuous time series of daily measurements in order to yield physically meaningful results. Also note that the output time series begins  $(nts-1)/2$  timesteps after the first timestep of the input time series and ends  $(nts-1)/2$  timesteps before the last. For input data which are complete but not continuous, such as time series of daily measurements for the same month or season within different years, the operator only yields physically meaningful results if the input time series does include the  $(nts-1)/2$  days before and after each period of interest.

$$\begin{aligned} o(001, x) &= \text{pth percentile}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 001\} \\ &\vdots \\ o(366, x) &= \text{pth percentile}\{i(t, x), i(t+1, x), \dots, i(t+nts-1, x); \text{day}[(i(t+(nts-1)/2)] = 366\} \end{aligned}$$

### Parameter

<code>p</code>	FLOAT	Percentile number in 0, ..., 100
<code>nts</code>	INTEGER	Number of timesteps

### Environment

<code>CDO_PCTL_NBINS</code>	Sets the number of histogram bins. The default number is 101.
-----------------------------	---

### Example

Assume the input data provide a continuous time series of daily measurements. To compute the running multi-year daily 90th percentile over all input timesteps for a running window of five days use:

```
cdo ydrunmin,5 infile minfile
cdo ydrunmax,5 infile maxfile
cdo ydrunpctl,90,5 infile minfile maxfile outfile
```

Or shorter using operator piping:

```
cdo ydrunpctl,90,5 infile -ydrunmin infile -ydrunmax infile outfile
```

## 2.9. Correlation and co.

This sections contains modules for correlation and co. in grid space and over time.  
In this section the abbreviations as in the following table are used:

Covariance <b>covar</b>	$n^{-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$
<b>covar</b> weighted by $\{w_i, i = 1, \dots, n\}$	$\left( \sum_{j=1}^n w_j \right)^{-1} \sum_{i=1}^n w_i \left( x_i - \left( \sum_{j=1}^n w_j \right)^{-1} \sum_{j=1}^n w_j x_j \right) \left( y_i - \left( \sum_{j=1}^n w_j \right)^{-1} \sum_{j=1}^n w_j y_j \right)$

Here is a short overview of all operators in this section:

<b>fldcor</b>	Correlation in grid space
<b>timcor</b>	Correlation over time
<b>fldcovar</b>	Covariance in grid space
<b>timcovar</b>	Covariance over time

### 2.9.1. FLDCOR - Correlation in grid space

#### Synopsis

```
fldcor infile1 infile2 outfile
```

#### Description

The correlation coefficient is a quantity that gives the quality of a least squares fitting to the original data. This operator correlates all gridpoints of two fields for each timestep. With

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

it is

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

where  $w(x)$  are the area weights obtained by the input streams. For every timestep  $t$  only those field elements  $x$  belong to the sample, which have  $i_1(t, x) \neq \text{missval}$  and  $i_2(t, x) \neq \text{missval}$ .

### 2.9.2. TIMCOR - Correlation over time

#### Synopsis

```
timcor infile1 infile2 outfile
```

#### Description

The correlation coefficient is a quantity that gives the quality of a least squares fitting to the original data. This operator correlates each gridpoint of two fields over all timesteps. With

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

it is

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

For every gridpoint  $x$  only those timesteps  $t$  belong to the sample, which have  $i_1(t, x) \neq \text{missval}$  and  $i_2(t, x) \neq \text{missval}$ .



### 2.9.3. FLDCOVAR - Covariance in grid space

#### Synopsis

```
fldcovar infile1 infile2 outfile
```

#### Description

This operator calculates the covariance of two fields over all gridpoints for each timestep. With

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

it is

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

where  $w(x)$  are the area weights obtained by the input streams. For every timestep  $t$  only those field elements  $x$  belong to the sample, which have  $i_1(t, x) \neq \text{missval}$  and  $i_2(t, x) \neq \text{missval}$ .

### 2.9.4. TIMCOVAR - Covariance over time

#### Synopsis

```
timcovar infile1 infile2 outfile
```

#### Description

This operator calculates the covariance of two fields at each gridpoint over all timesteps. With

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

it is

$$o(1, x) = n^{-1} \sum_{t \in S(x)} \left( i_1(t, x) - \overline{i_1(t, x)} \right) \left( i_2(t, x) - \overline{i_2(t, x)} \right)$$

For every gridpoint  $x$  only those timesteps  $t$  belong to the sample, which have  $i_1(t, x) \neq \text{missval}$  and  $i_2(t, x) \neq \text{missval}$ .

## 2.10. Regression

This sections contains modules for linear regression of time series.

Here is a short overview of all operators in this section:

<b>regres</b>	Regression
<b>detrend</b>	Detrend
<b>trend</b>	Trend
<b>subtrend</b>	Subtract trend

### 2.10.1. REGRES - Regression

#### Synopsis

```
regres infile outfile
```

#### Description

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

$$o(1, x) = \frac{\sum_{t \in S(x)} \left( i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)^2}$$

### 2.10.2. DETREND - Detrend time series

#### Synopsis

```
detrend infile outfile
```

#### Description

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

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

and

$$b(x) = \frac{\sum_{t \in S(x)} \left( i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)^2}$$

it is

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

#### Note

This operator has to keep the fields of all timesteps concurrently in the memory. If not enough memory is available use the operators [trend](#) and [subtrend](#).

#### Example

To detrend the data in `infile` and to store the detrended data in `outfile` use:

```
cdo detrend infile outfile
```

### 2.10.3. TREND - Trend of time series

#### Synopsis

```
trend infile outfile1 outfile2
```

#### Description

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

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

and

$$o_2(1, x) = \frac{\sum_{t \in S(x)} \left( i(t, x) - \frac{1}{\#S(x)} \sum_{t' \in S(x)} i(t', x) \right) \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)}{\sum_{t \in S(x)} \left( t - \frac{1}{\#S(x)} \sum_{t' \in S(x)} t' \right)^2}$$

Thus the estimation for  $a$  is stored in `outfile1` and that for  $b$  is stored in `outfile2`. To subtract the trend from the data see operator [subtrend](#).

### 2.10.4. SUBTREND - Subtract a trend

#### Synopsis

```
subtrend infile1 infile2 infile3 outfile
```

#### Description

This operator is for subtracting a trend computed by the operator [trend](#). It is

$$o(t, x) = i_1(t, x) - (i_2(1, x) + i_3(1, x) \cdot t)$$

where  $t$  is the timesteps.

#### Example

The typical call for detrending the data in `infile` and storing the detrended data in `outfile` is:

```
cdo trend infile afile bfile
cdo subtrend infile afile bfile outfile
```

The result is identical to a call of the operator [detrend](#):

```
cdo detrend infile outfile
```

## 2.11. EOFs

This section contains modules to compute Empirical Orthogonal Functions and - once they are computed - their principal coefficients.

An introduction to the theory of principal component analysis as applied here can be found in:

Principal Component Analysis [Peisendorfer]

Details about calculation in the time- and spatial spaces are found in:

Statistical Analysis in Climate Research [vonStorch]

EOFs are defined as the eigen values of the scatter matrix (covariance matrix) of the data. For the sake of simplicity, samples are regarded as **time series of anomalies**

$$(z(t)), t \in \{1, \dots, n\}$$

of (column-) vectors  $z(t)$  with  $p$  entries (where  $p$  is the gridsize). Thus, using the fact, that  $z_j(t)$  are anomalies, i.e.

$$\langle z_j \rangle = n^{-1} \sum_{i=1}^n z_j(i) = 0 \quad \forall 1 \leq j \leq p$$

the scatter matrix  $\mathbf{S}$  can be written as

$$\mathbf{S} = \sum_{t=1}^n \left[ \sqrt{\mathbf{W}} z(t) \right] \left[ \sqrt{\mathbf{W}} z(t) \right]^T$$

where  $\mathbf{W}$  is the diagonal matrix containing the area weight of cell  $p_0$  in  $z$  at  $\mathbf{W}(x, x)$ .

The matrix  $\mathbf{S}$  has a set of orthonormal eigenvectors  $e_j, j = 1, \dots, p$ , which are called *empirical orthogonal functions (EOFs) of the sample  $z$* . (Please note, that  $e_j$  is the eigenvector of  $\mathbf{S}$  and not the weighted eigen-vector which would be  $\mathbf{W}e_j$ .) Let the corresponding eigenvalues be denoted  $\lambda_j$ . The vectors  $e_j$  are spatial patterns which explain a certain amount of variance of the time series  $z(t)$  that is related linearly to  $\lambda_j$ . Thus, the spatial pattern defined by the first eigenvector (the one with the largest eigenvalue) is the pattern which explains a maximum possible amount of variance of the sample  $z(t)$ . The orthonormality of eigenvectors reads as

$$\sum_{x=1}^p \left[ \sqrt{\mathbf{W}(x, x)} e_j(x) \right] \left[ \sqrt{\mathbf{W}(x, x)} e_k(x) \right] = \sum_{x=1}^p \mathbf{W}(x, x) e_j(x) e_k(x) = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } j = k \end{cases}$$

If all EOFs  $e_j$  with  $\lambda_j \neq 0$  are calculated, the data can be reconstructed from

$$z(t, x) = \sum_{j=1}^p \mathbf{W}(x, x) a_j(t) e_j(x)$$

where  $a_j$  are called the *principal components* or *principal coefficients* or *EOF coefficients* of  $z$ . These coefficients - as readily seen from above - are calculated as the projection of an EOF  $e_j$  onto a time step of the data sample  $z(t_0)$  as

$$a_j(t_0) = \sum_{x=1}^p \left[ \sqrt{\mathbf{W}(x, x)} e_j(x) \right] \left[ \sqrt{\mathbf{W}(x, x)} z(t_0, x) \right] = \left[ \sqrt{\mathbf{W}} z(t_0) \right]^T \left[ \sqrt{\mathbf{W}} e_j \right].$$

Here is a short overview of all operators in this section:

<b>eof</b>	Calculate EOFs in spatial or time space
<b>eoftime</b>	Calculate EOFs in time space
<b>eofspatial</b>	Calculate EOFs in spatial space
<b>eof3d</b>	Calculate 3-Dimensional EOFs in time space
<b>eofcoeff</b>	Calculate principal coefficients of EOFs

## 2.11.1. EOFs - Empirical Orthogonal Functions

### Synopsis

```
<operator> ,neof infile outfile1 outfile2
```

### Description

This module calculates empirical orthogonal functions of the data in `infile` as the eigen values of the scatter matrix (covariance matrix)  $S$  of the data sample  $z(t)$ . A more detailed description can be found above.

**Please note, that the input data are assumed to be anomalies.**

If operator `eof` is chosen, the EOFs are computed in either time or spatial space, whichever is the fastest. If the user already knows, which computation is faster, the module can be forced to perform a computation in time- or gridspace by using the operators `eoftime` or `eofspatial`, respectively. This can enhance performance, especially for very long time series, where the number of timesteps is larger than the number of grid-points. Data in `infile` are assumed to be anomalies. If they are not, the behavior of this module is **not well defined**. After execution `outfile1` will contain all eigen-values and `outfile2` the eigenvectors  $e_j$ . All EOFs and eigen-values are computed. However, only the first `neof` EOFs are written to `outfile2`. Nonetheless, `outfile1` contains all eigen-values.

Missing values are not fully supported. Support is only checked for non-changing masks of missing values in time. Although there still will be results, they are not trustworthy, and a warning will occur. In the latter case we suggest to replace missing values by 0 in `infile`.

### Operators

<code>eof</code>	Calculate EOFs in spatial or time space
<code>eoftime</code>	Calculate EOFs in time space
<code>eofspatial</code>	Calculate EOFs in spatial space
<code>eof3d</code>	Calculate 3-Dimensional EOFs in time space

### Parameter

<code>neof</code>	INTEGER	Number of eigen functions
-------------------	---------	---------------------------

### Environment

<code>CDO_SVD_MODE</code>	Is used to choose the algorithm for eigenvalue calculation. Options are 'jacobi' for a one-sided parallel jacobi-algorithm (only executed in parallel if -P flag is set) and 'danielson_lanczos' for a non-parallel d/l algorithm. The default setting is 'jacobi'.
<code>CDO_WEIGHT_MODE</code>	It is used to set the weight mode. The default is 'on'. Set it to 'off' for a non weighted version.
<code>MAX_JACOBI_ITER</code>	Is the maximum integer number of annihilation sweeps that is executed if the jacobi-algorithm is used to compute the eigen values. The default value is 12.
<code>FNORM_PRECISION</code>	Is the Frobenius norm of the matrix consisting of an annihilation pair of eigenvectors that is used to determine if the eigenvectors have reached a sufficient level of convergence. If all annihilation-pairs of vectors have a norm below this value, the computation is considered to have converged properly. Otherwise, a warning will occur. The default value 1e-12.

**Example**

To calculate the first 40 EOFs of a data-set containing anomalies use:

```
cdo eof,40 infile outfile1 outfile2
```

If the dataset does not contain anomalies, process them first, and use:

```
cdo sub infile1 -timmean infile1 anom_file  
cdo eof,40 anom_file outfile1 outfile2
```

## 2.11.2. EOFCOEFF - Principal coefficients of EOFs

### Synopsis

```
eofcoeff infile1 infile2 obase
```

### Description

This module calculates the time series of the principal coefficients for given EOF (empirical orthogonal functions) and data. Time steps in `infile1` are assumed to be the EOFs, time steps in `infile2` are assumed to be the time series. Note, that this operator calculates a non weighted dot product of the fields in `infile1` and `infile2`. For consistency set the environment variable `CDO_WEIGHT_MODE=off` when using `eof` or `eof3d`. Given a set of EOFs  $e_j$  and a time series of data  $z(t)$  with  $p$  entries for each timestep from which  $e_j$  have been calculated, this operator calculates the time series of the projections of data onto each EOF

$$o_j(t) = \sum_{x=1}^p z(t, x) e_j(x)$$

There will be a separate file  $o_j$  for the principal coefficients of each EOF.

As the EOFs  $e_j$  are uncorrelated, so are their principal coefficients, i.e.

$$\sum_{t=1}^n o_j(t) o_k(t) = \begin{cases} 0 & \text{if } j \neq k \\ \lambda_j & \text{if } j = k \end{cases} \quad \text{with } \sum_{t=1}^n o_j(t) = 0 \forall j \in \{1, \dots, p\}.$$

There will be a separate file containing a time series of principal coefficients with time information from `infile2` for each EOF in `infile1`. Output files will be numbered as `<obase><neof><suffix>` where `neof+1` is the number of the EOF (timestep) in `infile1` and `suffix` is the filename extension derived from the file format.

### Environment

`CDO_FILE_SUFFIX` Set the default file suffix. This suffix will be added to the output file names instead of the filename extension derived from the file format. Set this variable to `NULL` to disable the adding of a file suffix.

### Example

To calculate principal coefficients of the first 40 EOFs of `anom_file`, and write them to files beginning with `obase`, use:

```
export CDO_WEIGHT_MODE=off
cdo eof,40 anom_file eval_file eof_file
cdo eofcoeff eof_file anom_file obase
```

The principal coefficients of the first EOF will be in the file `obase000000.nc` (and so forth for higher EOFs,  $n$ th EOF will be in `obase<n-1>`).

If the dataset `infile` does not contain anomalies, process them first, and use:

```
export CDO_WEIGHT_MODE=off
cdo sub infile -timmean infile anom_file
cdo eof,40 anom_file eval_file eof_file
cdo eofcoeff eof_file anom_file obase
```



## 2.12. Interpolation

This section contains modules to interpolate datasets. There are several operators to interpolate horizontal fields to a new grid. Some of those operators can handle only 2D fields on a regular rectangular grid. Vertical interpolation of 3D variables is possible from hybrid model levels to height or pressure levels. Interpolation in time is possible between time steps and years.

