NCO User's Guide

A suite of netCDF operators
Edition 2.8.1, for NCO Version 2.8.1
August 2003

by Charlie Zender
Department of Earth System Science
University of California at Irvine

This is the first edition of the *NCO User's Guide*,
and is consistent with version 2 of '`texinfo.tex`'.

Published by Charlie Zender
Department of Earth System Science
University of California at Irvine
Irvine, CA 92697-3100 USA

The original author of this software, Charlie Zender, wants to improve it with the help of your suggestions, improvements, bug-reports, and patches.

Charlie Zender <zender at uci dot edu>
Department of Earth System Science
University of California at Irvine
Irvine, CA 92697-3100

Foreword

NCO is the result of software needs that arose while I worked on projects funded by NCAR, NASA, and ARM. Thinking they might prove useful as tools or templates to others, it is my pleasure to provide them freely to the scientific community. Many users (most of whom I have never met) have encouraged the development of NCO. Thanks espcially to Jan Polcher, Keith Lindsay, Arlindo da Silva, John Sheldon, and William Weibel for stimulating suggestions and correspondence. Your encouragment motivated me to complete the *NCO User's Guide*. So if you like NCO, send me a note! I should mention that NCO is not connected to or officially endorsed by Unidata, ACD, ASP, CGD, or Nike.

Charlie Zender
May 1997
Boulder, Colorado

Major feature improvements entitle me to write another Foreword. In the last five years a lot of work has been done refining NCO. NCO is now an honest-to-goodness open source project. It appears to be much healthier for it. The list of illustrious institutions which do not endorse NCO continues to grow, and now includes UCI.

Charlie Zender
October 2000
Irvine, California

The most remarkable advances in NCO capabilities in the last few years are due to contributions from the Open Source community. Especially noteworthy are the contributions of Henry Butowsky and Rorik Peterson.

Charlie Zender
January 2003
Irvine, California

Summary

This manual describes NCO, which stands for netCDF Operators. NCO is a suite of programs known as *operators*. Each operator is a standalone, command line program which is executed at the UNIX (or NT) shell-level like, e.g., `ls` or `mkdir`. The operators take netCDF file(s) (or HDF4 files) as input, perform an operation (e.g., averaging or hyperslabbing), and produce a netCDF file as output. The operators are primarily designed to aid manipulation and analysis of data. The examples in this documentation are typical applications of the operators for processing climate model output. This reflects their origin, but the operators are as general as netCDF itself.

1 Introduction

1.1 Availability

The complete NCO source distribution is currently distributed as a *compressed tarfile* from `http://sourceforge.net/projects/nco` and from `http://dust.ess.uci.edu/nco/nco.tar.gz`. The compressed tarfile must be uncompressed and untarred before building NCO. Uncompress the file with '`gunzip nco.tar.gz`'. Extract the source files from the resulting tarfile with '`tar -xvf nco.tar`'. GNU `tar` lets you perform both operations in one step with '`tar -xvzf nco.tar.gz`'.

The documentation for NCO is called the *NCO User's Guide*. The *User's Guide* is available in Postscript, HTML, DVI, TEXinfo, and Info formats. These formats are included in the source distribution in the files '`nco.ps`', '`nco.html`', '`nco.dvi`', '`nco.texi`', and '`nco.info*`', respectively. All the documentation descends from a single source file, '`nco.texi`'[1]. Hence the documentation in every format is very similar. However, some of the complex mathematical expressions needed to describe `ncwa` can only be displayed in the Postscript and DVI formats.

If you want to quickly see what the latest improvements in NCO are (without downloading the entire source distribution), visit the NCO homepage at `http://nco.sourceforge.net`. The HTML version of the *User's Guide* is also available online through the World Wide Web at URL `http://nco.sourceforge.net/nco.html`. To build and use NCO, you must have netCDF installed. The netCDF homepage is `http://www.unidata.ucar.edu/packages/netcdf`.

New NCO releases are announced on the netCDF list and on the `nco-announce` mailing list `http://lists.sourceforge.net/mailman/listinfo/nco-announce`.

1.2 Operating systems compatible with NCO

NCO has been successfully ported and tested and is known to work on the following 32 and 64 bit platforms: IBM AIX 4.x, 5.x, FreeBSD 4.x, GNU/Linux 2.x, LinuxPPC, LinuxAlpha, LinuxSparc64, SGI IRIX 5.x and 6.x, MacOS X 10.x, NEC Super-UX 10.x, DEC OSF, Sun SunOS 4.1.x, Solaris 2.x, CRAY UNICOS 8.x–10.x, all MS Windows. If you port the code to a new operating system, please send me a note and any patches you required.

The major prerequisite for installing NCO on a particular platform is the successful, prior installation of the netCDF library (and, as of 2003, the UDUnits library). Unidata has shown a commitment to maintaining netCDF and UDUnits on all popular UNIX platforms, and is moving towards full support for the Microsoft Windows operating system (OS). Given this, the only difficulty in implementing NCO on a particular platform

[1] To produce these formats, '`nco.texi`' was simply run through the freely available programs `texi2dvi`, `dvips`, `texi2html`, and `makeinfo`. Due to a bug in TEX, the resulting Postscript file, '`nco.ps`', contains the Table of Contents as the final pages. Thus if you print '`nco.ps`', remember to insert the Table of Contents after the cover sheet before you staple the manual.

is standardization of various C and Fortran interface and system calls. NCO code is tested for ANSI compliance by compiling with C compilers including those from GNU (`gcc -std=c99 -pedantic -D_BSD_SOURCE` -Wall)[2], Comeau Computing (`como --c99`), Cray (`cc`), HP/Compaq/DEC (`cc`), IBM (`xlc -c -qlanglvl=extended`), Intel (`icc`), NEC (`cc`), SGI (`cc -LANG:std`), and Sun (`cc`). NCO (all commands and the `libnco` library) and the C++ interface to netCDF (called `libnco_c++`) comply with the ISO C++ standards as implemented by Comeau Computing (`como`), Cray (`CC`), GNU (`g++ -Wall`), HP/Compaq/DEC (`cxx`), IBM (`xlC`), Intel (`icc`), NEC (`c++`), SGI (`CC -LANG:std`), and Sun (`CC -LANG:std`). See `nco/bld/Makefile` and `nco/src/nco_c++/Makefile.old` for more details.

Until recently (and not even yet), ANSI-compliant has meant compliance with the 1989 ISO C-standard, usually called C89 (with minor revisions made in 1994 and 1995). C89 does not allow variable-size arrays nor use of the `%z` format for `printf`. These are nice features of the 1999 ISO C-standard called C99. NCO is C99-compliant where possible and C89-compliant where necessary. Certain branches in the code are required to satisfy the native SGI and SunOS C compilers, which are strictly ANSI C89 compliant, and cannot benefit from C99 features. However, C99 features are fully supported by the GNU, UNICOS, Solaris, and AIX compilers.

The most time-intensive portion of NCO execution is spent in arithmetic operations, e.g., multiplication, averaging, subtraction. Until August, 1999, these operations were performed in Fortran by default. This was a design decision made in late 1994 based on the speed of Fortran-based object code vs. C-based object code. Since 1994 native C compilers have improved their vectorization capabilities and it has become advantageous to replace all Fortran subroutines with C subroutines. Furthermore, this greatly simplifies the task of compiling on nominally unsupported platforms. As of August 1999, NCO is built entirely in C by default. This allows NCO to compile on any machine with an ANSI C compiler. Furthermore, NCO automatically takes advantage of extensions to ANSI C when compiled with the GNU compiler collection, GCC.

As of July 2000 and NCO version 1.2, NCO no longer supports performing arithmetic operations in Fortran. We decided to sacrifice executable speed for code maintainability Since no objective statistics were ever performed to quantify the difference in speed between the Fortran and C code, the performance penalty incurred by this decision is unknown. Supporting Fortran involves maintaining two sets of routines for every arithmetic operation. The `USE_FORTRAN_ARITHMETIC` flag is still retained in the `Makefile`. The file containing the Fortran code, `nco_fortran.F`, has been deprecated but can be resurrected if a volunteer comes forward. If you would like to volunteer to maintain `nco_fortran.F` please contact me.

1.2.1 Compiling NCO for Microsoft Windows OS

NCO has been successfully ported and tested on the Microsoft Windows (98/NT) operating systems. The switches necessary to accomplish this are included in the standard distribution

[2] The `_BSD_SOURCE` token is required on some Linux platforms where `gcc` dislikes the network header files like `netinet/in.h`).

of NCO. Using the freely available Cygwin (formerly gnu-win32) development environment[3], the compilation process is very similar to installing NCO on a UNIX system. The preprocessor token `PVM_ARCH` should be set to `WIN32`. Note that defining `WIN32` has the side effect of disabling Internet features of NCO (see below). Unless you have a Fortran compiler (like `g77` or `f90`) available, no other tokens are required. Users with fast Fortran compilers may wish to activate the Fortran arithmetic routines. To do this, define the preprocessor token `USE_FORTRAN_ARITHMETIC` in the makefile which comes with NCO, '`Makefile`', or in the compilation shell.

The least portable section of the code is the use of standard UNIX and Internet protocols (e.g., `ftp`, `rcp`, `scp`, `getuid`, `gethostname`, and header files '`<arpa/nameser.h>`' and '`<resolv.h>`'). Fortunately, these UNIXy calls are only invoked by the single NCO subroutine which is responsible for retrieving files stored on remote systems (see Section 3.3 [Remote storage], page 20). In order to support NCO on the Microsoft Windows platforms, this single feature was disabled (on Windows OS only). This was required by Cygwin 18.x—newer versions of Cygwin may support these protocols (let me know if this is the case). The NCO operators should behave identically on Windows and UNIX platforms in all other respects.

1.3 Libraries

Like all executables, the NCO operators can be built using dynamic linking. This reduces the size of the executable and can result in significant performance enhancements on multiuser systems. Unfortunately, if your library search path (usually the `LD_LIBRARY_PATH` environment variable) is not set correctly, or if the system libraries have been moved, renamed, or deleted since NCO was installed, it is possible an NCO operator will fail with a message that it cannot find a dynamically loaded (aka *shared object* or '`.so`') library. This usually produces a distinctive error message, such as '`ld.so.1: /usr/local/bin/ncea: fatal: libsunmath.so.1: can't open file: errno=2`'. If you received an error message like this, ask your system administrator to diagnose whether the library is truly missing[4], or whether you simply need to alter your library search path. As a final remedy, you can reinstall NCO with all operators statically linked.

1.4 netCDF 2.x vs. 3.x

netCDF version 2.x was released in 1993. NCO (specifically `ncks`) began with netCDF 2.x in 1994. netCDF 3.0 was released in 1996, and we were eager to reap the performance advantages of the newer netCDF implementation. One netCDF 3.x interface call (`nc_inq_libvers`) was added to NCO in January, 1998, to aid in maintainance and debugging. In March, 2001, the final conversion of NCO to netCDF 3.x was completed (coincidentally on

[3] The Cygwin package is available from
`http://sourceware.redhat.com/cygwin`
Currently, Cygwin 20.x comes with the GNU C/C++/Fortran compilers (`gcc`, `g++`, `g77`). These GNU compilers may be used to build the netCDF distribution itself.

[4] The `ldd` command, if it is available on your system, will tell you where the executable is looking for each dynamically loaded library. Use, e.g., `ldd `which ncea``.

the same day netCDF 3.5 was released). NCO versions 2.0 and higher are built with the `-DNO_NETCDF_2` flag to ensure no netCDF 2.x interface calls are used.

However, the ability to compile NCO with only netCDF 2.x calls is worth maintaining because HDF version 4[5] (available from HDF (`http://hdf.ncsa.uiuc.edu`)) supports only the netCDF 2.x library calls (see `http://hdf.ncsa.uiuc.edu/UG41r3_html/SDS_SD.fm12.html#47784`). Note that there are multiple versions of HDF. Currently HDF version 4.x supports netCDF 2.x and thus NCO version 1.2.x. If NCO version 1.2.x (or earlier) is built with only netCDF 2.x calls then all NCO operators should work with HDF4 files as well as netCDF files[6]. The preprocessor token `NETCDF2_ONLY` exists in NCO version 1.2.x to eliminate all netCDF 3.x calls. Only versions of NCO numbered 1.2.x and earlier have this capability. The NCO 1.2.x branch will be maintained with bugfixes only (no new features) until HDF begins to fully support the netCDF 3.x interface (which is employed by NCO 2.x). If, at compilation time, `NETCDF2_ONLY` is defined, then NCO version 1.2.x will not use any netCDF 3.x calls and, if linked properly, the resulting NCO operators will work with HDF4 files. The '`Makefile`' supplied with NCO 1.2.x has been written to simplify building in this HDF capability. When NCO is built with `make HDF4=Y`, the '`Makefile`' will set all required preprocessor flags and library links to build with the HDF4 libraries (which are assumed to reside under `/usr/local/hdf4`, edit the '`Makefile`' to suit your installation).

HDF version 5.x became available in 1999, but did not support netCDF (or, for that matter, Fortran) as of December 1999. By early 2001, HDF version 5.x did support Fortran90. However, support for netCDF 3.x in HDF 5.x is incomplete. Much of the HDF5-netCDF3 interface is complete, however, and it may be separately downloaded from the HDF5-netCDF (`http://hdf.ncsa.uiuc.edu/HDF5/papers/netcdfh5.html`) website. Now that NCO uses only netCDF 3.x system calls we are eager for HDF5 to complete their netCDF 3.x support.

1.5 Help and Bug reports

We generally receive three categories of mail from user's: requests for help, bug reports, and requests for new features. Notes saying the equivalent of "Hey, NCO continues to work great and it saves me more time everyday than it took to write this note" are a distant fourth. There is a different protocol for each type of request. Our request is that you communicate with the project via NCO Project Forums. Before posting to the NCO forums described below, you might first register (`https://sourceforge.net/account/register.php`) your name and email address with SourceForge.org or else all of your postings will be attributed to "nobody". Once registered you may choose to "monitor" any forum and to receive (or not) email when there are any postings.

If you would like NCO to include a new feature, first check to see if that feature is already on the TODO (`file:./TODO`) list. If it is, please consider implementing that feature yourself

[5] The Hierarchical Data Format, or HDF, is another self-describing data format similar to, but more elaborate than, netCDF.

[6] One must link the NCO code to the HDF4 MFHDF library instead of the usual netCDF library. However, the MFHDF library only supports netCDF 2.x calls. Thus I will try to keep this capability in NCO as long as it is not too much trouble.

and sending us the patch! If the feature is not yet on the list then send a note to the NCO Discussion forum (`http://sourceforge.net/forum/forum.php?forum_id=9829`).

Please read the manual before reporting a bug or posting a request for help. Sending questions whose answers are not in the manual is the best way to motivate us to write more documentation. We would also like to accentuate the contrapositive of this statement. If you think you have found a real bug *the most helpful thing you can do is simplify the problem to a manageable size and report it*. The first thing to do is to make sure you are running the latest publicly released version of NCO.

Once you have read the manual, if you are still unable to get NCO to perform a documented function, write help request. Follow the same procedure as described below for reporting bugs (after all, it might be a bug). That is, describe what you are trying to do, and include the complete commands (with '`-D 5`'), error messages, and version of NCO. Post your help request to the NCO Help forum (`http://sourceforge.net/forum/forum.php?forum_id=9830`).

If you think you are using the right command, but NCO is misbehaving, then you might have found a bug. A core dump, sementation violation, or incorrect numerical answers is always considered a high priority bug. How do you simplify a problem that may be revealing a bug? Cut out extraneous variables, dimensions, and metadata from the offending files and re-run the command until it no longer breaks. Then back up one step and report the problem. Usually the file(s) will be very small, i.e., one variable with one or two small dimensions ought to suffice. Include in the report your run-time environment, the exact error messages (and run the operator with '`-D 5`' to increase the verbosity of the debugging output), and a copy, or the publically accessible location, of the file(s). Post the bug report to the NCO Project buglist (`http://sourceforge.net/bugs/?group_id=3331`).

2 Operator Strategies

2.1 NCO operator philosophy

The main design goal has been to produce operators that can be invoked from the command line to perform useful operations on netCDF files. Many scientists work with models and observations which produce too much data to analyze in tabular format. Thus, it is often natural to reduce and massage this raw or primary level data into summary, or second level data, e.g., temporal or spatial averages. These second level data may become the inputs to graphical and statistical packages, and are often more suitable for archival and dissemination to the scientific community. NCO performs a suite of operations useful in manipulating data from the primary to the second level state. Higher level interpretive languages (e.g., IDL, Yorick, Matlab, NCL, Perl, Python), and lower level compiled languages (e.g., C, Fortran) can always perform any task performed by NCO, but often with more overhead. NCO, on the other hand, is limited to a much smaller set of arithmetic and metadata operations than these full blown languages.

Another goal has been to implement enough command line switches so that frequently used sequences of these operators can be executed from a shell script or batch file. Finally, NCO was written to consume the absolute minimum amount of system memory required to perform a given job. The arithmetic operators are extremely efficient; their exact memory usage is detailed in Section 2.9 [Memory usage], page 15.

2.2 Climate model paradigm

NCO was developed at NCAR to aid analysis and manipulation of datasets produced by General Circulation Models (GCMs). Datasets produced by GCMs share many features with all gridded scientific datasets and so provide a useful paradigm for the explication of the NCO operator set. Examples in this manual use a GCM paradigm because latitude, longitude, time, temperature and other fields related to our natural environment are as easy to visualize for the layman as the expert.

2.3 Temporary output files

NCO operators are designed to be reasonably fault tolerant, so that if there is a system failure or the user aborts the operation (e.g., with `C-c`), then no data are lost. The user-specified *output-file* is only created upon successful completion of the operation[1]. This is accomplished by performing all operations in a temporary copy of *output-file*. The name of the temporary output file is constructed by appending `.pid<process ID>.<operator name>.tmp` to the user-specified *output-file* name. When the operator completes its task with no fatal errors, the temporary output file is moved to the user-specified *output-file*.

[1] The `ncrename` operator is an exception to this rule. See Section 4.10 [ncrename netCDF Renamer], page 70.

Note the construction of a temporary output file uses more disk space than just overwriting existing files "in place" (because there may be two copies of the same file on disk until the NCO operation successfully concludes and the temporary output file overwrites the existing *output-file*). Also, note this feature increases the execution time of the operator by approximately the time it takes to copy the *output-file*. Finally, note this feature allows the *output-file* to be the same as the *input-file* without any danger of "overlap".

Other safeguards exist to protect the user from inadvertently overwriting data. If the *output-file* specified for a command is a pre-existing file, then the operator will prompt the user whether to overwrite (erase) the existing *output-file*, attempt to append to it, or abort the operation. However, in processing large amounts of data, too many interactive questions can be a curse to productivity. Therefore NCO also implements two ways to override its own safety features, the '`-O`' and '`-A`' switches. Specifying '`-O`' tells the operator to overwrite any existing *output-file* without prompting the user interactively. Specifying '`-A`' tells the operator to attempt to append to any existing *output-file* without prompting the user interactively. These switches are useful in batch environments because they suppress interactive keyboard input.

2.4 Appending variables to a file

A frequently useful operation is adding variables from one file to another. This is referred to as *appending*, although some prefer the terminology *merging*[2] or *pasting*. Appending is often confused with what NCO calls *concatenation*. In NCO, concatenation refers to splicing a variable along the record dimension. Appending, on the other hand, refers to adding variables from one file to another[3]. In this sense, `ncks` can append variables from one file to another file. This capability is invoked by naming two files on the command line, *input-file* and *output-file*. When *output-file* already exists, the user is prompted whether to *overwrite*, *append/replace*, or *exit* from the command. Selecting *overwrite* tells the operator to erase the existing *output-file* and replace it with the results of the operation. Selecting *exit* causes the operator to exit—the *output-file* will not be touched in this case. Selecting *append/replace* causes the operator to attempt to place the results of the operation in the existing *output-file*, See Section 4.7 [ncks netCDF Kitchen Sink], page 61.

2.5 Addition Subtraction Division Multiplication and Interpolation

Users comfortable with NCO semantics may find it easier to perform some simple mathematical operations in NCO rather than higher level languages. `ncbo` (see Section 4.3 [ncbo netCDF Binary Operator], page 52) does file addition, subtraction, multiplication, division, and broadcasting. `ncflint` (see Section 4.6 [ncflint netCDF File Interpolator], page 59) does

[2] The terminology *merging* is reserved for an (unwritten) operator which replaces hyperslabs of a variable in one file with hyperslabs of the same variable from another file

[3] Yes, the terminology is confusing. By all means mail me if you think of a better nomenclature. Should NCO use *paste* instead of *append*?

file addition, subtraction, multiplication and interpolation. Sequences of these commands can accomplish simple but powerful operations from the command line.

2.6 Averagers vs. Concatenators

The most frequently used operators of NCO are probably the averagers and concatenators. Because there are so many permutations of averaging (e.g., across files, within a file, over the record dimension, over other dimensions, with or without weights and masks) and of concatenating (across files, along the record dimension, along other dimensions), there are currently no fewer than five operators which tackle these two purposes: `ncra`, `ncea`, `ncwa`, `ncrcat`, and `ncecat`. These operators do share many capabilities[4], but each has its unique specialty. Two of these operators, `ncrcat` and `ncecat`, are for concatenating hyperslabs across files. The other two operators, `ncra` and `ncea`, are for averaging hyperslabs across files[5]. First, let's describe the concatenators, then the averagers.

2.6.1 Concatenators `ncrcat` and `ncecat`

Joining independent files together along a record coordinate is called *concatenation*. `ncrcat` is designed for concatenating record variables, while `ncecat` is designed for concatenating fixed length variables. Consider five files, '`85.nc`', '`86.nc`', ... '`89.nc`' each containing a year's worth of data. Say you wish to create from them a single file, '`8589.nc`' containing all the data, i.e., spanning all five years. If the annual files make use of the same record variable, then `ncrcat` will do the job nicely with, e.g., `ncrcat 8?.nc 8589.nc`. The number of records in the input files is arbitrary and can vary from file to file. See Section 4.9 [ncrcat netCDF Record Concatenator], page 68, for a complete description of `ncrcat`.

However, suppose the annual files have no record variable, and thus their data are all fixed length. For example, the files may not be conceptually sequential, but rather members of the same group, or *ensemble*. Members of an ensemble may have no reason to contain a record dimension. `ncecat` will create a new record dimension (named *record* by default) with which to glue together the individual files into the single ensemble file. If `ncecat` is used on files which contain an existing record dimension, that record dimension will be converted into a fixed length dimension of the same name and a new record dimension will be created. Consider five realizations, '`85a.nc`', '`85b.nc`', ... '`85e.nc`' of 1985 predictions from the same climate model. Then `ncecat 85?.nc 85_ens.nc` glues the individual realizations together into the single file, '`85_ens.nc`'. If an input variable was dimensioned [`lat`,`lon`], it will have dimensions [`record`,`lat`,`lon`] in the output file. A restriction of `ncecat` is that the hyperslabs of the processed variables must be the same from file to file. Normally this means all the input files are the same size, and contain data on different realizations of the

---

[4] Currently `ncea` and `ncrcat` are symbolically linked to the `ncra` executable, which behaves slightly differently based on its invocation name (i.e., '`argv[0]`'). These three operators share the same source code, but merely have different inner loops.

