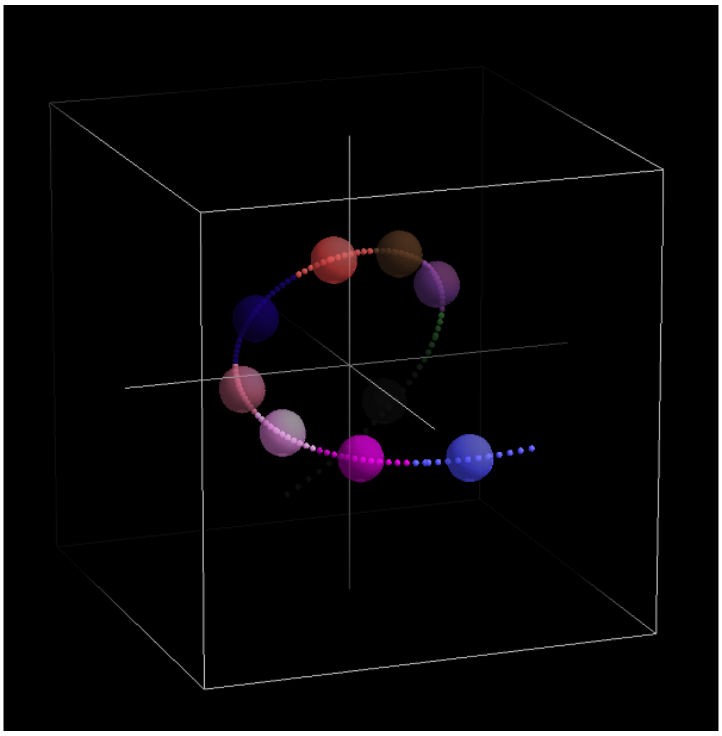
cost733class-1.2
User guide

Andreas Philipp[1], Christoph Beck[1], Pere Esteban[5,6], Frank Kreienkamp[2], Thomas Krennert[9], Kai Lochbihler[1], Spyros P. Lykoudis[3], Krystyna Pianko-Kluczynska[8], Piia Post[7], Domingo Rasilla Alvarez[10], Arne Spekat[2], and Florian Streicher[1]

[1] *University of Augsburg, Germany*
[2] *Climate and Environment Consulting Potsdam GmbH, Germany*
[3] *Institute of Environmental Research and Sustainable Development, National Observatory of Athens, Greece*
[5] *Group of Climatology, University of Barcelona, Spain*
[6] *Institut d'Estudis Andorrans (CENMA/IEA), Principality of Andorra*
[7] *University of Tartu, Estonia*
[8] *Institute of Meteorology and Water Management, Warsaw, Poland*
[9] *Zentralanstalt fuer Meteorologie und Geodynamik, Vienna, Austria*
[10] *University of Cantabria, Santander, Spain*

02/10/2014



Contents

1 Introduction

`cost733class` is a FORTRAN software package focused on creating and evaluating weather and circulation type classifications utilizing various different methods. The name refers to COST Action 733 which has been an initiative started in the year 2005 within the ESSEM (Earth System Science and Environmental Management) domain of the COST (European Cooperation in Science and Technology) framework. The topic of COST 733 is "Harmonisation and Applications of Weather Type Classifications for European regions". `cost733class` is released under GNU General Public License v3 (GPL) and freely available.

2 Installation

The software is developed on Unix/Linux operating systems, however it may possible to compile and run it on other operating systems too, although some features may not work. At the time of writing this affects the NetCDF and grib data format and the OpenGL visualization, which are not necessary to run the program.

2.1 Getting the source code

When you read this, you might have downloaded the software package and unpacked it already. However here is the way how to get and compile the software:

To download direct your web browser to:

```
http://cost733class.geo.uni-augsburg.de/cost733class-1.2
```

Here you will find instructions how to download the source code by SVN or how to get a tar package.

If you have the tar-package unpack the bzip2 compressed tar-file e.g. by:

```
tar xvfj cost733class-1.2_RC_revision??.tar.bz2
```

where `??` stands for the svn version number.

Change into the new directory:

```
cd cost733class-1.2
```

2.2 Using configure and make

The next step is to compile the source code to generate the executable called `cost733class`. For this you need to have a C and a FORTRAN90 compiler installed on your system. The compiler reads the source code from the `src` directory and generates the executable `src/cost733class` in the same directory which can be started on the command line of a terminal. In order to prepare the compilation process for automatic execution, so called `Makefiles` are generated by a script called `configure`. The `configure` script checks whether everything (tools, libraries, compilers) which is needed for compilation is installed on your system. If `configure` stops with an error, you have to install the package it is claiming about (see troubleshooting) and rerun `configure` until it is happy and creates the `Makefiles`. The package include some example shell scripts containing the command to configure and make the software in one step:

- `compile_gnu_debug.sh`: this script uses the GNU Linux compilers gcc, gfortran and c++ to compile the software with NetCDF support but without GRIB and OpenGL support. Debugging information will be given in case of software errors.
- `compile_gnu_debug_opengl.sh`: this script additionally includes OpenGL support for visualization.
- `compile_gnu_debug_grib.sh`: this script additionally includes grib support.
- `compile_gnu_debug_omp.sh`:this script additionally includes compiler options to run parts of the code in parallel.
- `compile_intel_omp.sh`: this script uses the intel compiler suite.

In order to execute these scripts you type e.g.:

```
./compile_gnu_debug_opengl.sh
```

or

```
sh compile_gnu_debug_opengl.sh
```

These scripts can be easily copied and modified to save compiler options which are often needed. However it is also possible to run the two commands `configure` and `make` manually one after the other, as described in the following.

2.2.1 configure

The configure script tries to guess which compilers should be used, however it is advisable to specify the compilers by setting the `FC=` and `CC=` flags. E.g. if you want to use the GNU compilers `gfortran` and `gcc` say:

```
./configure FC=gfortran CC=gcc
```

or if the intel compilers should be used:

```
./configure FC=ifort CC=icc
```

Note that the FORTRAN and C compilers should be able to work together, i.e. the binaries must be compatible. This is not always the case e.g. when mixing other and GNU compilers (depending on versions).

Also you can use some special compiler options, e.g. for running parts of the classifications in parallel:

```
./configure FC=ifort CC=icc FCFLAGS="-parallel -openmp"
```

In the same manner options for the C-compiler can be set by the `CCFLAGS="..."` option.

By default `cost733class` compiles with netCDF, GRIB and OpenGL support. Further options for the configure script control and disable special features of the package:

```
./configure --disable-grib
```

switches off the compilation of the GRIB_API, thus grib data can't be read by the software.

```
./configure --disable-opengl
```

switches off the compilation of routines which visualize some of the classification processes like SOM, SAN or CKM. This feature is working only for unix systems with opengl-development packages (gl-dev glut-dev x11-dev) installed.

2.2.2 make

The compiling process itself is done by the tool called `make`:

```
make
```

If everthing went well, you will find the executable `cost733class` in the src directory. Check it by running the command:

```
src/cost733class
```

If you have administrator privileges on your system you can install the executables into `/usr/local/bin`. Note that the NetCDF tools coming along with cost733class are also installed there.

```
sudo su -
make install
```

Alternatively you can copy only the executable to any `bin` directory to have it in your command search path:

```
sudo copy src/cost733class /usr/local/bin
```

That's it, you can now start to run classifications, which of course needs some input data describing the atmospheric state of each time step (i.e. object) you want to classify. You can now try with the next chapter (Quick start) or look into the Data input chapter.

2.3 Manual compilation as an alternative

Within the package directory there is a batch file called `compile_mingw.bat` for Windows. This script file contains the direct compile command for the `gfortran` and `g95` compiler without any dependencies to any libraries. It can be used if no NetCDF input is needed or if there are troubles compiling the NetCDF package.

2.4 Troubleshooting

2.4.1 For configure

- If `configure` claims about something missing or a wrong version number, you have to install or update the concerning software packages. For Debian and Ubuntu systems that's rather easy. First you have to find out the official name of the package which contains the files `configure` was claiming about. Here you can use the tool `apt-file` which must be installed first, updated and run:

  ```
  sudo apt-get install apt-file
  apt-file update
  apt-file search <filename>
  ```

  It will then print the name of the package in the first column and you can use it to install the missing package:

  ```
  sudo apt-get install <package>
  ```

2.4.2 For make

- If the following error message appears:

  ```
  Catastrophic error: could not set locale "" to allow processing of
  multibyte characters
  ```

  setting the environment variable `LANG` to `C`

  ```
  export LANG=C
  ```

  in the shell you use for compilation will fix it.

2.4.3 For runtime problems

- Some methods need a lot of RAM and maybe more than is available. If this is the case you can try to enlarge stack size each time you run the software by

  ```
  ulimit -s unlimited
  ```

- If the following error or a similar error occurs:

  ```
  *** glibc detected *** double free or corruption (!prev): 0x08196948 ***
  ```

  this is probably due to differing compiler/library versions. You can try to get rid of it by:

  ```
  export MALLOC_CHECK_=0
  ```

- If compiled with grib support and running `cost733class` results in the following error:

  ```
  GRIB_API ERROR    :   Unable to find boot.def
  grib_context.c at line 156: assertion failure Assert(0)Aborted
  ```

  one has to define the path to the so called grib definitions. In an Unix environment something like

  ```
  export GRIB_DEFINITION_PATH="/PATH/cost733class-1.2/grib_api-1.9.18/definitions"
  ```

  should do it. Use absolute paths!

- Depending on the processor type and data size method SAN and SOM may run for several hours up to days. This is no bug!

3 Getting started

3.1 Principal usage

`cost733class` has been written to simplify and unify the generation of classification catalogs. It's functionality is controlled by command line arguments, i.e. options are given as key words after the command which is entered into a terminal or console providing a command prompt. The command line interface (CLI) makes the software suitable for shell scripts and can easily run on compute servers or clusters in the background. Recently a graphical user interface (GUI) has been added by using the OpenGL menu functions which is accessible by a right click into the OpenGL window, however it is not very intuitive yet. Therefore the documentation will concentrate on the CLI interface.

If you type `src/cost733class` just after the compilation process you will see the output of the `-help` function. In order to run a classification, you have to provide command line arguments for the program, i.e. you have to type expressions behind the `cost733class` command which are separated by blanks (`" "`). The program will scan the command line and recognize all arguments beginning with a `"-"`, several of which have to be followed by another expression. All expressions have to be separated by one or more blanks, which is the reason that in between any expression (e.g. a file name) no blank is allowed. Also all expressions have to be written in lower/upper case letters if said so in the `-help` output. Command lines can be rather simple, however some methods and the data configurations can be complex, thus that the command line gets longer and longer the more you want to fine tune your classification. In this case, and especially if just variations of a classifications should be run one after another, it can be useful to use shell scripts to run the software which will be explained below.

In order to understand which options play a role for which part of the classification it is helpful to distinguish between options relevant for the data input and preprocessing and options relevant for the routines itself. In order to better understand how the software works and how it can be used, the workflow is briefly described in the following:

In the first step the command line arguments are analyzed (subroutine arguments()). For each option default values are set within the program, however they are changed if respective options are provided by the user. The next step is to evaluate the data which should be read from a file, especially the size of the data set has to be determined here in order to reserve (allocate) computer memory for it. Then the data are read into a preliminary array (RAWDAT) and eventually are preprocessed (selected, filtered, etc.) before they are put into the final, synchronized, DAT array which is used by the methods. In some cases (depending on the classification/evaluation operation) existing catalogs have to read. This is done in the next step by the subroutine classinput(). Afterwards the subroutine for the desired operation is called. If this subroutine finished with a new catalog, it is written to a classification catalogue file (*.cla) and the program ends. Other routines e.g. for comparison or evaluation of classification catalogs might just produce screen output.

3.2 Quick start

As an requirement to write complex commands it is necessary to understand the basic synopsis of `cost733class`. There are several different use cases in which `cost733class` can be used:

1. Creation of classifications of any numerical data following an entity-attribute-value model (main purpose of `cost733class`). For more information about the data model used by `cost733class` see section 4.
2. Evaluation of such classifications
3. Comparison of such classifications
4. Assignment of existing classifications to new data
5. Rather simple data (pre)processing

All of these use cases require different commands following different structures. Therefor the next sections give brief instructions on how to use `cost733class` for each of these five cases.

3.2.1 Creating classifications

For the creation of classifications a basic command follows the scheme:

```
cost733class -dat <specification> [-dat <specification >] -met <method> [-ncl
    <integer >] [-cnt <filename >] [-cla <file >] [more method specific options]
```

Essential for a successful completion of each classification run is the `-dat <specification>` option which provides necessary information about the input data. For some methods it is possible to give more than one data specification option, for others it is prerequisite. Furthermore the classification method must be named by the `-met <method>` option. The number of classes can be specified with `-ncl <integer>`. The filenames for the output of the classification catalog and the corresponding class centroids can be defined by `-cla <filename>` and `-cnt <filename>`. Depending on the used classification method several other options can be provided by special options. For further documentation of these consult the relative section.
Based on this the command for a classification with the method KMN may look like:

```
cost733class -dat pth:slp.dat -ncl 9 -met KMN -cla KMN_ncl9.cla
```

3.2.2 Evaluating classifications

The basic scheme of an evaluation command is:

```
cost733class -dat <specification> [-dat <specification>] -clain <specification> -met
    <method> [-idx <filename>] [more method specific options]
```

The most important difference to a classification run is the `-clain <specification>` option. It defines the path to an existing classification catalog file and is mandatory for all evaluation methods. The desired evaluation method must be chosen by the `-met <method>` option. The results are written to one or more files for which the base name can be given by the `-idx <filename>` option. Analogous to the previous command every method has its additional options which are explained in the corresponding sections below. At least one `-dat <specification>` option describes the data one wants to evaluate with.
A `cost733class` run which evaluates the classification generated by the previous command using the Brier Score would be:

```
cost733class -dat pth:slp.dat -clain pth:KMN_ncl9.cla -met BRIER -idx
    brier_KMN_ncl9.txt
```

3.2.3 Comparing classifications

Comparing two or more classifications is rather easy:

```
cost733class -clain <specification> [-clain <specification>] -met <method> [-idx
    <filename>] [more method specific options]
```

The only difference to an evaluation and classification run is the absence of the `-dat <specification>` option. Instead at least two classification catalogs (more than one per file are possible) must be provided.
A comparison of two partitions based on different methods could be done with:

```
cost733class -clain pth:DKM_ncl9.cla -clain pth:KMN_ncl9.cla -met CPART -idx
    KMN_DKM_ncl9_compared.txt
```

3.2.4 Assignment to existing classifications

Assignments to existing classifications can be done on two different bases:

1. for a given classification catalog

2. for predefined class centroids

In both cases one has to provide and describe data which is then assigned to an existing classification. Other required `cost733class` options differ in both cases. Please refer to the corresponding sections below.

3.2.5 Rather simple (pre)processing

Beyond the use cases mentioned above there are a few methods in `cost733class` which process input data in a different way. They are grouped together under the term miscellaneous functions.
Furthermore it is possible to just preprocess input data without calling a distinctive method.

```
cost733class -dat <specification> [options for selecting dates] -writedat <filename>
```

In this case data output can be accomplished by the `-witedat <filename>` option. There are several flags for preprocessing and spatial selection which are listed and described in the relative section. An example might be:

```
cost733class -dat pth:slp.dat lon:-10:30:2.5  lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d ano:3 fil:30 -per 2000:1:1:12,2005:12:31:12,1d -writedat
    output.bin
```

Note that here the output data are unformatted binary files (extension: bin). These files can be read by `cost733class` in a subsequent run. This proceeding can be very helpful if many runs of `cost733class` are planned.