Here is a short overview of all operators in this section:

<b>remapbil</b>	Bilinear interpolation
<b>genbil</b>	Generate bilinear interpolation weights
<b>remapbic</b>	Bicubic interpolation
<b>genbic</b>	Generate bicubic interpolation weights
<b>remapnn</b>	Nearest neighbor remapping
<b>gennn</b>	Generate nearest neighbor remap weights
<b>remapdis</b>	Distance-weighted average remapping
<b>gendis</b>	Generate distance-weighted average remap weights
<b>remapycon</b>	First order conservative remapping
<b>genycon</b>	Generate 1st order conservative remap weights
<b>remapcon</b>	First order conservative remapping
<b>gencon</b>	Generate 1st order conservative remap weights
<b>remapcon2</b>	Second order conservative remapping
<b>gencon2</b>	Generate 2nd order conservative remap weights
<b>remaplaf</b>	Largest area fraction remapping
<b>genlaf</b>	Generate largest area fraction remap weights
<b>remap</b>	Grid remapping
<b>remapeta</b>	Remap vertical hybrid level
<b>ml2pl</b>	Model to pressure level interpolation
<b>ml2hl</b>	Model to height level interpolation
<b>ap2pl</b>	Air pressure to pressure level interpolation
<b>ap2hl</b>	Air pressure to height level interpolation
<b>intlevel</b>	Linear level interpolation
<b>intlevel3d</b>	Linear level interpolation onto a 3d vertical coordinate
<b>intlevelx3d</b>	like intlevel3d but with extrapolation
<b>inttime</b>	Interpolation between timesteps
<b>intntime</b>	Interpolation between timesteps
<b>intyear</b>	Interpolation between two years

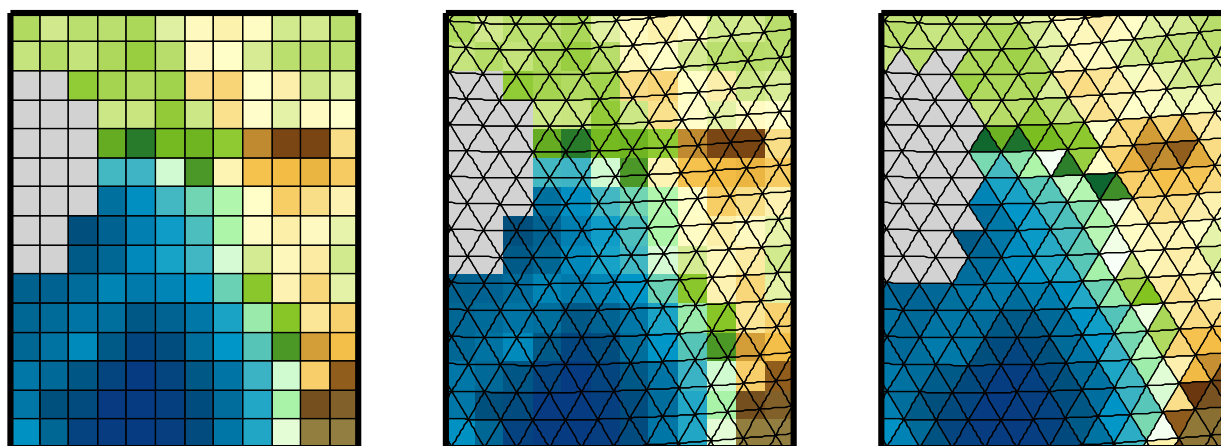
### 2.12.1. REMAPBIL - Bilinear interpolation

#### Synopsis

```
<operator>,grid infile outfile
```

#### Description

This module contains operators for a bilinear remapping of fields between grids in spherical coordinates. The interpolation is based on an adapted SCRIP library version. For a detailed description of the interpolation method see [SCRIP]. This interpolation method only works on quadrilateral curvilinear source grids. Below is a schematic illustration of the bilinear remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

#### Operators

<b>remapbil</b>	Bilinear interpolation Performs a bilinear interpolation on all input fields.
<b>genbil</b>	Generate bilinear interpolation weights Generates bilinear interpolation weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

#### Parameter

*grid*    STRING    Target grid description file or name

#### Environment

REMAP\_EXTRAPOLATE    This variable is used to switch the extrapolation feature 'on' or 'off'. By default the extrapolation is enabled for circular grids.

#### Example

Say *infile* contains fields on a quadrilateral curvilinear grid. To remap all fields bilinear to a Gaussian N32 grid, type:

```
cdo remapbil,n32 infile outfile
```

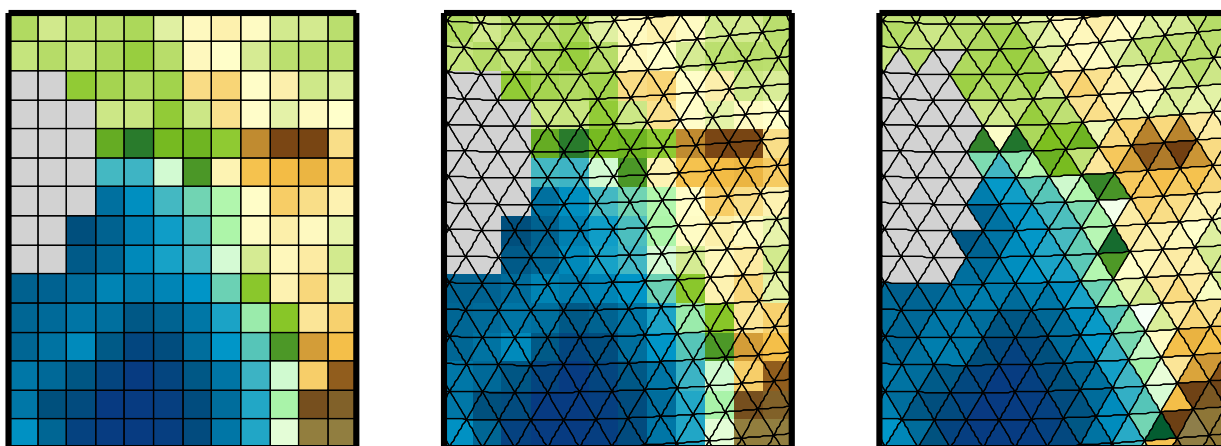
## 2.12.2. REMAPBIC - Bicubic interpolation

### Synopsis

```
<operator> ,grid infile outfile
```

### Description

This module contains operators for a bicubic remapping of fields between grids in spherical coordinates. The interpolation is based on an adapted SCRIP library version. For a detailed description of the interpolation method see [SCRIP]. This interpolation method only works on quadrilateral curvilinear source grids. Below is a schematic illustration of the bicubic remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remapbic</b>	Bicubic interpolation Performs a bicubic interpolation on all input fields.
<b>genbic</b>	Generate bicubic interpolation weights Generates bicubic interpolation weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

*grid*     STRING     Target grid description file or name

### Environment

REMAP\_EXTRAPOLATE     This variable is used to switch the extrapolation feature 'on' or 'off'. By default the extrapolation is enabled for circular grids.

### Example

Say *infile* contains fields on a quadrilateral curvilinear grid. To remap all fields bicubic to a Gaussian N32 grid, type:

```
cdo remapbic,n32 infile outfile
```

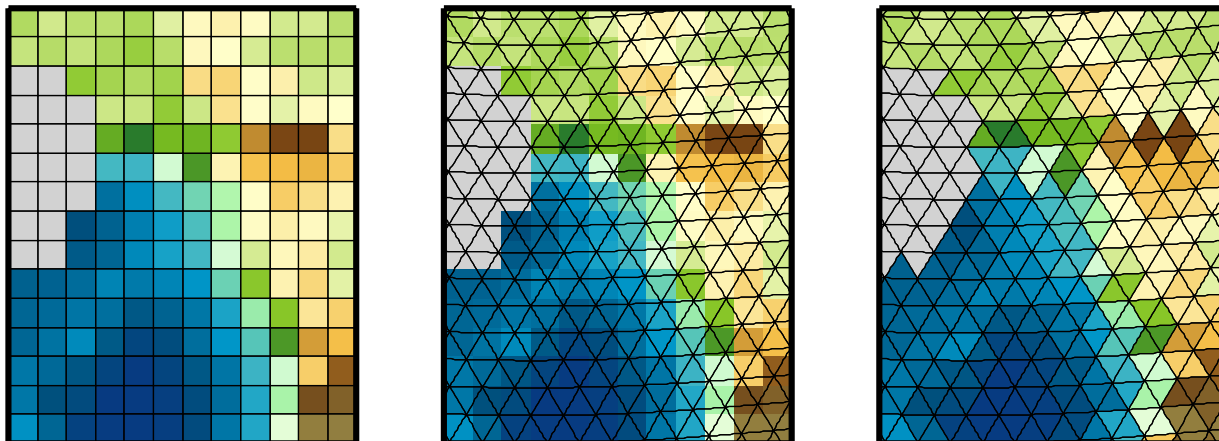
### 2.12.3. REMAPNN - Nearest neighbor remapping

#### Synopsis

```
<operator>,grid infile outfile
```

#### Description

This module contains operators for a nearest neighbor remapping of fields between grids in spherical coordinates. Below is a schematic illustration of the nearest neighbor remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

#### Operators

<b>remapnn</b>	Nearest neighbor remapping Performs a nearest neighbor remapping on all input fields.
<b>gennn</b>	Generate nearest neighbor remap weights Generates nearest neighbor remapping weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

#### Parameter

*grid*     STRING     Target grid description file or name

#### Environment

REMAP_EXTRAPOLATE	This variable is used to switch the extrapolation feature 'on' or 'off'. By default the extrapolation is enabled for this remapping method.
CDO_GRIDSEARCH_RADIUS	Grid search radius in degree, default 180 degree.

## 2.12.4. REMAPDIS - Distance-weighted average remapping

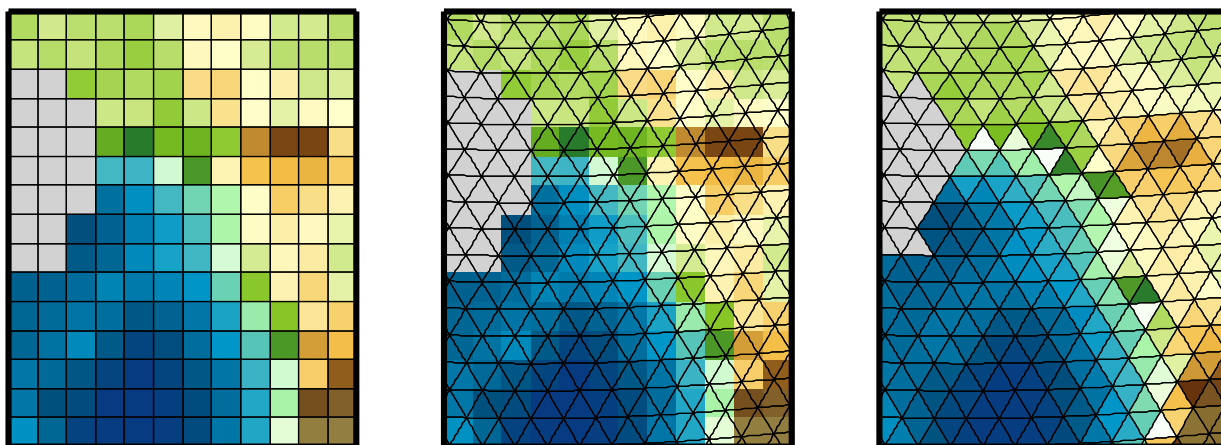
### Synopsis

```
remapdis,grid[,neighbors] infile outfile
```

```
gendis,grid infile outfile
```

### Description

This module contains operators for a distance-weighted average remapping of the four nearest neighbor values of fields between grids in spherical coordinates. The interpolation is based on an adapted SCRIP library version. For a detailed description of the interpolation method see [SCRIP]. Below is a schematic illustration of the distance-weighted average remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remapdis</b>	Distance-weighted average remapping Performs a distance-weighted average remapping of the nearest neighbors value on all input fields. The default number of nearest neighbors is 4.
<b>gendis</b>	Generate distance-weighted average remap weights Generates distance-weighted average remapping weights of the four nearest neighbor values for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
<i>neighbors</i>	INTEGER	Number of nearest neighbors

### Environment

REMAP_EXTRAPOLATE	This variable is used to switch the extrapolation feature 'on' or 'off'. By default the extrapolation is enabled for this remapping method.
CDO_GRIDSEARCH_RADIUS	Grid search radius in degree, default 180 degree.

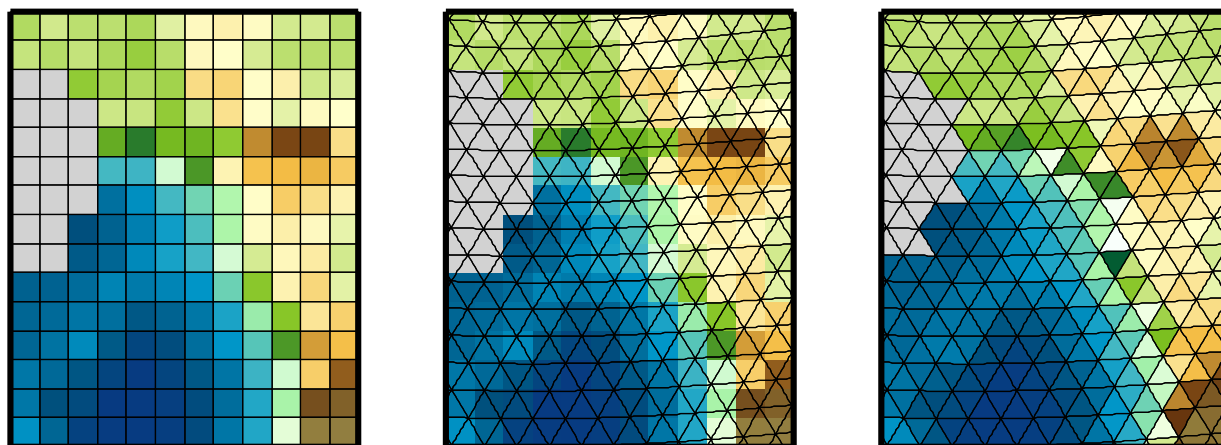
## 2.12.5. REMAPYCON - First order conservative remapping

### Synopsis

```
<operator> ,grid infile outfile
```

### Description

This module contains operators for a first order conservative remapping of fields between grids in spherical coordinates. The operators in this module uses code from the YAC software package to compute the conservative remapping weights. For a detailed description of the interpolation method see [YAC]. The interpolation method is completely general and can be used for any grid on a sphere. The search algorithm for the conservative remapping requires that no grid cell occurs more than once. Below is a schematic illustration of the 1st order conservative remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remapycon</b>	First order conservative remapping Performs a first order conservative remapping on all input fields.
<b>genycon</b>	Generate 1st order conservative remap weights Generates first order conservative remapping weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
-------------	--------	--------------------------------------

### Environment

CDO_REMAP_NORM	This variable is used to choose the normalization of the conservative interpolation. By default CDO_REMAP_NORM is set to 'fracarea'. 'fracarea' uses the sum of the non-masked source cell intersected areas to normalize each target cell field value. This results in a reasonable flux value but the flux is not locally conserved. The option 'destarea' uses the total target cell area to normalize each target cell field value. Local flux conservation is ensured, but unreasonable flux values may result.
----------------	--

REMAP_AREA_MIN	This variable is used to set the minimum destination area fraction. The default of this variable is 0.0.
----------------	--

## Example

Say `infile` contains fields on a quadrilateral curvilinear grid. To remap all fields conservative to a Gaussian N32 grid, type:

```
cdo remapycon,n32 infile outfile
```

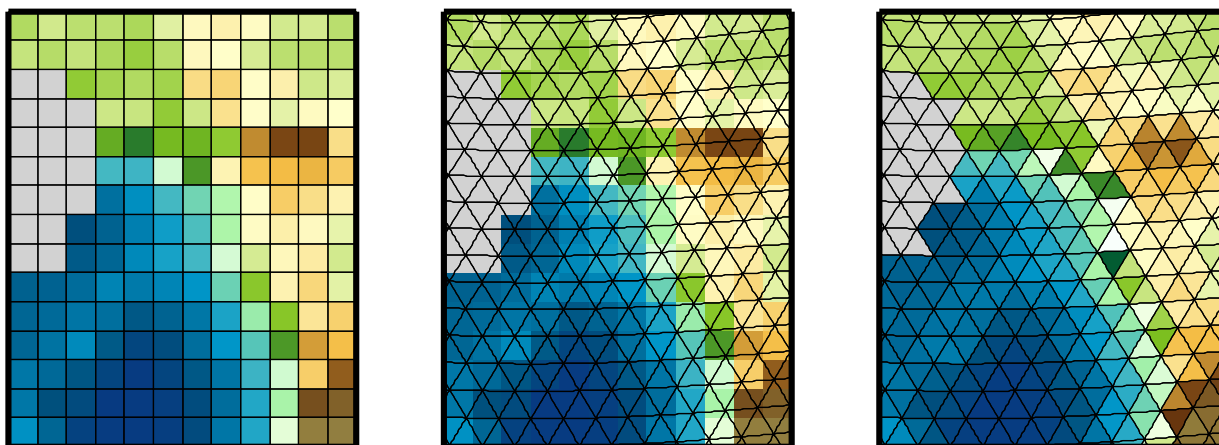
## 2.12.6. REMAPCON - First order conservative remapping

### Synopsis

```
<operator>,>grid infile outfile
```

### Description

This module contains operators for a first order conservative remapping of fields between grids in spherical coordinates. The interpolation is based on an adapted SCRIP library version. For a detailed description of the interpolation method see [SCRIP]. The interpolation method is completely general and can be used for any grid on a sphere. The search algorithm for the conservative remapping requires that no grid cell occurs more than once. Below is a schematic illustration of the 1st order conservative remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remapcon</b>	First order conservative remapping Performs a first order conservative remapping on all input fields.
<b>gencon</b>	Generate 1st order conservative remap weights Generates first order conservative remapping weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
-------------	--------	--------------------------------------

### Environment

CDO_REMAP_NORM	This variable is used to choose the normalization of the conservative interpolation. By default CDO_REMAP_NORM is set to 'fracarea'. 'fracarea' uses the sum of the non-masked source cell intersected areas to normalize each target cell field value. This results in a reasonable flux value but the flux is not locally conserved. The option 'destarea' uses the total target cell area to normalize each target cell field value. Local flux conservation is ensured, but unreasonable flux values may result.
----------------	--



REMAP_AREA_MIN	This variable is used to set the minimum destination area fraction. The default of this variable is 0.0.
----------------	--

## Note

The SCRIP conservative remapping method doesn't work correctly for some grid combinations. Please use [remapycon](#) or [genycon](#) in case of problems.