[5] The third averaging operator, `ncwa`, is the most sophisticated averager in NCO. However, `ncwa` is in a different class than `ncra` and `ncea` because it can only operate on a single file per invocation (as opposed to multiple files). On that single file, however, `ncwa` provides a richer set of averaging options—including weighting, masking, and broadcasting.

same variables. See Section 4.5 [ncecat netCDF Ensemble Concatenator], page 58, for a complete description of `ncecat`.

Note that `ncrcat` cannot concatenate fixed-length variables, whereas `ncecat` can concatenate both fixed-length and record variables. To conserve system memory, use `ncrcat` rather than `ncecat` when concatenating record variables.

2.6.2 Averagers `ncea`, `ncra`, and `ncwa`

The differences between the averagers `ncra` and `ncea` are analogous to the differences between the concatenators. `ncra` is designed for averaging record variables from at least one file, while `ncea` is designed for averaging fixed length variables from multiple files. `ncra` performs a simple arithmetic average over the record dimension of all the input files, with each record having an equal weight in the average. `ncea` performs a simple arithmetic average of all the input files, with each file having an equal weight in the average. Note that `ncra` cannot average fixed-length variables, but `ncea` can average both fixed-length and record variables. To conserve system memory, use `ncra` rather than `ncea` where possible (e.g., if each *input-file* is one record long). The file output from `ncea` will have the same dimensions (meaning dimension names as well as sizes) as the input hyperslabs (see Section 4.4 [ncea netCDF Ensemble Averager], page 56, for a complete description of `ncea`). The file output from `ncra` will have the same dimensions as the input hyperslabs except for the record dimension, which will have a size of 1 (see Section 4.8 [ncra netCDF Record Averager], page 66, for a complete description of `ncra`).

2.6.3 Interpolator `ncflint`

`ncflint` can interpolate data between or two files. Since no other operators have this ability, the description of interpolation is given fully on the `ncflint` reference page (see Section 4.6 [ncflint netCDF File Interpolator], page 59). Note that this capability also allows `ncflint` to linearly rescale any data in a netCDF file, e.g., to convert between differing units.

2.7 Working with large numbers of input files

Occasionally one desires to digest (i.e., concatenate or average) hundreds or thousands of input files. One brave user, for example, recently created a five year time-series of satellite observations by using `ncecat` to join thousands of daily data files together. Unfotunately, data archives (e.g., NASA EOSDIS) are unlikely to distribute netCDF files conveniently named in a format the '`-n` *loop*' switch (which automatically generates arbitrary numbers of input filenames) understands. If there is not a simple, arithmetic pattern to the input filenames (e.g., '`h00001.nc`', '`h00002.nc`', ... '`h90210.nc`') then the '`-n` *loop*' switch is useless. Moreover, when the input files are so numerous that the input filenames are too lengthy (when strung together as a single argument) to be passed by the calling shell to the NCO operator[6], then the following strategy has proven useful to specify the input filenames

[6] The exact length which exceeds the operating system internal limit for command line lengths varies from OS to OS and from shell to shell. GNU `bash` may not have any arbitrary fixed limits to the size of

to NCO. Write a script that creates symbolic links between the irregular input filenames and a set of regular, arithmetic filenames that '-n *loop*' switch understands. The NCO operator will then succeed at automatically generating the filnames with the '-n *loop*' option (which circumvents any OS and shell limits on command line size). You can remove the symbolic links once the operator completes its task.

2.8 Working with large files

Large files are those files that are comparable in size to the amount of memory (RAM) in your computer. Many users of NCO work with files larger than 100 MB. Files this large not only push the current edge of storage technology, they present special problems for programs which attempt to access the entire file at once, such as `ncea`, and `ncecat`. If you need to work with a 300 MB file on a machine with only 32 MB of memory then you will need large amounts of swap space (virtual memory on disk) and NCO will work slowly, or else NCO will fail. There is no easy solution for this and the best strategy is to work on a machine with massive amounts of memory and swap space. That is, if your local machine has problems working with large files, try running NCO from a more powerful machine, such as a network server. Certain machine architectures, e.g., Cray UNICOS, have special commands which allow one to increase the amount of interactive memory. If you get a core dump on a Cray system (e.g., '`Error exit (core dumped)`'), try increasing the available memory by using the `ilimit` command.

The speed of the NCO operators also depends on file size. When processing large files the operators may appear to hang, or do nothing, for large periods of time. In order to see what the operator is actually doing, it is useful to activate a more verbose output mode. This is accomplished by supplying a number greater than 0 to the '-D *debug-level*' (or '`--debug-level`', or '`--dbg_lvl`') switch. When the *debug-level* is nonzero, the operators report their current status to the terminal through the *stderr* facility. Using '`-D`' does not slow the operators down. Choose a *debug-level* between 1 and 3 for most situations, e.g., `ncea -D 2 85.nc 86.nc 8586.nc`. A full description of how to estimate the actual amount of memory the multi-file NCO operators consume is given in Section 2.9 [Memory usage], page 15.

2.9 Approximate NCO memory requirements

The multi-file operators currently comprise the record operators, `ncra` and `ncrcat`, and the ensemble operators, `ncea` and `ncecat`. The record operators require *much less* memory than the ensemble operators. This is because the record operators are designed to operate on a single record of a file at a time, while the ensemble operators must retrieve an entire variable at a time into memory. Let MS be the peak sustained memory demand of an operator, FT be the memory required to store the entire contents of all the variables to be processed in an input file, FR be the memory required to store the entire contents of a single

---

command line arguments. Many OSs cannot handle command line arguments longer than a few thousand characters. When this occurs, the ANSI C-standard `argc-argv` method of passing arguments from the calling shell to a C-program (i.e., an NCO operator) breaks down.

record of each of the variables to be processed in an input file, VR be the memory required to store a single record of the largest record variable to be processed in an input file, VT be the memory required to store the largest variable to be processed in an input file, VI be the memory required to store the largest variable which is not processed, but is copied from the initial file to the output file. All operators require $MI = VI$ during the initial copying of variables from the first input file to the output file. This is the *initial* (and transient) memory demand. The *sustained* memory demand is that memory required by the operators during the processing (i.e., averaging, concatenation) phase which lasts until all the input files have been processed. The operators have the following memory requirements: `ncrcat` requires $MS <= VR$. `ncecat` requires $MS <= VT$. `ncra` requires $MS = 2FR+VR$. `ncea` requires $MS = 2FT + VT$. `ncbo` requires $MS <= 2VT$. `ncflint` requires $MS <= 2VT$. Note that only variables which are processed, i.e., averaged or concatenated, contribute to MS. Memory is never allocated to hold variables which do not appear in the output file (see Section 3.5 [Variable subsetting], page 22).

2.10 Performance limitations of the operators

1. No buffering of data is performed during `ncvarget` and `ncvarput` operations. Hyperslabs too large too hold in core memory will suffer substantial performance penalties because of this.
2. Since coordinate variables are assumed to be monotonic, the search for bracketing the user-specified limits should employ a quicker algorithm, like bisection, than the two-sided incremental search currently implemented.
3. *C_format*, *FORTRAN_format*, *signedness*, *scale_format* and *add_offset* attributes are ignored by `ncks` when printing variables to screen.
4. Some random access operations on large files on certain architectures (e.g., 400 MB on UNICOS) are *much* slower with these operators than with similar operations performed using languages that bypass the netCDF interface (e.g., Yorick). The cause for this is not understood at present.

3 Features common to most operators

Many features have been implemented in more than one operator and are described here for brevity. The description of each feature is preceded by a box listing the operators for which the feature is implemented. Command line switches for a given feature are consistent across all operators wherever possible. If no "key switches" are listed for a feature, then that particular feature is automatic and cannot be controlled by the user.

3.1 Command line options

Availability: All operators
Short options: All
Long options: All

NCO achieves flexibility by using *command line options*. These options are implemented in all traditional UNIX commands as single letter *switches*, e.g., '`ls -l`'. For many years NCO used only single letter option names. In late 2002, we implemented GNU/POSIX extended or long option names for all options. This was done in a backward compatible way such that the full functionality of NCO is still available through the familiar single letter options. In the future, however, some features of NCO may require the use of long options, simply because we have nearly run out of single letter options. More importantly, mnemonics for single letter options are often non-intuitive so that long options provide a more natural way of expressing intent.

Extended options are implemented using the system-supplied '`getopt.h`' header file, if possible. This provides the `getopt_long` function to NCO[1].

The syntax of *short options* (single letter options) is `-key value` (dash-key-space-value). Here, *key* is the single letter option name, e.g., '`-D 2`'.

The syntax of *long options* (multi-letter options) is `--long_name value` (dash-dash-key-space-value), e.g., '`--dbg_lvl 2`' or `--long_name=value` (dash-dash-key-equal-value), e.g., '`--dbg_lvl=2`'. Thus the following are all valid for the '`-D`' (short version) or '`--dbg_lvl`' (long version) command line option.

```
ncks -D 3 in.nc        # Short option
ncks --dbg_lvl=3 in.nc # Long option, preferred form
ncks --dbg_lvl 3 in.nc # Long option, alternate form
```

The last example is preferred for two reasons. First, '`--dbg_lvl`' is more specific and less ambiguous than '`-D`'. The long option form makes scripts more self documenting and less error prone. Often long options are named after the source code variable whose value they carry. Second, the equals sign `=` joins the key (i.e., *long_name*) to the value in an uninterruptible text block. Experience shows that users are less likely to mis-parse commands when restricted to this form.

[1] If a `getopt_long` function cannot be found on the system, NCO will use the `getopt_long` from the `my_getopt` package by Benjamin Sittler `bsittler@iname.com`. This is BSD-licensed software available from `http://www.geocities.com/ResearchTriangle/Node/9405/#my_getopt`.

GNU implements a superset of the POSIX standard which allows any unambiguous truncation of a valid option to be used.

```
ncks -D 3 in.nc        # Short option
ncks --dbg_lvl=3 in.nc # Long option, full form
ncks --dbg=3 in.nc     # Long option, unambiguous truncation
ncks --db=3 in.nc      # Long option, unambiguous truncation
ncks --d=3 in.nc       # Long option, ambiguous truncation
```

The first four examples are equivalent and will work as expected. The final example will exit with an error since `ncks` cannot disambiguate whether '`--d`' is intended as a truncation of '`--dbg_lvl`', of '`--dimension`', or of some other long option.

NCO provides many long options for common switches. For example, the debugging level may be set in all operators with any of the switches '`-D`', '`--debug-level`', or '`--dbg_lvl`'. This flexibility allows users to choose their favorite mnemonic. For some, it will be '`--debug`' (an unambiguous truncation of '`--debug-level`', and other will prefer '`--dbg`'. Interactive users usually prefer the minimal amount of typing, i.e., '`-D`'. We recommend that scripts which are re-usable employ some form of the long options for future maintainability.

This manual generally uses the short option syntax. This is for historical reasons and to conserve space. The remainder of this manual specifies the full *long_name* of each option. Users are expected to pick the unambiguous truncation of each option name that most suits their taste.

3.2 Specifying input files

Availability: All operators
Short options: '`-n`', '`-p`'
Long options: '`--nintap`', '`--pth`', '`--path`'

It is important that the user be able to specify multiple input files without tediously typing in each by its full name. There are four different ways of specifying input files to NCO: explicitly typing each, using UNIX shell wildcards, and using the NCO '`-n`' and '`-p`' switches (or their long option equivalents, '`--nintap`' or '`--pth`' and '`--path`', respectively). To illustrate these methods, consider the simple problem of using `ncra` to average five input files, '`85.nc`', '`86.nc`', ... '`89.nc`', and store the results in '`8589.nc`'. Here are the four methods in order. They produce identical answers.

```
ncra 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra 8[56789].nc 8589.nc
ncra -p input-path 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra -n 5,2,1 85.nc 8589.nc
```

The first method (explicitly specifying all filenames) works by brute force. The second method relies on the operating system shell to *glob* (expand) the *regular expression* `8[56789].nc`. The shell passes valid filenames which match the expansion to `ncra`. The third method uses the '`-p input-path`' argument to specify the directory where all the input files reside. NCO prepends *input-path* (e.g., '`/data/usrname/model`') to all *input-files* (but not to *output-file*). Thus, using '`-p`', the path to any number of input files need only be specified once. Note *input-path* need not end with '`/`'; the '`/`' is automatically generated if necessary.

The last method passes (with '`-n`') syntax concisely describing the entire set of filenames[2]. This option is only available with the *multi-file operators*: `ncra`, `ncrcat`, `ncea`, and `ncecat`. By definition, multi-file operators are able to process an arbitrary number of *input-files*. This option is very useful for abbreviating lists of filenames representable as *alphanumeric_prefix*+*numeric_suffix*+'`.`'+*filetype* where *alphanumeric_prefix* is a string of arbitrary length and composition, *numeric_suffix* is a fixed width field of digits, and *filetype* is a standard filetype indicator. For example, in the file '`ccm3_h0001.nc`', we have *alphanumeric_prefix* = '`ccm3_h`', *numeric_suffix* = '`0001`', and *filetype* = '`nc`'.

NCO is able to decode lists of such filenames encoded using the '`-n`' option. The simpler (3-argument) '`-n`' usage takes the form `-n file_number,digit_number,numeric_increment` where *file_number* is the number of files, *digit_number* is the fixed number of numeric digits comprising the *numeric_suffix*, and *numeric_increment* is the constant, integer-valued difference between the *numeric_suffix* of any two consecutive files. The value of *alphanumeric_prefix* is taken from the input file, which serves as a template for decoding the filenames. In the example above, the encoding `-n 5,2,1` along with the input file name '`85.nc`' tells NCO to construct five (5) filenames identical to the template '`85.nc`' except that the final two (2) digits are a numeric suffix to be incremented by one (1) for each successive file. Currently *filetype* may be either be empty, '`nc`', '`cdf`', '`hdf`', or '`hd5`'. If present, these *filetype* suffixes (and the preceding '`.`') are ignored by NCO as it uses the '`-n`' arguments to locate, evaluate, and compute the *numeric_suffix* component of filenames.

Recently the '`-n`' option has been extended to allow convenient specification of filenames with "circular" characteristics. This means it is now possible for NCO to automatically generate filenames which increment regularly until a specified maximum value, and then wrap back to begin again at a specified minimum value. The corresponding '`-n`' usage becomes more complex, taking one or two additional arguments for a total of four or five, respectively: `-n file_number,digit_number,numeric_increment[,numeric_max[,numeric_min]]` where *numeric_max*, if present, is the maximum integer-value of *numeric_suffix* and *numeric_min*, if present, is the minimum integer-value of *numeric_suffix*. Consider, for example, the problem of specifying non-consecutive input files where the filename suffixes end with the month index. In climate modeling it is common to create summertime and wintertime averages which contain the averages of the months June–July–August, and December–January–February, respectively:

```
ncra -n 3,2,1 85_06.nc 85_0608.nc
ncra -n 3,2,1,12 85_12.nc 85_1202.nc
ncra -n 3,2,1,12,1 85_12.nc 85_1202.nc
```

The first example shows that three arguments to the '`-n`' option suffice to specify consecutive months (`06, 07, 08`) which do not "wrap" back to a minimum value. The second example shows how to use the optional fourth and fifth elements of the '`-n`' option to specify a wrap value to NCO. The fourth argument to '`-n`', if present, specifies the maximum integer value of *numeric_suffix*. In this case the maximum value is 12, and will be formatted as '`12`'

[2] The '`-n`' option is a backward compatible superset of the `NINTAP` option from the NCAR CCM Processor.

in the filename string. The fifth argument to '-n', if present, specifies the minimum integer value of *numeric_suffix*. The default minimum filename suffix is 1, which is formatted as '01' in this case. Thus the second and third examples have the same effect, that is, they automatically generate, in order, the filenames '85_12.nc', '85_01.nc', and '85_02.nc' as input to NCO.

3.3 Accessing files stored remotely

Availability: All operators
Short options: '-p', '-l'
Long options: '--pth', '--path', '--lcl', '--local'

All NCO operators can retrieve files from remote sites as well as from the local file system. A remote site can be an anonymous FTP server, a machine on which the user has `rcp` or `scp` privileges, or NCAR's Mass Storage System (MSS). To access a file via an anonymous FTP server, supply the remote file's URL. To access a file using `rcp` or `scp`, specify the Internet address of the remote file. Of course in this case you must have `rcp` or `scp` privileges which allow transparent (no password entry required) access to the remote machine. This means that '~/.rhosts' or '~/ssh/authorized_keys' must be set accordingly on both local and remote machines.

To access a file on NCAR's MSS, specify the full MSS pathname of the remote file. NCO will attempt to detect whether the local machine has direct (synchronous) MSS access. In this case, NCO attempts to use the NCAR `msrcp` command[3], or, failing that, `/usr/local/bin/msread`. Otherwise NCO attempts to retrieve the MSS file through the (asynchronous) Masnet Interface Gateway System (MIGS) using the `nrnet` command.

The following examples show how one might analyze files stored on remote systems.

```
ncks -H -l ./ ftp://dust.ess.uci.edu/pub/zender/nco/in.nc
ncks -H -l ./ dust.ess.uci.edu:/home/zender/nco/in.nc
ncks -H -l ./ /ZENDER/nco/in.nc
ncks -H -l ./ mss:/ZENDER/nco/in.nc
ncks -H -l ./ -p http://www.cdc.noaa.gov/cgi-bin/nph-nc/Datasets/\
ncep.reanalysis.dailyavgs/surface air.sig995.1975.nc
```

The first example will work verbatim on your system if your system is connected to the Internet and is not behind a firewall. The second example will work on your system if you have `rcp` or `scp` access to the machine `dust.ess.uci.edu`. The third example will work from NCAR computers with local access to the `msrcp`, `msread`, or `nrnet` commands. The fourth command will work if your local version of NCO was built with DODS capability (see Section 3.3.1 [DODS], page 21). The above commands can be rewritten using the '-p *input-path*' option as follows:

```
ncks -H -p ftp://dust.ess.uci.edu/pub/zender/nco -l ./ in.nc
ncks -H -p dust.ess.uci.edu:/home/zender/nco -l ./ in.nc
```

[3] The `msrcp` command must be in the user's path and located in one of the following directories: `/usr/local/bin`, `/usr/bin`, `/opt/local/bin`, or `/usr/local/dcs/bin`.

```
ncks -H -p /ZENDER/nco -l ./ in.nc
ncks -H -p mss:/ZENDER/nco -l ./ in.nc
```

Using '`-p`' is recommended because it clearly separates the *input-path* from the filename itself, sometimes called the *stub*. When *input-path* is not explicitly specified using '`-p`', NCO internally generates an *input-path* from the first input filename. The automatically generated *input-path* is constructed by stripping the input filename of everything following the final '`/`' character (i.e., removing the stub). The '`-l output-path`' option tells NCO where to store the remotely retrieved file and the output file. Often the path to a remotely retrieved file is quite different than the path on the local machine where you would like to store the file. If '`-l`' is not specified then NCO internally generates an *output-path* by simply setting *output-path* equal to *input-path* stripped of any machine names. If '`-l`' is not specified and the remote file resides on the NCAR MSS system, then the leading character of *input-path*, '`/`', is also stripped from *output-path*. Specifying *output-path* as '`-l ./`' tells NCO to store the remotely retrieved file and the output file in the current directory. Note that '`-l .`' is equivalent to '`-l ./`' though the latter is recommended as it is syntactically more clear.

3.3.1 DODS

The Distributed Oceanographic Data System (DODS) provides replacements for common data interface libraries like netCDF. The DODS versions of these libraries implement network transparent access to data using the HTTP protocol. NCO may be DODS-enabled by linking NCO to the DODS libraries. Examples of how to do this are given in the DODS documentation and in the '`Makefile`' distributed with NCO. Building NCO with `make DODS=Y` adds the (non-intuitive) commands to link to the DODS libraries installed in the `$DODS_ROOT` directory. You will probably need to visit the DODS Homepage (`http://www.unidata.ucar.edu/packages/dods`) to learn which libraries to obtain and link to for the DODS-enabled NCO executables.

Once NCO is DODS-enabled the operators are DODS clients. All DODS clients have network transparent access to any files controlled by a DODS server. Simply specify the path to the file in URL notation

```
ncks -C -d lon,0 -v lon -l ./ -p http://www.cdc.noaa.gov/cgi-bin/nph-nc/
Datasets/ncep.reanalysis.dailyavgs/surface air.sig995.1975.nc foo.nc
```

NCO operates on these remote files without having to transfer the files to the local disk. DODS causes all the I/O to appear to NCO as if the files were local. Only the required data (e.g., the variable or hyperslab specified) are transferred over the network. The advantages of this are obvious if you are examining small parts of large files stored at remote locations.

Note that the remote retrieval features of NCO can be used to retrieve *any* file, including non-netCDF files, via `SSH`, anonymous FTP, or `msrcp`. Often this method is quicker than using a browser, or running an FTP session from a shell window yourself. For example, say you want to obtain a JPEG file from a weather server.

```
ncks -p ftp://weather.edu/pub/pix/jpeg -l ./ storm.jpg
```

In this example, `ncks` automatically performs an anonymous FTP login to the remote machine and retrieves the specified file. When `ncks` attempts to read the local copy of '`storm.jpg`' as a netCDF file, it fails and exits, leaving '`storm.jpg`' in the current directory.

3.4 Retention of remotely retrieved files

Availability: All operators
Short options: '`-R`'
Long options: '`--rtn`', '`--retain`'

In order to conserve local file system space, files retrieved from remote locations are automatically deleted from the local file system once they have been processed. Many NCO operators were constructed to work with numerous large (e.g., 200 MB) files. Retrieval of multiple files from remote locations is done serially. Each file is retrieved, processed, then deleted before the cycle repeats. In cases where it is useful to keep the remotely-retrieved files on the local file system after processing, the automatic removal feature may be disabled by specifying '`-R`' on the command line.