3.3 Help listing

If you run the `cost733class` command without any command line argument (or with the `-help` option) you should see the following help text, giving a brief description of the various options:

```

USAGE: ../src/cost733class -dat <specification> [-dat <specification>] [options]

OPTIONS:


 ______________________________________________________________________
 INPUT DATA:

 -dat <char>      : specification of input data. More than one '-dat' arguments
                  : are allowed to combine data of same sample size but from
                  : different files in different formats.
                  : <char> consists of various specifications separated by the
                  : character '@' or one or more blanks ' ':
                  :   @var:<name of variable> for fmt:netcdf this must be the
                  :        variable name in the netcdf file
                  :   @pth:<path for input data file>
                  :        in case of netcdf it may include "?" symbols to be
                  :        replaced by numbers given by the fdt: and ldt: flags.
                  :   @fmt:<"ascii" or "netcdf" or "grib"> default ascii
                  :         If file name ends with ".nc" netcdf is assumed;
                  :         for ".grib/.grb/.gribX/.grbX" (X=[1,2]) grib is assumed.
                  :      ascii: (default) ascii file with one line per day (object
                  :             to classify) and one column per variable (parameter
                  :             defining the objects) the columns have to be delimited
                  :             by one or more blanks. The number of objects and
                  :             variables is scanned by this program on its own.
                  :      netcdf: data in self describing netcdf format.
                  :             time and grid is scanned automatically.
                  :      grib: data in self describing format.
                  :             time and grid is scanned automatically.
                  :    @dtc:<1-4> (number of leading date columns: year, month,
                  :         day, hour in ascii file)
                  :    @fdt:<YYYY:MM:DD:HH> first date in dataset (description)
                  :    @ldt:<YYYY:MM:DD:HH> last date in dataset (description)
                  :    @ddt:<int><y|m|d|h> time step in data set in years, months,
                  :         days or hours
                  :    @mdt:<list of months> months in data set, e.g. @mdt:1:2:12
                  :    @lon:<minlon>:<maxlon>:<diflon> longitude description
                  :    @lat:<minlat>:<maxlat>:<diflat> latitude description
                  :    @slo:<minlon>:<maxlon>:<diflon> longitude selection
                  :    @sla:<minlat>:<maxlat>:<diflat> latitude selection
                  :    @sle:<level between 1000 and 10> selection
                  :    @arw:<integer> area weighting 1=cos(latitude),
                  :         2=sqrt(cos(latitude)),
                  :         3=calculated weights by area of grid box which is the same
                  :         as cos(latitude) of option 1
                  :    @scl:<float> scaling of data values
                  :    @off:<float> offset value to add after scaling
                  :    @nrm:<int> object (row-wise) normalisation:
                  :         1=centering,2=std(sample),3=std(population)
                  :    @ano:<int> variable (column-wise) normalisation:
                  :         done after selection of time steps:
                  :           -1=centering,-2=std(sample),-3=std(population)
                  :         done before selection of time steps:
                  :           1=centering,2=std(sample),3=std(population)
                  :           11=centering for days of year,
                  :           12=std for days (sample),13=std(population)
                  :           21=centering for months,
                  :           22=std for months (sample),
                  :           23=std(population)
                  :           31=centering on monthly mean (running 31day window),
                  :           32=std for months (sample) (running 31day window),
                  :           33=std (population) (running 31day window)
                  :    @fil:<integer> gaussian time filter int>0 -> low-pass
                  :         int<0 -> high-pass
                  :    @pca:<float|integer> PCA of parameter data set:
                  :         if <float>: for retaining explained variance
                  :         fraction, if <int>: number of PCs
                  :    @pcw:<float|integer> as @pca but with weighting by
                  :         explained variance
                  :    @seq:<sequence length for extension>
                  :    @wgt:<weighting factor>

                  :    @cnt:<file name> file name to write centroid/composite
                  :         values to if extension = "*.nc" it is netcdf format,
                  :         ascii otherwise (with coordinates for *.txt, without
                  :         for *.dat).

-readncdate       : read the date of observations from netcdf time variable rather
                  :         than calculating it from "actual range" attribute of the
                  :         time variable (slows down the data input but can override
                  :         buggy attribute entries as e.g in slp.2011.nc).
```

```
-per <char>     : period selection, <char> is e.g. 2000:1:1:12,2008:12:31:12,1d

-mon <char>     : list of months to classify: MM,MM,MM,...
                :    e.g.: -mon 12,01,02 classifies winter data (default is all
                :     months), only applies if -per is defined!

-mod            : LIT: set all months to be 30 days long (relevant for -met lit)

-dlist <char>   : list file name for selecting a subset of dates within the
                : given period for classification. Each line has to hold one
                : date in form: "YYYY MM DD HH" for year, month, day, hour.
                : If hour, day or month is irrelevant please provide the constant
                : dummy number 00.

-cat <spec>     : classification catalog input, where <spec> consists of the following
                  flags:
                   @pth:<path for file>
                   @fdt:<first date>
                   @ldt:<last date>
                   @ddt:<time step, e.g. "@ddt:1d" for one day>
                   @dtc:<number of date columns>
                   @mdt:<list of months> e.g. "@mdt:01,02,12"
-catname <char> : file with as many lines as catalogs read in by "-cat",
                  each line contains the name of the catalog of the corresonding
                  column in the catalog file.

-cntin <char>   : centroid input file for -met ASS and ASC

-pca <float|integer>: PCA of input data all together:
                : if <float>: for retaining explained variance fraction
                : if <int>: number of PCs

-pcw <float|integer>: PCA of input data, PCs weighted by explained variance:
                : if <float>: for retaining explained variance fraction
                : if <int>: number of PCs
________________________________________________________________________________
OUTPUT:

-cla <clafile> : name of classification output file (contains number of class
               : for each object in one line).
               : default = ./<basename(datfile)>_<method>_ncl<int>.cla

-mcla <clafile>: multiple classification output file. Some methods generate
               : more than one (-nrun) classification and select the best.
               : option -mcla <file> makes them writing out all.

-skipempty <F|T> : skip empty classes in the numbering scheme? T=yes, F=no
                : Default is "T"

-writedat <char>: write preprocessed input data into file name <char>

-dcol <int>     : write datum columns to the cla outputfile:
                : <int>=1: year, <int>=2: year,month, <int>=3: year,month,day,
                : <int>=4: year,month,day,hour

-cnt <char>     : write centroid data to file named <char> (contains composites
                : of the input data for each type in a column).
-sub <char>     : method SUB: write substitute data to file named <char>
-agg <char>     : method AGG: write aggregated data to file named <char>
-idx <char>     : write index data used for classification to file named
                : <char>.<ext>
```

```
                 : the type of the indices dependes on the method (e.g. scores
                 :  and loading for PCT)

-opengl          : this switch activates the 3D-visualisation output calls for
                 :  the following methods:
                 : SOM -crit 2, SAN, CKM.
                 : This is only working without parallelization and probably on
                 :  unix/linux systems
                 : The software compilation has to be configured by
                 : "./configure --enable-opengl".

-gljpeg          : in conjunction with -opengl this switch produces single
                 : jpg-images which can be used to create animations.

-glwidth         : width of opengl graphics window (default=800)

-glheight        : height of opengl graphics window (default=800)

-glpsize         : size of data points (default=0.004D0)

-glcsize         : size of centroid points (default=0.03D0)

-glxangle <real> : angle to tilt view on data cube (default = -60.D0)

-glyangle <real> : angle to tilt view on data cube (default = 0.D0)

-glzangle <real> : angle to tilt view on data cube (default = 35.D0)

-glstep          : time stepping (default=10)

-glpause         : pause length (default=1)

-glbackground <int> : background color: 0=black (default), 1=white

-glrotangle <angle> : rotation angle step for spinning cube

__________________________________________________________________________
METHODS:

-met <method>    : method :

                 : NON|none       : just read (and write) data and exit

                 : INT|interval|BIN : classify into intervals of variable -svar

                 : GWT|prototype  : prototype 'grosswetterlagen', correlation based,
                 :                  resulting in 26 types
                 : GWTWS|gwtws    : based on GWT (above) using 8 types,
                 :                  resulting in 11 types
                 : LIT|lit        : litynski thresholds, one circulation field,
                 :                  dates, ncl=9|18|27
                 : JCT|jenkcoll   : Jenkinson Collison scheme
                 : WLK|wlk        : threshold based using pressure, wind,
                 :                  temperature and humidity

                 : PCT|TPC|tmodpca: t-mode principal component analysis of 10 data
                 :                  subsets, oblique rotation
                 : PTT|TPT|tmodpcat: t-mode principal component analysis,
                 :                  varimax rotation
                 : PXE|pcaxtr     : s-mode PCA using high positive and negative
                 :                  scores to classify objects
                 : KRZ|kruiz      : Kruizinga PCA scheme
```

```
                  : LND|lund       : count most frequent similar patterns (-thres)
                  : KIR|kirchhofer: count most frequent similar patterns (-thres)
                  : ERP|erpicum    : count most frequent similar patterns, angle
                  :                  distance, adjusting thresholds

                  : HCL|hclust     : hierarchical cluster analysis (Murtagh 1986),
                  :                  see parameter -crit !

                  : KMN|kmeans     : k-means cluster analyis (Hartigan/Wong 1979
                  :                  algorithm )
                  : CKM|ckmeans    : like dkmeans but eventually skips small clusters
                  :                 <5% population
                  : DKM|dkmeans    : k-means (simple algorithm) with most different
                  :                  start patterns
                  : SAN|sandra     : simulated annealing and diversified randomisation
                  :                  clustering
                  : SOM|som        : self organising maps (Kohonen neural network)
                  : KMD|kmedoids   : Partitioning Around Medoids

                  : RAN|random     : just produce random classification catalogues
                  : RAC|randomcent: determine centroids by random and assign objects
                  :                  to it.

                  : ASC|assign     : no real method: just assign data to given
                  :                  centroids provided by -cntin
                  : SUB|substitute: substitute catalog numbers by values given in a
                    -cntin file
                  :
                  : AGG|aggregate : build seasonal values out of daily or monthly values.
                  :
                  : COR|correlate : calculate correlation metrics comparing the input
                    data variables.
                  :
                  : CNT|centroid   : calculate centroids of given catalog (-clain)
                  :                  and data (see -dat), see also -cnt

                  : ECV|exvar      : evaluation of classifications by Explained
                  :                  Cluster Variance (see -clain -crit)
                  : EVPF|evpf     : evaluation in terms of explained variation
                  :                  and pseudo F value (-clain)
                  : WSDCIM|wsdcim: evaluation in terms of within-type SD and
                  :                 confidence interval (-clain)
                  : FSIL|fsil      : evaluation in terms of the Fast Silhouette
                  :                  Index (-clain)
                  : SIL|sil        : evaluation in terms of the Silhouette Index
                  :                  (-clain)
                  : DRAT|drat      : evaluation in terms of the distance ratio
                  :                  within and between classes (-clain)
                  : CPART|cpart    : calculate comparison indices for >= 2 given
                  :                  partitions (-clain)
                  : ARI|randindex : calculate only (adjusted) Rand indices for
                  :                  two or more partitions given by -clain

-crit <int>       : INT|interval:
                  :   1 = calculate thresholds as i'th percentile where i=cl*1/ncl
                  :   2 = bins centered around the mean value
                  :   3 = bin size is the data-range/ncl, the bins are not centered.
                  :   4 = 2 bins divided by -thres <real> for -svar <int>.
                  :   5 = as 4 but threshold is interpreted a percentile (0 to 100).
                  : HCL: (hierarchical clustering): number of criterion :
                  :   1 = Ward's minimum variance method
                  :   2 = single linkage
                  :   3 = complete linkage
```

```
                        :    4 = average linkage
                        :    5 = Mc Quitty's method
                        :    6 = median (Gower's) method
                        :    7 = centroid method
                        : GWT:
                        :    1 = raw coefficients for vorticity (default)
                        :    2 = normalized vorticity coefficients
                        : GWTWS:
                        :    1 = classification based on absolut values (default)
                        :    2 = classification based on percentiles
                        : ECV:
                        :    1 = monthly normalized data (default)
                        :    0 = raw data for calculating explained cluster variance.
                        : PCT: rotation criteria:
                        :    1 = direct oblimin, gamma=0 (default)
                        : WLK:
                        :    0 = use raw cyclonicity for deciding anticyclonic or cylonic
                             (default)
                        :    1 = use anomalies of cyclonicity
                        : JCT:
                        :    1 = centered classification grid with an extend of 30 W-E;20 N-S
                             (default)
                        :    2 = classification grid extended to data region
                        : SOM:
                        :    1 = 1-dimensional network topology
                        :    2 = 2-dimensional network topology
                        : KMD:
                        :    0 = use Chebychev distance d=max(|xa-xb|)
                        :    1 = use Manhattan distance d=sum(|xa-xb|)
                        :    2 = use Euclidean distance d=sqrt(sum((xa-xb)**2))
                        :    p = use Minkovski distance of order p: d=(sum(|xa-xb|**p))**(1/p)
                        : PXE/PXK:
                        :    0 = only normalize patterns for PCA (original)
                        :    1 = normalize patterns and normalize gridpoint values afterwards
                             (default)
                        :    2 = normalize patterns and center gridpoint values afterwards
                        : EVPF, WSDCIM, FSIL, SIL, DRAT:
                        :    0 = evaluate on the basis of the original data values
                        :    1 = evaluate on the basis of daily anomaly values
                        :    2 = evaluate on the basis of monthly anomlay values
                        : BRIER:
                        :    1 = quantile to absolut values (default)
                        :    2 = quantile to euclidean distances between patterns

-thres <real>           : KRC and LND: distance threshold to search for key patterns
                        :   default = 0.4 for kirchhofer and 0.7 for lund.
                        : INT: threshold between bins.
                        : WLK: fraction of gridpoints for decision on main wind sector
                             (default=0.6)
                        : PXE/PXK: threshold defining key group (default=2.0)
                        : BRIER: quantile [0,1] (default=0.9) to define extreme-events. An
                             event is
                        :   defined when the euclidean distance to the periods/seasonal/monthly
                             mean-pattern
                        :   is greater than the given quantile. If <thres> is signed negative
                             (e.g. -0.8),
                        :   than events are defined if smaller than the given quantile.

-shift                  : WLK: shift 90 degree wind sectors by 45 degree. Default is no shift.

-ncl <int>              : number of classes (must be between 2 and 256)

-nrun <int>             : number of runs (for SAN, SAT, SOM, KMN) for selection of best result.
```

```
                   :  Cluster analysis is by design an unstable method for complex
                       datasets.
                   :  The more repeated runs are used to select the best result the more
                       robust
                   :  is the result. SOM and SAN are designed to be much more robust than
                       KMN.
                   :  default = -nrun 1000 to produce reliable results!

-step   <int>      : SOM: number of epochs after which neighbourhood radius is to be
     reduced
                   :  For training the neurons, also neighboured neurons of the winner
                       neuron
                   :  in the network-map are affected and adapted to the training pattern
                       (to a
                   :  lower degree though). The neighbourhood radius covers all neurons
                       (classes
                   :  at the beginning and is reduced during the process until only the
                       winner neuron
                   :  is affected. This slow decrease helps to overcome local minima in
                       the optimisation
                   :  function.
                   :  default = -step 10 (meaning after 10 epochs neighbourhood radius is
                       reduced
                   :  by one).
                   : WLK: number of windsectors
                   : EVPF, WSDCIM, FSIL, SIL, DRAT, CPART: missing value indicator for
                       catalogue data

-niter <int>        : SOM, SAN, PXE, PXK: maximum number of epochs/iterations to run
                   :  defaults:
                   :  -niter 0 for pcaxtr means that only the first assignment to the
                       pc-centroids is done.
                   :  for PXK -niter is 9999999 (means 9999999 k-means iterations)
                   :  for SAN there is no default for -niter (infinity). Enable it if SAN
                       ends up in an endless loop.

-temp <real>       : simulated annealing start temperature (for CND)
                   :  default = 1

-cool <real>       : cooling rate (for som & sandra)
                   :  default = -cool 0.99D0 ; set to 0.999D0 or closer to 1.D0 to
                       enhance (and slow down).

-svar <int>        : tuning parameter
                   :  INT: number of variable/column to use for calculatin interval
                       thresholds.

-alpha <real>      : tuning parameter
                   :  WLK: central weight for weighting mask (default=3.D0)
                   :  EVPF, WSDCIM, FSIL, SIL, DRAT: scale factor for evaluation data
                   :  BRIER:
                   :     if < 0 => (default) use all values (-crit 1) or patterns (-crit 2)
                   :     if >=0 => a value or pattern is processed only if itself or
                       mean(pattern) > alpha.
                   :  GWTWS: value/percentile for low winds (main threshold for types
                       9,10,11)

-beta <real>       : tuning parameter
                   :  WLK: middle zone weight for weighting mask (default=2.D0)
                   :  EVPF, WSDCIM, FSIL, SIL, DRAT: offset value for evaluation data
                   :  GWTWS: value/percentile for flat winds (type 11)

-gamma <real>      : tuning parameter
```

```
                    :  WLK: margin zone weight for weighting mask (default=1.D0)
                    :  WSDCIM: confidence level for estimating the confidence interval of
                         the mean
                    :  GWTWS: value/percentile for low pressure (type 9)

-delta <real>       : tuning parameter
                    :  WLK: width factor for weigthing zones (nx*delta|ny*delta)
                         (default=0.2)
                    :  PXE/PXK: score limit for other PCs to define uniquely leading PC
                         (default 1.0)
                    :  GWTWS: value/percentile for high pressure (type 10)

-lambda <real>      : tuning parameter
                    :  SAT:  weighting factor for time constrained clustering
                         (default=1.D0)

-dist <int>         : distance metric (not for all methods yet):
                    :    if int > 0: Minkowski distance of order int (0=Chebychev),
                    :    if int =-1: 1-correlation coefficient.

-nx <int>           : KIR: number of longitudes, needed for row and column correlations
                    : DRAT: distance measure to use (1=euclidean distance, 2=pearson
                         correlation)

-ny <int>           : KIR: number of latitudes

-wgttyp euclid|normal : adjust weights for simulating Euclidean distances of original
                    :  data or not.

________________________________________________________________________________
OTHER:

-v <int>            : verbose: default = '-v 0', i.e. quiet (not working for all routines
     yet),
                    : 0 = show nothing, 1 = show essential information, 2 = information
                         about routines
                    : 3 = show detailed informations about routines work (slows down
                         computation significantly).

-help               : generate this output and exit

________________________________________________________________________________
```

4 Data input

The first step of a `cost733class` run is to read input data. Depending on the use case these data can have very different formats. Principally there are two kinds: Foreign data were not produced by `cost733class`, whereas self-generated indeed are.

4.1 Foreign formats

`cost733class` can read any data in order to classify it. It is originally made for weather and circulation type classification but most of the methods are able to classify anything, regardless of the meaning of the data (only in some cases information about dates and spatial coordinates are necessary). The concept of classification is to define *groups*, *classes* or *types* (all these terms are used equivalently) encompassing *objects*, *entities*, *cases* or *objects* (again these terms mean more or less the same) belonging together. The rule how the grouping should be established differs from method to method, however in most of the cases the similarity between objects is utilized, while the definition of similarity differs again. In order to distinguish between various objects and to describe similarity or dissimilarity, each object is defined by a set of *attributes* or *variables* (this is what an entity is made of). A useful model for the data representation is the concept of a rectangular matrix, where each row represents one object and each column represents one attribute of the objects. In case of ASCII-formatted input data this is exactly the way how the input data format is defined.

4.1.1 ASCII data file format

ASCII text files have to be formatted in such a way that they contain the values of one object (time slice) in one line and the values for one variable (commonly a grid point) in a column. For more than one variable (grid point) the following columns have to be separated by one or more blank letters (`" "`). Note that tabulator-characters (sometimes used by spread sheet programs as default) are not sufficient to separate columns! The file should contain a rectangular input matrix of numbers, i.e. constant number of columns at each row and a constant number of rows for each column. No missing values are allowed. The software evaluates an ASCII file on its own to find out the number of lines (days, objects, entities) and the number of variables (attributes, parameters, grid points). For this the numbers in one line have to be separated by one or more blanks (depending on the compiler commas or slashes may also be used as separators between different numbers, but the comma never as decimal marker!!! The decimal marker must always be a point!). Thus you don't have to tell how many lines and columns are in the file, but the number of columns has to be constant throughout the file (the number of blanks between two attributes may vary though). This format is fully compatible to the CSV (comma separated values) format which can be written e.g. by spread sheet calculation programs. Note that no empty line is allowed. This can lead to read errors especially if there is an empty line at the bottom of the file which is hard to see. If necessary, information about time and grid coordinates describing the data set additionally have to be provided within the data set specification at the command line. In this case the ordering of rows and columns in the file must fit the following scheme: The first column is for the southernmost latitude and the westernmost longitude, the second column for the second longitude from west on the southernmost latitude, etc. The last column is for the easternmost and northernmost grid point. The rows represent the time steps. All specifications have to fit to the number of rows and columns in the ASCII file.

4.1.2 COARDS NetCDF data format

Cost733class includes the NetCDF library and is able to read NetCDF files directly and use the information stored in this self describing data format. It has been developed using 6-hourly NCEP/NCAR reanalysis data (Kalnay et al., 1996), which are described and available at: `http://www.cdc.noaa.gov/data/gridded/data.ncep.reanalysis.html` but also other data sets following the COARDS conventions (`http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html`) should be readable if they use the time unit of `hours since 1-1-1 00:00:0.0` or any other year. In case of multi-file data sets the files have to be stored into one common directory and a running number within the file names has to be indicated by `?` symbols, which are replaced by a running number given by the `fdt:` and `ldt:` flag (see below). Information about time and grid coordinates of the data set are retrieved from the NetCDF data file automatically. In case of errors resulting from buggy attributes of the time axis in NetCDF files try `-readncdate`. Then the dates of observations are retrieved from the NetCDF time variable rather than calculating it from "actual range" attribute of the time variable.

4.1.3 GRIB data format

Cost733class includes the GRIB_API package and is able to read grib (version 1 & 2) files directly and use the information stored in this self describing data format. It has been developed using 6-hourly ECMWF data, which are described and available at: `http://data-portal.ecmwf.int/data/d/interim_daily/`

In case of multi-file data sets the file paths have to contain running numbers indicated by `?` symbols, or a combination of the following strings: `YYYY` / `YY`, `MM`, `DD` / `DDD` ; they are replaced by running numbers given by the `fdt:` and `ldt:` flags (see below).

Information about time and grid coordinates of the data set are retrieved from the grid data file automatically.