## Example

Say `infile` contains fields on a quadrilateral curvilinear grid. To remap all fields conservative to a Gaussian N32 grid, type:

```
cdo remapcon,n32 infile outfile
```

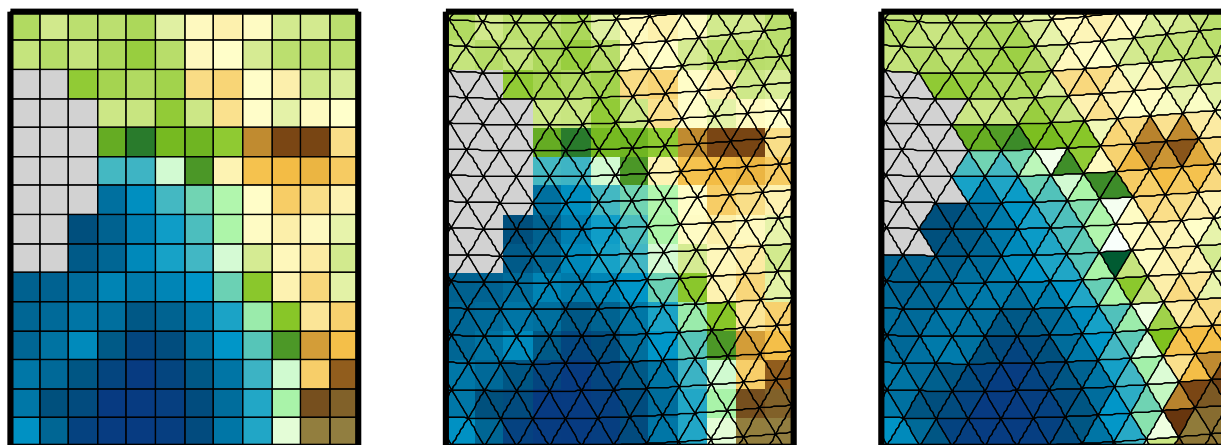
## 2.12.7. REMAPCON2 - Second order conservative remapping

### Synopsis

```
<operator> ,grid infile outfile
```

### Description

This module contains operators for a second order conservative remapping of fields between grids in spherical coordinates. The interpolation is based on an adapted SCRIP library version. For a detailed description of the interpolation method see [SCRIP]. The interpolation method is completely general and can be used for any grid on a sphere. The search algorithm for the conservative remapping requires that no grid cell occurs more than once. Below is a schematic illustration of the 2nd order conservative remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remapcon2</b>	Second order conservative remapping Performs a second order conservative remapping on all input fields.
<b>gencon2</b>	Generate 2nd order conservative remap weights Generates second order conservative remapping weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
-------------	--------	--------------------------------------

### Environment

CDO_REMAP_NORM	This variable is used to choose the normalization of the conservative interpolation. By default CDO_REMAP_NORM is set to 'fracarea'. 'fracarea' uses the sum of the non-masked source cell intersected areas to normalize each target cell field value. This results in a reasonable flux value but the flux is not locally conserved. The option 'destarea' uses the total target cell area to normalize each target cell field value. Local flux conservation is ensured, but unreasonable flux values may result.
----------------	--

REMAP_AREA_MIN	This variable is used to set the minimum destination area fraction. The default of this variable is 0.0.
----------------	--

## Note

The SCRIP conservative remapping method doesn't work correctly for some grid combinations.

## Example

Say `infile` contains fields on a quadrilateral curvilinear grid. To remap all fields conservative (2nd order) to a Gaussian N32 grid, type:

```
cdo remapcon2,n32 infile outfile
```

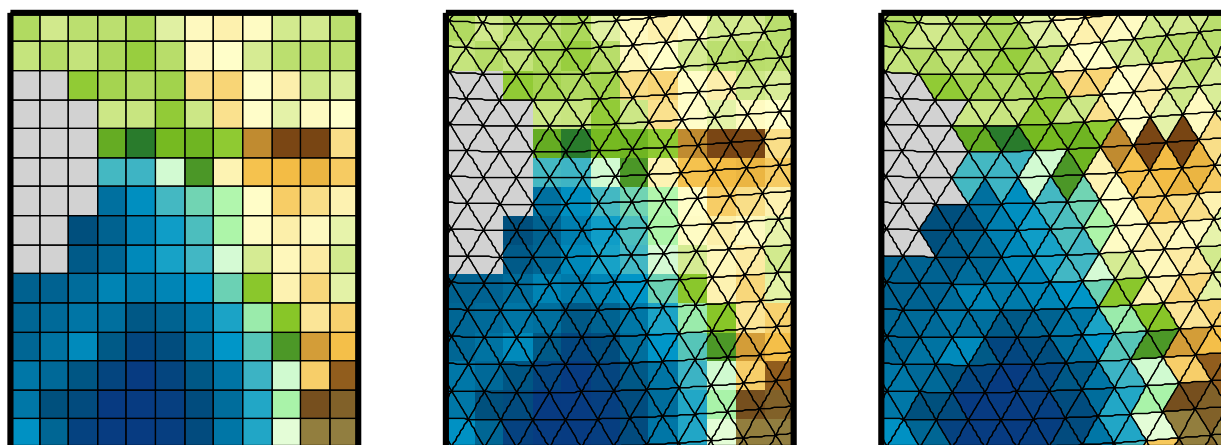
## 2.12.8. REMAPLAF - Largest area fraction remapping

### Synopsis

```
<operator>,grid infile outfile
```

### Description

This module contains operators for a largest area fraction remapping of fields between grids in spherical coordinates. The operators in this module uses code from the YAC software package to compute the largest area fraction. For a detailed description of the interpolation method see [YAC]. The interpolation method is completely general and can be used for any grid on a sphere. The search algorithm for this remapping method requires that no grid cell occurs more than once. Below is a schematic illustration of the largest area fraction conservative remapping:



The figure on the left side shows the input data on a regular lon/lat source grid and on the right side the remapped result on an unstructured triangular target grid. The figure in the middle shows the input data with the target grid. Grid cells with missing value are grey colored.

### Operators

<b>remaplaf</b>	Largest area fraction remapping Performs a largest area fraction remapping on all input fields.
<b>genlaf</b>	Generate largest area fraction remap weights Generates largest area fraction remapping weights for the first input field and writes the result to a file. The format of this file is NetCDF following the SCRIP convention. Use the operator <a href="#">remap</a> to apply this remapping weights to a data file with the same source grid.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
-------------	--------	--------------------------------------

### Environment

REMAP_AREA_MIN	This variable is used to set the minimum destination area fraction. The default of this variable is 0.0.
----------------	--

## 2.12.9. REMAP - Grid remapping

### Synopsis

```
remap,grid,weights infile outfile
```

### Description

Interpolation between different horizontal grids can be a very time-consuming process. Especially if the data are on an unstructured and/or a large grid. In this case the interpolation process can be split into two parts. Firstly the generation of the interpolation weights, which is the most time-consuming part. These interpolation weights can be reused for every remapping process with the operator [remap](#). This operator remaps all input fields to a new horizontal grid. The remap type and the interpolation weights of one input grid are read from a NetCDF file. More weights are computed if the input fields are on different grids. The NetCDF file with the weights should follow the [\[SCRIP\]](#) convention. Normally these weights come from a previous call to one of the genXXX operators (e.g. [genbil](#)) or were created by the original SCRIP package.

### Parameter

<i>grid</i>	STRING	Target grid description file or name
<i>weights</i>	STRING	Interpolation weights (SCRIP NetCDF file)

### Environment

CDO_REMAP_NORM	This variable is used to choose the normalization of the conservative interpolation. By default CDO_REMAP_NORM is set to 'fracarea'. 'fracarea' uses the sum of the non-masked source cell intersected areas to normalize each target cell field value. This results in a reasonable flux value but the flux is not locally conserved. The option 'destarea' uses the total target cell area to normalize each target cell field value. Local flux conservation is ensured, but unreasonable flux values may result.
REMAP_EXTRAPOLATE	This variable is used to switch the extrapolation feature 'on' or 'off'. By default the extrapolation is enabled for remapdis, remapnn and for circular grids.
REMAP_AREA_MIN	This variable is used to set the minimum destination area fraction. The default of this variable is 0.0.
CDO_GRIDSEARCH_RADIUS	Grid search radius in degree, default 180 degree.

### Example

Say infile contains fields on a quadrilateral curvilinear grid. To remap all fields bilinear to a Gaussian N32 grid use:

```
cdo genbil,n32 infile remapweights.nc
cdo remap,n32,remapweights.nc infile outfile
```

The result will be the same as:

```
cdo remapbil,n32 infile outfile
```

## 2.12.10. REMAPETA - Remap vertical hybrid level

### Synopsis

```
remapeta,vct[,oro] infile outfile
```

### Description

This operator interpolates between different vertical hybrid levels. This include the preparation of consistent data for the free atmosphere. The procedure for the vertical interpolation is based on the HIRLAM scheme and was adapted from [\[INTERA\]](#). The vertical interpolation is based on the vertical integration of the hydrostatic equation with few adjustments. The basic tasks are the following one:

- at first integration of hydrostatic equation
- extrapolation of surface pressure
- Planetary Boundary-Layer (PBL) proutfile interpolation
- interpolation in free atmosphere
- merging of both proutfiles
- final surface pressure correction

The vertical interpolation corrects the surface pressure. This is simply a cut-off or an addition of air mass. This mass correction should not influence the geostrophic velocity field in the middle troposphere. Therefore the total mass above a given reference level is conserved. As reference level the geopotential height of the 400 hPa level is used. Near the surface the correction can affect the vertical structure of the PBL. Therefore the interpolation is done using the potential temperature. But in the free atmosphere above a certain  $n$  ( $n=0.8$  defining the top of the PBL) the interpolation is done linearly. After the interpolation both proutfiles are merged. With the resulting temperature/pressure correction the hydrostatic equation is integrated again and adjusted to the reference level finding the final surface pressure correction. A more detailed description of the interpolation can be found in [\[INTERA\]](#). This operator requires all variables on the same horizontal grid.

### Parameter

<code>vct</code>	STRING	File name of an ASCII dataset with the vertical coordinate table
<code>oro</code>	STRING	File name with the orography (surf. geopotential) of the target dataset (optional)

### Environment

REMAPETA_PTOP	Sets the minimum pressure level for condensation. Above this level the humidity is set to the constant 1.E-6. The default value is 0 Pa.
---------------	--

### Note

The code numbers or the variable names of the required parameter have to follow the [\[ECHAM\]](#) convention. Presently, the vertical coordinate definition of a NetCDF file has also to follow the ECHAM convention. This means:

- the dimension of the full level coordinate and the corresponding variable is called `mlev`,
- the dimension of the half level coordinate and the corresponding variable is called `ilev` (`ilev` must have one element more than `mlev`)
- the hybrid vertical coefficient `a` is given in units of Pa and called `hyai` (`hyam` for level midpoints)
- the hybrid vertical coefficient `b` is given in units of 1 and called `hybi` (`hybm` for level midpoints)

- the `mlev` variable has a `borders` attribute containing the character string 'ilev'

Use the [sinfo](#) command to test if your vertical coordinate system is recognized as hybrid system.

In case [remapeta](#) complains about not finding any data on hybrid model levels you may wish to use the [setzaxis](#) command to generate a `zaxis` description which conforms to the ECHAM convention. See section "1.4 Z-axis description" for an example how to define a hybrid Z-axis.

## Example

To remap between different hybrid model level data use:

```
cdo remapeta,vct infile outfile
```

Here is an example `vct` file with 19 hybrid model level:

0	0.0000000000000000	0.0000000000000000
1	2000.0000000000000000	0.0000000000000000
2	4000.0000000000000000	0.0000000000000000
3	6046.1093750000000000	0.00033899326808751
4	8267.9296875000000000	0.00335718691349030
5	10609.5117187500000000	0.01307003945112228
6	12851.1015625000000000	0.03407714888453484
7	14698.5000000000000000	0.07064980268478394
8	15861.1289062500000000	0.12591671943664551
9	16116.2382812500000000	0.20119541883468628
10	15356.9218750000000000	0.29551959037780762
11	13621.4609375000000000	0.40540921688079834
12	11101.5585937500000000	0.52493220567703247
13	8127.1445312500000000	0.64610791206359863
14	5125.1406250000000000	0.75969839096069336
15	2549.9689941406250000	0.85643762350082397
16	783.1950683593750000	0.92874687910079956
17	0.0000000000000000	0.97298520803451538
18	0.0000000000000000	0.99228149652481079
19	0.0000000000000000	1.0000000000000000

### 2.12.11. VERTINTML - Vertical interpolation

#### Synopsis

```
ml2pl,plevels infile outfile
```

```
ml2hl,hlevels infile outfile
```

#### Description

Interpolate 3D variables on hybrid sigma pressure level to pressure or height levels. The input file should contain the log. surface pressure or the surface pressure. To extrapolate the temperature, the surface geopotential is also needed. The pressure, temperature, and surface geopotential are identified by their GRIB1 code number or NetCDF CF standard name. Supported parameter tables are: WMO standard table number 2 and ECMWF local table number 128. Use the alias **ml2plx/ml2hlx** or the environment variable EXTRAPOLATE to extrapolate missing values. This operator requires all variables on the same horizontal grid.

#### Operators

**ml2pl** Model to pressure level interpolation  
Interpolates 3D variables on hybrid sigma pressure level to pressure level.

**ml2hl** Model to height level interpolation  
Interpolates 3D variables on hybrid sigma pressure level to height level. The procedure is the same as for the operator [ml2pl](#) except for the pressure levels being calculated from the heights by:  $plevel = 101325 * \exp(hlevel / -7000)$

#### Parameter

<i>plevels</i>	FLOAT	Pressure levels in pascal
<i>hlevels</i>	FLOAT	Height levels in meter (max level: 65535 m)

#### Environment

EXTRAPOLATE If set to 1 extrapolate missing values.

#### Example

To interpolate hybrid model level data to pressure levels of 925, 850, 500 and 200 hPa use:

```
cdo ml2pl,92500,85000,50000,20000 infile outfile
```

### 2.12.12. VERTINTAP - Vertical interpolation

#### Synopsis

```
ap2pl,plevels infile outfile
```

```
ap2hl,hlevels infile outfile
```



## Description

Interpolate 3D variables on hybrid sigma height coordinates to pressure or height levels. The input file must contain the 3D air pressure. The air pressure is identified by the NetCDF CF standard name `air_pressure`. Use the alias **ap2plx/ap2hlx** or the environment variable `EXTRAPOLATE` to extrapolate missing values. This operator requires all variables on the same horizontal grid.

## Operators

- ap2pl**     Air pressure to pressure level interpolation  
Interpolates 3D variables on hybrid sigma height coordinates to pressure level.
- ap2hl**     Air pressure to height level interpolation  
Interpolates 3D variables on hybrid sigma height coordinates to height level. The procedure is the same as for the operator [ap2pl](#) except for the pressure levels being calculated from the heights by:  $p_{level} = 101325 * \exp(h_{level} / -7000)$

## Parameter

<i>plevels</i>	FLOAT	Pressure levels in pascal
<i>hlevels</i>	FLOAT	Height levels in meter (max level: 65535 m)

## Environment

`EXTRAPOLATE`     If set to 1 extrapolate missing values.

## Note

This is a specific implementation for NetCDF files from the ICON model, it may not work with data from other sources.

## Example

To interpolate 3D variables on hybrid sigma height level to pressure levels of 925, 850, 500 and 200 hPa use:

```
cdo ap2pl,92500,85000,50000,20000 infile outfile
```

## 2.12.13. INTLEVEL - Linear level interpolation

### Synopsis

```
intlevel,levels infile outfile
```

### Description

This operator performs a linear vertical interpolation of non hybrid 3D variables.

### Parameter

<i>levels</i>	FLOAT	Target levels
---------------	-------	---------------

**Example**

To interpolate 3D variables on height levels to a new set of height levels use:

```
cdo intlevel,10,50,100,500,1000 infile outfile
```

## 2.12.14. INTLEVEL3D - Linear level interpolation from/to 3d vertical coordinates

### Synopsis

```
<operator>,icoordinate infile1 infile2 outfile
```

### Description

This operator performs a linear vertical interpolation of 3D variables fields with given 3D vertical coordinates.

### Operators

**intlevel3d**      Linear level interpolation onto a 3d vertical coordinate

**intlevelx3d**    like intlevel3d but with extrapolation

### Parameter

<i>icoordinate</i>	STRING	filename for vertical source coordinates variable
<i>infile2</i>	STRING	target vertical coordinate field (intlevel3d only)

### Example

To interpolate 3D variables from one set of 3d height levels into another one where

- *icoordinate* contains a single 3d variable, which represents the input 3d vertical coordinate
- *infile1* contains the source data, which the vertical coordinate from *icoordinate* belongs to
- *infile2* only contains the target 3d height levels

```
cdo intlevel3d,icoordinate infile1 infile2 outfile
```

## 2.12.15. INTTIME - Time interpolation

### Synopsis

```
inttime,date,time[,inc] infile outfile
```

```
intntime,n infile outfile
```

### Description

This module performs linear interpolation between timesteps.

### Operators

<b>inttime</b>	Interpolation between timesteps This operator creates a new dataset by linear interpolation between timesteps. The user has to define the start date/time with an optional increment.
<b>intntime</b>	Interpolation between timesteps This operator performs linear interpolation between timesteps. The user has to define the number of timesteps from one timestep to the next.