3.5 Including/Excluding specific variables

Availability: (`ncap`), `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncks`, `ncra`, `ncrcat`, `ncwa`
Short options: '`-v`', '`-x`'
Long options: '`--variable`', '`--exclude`' or '`--xcl`'

Variable subsetting is implemented with the '`-v` *var*`[,...]`' and '`-x`' options. A list of variables to extract is specified following the '`-v`' option, e.g., '`-v time,lat,lon`'. Not using the '`-v`' option is equivalent to specifying all variables. The '`-x`' option causes the list of variables specified with '`-v`' to be *excluded* rather than *extracted*. Thus '`-x`' saves typing when you only want to extract fewer than half of the variables in a file. Remember, if averaging or concatenating large files stresses your systems memory or disk resources, then the easiest solution is often to use the '`-v`' option to retain only the variables you really need (see Section 2.9 [Memory usage], page 15). Note that, due to its special capabilities, `ncap` interprets the '`-v`' switch differently (see Section 4.1 [ncap netCDF Arithmetic Processor], page 42). For `ncap`, the '`-v`' switch takes no arguments and indicates that *only* user-defined variables should be output. `ncap` neither accepts nor understands the -x switch.

As of NCO 2.8.1 (August, 2003), variable name arguments of the '`-v`' switch may contain *extended regular expressions*. For example, '`-v '^DST'`' selects all variables beginning with the string `DST`. Extended regular expressions are defined by the GNU `egrep` command. The meta-characters used to express pattern matching operations are '`^$+?.*[]{}|`'. If the regular expression pattern matches *any* part of a variable name then that variable is selected. This capability is called *wildcarding*, and is very useful for sub-setting large data files.

Because of its wide availability, NCO uses the POSIX regular expression library `regex`. Regular expressions of arbitary complexity may be used. Since netCDF variable names are relatively simple constructs, only a few varieties of variable wildcards are likely to be useful. Consider the variables `Q01`–`Q99`, `Q100`, `Q_H2O`, `X_H2O`, `Q_CO2`, `X_CO2`.

```
ncks -v 'Q+' in.nc               # Select variables that start with Q
ncks -v 'Q??' in.nc              # Select Q01--Q99, QAA--QZZ
ncks -v 'Q[0-9][0-9]' in.nc      # Select Q01--Q99
ncks -v 'H2O$' in.nc             # Select Q_H2O, X_H2O
ncks -v '^Q[0-9][0-9]' in.nc     # Select Q01--Q99, Q100
ncks -v '^Q[0-9][0-9]$' in.nc    # Select Q01--Q99
ncks -v '^[a-z]_[a-z]{3}$' in.nc # Select Q_H2O, X_H2O, Q_CO2, X_CO2
```

Beware—the repetition patter matching operator '`*`' matches zero or more occurences of a regular expression. Thus '`^o*`', '`^t*`', and '`^[a-z]*`' select all variables. The documentation for the UNIX `egrep` command contains the detailed description of the extended regular expressions that NCO supports.

One must be careful to protect any special characters in the regular expression specification from being interpreted (globbed) by the shell. This is accomplish by enclosing special characters within single or double quotes

```
ncra -v Q?? in.nc out.nc   # Error: Shell attempts to glob wildcards
ncra -v 'Q??' in.nc out.nc # Correct: NCO interprets wildcards
ncra -v 'Q??' in*.nc out.nc
```

The final example shows that commands may use a combination of variable wildcarding and shell filename expansion (globbing).

3.6 Including/Excluding coordinate variables

Availability: `ncap`, `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncks`, `ncra`, `ncrcat`, `ncwa`
Short options: '`-C`', '`-c`'
Long options: '`--no-coords`', '`--no-crd`', '`--crd`', '`--coords`'

By default, coordinates variables associated with any variable appearing in the *output-file* will also appear in the *output-file*, even if they are not explicitly specified, e.g., with the '`-v`' switch. Thus variables with a latitude coordinate `lat` always carry the values of `lat` with them into the *output-file*. This feature can be disabled with '`-C`', which causes NCO to not automatically add coordinates to the variables appearing in the *output-file*. However, using '`-C`' does not preclude the user from including some coordinates in the output files simply by explicitly selecting the coordinates with the *-v* option. The '`-c`' option, on the other hand, is a shorthand way of automatically specifying that *all* coordinate variables in the *input-files* should appear in the *output-file*. Thus '`-c`' allows the user to select all the coordinate variables without having to know their names.

3.7 C & Fortran index conventions

Availability: `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncks`, `ncra`, `ncrcat`, `ncwa`
Short options: '`-F`'
Long options: '`--fortran`'

By default, NCO uses C-style (0-based) indices for all I/O. The '`-F`' switch tells NCO to switch to reading and writing with Fortran index conventions. In Fortran, indices begin counting from 1 (rather than 0), and dimensions are ordered from fastest varying to slowest varying. Consider a file '`85.nc`' containing 12 months of data in the record dimension `time`. The following hyperslab operations produce identical results, a June-July-August average of the data:

```
ncra -d time,5,7 85.nc 85_JJA.nc
ncra -F -d time,6,8 85.nc 85_JJA.nc
```

Printing variable *three_dmn_var* in file '`in.nc`' first with C indexing conventions, then with Fortran indexing conventions results in the following output formats:

```
% ncks -H -v three_dmn_var in.nc
lat[0]=-90 lev[0]=1000 lon[0]=-180 three_dmn_var[0]=0
...
% ncks -F -H -v three_dmn_var in.nc
lon(1)=0 lev(1)=100 lat(1)=-90 three_dmn_var(1)=0
...
```

3.8 Hyperslabs

Availability: `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncks`, `ncra`, `ncrcat`, `ncwa`
Short options: '`-d`'
Long options: '`--dimension`', '`--dmn`'

A *hyperslab* is a subset of a variable's data. The coordinates of a hyperslab are specified with the `-d` *`dim`*`,[`*`min`*`][,[`*`max`*`]]` short option (or with the '`--dimension`' or '`--dmn`' long options). The bounds of the hyperslab to be extracted are specified by the associated *min* and *max* values. A half-open range is specified by omitting either the *min* or *max* parameter but including the separating comma. The unspecified limit is interpreted as the maximum or minimum value in the unspecified direction. A cross-section at a specific coordinate is extracted by specifying only the *min* limit and omitting a trailing comma. Dimensions not mentioned are passed with no reduction in range. The dimensionality of variables is not reduced (in the case of a cross-section, the size of the constant dimension will be one). If values of a coordinate-variable are used to specify a range or cross-section, then the coordinate variable must be monotonic (values either increasing or decreasing). In this case, command-line values need not exactly match coordinate values for the specified dimension. Ranges are determined by seeking the first coordinate value to occur in the closed range [*min*,*max*] and including all subsequent values until one falls outside the range. The coordinate value for a cross-section is the coordinate-variable value closest to the specified value and must lie within the range or coordinate-variable values.

Coordinate values should be specified using real notation with a decimal point required in the value, whereas dimension indices are specified using integer notation without a decimal point. This convention serves only to differentiate coordinate values from dimension indices. It is independent of the type of any netCDF coordinate variables. For a given dimension, the specified limits must both be coordinate values (with decimal points) or dimension indices (no decimal points).

User-specified coordinate limits are promoted to double precision values while searching for the indices which bracket the range. Thus, hyperslabs on coordinates of type `NC_BYTE` and `NC_CHAR` are computed numerically rather than lexically, so the results are unpredictable.

The relative magnitude of *min* and *max* indicate to the operator whether to expect a *wrapped coordinate* (see Section 3.11 [Wrapped coordinates], page 28), such as longitude. If *min* > *max*, the NCO expects the coordinate to be wrapped, and a warning message will be printed. When this occurs, NCO selects all values outside the domain [*max* < *min*], i.e., all the values exclusive of the values which would have been selected if *min* and *max* were swapped. If this seems confusing, test your command on just the coordinate variables with `ncks`, and then examine the output to ensure NCO selected the hyperslab you expected (coordinate wrapping is currently only supported by `ncks`).

Because of the way wrapped coordinates are interpreted, it is very important to make sure you always specify hyperslabs in the monotonically increasing sense, i.e., *min* < *max* (even if the underlying coordinate variable is monotonically decreasing). The only exception to this is when you are indeed specifying a wrapped coordinate. The distinction is crucial to understand because the points selected by, e.g., `-d longitude,50.,340.`, are exactly the complement of the points selected by `-d longitude,340.,50.`.

Not specifying any hyperslab option is equivalent to specifying full ranges of all dimensions. This option may be specified more than once in a single command (each hyperslabed dimension requires its own `-d` option).

3.9 Multislabs

Availability: `ncks`
Short options: '`-d`'
Long options: '`--dimension`', '`--dmn`'

In late 2002, `ncks` added support for specifying a *multislab* for any variable. A multislab is a union of one or more hyperslabs which is specified by chaining together hyperslab commands, i.e., `-d` options (see Section 3.8 [Hyperslabs], page 24). This allows multislabs to overcome some restraints which limit hyperslabs.

A single `-d` option can only specify a contiguous and/or regularly spaced multi-dimensional array of data. Multislabs are constructed from multiple `-d` options and may therefore have non-regularly spaced arrays. For example, suppose it is desired to operate on all longitudes from 10.0 to 20.0 and from 80.0 to 90.0 degrees. The combined range of longitudes is not selectable in a single hyperslab specfication of the form '`-d` *lon,min,max*' or '`-d` *lon,min,max,stride*' because its elements are irregularly spaced in coordinate space (and presumably in index space too). The multislab specification for obtaining these values is simply the union of the hyperslabs specifications that comprise the multislab, i.e.,

```
ncks -d lon,10.,20. -d lon,80.,90. in.nc out.nc
ncks -d lon,10.,15. -d lon,15.,20. -d lon,80.,90. in.nc out.nc
```

Any number of hyperslabs specifications may be chained together to specify the multislab. Multislabs are more efficient than the alternative of sequentially performing hyperslab operations and concatenating the results. This is because NCO employs a novel multislab algorithm to minimize the number of I/O operations when retrieving irregularly spaced data from disk.

Users may specify redundant ranges of indices in a multislab, e.g.,

```
ncks -d lon,0,4 -d lon,2,9,2 in.nc out.nc
```

This command retrieves the first five longitudes, and then every other longitude value up to the tenth. Elements 0, 2, and 4 are specified by both hyperslab arguments (hence this is redundant) but will count only once if an arithmetic operation is being performed. The NCO multislab algorithm retrieves each element from disk once and only once. Thus users may take some shortcuts in specifying multislabs and the algorithm will obtain the intended values. Specifying redundant ranges is not encouraged, but may be useful on occasion and will not result in unintended consequences.

A final example shows the real power of multislabs. Suppose the *Q* variable contains three dimensional arrays of distinct chemical constituents in no particular order. We are interested in the NOy species in a certain geographic range. Say that NO, NO2, and N2O5 are elements 0, 1, and 5 of the *species* dimension of *Q*. The multislab specification might look something like

```
ncks -d species,0,1 -d species,5 -d lon,0,4 -d lon,2,9,2 in.nc out.nc
```

Multislabs are powerful because they may be specified for every dimension at the same time. Thus multislabs obsolete the need to execute multiple `ncks` commands to gather the desired range of data. We envision adding multislab support to all arithmetic operators in the future.

3.10 UDUnits Support

Availability: `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncks`, `ncra`, `ncrcat`, `ncwa`
Short options: '`-d`'
Long options: '`--dimension`', '`--dmn`'

There is more than one way to hyperslab a cat. The UDUnits (`http://www.unidata.ucar.edu/packages/u`
package provides a library which, if present, NCO uses to translate user-specified physical dimensions into the physical dimensions of data stored in netCDF files. Unidata provides UDUnits under the same terms as netCDF, so sites should install both. Compiling NCO

with UDUnits support is currently optional but may become required in a future version of NCO.

Two examples suffice to demonstrate the power and convenience of UDUnits support. First, consider extraction of a variable containing non-record coordinates with physical dimensions stored in MKS units. In the following example, the user extracts all wavelengths in the visible portion of the spectrum in terms of the units very frequently used in visible spectroscopy, microns:

```
% ncks -O -C -H -u -v wvl -d wvl,"0.4 micron","0.7 micron" in.nc
wvl[0]=5e-07 meter
```

The hyperslab returns the correct values because the *wvl* variable is stored on disk with a length dimension that UDUnits recognizes in the `units` attribute. The automagical algorithm that implements this functionality is worth describing since understanding it helps one avoid some potential pitfalls. First, the user includes the physical units of the hyperslab dimensions she supplies, separated by a simple space from the numerical values of the hyperslab limits. She encloses each coordinate specifications in quotes so that the shell does not break the *value-space-unit* string into separate arguments before passing them to NCO. Double quotes (`"foo"`) or single quotes (`'foo'`) are equally valid for this purpose. Second, NCO recognizes that units translation is requested because each hyperslab argument contains text characters and non-initial spaces. Third, NCO determines whether the *wvl* is dimensioned with a coordinate variable that has a `units` attribute. In this case, *wvl* itself is a coordinate variable. The value of its `units` attribute is `meter`. Thus *wvl* passes this test so UDUnits conversion is attempted. If the coordinate associated with the variable does not contain a `units` attribute, then NCO aborts. Fourth, NCO passes the specified and desired dimension strings (microns are specified by the user, meters are required by NCO) to the UDUnits library. Fifth, the UDUnits library that these dimension are commensurate and it returns the appropriate linear scaling factors to convert from microns to meters to NCO. If the units are incommensurate (i.e., not expressible in the same fundamental MKS units), or are not listed in the UDUnits database, then NCO aborts since it cannot determine the user's intent. Finally, NCO uses the scaling information to convert the user-specified hyperslab limits into the same physical dimensions as those of the corresponding cooridinate variable on disk. At this point, NCO can perform a coordinate hyperslab using the same algorithm as if the user had specified the hyperslab without requesting units conversion.

The translation and dimennterpretation of time coordinates shows a more powerful, and probably more common, application of the UDUnits feature. In this example, the user prints all data between the eighth and ninth of December, 1999, from a variable whose time dimension is hours since the year 1900:

```
% ncks -O -C -H -u -v time_udunits -d time_udunits,"1999-12-08 \
12:00:0.0","1999-12-09 00:00:0.0",2 in.nc foo2.nc
% time_udunits[1]=876018 hours since 1900-01-01 00:00:0.0
```

Here, the user invokes the stride (see Section 3.12 [Stride], page 29) capability to obtain every other timeslice. This is possible because the UDUnits feature is additive, not exclusive—it works in conjunction with all other hyperslabbing (see Section 3.8 [Hyperslabs], page 24) options and in all operators which support hyperslabbing. The following example shows how one might average data in a time period spread across multiple input files

```
ncra -O -d time,"1939-09-09 12:00:0.0","1945-05-08 00:00:0.0" \
in1.nc in2.nc in3.nc out.nc
```

Note that there is no excess whitespace before or after the individual elements of the '`-d`' argument. This is important since, as far as the shell knows, '`-d`' takes only *one* command-line argument. Parsing this argument into its component `dim,[min],[max],stride` elements (see Section 3.8 [Hyperslabs], page 24) is the job of NCO. When unquoted whitespace is present between these elements, the shell passes NCO arugment fragments which will not parse as intended.

The UDUnits (`http://www.unidata.ucar.edu/packages/udunits`) package documentation describes the supported formats of time dimensions. Among the metadata conventions which adhere to these formats are the Climate and Forecast (CF) Conventions (`http://www.cgd.ucar.edu/cms/eaton/cf-metadata/CF-working.html`) and the Cooperative Ocean/Atmosphere Research Data Service (COARDS) Conventions (`http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html`). The following '`-d arguments`' extract the same data using commonly encountered time dimension formats:

```
-d time,"1918-11-11 11:00:0.0","1939-09-09 00:00:0.0"
```

All of these formats include at least one dash `-` in a non-leading character position (a dash in a leading character position is a negative sign). NCO assumes that a non-leading dash in a limit string indicates that a UDUnits date conversion is requested.

netCDF variables should always be stored with MKS units, so that application programs may assume MKS dimensions apply to all input variables. The UDUnits feature is intended to alleviate some of the NCO user's pain when handling MKS units. It connects users who think in human-friendly units (e.g., miles, millibars, days) to extract data which are always stored in God's units, MKS (e.g., meters, Pascals, seconds). The feature is not intended to encourage writers to store data in esoteric units (e.g., furlongs, pounds per square inch, fortnights).

3.11 Wrapped coordinates

Availability: `ncks`
Short options: '`-d`'
Long options: '`--dimension`', '`--dmn`'

A *wrapped coordinate* is a coordinate whose values increase or decrease monotonically (nothing unusual so far), but which represents a dimension that ends where it begins (i.e., wraps around on itself). Longitude (i.e., degrees on a circle) is a familiar example of a wrapped coordinate. Longitude increases to the East of Greenwich, England, where it is defined to be zero. Halfway around the globe, the longitude is 180 degrees East (or West). Continuing eastward, longitude increases to 360 degrees East at Greenwich. The longitude values of most geophysical data are either in the range [0,360), or [−180,180). In either case, the Westernmost and Easternmost longitudes are numerically separated by 360 degrees, but represent contiguous regions on the globe. For example, the Saharan desert stretches from roughly 340 to 50 degrees East. Extracting the hyperslab of data representing the Sahara from a global dataset presents special problems when the global dataset is stored consecutively in longitude from 0 to 360 degrees. This is because the data for the Sahara will not be contiguous in the *input-file* but is expected by the user to be contiguous in the *output-file*. In this case, `ncks` must invoke special software routines to assemble the desired output hyperslab from multiple reads of the *input-file*.

Assume the domain of the monotonically increasing longitude coordinate `lon` is $0 < lon < 360$. `ncks` will extract a hyperslab which crosses the Greenwich meridian simply by specifying the westernmost longitude as *min* and the easternmost longitude as *max*. The following commands extract a hyperslab containing the Saharan desert:

```
ncks -d lon,340.,50. in.nc out.nc
ncks -d lon,340.,50. -d lat,10.,35. in.nc out.nc
```

The first example selects data in the same longitude range as the Sahara. The second example further constrains the data to having the same latitude as the Sahara. The coordinate `lon` in the *output-file*, '`out.nc`', will no longer be monotonic! The values of `lon` will be, e.g., '`340, 350, 0, 10, 20, 30, 40, 50`'. This can have serious implications should you run '`out.nc`' through another operation which expects the `lon` coordinate to be monotonically increasing. Fortunately, the chances of this happening are slim, since `lon` has already been hyperslabbed, there should be no reason to hyperslab `lon` again. Should you need to hyperslab `lon` again, be sure to give dimensional indices as the hyperslab arguments, rather than coordinate values (see Section 3.8 [Hyperslabs], page 24).

3.12 Stride

Availability: `ncks`, `ncra`, `ncrcat`
Short options: '`-d`'
Long options: '`--dimension`', '`--dmn`'

`ncks` offers support for specifying a *stride* for any hyperslab, while `ncra` and `ncrcat` suport the *stride* argument only for the record dimension. The *stride* is the spacing between consecutive points in a hyperslab. A *stride* of 1 means pick all the elements of the hyperslab, but a *stride* of 2 means skip every other element, etc. Using the *stride* option with `ncra` and `ncrcat` makes it possible, for instance, to average or concatenate regular intervals across multi-file input data sets.

The *stride* is specified as the optional fourth argument to the '`-d`' hyperslab specification: `-d dim,[min][,[max]][,[stride]]`. Specify *stride* as an integer (i.e., no decimal point) following the third comma in the '`-d`' argument. There is no default value for *stride*. Thus using '`-d time,,,2`' is valid but '`-d time,,,2.0`' and '`-d time,,,`' are not. When *stride* is specified but *min* is not, there is an ambiguity as to whether the extracted hyperslab should begin with (using C-style, 0-based indexes) element 0 or element '`stride-1`'. NCO must resolve this ambiguity and it chooses element 0 as the first element of the hyperslab when *min* is not specified. Thus '`-d time,,,stride`' is syntactically equivalent to '`-d time,0,,stride`'. This means, for example, that specifying the operation '`-d time,,,2`' on the array '1,2,3,4,5' selects the hyperslab '1,3,5'. To obtain the hyperslab '2,4' instead, simply explicitly specify the starting index as 1, i.e., '-d time,1,,2'.

For example, consider a file '8501_8912.nc' which contains 60 consecutive months of data. Say you wish to obtain just the March data from this file. Using 0-based subscripts (see Section 3.7 [Fortran indexing], page 24) these data are stored in records 2, 14, ... 50 so the desired *stride* is 12. Without the *stride* option, the procedure is very awkward. One could use ncks five times and then use ncrcat to concatenate the resulting files together:

```
for idx in 02 14 26 38 50; do # Bourne Shell
ncks -d time,${idx} 8501_8912.nc foo.${idx}
done
foreach idx (02 14 26 38 50) # C Shell
ncks -d time,${idx} 8501_8912.nc foo.${idx}
end
ncrcat foo.?? 8589_03.nc
rm foo.??
```

With the *stride* option, ncks performs this hyperslab extraction in one operation:

```
ncks -d time,2,,12 8501_8912.nc 8589_03.nc
```

See Section 4.7 [ncks netCDF Kitchen Sink], page 61, for more information on ncks.

The *stride* option is supported by ncra and ncrcat for the record dimension only. This makes it possible, for instance, to average or concatenate regular intervals across multi-file input data sets.

```
ncra -F -d time,3,,12 85.nc 86.nc 87.nc 88.nc 89.nc 8589_03.nc
ncrcat -F -d time,3,,12 85.nc 86.nc 87.nc 88.nc 89.nc 8503_8903.nc
```

3.13 Missing values

Availability: ncap, ncbo, ncea, ncflint, ncra, ncwa
Short options: None

The phrase *missing data* refers to data points that are missing, invalid, or for any reason not intended to be arithmetically processed in the same fashion as valid data. The NCO arithmetic operators attempt to handle missing data in an intelligent fashion. There are four steps in the NCO treatment of missing data:

1. Identifying variables which may contain missing data.

   NCO follows the convention that missing data should be stored with the *missing_value* specified in the variable's missing_value attribute[4]. The *only* way NCO recognizes that a variable *may* contain missing data is if the variable has a missing_value attribute. In

[4] NCO averagers have a bug (TODO 121) which may cause them to behave incorrectly if the *missing_value* = '0.0' for a variable to be averaged. The workaround for this bug is to change *missing_value* to anything besides zero.

this case, any elements of the variable which are numerically equal to the *missing_value* are treated as missing data.

2. Converting the *missing_value* to the type of the variable, if neccessary.

   Consider a variable *var* of type *var_type* with a `missing_value` attribute of type *att_type* containing the value *missing_value*. As a guideline, the type of the `missing_value` attribute should be the same as the type of the variable it is attached to. If *var_type* equals *att_type* then NCO straightforwardly compares each value of *var* to *missing_value* to determine which elements of *var* are to be treated as missing data. If not, then NCO will internally convert *att_type* to *var_type* by using the implicit conversion rules of C, or, if *att_type* is `NC_CHAR`[5], by typecasting the results of the C function `strtod(missing_value)`. You may use the NCO operator `ncatted` to change the `missing_value` attribute and all data whose data is *missing_value* to a new value (see Section 4.2 [ncatted netCDF Attribute Editor], page 48).