4.1.4 Other data formats

At the moment there is no direct support for other data formats. Also it may be necessary to convert NetCDF-files first if e.g. the time axis is not compatible. A good choice in many aspects might be the software package CDO (climate data operators): `https://code.zmaw.de/projects/cdo` which is released under GNU General Public License v2 (GPL). Attention has to be paid to the type of calendar used. It can be adjusted by

```
cdo setreftime,1-01-01,00:00,hours -setcalendar,standard input.nc output.nc
```

where `hours` has to be replaced e.g. by `days` or another appropriate time step if the time step given in the input file differs.

4.2 Self-generated formats

4.2.1 Binary data format

Binary data files are those written with `form="unformatted"` by FORTRAN programs. For input their structure has to be known and provided by the specifications `@lon:`, `@lat:`, `@fdt:`, `@ldt:` and eventually `@ddt:` and `@mdt`. Reading of unformatted data is considerably faster than reading formatted files. Therefore it might be a good idea to use the `-writedat` option to create a binary file from one or more other files if many runs of `cost733class` are planned. If the extension of the file name is `*.bin`, unformatted files are assumed by the program.

4.2.2 Catalog files

For some procedures (like comparison, centroid calculation, assignment of data to existing catalogs, etc.) it is necessary to read catalog files from disk. This is done using the option `-clain <specification>`, where `<specification>` is similar to the specification of the data input option. The following specifications are recognized:

- `@pth:<file name>`
- `@fdt:<first date>`
- `@ldt:<last date>`
- `@ddt:<time step, e.g. "@ddt:1d" for one day>`

- `@dtc:<number of date columns>`
- `@mdt:<list of months> e.g. @mdt:01,02,12`

See specifying data input for more details on these specifications. As with the data input, more than one classification catalog file may be used by providing more than one `-clain` option. Note that all catalog files have to be specified with all necessary flags. Especially a wrong number of date columns could lead to errors in further steps.

4.2.3 Files containing class centroids

For the assignment to existing classifications it can be necessary to read files that contain class centroids. This is done with the `-cntin <filename>` option. Although there is support for NetCDF output of class centroids in `cost733class` at the time only ASCII files are supposed to be read successfully. A suitable file might be generated with the `-cnt <filename>` option in a previous run. The extension of `<filename>` must be `.dat`. For further information see section 7.1.
Considering that such files can also be easily written with any text editor this feature of `cost733class` makes it possible to predefine class centroids or composites and assign data to them.

4.3 Specifying data input and preprocessing

The data input and preprocessing steps are carried out in the following order:

1. reading input data files (pth:)
2. grid point selection (slo: and sla:)
3. data scaling (scl:)
4. adding offset (off:)
5. centering/normalization of object (nrm:)
6. calculation of anomalies (ano:)
7. filtering (fil:)
8. area weighting (arw:)
9. sequence construction (seq:)
10. date selection (options -dlist, -per, -mon, -hrs)
11. PCA of each parameter/data set separately (pca:, pcw:)

12. parameter weighting (wgt:)

13. overall PCA (options -pca, -pcw)

For reading data sets, the software first has to know how many variables (columns) and time steps (rows) are given in the files. In case of ASCII files the software will find out these numbers by its own, if no description is given. However in this case no information about time and space is available and some functions will not work. In case of NetCDF files this information is always given in the self describing file format. Thus the description of space and time by specification flags on the command line can be omitted but will be available. Some methods like `lit` need to know about the date (provided e.g. by `-per`, `-mon`, etc.) of each object (e.g. day), or the coordinates (provided by `@lon:` and `@lat:`) while other methods do not depend on it. If necessary the given dates of a data set have to be described for each data set (or the first in case of same lengths) separately (see `@fdt:`, `@ldt:` and `@ddt:` or `@dtc:`).

After the data set(s) is(are) loaded some selections and preprocessing can be done, again by giving specification flags (key words for only one single data set without a leading `-`) and options (keywords which are applied to all data sets together with a leading `-`).

4.3.1 Specification flags

In order to read data from files which should be classified, these data have to be specified by the `-dat <specification>` option.

It is important to distinguish between specification flags to describe the given data set as contained in the input files on the one hand and flags and options to select or preprocess these data.

The `<specification>` is a text string (or sequence of key words) providing all information needed to read and process one data set (note that more data sets can be read, thus more than one specifications are needed). Each kind of information within such a string (or key word sequence) is provided by a `<flag>` followed by a `:` and a `<value>`. The different information sub strings have to follow the `-dat` option, i.e. they are recognized to belong together to one data set as long as no other option beginning with `-` appears. They may be concatenated by the `@` separator symbol directly together (without any blank (`" "`) in between) or they may be separated by one or more blanks. The order of their appearance is not important, however if one flag is provided more than once, the last one will be used. Please note that `flags` differ from `options` by the missing `-` (leading minus). The end of a specification of a data set is recognized by the occurrence of an option (i.e. by a leading `-`).

The following flags are recognized:

4.3.2 Flags for data set description

- `var:<character>`
  This is the Name of the variable. If the data are NetCDF files this must be exactly the variable name as in the NetCDF file, else it could not be found resulting in an error. If this flag is omitted the program tries to find the name on its own. This name is also used to construct the file names for NCEP/NCAR Reanalysis data file (see the `pth:` flag). In case the format is ASCII the name is not important and this flag can be omitted, except for special methods based on special parameters (e.g. the wlk-method needs uwnd, vwnd, etc.). The complete `var` flag might look e.g. like: `var:slp`

- `pth:<character>`

  - In case of ASCII format this is the path of the input file (its location directory including the file name). Please note that, depending on the system, some shortcuts for directories may not be recognized, like the symbol for the home directory. E.g.: `pth:/home/user/dat/slp.txt`.
  - In case of a `fmt:netcdf` data set consisting of multiple subsequent files the path can include `?` symbols to indicate that these letters should be replaced by a running number between the number given by `fdt:` and the number given by `ldt:` . E.g. `pth:/geo20raid/dat/ncep/slp.????.nc fdt:1948 ldt:2010` (for a NetCDF multi-file data set).
  - In case of `fmt:grib` the file name/path can include `?` symbols, or a combination of the following strings: `YYYY` / `YY`, `MM`, `DD` / `DDD` ; to indicate that these letters should be replaced by a running number between the number given by `fdt:` and the number given by `ldt:`, which has to have the same order and format as given in the path. E.g.
    `pth:/data/YYYY/MM/slp.DD.grib fdt:2011:12:31 ldt:2012:01:01` or
    `pth:/data/slp.YYYY-MM-DD.grib fdt:2011:12:31 ldt:2012:01:01` or
    `pth:/data/slp.MMDDYYYY.grib fdt:12:31:2011 ldt:01:01:2012` or
    `pth:/data/slp.YYYY.grib fdt:2011:12:31 ldt:2012:01:01` or
    `pth:/data/slp.YYYYDDD.grib fdt:2011:001 ldt:2012:365`
    (for grib multifile data sets).

- `fmt:<character>`
  This can be either ASCII, binary or NetCDF. `ftm:ascii` means the data are organized in text files: Each line holds one observation (usually the values of a day). Each column holds one parameter specifying the observations. Usually the columns represent grid points. These files have to hold a rectangular matrix of values, i.e. all lines have to have the same number of columns. Columns have to be separated by one or more blanks (" ") or by commas. Missing values are not

allowed. The decimal marker must be a point!
`fmt:netcdf` means the data should be extracted from NetCDF files. If the file name extension of the provided file name is `.nc` NetCDF format is assumed. The information about data dimensions are extracted form the NetCDF files.
`fmt:binary` denotes unformatted data. The dimensions of the data set have to provided by the specifications `@lon:`, `@lat:`, `@fdt:`, `@ldt:` and eventually `@ddt:` and `@mdt`. This format is assumed if the file name extension is `bin`. The default, which is assumed if no `fmt:` flag is given, is ASCII.

- `dtc:<integer>`
  If the file format is `fmt:ascii` and there are leading columns specifying the date of each observation, this flags has to be used to specify how many date columns exist: "1" means that there is only one date column (the first column) holding the year, "2" means the two first columns hold the year and the month, "3" means the first three columns hold the year, the month and the day and "4" means the first four columns holds the year, month, day and hour of the observation.

- `fdt:YYYY:MM:DD:HH`
  first date of data in data file (date for first line or row). Hours, days and months may be omitted. For data with `fmt:netcdf` in multiple files, it can be an integer number indicating the first year or running number which will be inserted to replace the `?` placeholder symbols in the file name.

- `ldt:YYYY:MM:DD:HH`
  last date of data in data file (date for last line or row). Hours, days and months may be omitted. If so and `ldt` indicates a number of rows smaller than actually given in the file, only this smaller number of lines is used (omitting the rest of the file). For data with `fmt:netcdf` in multiple files, it can be an integer number indicating the last year or running number which will be inserted to replace the `?` placeholder symbols in the file name.

- `ddt:<int><y|m|d|h>`
  time step of dates in data file in years, months, days or hours, e.g.: `1d` for daily data. If `ddt` is omitted but both, `fdt` and `ldt` have same resolution it is automatically set to one for that temporal resolution, e.g.: `fdt:1850:01` and `ldt:2008:02` and omitting `ddt` will lead to `ddt:1m`, i.e. monthly resolution.

- `mdt:<list>`
  list of months covered in data file, e.g. `mdt:01:02:12` if only winter data are given in the file. The list separator symbol may also be the comma "," instead of the ":".

- `lon:<number>:<number>:<number>`
  This specifies the longitude dimensions of the input data as given in the file. The

first `<number>` is the minimum longitude where longitudes west of 0 degree are given by negative numbers. The second `<number>` is the maximum longitude. The third `<number>` is the grid spacing in longitudes. e.g.: `lon:-30:50:2.5` This flag denotes just the description of the input data. For NetCDF data (which are self describing) it is superfluous!

- `lat:<number>:<number>:<number>`
  This specifies the latitude dimensions of the input data. The first `<number>` is the minimum latitude where latitudes south of the equator are given by negative numbers. The second `<number>` is the maximum latitude. The third `<number>` is the grid spacing in latitudes. e.g.: `lat:30:70:2.5` Note that at the moment south-to-north order must be given in the dat file, i.e. the left columns hold the southern-most latitudes.

4.3.3 Flags for spatial data Selection

- `slo:<number>:<number>:<number>`
  If given, this flag selects a subset of grid points from the grid in the input data. The first `<number>` is the minimum longitude where longitudes west of 0 degree are given by negative numbers. The second `<number>` is the maximum longitude. The third `<number>` is the grid spacing in longitudes. The user must take care that the selected grid fits into the given grid!

- `sla:<number>:<number>:<number>`
  If given, this flag selects a subset of grid points from the grid in the input data. The first `<number>` is the minimum latitude where latitudes south of the equator are given by negative numbers. The second `<number>` is the maximum latitude. The third `<number>` is the grid spacing in latitudes. The user must take care that the selected grid fits into the given grid!

- `sle:<integer>`
  This flag specifies the atmospheric level to be read if the `fmt:ncepr` flag is given. This is relevant for NetCDF data which may contain data of more than one level. If omitted the first level in the file is selected.

4.3.4 Flags for data Preprocessing

- `seq:<integer>`
  This specifies whether this input data matrix should be extended in order to build sequences of observations for the classification. The `<integer>` specifies the length of the sequence. If this flag isn't provided then the sequence length is `"1"`, i.e. no extension will be made. If it is `"2"` then one copy of the data matrix (shifted by one row/time step into the past) will be concatenated to the right side of the

original data matrix, resulting in a doubling of the variables (columns). This copy is shifted by 1 observation (line) downwards, thus each line holds the values of the original variable and the values of the preceding observation (e.g. day). In this way each observation is characterized additionally by the information of the preceding observation and the history of the variables are included for the classification. <integer> can be as high as desired, e.g. 12 for a sequence of 12 observations. In order to keep the original number of observations for the first lines (which do not have preceding observations) the values of the same observation are copied to fill up the gaps.

- `wgt:<number>`
  If more than one data set is given each data set can be weighted relative to others, i.e. its values count more (or less) for determination of the similarity between the observations. If this flag is omitted the weight is 1.0 for all data and all data sets (each one defined by a `-dat` specification) are treated to have the same weight compared to each other by normalizing them separately as a whole (over space and time) and multiplying them with a factor `1/nvar`, where `nvar` is the number of variables or attributes of each data set. After that each data set is multiplied by the user weight `<number>`.

- `scl:<float>`
  Scaling factor to apply to the input data of this parameter.

- `off:<float>`
  Offset value which will be added to input data after scaling.

- `nrm:<integer>`
  row-wise normalization: 1 = row-wise centralization of objects/patterns, 2 = row-wise normalization of objects/patterns (sample standard deviation), 3 = row-wise normalization of objects/patterns (population standard deviation)

- `ano:<integer>`, where integer can be
  - -1 : column wise centralization of variables (grid points) after selection of time steps
  - -2 : column wise normalization of variables using sample standard deviation (sum of squared deviations divided by n) after selection of time steps
  - -3 : column-wise normalization of variables using population standard deviation (sum of squared deviations divided by n-1) after selection of time steps
  - 1 : column wise centralization of variables (grid points) before selection of time steps
  - 2 : column wise normalization of variables using sample standard deviation (sum of squared deviations divided by n) before selection of time steps

3 : column-wise normalization of variables using population standard deviation (sum of squared deviations divided by n-1) before selection of time steps

11 column-wise centralization of variables using the daily long-term mean for removing the annual cycle

12 column-wise normalization of variables using the daily long-term mean and sample standard deviation for removing the annual cycle,

13 column-wise normalization of variables using the daily long-term mean and population standard deviation for removing the annual cycle,

21 column-wise centralization of variables using the monthly long-term mean for removing the annual cycle,

22 column-wise normalization of variables using the monthly long-term mean and sample standard deviation for removing the annual cycle,

23 column-wise normalization of variables using the monthly long-term mean and population standard deviation for removing the annual cycle,

31 column-wise centralization of variables using the daily long-term mean of 31-day wide moving windows for removing the annual cycle

32 column-wise normalization of variables using the daily long-term mean and sample standard deviation of 31-day wide moving windows for removing the annual cycle

33 column-wise normalization of variables using the daily long-term mean and population standard deviation of 31-day wide moving windows for removing the annual cycle

- `fil:<integer>`
  gaussian time filter of period <int>; int<0 = high-pass, int>0 = low-pass

- `arw:<integer>`
  Area weighting of input data grids: 0 = no (default), 1 = cos(latitude), 2 = sqrt(cos(latitude)), 3 = calculated weights by area of grid box which is the same as cos(latitude) of option 1.

- `pca:<integer|float>`
  parameter-wise pca: If provided, this flag triggers a principal component analysis for compression of the input data. The PCA retains as many principal components as needed to explain at least a fraction of `<real>` of the total variance of the data or as determined by `<integer>`. This can speed up the classification for large data sets considerably. Useful values are 0.99 or 0.95.

- `pcw:<integer|float>`
  parameter-wise pca with weighting of scores by explained variance: This preprocessing step works like `pca` but, weights the PCs according to their explained

variance. This simulates the original data set in contrast to the unweighted PCs. In order to simulate the Euclidean distances calculated from the original input data (for methods depending on this similarity metric), the scores of each PC are weighted by sqrt(exvar(PC)) if `-wgttyp euclid` is set. Therefore if `-pcw 1.D0` is given (which actually doesn't lead to compression since as may PCs as original variables are used) exactly the same euclidean distances (relative to each other) are calculated.

4.3.5 Flags for data post processing

- `cnt:<file>.<ext>`
  write parameter composite (centroid) to file of format depending on extension: .txt = ascii-xyz data, .nc = netcdf,

4.3.6 Options for selecting dates

If there is at least one input data set with given flags for date description, this information about the time dimension can be used to select dates or time steps for the procedure. Such a selection applies to all data sets if more than one is used.

Note that the following switches are options beginning by a `-` because they apply to the whole of the data set:

- `-per YYYY:MM:DD:HH,YYYY:MM:DD:HH,nX`
  This option selects a certain time period, starting at a certain year (1st YYYY), month (1st MM), day (1st DD) and hour (1st HH), ending at another year (2nd YYYY), month (2nd MM), day (2nd DD) and hour (2nd HH), and stepping by n time units (nX). The stepping information can be omitted. Time unit X can be:
  - `y` for years
  - `m` for months
  - `d` for days
  - `h` for hours. Stepping for hours must be 12, 6, 3, 2 or 1

  The default stepping is 1 while the time unit for the stepping (if omitted) is assumed to be the last date description value (years if only is given YYYY, months if also MM is provided, days for DD or hours for HH).

- `-mon 1,2,3`
  In combination with the `-per` argument the number of months used by the software can be restricted by the `-mon` option followed by a list of month numbers.

- `-dlist <file>`
  Alternatively it is possible to select time steps (continuously or discontinuously)

and for both NetCDF and ASCII data by the `-dlist <file>` option. The given file should hold as many lines as time steps should be selected and four columns for the year, month, day and hour of the time step. These numbers should be integer and separated by at least one blank. Even if no hours are needed they must be given as a dummy.

Note that it is possible to work with no (or fake) dates, allowing for input of ASCII-data sets which have been selected by another software before.

4.3.7 Using more than one data set

More than one data set can be used by just giving more than one `-dat <specification>` option (explained above) within the command line. The data of a second (or further) data set are then pasted behind the data of the first (or previous) data set as additional variables (e.g. grid points or data matrix columns) thus defining each object (e.g. day or data matrix rows) additionally. Note that these data sets must have the same number of objects (lines or records) in order to fit together.

If one of the data sets is given in different physical units than the others (e.g. hPa and Kelvin), there must be taken care of the effect of this discrepancy for the distance metric used for classification.

1. For all metrics the data set with higher total variance (larger numbers) will have larger influence.
2. For correlation based metrics, if there is a considerable difference of the mean between two data sets, this jump can result in strange effects. E.g. for t-mode PCA (PPT) there might be just one single class explaining this jump while all other classes are empty.

Therefore in case of using more than one data set (i.e. multi-field classification) each data set (separately from the others) should be normalized as a whole (one single mean and standard deviation for all columns and rows of this single data set together). This can be achieved by using the flag

- `@wgt:<float>`

at least for one data set. Where `<float>` can be `1.D0` (the `D` is for double precision) in order to apply a weight of 1.0.

If a weighting factor is given using the flag `wgt:<float>`, the normalized data sets are multiplied with this factor respectively. The default value is $1.D0$, however it will be applied only if at least for one data set the flag `@wgt:<float>` is given! If this is the case, always all data sets will be normalized and weighted (eventually with $1.D0$ if no individual weight is given).

In case of methods using the Euclidean distance the square root of the weight is used in order to account for the square in the distance calculations. This allows for a correct weight of the data set in the distance measure. Also different numbers of variables (grid points) are accounted for by dividing the data by the number of variables. Therefore the weight is applied to each data set as a whole in respect to the others. If e.g. a weight of 1.0 is given to data set A (`wgt:1.0`) and a weight of 0.5 is given to data set B (`wgt:0.5`) then data set A has a doubled influence on the assignment of objects to a type compared to data set B, regardless how many variables (grid points) data set A or B has and regardless what the units of the data has been.