### Parameter

<i>date</i>	STRING	Start date (format YYYY-MM-DD)
<i>time</i>	STRING	Start time (format hh:mm:ss)
<i>inc</i>	STRING	Optional increment (seconds, minutes, hours, days, months, years) [default: 0hour]
<i>n</i>	INTEGER	Number of timesteps from one timestep to the next

### Example

Assumed a 6 hourly dataset starts at 1987-01-01 12:00:00. To interpolate this time series to a one hourly dataset use:

```
cdo inttime,1987-01-01,12:00:00,1hour infile outfile
```

## 2.12.16. INTYEAR - Year interpolation

### Synopsis

```
intyear,years infile1 infile2 obase
```

### Description

This operator performs linear interpolation between two years, timestep by timestep. The input files need to have the same structure with the same variables. The output files will be named `<obase><yyyy><suffix>` where `yyyy` will be the year and `suffix` is the filename extension derived from the file format.

### Parameter

<code>years</code>	INTEGER	Comma separated list of years
--------------------	---------	-------------------------------

### Environment

<code>CDO_FILE_SUFFIX</code>	Set the default file suffix. This suffix will be added to the output file names instead of the filename extension derived from the file format. Set this variable to NULL to disable the adding of a file suffix.
------------------------------	---

### Note

This operator needs to open all output files simultaneously. The maximum number of open files depends on the operating system!

### Example

Assume there are two monthly mean datasets over a year. The first dataset has 12 timesteps for the year 1985 and the second one for the year 1990. To interpolate the years between 1985 and 1990 month by month use:

```
cdo intyear,1986,1987,1988,1989 infile1 infile2 year
```

Example result of `'dir year*'` for NetCDF datasets:

```
year1986.nc year1987.nc year1988.nc year1989.nc
```

## 2.13. Transformation

This section contains modules to perform spectral transformations.

Here is a short overview of all operators in this section:

<b>sp2gp</b>	Spectral to gridpoint
<b>sp2gpl</b>	Spectral to gridpoint (linear)
<b>gp2sp</b>	Gridpoint to spectral
<b>gp2spl</b>	Gridpoint to spectral (linear)
<b>sp2sp</b>	Spectral to spectral
<b>dv2uv</b>	Divergence and vorticity to U and V wind
<b>dv2uvl</b>	Divergence and vorticity to U and V wind (linear)
<b>uv2dv</b>	U and V wind to divergence and vorticity
<b>uv2dvl</b>	U and V wind to divergence and vorticity (linear)
<b>dv2ps</b>	D and V to velocity potential and stream function

### 2.13.1. SPECTRAL - Spectral transformation

#### Synopsis

```
<operator> infile outfile
```

```
sp2sp, trunc infile outfile
```

#### Description

This module transforms fields on Gaussian grids to spectral coefficients and vice versa.

#### Operators

- sp2gp**      Spectral to gridpoint  
Convert all fields with spectral coefficients to a regular Gaussian grid. The number of latitudes of the resulting Gaussian grid is calculated from the triangular truncation by:  
$$nlat = NINT((trunc * \sqrt{3} + 1.) / 2.)$$
- sp2gpl**      Spectral to gridpoint (linear)  
Convert all fields with spectral coefficients to a regular Gaussian grid. The number of latitudes of the resulting Gaussian grid is calculated from the triangular truncation by:  
$$nlat = NINT((trunc * \sqrt{2} + 1.) / 2.)$$
  
Use this operator to convert ERA40 data e.g. from TL159 to N80.
- gp2sp**      Gridpoint to spectral  
Convert all Gaussian gridpoint fields to spectral coefficients. The triangular truncation of the resulting spherical harmonics is calculated from the number of latitudes by:  
$$trunc = (nlat * 2 - 1) / \sqrt{3}$$
- gp2spl**      Gridpoint to spectral (linear)  
Convert all Gaussian gridpoint fields to spectral coefficients. The triangular truncation of the resulting spherical harmonics is calculated from the number of latitudes by:  
$$trunc = (nlat * 2 - 1) / \sqrt{2}$$
  
Use this operator to convert ERA40 data e.g. from N80 to TL159 instead of T106.
- sp2sp**      Spectral to spectral  
Change the triangular truncation of all spectral fields. The operator performs downward conversion by cutting the resolution. Upward conversions are achieved by filling in zeros.

#### Parameter

*trunc*      INTEGER      New spectral resolution

#### Example

To transform spectral coefficients from T106 to N80 Gaussian grid use:

```
cdo sp2gp infile outfile
```

To transform spectral coefficients from TL159 to N80 Gaussian grid use:

```
cdo sp2gpl infile outfile
```

## 2.13.2. WIND - Wind transformation

### Synopsis

```
<operator> infile outfile
```

### Description

This module converts relative divergence and vorticity to U and V wind and vice versa. Divergence and vorticity are spherical harmonic coefficients in spectral space and U and V are on a regular Gaussian grid. The Gaussian latitudes need to be ordered from north to south.

### Operators

- dv2uv**      Divergence and vorticity to U and V wind  
Calculate U and V wind on a Gaussian grid from spherical harmonic coefficients of relative divergence and vorticity. The divergence and vorticity need to have the names sd and svo or code numbers 155 and 138. The number of latitudes of the resulting Gaussian grid is calculated from the triangular truncation by:  
$$nlat = NINT((trunc * \boxed{3} + 1.) / 2.)$$
- dv2uyl**      Divergence and vorticity to U and V wind (linear)  
Calculate U and V wind on a Gaussian grid from spherical harmonic coefficients of relative divergence and vorticity. The divergence and vorticity need to have the names sd and svo or code numbers 155 and 138. The number of latitudes of the resulting Gaussian grid is calculated from the triangular truncation by:  
$$nlat = NINT((trunc * \boxed{2} + 1.) / 2.)$$
- uv2dv**      U and V wind to divergence and vorticity  
Calculate spherical harmonic coefficients of relative divergence and vorticity from U and V wind. The U and V wind need to be on a Gaussian grid and need to have the names u and v or the code numbers 131 and 132. The triangular truncation of the resulting spherical harmonics is calculated from the number of latitudes by:  
$$trunc = (nlat * 2 - 1) / \boxed{3}$$
- uv2dyl**      U and V wind to divergence and vorticity (linear)  
Calculate spherical harmonic coefficients of relative divergence and vorticity from U and V wind. The U and V wind need to be on a Gaussian grid and need to have the names u and v or the code numbers 131 and 132. The triangular truncation of the resulting spherical harmonics is calculated from the number of latitudes by:  
$$trunc = (nlat * 2 - 1) / \boxed{2}$$
- dv2ps**      D and V to velocity potential and stream function  
Calculate spherical harmonic coefficients of velocity potential and stream function from spherical harmonic coefficients of relative divergence and vorticity. The divergence and vorticity need to have the names sd and svo or code numbers 155 and 138.

### Example

Assume a dataset has at least spherical harmonic coefficients of divergence and vorticity. To transform the spectral divergence and vorticity to U and V wind on a Gaussian grid use:

```
cdo dv2uv infile outfile
```



## 2.14. Import/Export

This section contains modules to import and export data files which can not read or write directly with CDO.

Here is a short overview of all operators in this section:

<b>import_binary</b>	Import binary data sets
<b>import_cmsaf</b>	Import CM-SAF HDF5 files
<b>import_amsr</b>	Import AMSR binary files
<b>input</b>	ASCII input
<b>inputsrv</b>	SERVICE ASCII input
<b>inputtext</b>	EXTRA ASCII input
<b>output</b>	ASCII output
<b>outputf</b>	Formatted output
<b>outputint</b>	Integer output
<b>outputsrv</b>	SERVICE ASCII output
<b>outputtext</b>	EXTRA ASCII output
<b>outputtab</b>	Table output
<b>gmtxyz</b>	GMT xyz format
<b>gmtcells</b>	GMT multiple segment format

### 2.14.1. IMPORTBINARY - Import binary data sets

#### Synopsis

```
import_binary infile outfile
```

#### Description

This operator imports gridded binary data sets via a GrADS data descriptor file. The GrADS data descriptor file contains a complete description of the binary data as well as instructions on where to find the data and how to read it. The descriptor file is an ASCII file that can be created easily with a text editor. The general contents of a gridded data descriptor file are as follows:

- Filename for the binary data
- Missing or undefined data value
- Mapping between grid coordinates and world coordinates
- Description of variables in the binary data set

A detailed description of the components of a GrADS data descriptor file can be found in [\[GrADS\]](#). Here is a list of the supported components: BYTESWAPPED, CHSUB, DSET, ENDVARS, FILE-HEADER, HEADERBYTES, OPTIONS, TDEF, TITLE, TRAILERBYTES, UNDEF, VARS, XDEF, XYHEADER, YDEF, ZDEF

#### Note

Only 32-bit IEEE floats are supported for standard binary files!

#### Example

To convert a binary data file to NetCDF use:

```
cdo -f nc import_binary infile.ctl outfile.nc
```

Here is an example of a GrADS data descriptor file:

```
DSET ^infile.bin
OPTIONS sequential
UNDEF -9e+33
XDEF 360 LINEAR -179.5 1
YDEF 180 LINEAR -89.5 1
ZDEF 1 LINEAR 1 1
TDEF 1 LINEAR 00:00Z15jun1989 12hr
VARS 1
  param 1 99 description of the variable
ENDVARS
```

The binary data file infile.bin contains one parameter on a global 1 degree lon/lat grid written with FORTRAN record length headers (sequential).

## 2.14.2. IMPORTCMSAF - Import CM-SAF HDF5 files

### Synopsis

```
import_cmsaf infile outfile
```

### Description

This operator imports gridded CM-SAF (Satellite Application Facility on Climate Monitoring) HDF5 files. CM-SAF exploits data from polar-orbiting and geostationary satellites in order to provide climate monitoring products of the following parameters:

**Cloud parameters:** cloud fraction (CFC), cloud type (CTY), cloud phase (CPH), cloud top height, pressure and temperature (CTH,CTP,CTT), cloud optical thickness (COT), cloud water path (CWP).

**Surface radiation components:** Surface albedo (SAL); surface incoming (SIS) and net (SNS) shortwave radiation; surface downward (SDL) and outgoing (SOL) longwave radiation, surface net longwave radiation (SNL) and surface radiation budget (SRB).

**Top-of-atmosphere radiation components:** Incoming (TIS) and reflected (TRS) solar radiative flux at top-of-atmosphere. Emitted thermal radiative flux at top-of-atmosphere (TET).

**Water vapour:** Vertically integrated water vapour (HTW), layered vertically integrated water vapour and layer mean temperature and relative humidity for 5 layers (HLW), temperature and mixing ratio at 6 pressure levels.

Daily and monthly mean products can be ordered via the CM-SAF web page ([www.cmsaf.eu](http://www.cmsaf.eu)). Products with higher spatial and temporal resolution, i.e. instantaneous swath-based products, are available on request ([contact.cmsaf@dwd.de](mailto:contact.cmsaf@dwd.de)). All products are distributed free-of-charge. More information on the data is available on the CM-SAF homepage ([www.cmsaf.eu](http://www.cmsaf.eu)).

Daily and monthly mean products are provided in equal-area projections. **CDO** reads the projection parameters from the metadata in the HDF5-headers in order to allow spatial operations like remapping. For spatial operations with instantaneous products on original satellite projection, additional files with arrays of latitudes and longitudes are needed. These can be obtained from CM-SAF together with the data.

### Note

To use this operator, it is necessary to build **CDO** with HDF5 support (version 1.6 or higher). The PROJ.4 library (version 4.6 or higher) is needed for full support of the remapping functionality.

### Example

A typical sequence of commands with this operator could look like this:

```
cdo -f nc remapbil,r360x180 -import_cmsaf cmsaf_product.hdf output.nc
```

(bilinear remapping to a predefined global grid with 1 deg resolution and conversion to NetCDF).

If you work with CM-SAF data on original satellite project, an additional file with information on geolocation is required, to perform such spatial operations:

```
cdo -f nc remapbil,r720x360 -setgrid,cmsaf_latlon.h5 -import_cmsaf cmsaf.hdf out.nc
```

Some CM-SAF data are stored as scaled integer values. For some operations, it could be desirable (or necessary) to increase the accuracy of the converted products:

```
cdo -b f32 -f nc fldmean -sellonlatbox,0,10,0,10 -remapbil,r720x360 \  
-import_cmsaf cmsaf_product.hdf output.nc
```

### 2.14.3. IMPORTAMSR - Import AMSR binary files

#### Synopsis

```
import_amsr infile outfile
```

#### Description

This operator imports gridded binary AMSR (Advanced Microwave Scanning Radiometer) data. The binary data files are available from the AMSR ftp site (<ftp://ftp.ssmi.com/amsre>). Each file consists of twelve (daily) or five (averaged) 0.25 x 0.25 degree grid (1440,720) byte maps. For daily files, six daytime maps in the following order, Time (UTC), Sea Surface Temperature (SST), 10 meter Surface Wind Speed (WSPD), Atmospheric Water Vapor (VAPOR), Cloud Liquid Water (CLOUD), and Rain Rate (RAIN), are followed by six nighttime maps in the same order. Time-Averaged files contain just the geophysical layers in the same order [SST, WSPD, VAPOR, CLOUD, RAIN]. More information to the data is available on the AMSR homepage <http://www.remss.com/amsr>.

#### Example

To convert monthly binary AMSR files to NetCDF use:

```
cdo -f nc amsre_yyyymm5 amsre_yyyymm5.nc
```

## 2.14.4. INPUT - Formatted input

### Synopsis

**input**,*grid*[,*zaxis*] outfile

**inputsrv** outfile

**inputtext** outfile

### Description

This module reads time series of one 2D variable from standard input. All input fields need to have the same horizontal grid. The format of the input depends on the chosen operator.

### Operators

<b>input</b>	ASCII input Reads fields with ASCII numbers from standard input and stores them in outfile. The numbers read are exactly that ones which are written out by the <a href="#">output</a> operator.
<b>inputsrv</b>	SERVICE ASCII input Reads fields with ASCII numbers from standard input and stores them in outfile. Each field should have a header of 8 integers (SERVICE likely). The numbers that are read are exactly that ones which are written out by the <a href="#">outputsrv</a> operator.
<b>inputtext</b>	EXTRA ASCII input Read fields with ASCII numbers from standard input and stores them in outfile. Each field should have header of 4 integers (EXTRA likely). The numbers read are exactly that ones which are written out by the <a href="#">outputtext</a> operator.

### Parameter

<i>grid</i>	STRING	Grid description file or name
<i>zaxis</i>	STRING	Z-axis description file

### Example

Assume an ASCII dataset contains a field on a global regular grid with 32 longitudes and 16 latitudes (512 elements). To create a GRIB1 dataset from the ASCII dataset use:

```
cdo -f grb input,r32x16 outfile.grb < my_ascii_data
```

## 2.14.5. OUTPUT - Formatted output

### Synopsis

```

output   infiles
outputf,format[,nelem] infiles
outputint infiles
outputsrv infiles
outputtext infiles

```

### Description

This module prints all values of all input datasets to standard output. All input fields need to have the same horizontal grid. All input files need to have the same structure with the same variables. The format of the output depends on the chosen operator.

### Operators

<b>output</b>	ASCII output Prints all values to standard output. Each row has 6 elements with the C-style format "%13.6g".
<b>outputf</b>	Formatted output Prints all values to standard output. The format and number of elements for each row have to be specified by the parameters <i>format</i> and <i>nelem</i> . The default for <i>nelem</i> is 1.
<b>outputint</b>	Integer output Prints all values rounded to the nearest integer to standard output.
<b>outputsrv</b>	SERVICE ASCII output Prints all values to standard output. Each field with a header of 8 integers (SERVICE likely).
<b>outputtext</b>	EXTRA ASCII output Prints all values to standard output. Each field with a header of 4 integers (EXTRA likely).

### Parameter

<i>format</i>	STRING	C-style format for one element (e.g. %13.6g)
<i>nelem</i>	INTEGER	Number of elements for each row (default: <i>nelem</i> = 1)

### Example

To print all field elements of a dataset formatted with "%8.4g" and 8 values per line use:

```
cdo outputf,%8.4g,8 infile
```

Example result of a dataset with one field on 64 grid points:

261.7	262	257.8	252.5	248.8	247.7	246.3	246.1
250.6	252.6	253.9	254.8	252	246.6	249.7	257.9
273.4	266.2	259.8	261.6	257.2	253.4	251	263.7
267.5	267.4	272.2	266.7	259.6	255.2	272.9	277.1
275.3	275.5	276.4	278.4	282	269.6	278.7	279.5
282.3	284.5	280.3	280.3	280	281.5	284.7	283.6
292.9	290.5	293.9	292.6	292.7	292.8	294.1	293.6
293.8	292.6	291.2	292.6	293.2	292.8	291	291.2

## 2.14.6. OUTPUTTAB - Table output

### Synopsis

```
outputtab,params infiles outfile
```

### Description

This operator prints a table of all input datasets to standard output. `infiles` is an arbitrary number of input files. All input files need to have the same structure with the same variables on different timesteps. All input fields need to have the same horizontal grid.