3. Identifying missing data during arithmetic operations.

   When an NCO arithmetic operator is processing a variable *var* with a `missing_value` attribute, it compares each value of *var* to *missing_value* before performing an operation. Note the *missing_value* comparison inflicts a performance penalty on the operator. Arithmetic processing of variables which contain the `missing_value` attribute always incurs this penalty, even when none of the data are missing. Conversely, arithmetic processing of variables which do not contain the `missing_value` attribute never incurs this penalty. In other words, do not attach a `missing_value` attribute to a variable which does not contain missing data. This exhortation can usually be obeyed for model generated data, but it may be harder to know in advance whether all observational data will be valid or not.

4. Treatment of any data identified as missing in arithmetic operators.

   NCO averagers (`ncra`, `ncea`, `ncwa`) do not count any element with the value *missing_value* towards the average. `ncbo` and `ncflint` define a *missing_value* result when either of the input values is a *missing_value*. Sometimes the *missing_value* may change from file to file in a multi-file operator, e.g., `ncra`. NCO is written to account for this (it always compares a variable to the *missing_value* assigned to that variable in the current file). Suffice it to say that, in all known cases, NCO does "the right thing".

3.14 Operation Types

Availability: `ncra`, `ncea`, `ncwa`
Short options: '`-y`'
Long options: '`--operation`', '`--op_typ`'

The '`-y op_typ`' switch allows specification of many different types of operations Set *op_typ* to the abbreviated key for the corresponding operation:

`avg` Mean value (default)

`sqravg` Square of the mean

[5] For example, the DOE ARM program often uses *att_type* = `NC_CHAR` and *missing_value* = '`-99999.`'.

`avgsqr` Mean of sum of squares

`max` Maximium value

`min` Minimium value

`rms` Root-mean-square (normalized by N)

`rmssdn` Root-mean square normalized by *N-1*

`sqrt` Square root of the mean

`ttl` Sum of values

If an operation type is not specified with '`-y`' then the operator will perform an arithmetic average by default. The mathematical definition of each operation is given below. See Section 4.11 [ncwa netCDF Weighted Averager], page 72, for additional information on masks and normalization. Averaging is the default, and will be described first so the terminology for the other operations is familiar.

The masked, weighted average of a variable x can be generally represented as

$$\bar{x}_j = \frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i}{\sum_{i=1}^{i=N} \mu_i m_i w_i}$$

where $\bar{x}_j$ is the j'th element of the output hyperslab, x_i is the i'th element of the input hyperslab, μ_i is 1 unless x_i equals the missing value, m_i is 1 unless x_i is masked, and w_i is the weight. This formiddable looking formula represents a simple weighted average whose bells and whistles are all explained below. It is not early to note, however, that when $\mu_i = m_i = w_i = 1$, the generic averaging expression above reduces to a simple arithmetic average. Furthermore, $m_i = w_i = 1$ for all operators besides `ncwa`, but these variables are included in the discussion below for completeness and for possible future use in other operators.

The size J of the output hyperslab for a given variable is the product of all the dimensions of the input variable which are not averaged over. The size N of the input hyperslab contributing to each $\bar{x}_j$ is simply the product of the sizes of all dimensions which are averaged over (i.e., dimensions specified with '`-a`'). Thus N is the number of input elements which *potentially* contribute to each output element. An input element x_i contributes to the output element x_j except in two conditions:

1. x_i equals the *missing value* (see Section 3.13 [Missing values], page 30) for the variable.
2. x_i is located at a point where the masking condition (see Section 4.11.1 [Masking condition], page 73) is false.

Points x_i in either of these two categories do not contribute to x_j, they are ignored. We now define these criteria more rigorously.

Each x_i has an associated Boolean weight μ_i whose value is 0 or 1 (false or true). The value of μ_i is 1 (true) unless x_i equals the *missing value* (see Section 3.13 [Missing values], page 30) for the variable. Thus, for a variable with no `missing_value` attribute, μ_i is always 1. All NCO arithmetic operators (`ncbo`, `ncra`, `ncea`, `ncflint`, `ncwa`) treat missing values analogously.

Besides (weighted) averaging, `ncwa`, `ncra`, and `ncea` also compute some common non-linear operations which may be specified with the '`-y`' switch (see Section 3.14 [Operation Types], page 31). The other rank-reducing operations are simple variations of the generic weighted mean described above. The total value of x (`-y ttl`) is

$$\bar{x}_j = \sum_{i=1}^{i=N} \mu_i m_i w_i x_i$$

Note that the total is the same as the numerator of the mean of x, and may also be obtained in `ncwa` by using the '`-N`' switch (see Section 4.11 [ncwa netCDF Weighted Averager], page 72).

The minimum value of x (`-y min`) is

$$\bar{x}_j = \min[\mu_1 m_1 w_1 x_1, \mu_2 m_2 w_2 x_2, \ldots, \mu_N m_N w_N x_N]$$

Analogously, the maximum value of x (`-y max`) is

$$\bar{x}_j = \max[\mu_1 m_1 w_1 x_1, \mu_2 m_2 w_2 x_2, \ldots, \mu_N m_N w_N x_N]$$

Thus the minima and maxima are determined after any weights are applied.

The square of the mean value of x (`-y sqravg`) is

$$\bar{x}_j = \left(\frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i}{\sum_{i=1}^{i=N} \mu_i m_i w_i} \right)^2$$

The mean of the sum of squares of x (`-y avgsqr`) is

$$\bar{x}_j = \frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i^2}{\sum_{i=1}^{i=N} \mu_i m_i w_i}$$

If x represents a deviation from the mean of another variable, $x_i = y_i - \bar{y}$ (possibly created by `ncbo` in a previous step), then applying `avgsqr` to x computes the approximate variance of y. Computing the true variance of y requires subtracting 1 from the denominator, discussed below. For a large sample size however, the two results will be nearly indistinguishable.

The root mean square of x (`-y rms`) is

$$\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i^2}{\sum_{i=1}^{i=N} \mu_i m_i w_i}}$$

Thus `rms` simply computes the squareroot of the quantity computed by `avgsqr`.

The root mean square of x with standard-deviation-like normalization (`-y rmssdn`) is implemented as follows. When weights are not specified, this function is the same as the root mean square of x except one is subtracted from the sum in the denominator

$$\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{i=N} \mu_i m_i x_i^2}{-1 + \sum_{i=1}^{i=N} \mu_i m_i}}$$

If x represents the deviation from the mean of another variable, $x_i = y_i - \bar{y}$, then applying `rmssdn` to x computes the standard deviation of y. In this case the -1 in the denominator compensates for the degree of freedom already used in computing $\bar{y}$ in the numerator. Consult a statistics book for more details.

When weights are specified it is unclear how to compensate for this extra degree of freedom. Weighting the numerator and denominator of the above by w_i and subtracting one from the denominator is only appropriate when all the weights are 1.0. When the weights are arbitrary (e.g., Gaussian weights), subtracting one from the sum in the denominator does not necessarily remove one degree of freedom. Therefore when `-y rmssdn` is requested and weights are specified, `ncwa` actually implements the `rms` procedure. `ncea` and `ncra`, which do not allow weights to be specified, always implement the `rmssdn` procedure when asked.

The square root of the mean of x (`-y sqrt`) is

$$\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i}{\sum_{i=1}^{i=N} \mu_i m_i w_i}}$$

The definitions of some of these operations are not universally useful. Mostly they were chosen to facilitate standard statistical computations within the NCO framework. We are open to redefining and or adding to the above. If you are interested in having other statistical quantities defined in NCO please contact the NCO project (see Section 1.5 [Help and Bug reports], page 8).

EXAMPLES

Suppose you wish to examine the variable `prs_sfc(time,lat,lon)` which contains a time series of the surface pressure as a function of latitude and longitude. Find the minimium value of `prs_sfc` over all dimensions:

```
ncwa -y min -v prs_sfc in.nc foo.nc
```

Find the maximum value of `prs_sfc` at each time interval for each latitude:

```
ncwa -y max -v prs_sfc -a lon in.nc foo.nc
```

Find the root-mean-square value of the time-series of `prs_sfc` at every gridpoint:

```
ncra -y rms -v prs_sfc in.nc foo.nc
ncwa -y rms -v prs_sfc -a time in.nc foo.nc
```

The previous two commands give the same answer but `ncra` is preferred because it has a smaller memory footprint. Also, `ncra` leaves the (degenerate) `time` dimension in the output file (which is usually useful) whereas `ncwa` removes the `time` dimension.

These operations work as expected in multi-file operators. Suppose that `prs_sfc` is stored in multiple timesteps per file across multiple files, say '`jan.nc`', '`feb.nc`', '`march.nc`'. We can now find the three month maximium surface pressure at every point.

```
ncea -y max -v prs_sfc jan.nc feb.nc march.nc out.nc
```

It is possible to use a combination of these operations to compute the variance and standard deviation of a field stored in a single file or across multiple files. The procedure to compute the temporal standard deviation of the surface pressure at all points in a single file 'in.nc' involves three steps.

```
ncwa -O -v prs_sfc -a time in.nc out.nc
ncbo -O --op_typ=sub -v prs_sfc in.nc out.nc out.nc
ncra -O -y rmssdn out.nc out.nc
```

First the output file 'out.nc' is contructed containing the temporal mean of prs_sfc. Next 'out.nc' is overwritten with the deviation from the mean. Finally 'out.nc' is overwritten with the root-mean-square of itself. Note the use of '-y rmssdn' (rather than '-y rms') in the final step. This ensures the standard deviation is correctly normalized by one fewer than the number of time samples. The procedure to compute the variance is identical except for the use of '-y var' instead of '-y rmssdn' in the final step.

The procedure to compute the spatial standard deviation of a field in a single file 'in.nc' involves three steps.

```
ncwa -O -v prs_sfc,gw -a lat,lon -w gw in.nc out.nc
ncbo -O --op_typ=sub -v prs_sfc,gw in.nc out.nc out.nc
ncwa -O -y rmssdn -v prs_sfc -a lat,lon -w gw out.nc out.nc
```

First the appropriately weighted (with '-w gw') spatial mean values are written to the output file. This example includes the use of a weighted variable specified with '-w gw'. When using weights to compute standard deviations one must remember to include the weights in the initial output files so that they may be used again in the final step. The initial output file is then overwritten with the gridpoint deviations from the spatial mean. Finally the root-mean-square of the appropriately weighted spatial deviations is taken.

The procedure to compute the standard deviation of a time-series across multiple files involves one extra step since all the input must first be collected into one file.

```
ncrcat -O -v tpt in.nc in.nc foo1.nc
ncwa -O -a time foo1.nc foo2.nc
ncbo -O --op_typ=sub -v tpt foo1.nc foo2.nc foo2.nc
ncra -O -y rmssdn foo2.nc out.nc
```

The first step assembles all the data into a single file. This may require a lot of temporary disk space, but is more or less required by the ncbo operation in the third step.

3.15 Type conversion

Availability: ncap, ncbo, ncea, ncra, ncwa
Short options: None

Type conversion refers to the casting of one fundamental data type to another, e.g., converting NC_SHORT (2 bytes) to NC_DOUBLE (8 bytes). As a general rule, type conversions should be avoided for at least two reasons. First, type conversions are expensive since they require creating (temporary) buffers and casting each element of a variable from the type it was stored at to some other type. Second, the dataset's creator probably had a good reason for storing data as, say, NC_FLOAT rather than NC_DOUBLE. In a scientific framework there is no reason to store data with more precision than the observations were made. Thus NCO tries to avoid performing type conversions when performing arithmetic.

Type conversion during arithmetic in the languages C and Fortran is performed only when necessary. All operands in an operation are converted to the most precise type before the operation takes place. However, following this parsimonious conversion rule dogmatically results in numerous headaches. For example, the average of the two NC_SHORTs 17000s and 17000s results in garbage since the intermediate value which holds their sum is also of type NC_SHORT and thus cannot represent values greater than 32,767[6]. There are valid reasons for expecting this operation to succeed and the NCO philosophy is to make operators do what you want, not what is most pure. Thus, unlike C and Fortran, but like many other higher level interpreted languages, NCO arithmetic operators will perform automatic type conversion when all the following conditions are met[7]:

1. The operator is ncea, ncra, or ncwa. ncbo is not yet included in this list because subtraction did not benefit from type conversion. This will change in the future
2. The arithmetic operation could benefit from type conversion. Operations that could benefit (e.g., from larger representable sums) include averaging, summation, or any "hard" arithmetic. Type conversion does not benefit searching for minima and maxima ('-y min', or '-y max').
3. The variable on disk is of type NC_BYTE, NC_CHAR, NC_SHORT, or NC_INT. Type NC_DOUBLE is not type converted because there is no type of higher precision to convert to. Type NC_FLOAT is not type converted because, in our judgement, the performance penalty of always doing so would outweigh the (extremely rare) potential benefits.

When these criteria are all met, the operator converts the variable in question to type NC_DOUBLE, performs all the arithmetic operations, casts the NC_DOUBLE type back to the original type, and finally writes the result to disk. The result written to disk may not be what you expect, because of incommensurate ranges represented by different types, and because of (lack of) rounding. First, continuing the example given above, the average (e.g., '-y avg') of 17000s and 17000s is written to disk as 17000s. The type conversion feature of NCO makes this possible since the arithmetic and intermediate values are stored as NC_DOUBLEs, i.e., 34000.0d and only the final result must be represented as an NC_SHORT. Without the type conversion feature of NCO, the average would have been garbage (albeit predictable garbage near -15768s). Similarly, the total (e.g., '-y ttl') of 17000s and 17000s written to disk is garbage (actually -31536s) since the final result (the true total) of 34000 is outside the range of type NC_SHORT.

Type conversions use the floor function to convert floating point number to integers. Type conversions do not attempt to round floating point numbers to the nearest integer. Thus the average of 1s and 2s is computed in double precisions arithmetic as (1.0d + 1.5d)/2) = 1.5d. This result is converted to NC_SHORT and stored on disk as floor(1.5d) =

[6] $32767 = 2^{15} - 1$

[7] Operators began performing type conversions before arithmetic in NCO version 1.2, August, 2000. Previous versions never performed unnecessary type conversion for arithmetic.

`1s`[8]. Thus no "rounding up" is performed. The type conversion rules of C can be stated as follows: If n is an integer then any floating point value x satisfying $n \leq x < n + 1$ will have the value n when converted to an integer.

3.16 Suppressing interactive prompts

Availability: All operators
Short options: '`-O`', '`-A`'
Long options: '`--ovr`', '`--overwrite`', '`--apn`', '`--append`'

If the *output-file* specified for a command is a pre-existing file, then the operator will prompt the user whether to overwrite (erase) the existing *output-file*, attempt to append to it, or abort the operation. However, in processing large amounts of data, too many interactive questions can be a curse to productivity. Therefore NCO also implements two ways to override its own safety features, the '`-O`' and '`-A`' switches. Specifying '`-O`' tells the operator to overwrite any existing *output-file* without prompting the user interactively. Specifying '`-A`' tells the operator to attempt to append to any existing *output-file* without prompting the user interactively. These switches are useful in batch environments because they suppress interactive keyboard input.

3.17 History attribute

Availability: All operators
Short options: '`-h`'
Long options: '`--hst`', '`--history`'

All operators automatically append a `history` global attribute to any file they modify or create. The `history` attribute consists of a timestamp and the full string of the invocation command to the operator, e.g., '`Mon May 26 20:10:24 1997: ncks in.nc foo.nc`'. The full contents of an existing `history` attribute are copied from the first *input-file* to the *output-file*. The timestamps appear in reverse chronological order, with the most recent timestamp appearing first in the `history` attribute. Since NCO and many other netCDF operators adhere to the `history` convention, the entire data processing path of a given netCDF file may often be deduced from examination of its `history` attribute. As of May, 2002, NCO is case-insensitive to the spelling of the `history` attribute name. Thus attributes named `History` or `HISTORY` (which are non-standard and not recommended) will be treated as valid history attributes. When more than one global attribute fits the case-insensitive search for "history", the first one found will be used. `history` attribute To avoid information overkill, all operators have an optional switch ('`-h`') to override automatically appending the `history` attribute (see Section 4.2 [ncatted netCDF Attribute Editor], page 48).

[8] The actual type conversions are handled by intrinsic C-language type conversion, so the `floor()` function is not explicitly called, but the results are the same as if it were.

3.18 NCAR CSM Conventions

Availability: `ncbo`, `ncea`, `ncecat`, `ncflint`, `ncra`, `ncwa`
Short options: None

NCO recognizes NCAR CSM history tapes, and treats them specially. If you do not work with NCAR CSM data then you may skip this section. The CSM netCDF convention is described at `http://www.cgd.ucar.edu/csm/experiments/output.format.html`. Most of the CSM netCDF convention is transparent to NCO[9]. There are no known pitfalls associated with using any NCO operator on files adhering to this convention[10]. However, to facilitate maximum user friendliness, NCO does treat certain variables in some CSM files specially. The special functions are not required by the CSM netCDF convention, but experience has shown they do make life easier.

Currently, NCO determines whether a datafile is a CSM output datafile simply by checking whether value of the global attribute `convention` (if it exists) equals '`NCAR-CSM`'. Should `convention` equal '`NCAR-CSM`' in the (first) *input-file*, NCO will attempt to treat certain variables specially, because of their meaning in CSM files. NCO will not average the following variables often found in CSM files: `ntrm`, `ntrn`, `ntrk`, `ndbase`, `nsbase`, `nbdate`, `nbsec`, `mdt`, `mhisf`. These variables contain scalar metadata such as the resolution of the host CSM model and it makes no sense to change their values. Furthermore, the `ncbo` operator does not operate on (i.e., add, subtract, etc.) the following variables: `gw`, `ORO`, `date`, `datesec`, `hyam`, `hybm`, `hyai`, `hybi`. These variables represent the Gaussian weights, the orography field, time fields, and hybrid pressure coefficients. These are fields which you want to remain unaltered in the output file 99% of the time. If you decide you would like any of the above CSM fields processed, you must use `ncrename` to rename them first.

3.19 ARM Conventions

Availability: `ncrcat`
Short options: None

`ncrcat` has been programmed to recognize ARM (Atmospheric Radiation Measurement Program) data files. If you do not work with ARM data then you may skip this section. ARM data files store time information in two variables, a scalar, `base_time`, and a record variable, `time_offset`. Subtle but serious problems can arise when these type of files are

[9] The exception is appending/altering the attributes `x_op`, `y_op`, `z_op`, and `t_op` for variables which have been averaged across space and time dimensions. This feature is scheduled for future inclusion in NCO.

[10] The CSM convention recommends `time` be stored in the format *time* since *base_time*, e.g., the `units` attribute of `time` might be '`days since 1992-10-8 15:15:42.5 -6:00`'. A problem with this format occurs when using `ncrcat` to concatenate multiple files together, each with a different *base_time*. That is, any `time` values from files following the first file to be concatenated should be corrected to the *base_time* offset specified in the `units` attribute of `time` from the first file. The analogous problem has been fixed in ARM files (see Section 3.19 [ARM Conventions], page 38) and could be fixed for CSM files if there is sufficient lobbying, and if Unidata fixes the UDUnits (`http://www.unidata.ucar.edu/packages/udunits`) package to build out of the box on Linux.

just blindly concatenated. Therefore `ncrcat` has been specially programmed to be able to chain together consecutive ARM *input-files* and produce and an *output-file* which contains the correct time information. Currently, `ncrcat` determines whether a datafile is an ARM datafile simply by testing for the existence of the variables `base_time`, `time_offset`, and the dimension `time`. If these are found in the *input-file* then `ncrcat` will automatically perform two non-standard, but hopefully useful, procedures. First, `ncrcat` will ensure that values of `time_offset` appearing in the *output-file* are relative to the `base_time` appearing in the first *input-file* (and presumably, though not necessarily, also appearing in the *output-file*). Second, if a coordinate variable named `time` is not found in the *input-files*, then `ncrcat` automatically creates the `time` coordinate in the *output-file*. The values of `time` are defined by the ARM convention *time* = *base_time* + *time_offset*. Thus, if *output-file* contains the `time_offset` variable, it will also contain the `time` coordinate. A short message is added to the `history` global attribute whenever these ARM-specific procedures are executed.

3.20 Operator version

Availability: All operators
Short options: '`-r`'
Long options: '`--revision`', '`--version`', or '`--vrs`'

All operators can be told to print their internal version number and copyright notice and then quit with the '`-r`' switch. The internal version number varies between operators, and indicates the most recent change to a particular operator's source code. This is useful in making sure you are working with the most recent operators. The version of NCO you are using might be, e.g., `1.2`. However using '`-r`' on, say, `ncks`, will produce something like '`NCO netCDF Operators version 1.2 Copyright (C) 1995--2000 Charlie Zender ncks version 1.30 (2000/07/31) "Bolivia"`'. This tells you `ncks` contains all patches up to version `1.30`, which dates from July 31, 2000.

4 Reference manual for all operators

This chapter presents reference pages for each of the operators individually. The operators are presented in alphabetical order. All valid command line switches are included in the syntax statement. Recall that descriptions of many of these command line switches are provided only in Chapter 3 [Common features], page 17, to avoid redundancy. Only options specific to, or most useful with, a particular operator are described in any detail in the sections below.

4.1 `ncap` netCDF Arithmetic Processor

SYNTAX

```
ncap [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]][,[stride]]] [-F] [-f]
[-l path] [-O] [-p path] [-R] [-r]
[-s algebra] [-S fl.nco] [-v]
input-file [output-file]
```

DESCRIPTION

Note: documentation for `ncap` is incomplete and evolving. The `ncap` parser tends to develop fitfully, and the best documentation for recent capabilities is the '`ChangeLog`' file.

`ncap` arithmetically processes a netCDF file. The processing instructions are contained either in the NCO script file '`fl.nco`' or in a sequence of command line arguments. The options '`-s`' (or long options '`--spt`' or '`--script`') are used for in-line scripts and '`-S`' (or long options '`--fl_spt`' or '`--script-file`') are used to provide the filename where (usually multiple) scripting commands are pre-stored. `ncap` was written to perform arbitrary albebraic transformations of data and archive the results as easily as possible. The results of the algebraic manipulations are called *derived fields*.

Unlike the other operators, `ncap` does not accept a list of variables to be operated on as an argument to '`-v`' (see Section 3.5 [Variable subsetting], page 22). Rather, the '`-v`' switch takes no arguments and indicates that `ncap` should output *only* user-defined variables. `ncap` does not accept or understand the -x switch.

4.1.1 Left hand casting

The following examples demonstrate the utility of the *left hand casting* ability of `ncap`. Consider first this simple, artificial, example. If *lat* and *lon* are one dimensional coordinates of dimensions *lat* and *lon*, respectively, then addition of these two one-dimensional arrays is intrinsically ill-defined because whether *lat_lon* should be dimensioned *lat* by *lon* or *lon* by *lat* is ambiguous (assuming that addition is to remain a *commutative* procedure, i.e., one that does not depend on the order of its arguments). Differing dimensions are said to be *orthogonal* to one another, and sets of dimensions which are mutually exclusive are orthogonal as a set and any arithmetic operation between variables in orthogonal dimensional spaces is ambiguous without further information.