It is advisable always to specify the `wgt:<float>` flag for all data sets if multiple data sets are used. E.g.

```
cost733class -dat pth:slp.txt wgt:1.D0 -dat pth:shum.txt wgt:1.D0 -met XYZ
```

4.3.8 Options for overall PCA preprocessing of all data sets together

Apart from the possibility to apply Principal Component Analysis to each parameter data set separately, the option `-pca <float|integer>` allows a final PCA of the whole data set. As with the separate procedure the number of PC's to retain can be given as an integer number or, in case of a floating number, the fraction of explained variance to retain can be specified. Also automatic weighting of each PC with it's fraction of explained variance can be obtained by using the `-pcw <float|int>` flag instead of `-pca`.

4.4 Examples

4.4.1 Simple ASCII data matrix

```
cost733class -dat pth:era40_MSLP.dat fmt:ascii lon:-37:56:3 lat:30:76:2 -met kmeans
    -ncl 9
```

This will classify data from an ASCII file `(era40_MSLP.dat)` which holds as many columns as given by the grid of -37 to 56 degree longitude (3 degree spacing) and 30 to 76 degree latitude (2 degree spacing). Thus the file has `(56-(-37))/31` by (76-30)/2+1 columns (=32x24=768). The ordering of the columns follows the principle: longitudes vary fastest from west to east, latitudes vary slowest from south to north. Thus the first column is for the southernmost latitude and the westernmost longitude, the second column for the second longitude from west on the southernmost latitude, etc. The last column is for the easternmost and northernmost grid point as illustrated in table 4.4.1
Note that the command line could have been written as

```
cost733class -dat pth:era40_MSLP.dat lon:-37:56:3 lat:30:76:2 -met kmeans -ncl 9
```

	colum #1 lon=-37,lat=30	column #2 lon=-34,lat=30	...	column #32 lon=56,lat=30	column #33 lon=-37,lat=32	...	column #768 lon=56,lat=76
t=1	mslp(-37,30,1)	mslp(-34,30,1)	...	mslp(56,30,1)	mslp(-37,32,1)	...	mslp(56,76,1)
t=2	mslp(-37,30,2)	mslp(-34,30,2)	...	mslp(56,30,2)	mslp(-37,32,2)	...	mslp(56,76,2)
...	...	...	...	...	...	...	...
t=nt	mslp(-37,30,nt)	mslp(-34,30,nt)	...	mslp(56,30,nt)	mslp(-37,32,nt)	...	mslp(56,76,nt)

Table 4.1: Example data matrix of mean sea level pressure values formatted to hold grid points in columns varying by $lon = -37, 56$ by 3° and by $lat = 30, 76$ by 2° and time steps $t = 1, nt$ in rows.

2000	01	01	-12.1
2000	01	02	-10.3
2000	01	03	-6
2000	01	04	-7.3
2000	01	05	-7.7
2000	01	06	-6.6
2000	01	07	-8
2000	01	08	-9.1
2000	01	09	-10.3
2000	01	10	-11.1
2000	01	11	-12
2000	01	12	-8.7
2000	01	13	-11.2
2000	01	14	-9.6
...	...	...	...
2008	12	31	-8

Table 4.2: Example ASCII file contents including three date columns for year, month and day.

to achieve exactly the same result. All flags are recognized to belong to the `-dat` option until another option beginning with - appears. Note that in this example no information about the time was made, i.e. the date of each line is unknown to the program. In this case this is unproblematic because k-means doesn't care about the meaning of objects, it just classifies it according to their attributes (the pressure values at the grid points in this case).

4.4.2 ASCII data file with date columns

```
cost733class -dat pth:station/zugspitze_2000-2008_Tmean.dat dtc:3 ano:33 -met BIN -ncl
    10
```

In this example the software is told, that there are three leading columns in the file holding the date of each line (year, month, day because there are 3 columns and this is the hierarchy per definition, there is no way to change this order). The total number of columns (four) and the number of rows is detected by the software itself by counting the blank-gaps in the lines. The beginning of this file looks like shown in table 4.4.2 Also there is an option `ano:33` in order to use the anomalies (deviation) referring to the long-term daily mean. Afterwards the method `BIN` is called which by default just calculates equally spaced percentile thresholds to define 10 (`-ncl 10`) classes, to which the data is assigned. Note that this method uses only the first column (after any date columns of course).

4.4.3 NetCDF data selection

```
cost733class -dat var:hgt pth:/data/20thC_ReanV2/hgt.????.nc fdt:1871 ldt:2008
    slo:-50:50:2.0 sla:20:50:2.0 sle:0950 -writedat
    hgt_0950_slo-50-50_sla-20-50_1871-2008.bin
```

This command selects a sub grid from 50°west to 50°east by 2°and from 20°north to 50°north (by 2°) out of the global reanalysis (`20thC_ReanV2`) grid. After input in this case the data are written to a binary file, which can be used for further classification by subsequent runs. But of course it would have been possible to provide a `-met` option to start a classification in this run. Note that this call needs 1.2 GB of memory.

4.4.4 Date selection for classification and centroids

```

cost733class -dat pth:grid/slp.dat lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1
    ldt:2008:12:31 ddt:1d -met CKM -ncl 8 -per 2000:1:1,2008:12:31,1d -mon 6,7,8 -v 3

cost733class -dat pth:grid/slp.dat lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1
    ldt:2008:12:31 ddt:1d cnt:DOM00.nc -clain pth:CKM08.cla dtc:3 fdt:2000:6:1
    ldt:2008:8:31 mdt:6:7:8 -per 2000:1:1,2008:12:31,1d -mon 6,7,8 -met CNT -v 3
```

Here classification is done only for summer months (`-per 2000:1:1,2008:12:31,1d -mon 6,7,8`) and written to `CKM08.cla` in the first run. In the second call `CKM08.cla` is used to build centroids, again only for the summer months. Note that `mdt:6:7:8` has to be provided for the `-clain` option in order to read the catalog correctly.

5 Data output

5.1 The classification catalog

This is a file containing the list of resulting class numbers. Each line represents one classified entity. If date information on the input data was provided the switch `-dcol <int>` can be used to write the datum to the classification output file as additional columns left to the class number. The argument `<int>` decides on the number of date columns: `-dcol 1` means only one column for the year or running number in case of fake dates (1i4), `-dcol 2` means year and month (1i4,1i3), `-dcol 3` means year, month and day (1i4,2i3) and `-dcol 4` means year, month, day and hour (1i4,3i3). If the `-dcol` option is missing the routine tries to guess the best number of date columns. This number might be important if the catalog file is used in subsequent runs of cost733class e.g. for evaluation etc.

5.2 Centroids or type composites

If the `-cnt <filename>` option is given, a file is created which contains the data of the centroids or class means. In each column of the file the data of each class is written. Each line/row of the file corresponds to the variables in the order (of the columns) provided in the input data.
If the `cnt:<file.nc>` flag has been given within the individual specification of a data set, the corresponding centroids for this parameter will be written to an extra file. Thus it is possible to easily discern the type composites for different parameters. The file format depends on the extension of the offered file name: the file name extension ".nc" leads to NetCDF output (which might be used for plotting with grads), the file name extension ".txt" denotes ASCII output including grid point coordinates in the first two columns if available, while the extension "dat" skips the coordinates in any case (useful for viewing or for further processing e.g. with `-met asc`).

5.3 Output on the screen

The verbosity of screen output can be controlled by the `-v <int>` flag. <int> can be:

0 : NONE: errors only

1 : MAIN: warnings, major calls and proceeding

2 : SUB: major subroutine's calls and proceeding, routine's major results

3 : DET: detailed/all calls/proceeding, routine's intermediate results (no arrays)

4 : ALL: extensive results (arrays etc.)

At the end of each classification the explained cluster variance (ECV) is calculated and printed on the screen.
Further on the final class frequencies are printed on the screen.

5.4 Output of the input data

Providing the option `-writedat <filename>` writes the input data to a single file as it would be used for classification, i.e. after application of the preprocessing steps if any. The type of the resulting file depends on its extension. `.bin` causes unformatted binary output whereas any other extension produces simple ASCII output. Note that this function can be used to prepare and compile data sets which can be used for input into `cost733class` or any other software later. Thus if no method is specified `cost733class` can be used as a pure data processing tool.

5.5 Output of indices used for classification

Some methods are based on indices which are calculated as intermediate results (e.g. PCA methods calculate scores or loadings time series). The flag `-idx <filename>` causes those methods to write out these indices into the according file.

5.6 Opengl graphics output

If compiled with opengl support (./configure –enable-opengl) the switch -opengl opens a x11-window for visualization of the classification process for some optimization methods. Additionally the switch -gljpeg produces single image files for each frame which can be used to produce an animation, eg. by using the unix command:

```
ffmpeg -y -r 20 -sameq -i out\%06d.jpg -vcodec mjpeg -f avi cost733ckm.avi
```

6 Classification methods

6.1 Methods using predefined types

6.1.1 INT | interval | BIN | binclass

This method classifies the objects into bins which are defined either by thresholds calculated as the fraction of the range of a variable or by percentiles of the variable distribution.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes/bins. Default is 9.

- `-svar <int>`:
  The variable/column number of the input data set which should be used to calculate the bin thresholds. Default is 1.

- `-crit <int>`:
  Determine the threshold type, where `<int>` may be:

  1 : Means the threshold is the i'th percentile where, i is cl*1/ncl.

  2 : Means the ncl bins are centered around the mean value and have the size of the dev*2/ncl, where dev is the largest deviation from the mean within data set.

  3 : Means the bin size is the data range divided by ncl, the bins are not centered.

  4 : Classification into 2 bins: a lower of values less than `-thres <real>` for `-svar <int>` and one above.

  5 : As in `-crit 4`: but the threshold is interpreted as a percentile threshold between 0 and 100%.

- `-dist 0`:
  For `-crit 1` and `-crit 5` the percentiles are calculated without the minimum value of var. This is useful e.g. for daily precipitation with many dry days.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.????.nc fmt:netcdf fdt:1951 ldt:1980 -met INT -ncl 9 -crit 1
    -svar 1 -cla INT.cla -dcol 4
```

Another example:

```
cost733class -dat pth:slp.????.nc fmt:netcdf fdt:1951 ldt:1980 -met INT -ncl 20 -crit
    2 -cnt INT.cnt -cla INT.cla -dcol 4
```

This run classifies the input data into 20 bins and writes a classification catalog and centroid file.

6.1.2 GWT | prototype - large scale circulation types

This method uses three prototype patterns and calculates the three Pearson correlation coefficients between each field in the input data set and the three prototypes (Beck et al., 2007). The first prototype is a strict zonal pattern with values increasing from north to south. The second is a strict meridional pattern with values increasing from west to east. And the third is a cyclonic pattern with a minimum in the center and increasing values to the margin of the field. Depending on the three correlation coefficients and their combination each input field is classified to one class. Since there are only fixed numbers of combinations, not all numbers of types can be achieved. This method makes sense only for single pressure fields. The possible numbers of types are: 8, 10, 11, 16, 18, 19, 24, 26, 27. For 8 types the main wind sectors (N, NE, E, SE, S, SW, W, NW) are used. Two additional types for pure cyclonic and pure anticyclonic situations lead to 10 types and an indifferent type according to cyclonicity to 11 types. For 16 types the following numbers apply: 1-8=cyclonic, 9-16=anticyclonic and for 24: 1-8=cyclonic, 9-16=anticyclonic, 17-24=indifferent. Adding 2 or 3 cyclonicity types then results in 18 or 19 and 26 or 27 types accordingly.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. Grid description is necessary!

The following options define the classification:

- `-ncl <int>`:
  The number of types as described above. Default is 8.
- `-crit <int>`:
  Vorticity index. `<int>` can be:
  - `1`: No normalization of the vorticity index.
  - `2`: Normalization of the vorticity index.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-idx <basename>`:
  Output base name for pattern correlations.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. The three prototype patterns are written to NetCDF files (`proto001.nc` to `proto003.nc`) in the directory where `cost733class` was executed (only for `-v 4` and higher). Overall class centroids as well as a file containing the pattern correlations at each time step are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-37:56:3 lat:30:76:2 -met GWT -ncl 8 -crit
    1 -cla GWT09.cla -dcol 3
```

Another example:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-37:56:3 lat:30:76:2 -met GWT -ncl 27
    -crit 2 -cla GWT27.cla -dcol 3
```

6.1.3 GWTWS | gwtws - large scale circulation types

Based on GWT (Cap. 6.1.2). The classification is done using GWT with 8 types for the 500mb geopotential. If the mean wind speed at 500mb (derived from the geopotential field) is lower than 7m/s, it's "convective", resulting in one of the following three types:

- if the mean sea level pressure is lower than 1010mb it's "low" (type 9)
- if the mean sea level pressure is higher than 1015mb it's "high" (type 10)
- else or if the mean wind speed at 500mb is lower than 3m/s it's "flat" (type 11)

If the mean wind speed at 500mb is higher than 7m/s, it's "advective", and the types are identical to GWT using 8 types.

Options

Strictly necessary options:

1. `-dat <specification>`:
   Input data, the 500mb geopotential field. Grid description is necessary!
2. `-dat <specification>`:
   Input data, the sea level pressure, as field or mean value. Grid description is necessary!

The following options define the classification:

- `-crit <int>`:
  Handling of thresholds. `<int>` can be:
  1 : the four thresholds (default: 7m/s, 3m/s, 1010mb and 1015mb) can be varied in the following matter:
    * a mean windspeed at 500mb lower than `-alpha <real>` [m/s] will result in one of the following three types:

  - ∗ if the mean sea level pressure is lower than `-gamma <real>` [mb] it's "low" (type 9)
  - ∗ if the mean sea level pressure is higher than `-delta <real>` [mb] it's "high" (type 10)
  - ∗ if the mean windspeed at 500mb is lower than `-beta <real>` [m/s] it's "flat" (type 11)

2 : the four thresholds (default: 0.275, 0.073, 0.153 and 0.377) can be varied half-automatized as the values for mean wind speed at 500mb and mean sea level pressure rise or descend a given percentile (alpha, beta, gamma or delta [0,1]):

  - ∗ a mean windspeed at 500mb lower than `-alpha <real>` will result in one of the following three types:
  - ∗ if the mean sea level pressure is lower than `-gamma <real>` it's "low" (type 9)
  - ∗ if the mean sea level pressure is higher than `-delta <real>` it's "high" (type 10)
  - ∗ if the mean windspeed at 500mb is lower than `-beta <real>` it's "flat" (type 11)

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-idx <basename>`:
  Output base name for means of MSLP and wind speed at 500mb per time step.
- `-dcol <int>`:
  Number of date columns in both files mentioned above.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. The three prototype patterns are written to NetCDF files (`proto001.nc` to `proto003.nc`) in the directory where `cost733class` was executed. Parameter wise class centroids as well as one file containing means of MSLP and wind speed at 500mb for each time step are optional.

Examples

An example with default values:

```
cost733class -dat pth:hgt500.dat fmt:ascii lon:-37:56:3 lat:30:76:2 -dat pth:slp.dat
    fmt:ascii lon:-37:56:3 lat:30:76:2 -met GWTWS -crit 1 -alpha 7 -beta 3 -gamma 1010
    -delta 1015 -cla GWTWS.cla -dcol 3
```

Another example:

```
cost733class -dat pth:hgt500.dat fmt:ascii lon:-37:56:3 lat:30:76:2
    cnt:GWTWS_hgt500.cnt -dat pth:slp.dat fmt:ascii lon:-37:56:3 lat:30:76:2
    cnt:GWTWS_slp.cnt -met GWTWS -crit 2 -alpha 0.275 -beta 0.073 -gamma 0.153 -delta
    0.377 -cla GWTWS.cla -dcol 3
```

Running this command, a classification catalog will be produced as well as parameter wise class centroids.

6.1.4 LIT | lit - litynski threshold based method

Litynski (1969) developed a classification scheme based on sea level pressure maps for the Polish region. The original LIT classification is based on three indices: meridional Wp, zonal Ws and $Cp = Pcentral$, where $Pcentral$ is the pressure at the central grid point for the domain. The main steps include:

- Calculate two indices: meridional Wp, zonal Ws . Wp and Ws are defined by the averaged components of the geostrophical wind vector and describe the advection of the air masses.
- Select the Cp index: $Cp = Pcentral$
- Calculate lower and upper boundary values for Wp, Ws and Cp for each month as the 33rd percentile (assuming normal distribution).
- Interpolate these thresholds throughout the year for each day.
- We have component N (Wp), E (Ws), C (Cp) when the indices Wp, Ws and Cp for a day are less than the lower boundary value
- We have component 0 (Wp), 0 (Ws), 0 (Cp) when the indices are between lower and upper boundary values
- We have component S (Wp), W (Ws), A (Cp) when the indices aren't less than the upper boundary value
- Finally, the types are the 27 superpositions of these three components.

Thus the 27 types are defined by:

```
        case('N0C')  ;  cla(obs)=1
        case('N00')  ;  cla(obs)=2
        case('N0A')  ;  cla(obs)=3
        case('NEC')  ;  cla(obs)=4
        case('NE0')  ;  cla(obs)=5
        case('NEA')  ;  cla(obs)=6
        case('0EC')  ;  cla(obs)=7
        case('0E0')  ;  cla(obs)=8
        case('0EA')  ;  cla(obs)=9
        case('SEC')  ;  cla(obs)=10
        case('SE0')  ;  cla(obs)=11
        case('SEA')  ;  cla(obs)=12
        case('S0C')  ;  cla(obs)=13
        case('S00')  ;  cla(obs)=14
        case('S0A')  ;  cla(obs)=15
        case('SWC')  ;  cla(obs)=16
        case('SW0')  ;  cla(obs)=17
        case('SWA')  ;  cla(obs)=18
        case('0WC')  ;  cla(obs)=19
        case('0W0')  ;  cla(obs)=20
        case('0WA')  ;  cla(obs)=21
        case('NWC')  ;  cla(obs)=22
        case('NW0')  ;  cla(obs)=23
        case('NWA')  ;  cla(obs)=24
        case('00C')  ;  cla(obs)=25
        case('000')  ;  cla(obs)=26
        case('00A')  ;  cla(obs)=27
```

18 types are achieved by omitting the intermediate interval for Cp according to the following code:

```
        case('N0C')  ;  cla(obs)=1
        case('N0A')  ;  cla(obs)=2
        case('NEC')  ;  cla(obs)=3
        case('NEA')  ;  cla(obs)=4
        case('0EC')  ;  cla(obs)=5
        case('0EA')  ;  cla(obs)=6
        case('SEC')  ;  cla(obs)=7
        case('SEA')  ;  cla(obs)=8
        case('S0C')  ;  cla(obs)=9
        case('S0A')  ;  cla(obs)=10
        case('SWC')  ;  cla(obs)=11
        case('SWA')  ;  cla(obs)=12
        case('0WC')  ;  cla(obs)=13
        case('0WA')  ;  cla(obs)=14
        case('NWC')  ;  cla(obs)=15
        case('NWA')  ;  cla(obs)=16
        case('00C')  ;  cla(obs)=17
        case('00A')  ;  cla(obs)=18
```

Ignoring the Cp index completely leads to 9 types:

```
        case('N0')  ;  cla(obs)=1
        case('NE')  ;  cla(obs)=2
        case('0E')  ;  cla(obs)=3
        case('SE')  ;  cla(obs)=4
        case('S0')  ;  cla(obs)=5
        case('SW')  ;  cla(obs)=6
        case('0W')  ;  cla(obs)=7
        case('NW')  ;  cla(obs)=8
        case('00')  ;  cla(obs)=9
```

In order to calculate geostrophic wind it is necessary to provide the coordinates of the input data set. Also this method is dedicated to sea level pressure since it includes a fixed threshold. Finally the dates of the objects in the input file have to be provided in order to calculate the annual cycle.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data, the sea level pressure. Grid description is necessary!

The following options define the classification:

- `-ncl <int>`:
  The number of types as described above. Default is 9.
- `-mod`:
  Set number of days per month to 30 (model months).

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met LIT -ncl 9 -cla LIT09.cla -dcol 3
```

Another example:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met LIT -ncl 27 -cla LIT27.cla -dcol 3 -cnt LIT27.cnt
```

Same but with a different number of classes and output of centroids.

6.1.5 JCT | jenkcol - Jenkinson-Collison Types

The classification scheme according to Jenkinson and Collison (1977) is based on the variability of 16 selected grid points of the pressure field around the region of interest. The possible numbers of types are: 8, 9, 10, 11, 12, 18, 19, 20, 26, 27 and 28.

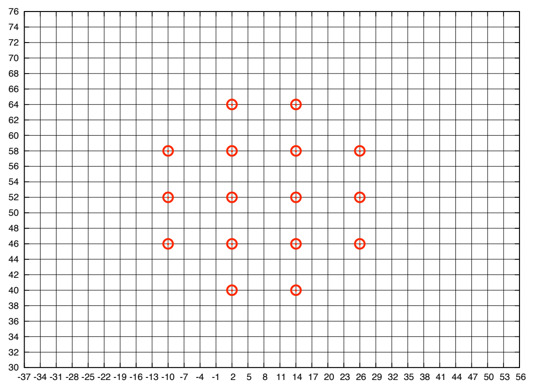


Figure 6.1: Selected grid-points (red circles) for ERA40-domain 00 [37W - 56E (32pts by 3°), 30N - 76N (24pts by 2°)]. 37W;30N is the first position in data, 56E;76N the last. The classification grid has an extent of 36°E - W x 24°N - S and is centered in 8E;52N.

This method was developed by Jenkinson and Collison (1977) and is intended to provide an objective scheme that acceptably reproduces the subjective Lamb weather types (Jenkinson and Collison, 1977; Jones et al., 1993). Therefore daily grid-point mean sea-level pressure data is classified as follows:

1. First the input data is analysed due to its grid:
   - Because the Jenkinson-Collison scheme uses 16 grid points out of a 5°north-south and 10°east-west resolved grid, with an overall extent of 20°x 30°latitude by longitude, the classification-grid has to be altered if resolution and extent of the data-grid differ (see Fig. 6.1). If the data spreads a broader region, the classification-grid is centered in the middle of it. If you use -crit 2 (default is -crit 1) the classification grid is extended to the whole data region (see Fig. 6.2).
   - Moreover the 16 grid points are chosen out of the grid; only their data is read in from the input-data-file and considered further on.
2. In a second step the classification criteria are calculated:

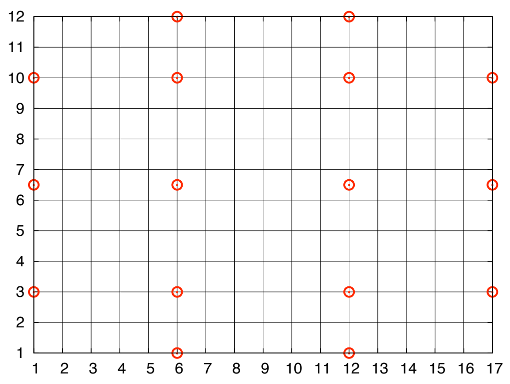
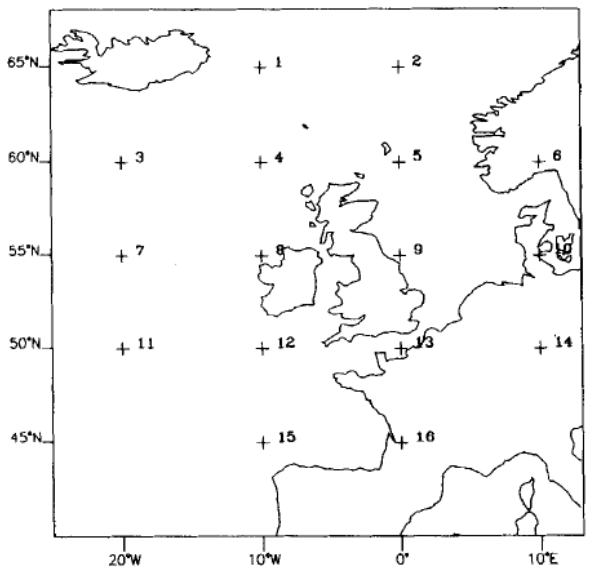


Figure 6.2: Selected grid-points (red circles) for a 17x12 grid, if -crit 2 is used. Linear interpolation between the neighboring points is done to achieve the values for the middle line. The grid is no more equally spaced in latitudinal direction.

- Therefore also some adjustment is done according to the relative grid-point spacings and the middle latitude of the classification region (Tang et al., 2008).
- Now the wind-flow characteristics are computed (Jones et al., 1993), thereafter each day's pressure pattern is represented by westerly, southerly and resultant flow, as well as westerly, southerly and total shear vorticity.

3. Finally the classification is done. By reason of being predefined, the possible numbers of types are 8, 9, 10, 11, 12, 18, 19, 20, 26, 27 and 28, explained as follows:
    - Types 1 to 8 (in 8 to 12 selected types) are using the prevailing wind directions (W, NW, N, NE, E, SE, S, and SW, where W = 1 etc.).
    - Types 1 to 16 (in 18, 19 or 20) are a subdivision of the 8 directional types into tendentious cyclonic and anticyclonic (1-8=cyclonic, 9-16=anticyclonic).
    - Types 1 to 24 (in 26, 27 or 28) combine mainly directions with partly cyclonic and anticyclonic ones (1-8=cyclonic, 9-16=straight, 17-24=anticyclonic).
    - Types 9/10, 17/18 or 25/26 indicate pure cyclonic / anticyclonic days.
    - The 9th, 11th, 19th or 27th type stands for a light indeterminate flow class and can be treated as unclassified (except for 10 types, where the 9th is something else).
    - Type 12, 20 or 28 figures out a gale day.



$$W = \tfrac{1}{2}(12+13) - \tfrac{1}{2}(4+5) \qquad \text{(westerly flow)}$$

$$S = 1{\cdot}74[\tfrac{1}{4}(5+2\times 9+13) - \tfrac{1}{4}(4+2\times 8+12)] \qquad \text{(southerly flow)}$$

$$F = (S^2 + W^2)^{1/2} \qquad \text{(resultant flow)}$$

$$ZW = 1{\cdot}07[\tfrac{1}{2}(15+16) - \tfrac{1}{2}(8+9)] - 0{\cdot}95[\tfrac{1}{2}(8+9) - \tfrac{1}{2}(1+2)] \qquad \text{(westerly shear vorticity)}$$

$$ZS = 1{\cdot}52[\tfrac{1}{4}(6+2\times 10+14) - \tfrac{1}{4}(5+2\times 9+13) - \tfrac{1}{4}(4+2\times 8+12) + \tfrac{1}{4}(3+2\times 7+11)] \qquad \text{(southerly shear vorticity)}$$

$$Z = ZW + ZS \qquad \text{(total shear vorticity)}$$

Figure 6.3: Grid point and index scheme from Jones et al. (1993)

Because the method is specialized on classifying daily mean sea-level pressure patterns only, some thresholds are hard coded to distinguish flow intensities. This is of course problematic if this method should be applied e.g. on geopotential heights. Therefore there are options modifying the default classification scheme.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. Grid description is necessary!

The following options define the classification:

- `-ncl <int>`:
  The number of types as described above. Default is 9.

- `-thres <real>`:
  If -thres is set to 1, no "weak" flows are classified (class 9, 11, 19 or 27; depending on -ncl <int>). Use this for other variables than MSLP.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

- `-idx <basename>`:
  Output base name for values of wind-flow characteristics (w/s/f=westerly/southerly/resultant flow, zw/zs/z=westerly/southerly/total shear vorticity).

Output

This method returns one file containing the classification catalog. Overall class centroids as well as a file containing values of wind-flow characteristics are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met JCT -ncl 9 -cla JCT09.cla -dcol 3
```

Another example:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met JCT -ncl 26 -idx JCT26  -cla JCT26.cla -dcol 3
```

Same but with a different number of classes and output for values of wind-flow characteristics.

6.1.6 WLK | wlk - automatic weather type classification according to German metservice

This method is based on the OWLK (objective weather type classification) by Dittmann et al. (1995) and Bissolli and Dittmann (2003), originally including 40 different types. The types are defined according to the wind field (U and V) of a certain level, as well as according to cyclonicity of pressure fields of a first and a second level and according to total precipitable water, the latter (cyclonicity and water) only if the corresponding data sets are provided at the command line. In comparison to the initial OWLK method the recent classification WLKC733 which is included in cost733cat provides a few simplifications regarding output parameters (see Philipp et al., 2010).

The alphanumeric output consists of five letters: the first two letters denote the flow pattern in terms of a dominant wind sector counting clockwise, i.e. 01 = NE, 02 = SE, 03 = SW, 04 = NW and 00 = undefined (varying directions). For determination of the dominant wind sector the true wind direction obtained from U and V-components at 700 hPa is used and circulation patterns are derived from a simple majority (threshold) of the weighted wind field vectors at 700 hPa. If no majority could be found, the class 00 will be selected. For counting the respective wind directions a weighting mask, putting higher weights on grid points in the center of the domain, is applied.

The third and fourth letter denote Anticyclonicity or Cyclonicity at 925 hPa and 500 hPa, respectively, based on the weighted mean value of the quasi-geostrophic vorticity, again putting higher weights on central grid points. The fifth letter denotes Dry or Wet conditions, according to an weighted area mean value of the towering water content (whole atmospheric column) which is compared to the long-term daily mean.

In order to achieve a classification system for 28 types (WLKC28) six main wind sector types are used (01 = 330-30°and so on in 60°steps) plus one undefined type, which are further discriminated by cyclonicity as described above. 18 types (WLKC18) are produced by using nine wind sector types (sector 01 = 345-15°, 02 = 15-75°, 03 = 75-105°, 04 = 105-165°, 05 = 165-195°, 06 = 195-255°, 07 = 255-285°, 08 = 285-345°, 00 = undefined) and cyclonicity at 925 hPa, while the latter is omitted for producing nine types (WLKC09).

The third and fourth letter denoting cyclonicity of pressure fields of a first and a second level, and total precipitable water, respectively, are analyzed only if the corresponding data sets are provided at the command line. Thus the order of the data sets provided by the command line arguments plays an important role for this method.

It is important to note that any other meteorological parameters than U and V for the first two data fields do not make any sense for this method!

For determination of the main wind sector and cyclonicity a weigthing mask is applied to the input fields putting highest weights on the central zone, intermediate weight to the middle zone and lowest weight to the margin zone. The mask is shown on verbose level > 1.

Since within this method the number of types is defined by the number of wind sectors and the input data the `-ncl` flag can be omitted and is ignored.

Options

Strictly necessary options:

1. `-dat <specification>`:
   Input data for U-component at the 700 hPa level. Grid description is necessary!

2. `-dat <specification>`:
   Input data for V-component at the 700 hPa level. Grid description is necessary!

Optional data input:

3. `-dat <specification>`:
   Input data for geopotential at the 925 hPa level. Grid description is necessary!

4. `-dat <specification>`:
   Input data for geopotential at the 500 hPa level. Grid description is necessary!

5. `-dat <specification>`:
   Input data for towering water content (whole atmospheric column). Grid description is necessary!

There are several special flags controlling the wlk classification (beside the control via input data sets):

- `-step <integer>`:
  The number of wind sectors. The 360○ of the wind rose will be divided by `<integer>` to determine the threshold angle for the different wind sectors.

- `-shift`:
  Shift the sectors defined by `-step <integer>` by half of the angle. I.e. North will be in the center of sector.

- `-thres <real>`:
  fraction of grid points for decision on main wind sector (default=0.6)

- `-alpha <real>`:
  central weight for weighting mask (default=3.D0)

- `-beta <real>`:
  middle zone weight for weighting mask (default=2.D0)

- `-gamma <real>`:
  margin zone weight for weighting mask (default=1.D0)

- `-delta <real>`:
  width factor for weighting zones (nx*delta|ny*delta) (default=0.2)

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.
- `-idx <filename>`:
  Output file name for values of main wind sector, cyclonicity (925 and 500 hPa level) and total precipitable water.

Output

This method returns one file containing the classification catalog. Overall, per parameter class centroids as well as a file containing values of wind-flow characteristics and precipitable water are optional.

Examples

For example a classification with the maximum number of data set fields and a division into 8 wind sectors:

```
cost733class \
-dat pth:era40_U700_12Z_195709-200208.domain00.dat lon:-37:56:3 lat:30:76:2 \
-dat pth:era40_V700_12Z_195709-200208.domain00.dat lon:-37:56:3 lat:30:76:2 \
-dat pth:era40_Z925_12Z_195709-200208.domain00.dat lon:-37:56:3 lat:30:76:2 \
-dat pth:era40_Z500_12Z_195709-200208.domain00.dat lon:-37:56:3 lat:30:76:2 \
-dat pth:era40_TWC_12Z_195709-200208.domain00.dat lon:-37:56:3 lat:30:76:2 \
-met wlk -step 8
```

Another more advanced example omitting the moisture index could be:

```
cost733class \
-dat pth:/alcc/ptmp/geodata/ERA40/ascii/era40_U700_12Z_195709-200208.domain00.dat \
        lon:-37:56:3 lat:30:76:2 \
-dat pth:/alcc/ptmp/geodata/ERA40/ascii/era40_V700_12Z_195709-200208.domain00.dat \
        lon:-37:56:3 lat:30:76:2 \
-dat pth:/alcc/ptmp/geodata/ERA40/ascii/era40_Z925_12Z_195709-200208.domain00.dat \
        lon:-37:56:3 lat:30:76:2 \
-dat pth:/alcc/ptmp/geodata/ERA40/ascii/era40_Z500_12Z_195709-200208.domain00.dat \
        lon:-37:56:3 lat:30:76:2 \
-met WLK -step 8 -shift -thres 0.35 -crit 0 -alpha 7 -delta 0.15 \
-idx WLK09_YR_S01_U7-V7_D00.txt
```

producing the output:

```
starting cost733class ...
 method   = WLK
 ncl      =  27
 period   = era40
 months   = 1,2,3,4,5,6,7,8,9,10,11,12
 hours    = 00,06,12,18
 verbose  =   2

 data input ...
 applyweighting = T
 writedat        = F
 writestd        = F
 npar      =   4
 got     16436 lines from era40_U700_12Z_195709-200208.domain00.dat, done!
 got     16436 lines from era40_V700_12Z_195709-200208.domain00.dat, done!
 got     16436 lines from era40_Z925_12Z_195709-200208.domain00.dat, done!
 got     16436 lines from era40_Z500_12Z_195709-200208.domain00.dat, done!

 calling wlk ...
 number of windir sectors          =            6
 fraction of gridpoint wind        =  0.350000000000000
 anomalies of cyclonicity (1=yes) =            0
 shift sectors centering to 0 deg = T
 central weight alpha              =   7.00000000000000
 middle weight beta                =   2.00000000000000
 margin weight gamma               =   1.00000000000000
 mask zone width factor delta      =  0.150000000000000
 grids for U and V have size     32 by     24

 weight mask:
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.7.7.7.7.7.7.7.7.7.7.7.7.7.7.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.1.1.1.
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
 divided z925 by 10, min, max =   4.57263000000000        1062.87168000000
 divided z500 by 10, min, max =   4471.24336000000        5913.77969000000
 number of classes =           28

      1    00AAx  -9999.9  -9999.9          402
      2    01AAx    330.0     30.0          294
      3    02AAx     30.0     90.0           80
      4    03AAx     90.0    150.0          117
      5    04AAx    150.0    210.0          443
      6    05AAx    210.0    270.0         3690
      7    06AAx    270.0    330.0         2308
      8    00ACx  -9999.9  -9999.9           35
      9    01ACx    330.0     30.0           85
     10    02ACx     30.0     90.0           19
     11    03ACx     90.0    150.0            7
     12    04ACx    150.0    210.0           60
     13    05ACx    210.0    270.0          802
     14    06ACx    270.0    330.0          695
     15    00CAx  -9999.9  -9999.9           15
     16    01CAx    330.0     30.0          11
     17    02CAx     30.0     90.0           1
     18    03CAx     90.0    150.0           5
     19    04CAx    150.0    210.0          78
     20    05CAx    210.0    270.0        1187
     21    06CAx    270.0    330.0         584
     22    00CCx  -9999.9  -9999.9          40
     23    01CCx    330.0     30.0          69
     24    02CCx     30.0     90.0           2
     25    03CCx     90.0    150.0           5
     26    04CCx    150.0    210.0         145
     27    05CCx    210.0    270.0        3205
     28    06CCx    270.0    330.0        2052

 catalog output = WLK27_YR_S01_U7-V7-Z9-Z5_D00.txt        16436

 final result:
           WLK ecv = 0.138173112002   csize =    402    294    80   117   443  3690  2308
           35     85     19      7     60    802   695    15    11     1     5    78  1187
           584     40     69      2      5    145  3205  2052
```

6.2 Methods based on Eigenvectors

cost733class contains source code to calculate the eigenvectors of the input data set by singular value decomposition.

6.2.1 PCT | tpca - t-mode principal component analysis using oblique rotation

The classification by obliquely rotated Principal Components in T-mode (PCT) (also called TPCA, Principal Component Analysis in T-mode) is based on Huth (2000). With regard to computer resources, and simultaneously to speed up the calculation of the PCA, the data is split into subsets. Consequently, the principal components obtained for each subset are projected onto the rest of data.

1. The Data is standardized spatially: each pattern's mean is subtracted from the data; then the patterns are divided by their standard deviations.

2. The data is split into ten subsets: selecting the 1st, 11th, 21st, 31st, 41st, etc. pattern as the first subset; the 2nd, 12th, 22nd, 32nd, 42nd, etc. as the second subset, and so on.

3. In order to achieve the principal components the algorithm based on the singular value decomposition is used.

4. The definition of the numbers of principal components in cost733class is in contrast to the original scheme, where the scree-plot method is used, defined by the user (-ncl).

5. An oblique rotation (using direct oblimin) is applied on the principal components, employing a FORTRAN90 adaptation of the GPA (Gradient Projection Algorithm) according to Bernaards and Jennrich (2005).

6. The principal components are mirrored, thereafter the respective maximum absolute loadings are positive.

7. The projection of the principal component scores of each subset onto the remaining data is realized by calculating the Pearson correlation coefficients between the data and the scores.

8. To evaluate the ten classifications based on the subsets, contingency tables are used. Subsequently CHI-square is calculated between each subset's types (every subset solution is compared with the other nine ones). The subset which gains the highest sum (of the nine comparisons) is selected, and only its types are returned.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of principal components. Default is 9.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

- `-idx <basename>`:
  Output base name for scores (*.sco), loadings (*.ldg), explained variances (*.exv) and total variance(*.tvr) of the rotated principal components (step 6), correlation values (*.cor) of original data and subset scores (step 7) and subset classifications (*sub.cla).

Output

This method returns one file containing the classification catalog. Overall class centroids as well as files containing results from various intermediate steps are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PCT -ncl 9 -cla PCT09.cla -dcol 3
```