The contents of the table depends on the chosen paramters. The format of each table parameter is `keyname[:len]`. `len` is the optional length of a table entry. Here is a list of all valid keynames:

Keyname	Type	Description
value	FLOAT	Value of the variable [len:8]
name	STRING	Name of the variable [len:8]
param	STRING	Parameter ID (GRIB1: code[.tabnum]; GRIB2: num[.cat[.dis]]) [len:11]
code	INTEGER	Code number [len:4]
lon	FLOAT	Longitude coordinate [len:6]
lat	FLOAT	Latitude coordinate [len:6]
lev	FLOAT	Vertical level [len:6]
xind	INTEGER	Grid x index [len:4]
yind	INTEGER	Grid y index [len:4]
timestep	INTEGER	Timestep number [len:6]
date	STRING	Date (format YYYY-MM-DD) [len:10]
time	STRING	Time (format hh:mm:ss) [len:8]
year	INTEGER	Year [len:5]
month	INTEGER	Month [len:2]
day	INTEGER	Day [len:2]
nohead	INTEGER	Disable output of header line

### Parameter

`params`      STRING      Comma separated list of keynames, one for each column of the table

### Example

To print a table with name, date, lon, lat and value information use:

```
cdo outputtab,name,date,lon,lat,value infile
```

Here is an example output of a time series with the yearly mean temperatur at lon=10/lat=53.5:

#	name	date	lon	lat	value
	tsurf	1991-12-31	10	53.5	8.83903
	tsurf	1992-12-31	10	53.5	8.17439
	tsurf	1993-12-31	10	53.5	7.90489
	tsurf	1994-12-31	10	53.5	10.0216
	tsurf	1995-12-31	10	53.5	9.07798

## 2.14.7. OUTPUTGMT - GMT output

### Synopsis

```
<operator> infile
```

### Description

This module prints the first field of the input dataset to standard output. The output can be used to generate 2D Lon/Lat plots with [\[GMT\]](#). The format of the output depends on the chosen operator.

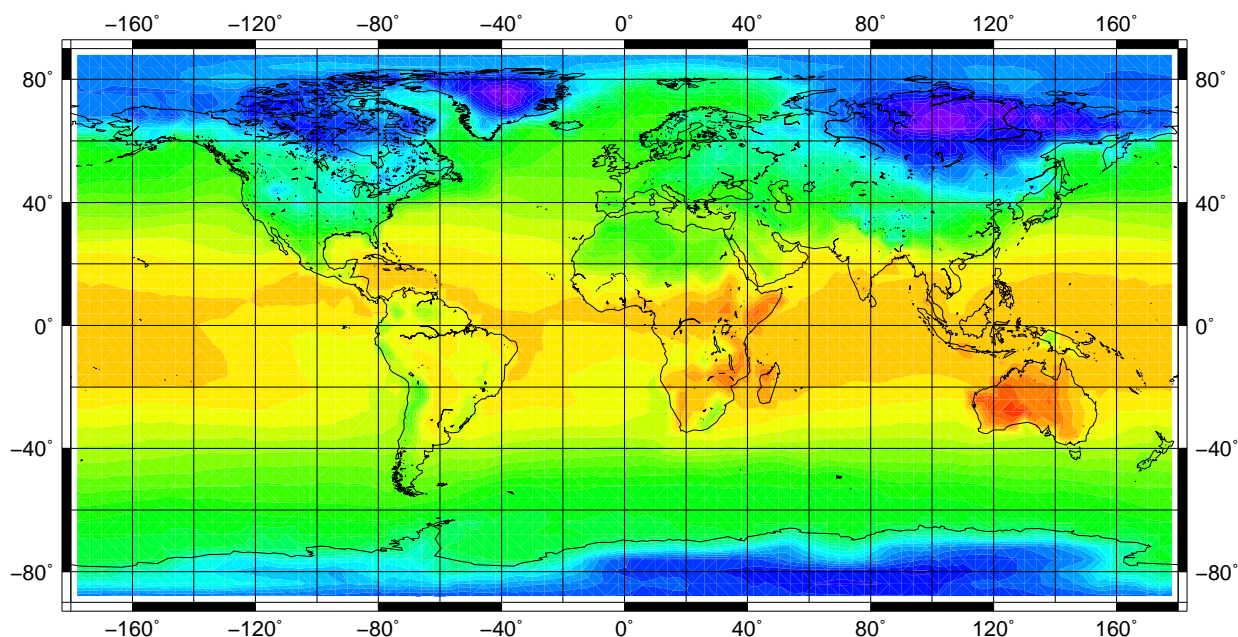
### Operators

- |                 |   |
|-----------------|---|
| <b>gmtxyz</b>   | <p>GMT xyz format</p> <p>The operator exports the first field to the GMT xyz ASCII format. The output can be used to create contour plots with the GMT module pscontour.</p>                              |
| <b>gmtcells</b> | <p>GMT multiple segment format</p> <p>The operator exports the first field to the GMT multiple segment ASCII format. The output can be used to create shaded gridfill plots with the GMT module psxy.</p> |

### Example

- 1) GMT shaded contour plot of a global temperature field with a resolution of 4 degree. The contour interval is 3 with a rainbow color table.

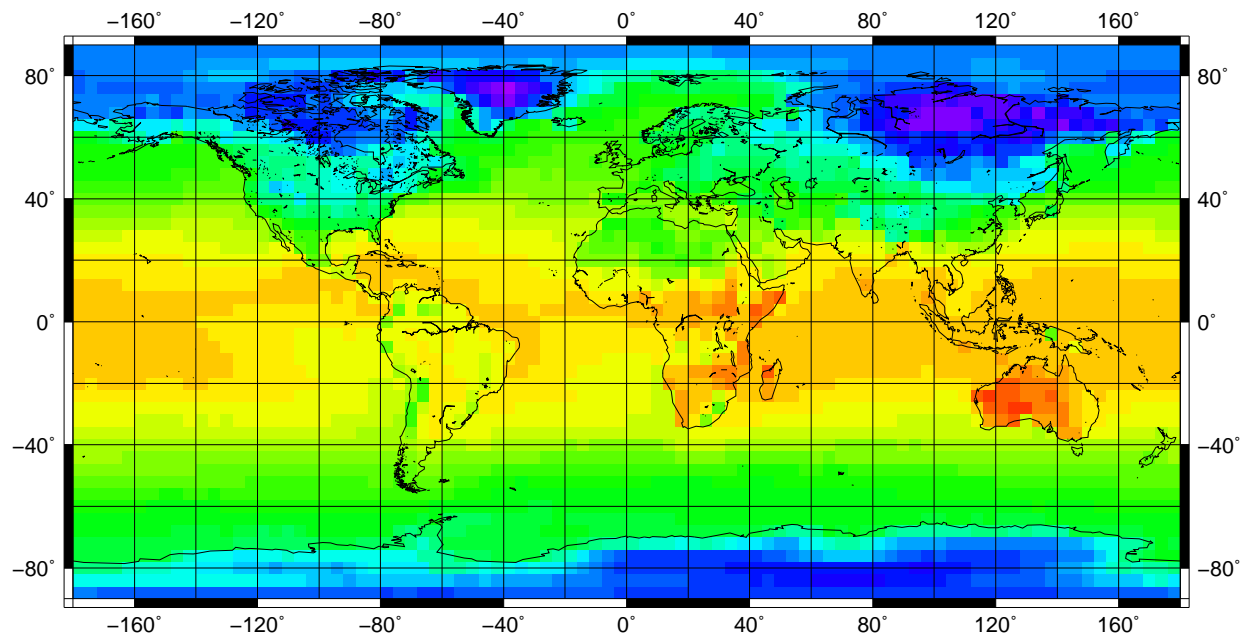
```
cdo gmtxyz temp > data.gmt
makecpt -T213/318/3 -Crainbow > gmt.cpt
pscontour -K -JQ0/10i -Rd -I -Cgmt.cpt data.gmt > gmtplot.ps
pscoast -O -J -R -Dc -W -B40g20 >> gmtplot.ps
```



- 2) GMT shaded gridfill plot of a global temperature field with a resolution of 4 degree. The contour interval is 3 with a rainbow color table.



```
cdo gmtcells temp > data.gmt  
makecpt -T213/318/3 -Crainbow > gmt.cpt  
psxy -K -JQ0/10i -Rd -L -Cgmt.cpt -m data.gmt > gmtplot.ps  
pscoast -O -J -R -Dc -W -B40g20 >> gmtplot.ps
```



## 2.15. Miscellaneous

This section contains miscellaneous modules which do not fit to the other sections before.

Here is a short overview of all operators in this section:

<b>gradsdes</b>	GrADS data descriptor file
<b>after</b>	ECHAM standard post processor
<b>bandpass</b>	Bandpass filtering
<b>lowpass</b>	Lowpass filtering
<b>highpass</b>	Highpass filtering
<b>gridarea</b>	Grid cell area
<b>gridweights</b>	Grid cell weights
<b>smooth</b>	Smooth grid points
<b>smooth9</b>	9 point smoothing
<b>setvals</b>	Set list of old values to new values
<b>setrtoc</b>	Set range to constant
<b>setrtoc2</b>	Set range to constant others to constant2
<b>timsort</b>	Sort over the time
<b>const</b>	Create a constant field
<b>random</b>	Create a field with random numbers
<b>topo</b>	Create a field with topography
<b>for</b>	Create a time series
<b>stdatm</b>	Create values for pressure and temperature for hydrostatic atmosphere
<b>uvDestag</b>	Destaggering of u/v wind components
<b>rotuvNorth</b>	Rotate u/v wind to North pole.
<b>projuvLatLon</b>	Cylindrical Equidistant projection
<b>rotuvb</b>	Backward rotation
<b>mastrfu</b>	Mass stream function
<b>sealevelpressure</b>	Sea level pressure
<b>adisit</b>	Potential temperature to in-situ temperature
<b>adipot</b>	In-situ temperature to potential temperature
<b>rhopot</b>	Calculates potential density
<b>histcount</b>	Histogram count
<b>histsum</b>	Histogram sum
<b>histmean</b>	Histogram mean
<b>histfreq</b>	Histogram frequency
<b>sethalo</b>	Set the left and right bounds of a field
<b>wct</b>	Windchill temperature
<b>fdns</b>	Frost days where no snow index per time period

---

<b>strwin</b>	Strong wind days index per time period
<b>strbre</b>	Strong breeze days index per time period
<b>strgal</b>	Strong gale days index per time period
<b>hurr</b>	Hurricane days index per time period
<b>cmorlite</b>	CMOR lite

### 2.15.1. GRADSDES - GrADS data descriptor file

#### Synopsis

```
gradsdes[,mapversion] infile
```

#### Description

Creates a [GrADS] data descriptor file. Supported file formats are GRIB1, NetCDF, SERVICE, EXTRA and IEG. For GRIB1 files the GrADS map file is also generated. For SERVICE and EXTRA files the grid have to be specified with the **CDO** option '-g <grid>'. This module takes infile in order to create filenames for the descriptor (infile.ct1) and the map (infile.gmp) file.

#### Parameter

**mapversion**    **INTEGER**      Format version of the GrADS map file for GRIB1 datasets. Use 1 for a machine specific version 1 GrADS map file, 2 for a machine independent version 2 GrADS map file and 4 to support GRIB files >2GB. A version 2 map file can be used only with GrADS version 1.8 or newer. A version 4 map file can be used only with GrADS version 2.0 or newer. The default is 4 for files >2GB, otherwise 2.

#### Example

To create a GrADS data descriptor file from a GRIB1 dataset use:

```
cdo gradsdes infile.grb
```

This will create a descriptor file with the name infile.ct1 and the map file infile.gmp.

Assumed the input GRIB1 dataset has 3 variables over 12 timesteps on a Gaussian N16 grid. The contents of the resulting GrADS data description file is approximately:

```
DSET ^infile.grb
DTYPE GRIB
INDEX ^infile.gmp
XDEF 64 LINEAR 0.000000 5.625000
YDEF 32 LEVELS -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
ZDEF 4 LEVELS 925 850 500 200
TDEF 12 LINEAR 12:00Z1jan1987 1mo
TITLE infile.grb T21 grid
OPTIONS yrev
UNDEF -9e+33
VARS 3
geosp 0 129,1,0 surface geopotential (orography) [m^2/s^2]
t      4 130,99,0 temperature [K]
tslm1 0 139,1,0 surface temperature of land [K]
ENDVARS
```

## 2.15.2. AFTERBURNER - ECHAM standard post processor

### Synopsis

```
after[,vct] infile outfile
```

### Description

The "afterburner" is the standard post processor for [\[ECHAM\]](#) data which provides the following operations:

- Extract specified variables and levels
- Compute derived variables
- Transform spectral data to Gaussian grid representation
- Vertical interpolation to pressure levels
- Compute temporal means

This operator reads selection parameters as namelist from stdin. Use the UNIX redirection "<namelistfile" to read the namelist from file.

### Namelist

Namelist parameter and there defaults:

```
TYPE=0, CODE=-1, LEVEL=-1, INTERVAL=0, MEAN=0, EXTRAPOLATE=0
```

**TYPE** controls the transformation and vertical interpolation. Transforming spectral data to Gaussian grid representation and vertical interpolation to pressure levels are performed in a chain of steps. The **TYPE** parameter may be used to stop the chain at a certain step. Valid values are:

```
TYPE = 0 : Hybrid   level spectral coefficients
TYPE = 10 : Hybrid  level fourier  coefficients
TYPE = 11 : Hybrid  level zonal mean sections
TYPE = 20 : Hybrid  level gauss grids
TYPE = 30 : Pressure level gauss grids
TYPE = 40 : Pressure level fourier  coefficients
TYPE = 41 : Pressure level zonal mean sections
TYPE = 50 : Pressure level spectral coefficients
TYPE = 60 : Pressure level fourier  coefficients
TYPE = 61 : Pressure level zonal mean sections
TYPE = 70 : Pressure level gauss grids
```

Vorticity, divergence, streamfunction and velocity potential need special treatment in the vertical transformation. They are not available as types 30, 40 and 41. If you select one of these combinations, type is automatically switched to the equivalent types 70, 60 and 61. The type of all other variables will be switched too, because the type is a global parameter.

**CODE** selects the variables by the ECHAM GRIB1 code number (1-255). The default value **-1** processes all detected codes. Derived variables computed by the afterburner:

Code	Name	Longname	Units	Level	Needed Codes
34	low_cld	low cloud		single	223 on modellevel
35	mid_cld	mid cloud		single	223 on modellevel
36	hih_cld	high cloud		single	223 on modellevel
131	u	u-velocity	m/s	atm (ml+pl)	138, 155
132	v	v-velocity	m/s	atm (ml+pl)	138, 155
135	omega	vertical velocity	Pa/s	atm (ml+pl)	138, 152, 155
148	stream	streamfunction	$m^2/s$	atm (ml+pl)	131, 132
149	velopot	velocity potential	$m^2/s$	atm (ml+pl)	131, 132
151	slp	mean sea level pressure	Pa	surface	129, 130, 152
156	geopoth	geopotential height	m	atm (ml+pl)	129, 130, 133, 152
157	rhumidity	relative humidity		atm (ml+pl)	130, 133, 152
189	scfs	surface solar cloud forcing		surface	176-185
190	tcfs	surface thermal cloud forcing		surface	177-186
191	sclf0	top solar cloud forcing		surface	178-187
192	tcf0	top thermal cloud forcing		surface	179-188
259	windspeed	windspeed	m/s	atm (ml+pl)	$\sqrt{u^2 + v^2}$
260	precip	total precipitation		surface	142+143

**LEVEL** selects the hybrid or pressure levels. The allowed values depends on the parameter **TYPE**. The default value **-1** processes all detected levels.

**INTERVAL** selects the processing interval. The default value **0** process data on monthly intervals. **INTERVAL=1** sets the interval to daily.

**MEAN=1** compute and write monthly or daily mean fields. The default value **0** writes out all timesteps.

**EXTRAPOLATE=0** switch of the extrapolation of missing values during the interpolation from model to pressure level (only available with **MEAN=0** and **TYPE=30**). The default value **1** extrapolate missing values.

Possible combinations of **TYPE**, **CODE** and **MEAN**:

TYPE	CODE	MEAN
0/10/11	130 temperature	0
0/10/11	131 u-velocity	0
0/10/11	132 v-velocity	0
0/10/11	133 specific humidity	0
0/10/11	138 vorticity	0
0/10/11	148 streamfunction	0
0/10/11	149 velocity potential	0
0/10/11	152 LnPs	0
0/10/11	155 divergence	0
>11	all codes	0/1

## Parameter

**vct**    **STRING**    File with VCT in ASCII format

## Example

To interpolate ECHAM hybrid model level data to pressure levels of 925, 850, 500 and 200 hPa, use:

```
cdo after infile outfile << EON
  TYPE=30 LEVEL=92500,85000,50000,20000
EON
```

### 2.15.3. FILTER - Time series filtering

#### Synopsis

```
bandpass,fmin,fmax infile outfile
lowpass,fmax infile outfile
highpass,fmin infile outfile
```

#### Description

This module takes the time series for each gridpoint in *infile* and (fast fourier) transforms it into the frequency domain. According to the particular operator and its parameters certain frequencies are filtered (set to zero) in the frequency domain and the spectrum is (inverse fast fourier) transformed back into the time domain. To determine the frequency the time-axis of *infile* is used. (Data should have a constant time increment since this assumption applies for transformation. However, the time increment has to be different from zero.) All frequencies given as parameter are interpreted per year. This is done by the assumption of a 365-day calendar. Consequently if you want to perform multiyear-filtering accurately you have to delete the 29th of February. If your *infile* has a 360 year calendar the frequency parameters *fmin* respectively *fmax* should be multiplied with a factor of 360/365 in order to obtain accurate results. For the set up of a frequency filter the frequency parameters have to be adjusted to a frequency in the data. Here *fmin* is rounded down and *fmax* is always rounded up. Consequently it is possible to use bandpass with *fmin=fmax* without getting a zero-field for *outfile*. Hints for efficient usage:

- to get reliable results the time-series has to be detrended (`cdo detrend`)
- the lowest frequency greater zero that can be contained in *infile* is  $1/(N \cdot dT)$ ,
- the greatest frequency is  $1/(2dT)$  (Nyquist frequency),

with *N* the number of timesteps and *dT* the time increment of *infile* in years.