The ambiguity may be resolved by enumerating the desired dimension ordering of the output expression inside square brackets on the left hand side (LHS) of the equals sign. This is called *left hand casting* because the user resolves the dimensional ordering of the RHS of the expression by specifying the desired ordering on the LHS.

```
ncap -O -s "lat_lon[lat,lon]=lat+lon" in.nc out.nc
ncap -O -s "lon_lat[lon,lat]=lat+lon" in.nc out.nc
```

The explicit list of dimensions on the LHS, `[lat,lon]` resolves the otherwise ambiguous ordering of dimensions in *lat_lon*. In effect, the LHS *casts* its rank properties onto the RHS.

Without LHS casting, the dimensional ordering of *lat_lon* would be undefined and, hopefully, `ncap` would print an error message.

Consider now a slightly more complex example. In geophysical models, a coordinate system based on a blend of terrain-following and density-following surfaces is called a *hybrid coordinate system*. In this coordinate system, four variables must be manipulated to obtain the pressure of the vertical coordinate: *PO* is the domain-mean surface pressure offset (a scalar), *PS* is the local (time-varying) surface pressure (usually two horizontal spatial dimensions, i.e, latitude by longitude), *hyam* is the weight given to surfaces of constant density (one spatial dimension, pressure, which is orthogonal to the horizontal dimensions), and *hybm* is the weight given to surfaces of constant elevation (also one spatial dimension). This command constructs a four-dimensional pressure `prs_mdp` from the four input variables of mixed rank and orthogonality:

```
ncap -O -s "prs_mdp[time,lat,lon,lev]=P0*hyam+PS*hybm" in.nc out.nc
```

Manipulating the four fields which define the pressure in a hybrid coordinate system is easy with left hand casting.

4.1.2 Syntax of `ncap` statements

Mastering `ncap` is relatively simple. Each valid statement *statement* consists of standard forward algebraic expression. The '`fl.nco`', if present, is simply a list of such statements, whitespace, and comments. The syntax of statements is most like the computer language C. The following characteristics of C are preserved:

Array syntax
: Arrays elements are placed within `[]` characters;

Array indexing
: Arrays are 0-based;

Array storage
: Last dimension is most rapidly varying;

Assignment statements
: A semi-colon '`;`' indicates the end of an assignment statement.

Comments
: Multi-line comments are enclosed within `/* */` characters. Single line comments are preceded by `//` characters.

Nesting
: Files may be nested in scripts using `#include` *`script`*. Note that the `#include` command is not followed by a semi-colon because it is a pre-processor directive, not an assignment statement. The filename '`script`' is interpreted relative to the run directory.

Attribute syntax
: The at-sign `@` is used to delineate an attribute name from a variable name.

4.1.3 Intrinsic functions

`ncap` contains a small but growing library of intrinsic functions. In addition to the standard mathematical functions (see Section 4.1.4 [Intrinsic mathematical functions], page 45), `ncap` currently supports packing and unpacking.

Packing and Unpacking Functions

`pack(x)`
: *Packing* The standard packing algorithm is applied to variable *x*. The packing algorithm is lossy, and produces data with the same dynamic range as the original but which requires no more than half the space to store. The packed variable is stored (usually) as type `NC_SHORT` with the two attributes required to unpack the variable, `scale_factor` and `add_offset`, stored at the original precision of the variable. Let *min* and *max* be the minimum and maximum values of *x*.

$$
\begin{aligned}
\mathrm{scale\_factor} &= (\max - \min)/\mathrm{ndrv} \\
\mathrm{add\_offset} &= (\min + \max)/2 \\
\mathrm{pck} &= (\mathrm{upk} - \mathrm{add\_offset})/\mathrm{scale\_factor} \\
&= \frac{\mathrm{ndrv} \times [\mathrm{upk} - (\min + \max)/2]}{\max - \min}
\end{aligned}
$$

where *ndrv* is the number of discrete representable values for given type of packed variable. The theoretical maximum value for *ndrv* is two raised to the number of bits used to store the packed variable. Thus if the variable is packed into type `NC_SHORT`, a 2 byte datatype, then there are at most $2^16 = 65536$ distinct values representible. In practice, the number of discretely representible values is taken to be one less than the theoretical maximum. This leaves extra space and solves potential problems with rounding which can occur during the unpacking of the variable. Thus for `NC_SHORT`, $ndrv = 65536 - 1 = 65535$. Less often, the variable may be packed into type `NC_CHAR`, where $ndrv = 256 - 1 = 255$, or type `NC_INT` where where $ndrv = 4294967295 - 1 = 4294967294$.

`unpack(x)`
: *Unpacking* The standard unpacking algorithm is applied to variable *x*. The unpacking algorithm depends on the presence of two attributes, `scale_factor` and `add_offset`. If `scale_factor` is present for a variable, the data are multiplied by the value *scale_factor* after the data are read. If `add_offset` is present for a variable, then the *add_offset* value is added to the data after the data are read. If both `scale_factor` and `add_offset` attributes are present, the data are first scaled by *scale_factor* before the offset *add_offset* is added.

$$
\begin{aligned}
\mathrm{upk} &= \mathrm{scale\_factor} \times \mathrm{pck} + \mathrm{add\_offset} \\
&= \frac{\mathrm{pck} \times (\max - \min)}{\mathrm{ndrv}} + \frac{\min + \max}{2}
\end{aligned}
$$

When `scale_factor` and `add_offset` are used for packing, the associated variable (containing the packed data) is typically of type `byte` or `short`, whereas the unpacked values are intended to be of type `float` or `double`. An attribute's `scale_factor` and `add_offset` should both be of the type intended for the unpacked data, e.g., `float` or `double`.

Type Conversion Functions

`byte(x)` *Convert to* `NC_BYTE` Converts x to external type `NC_BYTE`, a C-type `signed char`.

`char(x)` *Convert to* `NC_CHAR` Converts x to external type `NC_CHAR`, a C-type `unsigned char`.

`double(x)`
Convert to `NC_DOUBLE` Converts x to external type `NC_DOUBLE`, a C-type `double`.

`float(x)` *Convert to* `NC_FLOAT` Converts x to external type `NC_FLOAT`, a C-type `float`.

`int(x)` *Convert to* `NC_INT` Converts x to external type `NC_INT`, a C-type `int`.

`short(x)` *Convert to* `NC_SHORT` Converts x to external type `NC_SHORT`, a C-type `short`.

4.1.4 Intrinsic mathematical functions

`ncap` supports the standard mathematical functions supplied with most operating systems. Standard calculator notation is used for addition `+`, subtraction `-`, multiplication `*`, division `/`, exponentiation `^`, and modulus `%`. The available elementary mathematical functions are:

`abs(x)` *Absolute value* Absolute value of x.

`acos(x)` *Arc-cosine* Arc-cosine of x where x is specified in radians.

`acosh(x)` *Hyperbolic arc-cosine* Hyperbolic arc-cosine of x where x is specified in radians.

`asin(x)` *Arc-sine* Arc-sine of x where x is specified in radians.

`asinh(x)` *Hyperbolic arc-sine* Hyperbolic arc-sine of x where x is specified in radians.

`atan(x)` *Arc-tangent* Arc-tangent of x where x is specified in radians between $-\pi/2$ and $\pi/2$.

`atanh(x)` *Hyperbolic arc-tangent* Hyperbolic arc-tangent of x where x is specified in radians between $-\pi/2$ and $\pi/2$.

`ceil(x)` *Ceil* Ceiling of x.

`cerf(x)` *Complementary error function* Complementary error function of x where x is specified between -1 and 1.

`cos(x)` *Cosine* Cosine of x where x is specified in radians.

`cosh(x)` *Hyperbolic cosine* Hyperbolic cosine of x where x is specified in radians.

`erf(x)` *Error function* Error function of x where x is specified between -1 and 1.

`exp(x)` *Exponential* Exponential of x, e^x.

`floor(x)` *Floor* Floor of x.

`gamma(x)` *Gamma function* Gamma function of x, $\Gamma(x)$.

`log(x)` *Natural Logarithm* Natural logarithm of x, $\ln(x)$.

`log10(x)` *Base 10 Logarithm* Base 10 logarithm of x, $\log_{10}(x)$.

`nearbyint(x)`
Round inexactly Nearest integer to x is returned in floating point format. No exceptions are raised for *inexact conversions*.

`rint(x)` *Round exactly* Nearest integer to x is returned in floating point format. Exceptions are raised for *inexact conversions*.

`round(x)` *Round* Nearest integer to x is returned in floating point format. Round halfway cases away from zero, regardless of current IEEE rounding direction.

`sin(x)` *Sine* Sine of x where x is specified in radians.

`sinh(x)` *Hyperbolic sine* Hyperbolic sine of x where x is specified in radians.

`sqrt(x)` *Square Root* Square Root of x, $\sqrt{x}$.

`tan(x)` *Tangent* Tangent of x where x is specified in radians.

`tanh(x)` *Hyperbolic tangent* Hyperbolic tangent of x where x is specified in radians.

`trunc(x)` *Truncate* Nearest integer to x is returned in floating point format. Round halfway cases toward zero, regardless of current IEEE rounding direction.

The complete list of mathematical functions supported is platform-specific. Functions mandated by ANSI C are *guaranteed* to be present and are indicated with an asterisk [1]. and are indicated with an asterisk. Use the '`-f`' (or '`fnc_tbl`' or '`prn_fnc_tbl`') switch to print a complete list of functions supported on your platform. This prints a list of functions and whether they are supported for netCDF variables of intrinsic type NC_FLOAT and NC_DOUBLE. [2]

EXAMPLES

Define new attribute *new* for existing variable *one* as twice the existing attribute *double_att* of variable *att_var*:

```
ncap -O -s "one@new=2*att_var@double_att" in.nc out.nc
```

Average variables of mixed types (result is of type `double`):

[1] ANSI C compilers are guaranteed to support double precision versions of these functions. These functions normally operate on netCDF variables of type NC_DOUBLE without having to perform intrinsic conversions. For example, ANSI compilers provide `sin` for the sine of C-type `double` variables. The ANSI standard does not require, but many compilers provide, an extended set of mathematical functions that apply to single (`float`) and quadruple (`long double`) precision variables. Using these functions (e.g., `sinf` for `float`, `sinl` for `long double`), when available, is more efficient than casting variables to type `double`, performing the operation, and then recasting. NCO uses the faster intrinsic functions when they are available, and uses the casting method when they are not.

[2] Linux supports more of these intrinsic functions than other OSs.

```
ncap -O -s "average=(var_float+var_double+var_int)/3" in.nc out.nc
```

Multiple commands may be given to `ncap` in three ways. First, the commands may be placed in a script which is executed, e.g., '`tst.nco`'. Second, the commands may be individually specified with multiple '`-s`' arguments to the same `ncap` invocation. Third, the commands may be chained together into a single '`-s`' argument to `ncap`. Assuming the file '`tst.nco`' contains the commands `a=3;b=4;c=sqrt(a^2+b^2);`, then the following `ncap` invocations produce identical results:

```
ncap -O -v -S tst.nco in.nc out.nc
ncap -O -v -s "a=3" -s "b=4" -s "c=sqrt(a^2+b^2)" in.nc out.nc
ncap -O -v -s "a=3;b=4;c=sqrt(a^2+b^2)" in.nc out.nc
```

The second and third examples show that `ncap` does not require that a trailing semi-colon '`;`' be placed at the end of a '`-s`' argument, although a trailing semi-colon '`;`' is always allowed. However, semi-colons are required to separate individual assignment statements chained together as a single '`-s`' argument.

Imagine you wish to create a binary flag based on the value of an array. The flag should have value 1.0 where the array exceeds 1.0, and a value of 0.0 elsewhere. Assume the array named `ORO` is in '`in.nc`'. The variable `ORO_flg` in '`out.nc`'

```
# Add degenerate "record" dimension to ORO for averaging
ncecat -O -v ORO in.nc foo.nc
# Average degenerate "record" dimension using ORO as mask
ncwa -a record -O -m ORO -M 1.0 -o gt foo.nc foo.nc
# ORO is either 0.0 or > 1.0 everywhere
# Create ORO_frc in [0.0,1.0) then add 0.99 and convert to int
ncap -O -s "ORO_frc=ORO-int(ORO)" -s "ORO_flg=int(ORO_frc+0.99)" foo.nc out.nc
# ORO_flg now equals 0 or 1
```

This example uses `ncap` to compute the covariance of two variables. Let the variables u and v be the horizontal wind components. The *covariance* of u and v is defined as the time mean product of the deviations of u and v from their respective time means. Symbolically, the covariance $[u'v'] = [uv] - [u][v]$ where $[x]$ denotes the time-average of x, $[x] \equiv \frac{1}{\tau}\int_{t=0}^{t=\tau} x(t)\,dt$ and x' denotes the deviation from the time-mean. The covariance tells us how much of the correlation of two signals arises from the signal fluctuations versus the mean signals. Sometimes this is called the *eddy covariance*. We will store the covariance in the variable `uprmvprm`.

```
ncra -O -v u,v in.nc foo.nc # Compute time mean of u,v
ncrename -O -v u,uavg -v v,vavg foo.nc # Rename to avoid conflict
ncks -A -v u,v in.nc foo.nc # Place originals with time means
ncap -O -s "uprmvprm=u*v-uavg*vavg" foo.nc foo.nc # Covariance
ncra -O -v uprmvprm foo.nc out.nc # Time-mean covariance
```

The same answer would be obtained by replacing the step involving `ncap` with

```
ncap -O -s "uprmvprm=(u-uavg)*(v-vavg)" foo.nc foo.nc # Covariance
```

4.2 `ncatted` netCDF Attribute Editor

SYNTAX

```
ncatted [-a att_dsc] [-a ...] [-D dbg] [-h]
[-l path] [-O] [-p path] [-R] [-r]
input-file [output-file]
```

DESCRIPTION

`ncatted` edits attributes in a netCDF file. If you are editing attributes then you are spending too much time in the world of metadata, and `ncatted` was written to get you back out as quickly and painlessly as possible. `ncatted` can *append*, *create*, *delete*, *modify*, and *overwrite* attributes (all explained below). Furthermore, `ncatted` allows each editing operation to be applied to every variable in a file, thus saving you time when you want to change attribute conventions throughout a file. `ncatted` interprets character attributes as strings.

Because repeated use of `ncatted` can considerably increase the size of the `history` global attribute (see Section 3.17 [History attribute], page 37), the '`-h`' switch is provided to override automatically appending the command to the `history` global attribute in the *output-file*.

When `ncatted` is used to change the `missing_value` attribute, it changes the associated missing data self-consistently. If the internal floating point representation of a missing value, e.g., 1.0e36, differs between two machines then netCDF files produced on those machines will have incompatible missing values. This allows `ncatted` to change the missing values in files from different machines to a single value so that the files may then be concatenated together, e.g., by `ncrcat`, without losing any information. See Section 3.13 [Missing values], page 30, for more information.

The key to mastering `ncatted` is understanding the meaning of the structure describing the attribute modification, *att_dsc* specified by the required option '`-a`' or '`--attribute`'. Each *att_dsc* contains five elements, which makes using `ncatted` somewhat complicated, but powerful. The *att_dsc* argument structure contains five arguments in the following order:

att_dsc = *att_nm*, *var_nm*, *mode*, *att_type*, *att_val*

att_nm Attribute name. Example: `units`

var_nm Variable name. Example: `pressure`

mode Edit mode abbreviation. Example: `a`. See below for complete listing of valid values of *mode*.

att_type Attribute type abbreviation. Example: `c`. See below for complete listing of valid values of *att_type*.

att_val Attribute value. Example: `pascal`.

There should be no empty space between these five consecutive arguments. The description of these arguments follows in their order of appearance.

The value of *att_nm* is the name of the attribute you want to edit. This meaning of this should be clear to all users of the `ncatted` operator. If *att_nm* is omitted (i.e., left blank) and *Delete* mode is selected, then all attributes associated with the specified variable will be deleted.

The value of *var_nm* is the name of the variable containing the attribute (named *att_nm*) that you want to edit. There are two very important and useful exceptions to this rule. The value of *var_nm* can also be used to direct `ncatted` to edit global attributes, or to repeat the editing operation for every variable in a file. A value of *var_nm* of "global" indicates that *att_nm* refers to a global attribute, rather than a particular variable's attribute. This is the method `ncatted` supports for editing global attributes. If *var_nm* is left blank, on the other hand, then `ncatted` attempts to perform the editing operation on every variable in the file. This option may be convenient to use if you decide to change the conventions you use for describing the data.

The value of *mode* is a single character abbreviation (`a`, `c`, `d`, `m`, or `o`) standing for one of five editing modes:

`a` *Append*. Append value *att_val* to current *var_nm* attribute *att_nm* value *att_val*, if any. If *var_nm* does not have an attribute *att_nm*, there is no effect.

`c` *Create*. Create variable *var_nm* attribute *att_nm* with *att_val* if *att_nm* does not yet exist. If *var_nm* already has an attribute *att_nm*, there is no effect.

`d` *Delete*. Delete current *var_nm* attribute *att_nm*. If *var_nm* does not have an attribute *att_nm*, there is no effect. If *att_nm* is omitted (left blank), then all attributes associated with the specified variable are automatically deleted. When *Delete* mode is selected, the *att_type* and *att_val* arguments are superfluous and may be left blank.

`m` *Modify*. Change value of current *var_nm* attribute *att_nm* to value *att_val*. If *var_nm* does not have an attribute *att_nm*, there is no effect.

`o` *Overwrite*. Write attribute *att_nm* with value *att_val* to variable *var_nm*, overwriting existing attribute *att_nm*, if any. This is the default mode.

The value of *att_type* is a single character abbreviation (`f`, `d`, `l`, `i`, `s`, `c`, or `b`) standing for one of the seven primitive netCDF data types:

`f` *Float*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_FLOAT.

`d` *Double*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_DOUBLE.

`i` *Integer*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_INT.

`l` *Long*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_LONG.

`s` *Short*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_SHORT.

`c` *Char*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_CHAR.

`b` *Byte*. Value(s) specified in *att_val* will be stored as netCDF intrinsic type NC_BYTE.

The specification of *att_type* is optional in *Delete* mode.

The value of *att_val* is what you want to change attribute *att_nm* to contain. The specification of *att_val* is optional in *Delete* mode. Attribute values for all types besides NC_CHAR must have an attribute length of at least one. Thus *att_val* may be a single value or one-dimensional array of elements of type `att_type`. If the *att_val* is not set or is set to empty space, and the *att_type* is NC_CHAR, e.g., `-a units,T,o,c,""` or `-a units,T,o,c,`, then the corresponding attribute is set to have zero length. When specifying an array of values, it is safest to enclose *att_val* in single or double quotes, e.g., `-a levels,T,o,s,"1,2,3,4"` or `-a levels,T,o,s,'1,2,3,4'`. The quotes are strictly unnecessary around *att_val* except when *att_val* contains characters which would confuse the calling shell, such as spaces, commas, and wildcard characters.

NCO processing of NC_CHAR attributes is a bit like Perl in that it attempts to do what you want by default (but this sometimes causes unexpected results if you want unusual data storage). If the *att_type* is NC_CHAR then the argument is interpreted as a string and it may contain C-language escape sequences, e.g., `\n`, which NCO will interpret before writing anything to disk. NCO translates valid escape sequences and stores the appropriate ASCII code instead. Since two byte escape sequences, e.g., `\n`, represent one byte ASCII codes, e.g., ASCII 10 (decimal), the stored string attribute is one byte shorter than the input string length for each embedded escape sequence. The most frequently used C-language escape sequences are `\n` (for linefeed) and `\t` (for horizontal tab). These sequences in particular allow convenient editing of formatted text attributes. The other valid ASCII codes are `\a`, `\b`, `\f`, `\r`, `\v`, and `\\`. See Section 4.7 [ncks netCDF Kitchen Sink], page 61, for more examples of string formatting (with the `ncks` '`-s`' option) with special characters.

Analogous to `printf`, other special characters are also allowed by `ncatted` if they are "protected" by a backslash. The characters `"`, `'`, `?`, and `\` may be input to the shell as `\"`, `\'`, `\?`, and `\\`. NCO simply strips away the leading backslash from these characters before editing the attribute. No other characters require protection by a backslash. Backslashes which precede any other character (e.g., `3`, `m`, `$`, `|`, `&`, `@`, `%`, `{`, and `}`) will not be filtered and will be included in the attribute.

Note that the NUL character `\0` which terminates C language strings is assumed and need not be explicitly specified. If `\0` is input, it will not be translated (because it would terminate the string in an additional location). Because of these context-sensitive rules, if wish to use an attribute of type NC_CHAR to store data, rather than text strings, you should use `ncatted` with care.

EXAMPLES

Append the string "Data version 2.0.\n" to the global attribute `history`:

```
ncatted -O -a history,global,a,c,"Data version 2.0\n" in.nc
```

Note the use of embedded C language `printf()`-style escape sequences.

Change the value of the `long_name` attribute for variable `T` from whatever it currently is to "temperature":

```
ncatted -O -a long_name,T,o,c,temperature in.nc
```

Delete all existing `units` attributes:

```
ncatted -O -a units,,d,, in.nc
```

The value of *var_nm* was left blank in order to select all variables in the file. The values of *att_type* and *att_val* were left blank because they are superfluous in *Delete* mode.

Delete all attributes associated with the `tpt` variable:

```
ncatted -O -a ,tpt,d,, in.nc
```

The value of *att_nm* was left blank in order to select all attributes associate with the variable.

Modify all existing `units` attributes to "meter second-1"

```
ncatted -O -a units,,m,c,"meter second-1" in.nc
```

Overwrite the `quanta` attribute of variable `energy` to an array of four integers.

```
ncatted -O -a quanta,energy,o,s,"010,101,111,121" in.nc
```

Demonstrate input of C-language escape sequences (e.g., `\n`) and other special characters (e.g., `\"`)

```
ncatted -h -a special,global,o,c,
'\nDouble quote: \"\nTwo consecutive double quotes: \"\"\n
Single quote: Beyond my shell abilities!\nBackslash: \\\n
Two consecutive backslashes: \\\\\nQuestion mark: \?\n' in.nc
```

Note that the entire attribute is protected from the shell by single quotes. These outer single quotes are necessary for interactive use, but may be omitted in batch scripts.