Another example with additional output files and a different number of principal components:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PCT -ncl 4 -idx PCT04 -cla PCT04.cla -dcol 3
```

6.2.2 PTT | tpcat - t-mode principal component analysis using orthogonal rotation

`PTT` is a simplified variant of t-mode principal component analysis with subsequent assignment of the cases to the PC with the maximum loading. Since there is no split into sub samples the amount of computer memory is relatively high as well as the run time. In contrast to `PCT` the PCs are rotated orthogonally.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of principal components. Default is 9.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

- `-idx <basename>`:
  Output base name for scores (*.sco) and loadings (*.ldg).

Output

This method returns one file containing the classification catalog. Overall class centroids as well as files containing results from intermediate steps are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PTT -ncl 9 -cla PTT09.cla -dcol 3
```

Another example with additional output files and a different number of principal components:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PTT -ncl 4 -idx PTT04 -cla PTT04.cla -dcol 3
```

6.2.3 PXE | pcaxtr - the extreme score method

This methodological approach has as main goal to use Principal Component Analysis as an objective way for obtaining the number of circulation types (CT hereafter) and its initial centroids that our final classification will have.

The general procedure does not vary with respect to the usual outline of PCA: S-mode structure (grid points as the columns and cases as the rows of the matrix), use of the correlation matrix to obtain the new variables, and orthogonal rotation (Varimax) to better adjust the result to the climatic reality. There are few novelties:

- spatial standardization of the gridded data before the PCA procedure.
- after aplying PCA, the use of a rule (the extreme scores) which by being fulfilled or not by each original case, leads to the final number of CTs and their centroids.

Thus, based on those cases of our sample that according to their scores show a spatial structure similar to some rotated PC (cases with high score values with respect to that component, i.e higher/lower than +2/-2 respectively, and lower scores for the rest of the PCs, i.e. between +2 and -2) we obtain the centroids for each component and phase (positive/negative). These are then our provisional CTs, with clear climatic meaning since they derive from the real data (existence in the reality of the spatial patterns identified by the PCA). Despite that, in case that a PC does not have any real case (that fulfills the rule of the extremes scores) assigned to it, this spatial pattern is considered as a statistical artifact; this theoretical CT according to PCA is eliminated and the final number of CTs of the classification diminishes.

Finally, to assign each case of the sample to one of the CTs, the euclidean distance is used. This is calculated using the multivariate coordinates of each case expressed by its scores, and the location in the PC space of the centroid of each CT. Obviously, these centroids (and number of clusters) can also be used as initial seeds for an optimization algorithm like K-means (see method PCAXTRKM - PXK). Once the entire sample has been classified, we have our catalog of final CTs. For more details about the rule of the "extreme scores" see Esteban el al 2006 and Philipp el al 2010.

We could say that this approach is an attempt to reproduce the T-mode PCA approach, that uses the coefficients obtained with PCA (correlations) for distributing the cases among the different groups considered. PXE employs the PC-scores to do this assignment, reducing substantially the calculation requirements. On the contrary, some uncertainty is introduced by the use of a similarity threshold based on the PC-scores (normally -2 /+ 2), thus the suitability of this threshold value depends on the variability of the sample. The normally used -2 /+ 2 can be inappropriate for samples with very few cases or with little variability, being recommendable to change to, for example, a -1 /+ 1 threshold (see Esteban et al 2005).

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Has to be always an even number (2, 4, 8, 10,...). Note, however, that sometimes not all classes are realized, possibly leading to empty (less) types. Default is 10.

- `-niter <int>` : Maximum number of iterations. Use `-niter 0` (the default) for direct assignment of cases using the Euclidean distance. Use `-niter >0` for a k-means cluster analysis like `-met KMN` but with starting partitions derived by `PXE`.

- `-crit <int>`:
  Further relevant switch concerning the normalization method of the input data before the PCA is performed. `<int>` can be:

  0 : Only normalize patterns for PCA (original)

  1 : Normalize patterns and normalize grid point values afterwards (default)

  2 : Normalize patterns and center grid point values afterwards

- `-thres <real>`:
  Definition of extreme score cases. Threshold defining key group (default=2.0)
- `-delta <real>`:
  Definition of extreme score cases. Score limit for other PCs to define uniquely leading PC (default 1.0)

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.
- `-idx <basename>`:
  Output base name for rotated scores (*.sco).

Output

This method returns one file containing the classification catalog. Overall class centroids as well as files containing results from intermediate steps are optional.

Examples

An example with default values:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PXE -ncl 10 -niter 0 -thres 2.0 -delta 1.0 -cla
    PXE10.cla -dcol 3
```

Another example with additional output files and a different number of principal components:

```
cost733class -dat pth:slp.dat fmt:ascii lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1:12
    ldt:2008:12:31:12 ddt:1d -met PXE -ncl 4 -niter 0 -thres 2.0 -delta 1.0 -idx PXE
    -cla PXE04.cla -dcol 3
```

6.2.4 KRZ | kruiz - Kruizingas PCA-based types

P27 is a simple eigenvector based classification scheme (Kruizinga, 1979; Buishand and Brandsma, 1997). The scheme uses daily 500 hPa GPH on a regular grid (original one 6 x 6 points with the step of 5°in latitude and 10°in longitude). The actual 500 hPa height, htq, for the day t at grid point q is reduced first by subtracting the daily average height, ht, over the grid. This operation removes a substantial part of the annual cycle. The reduced 500 hPa heights ptq, are given by: ptq= htq - ht , q = 1,... n; n is the number of grid points, t = 1,...N, N the number of days.

The vector of reduced 500 hPa heights is approximated as:
pt = s1t*a1+ s2t*a2 + s3t*a3, t = 1,...N.
a-s are the first three principal component vectors (eigenvectors of the second-moment matrix of pt) and s-s are their amplitudes or scores.
The flow pattern of a particular day is described by three amplitudes: s1t, s2t, s3t
s1t*a1 characterizes the east-west component of the flow
s2t*a2 the north-south component and
s3t*a3 the cyclonicity (or anticyclonicity)
In the original classification, the range of every amplitude is divided into three equiprobable intervals and that gives 27 classes as each pressure pattern is on the basis of its amplitudes uniquely assigned to one of the 3 x 3 x 3 possible interval combinations.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Valid values for `<int>` are 8,9,12,18,27,30.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids are optional.

Examples

An example command could be:

```
cost733class -dat pth:hgt500.dat -met KRZ -ncl 27 -v 2
```

For producing 8 types the three amplitudes are divided into two intervals, which is indicated by the standard output on verbosity level 2.

```
 : pc = number of principal component
 : n = equiprobable intervalls per pc
 : i = number of percentile
 : s = percentile (in percent)
 : p = percentile as in timeseries
 : c = shifts class

 pc   n   i          s          p   c
  1   2   1   50.00000   -0.86810   4
  2   2   1   50.00000   -0.03095   2
  3   2   1   50.00000    0.07110   1
```

Thus observations of type 1 to 4 have scores below or equal to -0.86810 (the median) for PC 1 and those of type 5 to 8 have scores above -0.86810 in this example. Types 3,4 and 7,8 also have scores above -0.03095 for the second PC while types 1,2 and 5,6 have lower amplitudes. Types 1,3,5,7 have lower amplitudes for PC 3 than 0.07110, and types 2,4,6,8 have higher scores. Varying the number of used amplitudes and the number of percentiles the other numbers of types are created.
For `-ncl 9`:

```
 pc   n   i          s          p   c
  1   3   1   33.33333   -1.08003   3
  1   3   2   66.66667   -0.63176   3
  2   3   1   33.33333   -0.42237   1
  2   3   2   66.66667    0.40905   1
```

For `-ncl 12`:

```
 pc   n   i          s          p   c
  1   3   1   33.33333   -1.08003   4
  1   3   2   66.66667   -0.63176   4
  2   2   1   50.00000   -0.03095   2
  3   2   1   50.00000    0.07110   1
```

For `-ncl 18`:

```
 pc   n   i          s          p   c
  1   3   1   33.33333   -1.08003   6
  1   3   2   66.66667   -0.63176   6
  2   3   1   33.33333   -0.42237   2
  2   3   2   66.66667    0.40905   2
  3   2   1   50.00000    0.07110   1
```

For `-ncl 27`:

```
 pc   n   i          s          p   c
  1   3   1   33.33333   -1.08003   9
  1   3   2   66.66667   -0.63176   9
  2   3   1   33.33333   -0.42237   3
  2   3   2   66.66667    0.40905   3
  3   3   1   33.33333   −0.36353   1
  3   3   2   66.66667    0.51063   1
```

For `-ncl 30`:

```
 pc   n   i          s          p   c
  1   5   1   20.00000   −1.29727   6
  1   5   2   40.00000   −0.99266   6
  1   5   3   60.00000   −0.73334   6
  1   5   4   80.00000   −0.40355   6
  2   3   1   33.33333   −0.42237   2
  2   3   2   66.66667    0.40905   2
  3   2   1   50.00000    0.07110   1
```

6.3 Methods using the leader algorithm

Leader algorithms (Hartigan, 1975) search for leaders within the set of observations concerning the numbers of other observations being similar to them. They need a certain threshold for definition of another observation being similar. Commonly they use the Pearson correlation coefficient as similarity metric, although other metrics could be implemented.

6.3.1 LND | lund - the Lund-method

Lund (1963) published an early automated classification scheme based on pattern correlations. In a first part the so called leader patterns are determined as follows:

1. For each observation (case) the number of cases being more similar to it than a similarity threshold is counted.
2. The observation showing the maximum amount of similar cases is declared to be the leader of the first class.
3. The leader and all its similar cases are removed from the set.
4. Among the rest the second leader (leader of the second class) is determined in the same manner as the first leader.
5. After the 2nd leader and its similar cases are removed, the next leaders are determined until all classes have a leader.

In a second step all cases are put back into the pool and each case is assigned to the class of the nearest (most similar) leader (regardless of any threshold). In the `cost733class` package the default threshold is a correlation coefficient of 0.7.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.
- `-thres <real>`:
  Threshold for finding leaders and their members. Default is 0.7.
- `-dist <int>`:
  Distance metric used for similarity of observations to leaders. Default is Pearson correlation coefficient (-3).

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:hgt500.dat -met LND -ncl 9 -thres 0.7 -dist -3
```

6.3.2 KIR | kh - Kirchhofer

The Kirchhofer-method (Kirchhofer, 1974; Blair, 1998) works similar to the Lund (1963) technique. However the similarity measure accounts not only for the simple overall correlation coefficient. Instead the similarity measure is the minimum of all correlation coefficients calculated for each row (latitude) and each column (longitude) of the grid separately and the overall correlation coefficient. This should make sure that two patterns are similar in all parts of the map. As a consequence this similarity metric is usually smaller than the overall correlation alone. Accordingly the threshold for finding the key group (see 6.3.1) has to be smaller too. A typical value is 0.4 (the default). In order to know the grid geometry it is necessary to provide information about the coordinates (see 4).

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. Grid description is necessary!

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.
- `-thres <real>`:
  Threshold for finding leaders and their members. Default is 0.4.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:hgt500.dat -met KIR -ncl 9 -thres 0.4
```

6.3.3 ERP | erpicum

Even if the scheme originally was intended only for the classification of 500hPa geopotential or sea level pressure patterns, it is also possible to apply it to other sizes as well. Moreover multi-field data-input is available.
In a preprocessing step the data is standardized temporally. Then, for every single grid-point, variable and time step, the 3D geopotential direction is determined (see Fig. 6.4). Subsequently, for one variable's resulting directional fields, the (euclidean) differences between all pairs of fields, respectively time steps, are computed (compare Fig. 6.5). That is, every day (e.g.) is compared with each other. Finally, if one day is very similar to another, this couple of days will achieve an index of nearby 1. If two days are very different, their index will be set to around zero. In case of multi-field data-input, the daily mean of all the input variables is taken.

Now the classification can be done. Therefore it is counted, how often one day's

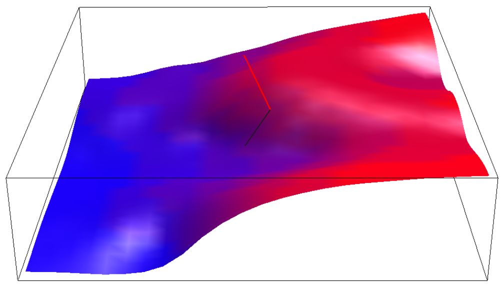


Figure 6.4: The red line indicates the 3D geopotential direction of the given grid-point according to the surrounding field. Red indicates high values in data, blue low ones.

similarity-index raises a given threshold (usually between 1 and 0). The day which reaches the highest amount of similar patterns will be the reference of the first type. All days that once are assigned to a type's reference pattern won't be considered further. The following types are determined recursively, one by one using only the remaining ergo unclassified days of all the preceding types.

```
Tab. 1:
    cl         ic        obs      count
     1   1.00000          1          1
     2   1.00000          1          1
     3   1.00000         1          1
     4   1.00000         1          1
     5   1.00000         1          1
     6   1.00000         1          1
     7   1.00000         1          1
     8   1.00000         1          1
     9   1.00000         1          1
                                16435  missing!
   ...
    cl        ic       obs      count
     1   0.63000      3809      16392
     2   0.63000      1940         18
     3   0.63000     14423         18
     4   0.63000      1628          5
     5   0.63000      5252          3
     6   0.63000         1          1
     7   0.63000         1          1
     8   0.63000         1          1
     9   0.63000         1          1
                                    0  missing!


Tab. 2:
    cl        ic       obs      count
     1   1.00000         1          1
     2   0.99000         1          1
     3   0.98000         1          1
     4   0.97000         1          1
     5   0.96000      1399          3
     6   0.95000     15704         16
     7   0.94000      9148         87
     8   0.93000      4009        280
     9   0.92000     11333        659
                                15390  missing!
   ...
    cl        ic       obs      count
     1   0.70000     12815      16093
     2   0.69000      9232        141
     3   0.68000      3521        111
     4   0.67000      7468         57
     5   0.66000      3780         24
     6   0.65000      7151          5
     7   0.64000      7769          3
     8   0.63000         1          1
     9   0.62000         1          1
                                    0  missing!

Tab. 3:
    cl        ic       obs      count
     1   1.00000         1          1
     2   0.95000     15704         16
     3   0.90000     11333       2498
     4   0.85000     14935       5069
     5   0.80000      3431       4559
     6   0.75000      7323       2602
     7   0.70000     10059       1103
     8   0.65000     12865        464
     9   0.60000      7407         88
                                   36  missing!
   ...
    cl        ic       obs      count
     1   0.91000     11333       1523
     2   0.86000      6594       5218
```

```
      3   0.81000       1157       4391
      4   0.76000       7976       3509
      5   0.71000      10059       1127
      6   0.66000      10710        456
      7   0.61000       7408        155
      8   0.56000      13201         48
      9   0.51000       1191          9
                                    0  missing!
```

For optimization, the threshold will generally be decreased until all days are classified (starting off with 1, only a few days get classified) (see Tab. 1). Furthermore the threshold is lowered with the type, so let it be 0.70 for the first one, it could be 0.69 for the second (see Tab. 2). In conclusion, several runs are done, gradually increasing the spacing of the thresholds, according to their types (compare Tab. 2 and 3). That is, for the first run there generally are the same thresholds (e.g. 0.63; Tab. 1) for all of the types. For the second run there may be 0.70, 0.69, 0.68, etc. for the first, second, third, etc. type respectively (Tab. 2). Following this for the third run there could be 0.75, 0.73, 0.71, etc., and so on.
All the resulting time series of types (one series for each run - or overall-decrement of the

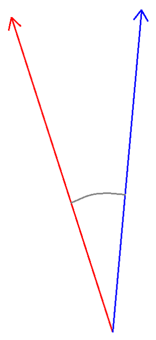


Figure 6.5: For a given grid point, the difference between two time steps is calculated from the cosine of the angle between the particular 3D geopotential directions.

threshold) are now evaluated due to their explained cluster variance in the source-data. Only the series which achieves the highest score is returned.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. Grid description is necessary!

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.
- `-nrun <real>`:
  Number of optimization runs. Default is 10000.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:hgt500.dat fmt:ascii lon:-37:56:3 lat:30:76:2 -met ERP -ncl 9
```

6.4 HCL | hclust - Hierarchical Cluster analysis

Hierarchical cluster analysis can be realized in two opposite ways. The first, divisive clustering, splits up the whole sample, according to some criterion, into two classes in the first step. On a second hierarchy level it splits up one of these classes again into two groups and so. The opposite way is the agglomerative clustering: each single observation is treated as a single cluster and on each hierarchy level of the process the two nearest clusters are merged to build a new common cluster. In `cost733class` the agglomerative method is implemented.
The routine for agglomerative hierarchical clustering offers various criteria for deciding which clusters should be merged (see below).

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

- `-crit <int>`:
  Criterion for deciding which clusters should be merged. `<int>` may be:

1	:	Ward's minimum variance method (default)
2	:	single linkage
3	:	complete linkage
4	:	average linkage
5	:	Mc Quitty's method
6	:	median (Gower's) method
7	:	centroid method

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:hgt500.dat -met HCL -crit 1 -ncl 9
```

6.5 Optimization algorithms

6.5.1 KMN | kmeans - conventional k-means with random seeds

The $k - means$ algorithm for non-hierarchical cluster analysis works rather simple. k represents the number of clusters (types or classes) which should be derived. This number has to be decided by the user. There is no routine to determine an appropriate number automatically. *means* denotes the average of each type, which is called centroid in cluster analysis and plays a fundamental role for this algorithm. It begins with a so called starting partition. In case of the `kmeans` implementation this starting partition is determined by random, i.e. each object is assigned to a cluster by random. Starting from this (undesirable) state of partitioning, the centroids are calculated as the average of all objects (e.g. daily pressure fields) within each cluster. The rest of the process is organized in iterations. In each iteration each object is checked for being in the cluster with the most similar cluster centroid. Similarity is defined to be the Euclidean distance between the centroid and the object in question, but other distance metrics may be used as well. If there is a centroid being more similar than the one of the current cluster, the object is shifted and the centroids of the old cluster and of the new cluster are updated (the average is recalculated) in order to reflect the change in the membership. This means that some of the objects which have been checked before could be now in the wrong cluster, since its centroid has changed. Therefore all objects have to be checked again in the next iteration. This process keeps going on until no shifts occur and all objects are in the cluster with the most similar centroid. In this case the optimization process has converged to an optimized partition or in other words to an optimum of the distance minimization function.
Due to the existence of local optima in the minimization function (reducing the within-cluster variance) for different random starting partitions, different solutions (optimized partitions) may occur. Therefore the KMN routine runs 10 times by default and selects the best solution in terms of explained cluster variance (ECV). For better results it is advised to set `-nrun <int>` to higher values, e.g. 100 or 1000.
The standard output of KMN shows the run-number, the number of iterations needed to converge, the explained cluster variance, the sum of within-type variances and the cluster sizes. Eventually it is reported that there are "empty clusters". This can happen during the optimization process and stops this run. However, as many runs as needed to reach the given run number are triggered.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.
- `-nrun <int>`:
  The number of runs (see above). Default is 10.
- `-dist <int>`:
  Distance metric used for similarity. Default is Euclidean distance (2).
- `-crit <int>`:
  Switch for deciding which algorithm should be used. `<int>` can be:
  1 : Optimized algorithm. Default.
  2 : Original algorithm by Hartigan and Wong (1979).