#### Operators

<b>bandpass</b>	Bandpass filtering Bandpass filtering (pass for frequencies between <i>fmin</i> and <i>fmax</i> ). Suppresses all variability outside the frequency range specified by [ <i>fmin</i> , <i>fmax</i> ].
<b>lowpass</b>	Lowpass filtering Lowpass filtering (pass for frequencies lower than <i>fmax</i> ). Suppresses all variability with frequencies greater than <i>fmax</i> .
<b>highpass</b>	Highpass filtering Highpass filtering (pass for frequencies greater than <i>fmin</i> ). Suppresses all variability with frequencies lower than <i>fmin</i> .

#### Parameter

<i>fmin</i>	FLOAT	Minimum frequency per year that passes the filter.
<i>fmax</i>	FLOAT	Maximum frequency per year that passes the filter.

#### Note

For better performace of these operators use the **CDO** configure option `–with-fftw3`.

## Example

Now assume your data are still hourly for a time period of 5 years but with a 365/366-day- calendar and you want to suppress the variability on timescales greater or equal to one year (we suggest here to use a number  $x$  bigger than one (e.g.  $x=1.5$ ) since there will be dominant frequencies around the peak (if there is one) as well due to the issue that the time series is not of infinite length). Therefore you can use the following:

```
cdo highpass,x -del29feb infile outfile
```

Accordingly you might use the following to suppress variability on timescales shorter than one year:

```
cdo lowpass,1 -del29feb infile outfile
```

Finally you might be interested in 2-year variability. If you want to suppress the seasonal cycle as well as say the longer cycles in climate system you might use

```
cdo bandpass,x,y -del29feb infile outfile
```

with  $x \leq 0.5$  and  $y \geq 0.5$ .

## 2.15.4. GRIDCELL - Grid cell quantities

### Synopsis

```
<operator> infile outfile
```

### Description

This module reads the grid cell area of the first grid from the input stream. If the grid cell area is missing it will be computed from the grid description. Depending on the chosen operator the grid cell area or weights are written to the output stream.

### Operators

<b>gridarea</b>	Grid cell area Writes the grid cell area to the output stream. If the grid cell area have to be computed it is scaled with the earth radius to square meters.
<b>gridweights</b>	Grid cell weights Writes the grid cell area weights to the output stream.

### Environment

PLANET_RADIUS	This variable is used to scale the computed grid cell areas to square meters. By default PLANET_RADIUS is set to an earth radius of 6371000 meter.
---------------	--



## 2.15.5. SMOOTH - Smooth grid points

### Synopsis

```
smooth[,options] infile outfile
```

```
smooth9 infile outfile
```

### Description

Smooth all grid points of a horizontal grid. Options is a comma separated list of "key=value" pairs with optional parameters.

### Operators

- smooth** Smooth grid points  
Performs a N point smoothing on all input fields. The number of points used depend on the search radius (radius) and the maximum number of points (maxpoints). Per default all points within the search radius of 1degree are used. The weights for the points depend on the form of the curve and the distance. The implemented form of the curve is linear with constant default weights of 0.25 at distance 0 (weight0) and at the search radius (weightR).
- smooth9** 9 point smoothing  
Performs a 9 point smoothing on all fields with a quadrilateral curvilinear grid. The result at each grid point is a weighted average of the grid point plus the 8 surrounding points. The center point receives a weight of 1.0, the points at each side and above and below receive a weight of 0.5, and corner points receive a weight of 0.3. All 9 points are multiplied by their weights and summed, then divided by the total weight to obtain the smoothed value. Any missing data points are not included in the sum; points beyond the grid boundary are considered to be missing. Thus the final result may be the result of an averaging with less than 9 points.

### Parameter

<i>nsmooth</i>	INTEGER	Number of times to smooth, default nsmooth=1
<i>radius</i>	STRING	Search radius, default radius=1deg (units: deg, rad, km, m)
<i>maxpoints</i>	INTEGER	Maximum number of points, default maxpoints=2147483647
<i>form</i>	STRING	Form of the curve, default form=linear
<i>weight0</i>	FLOAT	Weight at distance 0, default weight0=0.25
<i>weightR</i>	FLOAT	Weight at the search radius, default weightR=0.25

## 2.15.6. REPLACEVALUES - Replace variable values

### Synopsis

```
setvals,oldval,newval[,...] infile outfile
setrtoc,rmin,rmax,c infile outfile
setrtoc2,rmin,rmax,c,c2 infile outfile
```

### Description

This module replaces old variable values with new values, depending on the operator.

### Operators

**setvals** Set list of old values to new values  
Supply a list of n pairs of old and new values.

**setrtoc** Set range to constant  

$$o(t, x) = \begin{cases} c & \text{if } i(t, x) \geq rmin \wedge i(t, x) \leq rmax \\ i(t, x) & \text{if } i(t, x) < rmin \vee i(t, x) > rmax \end{cases}$$

**setrtoc2** Set range to constant others to constant2  

$$o(t, x) = \begin{cases} c & \text{if } i(t, x) \geq rmin \wedge i(t, x) \leq rmax \\ c2 & \text{if } i(t, x) < rmin \vee i(t, x) > rmax \end{cases}$$

### Parameter

<i>oldval,newval,...</i>	FLOAT	Pairs of old and new values
<i>rmin</i>	FLOAT	Lower bound
<i>rmax</i>	FLOAT	Upper bound
<i>c</i>	FLOAT	New value - inside range
<i>c2</i>	FLOAT	New value - outside range

### 2.15.7. TIMSORT - Timsort

#### Synopsis

```
timsort infile outfile
```

#### Description

Sorts the elements in ascending order over all timesteps for every field position. After sorting it is:

$$o(t_1, x) \leq o(t_2, x) \quad \forall (t_1 < t_2), x$$

#### Example

To sort all field elements of a dataset over all timesteps use:

```
cdo timsort infile outfile
```

### 2.15.8. VARGEN - Generate a field

#### Synopsis

```
const,const,grid outfile
random,grid[,seed] outfile
topo[,grid] outfile
for,start,end[,inc] outfile
stdatm,levels outfile
```

#### Description

Generates a dataset with one or more fields

#### Operators

<b>const</b>	Create a constant field Creates a constant field. All field elements of the grid have the same value.
<b>random</b>	Create a field with random numbers Creates a field with rectangularly distributed random numbers in the interval [0,1].
<b>topo</b>	Create a field with topography Creates a field with topography data, per default on a global half degree grid.
<b>for</b>	Create a time series Creates a time series with field size 1 and field elements beginning with a start value in time step 1 which is increased from one time step to the next.
<b>stdatm</b>	Create values for pressure and temperature for hydrostatic atmosphere Creates pressure and temperature values for the given list of vertical levels. The formulas are: $P(z) = P_0 \exp \left( -\frac{g}{R} \frac{H}{T_0} \log \left( \frac{\exp \left( \frac{z}{H} \right) T_0 + \Delta T}{T_0 + \Delta T} \right) \right)$ $T(z) = T_0 + \Delta T \exp \left( -\frac{z}{H} \right)$

with the following constants

$$\begin{aligned}
 T_0 &= 213\text{K} && : \text{offset to get a surface temperature of } 288\text{K} \\
 \Delta T &= 75\text{K} && : \text{Temperature lapse rate for } 10\text{Km} \\
 P_0 &= 1013.25\text{hPa} && : \text{surface pressure} \\
 H &= 10000.0\text{m} && : \text{scale height} \\
 g &= 9.80665 \frac{\text{m}}{\text{s}^2} && : \text{earth gravity} \\
 R &= 287.05 \frac{\text{J}}{\text{kgK}} && : \text{gas constant for air}
 \end{aligned}$$

This is the solution for the hydrostatic equations and is only valid for the troposphere (constant positive lapse rate). The temperature increase in the stratosphere and other effects of the upper atmosphere are not taken into account.

## Parameter

<i>const</i>	FLOAT	Constant
<i>seed</i>	INTEGER	The seed for a new sequence of pseudo-random numbers [default: 1]
<i>grid</i>	STRING	Target grid description file or name
<i>start</i>	FLOAT	Start value of the loop
<i>end</i>	FLOAT	End value of the loop
<i>inc</i>	FLOAT	Increment of the loop [default: 1]
<i>levels</i>	FLOAT	Target levels in metre above surface

## Example

To create a standard atmosphere dataset on a given horizontal grid:

```
cdo enlarge,gridfile -stdatm,10000,8000,5000,3000,2000,1000,500,200,0 outfile
```

## 2.15.9. WINDTRANS - Wind Transformation

### Synopsis

```
uvDestag,u,v[, -/+0.5[, -/+0.5]] infile outfile
rotuvNorth,u,v infile outfile
projuvLatLon,u,v infile outfile
```

### Description

This module contains special operators for datasets with wind components on a rotated lon/lat grid, e.g. data from the regional model HIRLAM or REMO.

### Operators

<b>uvDestag</b>	Destaggering of u/v wind components This is a special operator for destaggering of wind components. If the file contains a grid with temperature (name='t' or code=11) then grid_temp will be used for destaggered wind.
<b>rotuvNorth</b>	Rotate u/v wind to North pole. This is an operator for transformation of wind-vectors from grid-relative to north-pole relative for the whole file. (FAST implementation with JACOBIANS)
<b>projuvLatLon</b>	Cylindrical Equidistant projection Thus is an operator for transformation of wind-vectors from the globe-spherical coordinate system into a flat Cylindrical Equidistant (lat-lon) projection. (FAST JACOBIAN implementation)

### Parameter

<i>u,v</i>	STRING	Pair of u,v wind components (use variable names or code numbers)
<i>-/+0.5,-/+0.5</i>	STRING	Destaggered grid offsets are optional (default -0.5,-0.5)

### Example

Typical operator sequence on HIRLAM NWP model output (LAMH\_D11 files):

```
cdo uvDestag,33,34 inputfile inputfile_destag
cdo rotuvNorth,33,34 inputfile_destag inputfile_rotuvN
```

### 2.15.10. ROTUVB - Rotation

#### Synopsis

```
rotuvb,u,v,... infile outfile
```

#### Description

This is a special operator for datasets with wind components on a rotated grid, e.g. data from the regional model REMO. It performs a backward transformation of velocity components U and V from a rotated spherical system to a geographical system.

#### Parameter

<code>u,v,...</code>	STRING	Pairs of zonal and meridional velocity components (use variable names or code numbers)
----------------------	--------	--

#### Example

To transform the u and v velocity of a dataset from a rotated spherical system to a geographical system use:

```
cdo rotuvb,u,v infile outfile
```

### 2.15.11. MASTRFU - Mass stream function

#### Synopsis

```
mastrfu infile outfile
```

#### Description

This is a special operator for the post processing of the atmospheric general circulation model [\[ECHAM\]](#). It computes the mass stream function (code=272). The input dataset have to be a zonal mean of v-velocity [m/s] (code=132) on pressure levels.

#### Example

To compute the mass stream function from a zonal mean v-velocity dataset use:

```
cdo mastrfu infile outfile
```

### 2.15.12. DERIVEPAR - Sea level pressure

#### Synopsis

```
sealevelpressure infile outfile
```

#### Description

This operator computes the sea level pressure (`air_pressure_at_sea_level`). Required input fields are `surface_air_pressure`, `surface_geopotential` and `air_temperature` on hybrid sigma pressure levels.

### 2.15.13. ADISIT - Potential temperature to in-situ temperature and vice versa

#### Synopsis

```
adisit[,pressure] infile outfile
adipot infile outfile
```

#### Description

#### Operators

- adisit** Potential temperature to in-situ temperature  
 This is a special operator for the post processing of the ocean and sea ice model output. It converts potential temperature adiabatically to in-situ temperature to(t, s, p). Required input fields are sea water potential temperature (name=tho; code=2) and sea water salinity (name=sao; code=5). Pressure is calculated from the level information or can be specified by the optional parameter. Output fields are sea water temperature (name=to; code=20) and sea water salinity (name=s; code=5).
- adipot** In-situ temperature to potential temperature  
 This is a special operator for the post processing of the ocean and sea ice model output. It converts in-situ temperature to potential temperature tho(to, s, p). Required input fields are sea water in-situ temperature (name=t; code=2) and sea water salinity (name=sao; code=5). Pressure is calculated from the level information or can be specified by the optional parameter. Output fields are sea water temperature (name=tho; code=2) and sea water salinity (name=s; code=5).

#### Parameter

*pressure*      FLOAT      Pressure in bar (constant value assigned to all levels)

### 2.15.14. RHOPOT - Calculates potential density

#### Synopsis

```
rhopot[,pressure] infile outfile
```

#### Description

This is a special operator for the post processing of the ocean and sea ice model [MPIOM]. It calculates the sea water potential density (name=rhopot; code=18). Required input fields are sea water in-situ temperature (name=to; code=20) and sea water salinity (name=sao; code=5). Pressure is calculated from the level information or can be specified by the optional parameter.

#### Parameter

*pressure*      FLOAT      Pressure in bar (constant value assigned to all levels)

#### Example

To compute the sea water potential density from the potential temperature use this operator in combination with [adisit](#):

```
cdo rhopot -adisit infile outfile
```

## 2.15.15. HISTOGRAM - Histogram

### Synopsis

```
<operator>,<bounds> infile outfile
```

### Description

This module creates bins for a histogram of the input data. The bins have to be adjacent and have non-overlapping intervals. The user has to define the bounds of the bins. The first value is the lower bound and the second value the upper bound of the first bin. The bounds of the second bin are defined by the second and third value, aso. Only 2-dimensional input fields are allowed. The output file contains one vertical level for each of the bins requested.

### Operators

<b>histcount</b>	Histogram count Number of elements in the bin range.
<b>histsum</b>	Histogram sum Sum of elements in the bin range.
<b>histmean</b>	Histogram mean Mean of elements in the bin range.
<b>histfreq</b>	Histogram frequency Relative frequency of elements in the bin range.

### Parameter

*bounds*      FLOAT      Comma separated list of the bin bounds (-inf and inf valid)

## 2.15.16. SETHALO - Set the left and right bounds of a field

### Synopsis

```
sethalo,<lhs>,<rhs> infile outfile
```

### Description

This operator sets the left and right bounds of the rectangularly understood fields. Positive numbers of the parameter *lhs* enlarges the left bound by the given number of columns from the right bound. The parameter *rhs* does the similar for the right bound. Negative numbers of the parameter *lhs/rhs* can be used to remove the given number of columns of the left and right bounds.

### Parameter

<i>lhs</i>	INTEGER	Left halo
<i>rhs</i>	INTEGER	Right halo



## 2.15.17. WCT - Windchill temperature

### Synopsis

```
wct infile1 infile2 outfile
```

### Description

Let `infile1` and `infile2` be time series of temperature and wind speed records, then a corresponding time series of resulting windchill temperatures is written to `outfile`. The wind chill temperature calculation is only valid for a temperature of  $T \leq 33$  °C and a wind speed of  $v \geq 1.39$  m/s. Whenever these conditions are not satisfied, a missing value is written to `outfile`. Note that temperature and wind speed records have to be given in units of °C and m/s, respectively.

## 2.15.18. FDNS - Frost days where no snow index per time period

### Synopsis

```
fdns infile1 infile2 outfile
```

### Description

Let `infile1` be a time series of the daily minimum temperature `TN` and `infile2` be a corresponding series of daily surface snow amounts. Then the number of days where  $TN < 0$  °C and the surface snow amount is less than 1 cm is counted. The temperature `TN` have to be given in units of Kelvin. The date information of a timestep in `outfile` is the date of the last contributing timestep in `infile`.

## 2.15.19. STRWIN - Strong wind days index per time period

### Synopsis

```
strwin[,v] infile outfile
```

### Description

Let `infile` be a time series of the daily maximum horizontal wind speed `VX`, then the number of days where  $VX > v$  is counted. The horizontal wind speed `v` is an optional parameter with default  $v = 10.5$  m/s. A further output variable is the maximum number of consecutive days with maximum wind speed greater than or equal to `v`. Note that both `VX` and `v` have to be given in units of m/s. Also note that the horizontal wind speed is defined as the square root of the sum of squares of the zonal and meridional wind speeds. The date information of a timestep in `outfile` is the date of the last contributing timestep in `infile`.

### Parameter

<code>v</code>	FLOAT	Horizontal wind speed threshold (m/s, default $v = 10.5$ m/s)
----------------	-------	---

## 2.15.20. STRBRE - Strong breeze days index per time period

### Synopsis

```
strbre infile outfile
```

### Description

Let `infile` be a time series of the daily maximum horizontal wind speed  $VX$ , then the number of days where  $VX$  is greater than or equal to 10.5 m/s is counted. A further output variable is the maximum number of consecutive days with maximum wind speed greater than or equal to 10.5 m/s. Note that  $VX$  is defined as the square root of the sum of squares of the zonal and meridional wind speeds and have to be given in units of m/s. The date information of a timestep in `outfile` is the date of the last contributing timestep in `infile`.