4.3 ncbo netCDF Binary Operator

SYNTAX

```
ncbo [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]]] [-F] [-h] [-l path]
[-O] [-p path] [-R] [-r] [-v var[,...]]
[-x] [-y op_typ] file_1 file_2 file_3
```

DESCRIPTION

ncbo performs binary operations on variables in *file_1* and the corresponding variables (those with the same name) in *file_2* and stores the results in *file_3*. The binary operation operates on the entire files (modulo any excluded variables). One of the four standard arithmetic binary operations currently supported must be selected with the '-y *op_typ*' switch (or long options '--op_typ' or '--operation'). The valid binary operations for ncbo, their definitions, and corresponding values of the *op_typ* key are:

Addition
Definition: *file_3* = *file_1* + *file_2*
Alternate invocation: ncadd
op_typ key values: 'add', '+', 'addition'
Examples: 'ncbo --op_typ=add 1.nc 2.nc 3.nc', 'ncadd 1.nc 2.nc 3.nc'

Subtraction
Definition: *file_3* = *file_1* - *file_2*
Alternate invocations: ncdiff, ncsub, ncsubtract
op_typ key values: 'sbt', '-', 'dff', 'diff', 'sub', 'subtract', 'subtraction'
Examples: 'ncbo --op_typ=- 1.nc 2.nc 3.nc', 'ncdiff 1.nc 2.nc 3.nc'

Multiplication
Definition: *file_3* = *file_1* * *file_2*
Alternate invocations: ncmult, ncmultiply
op_typ key values: 'mlt', '*', 'mult', 'multiply', 'multiplication'
Examples: 'ncbo --op_typ=mlt 1.nc 2.nc 3.nc', 'ncmult 1.nc 2.nc 3.nc'

Division
Definition: *file_3* = *file_1* / *file_2*
Alternate invocation: ncdivide
op_typ key values: 'dvd', '/', 'divide', 'division'
Examples: 'ncbo --op_typ=/ 1.nc 2.nc 3.nc', 'ncdivide 1.nc 2.nc 3.nc'

Care should be taken when using the shortest form of key values, i.e., '+', '-', '*', '/'. Some of these single characters may have special meanings to the shell [1]. They should be

[1] A naked (i.e., unprotected or unquoted) '*' is a wildcard character. A naked '-' may interpretedconfuse command line parser. '+' and '/' are relatively harmless.

protected from the shell by placing them in quotes so that the shell does not attempt to interpret (glob) them[2]. For example, the following commands are equivalent

```
ncbo --op_typ=* 1.nc 2.nc 3.nc # Dangerous? (shell may attempt globbing)
ncbo --op_typ='*' 1.nc 2.nc 3.nc # Safe ('*' protected from shell)
ncbo --op_typ="*" 1.nc 2.nc 3.nc # Safe ('*' protected from shell)
ncbo --op_typ=mlt 1.nc 2.nc 3.nc
ncbo --op_typ=mult 1.nc 2.nc 3.nc
ncbo --op_typ=multiply 1.nc 2.nc 3.nc
ncbo --op_typ=multiplication 1.nc 2.nc 3.nc
ncmult 1.nc 2.nc 3.nc # First use ln -s ncbo ncmult
ncmultiply 1.nc 2.nc 3.nc # First use ln -s ncbo ncmult
```

No particular argument or invocation form is preferred. Users are encouraged to use the forms which are most intuitive to them.

Normally, an operation type must be specified with '`-y`' or `ncbo` will fail. Exceptions to this rule may be created to suit that tastes of your particular site. For many years, `ncdiff` was the main binary file operator. As a result, many users prefer to continue invoking `ncdiff` rather than memorizing a new command ('`ncbo -y` *`sbt`*') which behaves identically to the old `ncdiff` command. The is much to be said for the simplicity of `ncdiff`. However, from a software maintenance standpoint, maintaining a distinct executable for each binary operation (e.g., `ncadd`) is untenable.

`ncbo` subtracts variables in *file_2* from the corresponding variables (those with the same name) in *file_1* and stores the results in *file_3*. Variables in *file_2* are *broadcast* to conform to the corresponding variable in *file_1* if necessary. Broadcasting a variable means creating data in non-existing dimensions from the data in existing dimensions. For example, a two dimensional variable in *file_2* can be subtracted from a four, three, or two (but not one or zero) dimensional variable (of the same name) in `file_1`. This functionality allows the user to compute anomalies from the mean. Note that variables in *file_1* are *not* broadcast to conform to the dimensions in *file_2*. Thus, `ncbo`, the number of dimensions, or *rank*, of any processed variable in *file_1* must be greater than or equal to the rank of the same variable in *file_2*. Furthermore, the size of all dimensions common to both *file_1* and *file_2* must be equal.

When computing anomalies from the mean it is often the case that *file_2* was created by applying an averaging operator to a file with initially the same dimensions as *file_1* (often *file_1* itself). In these cases, creating *file_2* with `ncra` rather than `ncwa` will cause the `ncbo` operation to fail. For concreteness say the record dimension in `file_1` is `time`. If *file_2* were created by averaging *file_1* over the `time` dimension with the `ncra` operator rather than with the `ncwa` operator, then *file_2* will have a `time` dimension of size 1 rather than having no `time` dimension at all [3]. In this case the input files to `ncbo`, *file_1* and *file_2*,

[2] The widely used shell Bash correctly interprets all these special characters even when they are not quoted. That is, Bash does not prevent NCO from correctly interpreting the intended arithmetic operation when the following arguments are given (without quotes) to `ncbo`: '`--op_typ=+`', '`--op_typ=-`', '`--op_typ=*`', and '`--op_typ=/`'

[3] This is because `ncra` collapses the record dimension to a size of 1 (making it a *degenerate* dimension), but does not remove it, while `ncwa` removes all dimensions it averages over. In other words, `ncra` changes the size but not the rank of variables, while `ncwa` changes both the size and the rank of variables.

will have unequally sized `time` dimensions which causes `ncbo` to fail. To prevent this from occuring, use `ncwa` to remove the `time` dimension from *file_2*. An example is given below.

`ncbo` will never difference coordinate variables or variables of type `NC_CHAR` or `NC_BYTE`. This ensures that coordinates like (e.g., latitude and longitude) are physically meaningful in the output file, *file_3*. This behavior is hardcoded. `ncbo` applies special rules to some NCAR CSM fields (e.g., `ORO`). See Section 3.18 [NCAR CSM Conventions], page 38 for a complete description. Finally, we note that `ncflint` (see Section 4.6 [ncflint netCDF File Interpolator], page 59) is designed for file interpolation. As such, it also performs file subtraction, addition, multiplication, albeit in a more convoluted way than `ncbo`.

EXAMPLES

Say files '`85_0112.nc`' and '`86_0112.nc`' each contain 12 months of data. Compute the change in the monthly averages from 1985 to 1986:

```
ncbo -op_typ=sub 86_0112.nc 85_0112.nc 86m85_0112.nc
ncdiff 86_0112.nc 85_0112.nc 86m85_0112.nc
```

The following examples demonstrate the broadcasting feature of `ncbo`. Say we wish to compute the monthly anomalies of `T` from the yearly average of `T` for the year 1985. First we create the 1985 average from the monthly data, which is stored with the record dimension `time`.

```
ncra 85_0112.nc 85.nc
ncwa -O -a time 85.nc 85.nc
```

The second command, `ncwa`, gets rid of the `time` dimension of size 1 that `ncra` left in '`85.nc`'. Now none of the variables in '`85.nc`' has a `time` dimension. A quicker way to accomplish this is to use `ncwa` from the beginning:

```
ncwa -a time 85_0112.nc 85.nc
```

We are now ready to use `ncbo` to compute the anomalies for 1985:

```
ncdiff -v T 85_0112.nc 85.nc t_anm_85_0112.nc
```

Each of the 12 records in '`t_anm_85_0112.nc`' now contains the monthly deviation of `T` from the annual mean of `T` for each gridpoint.

Say we wish to compute the monthly gridpoint anomalies from the zonal annual mean. A *zonal mean* is a quantity that has been averaged over the longitudinal (or x) direction. First we use `ncwa` to average over longitudinal direction `lon`, creating '`85_x.nc`', the zonal mean of '`85.nc`'. Then we use `ncbo` to subtract the zonal annual means from the monthly gridpoint data:

```
ncwa -a lon 85.nc 85_x.nc
ncdiff 85_0112.nc 85_x.nc tx_anm_85_0112.nc
```

This examples works assuming '`85_0112.nc`' has dimensions `time` and `lon`, and that '`85_x.nc`' has no `time` or `lon` dimension.

As a final example, say we have five years of monthly data (i.e., 60 months) stored in '`8501_8912.nc`' and we wish to create a file which contains the twelve month seasonal cycle of the average monthly anomaly from the five-year mean of this data. The following method is just one permutation of many which will accomplish the same result. First use `ncwa` to create the file containing the five-year mean:

```
ncwa -a time 8501_8912.nc 8589.nc
```

Next use `ncbo` to create a file containing the difference of each month's data from the five-year mean:

```
ncbo 8501_8912.nc 8589.nc t_anm_8501_8912.nc
```

Now use `ncks` to group the five January anomalies together in one file, and use `ncra` to create the average anomaly for all five Januarys. These commands are embedded in a shell loop so they are repeated for all twelve months:

```
for idx in 01 02 03 04 05 06 07 08 09 10 11 12; do # Bourne Shell
ncks -F -d time,${idx},,12 t_anm_8501_8912.nc foo.${idx}
ncra foo.${idx} t_anm_8589_${idx}.nc
done
foreach idx (01 02 03 04 05 06 07 08 09 10 11 12) # C Shell
ncks -F -d time,${idx},,12 t_anm_8501_8912.nc foo.${idx}
ncra foo.${idx} t_anm_8589_${idx}.nc
end
```

Note that `ncra` understands the `stride` argument so the two commands inside the loop may be combined into the single command

```
ncra -F -d time,${idx},,12 t_anm_8501_8912.nc foo.${idx}
```

Finally, use `ncrcat` to concatenate the 12 average monthly anomaly files into one twelve-record file which contains the entire seasonal cycle of the monthly anomalies:

```
ncrcat t_anm_8589_??.nc t_anm_8589_0112.nc
```

4.4 ncea netCDF Ensemble Averager

SYNTAX

```
ncea [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]]] [-F] [-h] [-l path]
[-n loop] [-O] [-p path] [-R] [-r] [-v var[,...]]
[-x] [-y op_typ] input-files output-file
```

DESCRIPTION

`ncea` performs gridpoint averages of variables across an arbitrary number (an *ensemble*) of input files, with each file receiving an equal weight in the average. Each variable in the *output-file* will be the same size as the same variable in any one of the in the *input-files*, and all *input-files* must be the same size. Whereas `ncra` only performs averages over the record dimension (e.g., time), and weights each record in the record dimension evenly, `ncea` averages entire files, and weights each file evenly. All dimensions, including the record dimension, are treated identically and preserved in the *output-file*. See Section 2.6 [Averaging vs. Concatenating], page 13, for a description of the distinctions between the various averagers and concatenators.

The file is the logical unit of organization for the results of many scientific studies. Often one wishes to generate a file which is the gridpoint average of many separate files. This may be to reduce statistical noise by combining the results of a large number of experiments, or it may simply be a step in a procedure whose goal is to compute anomalies from a mean state. In any case, when one desires to generate a file whose properties are the mean of all the input files, then `ncea` is the operator to use. `ncea` assumes coordinate variable are properties common to all of the experiments and so does not average them across files. Instead, `ncea` copies the values of the coordinate variables from the first input file to the output file.

EXAMPLES

Consider a model experiment which generated five realizations of one year of data, say 1985. You can imagine that the experimenter slightly perturbs the initial conditions of the problem before generating each new solution. Assume each file contains all twelve months (a seasonal cycle) of data and we want to produce a single file containing the ensemble average (mean) seasonal cycle. Here the numeric filename suffix denotes the experiment number (*not* the month):

```
ncea 85_01.nc 85_02.nc 85_03.nc 85_04.nc 85_05.nc 85.nc
ncea 85_0[1-5].nc 85.nc
ncea -n 5,2,1 85_01.nc 85.nc
```

These three commands produce identical answers. See Section 3.2 [Specifying input files], page 18, for an explanation of the distinctions between these methods. The output file, '`85.nc`', is the same size as the inputs files. It contains 12 months of data (which might or might not be stored in the record dimension, depending on the input files), but each value in the output file is the average of the five values in the input files.

In the previous example, the user could have obtained the ensemble average values in a particular spatio-temporal region by adding a hyperslab argument to the command, e.g.,

```
ncea -d time,0,2 -d lat,-23.5,23.5 85_??.nc 85.nc
```

In this case the output file would contain only three slices of data in the *time* dimension. These three slices are the average of the first three slices from the input files. Additionally, only data inside the tropics is included.

4.5 `ncecat` netCDF Ensemble Concatenator

SYNTAX

```
ncecat [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]]] [-F] [-h] [-l path]
[-n loop] [-O] [-p path] [-R] [-r] [-v var[,...]]
[-x] input-files output-file
```

DESCRIPTION

`ncecat` concatenates an arbitrary number of input files into a single output file. Input files are glued together by creating a record dimension in the output file. Input files must be the same size. Each input file is stored consecutively as a single record in the output file. Thus, the size of the output file is the sum of the sizes of the input files. See Section 2.6 [Averaging vs. Concatenating], page 13, for a description of the distinctions between the various averagers and concatenators.

Consider five realizations, '`85a.nc`', '`85b.nc`', ... '`85e.nc`' of 1985 predictions from the same climate model. Then `ncecat 85?.nc 85_ens.nc` glues the individual realizations together into the single file, '`85_ens.nc`'. If an input variable was dimensioned [`lat`,`lon`], it will have dimensions [`record`,`lat`,`lon`] in the output file. A restriction of `ncecat` is that the hyperslabs of the processed variables must be the same from file to file. Normally this means all the input files are the same size, and contain data on different realizations of the same variables.

EXAMPLES

Consider a model experiment which generated five realizations of one year of data, say 1985. You can imagine that the experimenter slightly perturbs the initial conditions of the problem before generating each new solution. Assume each file contains all twelve months (a seasonal cycle) of data and we want to produce a single file containing all the seasonal cycles. Here the numeric filename suffix denotes the experiment number (*not* the month):

```
ncecat 85_01.nc 85_02.nc 85_03.nc 85_04.nc 85_05.nc 85.nc
ncecat 85_0[1-5].nc 85.nc
ncecat -n 5,2,1 85_01.nc 85.nc
```

These three commands produce identical answers. See Section 3.2 [Specifying input files], page 18, for an explanation of the distinctions between these methods. The output file, '`85.nc`', is five times the size as a single *input-file*. It contains 60 months of data (which might or might not be stored in the record dimension, depending on the input files).

4.6 ncflint netCDF File Interpolator

SYNTAX

```
ncflint [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]]] [-F] [-h]
[-i var,val3]
[-l path] [-O] [-p path] [-R] [-r] [-v var[,...]]
[-w wgt1[,wgt2]] [-x] file_1 file_2 file_3
```

DESCRIPTION

`ncflint` creates an output file that is a linear combination of the input files. This linear combination can be a weighted average, a normalized weighted average, or an interpolation of the input files. Coordinate variables are not acted upon in any case, they are simply copied from *file_1*.

There are two conceptually distinct methods of using `ncflint`. The first method is to specify the weight each input file is to have in the output file. In this method, the value *val3* of a variable in the output file *file_3* is determined from its values *val1* and *val2* in the two input files according to $val3 = wgt1 \times val1 + wgt2 \times val2$. Here at least *wgt1*, and, optionally, *wgt2*, are specified on the command line with the '`-w`' (or '`--weight`' or '`--wgt_var`') switch. If only *wgt1* is specified then *wgt2* is automatically computed as *wgt2* = 1-*wgt1*. Note that weights larger than 1 are allowed. Thus it is possible to specify *wgt1* = 2 and *wgt2* = −3. One can use this functionality to multiply all the values in a given file by a constant.

The second method of using `ncflint` is to specify the interpolation option with '`-i`' (or with the '`--ntp`' or '`--interpolate`' long options). This is really the inverse of the first method in the following sense. When the user specifies the weights directly, `ncflint` has no work to do besides multiplying the input values by their respective weights and adding the results together to produce the output values. This assumes it is the weights that are known a priori. In another class of cases it is the *arrival value* (i.e., *val3*) of a particular variable *var* that is known a priori. In this case, the implied weights can always be inferred by examining the values of *var* in the input files. This results in one equation in two unknowns, *wgt1* and *wgt2*: $val3 = wgt1 \times val1 + wgt2 \times val2$. Unique determination of the weights requires imposing the additional constraint of normalization on the weights: *wgt1* + *wgt2* = 1. Thus, to use the interpolation option, the user specifies *var* and *val3* with the '`-i`' option. `ncflint` will compute *wgt1* and *wgt2*, and use these weights on all variables to generate the output file. Although *var* may have any number of dimensions in the input files, it must represent a single, scalar value. Thus any dimensions associated with *var* must be *degenerate*, i.e., of size one.

If neither '`-i`' nor '`-w`' is specified on the command line, `ncflint` defaults to weighting each input file equally in the output file. This is equivalent to specifying '`-w 0.5`' or '`-w 0.5,0.5`'. Attempting to specify both '`-i`' and '`-w`' methods in the same command is an error.

`ncflint` is programmed not to interpolate variables of type `NC_CHAR` and `NC_BYTE`. This behavior is hardcoded.

EXAMPLES

Although it has other uses, the interpolation feature was designed to interpolate *file_3* to a time between existing files. Consider input files '85.nc' and '87.nc' containing variables describing the state of a physical system at times time = 85 and time = 87. Assume each file contains its timestamp in the scalar variable time. Then, to linearly interpolate to a file '86.nc' which describes the state of the system at time at time = 86, we would use

```
ncflint -i time,86 85.nc 87.nc 86.nc
```

Say you have observational data covering January and April 1985 in two files named '85_01.nc' and '85_04.nc', respectively. Then you can estimate the values for February and March by interpolating the existing data as follows. Combine '85_01.nc' and '85_04.nc' in a 2:1 ratio to make '85_02.nc':

```
ncflint -w 0.667 85_01.nc 85_04.nc 85_02.nc
ncflint -w 0.667,0.333 85_01.nc 85_04.nc 85_02.nc
```

Multiply '85.nc' by 3 and by −2 and add them together to make 'tst.nc':

```
ncflint -w 3,-2 85.nc 85.nc tst.nc
```

This is an example of a null operation, so 'tst.nc' should be identical (within machine precision) to '85.nc'.

Add '85.nc' to '86.nc' to obtain '85p86.nc', then subtract '86.nc' from '85.nc' to obtain '85m86.nc'

```
ncflint -w 1,1 85.nc 86.nc 85p86.nc
ncflint -w 1,-1 85.nc 86.nc 85m86.nc
ncdiff 85.nc 86.nc 85m86.nc
```

Thus ncflint can be used to mimic some ncbo operations. However this is not a good idea in practice because ncflint does not broadcast (see Section 4.3 [ncbo netCDF Binary Operator], page 52) conforming variables during arithmetic. Thus the final two commands would produce identical results except that ncflint would fail if any variables needed to be broadcast.

Rescale the dimensional units of the surface pressure prs_sfc from Pascals to hectopascals (millibars)

```
ncflint -O -C -v prs_sfc -w 0.01,0.0 in.nc in.nc out.nc
ncatted -O -a units,prs_sfc,o,c,millibar out.nc
```

4.7 ncks netCDF Kitchen Sink

SYNTAX

```
ncks [-A] [-a] [-B] [-b binary-file] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]][,[stride]]]
[-F] [-H] [-h] [-l path] [-M] [-m] [-O] [-p path] [-q]
[-R] [-r] [-s format] [-u] [-v var[,...]] [-x]
input-file [output-file]
```

DESCRIPTION

`ncks` combines selected features of `ncdump`, `ncextr`, and the nccut and ncpaste specifications into one versatile utility. `ncks` extracts a subset of the data from *input-file* and prints it as ASCII text to '`stdout`', writes it in flat binary format to '`binary-file`', and writes (or pastes) it in netCDF format to *output-file*.

`ncks` will print netCDF data in ASCII format to `stdout`, like `ncdump`, but with these differences: `ncks` prints data in a tabular format intended to be easy to search for the data you want, one datum per screen line, with all dimension subscripts and coordinate values (if any) preceding the datum. Option '`-s`' (or lon options '`--sng`', '`--string`', '`--fmt`', or '`--format`') allows the user the format the data using C-style format strings.

Options '`-a`', '`-F`' , '`-H`', '`-M`', '`-m`', '`-q`', '`-s`', and '`-u`' (and their long option counterparts) control the formatted appearance of the data.

`ncks` will extract (and optionally create a new netCDF file comprised of) only selected variable from the input file, like `ncextr` but with these differences: Only variables and coordinates may be specifically included or excluded—all global attributes and any attribute associated with an extracted variable will be copied to the screen and/or output netCDF file. Options '`-c`', '`-C`', '`-v`', and '`-x`' (and their long option synomyms) control which variables are extracted.

`ncks` will extract hyperslabs from the specified variables. In fact `ncks` implements the nccut specification exactly. Option '`-d`' controls the hyperslab specification.

Input dimensions that are not associated with any output variable will not appear in the output netCDF. This feature removes superfluous dimensions from a netCDF file.

`ncks` will append variables and attributes from the *input-file* to *output-file* if *output-file* is a pre-existing netCDF file whose relevant dimensions conform to dimension sizes of *input-file*. The append features of `ncks` are intended to provide a rudimentary means of adding data from one netCDF file to another, conforming, netCDF file. When naming conflicts exists between the two files, data in *output-file* is usually overwritten by the corresponding data from *input-file*. Thus it is recommended that the user backup *output-file* in case valuable data are accidentally overwritten.

If *output-file* exists, the user will be queried whether to *overwrite*, *append*, or *exit* the `ncks` call completely. Choosing *overwrite* destroys the existing *output-file* and create an entirely new one from the output of the `ncks` call. Append has differing effects depending on the uniqueness of the variables and attributes output by `ncks`: If a variable or attribute extracted from *input-file* does not have a name conflict with the members of *output-file* then it will be added to *output-file* without overwriting any of the existing contents of *output-file*. In this case the relevant dimensions must agree (conform) between the two files; new dimensions are created in *output-file* as required. When a name conflict occurs, a global attribute from *input-file* will overwrite the corresponding global attribute from *output-file*. If the name conflict occurs for a non-record variable, then the dimensions and type of the variable (and of its coordinate dimensions, if any) must agree (conform) in both files. Then the variable values (and any coordinate dimension values) from *input-file* will overwrite the corresponding variable values (and coordinate dimension values, if any) in *output-file*[1].

Since there can only be one record dimension in a file, the record dimension must have the same name (but not necessarily the same size) in both files if a record dimension variable is to be appended. If the record dimensions are of differing sizes, the record dimension of *output-file* will become the greater of the two record dimension sizes, the record variable from *input-file* will overwrite any counterpart in *output-file* and fill values will be written to any gaps left in the rest of the record variables (I think). In all cases variable attributes in *output-file* are superseded by attributes of the same name from *input-file*, and left alone if there is no name conflict.