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat lon:3:20:1 lat:41:52:1 -met KMN -nrun 10 -ncl 9
```

Another example using the original algorithm by Hartigan and Wong (1979) with 1000 runs and a different number of classes:

```
cost733class -dat pth:test.dat lon:3:20:1 lat:41:52:1 -met KMN -nrun 1000 -ncl 18
    -crit 2
```

6.5.2 CAP | pcaca - k-means of time filtered PC-scores and HCL starting partition

Actually this classification scheme is not a method on its own, but a combination of two methods and certain data preprocessing steps. This is the reason why there is no argument `-met CAP` provided or allowed. Instead this classification can be obtained by first running a hierarchical cluster analysis (`-met HCL`) followed by a k-means cluster analysis (`-met KMN`), the latter using the catalog of the former as starting partition.
However, since this classification procedure was used in COST Action 733 as a method on its own, it is described in the following: It is a two-stage procedure which comprises a Principal Component Analysis to derive the dominant patterns of variability and a clustering procedure to classify time series of the principal components (Comrie, 1996; Ekstroem et al., 2002).
If to analyze the entire year is considered (not splitting the data in seasons) raw data fields can be submitted to a temporal filtering to remove variability on time scales longer than the typical duration of regional weather systems but retaining the spatial patterns; otherwise the PCA can be strongly influenced by seasonal variability in the pressure data. Such temporal variability is commonly removed following a method outlined in Hewitson and Crane (1992). Instead of a running mean, however, the `cost733class` software can be used to apply a gaussian filter. Literature shows that different length filters have been used (from 8 to 13 days for running means corresponding with a length of 31 days for gaussian filters), depending on the area of analysis. To accomplish, the average value of the map on each time step is first calculated, and then an n-days moving average is obtained. Finally, the difference between the original grid values and the filtered average time series values is then obtained, transforming the raw data at each grid point on departures from the smoothed n-days map mean, which are used in all the subsequent analyzes.
PCA is used to reduce the original data (usually a large dimension database) into a small set of new variables (principal components) that explain most of the original variability, while the contribution from small-scale variability and noise is ignored.
The retained components were rotated using a VARIMAX procedure to facilitate the spatial interpretation of the principal components and to avoid well-known problems due to the orthogonality constraint of the eigenvector model.
Actual maps mix those modes of variability, whose intensity, expressed by the weights of the component scores, varies from time step to time step. Thus, it is necessary to apply a clustering procedure to identify the most common combinations. First, the initial number of clusters and the mean conditions within each cluster (mean scores) are determined by Ward's hierarchical algorithm. Later, those initial clusters are used as seeds to obtain the final solution through a K-means agglomerative algorithm.
In terms of `cost733class` command line arguments this method can be realized by the following two commands, the first providing the hierarchical starting partition of the

preprocessed data, the second to apply k-means to that starting partition.

Options

This is not a subroutine on it own but a combination of two subsequent `cost733class` runs. Please refer to sections 6.4 and 6.5.1.

Output

This is not a subroutine on it own but a combination of two subsequent `cost733class` runs. Please refer to sections 6.4 and 6.5.1.

Examples

At first we provide a starting partition by hierarchical cluster analysis of low-pass-filtered PC-scores:

```
cost733class -dat pth:slp.dat lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1 ldt:2008:12:31
    ddt:1d nrm:1 fil:31 -pcw 0.9 -met HCL -ncl 10
```

Then we run `cost733class` with k-means using the starting partition created above:

```
cost733class -dat pth:slp.dat lon:-10:30:2.5 lat:35:60:2.5 fdt:2000:1:1 ldt:2008:12:31
    ddt:1d nrm:1 fil:31 -pcw 0.9 -clain pth:HCL10.cla dtc:3 -met KMN -ncl 10
```

6.5.3 CKM | ckmeans - k-means with dissimilar seeds

In this classification, a modified minimum distance method is used for producing the seeds for k-means clustering. According to the concept of optimum complexity which is applied to the gain in explained variance with growing number of classes the authors of this method suggest a number of around 10 clusters (Enke and Spekat, 1997). The following steps are performed:

- The initialization takes place by randomly selecting one weather situation (object).
- In a stepwise procedure the starting partition, consisting of the ten most dissimilar weather situations (days) is gradually identified, see Fig. 6.6. Similarity and dissimilarity are being defined by a variant of the euclidean distance measure RMSD.

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n} R_i^2} \tag{6.1}$$

With R_i =: Difference between class centroid value and current day value; n =: Number of grid points. If more than one atmospheric field is used, a normalization is necessary to maintain comparability and to retain the additive nature of the distances (Enke et al., 2005, for more details see).

- All remaining days of the archive are assigned to their most similar class. With each day entering a class, a position shift takes place which in turn makes it necessary to re-compute the centroid positions. As a consequence, the multi-dimensional distance between class centroids continually decreases (see Fig. 6.7); the variability within the individual classes, on the other hand, increases.
- After the assignment of all days has been performed and an intermediate stage has been reached, an iterative process is launched. With each pass, members of the clusters are exchanged in order to improve the separation of the classes and their compactness. Gradually, a final centroid configuration emerges. In order to retain representative results, classes are kept in the process only if they do not fall short of a certain number, e.g., 5% of all days. Otherwise the class will be removed and its contents distributed among the remaining classes. The iterative process will be stopped if two successive iterations leaves the contents of the classes unchanged.

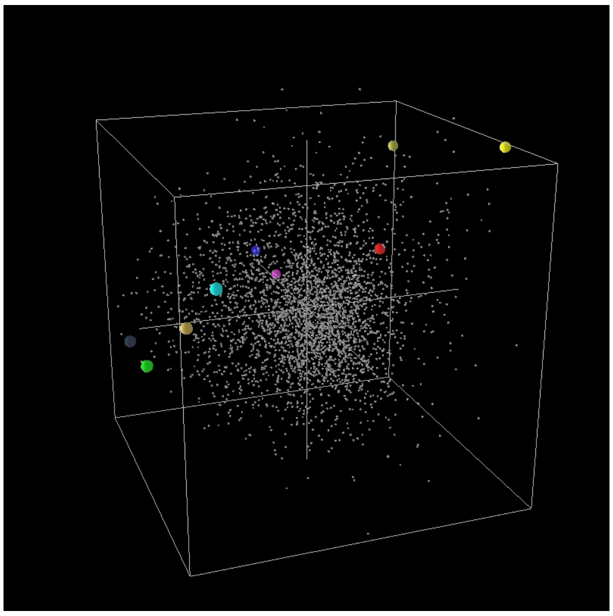


Figure 6.6: Starting seed points for CKM classification method using the example data set slp.dat and 9 classes. The projection of each daily pattern into the three-dimensional PCA phase space is shown by small spheres. Large spheres represent the projected centroids. —> link to command line + revert black background

The centroids converge towards a final configuration which has no similarity with the starting partition. The process is sketched in Fig. 6.7 (from Enke and Spekat, 1997).

Beginning with the initial conditions (denoted 'a'), the intermediate stage (denoted 'b') is reached which is characterized on the one hand by a higher proximity of the classes but on the other hand by large 'class volumes' (i.e., high within-type variability) since the assignment of days is not optimized. After the iterative exchange process, the final stage (denoted 'c') is free of overlap (i.e., it is well separated, exhibiting high between-type variability) and consists of comparably compact classes (i.e., it exhibits low within-type variability), see Fig. 6.8.

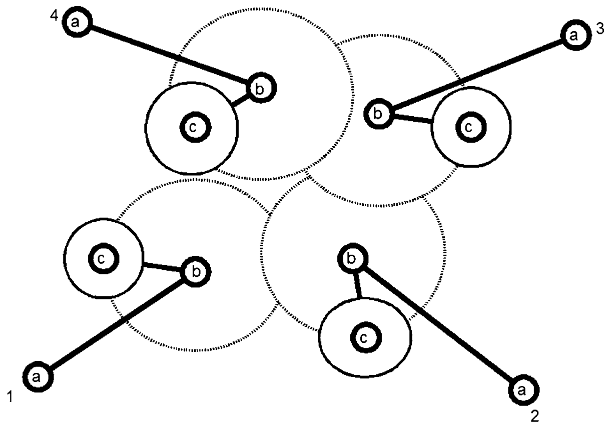


Figure 6.7: Sketch of the iterative process for ckmeans that leads from the starting partition to the final stage. For details see main text.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

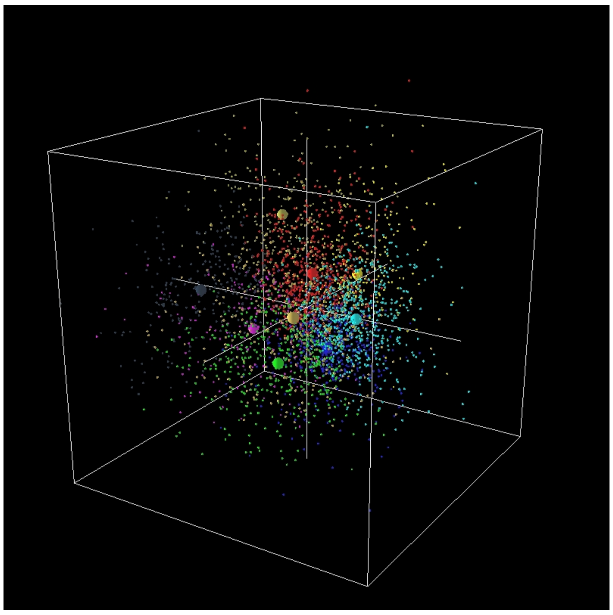


Figure 6.8: Final partition for CKM classification method using the example data set slp.dat and 9 classes. The projection of each daily pattern into the three-dimensional PCA phase space is shown by small spheres. Large spheres represent the projected centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat lon:3:20:1 lat:41:52:1 -met CKM -ncl 9
```

6.5.4 DKM | dkmeans - a variant of ckmeans

DKM differs from CKM in three points:

1. For finding dissimilar seed patterns for the 2nd and further clusters not only the sum of the distances to the preceding seeds is used but the minimum of these distances is maximized.

2. There is no 5% minimum frequency limit for keeping all clusters. In most cases if the population temporarily decreases below 5% of the total sample size, it is filled up again later during the iterations.

3. Furthermore it is possible to select a number of runs. As the initiation of the starting partition partially follows random assignment, results differ from run to run. Therefore the solution with highest explained cluster variance will be chosen.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

- `-nrun <int>`:
  The number of runs (see above). Default is 10.

- `-crit <int>`:
  Switch for choosing the method for initial pattern selection . `<int>` can be:

  1 : Choose the observation which minimizes the distance to the total centroid.

  2 : Choose the observation which maximizes the distance to the total centroid.

  3 : Randomly choose a observation. Default.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat lon:3:20:1 lat:41:52:1 -met DKM -ncl 9 -nrun 10 -crit 3
```

6.5.5 PXK | pcaxtrkm - k-means using PXE starting partitions

This method follows the application of PXE but differs in so far that the final CTs are established using the iterative clustering method (K-means). In this way, for the optimization of the final clusters, the K-means method starts using the centroids obtained with the" extreme scores" criteria as starting partitions.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Always has to be a pair value / even number (2, 4, 8, 10,...) Default is 8.
- `-niter <int>`:
  Maximum number of iterations. Leave it unchanged for PXK in order to reach convergence (default: `-niter 9999999`), i.e. this option should be omitted for PXK.
- `-crit <int>`:
  Normalization method of the input data before the PCA is performed: . `<int>` can be:

  0 : Only normalize patterns for PCA (original).

  1 : Normalize patterns and normalize grid point values afterwards.

  2 : Normalize patterns and center grid point values afterwards (default).

  The thresholds for definition of extreme score cases can be changed by:
- `-thres <real>`:
  Threshold defining key group (default is 2.0).
- `-delta <real>`:
  Score limit for other PCs to define uniquely leading PC (default is 1.0).

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.
- `-idx <basename>`:
  Output base name for rotated scores (*.sco).

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set as well as a file containing the rotated scores are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met PXK -ncl 8 -niter 9999999 -crit 2 -thres 2.0
   -delta 1.0
```

6.5.6 SAN | sandra - simulated annealing and diversified randomization

SANDRA means Simulated ANealing and Diversified RAndomization. Essentially it is a non-hierarchical cluster analysis method like k-means. However, it usually finds better solutions than conventional k-means.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

- `-nrun <int>`:
  The number of diversified runs for searching the best result. Note that if compilation has been started using the `-openmp` option (i.e. `.\configure CC=icc FC=ifort FCFLAGS="-openmp"`) the runs are executed in parallel. How many parallel threads are used might depend on a environment variable (e.g. `NTHREAD=4`) of your operating system. For details check the documentation of your compiler. Default is 1000. Note that this may cause very long runtimes and is no bug.

- `-niter <int>`:
  Maximum number of iterations. Leave it unchanged in order to reach convergence (default: infinity). Sometimes SAN ends up in an endless loop. Then set set it to a finite value, i. e. `-niter 10000`. However, this is very unlikely.

- `-cool <real>`:
  This parameter controls the speed of cooling down the temperature parameter (i.e. how fast the probability for so called wrong shifts of objects is reduced). The higher `<real>`, the slower the decrease. Note that it must be less than one (e.g. 0.999), else the routine will never stop. Default is 0.99.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met SAN -ncl 9 -nrun 1000 -cool 0.99
```

6.5.7 SOM | som - self organizing (feature) maps (neural network according to Kohonen)

Self organizing maps describe a way to arrange types defined by their features (grid point values) in a structure where similar types are adjacent to each other (a map). This structure commonly is a two dimensional matrix describing a network of neurons (i.e. types) which are connected to their neighbors in the four directions. However this structure can also be a one dimensional arrangement where the neurons are connected to their left and right neighbors only, which is the case for the method implemented in `cost733class`.
The number of neurons is given by the number of types provided by the `-ncl <int>` option. Each neuron has as many features as there are grid points or variables in the input data set (columns in case of ASCII-input). The values of each neuron are initialized by random numbers. Then for each iteration the data of the objects to classify are presented one after the other to these neurons and in each case a winner neuron is determined by the minimum Euclidean distance between the presented object and the neurons. The winner neuron and its neighbors are then modified to become more similar to the presented object by calculating the weighted mean between the old neuron values and the object, where the weight is a tuning parameter called `alpha`. How many neighbors of the winner neuron are affected of this modification of the neuron values depends on the `radius`-parameter. At the beginning of the classification process it is large, thus all neurons are affected, but it is slowly reduced so that at the end only the winner neuron will be modified. Also the weight `alpha` is slowly modified during the iterations (training epochs) thus at the end the object (pattern) which is assigned to the winner neuron has a large impact on the modification of the neuron values. If the neuron values do not change any more (convergence) the process is finished and each object is assigned to the most similar neuron (type).

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of neurons/classes. Default is 9.

- `-nrun <int>`:
  The number of diversified runs for searching the best result. Note that if compilation has been started using the `-openmp` option (i.e. `.\configure CC=icc FC=ifort FCFLAGS="-openmp"`) the runs are executed

in parallel. How many parallel threads are used might depend on a environment variable (e.g. `NTHREAD=4`) of your operating system. For details check the documentation of your compiler. Default is 10.

- `-cool <real>`:
  Factor for reduction of learning rate. Default is 0.99.
- `-niter <int>`
  Number of iterations (training epochs). Default is 10000.

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met SOM -ncl 9 -nrun 10 -niter 10000 -cool 0.99
```

6.5.8 KMD | kmedoids - partitioning around medoids

K-medoids is similar to k-means as it iteratively minimizes a cost function (within cluster variability). However it differs, as it does not use the cluster mean (i.e. centroids) for measurement of within-type variability but uses given data points as centers of the clusters. The algorithm consists of the following steps:

1. select k objects by random to be the initial cluster centers, i.e. medoids.
2. assign all other objects to them according to the minimum distance
3. calculate the cost function, which is the sum of the distances between each object and its nearest medoid

4. begin an iteration
5. for each cluster check all objects whether they would be a better medoid, if yes change the medoid and update the assignments and cost function
6. if no enhancement is possible by changing the medoid stop the iterations.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.
- `-nrun <int>`:
  Number of runs with initial random medoids. The solution with minimum cost function will be chosen. Default for is 10.
- `-dist <int>`:
  Distance metric used for similarity. Default is Euclidean distance (2).

Options for data output:

- `-cla <filename>`:
  Output file name for the classification catalog.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids and centroids for each input data set are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met KMD -ncl 9 -nrun 10 -dist 2
```

6.6 Random classifications

6.6.1 RAN | random

This method does not use any input data but randomly selects any arbitrary number as type number for each object. The respective number is retrieved from a random number generator between 1 and the maximum number of classes given by the `-ncl <integer>` option. More than one catalog can be produced by using the `-nrun <int>` option. By default the best catalog out of these according to the explained cluster variance (see 8) is selected and written to the `-cla` file. If the option `-mcla <filename>` is given all catalogs will be written to that file (each catalog one additional column).

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

- `-nrun <int>`:
  Number of runs. By default the best solution will be chosen. Default for is 1000.

Options for data output:

- `-cla <filename>`:
  Output file name for the catalog with the best classification according to explained cluster variance.

- `-mcla <filename>`:
  Output file name for all classification catalogs.

- `-dcol <int>`:
  Number of date columns in the classification catalog.

- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids, centroids for each input data set as well as a file containing a catalog for each run are optional.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met RAN -ncl 9 -nrun 1000
```

An example producing an output file with 100 catalogs.

```
cost733class -dat pth:test.dat -met RAN -ncl 9 -nrun 100 -mcla RAN9ncl_all.cla
```

6.6.2 RAC | randomcent

In contrast to the method above input data have to be provided for method `RAC`. Any arbitrary object of these input data is used as seed or key object (centroid) for each class. All the rest of the input data are then assigned to these centroids by determination of the minimum Euclidean distance. If the option `-idx <filename>` provides a name of an non-existing file, the numbers of the objects used as medoids, which are determined by random, are stored to this file. It has as many rows as there are different classification (`-nrun <integer>`) and as many columns as there are types (`-ncl <integer>`). If in contrast the file is old and contains at least enough numbers to provide the seeds, these numbers will be used for seeding the types. However they all have to less or equal than the number of objects to be classified. This feature allows for comparable random medoid classifications just differing e.g. by data preprocessing.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.