## 2.15.21. STRGAL - Strong gale days index per time period

### Synopsis

```
strgal infile outfile
```

### Description

Let `infile` be a time series of the daily maximum horizontal wind speed  $VX$ , then the number of days where  $VX$  is greater than or equal to 20.5 m/s is counted. A further output variable is the maximum number of consecutive days with maximum wind speed greater than or equal to 20.5 m/s. Note that  $VX$  is defined as the square root of the sum of square of the zonal and meridional wind speeds and have to be given in units of m/s. The date information of a timestep in `outfile` is the date of the last contributing timestep in `infile`.

## 2.15.22. HURR - Hurricane days index per time period

### Synopsis

```
hurr infile outfile
```

### Description

Let `infile` be a time series of the daily maximum horizontal wind speed  $VX$ , then the number of days where  $VX$  is greater than or equal to 32.5 m/s is counted. A further output variable is the maximum number of consecutive days with maximum wind speed greater than or equal to 32.5 m/s. Note that  $VX$  is defined as the square root of the sum of squares of the zonal and meridional wind speeds and have to be given in units of m/s. The date information of a timestep in `outfile` is the date of the last contributing timestep in `infile`.

### 2.15.23. CMORLITE - CMOR lite

#### Synopsis

```
cmorlite,table[,convert] infile outfile
```

#### Description

The [\[CMOR\]](#) (Climate Model Output Rewriter) library comprises a set of functions, that can be used to produce CF-compliant NetCDF files that fulfill the requirements of many of the climate community's standard model experiments. These experiments are collectively referred to as MIP's. Much of the metadata written to the output files is defined in MIP-specific tables, typically made available from each MIP's web site.

The **CDO** operator cmorlite process the header and variable section of such MIP tables and writes the result with the internal IO library [\[CDI\]](#). In addition to the CMOR 2 and 3 table format, the **CDO** parameter table format is also supported. The following parameter table entries are available:

Entry	Type	Description
<b>name</b>	WORD	Name of the variable
<b>out_name</b>	WORD	New name of the variable
<b>type</b>	WORD	Data type (real or double)
<b>standard_name</b>	WORD	As defined in the CF standard name table
<b>long_name</b>	STRING	Describing the variable
<b>units</b>	STRING	Specifying the units for the variable
<b>comment</b>	STRING	Information concerning the variable
<b>cell_methods</b>	STRING	Information concerning calculation of means or climatologies
<b>cell_measures</b>	STRING	Indicates the names of the variables containing cell areas and volumes
<b>missing_value</b>	FLOAT	Specifying how missing data will be identified
<b>valid_min</b>	FLOAT	Minimum valid value
<b>valid_max</b>	FLOAT	Maximum valid value
<b>ok_min_mean_abs</b>	FLOAT	Minimum absolute mean
<b>ok_max_mean_abs</b>	FLOAT	Maximum absolute mean
<b>factor</b>	FLOAT	Scale factor
<b>delete</b>	INTEGER	Set to 1 to delete variable
<b>convert</b>	INTEGER	Set to 1 to convert the unit if necessary

Most of the above entries are stored as variables attributes, some of them are handled differently. The variable **name** is used as a search key for the parameter table. **valid\_min**, **valid\_max**, **ok\_min\_mean\_abs** and **ok\_max\_mean\_abs** are used to check the range of the data.

#### Parameter

<i>table</i>	STRING	Name of the CMOR table as specified from PCMDI
<i>convert</i>	STRING	Converts the units if necessary

#### Example

Here is an example of a parameter table for one variable:

```
prompt> cat mypartab
&parameter
  name          = t
```

```
out_name      = ta
standard_name = air_temperature
units         = "K"
missing_value = 1e+20
valid_min     = 157.1
valid_max     = 336.3
/
```

To apply this parameter table to a dataset use:

```
cdo -f nc cmorlite,mypartab,convert infile outfile
```

This command renames the variable **t** to **ta**. The standard name of this variable is set to **air\_temperature** and the unit is set to **[K]** (converts the unit if necessary). The missing value will be set to **1e+20**. In addition it will be checked whether the values of the variable are in the range of **157.1** to **336.3**. The result will be stored in NetCDF.

## 3. Contributors

### 3.1. History

**CDO** was originally developed by Uwe Schulzweida at the Max Planck Institute for Meteorology (MPI-M). The MPI-M, together with the DKRZ, has a long history in the development of tools for processing climate data. **CDO** was inspired by some of these tools, such as the PINGO package and the GRIB-Modules.

PINGO<sup>1</sup> was developed by Jürgen Waszkewitz, Peter Lenzen, and Nathan Gillet in 1995 at the DKRZ, Hamburg (Germany). **CDO** has a similar user interface and uses some of the PINGO routines.

The GRIB-Modules was developed by Heiko Borgert and Wolfgang Welke in 1991 at the MPI-M. **CDO** is using a similar module structure and also some of the routines.

### 3.2. External sources

**CDO** has incorporated code from several sources:

**afterburner** is a postprocessing program for ECHAM data and ECMWF analysis data, originally developed by Edilbert Kirk, Michael Ponater and Arno Hellbach. The afterburner code was modified for the **CDO** operators [after](#), [sp2gp](#), [gp2sp](#).

**SCRIP** is a software package used to generate interpolation weights for remapping fields from one grid to another in spherical geometry [[SCRIP](#)]. It was developed at the Los Alamos National Laboratory by Philip W. Jones. The SCRIP library was converted from Fortran to ANSI C and is used as the base for the remapping operators in **CDO**.

**YAC** (Yet Another Coupler) was jointly developed by DKRZ and MPI-M by Moritz Hanke and Rene Redler [[YAC](#)]. **CDO** is using the clipping routines for the conservative remapping with [remapycon](#).

**libkdtree** a C99 implementation of the kd-tree algorithm developed by Jörg Dietrich.

**CDO** uses tools from the GNU project, including automake, and libtool.

### 3.3. Contributors

The primary contributors to the **CDO** development have been:

**Uwe Schulzweida** : Concept, design and implementation of **CDO**, project coordination, and releases.

**Luis Kornblüeh** : He supports **CDO** from the beginning. His main contributions are GRIB performance and compression, GME and unstructured grid support. Luis also helps with design and planning.

**Ralf Müller** : He is working on **CDO** since 2009. His main contributions are the implementation of the User Portal, the ruby and python interface for all **CDO** operators, the building process and the Windows support. The **CDO** User Portal was funded by the European Commission infrastructure project IS-ENES. Ralf also helps a lot with the user support. Implemented operators: [showunit](#), [stdatm](#), [intlevel3d](#), [consecsum](#), [consects](#), [ngrid](#), [ngridpoints](#), [reducegrid](#)

**Cedrick Anson** : He worked on the **CDO** software package as a student assistant at MPI-M from 2007-2011. Implemented operators: [eof](#), [eof3d](#), [enscrps](#), [ensbrs](#), [maskregion](#), [bandpass](#), [lowpass](#), [highpass](#), [smooth9](#)

---

<sup>1</sup>Procedural INterface for GRIB formatted Objects

**Oliver Heidmann** : He worked on the **CDO** software package as a student assistant at MPI-M from 2015-2017.

**Fabian Wachsmann** : He is working on **CDO** for the CMIP6 project and is responsible for the operator `cmor`.

**Ralf Quast** : He worked on **CDO** on behalf of the Service Gruppe Anpassung (SGA), DKRZ in 2006. Implemented all ECA Indices of Daily Temperature and Precipitation Extremes, all percentile operators, module `YDRUNSTAT` and `wct`.

**Kameswarrao Modali** : Implemented operators: `contour`, `shaded`, `grfill`, `vector`, `graph`.

**Michal Koutek** : Implemented operators: `selmulti delmulti`, `changemulti`, `samplegrid`, `uvDestag`, `rotuvNorth`, `projuvLatLon`.

**Etienne Tourigny** : Implemented operators: `setclonlatbox`, `setcindexbox`, `setvals`, `splitsel`, `histfreq`, `setrtoc`, `setrtoc2`.

**Karl-Hermann Wieners** : Implemented operators: `aexpr`, `aexprf`, `selzaxisname`.

Many users have contributed to **CDO** by sending bug reports, patches and suggestions over time. Very helpful is also the active participation in the user forum of some users. Here is an incomplete list:

Jaison-Thomas Ambadan, Harald Anlauf, Andy Aschwanden, Stefan Bauer, Simon Blessing, Renate Brokopf, Michael Boettinger, Tim Brücher, Reinhard Budich, Martin Claus, Traute Crüger, Irene Fischer-Bruns, Chris Fletscher, Helmut Frank, Kristina Fröhlich, Oliver Fuhrer, Monika Esch, Pier Giuseppe Fogli, Beate Gayer, Veronika Gayler, Marco Giorgetta, David Gobbett, Holger Goettel, Helmut Haak, Stefan Hagemann, Angelika Heil, Barbara Hennemuth, Daniel Hernandez, Nathanael Huebbe, Thomas Jahns, Frank Kaspar, Daniel Klocke, Edi Kirk, Yvonne Küstermann, Stefanie Legutke, Leonidas Linardakis, Stephan Lorenz, Frank Lunkeit, Karin Meier-Fleischer, Uwe Mikolajewicz, Laura Niederdrenk, Dirk Notz, Hans-Jürgen Panitz, Ronny Petrik, Swantje Preuschmann, Florian Prill, Asela Rajapakse, Daniel Reinert, Hannes Reuter, Mathis Rosenhauer, Reiner Schnur, Martin Schultz, Dennis Shea, Kevin Sieck, Martin Stendel, Bjorn Stevens, Martina Stockhaus, Claas Teichmann, Jörg Trentmann, Álvaro M. Valdebenito, Geert Jan van Oldenborgh, Jin-Song von Storch, David Wang, Joerg Wegner, Heiner Widmann, Claudia Wunram, Klaus Wyser

Please let me know if your name was omitted!

# Bibliography

- [CDI]  
[Climate Data Interface](#), from the [Max Planck Institute for Meteorologie](#)
- [CM-SAF]  
[Satellite Application Facility on Climate Monitoring](#), from the [German Weather Service \(Deutscher Wetterdienst, DWD\)](#)
- [CMOR]  
[Climate Model Output Rewriter](#), from the [Program For Climate Model Diagnosis and Intercomparison \(PCMDI\)](#)
- [ecCodes]  
[API for GRIB decoding/encoding](#), from the [European Centre for Medium-Range Weather Forecasts \(ECMWF\)](#)
- [ECHAM]  
[The atmospheric general circulation model ECHAM5](#), from the [Max Planck Institute for Meteorologie](#)
- [GMT]  
[The Generic Mapping Tool](#), from the [School of Ocean and Earth Science and Technology \(SOEST\)](#)
- [GrADS]  
[Grid Analysis and Display System](#), from the [Center for Ocean-Land-Atmosphere Studies \(COLA\)](#)
- [GRIB]  
[GRIB version 1](#), from the [World Meteorological Organisation \(WMO\)](#)
- [HDF5]  
[HDF version 5](#), from the [HDF Group](#)
- [INTERA]  
[INTERA Software Package](#), from the [Max Planck Institute for Meteorologie](#)
- [Magics]  
[Magics Software Package](#), from the [European Centre for Medium-Range Weather Forecasts \(ECMWF\)](#)
- [MPIOM]  
[Ocean and sea ice model](#), from the [Max Planck Institute for Meteorologie](#)
- [NetCDF]  
[NetCDF Software Package](#), from the [UNIDATA Program Center of the University Corporation for Atmospheric Research](#)
- [PINGO]  
[The PINGO package](#), from the [Model & Data group](#) at the [Max Planck Institute for Meteorologie](#)
- [REMO]  
[Regional Model](#), from the [Max Planck Institute for Meteorologie](#)
- [Peisendorfer]  
Rudolph W. Peisendorfer: [Principal Component Analysis](#), Elsevier (1988)
- [PROJ.4]  
[Cartographic Projections Library](#), originally written by Gerald Evenden then of the USGS.
- [SCRIP]  
[SCRIP Software Package](#), from the [Los Alamos National Laboratory](#)

[szip]

[Szip compression software](#), developed at University of New Mexico.

[vonStorch]

Hans von Storch, Walter Zwiers: Statistical Analysis in Climate Research, Cambridge University Press (1999)

[YAC]

[YAC - Yet Another Coupler Software Package](#), from DKRZ and MPI for Meteorologie



## A. Environment Variables

The following table describes the environment variables that affect **CDO**.

Variable name	Default	Description
CDO_FILE_SUFFIX	None	Default file suffix. This suffix will be added to the output file name instead of the filename extension derived from the file format. NULL will disable the adding of a file suffix.
CDO_HISTORY_INFO	1	Append NetCDF global attribute histroy
CDO_PCTL_NBINS	101	Number of histogram bins.
CDO_RESET_HISTORY	0	Set to 1 to reset the NetCDF <i>history</i> global attribute.
CDO_REMAP_NORM	fracarea	Choose the normalization for the conservative interpolation
CDO_GRIDSEARCH_RADIUS	180	Grid search radius in degree. Used by the operators setmisstonn, remapdis and remapnn.
CDO_TIMESTAT_DATE	None	Set target timestamp of a time statistic operator to the "first", "middle", "midhigh" or "last" contributing source timestep.
CDO_USE_FFTW	1	Set to 0 to switch off usage of FFTW. Used in the Filter module.
CDO_VERSION_INFO	1	Set to 0 to disable NetCDF global attribute CDO

## B. Parallelized operators

Some of the **CDO** operators are parallelized with OpenMP. To use **CDO** with multiple OpenMP threads, you have to set the number of threads with the option '-P'. Here is an example to distribute the bilinear interpolation on 8 OpenMP threads:

```
cdo -P 8 remapbil,targetgrid infile outfile
```

The following **CDO** operators are parallelized with OpenMP:

Module	Operator	Description
Detrend	detrend	Detrend
Ensstat	ensmin	Ensemble minimum
Ensstat	ensmax	Ensemble maximum
Ensstat	enssum	Ensemble sum
Ensstat	ensmean	Ensemble mean
Ensstat	ensavg	Ensemble average
Ensstat	ensvar	Ensemble variance
Ensstat	ensstd	Ensemble standard deviation
Ensstat	enspctl	Ensemble percentiles
Filter	bandpass	Bandpass filtering
Filter	lowpass	Lowpass filtering
Filter	highpass	Highpass filtering
Fourier	fourier	Fourier transformation
Genweights	genbil	Generate bilinear interpolation weights
Genweights	genbic	Generate bicubic interpolation weights
Genweights	gendis	Generate distance-weighted average remap weights
Genweights	gennn	Generate nearest neighbor remap weights
Genweights	gencon	Generate 1st order conservative remap weights
Genweights	gencon2	Generate 2nd order conservative remap weights
Genweights	genlaf	Generate largest area fraction remap weights
Gridboxstat	gridboxmin	Gridbox minimum
Gridboxstat	gridboxmax	Gridbox maximum
Gridboxstat	gridboxsum	Gridbox sum
Gridboxstat	gridboxmean	Gridbox mean
Gridboxstat	gridboxavg	Gridbox average
Gridboxstat	gridboxvar	Gridbox variance
Gridboxstat	gridboxstd	Gridbox standard deviation
Remapeta	remapeta	Remap vertical hybrid level
Remap	remapbil	Bilinear interpolation
Remap	remapbic	Bicubic interpolation
Remap	remapdis	Distance-weighted average remapping
Remap	remapnn	Nearest neighbor remapping
Remap	remapcon	First order conservative remapping
Remap	remapcon2	Second order conservative remapping
Remap	remaplaf	Largest area fraction remapping

## C. Standard name table

The following CF standard names are supported by **CDO**.

CF standard name	Units	GRIB 1 code	variable name
surface_geopotential	m2 s-2	129	geosp
air_temperature	K	130	ta
specific_humidity	1	133	hus
surface_air_pressure	Pa	134	aps
air_pressure_at_sea_level	Pa	151	psl
geopotential_height	m	156	zg

## D. Grid description examples

### D.1. Example of a curvilinear grid description

Here is an example for the **CDO** description of a curvilinear grid. `xvals/yvals` describe the positions of the 6x5 quadrilateral grid cells. The first 4 values of `xbounds/ybounds` are the corners of the first grid cell.