Some users may wish to avoid interactive `ncks` queries about whether to overwrite existing data. For example, batch scripts will fail if `ncks` does not receive responses to its queries. Options '`-O`' and '`-A`' are available to force overwriting existing files and variables, respectively.

Options specific to `ncks`

The following list provides a short summary of the features unique to `ncks`. Features common to many operators are described in Chapter 3 [Common features], page 17.

'`-a`' Do not alphabetize extracted fields. By default, the specified output variables are extracted, printed, and written to disk in alphabetical order. This tends to make long output lists easier to search for particular variables. Specifying `-a` results in the variables being extracted, printed, and written to disk in the order in which they were saved in the input file. Thus `-a` retains the original ordering of the variables. Also '`--abc`' and '`--alphabetize`'.

'`-B 'file'`'
Activate native machine binary output writing to the default binary file, '`ncks.bnr`'. The `-B` switch is redundant when the `-b` '`file`' option is specified, and native binary output will be directed to the binary file '`file`'. Also '`--bnr`' and '`--binary`'. Writing packed variables in binary format is not supported.

[1] Those familiar with netCDF mechanics might wish to know what is happening here: `ncks` does not attempt to redefine the variable in *output-file* to match its definition in *input-file*, `ncks` merely copies the values of the variable and its coordinate dimensions, if any, from *input-file* to *output-file*.

'-b 'file''
: Activate native machine binary output writing to binary file 'file'. Also '--fl_bnr' and '--binary-file'. Writing packed variables in binary format is not supported.

'-d *dim*,[*min*][,[*max*]][,[*stride*]]'
: Add *stride* argument to hyperslabber. For a complete description of the *stride* argument, See Section 3.12 [Stride], page 29.

'-H'
: Print data to screen. Also activated using '--print' or '--prn'. Unless otherwise specified (with -s), each element of the data hyperslab is printed on a separate line containing the names, indices, and, values, if any, of all of the variables dimensions. The dimension and variable indices refer to the location of the corresponding data element with respect to the variable as stored on disk (i.e., not the hyperslab).

```
% ncks -H -C -v three_dmn_var in.nc
lat[0]=-90 lev[0]=100 lon[0]=0 three_dmn_var[0]=0
lat[0]=-90 lev[0]=100 lon[1]=90 three_dmn_var[1]=1
lat[0]=-90 lev[0]=100 lon[2]=180 three_dmn_var[2]=2
...
lat[1]=90 lev[2]=1000 lon[1]=90 three_dmn_var[21]=21
lat[1]=90 lev[2]=1000 lon[2]=180 three_dmn_var[22]=22
lat[1]=90 lev[2]=1000 lon[3]=270 three_dmn_var[23]=23
```

Printing the same variable with the '-F' option shows the same variable indexed with Fortran conventions

```
% ncks -F -H -C -v three_dmn_var in.nc
lon(1)=0 lev(1)=100 lat(1)=-90 three_dmn_var(1)=0
lon(2)=90 lev(1)=100 lat(1)=-90 three_dmn_var(2)=1
lon(3)=180 lev(1)=100 lat(1)=-90 three_dmn_var(3)=2
...
```

Printing a hyperslab does not affect the variable or dimension indices since these indices are relative to the full variable (as stored in the input file), and the input file has not changed. However, if the hypserslab is saved to an output file and those values are printed, the indices will change:

```
% ncks -O -H -d lat,90.0 -d lev,1000.0 -v three_dmn_var in.nc out.nc
...
lat[1]=90 lev[2]=1000 lon[0]=0 three_dmn_var[20]=20
lat[1]=90 lev[2]=1000 lon[1]=90 three_dmn_var[21]=21
lat[1]=90 lev[2]=1000 lon[2]=180 three_dmn_var[22]=22
lat[1]=90 lev[2]=1000 lon[3]=270 three_dmn_var[23]=23
% ncks -C -H -v three_dmn_var out.nc
lat[0]=90 lev[0]=1000 lon[0]=0 three_dmn_var[0]=20
lat[0]=90 lev[0]=1000 lon[1]=90 three_dmn_var[1]=21
lat[0]=90 lev[0]=1000 lon[2]=180 three_dmn_var[2]=22
lat[0]=90 lev[0]=1000 lon[3]=270 three_dmn_var[3]=23
```

'-M'
: Print to screen the global metadata describing the file. This includes file summary information and global attributes. Also '--Mtd' and '--Metadata'.

'-m' Print variable metadata to screen (similar to `ncdump -h`). This displays all metadata pertaining to each variable, one variable at a time. Also '--mtd' and '--metadata'.

'-q' Toggle printing of dimension indices and coordinate values when printing arrays. The name of each variable will appear flush left in the output. This is useful when trying to locate specific variables when displaying many variables with different dimensions. Also '--quiet'.

'-s *format*'
String format for text output. Accepts C language escape sequences and `printf()` formats. Also '--string', '--format', and '--fmt'.

'-u' Accompany the printing of a variable's values with its `units` attribute, if any. Also '--units'.

EXAMPLES

View all data in netCDF 'in.nc', printed with Fortran indexing conventions:

```
ncks -H -F in.nc
```

Copy the netCDF file 'in.nc' to file 'out.nc'.

```
ncks -O in.nc out.nc
```

Now the file 'out.nc' contains all the data from 'in.nc'. There are, however, two differences between 'in.nc' and 'out.nc'. First, the `history` global attribute (see Section 3.17 [History attribute], page 37) will contain the command used to create 'out.nc'. Second, the variables in 'out.nc' will be defined in alphabetical order. Of course the internal storage of variable in a netCDF file should be transparent to the user, but there are cases when alphabetizing a file is useful (see description of `-a` switch).

Print variable `three_dmn_var` from file 'in.nc' with default notations. Next print `three_dmn_var` as an un-annotated text column. Then print `three_dmn_var` signed with very high precision. Finally, print `three_dmn_var` as a comma-separated list.

```
% ncks -H -C -v three_dmn_var in.nc
lat[0]=-90 lev[0]=100 lon[0]=0 three_dmn_var[0]=0
lat[0]=-90 lev[0]=100 lon[1]=90 three_dmn_var[1]=1
...
lat[1]=90 lev[2]=1000 lon[3]=270 three_dmn_var[23]=23
% ncks -s "%f\n" -H -C -v three_dmn_var in.nc
0.000000
1.000000
...
23.000000
% ncks -s "%+16.10f\n" -H -C -v three_dmn_var in.nc
   +0.0000000000
   +1.0000000000
...
  +23.0000000000
% ncks -s "%f, " -H -C -v three_dmn_var in.nc
```

```
0.000000, 1.000000, ..., 23.000000,
```

The second and third options are useful when pasting data into text files like reports or papers. See Section 4.2 [ncatted netCDF Attribute Editor], page 48, for more details on string formatting and special characters.

One dimensional arrays of characters stored as netCDF variables are automatically printed as strings, whether or not they are NUL-terminated, e.g.,

```
ncks -v fl_nm in.nc
```

The `%c` formatting code is useful for printing multidimensional arrays of characters representing fixed length strings

```
ncks -H -s "%c" -v fl_nm_arr in.nc
```

Using the `%s` format code on strings which are not NUL-terminated (and thus not technically strings) is likely to result in a core dump.

Create netCDF '`out.nc`' containing all variables, and any associated coordinates, except variable `time`, from netCDF '`in.nc`':

```
ncks -x -v time in.nc out.nc
```

Extract variables `time` and `pressure` from netCDF '`in.nc`'. If '`out.nc`' does not exist it will be created. Otherwise the you will be prompted whether to append to or to overwrite '`out.nc`':

```
ncks -v time,pressure in.nc out.nc
ncks -C -v time,pressure in.nc out.nc
```

The first version of the command creates an '`out.nc`' which contains `time`, `pressure`, and any coordinate variables associated with *pressure*. The '`out.nc`' from the second version is guaranteed to contain only two variables `time` and `pressure`.

Create netCDF '`out.nc`' containing all variables from file '`in.nc`'. Restrict the dimensions of these variables to a hyperslab. Print (with `-H`) the hyperslabs to the screen for good measure. The specified hyperslab is: the fifth value in dimension `time`; the half-open range $lat > 0.$ in coordinate `lat`; the half-open range $lon < 330.$ in coordinate `lon`; the closed interval $0.3 < band < 0.5$ in coordinate `band`; and cross-section closest to 1000. in coordinate `lev`. Note that limits applied to coordinate values are specified with a decimal point, and limits applied to dimension indices do not have a decimal point See Section 3.8 [Hyperslabs], page 24.

```
ncks -H -d time,5 -d lat,,0.0 -d lon,330.0, -d band,0.3,0.5
-d lev,1000.0 in.nc out.nc
```

Assume the domain of the monotonically increasing longitude coordinate `lon` is $0 < lon < 360$. Here, `lon` is an example of a wrapped coordinate. `ncks` will extract a hyperslab which crosses the Greenwich meridian simply by specifying the westernmost longitude as *min* and the easternmost longitude as *max*, as follows:

```
ncks -d lon,260.0,45.0 in.nc out.nc
```

For more details See Section 3.11 [Wrapped coordinates], page 28.

4.8 ncra netCDF Record Averager

SYNTAX

```
ncra [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]][,[stride]]] [-F] [-h] [-l path]
[-n loop] [-O] [-p path] [-R] [-r] [-v var[,...]]
[-x] [-y op_typ] input-files output-file
```

DESCRIPTION

`ncra` averages record variables across an arbitrary number of input files. The record dimension is retained as a degenerate (size 1) dimension in the output variables. See Section 2.6 [Averaging vs. Concatenating], page 13, for a description of the distinctions between the various averagers and concatenators.

Input files may vary in size, but each must have a record dimension. The record coordinate, if any, should be monotonic for (or else non-fatal warnings may be generated). Hyperslabs of the record dimension which include more than one file are handled correctly. `ncra` supports the *stride* argument to the '`-d`' hyperslab option for the record dimension only, *stride* is not supported for non-record dimensions.

`ncra` weights each record (e.g., time slice) in the *input-files* equally. `ncra` does not attempt to see if, say, the `time` coordinate is irregularly spaced and thus would require a weighted average in order to be a true time average.

EXAMPLES

Average files '`85.nc`', '`86.nc`', ... '`89.nc`' along the record dimension, and store the results in '`8589.nc`':

```
ncra 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra 8[56789].nc 8589.nc
ncra -n 5,2,1 85.nc 8589.nc
```

These three methods produce identical answers. See Section 3.2 [Specifying input files], page 18, for an explanation of the distinctions between these methods.

Assume the files '`85.nc`', '`86.nc`', ... '`89.nc`' each contain a record coordinate *time* of length 12 defined such that the third record in '`86.nc`' contains data from March 1986, etc. NCO knows how to hyperslab the record dimension across files. Thus, to average data from December, 1985 through February, 1986:

```
ncra -d time,11,13 85.nc 86.nc 87.nc 8512_8602.nc
ncra -F -d time,12,14 85.nc 86.nc 87.nc 8512_8602.nc
```

The file '`87.nc`' is superfluous, but does not cause an error. The '`-F`' turns on the Fortran (1-based) indexing convention. The following uses the *stride* option to average all the March temperature data from multiple input files into a single output file

```
ncra -F -d time,3,,12 -v temperature 85.nc 86.nc 87.nc 858687_03.nc
```

See Section 3.12 [Stride], page 29, for a description of the *stride* argument.

Assume the *time* coordinate is incrementally numbered such that January, 1985 = 1 and December, 1989 = 60. Assuming '`??`' only expands to the five desired files, the following averages June, 1985–June, 1989:

```
ncra -d time,6.,54. ??.nc 8506_8906.nc
```

4.9 `ncrcat` netCDF Record Concatenator

SYNTAX

```
ncrcat [-A] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]][,[stride]]] [-F] [-h] [-l path]
[-n loop] [-O] [-p path] [-R] [-r] [-v var[,...]]
[-x] input-files output-file
```

DESCRIPTION

`ncrcat` concatenates record variables across an arbitrary number of input files. The final record dimension is by default the sum of the lengths of the record dimensions in the input files. See Section 2.6 [Averaging vs. Concatenating], page 13, for a description of the distinctions between the various averagers and concatenators.

Input files may vary in size, but each must have a record dimension. The record coordinate, if any, should be monotonic (or else non-fatal warnings may be generated). Hyperslabs of the record dimension which include more than one file are handled correctly. `ncra` supports the *stride* argument to the '`-d`' hyperslab option for the record dimension only, *stride* is not supported for non-record dimensions.

`ncrcat` applies special rules to ARM convention time fields (e.g., `time_offset`). See Section 3.19 [ARM Conventions], page 38 for a complete description.

EXAMPLES

Concatenate files '`85.nc`', '`86.nc`', ... '`89.nc`' along the record dimension, and store the results in '`8589.nc`':

```
ncrcat 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncrcat 8[56789].nc 8589.nc
ncrcat -n 5,2,1 85.nc 8589.nc
```

These three methods produce identical answers. See Section 3.2 [Specifying input files], page 18, for an explanation of the distinctions between these methods.

Assume the files '`85.nc`', '`86.nc`', ... '`89.nc`' each contain a record coordinate *time* of length 12 defined such that the third record in '`86.nc`' contains data from March 1986, etc. NCO knows how to hyperslab the record dimension across files. Thus, to concatenate data from December, 1985–February, 1986:

```
ncrcat -d time,11,13 85.nc 86.nc 87.nc 8512_8602.nc
ncrcat -F -d time,12,14 85.nc 86.nc 87.nc 8512_8602.nc
```

The file '`87.nc`' is superfluous, but does not cause an error. The '`-F`' turns on the Fortran (1-based) indexing convention.

The following uses the *stride* option to concatenate all the March temperature data from multiple input files into a single output file

```
ncrcat -F -d time,3,,12 -v temperature 85.nc 86.nc 87.nc 858687_03.nc
```

See Section 3.12 [Stride], page 29, for a description of the *stride* argument.

Assume the *time* coordinate is incrementally numbered such that January, 1985 = 1 and December, 1989 = 60. Assuming `??` only expands to the five desired files, the following concatenates June, 1985–June, 1989:

```
ncrcat -d time,6.,54. ??.nc 8506_8906.nc
```

4.10 ncrename netCDF Renamer

SYNTAX

```
ncrename [-a old_name,new_name] [-a ...] [-D dbg]
[-d old_name,new_name] [-d ...] [-h] [-l path] [-O] [-p path]
[-R] [-r] [-v old_name,new_name] [-v ...]
input-file [output-file]
```

DESCRIPTION

`ncrename` renames dimensions, variables, and attributes in a netCDF file. Each object that has a name in the list of old names is renamed using the corresponding name in the list of new names. All the new names must be unique. Every old name must exist in the input file, unless the old name is preceded by the character '`.`'. The validity of *old_name* is not checked prior to the renaming. Thus, if *old_name* is specified without the the '`.`' prefix and is not present in *input-file*, `ncrename` will abort. The *new_name* should never be prefixed by a '`.`' (the period will be included as part of the new name). The OPTIONS and EXAMPLES show how to select specific variables whose attributes are to be renamed.

`ncrename` is the exception to the normal rules that the user will be interactively prompted before an existing file is changed, and that a temporary copy of an output file is constructed during the operation. If only *input-file* is specified, then `ncrename` will change the names of the *input-file* in place without prompting and without creating a temporary copy of `input-file`. This is because the renaming operation is considered reversible if the user makes a mistake. The *new_name* can easily be changed back to *old_name* by using `ncrename` one more time.

Note that renaming a dimension to the name of a dependent variable can be used to invert the relationship between an independent coordinate variable and a dependent variable. In this case, the named dependent variable must be one-dimensional and should have no missing values. Such a variable will become a coordinate variable.

According to the *netCDF User's Guide*, renaming properties in netCDF files does not incur the penalty of recopying the entire file when the *new_name* is shorter than the *old_name*.

OPTIONS

'`-a old_name,new_name`'
: Attribute renaming. The old and new names of the attribute are specified by the associated *old_name* and *new_name* values. Global attributes are treated no differently than variable attributes. This option may be specified more than once. As mentioned above, all occurrences of the attribute of a given name will be renamed unless the '`.`' form is used, with one exception. To change the attribute name for a particular variable, specify the *old_name* in the format *old_var_name@old_att_name*. The '`@`' symbol serves to delimit the variable name from the attribute name. If the attribute is uniquely named (no other variables contain the attribute) then the *old_var_name@old_att_name* syntax is redundant. The *var_name@att_name* syntax is accepted, but not required, for the *new_name*.

'-d *old_name,new_name*'
: Dimension renaming. The old and new names of the dimension are specified by the associated *old_name* and *new_name* values. This option may be specified more than once.

'-v *old_name,new_name*'
: Variable renaming. The old and new names of the variable are specified by the associated *old_name* and *new_name* values. This option may be specified more than once.

EXAMPLES

Rename the variable `p` to `pressure` and `t` to `temperature` in netCDF '`in.nc`'. In this case `p` must exist in the input file (or `ncrename` will abort), but the presence of `t` is optional:

```
ncrename -v p,pressure -v .t,temperature in.nc
```

`ncrename` does not automatically attach dimensions to variables of the same name. If you want to rename a coordinate variable so that it remains a coordinate variable, you must separately rename both the dimension and the variable:

```
ncrename -d lon,longitude -v lon,longitude in.nc
```

Create netCDF '`out.nc`' identical to '`in.nc`' except the attribute `_FillValue` is changed to `missing_value` which possess it), the attribute `units` is renamed to `CGS_units` (but only in those variables which possess it) and the global attribute `Zaire` is renamed to `Congo`:

```
ncrename -a _FillValue,missing_value -a .units,CGS_units \
-a tpt@hieght,height -a prs_sfc@.hieght,height in.nc out.nc
```

The presence and absence of the '`.`' and '`@`' features cause this command to execute successfully only if a number of conditions are met. All variables *must* have a `_FillValue` attribute *and* `_FillValue` must also be a global attribute. The `units` attribute, on the other hand, will be renamed to `CGS_units` wherever it is found but need not be present in the file at all (either as a global or a variable attribute). The variable `tpt` must contain the `hieght` attribute. The variable `prs_sfc` need not exist, and need not contain the `hieght` attribute.

4.11 `ncwa` netCDF Weighted Averager

SYNTAX

```
ncwa [-A] [-a dim[,...]] [-C] [-c] [-D dbg]
[-d dim,[min][,[max]]] [-F] [-h] [-I] [-l path]
[-M mask_val] [-m mask_var] [-N] [-n] [-O] [-o condition]
[-p path] [-R] [-r] [-v var[,...]] [-W] [-w weight]
[-x] [-y op_typ] input-file output-file
```

DESCRIPTION

`ncwa` averages variables in a single file over arbitrary dimensions, with options to specify weights, masks, and normalization. See Section 2.6 [Averaging vs. Concatenating], page 13, for a description of the distinctions between the various averagers and concatenators. The default behavior of `ncwa` is to arithmetically average every numerical variable over all dimensions and produce a scalar result. To average variables over only a subset of their dimensions, specify these dimensions in a comma-separated list following '`-a`', e.g., '`-a time,lat,lon`'. As with all arithmetic operators, the operation may be restricted to an arbitrary hypserslab by employing the '`-d`' option (see Section 3.8 [Hyperslabs], page 24). `ncwa` also handles values matching the variable's `missing_value` attribute correctly. Moreover, `ncwa` understands how to manipulate user-specified weights, masks, and normalization options. With these options, `ncwa` can compute sophisticated averages (and integrals) from the command line.

mask_var and *weight*, if specified, are broadcast to conform to the variables being averaged. The rank of variables is reduced by the number of dimensions which they are averaged over. Thus arrays which are one dimensional in the *input-file* and are averaged by `ncwa` appear in the *output-file* as scalars. This allows the user to infer which dimensions may have been averaged. Note that that it is impossible for `ncwa` to make make a *weight* or *mask_var* of rank W conform to a *var* of rank V if $W > V$. This situation often arises when coordinate variables (which, by definition, are one dimensional) are weighted and averaged. `ncwa` assumes you know this is impossible and so `ncwa` does not attempt to broadcast *weight* or *mask_var* to conform to *var* in this case, nor does `ncwa` print a warning message telling you this, because it is so common. Specifying *dbg* > *2* does cause `ncwa` to emit warnings in these situations, however.

Non-coordinate variables are always masked and weighted if specified. Coordinate variables, however, may be treated specially. By default, an averaged coordinate variable, e.g., `latitude`, appears in *output-file* averaged the same way as any other variable containing an averaged dimension. In other words, by default `ncwa` weights and masks coordinate variables like all other variables. This design decision was intended to be helpful but for some applications it may be preferable not to weight or mask coordinate variables just like all other variables. Consider the following arguments to `ncwa`: `-a latitude -w lat_wgt -d latitude,0.,90.` where `lat_wgt` is a weight in the `latitude` dimension. Since, by default `ncwa` weights coordinate variables, the value of `latitude` in the *output-file* depends on the weights in *lat_wgt* and is not likely to be 45.0, the midpoint latitude of the hyperslab. Option '`-I`' overrides this default behavior and causes `ncwa` not to weight or mask coordinate variables[1]. In the above case, this causes the value of `latitude` in the *output-file* to be 45.0, an appealing result. Thus, '`-I`' specifies simple arithmetic averages for the coordinate variables. In the case of latitude, '`-I`' specifies that you prefer to archive the central latitude of the hyperslab over which variables were averaged rather than the area weighted centroid of the hyperslab[2]. The mathematical definition of operations involving rank reduction is given above (see Section 3.14 [Operation Types], page 31).

4.11.1 Masking condition

Each x_i also has an associated masking weight m_i whose value is 0 or 1 (false or true). The value of m_i is always 1 unless a *mask_var* is specified (with '`-m`'). As noted above, *mask_var* is broadcast, if possible, to conform to the variable being averaged. In this case, the value of m_i depends on the *masking condition*. As expected, $m_i = 1$ when the masking condition is *true* and $m_i = 0$ otherwise.

The masking condition has the syntax *mask_var condition mask_val*. Here *mask_var* is the name of the masking variable (specified with '`-m`', '`--mask-variable`', '`--mask_variable`', '`--msk_nm`', or '`--msk_var`'). The truth *condition* argument (specified with '`-o`', '`--op_rlt`', '`--cmp`', '`--compare`', or '`--op_cmp`' may be any one of the six arithmetic comparatives: *`eq`*, *`ne`*, *`gt`*, *`lt`*, *`ge`*, *`le`*. These are the Fortran-style character abbreviations for the logical operations $=$, $\neq$, $>$, $<$, $\geq$, $\leq$. The masking condition defaults to *`eq`* (equality). The *mask_val* argument to '`-M`' (or '`--mask-value`', or '`--msk_val`') is the right hand side of the *masking condition*. Thus for the *i*'th element of the hyperslab to be averaged, the masking condition is $mask_i$ *condition mask_val*.

Each x_i is also associated with an additional weight w_i whose value may be user-specified. The value of w_i is identically 1 unless the user specifies a weighting variable *weight* (with '`-w`', '`--weight`', or '`--wgt_var`'). In this case, the value of w_i is determined by the *weight* variable in the *input-file*. As noted above, *weight* is broadcast, if possible, to conform to the variable being averaged.

M is the number of input elements x_i which actually contribute to output element x_j. M is also known as the *tally* and is defined as

$$M = \sum_{i=1}^{i=N} \mu_i m_i$$

M is identical to the denominator of the generic averaging expression except for the omission of the weight w_i. Thus $M = N$ whenever no input points are missing values or are masked. Whether an element contributes to the output, and thus increments M by one, has more to do with the above two criteria (missing value and masking) than with the numeric value of the element per se. For example, $x_i = 0.0$ does contribute to x_j (assuming the `missing_value` attribute is not 0.0 and location i is not masked). The value $x_i = 0.0$ will not change the numerator of the generic averaging expression, but it will change the denominator (unless its weight $w_i = 0.0$ as well).

[1] The default behavior of ('`-I`') changed on 1998/12/01—before this date the default was not to weight or mask coordinate variables.

[2] If `lat_wgt` contains Gaussian weights then the value of `latitude` in the *output-file* will be the area-weighted centroid of the hyperslab. For the example given, this is about 30 degrees.

4.11.2 Normalization

`ncwa` has one switch which controls the normalization of the averages appearing in the *output-file*. Short option '`-N`' (or long options '`--nmr`' or '`--numerator`') prevents `ncwa` from dividing the weighted sum of the variable (the numerator in the averaging expression) by the weighted sum of the weights (the denominator in the averaging expression). Thus '`-N`' tells `ncwa` to return just the numerator of the arithmetic expression defining the operation (see Section 3.14 [Operation Types], page 31).

EXAMPLES

Given file '`85_0112.nc`':

```
netcdf 85_0112 {
dimensions:
        lat = 64 ;
        lev = 18 ;
        lon = 128 ;
        time = UNLIMITED ; // (12 currently)
variables:
        float lat(lat) ;
        float lev(lev) ;
        float lon(lon) ;
        float time(time) ;
        float scalar_var ;
        float three_dmn_var(lat, lev, lon) ;
        float two_dmn_var(lat, lev) ;
        float mask(lat, lon) ;
        float gw(lat) ;
}
```

Average all variables in '`in.nc`' over all dimensions and store results in '`out.nc`':

```
ncwa in.nc out.nc
```

Every variable in '`in.nc`' is reduced to a scalar in '`out.nc`' because, by default, averaging is performed over all dimensions.

Store the zonal (longitudinal) mean of '`in.nc`' in '`out.nc`':

```
ncwa -a lon in.nc out.nc
```

Here the tally is simply the size of `lon`, or 128.

Compute the meridional (latitudinal) mean, with values weighted by the corresponding element of *gw*[3]:

```
ncwa -w gw -a lat in.nc out.nc
```

Here the tally is simply the size of `lat`, or 64. The sum of the Gaussian weights is 2.0.

Compute the area mean over the tropical Pacific:

[3] `gw` stands for *Gaussian weight* in the NCAR climate model.

```
ncwa -w gw -a lat,lon -d lat,-20.,20. -d lon,120.,270.
in.nc out.nc
```

Here the tally is $64 \times 128 = 8192$.

Compute the area mean over the globe, but include only points for which $ORO < 0.5$[4]:

```
ncwa -m ORO -M 0.5 -o lt -w gw -a lat,lon in.nc out.nc
```

Assuming 70% of the gridpoints are maritime, then here the tally is $0.70 \times 8192 \approx 5734$.

Compute the global annual mean over the maritime tropical Pacific:

```
ncwa -m ORO -M 0.5 -o lt -w gw -a lat,lon,time
-d lat,-20.0,20.0 -d lon,120.0,270.0 in.nc out.nc
```

Determine the total area of the maritime tropical Pacific, assuming the variable *area* contains the area of each gridcell

```
ncwa -N -v area -m ORO -M 0.5 -o lt -a lat,lon
-d lat,-20.0,20.0 -d lon,120.0,270.0 in.nc out.nc
```

Weighting *area* (e.g., by *gw*) is not appropriate because *area* is *already* area-weighted by definition. Thus the '`-N`' switch, or, equivalently, the '`-y ttl`' switch, are all that are needed to correctly integrate the cell areas into a total regional area.

[4] `ORO` stands for *Orography* in the NCAR climate model. $ORO < 0.5$ selects the gridpoints which are covered by ocean.

5 Contributing

We welcome contributions from anyone. The NCO project homepage at `https://sf.net/projects/nco` contains more information on how to contribute.

Charlie Zender
: Concept, design and implementation of NCO from 1995–2000. Since then, mainly packing, NCO library redesign, `ncap` features, project coordination, code maintenance and porting, documentation, and `ncbo`.

Henry Butowsky
: Non-linear operations and `min()`, `max()`, `total()` support in `ncra` and `ncwa`. Type conversion for arithmetic. Migration to netCDF3 API. `ncap` parser, lexer, and I/O. Multislabbing algorithm. Various hacks.

Rorik Peterson
: Autotool build support, long options.

Brian Mays
: Packaging for Debian GNU/Linux, nroff man pages.

George Shapovalov
: Packaging for Gentoo GNU/Linux.

Bill Kocik
: Memory management.

Len Makin
: NECSX architecture support.

Jim Edwards
: AIX architecture support.

Juliana Rew
: Compatibility with large PIDs.

Keith Lindsay, Martin Dix
: Excellent bug reports.

General Index

"

" (double quote) 50

#

#include 43

%

% (modulus) 45

’

’ (end quote) 50

*

* 52
* (multiplication) 45

+

+ 52
+ (addition) 45

-

- 52
- (subtraction) 45
--abc 62
--alphabetize 62
--apn 12, 37
--append 12, 37
--binary 62
--bnr 62
--coords 23
--crd 23
--dbg_lvl *debug-level* 15, 17
--debug-level *debug-level* 15
--dimension *dim*,[*min*],[*max*],*stride* 29
--dimension *dim*,[*min*][,[*max*]] 24, 25, 26, 28
--dmn *dim*,[*min*],[*max*],*stride* 29
--dmn *dim*,[*min*][,[*max*]] 24, 25, 26, 28
--exclude 22
--fl_bnr 62
--fl_spt 42
--fmt 64
--fnc_tbl 46
--format 64
--fortran 24
--history 37
--hst 37
--lcl *output-path* 20
--local *output-path* 20
--mask-value *mask_val* 73
--mask-variable *mask_var* 72
--mask_value *mask_val* 73
--mask_variable *mask_var* 72
--metadata 63
--Metadata 63
--msk_nm *mask_var* 72
--msk_val *mask_val* 73
--msk_var *mask_var* 72
--mtd 63
--Mtd 63
--nintap *loop* 18
--no-coords 23
--no-crd 23
--op_typ *op_typ* 31, 52
--operation *op_typ* 31, 52
--overwrite 12, 37
--ovr 12, 37
--path *input-path* 18, 20
--print 63
--prn 63
--prn_fnc_tbl 46
--pth *input-path* 18, 20
--quiet 64
--retain 22
--revision 39
--rtn 22
--script 42
--script-file 42
--spt 42
--string 64
--units 64
--variable *var* 22
--version 39
--vrs 39
--weight *weight* 72
--weight *wgt1*[,*wgt2*] 59
--wgt_var *weight* 72
--wgt_var *wgt1*[,*wgt2*] 59
--xcl 22
-a 62, 64
-A 12, 37
-b 62
-B 62
-c 23
-C 23
-D *debug-level* 15, 17
-d *dim*,[*min*],[*max*],*stride* 29
-d *dim*,[*min*][,[*max*]] 24, 25, 26, 28, 72
-f 46
-F 24
-h 37, 48
-H 63
-I 72

-l *output-path* 20
-m 63
-M 63
-m *mask_var* 72
-N 33, 74
-n *loop* 14, 18
-O 12, 37
-p *input-path* 18, 20
-q 64
-r 39
-R 22
-s 64
-u 64
-v *var* 22
-w *weight* 72
-w *wgt1*[,*wgt2*] 59
-x 22
-y *op_typ* 31, 52

.

'.rhosts' 20

/

/ 52
/ (division) 45
/*...*/ (comment) 43
// (comment) 43

;

; (end of statement) 43

?

? (question mark) 50

@

@ (attribute) 43

[

[] (array delimiters) 43

^

^ (exponentiation) 45

_

_FillValue attribute 71

\

\ (backslash) 50
\" (protected double quote) 50
\' (protected end quote) 50
\? (protected question mark) 50
\\ (ASCII \, backslash) 50
\\ (protected backslash) 50
\a (ASCII BEL, bell) 50
\b (ASCII BS, backspace) 50
\f (ASCII FF, formfeed) 50
\n (ASCII LF, linefeed) 50
\n (linefeed) 64
\r (ASCII CR, carriage return) 50
\t (ASCII HT, horizontal tab) 50
\t (horizontal tab) 64
\v (ASCII VT, vertical tab) 50

0

0 (NUL) 50

A

abs 45
absolute value 45
acos 45
acosh 45
add 52
add_offset 16
add_offset 44
adding data 52, 59
addition 45, 52, 59
alphabetization 62
alphabetize output 64
anomalies 53
ANSI 6
ANSI C 46
appending data 61
appending to files 12, 37
appending variables 12
arccosine function 45
arcsine function 45
arctangent function 45
arithmetic operators 30, 31, 72
arithmetic processor 42
ARM conventions 38
ARM conventions 68
array indexing 43
array storage 43
array syntax 43
arrival value 59
ASCII 50
asin 45
asinh 45
assignment statement 43
asynchronous file access 20
atan 45
atanh 45

attribute names . . . 48, 70
attribute syntax . . . 43
attribute, `units` . . . 26
attributes . . . 48
attributes, appending . . . 49
attributes, creating . . . 49
attributes, deleting . . . 49
attributes, editing . . . 49
attributes, global . . . 37, 39, 49, 61, 62, 70, 71
attributes, modifying . . . 49
attributes, overwriting . . . 49
average . . . 31
averaging data . . . 30, 56, 66, 72
`avg` . . . 31
`avgsqr` . . . 31

B

`base_time` . . . 38
Bash shell . . . 52
binary format . . . 62
binary operations . . . 52
Bourne Shell . . . 30, 55
broadcasting variables . . . 53, 60, 72
`BSD` . . . 17
buffering . . . 16
bugs, reporting . . . 8
`byte(x)` . . . 45

C

C . . . 37
C index convention . . . 24
C language . . . 43, 50, 64
C Shell . . . 30, 55
`c++` . . . 6
`C++` . . . 6
C_format . . . 16
C89 . . . 6
C99 . . . 6
`cc` . . . 6
`CC` . . . 6
CCM Processor . . . 18, 66, 68
ceil . . . 45
ceiling function . . . 45
cerf . . . 45
CF convention . . . 28
`char(x)` . . . 45
characters, special . . . 50
Climate and Forecast Metadata Convention . . . 28
climate model . . . 11, 13, 19, 58, 74, 75
climate modeling . . . 11
Comeau . . . 5
command line options . . . 17
comments . . . 43
`como` . . . 6
Compaq . . . 5
compatability . . . 5
complementary error function . . . 45
concatenation . . . 12, 58, 68
contributing . . . 77
contributors . . . 77
coordinate limits . . . 24
coordinate variable . . . 27, 72
coordinate variables . . . 71
core dump . . . 8
`core dump` . . . 15, 65
cos . . . 45
cosh . . . 45
cosine function . . . 45
covariance . . . 47
Cray . . . 5, 15
CSM conventions . . . 38, 54
`cxx` . . . 6
`Cygwin` . . . 7

D

data safety . . . 11, 70
data, missing . . . 30, 48
`date` . . . 38
`datesec` . . . 38
dbg_lvl . . . 15
debug-level . . . 15
debugging . . . 15
DEC . . . 5
degenerate dimensions . . . 53, 59, 66
derived fields . . . 42
Digital . . . 5
dimension limits . . . 24
dimension names . . . 70
Distributed Oceanographic Data System . . . 21
`divide` . . . 52
dividing data . . . 52
division . . . 45
documentation . . . 5
DODS . . . 21
`DODS_ROOT` . . . 21
double precision . . . 46
`double(x)` . . . 45
dynamic linking . . . 7

E

eddy covariance . . . 47
editing attributes . . . 48
`egrep` . . . 22
ensemble . . . 13, 56
ensemble average . . . 56
ensemble concatenation . . . 58
erf . . . 45
error function . . . 45
error tolerance . . . 11
execution time . . . 7, 12, 16, 31, 70
exp . . . 45
exponentiation . . . 45

exponentiation function . . . 45
extended regular expressions . . . 22

F

features, requesting . . . 8
file deletion . . . 22
file removal . . . 22
file retention . . . 22
files, multiple . . . 19
files, numerous input . . . 14
flags . . . 47
`float` . . . 46
`float(x)` . . . 45
floor . . . 45
floor function . . . 45
force append . . . 37
force overwrite . . . 37
foreword . . . 1
fortran . . . 66, 68
Fortran index convention . . . 24
FORTRAN_format . . . 16
`ftp` . . . 7, 20

G

gamma . . . 45
gamma function . . . 45
Gaussian weights . . . 74
`gcc` . . . 6
GCM . . . 11
`getopt` . . . 17
'`getopt.h`' . . . 17
`getopt_long` . . . 17
global attributes . . . 37, 39, 49, 61, 62, 70, 71
globbing . . . 18, 23, 53, 66, 68
GNU . . . 17
GNU . . . 22
`gnu-win32` . . . 7
'`GNUmakefile`' . . . 7
God's units, i.e., MKS . . . 28
`gw` . . . 38, 74

H

HDF . . . 8
help . . . 8
Hierarchical Data Format . . . 8
`history` attribute . . . 37, 39, 48, 64
HP . . . 5
HTML . . . 5
HTTP protocol . . . 21
hybrid coordinate system . . . 42
hyperbolic arccosine function . . . 45
hyperbolic arcsine function . . . 45
hyperbolic arctangent function . . . 45
hyperbolic cosine function . . . 45
hyperbolic sine function . . . 45
hyperbolic tangent . . . 45
hyperslab . . . 24, 66, 68, 72

I

IBM . . . 5
`icc` . . . 6
IDL . . . 11
`ilimit` . . . 15
including files . . . 43
index conventions . . . 24
inexact conversion . . . 46
Info . . . 5
input-path . . . 18, 20
installation . . . 5
`int(x)` . . . 45
Intel . . . 5
interpolation . . . 59
introduction . . . 5
ISO . . . 6

K

kitchen sink . . . 61

L

large files . . . 15
`LD_LIBRARY_PATH` . . . 7
left hand casting . . . 42
lexer . . . 42
LHS . . . 42
`libnco` . . . 6
libraries . . . 7
Linux . . . 46
log . . . 45
log10 . . . 45
logarithm, base 10 . . . 45
logarithm, natural . . . 45
`long double` . . . 46
longitude . . . 28

M

Macintosh . . . 5
'`Makefile`' . . . 6, 7, 8, 21
masked average . . . 47, 72
masking condition . . . 73
Mass Store System . . . 20
mathematical functions . . . 45
`max` . . . 31
maximum . . . 31
mean . . . 31
memory requirements . . . 15, 22
merging files . . . 12, 61
metadata . . . 63
metadata, global . . . 63
Microsoft . . . 5, 6

min . . . 31
minimum . . . 31
missing values . . . 30, 32, 48
missing_value attribute . . . 30, 48, 71
MKS units . . . 27
modulus . . . 45
monotonic coordinates . . . 16
msrcp . . . 20
msread . . . 20
MSS . . . 20
multi-file operators . . . 19
multiplication . . . 45, 52
multiply . . . 52
multiplying data . . . 52, 59
multislab . . . 25

N

naked characters . . . 52
NC_BYTE . . . 25
NC_CHAR . . . 25
NC_DOUBLE . . . 46
ncap . . . 42
NCAR . . . 11
NCAR CSM conventions . . . 38, 54
NCAR MSS . . . 20
ncatted . . . 31, 37
ncatted . . . 48
ncbo . . . 31
ncbo . . . 52
ncdump . . . 63
ncea . . . 14, 31
ncea . . . 56
ncecat . . . 13
ncecat . . . 47, 58
ncextr . . . 61
ncflint . . . 14, 31
ncflint . . . 59
ncks . . . 61
NCL . . . 11
NCO availability . . . 5
NCO homepage . . . 5
NCO User's Guide . . . 5
ncra . . . 14, 31
ncra . . . 66
ncrcat . . . 13
ncrcat . . . 68
ncrename . . . 70
ncwa . . . 14, 31
ncwa . . . 47, 72
nearbyint . . . 45
nearest integer function (exact) . . . 45
nearest integer function (inexact) . . . 45
NEC . . . 5
nesting . . . 43
netCDF . . . 5
netCDF 2.x . . . 7
netCDF 3.x . . . 7
NETCDF2_ONLY . . . 8
NINTAP . . . 18, 66, 68
NO_NETCDF_2 . . . 8
normalization . . . 74
nrnet . . . 20
NUL . . . 50
NUL-termination . . . 50
null operation . . . 60
numerator . . . 33, 74

O

on-line documentation . . . 5
operation types . . . 31
operator speed . . . 7, 12, 16, 31, 70
operators . . . 3
ORO . . . 38, 75
OS . . . 5
output-path . . . 20
overwriting files . . . 12, 37

P

pack(x) . . . 44
packing . . . 44
parser . . . 42
pasting variables . . . 12
pattern matching . . . 22
performance . . . 7, 12, 16, 31, 70
Perl . . . 11, 50
philosophy . . . 11
portability . . . 5
POSIX . . . 17
POSIX . . . 22
precision . . . 46
preprocessor tokens . . . 7
printf() . . . 50, 64
printing files contents . . . 61
printing variables . . . 61
Processor . . . 66, 68
Processor, CCM . . . 18
promotion . . . 45

Q

quadruple precision . . . 46
quiet . . . 64
quotes . . . 23, 53

R

rank 53, 72
`rcp` 7, 20
RCS 39
record average 66
record concatenation 68
`regex` 22
regular expressions 18, 22
remote files 7, 20
renaming attributes 70
renaming dimensions 70
renaming variables 70
reporting bugs 8
rint 45
`rms` 31
`rmssdn` 31
root-mean-square 31
round 45
rounding functions 45
running average 66

S

safeguards 11, 70
`scale_factor` 44
scale_format 16
`scp` 7, 20
script file 42
semi-colon 43
server 15
SGI 5
shell 23, 28, 53
`short(x)` 45
signedness 16
sin 45
sine function 45
single precision 46
sinh 45
sort alphabetically 62, 64
source code 5
special characters 50
speed 7, 12, 15, 16, 31, 70
`sqravg` 31
`sqrt` 31
sqrt 45
square root function 45
SSH 7
standard deviation 31
statement 43
static linking 7
stride 27, 29, 63, 66, 68
strings 50
stub 21
`subtract` 52
subtracting data 52
subtraction 45, 52
summary 3
Sun 5
swap space 15
switches 17
synchronous file access 20
syntax 43

T

tan 45
tanh 45
temporary output files 11, 70
TeXinfo 5
`time` 27, 38
`time_offset` 38
timestamp 37
total 31
trunc 45
truncation function 45
`ttl` 31
type conversion 35, 45

U

UDUnits 26, 38
UNICOS 15
Unidata 26
`units` 51, 60
`units` attribute 26, 27
UNIX 5, 7
UNIX 17
UNIX 18
unpacking 44
URL 20
`USE_FORTRAN_ARITHMETIC` 6, 7
User's Guide 5

V

variable names 70
variance 31
version 39

W

weighted average 72
whitespace 27
wildcards 18, 22
`WIN32` 7
Windows 5, 6
wrapped coordinates 25, 28, 65
wrapped filenames 19
WWW documentation 5

X

`xlc` 6
`xlC` 6

Y

Yorick 11, 16

Table of Contents

Foreword **1**

Summary **3**

1 Introduction **5**
- 1.1 Availability 5
- 1.2 Operating systems compatible with NCO 5
  - 1.2.1 Compiling NCO for Microsoft Windows OS 6
- 1.3 Libraries 7
- 1.4 netCDF 2.x vs.3.x 7
- 1.5 Help and Bug reports 8

2 Operator Strategies **11**
- 2.1 NCO operator philosophy 11
- 2.2 Climate model paradigm 11
- 2.3 Temporary output files 11
- 2.4 Appending variables to a file 12
- 2.5 Addition Subtraction Division Multiplication and Interpolation 12
- 2.6 Averagers vs. Concatenators 13
  - 2.6.1 Concatenators `ncrcat` and `ncecat` 13
  - 2.6.2 Averagers `ncea`, `ncra`, and `ncwa` 14
  - 2.6.3 Interpolator `ncflint` 14
- 2.7 Working with large numbers of input files 14
- 2.8 Working with large files 15
- 2.9 Approximate NCO memory requirements 15
- 2.10 Performance limitations of the operators 16

3 Features common to most operators **17**
- 3.1 Command line options 17
- 3.2 Specifying input files 18
- 3.3 Accessing files stored remotely 20
  - 3.3.1 DODS 21
- 3.4 Retention of remotely retrieved files 22
- 3.5 Including/Excluding specific variables 22
- 3.6 Including/Excluding coordinate variables 23
- 3.7 C & Fortran index conventions 23
- 3.8 Hyperslabs 24
- 3.9 Multislabs 25
- 3.10 UDUnits Support 26
- 3.11 Wrapped coordinates 28
- 3.12 Stride 29

3.13 Missing values 30
3.14 Operation Types 31
3.15 Type conversion 35
3.16 Suppressing interactive prompts 37
3.17 History attribute 37
3.18 NCAR CSM Conventions 37
3.19 ARM Conventions 38
3.20 Operator version 39

4 Reference manual for all operators 41

4.1 `ncap` netCDF Arithmetic Processor 42
4.1.1 Left hand casting 42
4.1.2 Syntax of `ncap` statements 43
4.1.3 Intrinsic functions 43
Packing and Unpacking Functions 44
Type Conversion Functions 45
4.1.4 Intrinsic mathematical functions 45
4.2 `ncatted` netCDF Attribute Editor 48
4.3 `ncbo` netCDF Binary Operator 52
4.4 `ncea` netCDF Ensemble Averager 56
4.5 `ncecat` netCDF Ensemble Concatenator 58
4.6 `ncflint` netCDF File Interpolator 59
4.7 `ncks` netCDF Kitchen Sink 61
Options specific to `ncks` 62
4.8 `ncra` netCDF Record Averager 66
4.9 `ncrcat` netCDF Record Concatenator 68
4.10 `ncrename` netCDF Renamer 70
4.11 `ncwa` netCDF Weighted Averager 72
4.11.1 Masking condition 73
4.11.2 Normalization 73

5 Contributing 77

General Index 79