<code>gridtype</code>	=	<code>curvilinear</code>																
<code>gridsize</code>	=	30																
<code>xsize</code>	=	6																
<code>ysize</code>	=	5																
<code>xvals</code>	=	-21	-11	0	11	21	30	-25	-13	0	13							
		25	36	-31	-16	0	16	31	43	-38	-21							
		0	21	38	52	-51	-30	0	30	51	64							
<code>xbounds</code>	=	-23	-14	-17	-28		-14	-5	-6	-17		-5	5	6	-6			
		5	14	17	6		14	23	28	17		23	32	38	28			
		-28	-17	-21	-34		-17	-6	-7	-21		-6	6	7	-7			
		6	17	21	7		17	28	34	21		28	38	44	34			
		-34	-21	-27	-41		-21	-7	-9	-27		-7	7	9	-9			
		7	21	27	9		21	34	41	27		34	44	52	41			
		-41	-27	-35	-51		-27	-9	-13	-35		-9	9	13	-13			
		9	27	35	13		27	41	51	35		41	52	63	51			
		-51	-35	-51	-67		-35	-13	-21	-51		-13	13	21	-21			
<code>yvals</code>	=	13	35	51	21		35	51	67	51		51	63	77	67			
		29	32	32	32	29	26	39	42	42	42							
		39	35	48	51	52	51	48	43	57	61							
		62	61	57	51	65	70	72	70	65	58							
<code>ybounds</code>	=	23	26	36	32		26	27	37	36		27	27	37	37			
		27	26	36	37		26	23	32	36		23	19	28	32			
		32	36	45	41		36	37	47	45		37	37	47	47			
		37	36	45	47		36	32	41	45		32	28	36	41			
		41	45	55	50		45	47	57	55		47	47	57	57			
		47	45	55	57		45	41	50	55		41	36	44	50			
		50	55	64	58		55	57	67	64		57	57	67	67			
		57	55	64	67		55	50	58	64		50	44	51	58			
		58	64	72	64		64	67	77	72		67	67	77	77			
		67	64	72	77		64	58	64	72		58	51	56	64			

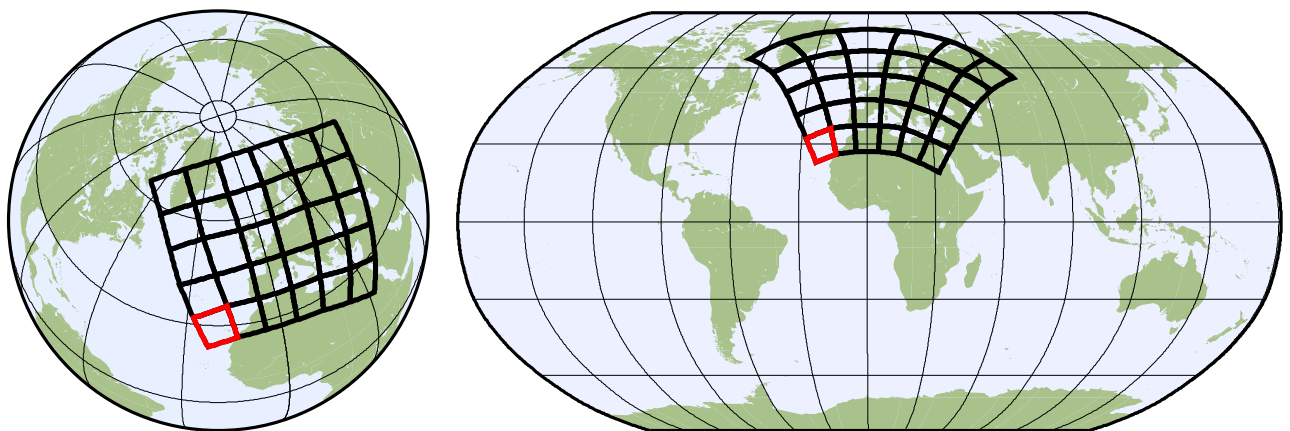


Figure D.1.: Orthographic and Robinson projection of the curvilinear grid, the first grid cell is colored red

## D.2. Example description for an unstructured grid

Here is an example of the **CDO** description for an unstructured grid. xvals/yvals describe the positions of 30 independent hexagonal grid cells. The first 6 values of xbounds/ybounds are the corners of the first grid cell. The grid cell corners have to rotate counterclockwise. The first grid cell is colored red.

```

gridtype = unstructured
gridsize = 30
nvertex = 6
xvals = -36 36 0 -18 18 108 72 54 90 180 144 126 162 -108 -144
        -162 -126 -72 -90 -54 0 72 36 144 108 -144 180 -72 -108 -36
xbounds = 339 0 0 288 288 309 21 51 72 72 0 0
          0 16 21 0 339 344 340 0 -0 344 324 324
          20 36 36 16 0 0 93 123 144 144 72 72
          72 88 93 72 51 56 52 72 72 56 36 36
          92 108 108 88 72 72 165 195 216 216 144 144
          144 160 165 144 123 128 124 144 144 128 108 108
          164 180 180 160 144 144 237 267 288 288 216 216
          216 232 237 216 195 200 196 216 216 200 180 180
          236 252 252 232 216 216 288 304 309 288 267 272
          268 288 288 272 252 252 308 324 324 304 288 288
          345 324 324 36 36 15 36 36 108 108 87 57
          20 15 36 57 52 36 108 108 180 180 159 129
          92 87 108 129 124 108 180 180 252 252 231 201
          164 159 180 201 196 180 252 252 324 324 303 273
          236 231 252 273 268 252 308 303 324 345 340 324
yvals = 58 58 32 0 0 58 32 0 0 58 32 0 0 58 32
        0 0 32 0 0 -58 -58 -32 -58 -32 -58 -32 -58 -32 -32
ybounds = 41 53 71 71 53 41 41 41 53 71 71 53
          11 19 41 53 41 19 -19 -7 11 19 7 -11
          -19 -11 7 19 11 -7 41 41 53 71 71 53
          11 19 41 53 41 19 -19 -7 11 19 7 -11
          -19 -11 7 19 11 -7 41 41 53 71 71 53
          11 19 41 53 41 19 -19 -7 11 19 7 -11
          -19 -11 7 19 11 -7 11 19 41 53 41 19
          -19 -7 11 19 7 -11 -19 -11 7 19 11 -7
          -41 -53 -71 -71 -53 -41 -53 -71 -71 -53 -41 -41
          -19 -41 -53 -41 -19 -11 -53 -71 -71 -53 -41 -41
          -19 -41 -53 -41 -19 -11 -53 -71 -71 -53 -41 -41
          -19 -41 -53 -41 -19 -11 -53 -71 -71 -53 -41 -41
          -19 -41 -53 -41 -19 -11 -19 -41 -53 -41 -19 -11

```

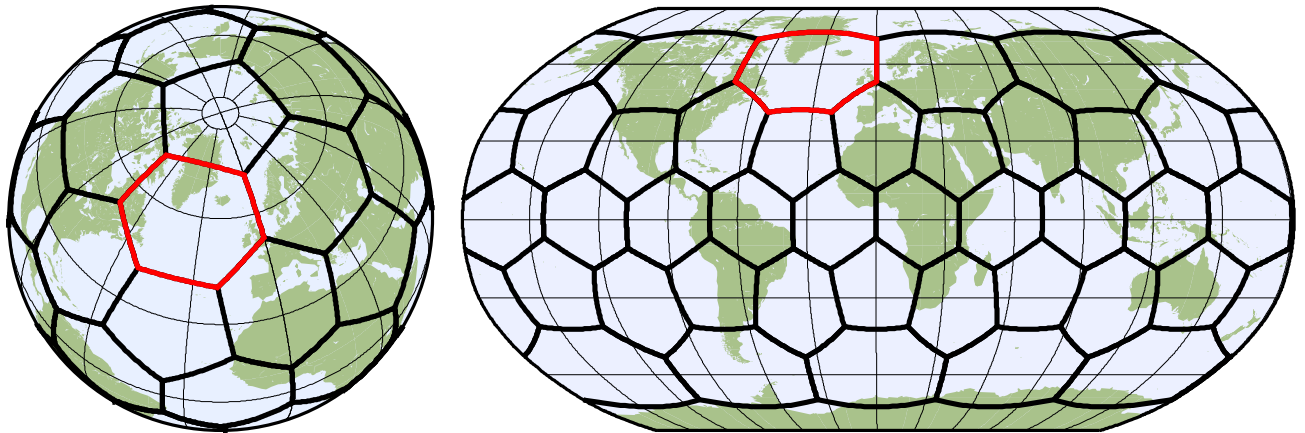


Figure D.2.: Orthographic and Robinson projection of the unstructured grid

# Operator index

## A

abs	77
acos	77
add	79
addc	78
adipot	191
adisit	191
aexpr	74
aexprf	74
after	181
ap2hl	160
ap2pl	160
asin	77
atan	77
atan2	79

## B

bandpass	183
----------	-----

## C

cat	27
changemulti	39
chcode	61
chlevel	61
chlevelc	61
chlevelv	61
chname	61
chparam	61
chunit	61
cmorlite	195
codetab	25
collgrid	35
consecsum	91
consects	91
const	187
copy	27
cos	77

## D

dayavg	112
daymax	112
daymean	112
daymin	112
daypctl	113
dayrange	112
daystd	112
daystd1	112
daysum	112
dayvar	112
dayvar1	112

delcode	40
delete	37
delgridcell	45
delmulti	39
delname	40
delparam	40
detrend	139
diff	22
diffn	22
distgrid	34
div	79
divc	78
divdpm	84
divdpy	84
duplicate	28
dv2ps	168
dv2uv	168
dv2uvl	168

## E

enlarge	69
ensavg	92
ensbrs	95
enscrps	95
ensmax	92
ensmean	92
ensmin	92
enspctl	92
ensrange	92
ensrkhistspace	94
ensrkhisttime	94
ensroc	94
ensstd	92
ensstd1	92
enssum	92
ensvar	92
ensvar1	92
eof	142
eof3d	142
eofcoeff	144
eofspatial	142
eoftime	142
eq	51
eqc	52
exp	77
expr	74
exprf	74

## F

fdns	193
------	-----

fldavg	97
fldcor	136
fldcovar	137
fldmax	97
fldmean	97
fldmin	97
fldpctl	97
fldrange	97
fldstd	97
fldstd1	97
fldsum	97
fldvar	97
fldvar1	97
for	187

## G

ge	51
gec	52
genbic	147
genbil	146
gencon	152
gencon2	154
gendis	149
genlaf	156
genlevelbounds	63
gennn	148
genycon	150
gmtcells	176
gmtxyz	176
gp2sp	167
gp2spl	167
gradsdes	180
gridarea	184
gridboxavg	102
gridboxmax	102
gridboxmean	102
gridboxmin	102
gridboxrange	102
gridboxstd	102
gridboxstd1	102
gridboxsum	102
gridboxvar	102
gridboxvar1	102
griddes	25
gridweights	184
gt	51
gtc	52

## H

highpass	183
histcount	192
histfreq	192
histmean	192
histsum	192
houravg	110
hourmax	110
hourmean	110
hourmin	110

hourpctl	111
hourrange	110
hourstd	110
hourstd1	110
hoursum	110
hourvar	110
hourvar1	110
hurr	194

## I

ifnotthen	47
ifnotthenc	48
ifthen	47
ifthenc	48
ifthenelse	47
import_amsr	172
import_binary	170
import_cmsaf	171
info	20
infor	20
input	173
inputtext	173
inputsrv	173
int	77
intlevel	161
intlevel3d	163
intlevelx3d	163
intntime	164
inttime	164
intyear	165
invertlat	64
invertlev	64

## L

le	51
lec	52
ln	77
log10	77
lowpass	183
lt	51
ltc	52

## M

map	20
maskindexbox	67
masklonlatbox	67
maskregion	66
mastrfu	190
max	79
meravg	100
merge	29
mergegrid	28
mergetime	29
mermax	100
mermean	100
mermin	100
merpctl	100
merrange	100

merstd	100
merstd1	100
mersum	100
mervar	100
mervar1	100
min	79
ml2hl	160
ml2pl	160
monadd	80
monavg	114
monddiv	80
monmax	114
monmean	114
monmin	114
monmul	80
monpctl	115
monrange	114
monstd	114
monstd1	114
monsub	80
monsum	114
monvar	114
monvar1	114
mul	79
mulc	78
muldpm	84
muldpv	84

## N

ndate	23
ne	51
nec	52
ngridpoints	23
ngrids	23
nint	77
nlevel	23
nmon	23
npar	23
ntime	23
nyear	23

## O

output	174
outputtext	174
outputf	174
outputint	174
outputsrv	174
outputtab	175

## P

partab	25
pow	77
projuvLatLon	189

## R

random	187
reci	77
reducegrid	49

regres	139
remap	157
remapbic	147
remapbil	146
remapcon	152
remapcon2	154
remapdis	149
remapeta	158
remaplaf	156
remapnn	148
remapycon	150
replace	28
rhopot	191
rotuvb	190
rotuvNorth	189
runavg	106
runmax	106
runmean	106
runmin	106
runpctl	107
runrange	106
runstd	106
runstd1	106
runsum	106
runvar	106
runvar1	106

## S

samplegrid	45
sealevelpressure	190
seasavg	119
seasmax	119
seasmean	119
seasmin	119
seaspctl	120
seasrange	119
seasstd	119
seasstd1	119
seassum	119
seasvar	119
seasvar1	119
selcode	40
seldate	42
selday	42
select	37
selgrid	40
selgridcell	45
selhour	42
selindexbox	44
sellevel	40
sellevidx	40
sellonlatbox	44
selltype	40
selmonth	42
selmulti	39
selname	40
selparam	40
selseason	42



selsmon	42
selstdname	40
seltabnum	40
selttime	42
seltimestep	42
selyear	42
selzaxis	40
selzaxisname	40
setattribute	55
setcalendar	59
setcindexbox	68
setclonlatbox	68
setcode	58
setcodetab	58
setctomiss	70
setdate	59
setday	59
setgrid	62
setgridarea	62
setgridtype	62
sethalo	192
setlevel	58
setltype	58
setmisstoc	70
setmisstodis	70
setmisstonn	70
setmissval	70
setmon	59
setname	58
setparam	58
setpartabn	56
setpartabp	56
setreftime	59
setrtoc	186
setrtoc2	186
setrtomiss	70
settaxis	59
settbounds	59
settime	59
settunits	59
setunit	58
setvals	186
setvrange	70
setyear	59
setzaxis	63
shifttime	59
shiftx	65
shifty	65
showcode	24
showdate	24
showformat	24
showlevel	24
showltype	24
showmon	24
showname	24
showstdname	24
showtime	24
showtimestamp	24

showyear	24
sin	77
sinfo	21
sinfon	21
smooth	185
smooth9	185
sp2gp	167
sp2gpl	167
sp2sp	167
splitcode	30
splitday	32
splitgrid	30
splithour	32
splitlevel	30
splitmon	32
splitname	30
splitparam	30
splitseas	32
splitsel	33
splittabnum	30
splityear	32
splityearmon	32
splitzaxis	30
sqr	77
sqrt	77
stdatm	187
strbre	194
strgal	194
strwin	193
sub	79
subc	78
subtrend	140

## T

tan	77
timavg	108
timcor	136
timcovar	137
timcumsum	91
timmax	108
timmean	108
timmin	108
timpctl	109
timrange	108
timselavg	104
timselmax	104
timselmean	104
timselmin	104
timselpctl	105
timselrange	104
timselstd	104
timselstd1	104
timselsum	104
timselvar	104
timselvar1	104
timsort	187
timstd	108
timstd1	108

timsum .....	108
timvar .....	108
timvar1 .....	108
topo .....	187
trend .....	140

## U

uv2dv .....	168
uv2dvl .....	168
uvDestag .....	189

## V

vct .....	25
vertavg .....	103
vertmax .....	103
vertmean .....	103
vertmin .....	103
vertrange .....	103
vertstd .....	103
vertstd1 .....	103
vertsum .....	103
vertvar .....	103
vertvar1 .....	103

## W

wct .....	193
-----------	-----

## Y

ydayadd .....	82
ydayavg .....	123
ydaydiv .....	82
ydaymax .....	123
ydaymean .....	123
ydaymin .....	123
ydaymul .....	82
ydaypctl .....	125
ydayrange .....	123
ydaystd .....	123
ydaystd1 .....	123
ydaysub .....	82
ydaysum .....	123
ydayvar .....	123
ydayvar1 .....	123
ydrunavg .....	132
ydrunmax .....	132
ydrunmean .....	132
ydrunmin .....	132
ydrunpctl .....	134
ydrunstd .....	132
ydrunstd1 .....	132
ydrunsum .....	132
ydrunvar .....	132
ydrunvar1 .....	132
yearavg .....	117
yearmax .....	117
yearmean .....	117
yearmin .....	117
yearmonmean .....	116

yearpctl .....	118
yearrange .....	117
yearstd .....	117
yearstd1 .....	117
yearsum .....	117
yearvar .....	117
yearvar1 .....	117
houradd .....	81
houravg .....	121
hourdiv .....	81
hourmax .....	121
hourmean .....	121
hourmin .....	121
hourmul .....	81
hourrange .....	121
hourstd .....	121
hourstd1 .....	121
hoursub .....	81
hoursum .....	121
hourvar .....	121
hourvar1 .....	121
ymonadd .....	83
ymonavg .....	126
ymondiv .....	83
ymonmax .....	126
ymonmean .....	126
ymonmin .....	126
ymonmul .....	83
ymonpctl .....	128
ymonrange .....	126
ymonstd .....	126
ymonstd1 .....	126
ymonsub .....	83
ymonsum .....	126
ymonvar .....	126
ymonvar1 .....	126
yseasadd .....	84
yseasavg .....	129
yseasdiv .....	84
yseasmax .....	129
yseasmean .....	129
yseasmin .....	129
yseasmul .....	84
yseaspctl .....	131
yseasrange .....	129
yseasstd .....	129
yseasstd1 .....	129
yseassub .....	84
yseassum .....	129
yseasvar .....	129
yseasvar1 .....	129

## Z

zaxisdes .....	25
zonavg .....	99
zonmax .....	99
zonmean .....	99
zonmin .....	99

---

zonpctl .....	<a href="#">99</a>
zonrange .....	<a href="#">99</a>
zonstd .....	<a href="#">99</a>
zonstd1 .....	<a href="#">99</a>
zonsum .....	<a href="#">99</a>
zonvar .....	<a href="#">99</a>
zonvar1 .....	<a href="#">99</a>