Optional data input:

- `-idx <filename>`:
  Input data for seeds.

The following options define the classification:

- `-ncl <int>`:
  Determine the number of classes. Default is 9.

- `-nrun <int>`:
  Number of runs. By default the best solution will be chosen. Default for is 1000.

Options for data output:

- `-cla <filename>`:
  Output file name for the catalog with the best classification according to explained cluster variance.
- `-mcla <filename>`:
  Output file name for all classification catalogs.
- `-dcol <int>`:
  Number of date columns in the classification catalog.
- `-idx <basename>`:
  Output base name for all seeds (*_med.list). Only if `<filename>` doesn't exist. Default is OUTPUT.
- `-cnt <filename>`:
  Output file name for the class centroids.

Output

This method returns one file containing the classification catalog. Overall class centroids, centroids for each input data set as well as a file containing a catalog for each run are optional. Moreover it is possible to obtain a file containing the seeds of initial random assignment.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met RAC -ncl 9 -nrun 1000 -idx OUTPUT_med.list
```

An example reading a seed file with 10 different seeds for 9 classes and output of each classification catalog:

```
cost733class -dat pth:test.dat -met RAC -idx seed_ncl9_nrun10_med.list -ncl 9 -nrun 10
    -mcla RAC9ncl_nrun10_all.cla
```

The seed file `seed_ncl9_nrun10_med.list` could look like this:

```
    1844     553     881    1612    1800    1886    2263    2663     420
     264    1376    2704     568     459    2542      63    2006    2022
    1948    2553      73    1077    2602    1517    2206    1319     479
    1717     145    2614    2030    2518    2510    1045     464    1460
     652    1082    2295    1467    2051      90    2050    1678    2094
    1691     718    1901    1927     165    2368    1634    1211      68
    1787     599     768    1791    2611     702     927     532    1616
     334    1058    1284    2626      52    2424    1122    1995    2576
    1987    1008    2198     487    2595    1380    1687    1736     614
    1753     390     996    1874     956    2519     770    1404     616
```

7 Assignments to existing classifications

7.1 ASC | assign

Using this option any data set of the same number of variables as a given centroid file can be assigned to the respective classes. For each record (object) of the input data set the dissimilarity/distance between the object and the centroids is then calculated and the class number is chosen to be the one with the minimum distance.

Options

Strictly necessary options:

- `-dat <specification>`:
  Input data. No grid description is necessary.
- `-cntin <filename>`:
  Input centroid file. Only ASCII is allowed.

The following options define the classification:

- `-dist <int>`:
  The distance metric used for measure of similarity between the given centroids and each observation. Default is Euclidean distance (2).

Options for data output:

- `-cla <filename>`:
  Output file name for the catalog with the best classification according to explained cluster variance.
- `-dcol <int>`:
  Number of date columns in the classification catalog.

Output

This method returns one file containing the classification catalog.

Examples

An example with default values:

```
cost733class -dat pth:test.dat -met ASC -dist 2 -cntin KMN09ncl.txt
```

7.2 CNT | centroid

This operation allows to create centroids of a given classification catalog using the given data. The flag `-clain <spec>` is used to provide an existing catalog file comprising the integer class numbers for each object in each line respectively. In order to calculate the centroids as means of all objects contained within the respective class the data used for building the centroids has to be provided by one or more `-dat <specification>` arguments. The number of records or lines (e.g. time steps) in the catalog file has to fit the number of records or lines of the data set. By providing the `-cnt <filename>` option the centroids are written to the respective file. In a subsequent run of `cost733class` these centroid data can be used for assigning additional data to the centroids (see method `ASC` above) in order to produce a new catalog file.

8 Evaluation of classifications

Several statistical metrics characterizing classifications in terms of separability among classes and variability within classes (see e.g. Beck and Philipp (2010)) can be applied to existing catalogs (provided by -clain <spec>) and for a given input data set (-dat <specification>).

8.1 EVPF | evpf - explained variation and pseudo F value

This routine calculates the explained variation (EV) on the basis of the ratio of the sum of Squares within classes/circulation types (WSS) and the total sum of squares (TSS).

$$EV = 1 - \frac{WSS}{TSS} \tag{8.1}$$

Additionally the so called Pseudo-F statistic (PF) according to Calinski and Harabasz (1974) is calculated as the ratio of the sum of squares between classes (BSS) and the sum of squares within classes (WSS) thereby taking into account the number of cases (n) and classes (k).

$$PF = \frac{BSS/(k-1)}{WSS/(n-k)} \tag{8.2}$$

Command line parameters relevant for EVPF:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-crit <integer>`: file output for [1] each individual variable and all variables (default) or [2] just the latter.
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_ev.list, <-idx>_pf.list`: EV- / PF-indices estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

- `<-idx>_ev.fld, <-idx>_pf.fld`: EV- / PF-indices estimated for each indvidual variable from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

8.2 ECV | exvar

This routine calculates the explained cluster variance (ECV is the same than EV, see above) values for existing catalogs (provided by `-clain <spec>`) for the given input data set (`-dat <specification>`). In contrast to EVPF it does not discern seasons and does not create output files. It just calculates the ECV, prints it to the screen and exits.

$$ECV = 1 - \frac{WSS}{TSS} \tag{8.3}$$

, where

$$WSS = \sum_{j=1}^{k} \sum_{i \in C_j} {D_{(X_i,\bar{X}_j)}}^2 \tag{8.4}$$

, C_j is the Class j of the k classes, and the squared Euclidean distance between an element and its centroid

$$D_{(X_i,\bar{X}_j)} = \sum_{l=1}^{m} (X_{il} - \bar{X}_{jl})^2 \tag{8.5}$$

TSS is the total sum of squared differences between all elements and the overall centroid (mean):

$$TSS = \sum_{i=1}^{n} \sum_{l=1}^{m} (X_{il} - \bar{X}_l)^2 \tag{8.6}$$

.

The input catalog file can have more than one catalog columns (e.g. as produced by `-met RAC`). The ECV is calculated for all catalogs within this file one after the other.

8.3 WSDCIM | wsdcim - Within-type standard deviation and confidence interval of the mean

The within-type standard deviation (WSD) is calculated as the pooled standard deviation (SDI) in order to take into account differing sizes (n) of classes (k).

$$WSD = \sqrt{\frac{\sum_{k=1}^{K}(n_k - 1) \cdot SDI_k^2}{\sum_{k=1}^{K}(n_k - 1)}} \tag{8.7}$$

The range of uncertainty of the variable's mean values associated to each classification is reflected by the confidence interval of the mean (CIM) calculated as the weighted mean (utilizing class sizes as weights) of the type specific confidence intervals of the mean for varying confidence levels (1 - alpha).

$$CIM = \frac{k \sum_{k=1}^{K} z_{1-\alpha/2} \cdot \frac{SDI_k}{\sqrt{n_k}} \cdot n_k}{k \sum_{k=1}^{K} n_k} \tag{8.8}$$

Command line parameters relevant for WSDCIM:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-step <integer>`: missing value indicator for catalog data.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-thres <real>`: confidence level for estimating the confidence interval CIM. Default value is 0.05.
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_wsd.list, <-idx>_cim.list`: WSD- / CIM-indices estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).
- `<-idx>_wsd.fld, <-idx>_cim.fld`: WSD- / CIM-indices estimated for each indvidual variable from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

8.4 DRAT | drat - Ratio of distances within and between circulation types

The ratio of the mean pattern correlation (Pearson's r) between days (daily gridded fields) assigned to the same circulation type and the mean pattern correlation between days assigned to different circulation types has been proposed by Huth (1996) as a measure for the separability among circulation types. Here we generalize this evaluation metric by using either the Pearson correlation or the euclidean distance as measure of similarity between cases (days) to calculate the mean "distance" between days (daily gridded fields) assigned to the same circulation type (DI) and the mean "distance" between days assigned to different circulation types (DO). Note that the interpretation of values of DRAT smaller or greater 1 resp. is different when using ED or PC as "distance" metric.

$$DRAT = \frac{DI}{DO} \tag{8.9}$$

Command line parameters relevant for DRAT:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-dist <integer>`: distance metric to use for calculating DRAT [1 = Euclidean distance (default), 2 = Pearson correlation]
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_drat.list`: DRAT-indices estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

8.5 FSIL | fsil - Fast Silhouette Index

The Silhouette index (SIL) according to Rousseeuw (1987) is often used for evaluating the quality of cluster separation. As the calculation of the Silhouette index after Rousseeuw (1987) is rather CPU-intensive when applied to large data sets a modified approach is used for estimating a "faster Silhouette Index" (FSIL). The difference between FSIL and SIL is that for FSIL for any case (day, i) the distances to its own class (fa_i) and its nearest neighboring class (fb_i) are estimated as the euclidean distances to the respective class centroids and not as for SIL as the average distance between the case and all cases in its own class and its closest class respectively.

$$FSIL = \frac{1}{n} \sum_{i=1}^{n} \frac{fb_i - fa_i}{max(fa_i, fb_i)} \tag{8.10}$$

Command line parameters relevant for FSIL:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-step <integer>`: missing value indicator for catalog data.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_fsil.list`: FSIL-indices estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

8.6 SIL | sil - Silhouette Index

This routine provides the original Silhouette index (SIL) according to Rousseeuw (1987). In contrast to FSIL for any case (day, i) the distances to its own class (a_i) and its nearest neighboring class (b_i) are calculated as the average distance (in terms of the euclidean distance) between the case and all cases in its own class and its closest class respectively.

$$SIL = \frac{1}{n}\sum_{i=1}^{n}\frac{b_i - a_i}{max(a_i, b_i)} \tag{8.11}$$

Command line parameters relevant for SIL:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-step <integer>`: missing value indicator for catalog data.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_sil.list`: SIL-indices estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

8.7 BRIER | brier - Brier Score

Brier Skill Score:

$$BSS = \frac{\frac{1}{N}\sum_{i=1}^{I} N_i(y_i - \bar{o})^2}{\bar{o}(1-\bar{o})} \tag{8.12}$$

BSS	=	Brier Skill Score $\in [0,1]$
N	=	number of cases
I	=	number of classes
y_i	=	conditional event frequency $(\frac{events\,for\,class\,i}{elements\,of\,class})$
$\bar{o}$	=	unconditional event frequency $(events/N)$

Command line parameters relevant for BRIER:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-dat <specification>`: input data set to which evaluation metrics are applied
- `-crit <int>`: if [1] quantile (-thres) is applied to absolute values (default), if [2] quantile (-thres) is applied to euclidean distances between patterns.
- `-thres <real>`: quantile [0,1] (default=0.9) to define events. An event is defined when the euclidean distance to the periods/seasonal/monthly mean-pattern is greater than the given quantile. If <thres> is signed negative (e.g. -0.8), than events are defined if smaller than the given quantile.
- `-alpha <real>`: [<0] (default) use all values (-crit 1) or patterns (-crit 2); [>=0] a value or pattern is processed only if itself or mean(pattern) > alpha.
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_brier.list`: Brier-scores estimated over all variables from the input data set for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

9 Comparison of classifications

9.1 CPART | cpart - Catalog comparison

The following indices determining the similarity between two partitions (classifications) of one set of observations are calculated by this routine: Rand Index [RI], Adjusted Rand Index [ARI], Jaccard Index [JI], Mutual Information [MI], Normalized Mutual Information [NMI]. See L.Hubert and Arabie (1985); Kuncheva and Hadjitodorov (2004); Southwood (1978); Strehl and Gosh (2002) Kalkstein et al. (1987) Milligan and Cooper (1985) Rand (1971) for more information on the different indices.

Command line parameters relevant for CPART:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-idx <character string>`: base string for naming of output file(s)

Output:

- `<-idx>_RI.fld, *_ARI.fld, *_JI.fld, *_MI.fld, *_NMI.fld`: Diversity indices estimated for all pairwise combinations of catalogs from -clain for months, seasons and the whole year (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, win, spr, sum, aut, yea).

10 Miscellaneous functions

There are some other functions in `cost733class` that do not fit in any of the previous chapters.

10.1 AGG | agg - Aggregation

This subroutine provides basic function for aggregation. At this point yearly sums (per variable) are supported, however it is intended to compute seasonal values with the `-mon` option. For this function season is identical with meteorological season (December, January and February for winter, and so forth).

Command line parameters relevant for AGG:

- `-agg <filename>`: output file name

10.2 COR | cor - Correlation

The Method `COR` calculates the so called reduction of (error) variance (RV), the Spearman and the Pearson correlation coefficients for all input data sets. Computation is carried out for all variable combinations and results in three matrices, which are written to one file named `correlations.txt`. The equation for RV is

$$RV = 1 - \left(\frac{RMSE_{v1}}{RMSE_{v2}}\right)^2 \tag{10.1}$$

where $RMSE_{v1}$ and $RMSE_{v2}$ are the root mean square errors of the two variables to be compared.

10.3 SUB | substitute - Substitute

This method replaces class numbers with their corresponding centroid values and results in a file with as many rows as in the given catalog file. Furthermore each column represents a time series of one variable. Thus the number of columns equals the number of rows in the necessary centroid file.

Command line parameters relevant for SUB:

- `-clain <spec>`: catalog input, see 4.2.2.
- `-cntin <filename>`: centroid input.
- `-sub <filename>`: file name for output of time series.

11 Visualization

If configured and compiled with the `--enable_opengl` option, `cost733class` accepts the option `-opengl` and starts a window for viewing the data and watching their classification. Data input and preprocessing has to be configured via command line interface though. However the choice of the classification method, the number of classes and some more options are available within a menu opened by the right mouse button. Moving the mouse with the left mouse button keeping pressed down allows to rotate the data cube within several directions, while the mouse wheel allows to zoom in and out. Holding the shift-key and moving the mouse while the left mouse button is pressed, shifts the data cube. Holding the ctrl-button pressed and selecting a data sphere with the left mouse button draws the map of this object at the lower left corner of the window.

`d`	`switch dimension maps at axis on and off`
`^`	`switch the data spheres off`
`0`	`switch the data spheres on`
`1`	`only show the data spheres of the first class`
`2`	`only show the data spheres of the second class`
`...`	`...`
`9`	`only show the data spheres of the nineth class`
`t`	`switch auto rotation off`
`r`	`set auto rotation angle (cycling through speeds from 0 to 0.5 degree per frame)`
`c`	`switch centroid spheres on and off`

Table 11.1: Keyboard shortcuts for controlling the visualization window.

12 Development

12.1 Implementing a new subroutine

Before you change anything you should make a copy of the complete directory in order to give it another version number and to be able to go back to the start if anything is messed up, e.g:

```
cp -r cost733class-0.19-03 cost733class-0.19-04
```

where `0.19-03` is the latest version you have downloaded.

If you write another subroutine, store it into the src directory, add the calling command to main.f90 and add its source code file name to the list `cost733class_SOURCES` in the file `src/Makefile.am`. Then run

```
autoreconf --install
.\configure && make && src/cost733class
```

Note that `autoreconf` needs the packages `autotools` and `libtool` being installed.

The single steps in detail:

In order to implement a new subroutine for a method variant or a complete new method the following steps are necessary:

- create a file containing the subroutine in the src directory
  This file begins with `subroutine xyz(arg1,arg2)` and ends with `end subroutine xyz`. In the brackets you can provide a list of arguments needed to calculate the classification. The file has to have the extension `.f90`. More on variables see below.

- add a calling function to `main.f90`
  This calling function should, for the example above, look like this:

```
        ! xyz-method
        if(trim(methodname)=="xyz")then
                if(verbose>0)then
        write(*,*)
        write(*,"(a)")" calling xyz ..."
        endif
                call xyz(arg1,arg2)
                call sort_cla(ncl)
        endif
```

  The `if(...` instruction says that the routine is called if the command line argument for `-met` was xyz. The `write(...` instructions gives information about that to the screen if the verbose level is `>0`. Then the subroutine is called, providing the arguments and after it has finished the resulting catalogs are sorted by size of the classes. You can omit the last step by writing a `!` before `call sort_cla(....`

- add an entry in the help function
  Edit the source file *arguments.f*90. Scroll down to the `function help()`, here to the `Methods` section. Copy the line from another method in the section for explaining `-met` and change it to briefly describe your method.

```
        write(*,"(a)")" : xyz : classifying by x according to y concerning z"
```

- organizing compilation
  In order to have the new subroutine compiled (otherwise the calling function in main.f90 produces `unknown function xyz`) you have to add it to the list of source code files in `src/Makefile.am` (line 5):

```
        bin_PROGRAMS = cost733class
        cost733class_SOURCES = globvar.f90 arguments.f90 avelink.f90 dkmeans.f90 \
        iguapcas.f90 pcaxtr.f90 random.f90 som.f90 assigncor.f90 kirchhofer.f90 \
        lund.f90 prototype.f90 sandra.f90 tmodpca.f90 assign.f90 datainput.f90 \
        hcluster.f90 kmeans.f90 main.f90 randomcent.f90 sandrat.f90 xyz.f90
        cost733class_LDADD = $(top_srcdir)/netcdf-4.0/libsrc/.libs/libnetcdf.a
        AM_FCFLAGS = -I$(top_srcdir)/netcdf-4.0/f90
```

  If you want to continue in a new line just add a backslash at the very end of the line before.

- rebuild the configure script
  in the main directory type:

```
        make distclean
        autoreconf --install
```

- test the new subroutine

```
        .\configure
        make
```

If it works you can pack it (see below) for others. If not you have to correct the subroutine source code and try again, however, now you can omit the cleaning and reconfigure steps from above (your subroutine is already implemented), but say only:

```
make
```

again, to retry the compilation process.

12.2 Packing the directory for shipping

In order to allow others to download the new package, you have to clean it and create a tar archive file. The cleaning is necessary since others might use another operating system and the binaries produced by your system won't work. Cleaning is done by:

```
make clean
```

Then you can pack it using the shell script `tools/pack.sh`. It automatically finds the directory name (which should include the new version number and runs `tar` and `bzip2` to compress it:

```
tools/pack.sh
```

Now you have a new package (e.g. `cost733class-99.99-00.tar.bz2`) in the top directory that might be uploaded to the homepage and made available for others.

12.3 Use of variables in the subroutine

In order to write a new method subroutine you need some input for your routine. The basic input is already provided by the program and you can just use it. There are two different variable types according to the way you can access them:

- global variables being declared in any case
  Global variables can be used if you say `use globvar` at the very beginning of your subroutine. They are declared by a module in `globvar.f90` and values are provided by other subroutines (in particular datainput.f90) depending on the arguments given on the command line by the user. Thus most of them are ready for use:
  - `integer :: nobs`
    the number of observation, i.e. objects or entities to classify or lines in the ASCII input data. nobs is the total number of days for a daily data set.
  - `integer :: nvar`
    the number of variables, i.e. attributes or parameters describing each object. This commonly corresponds to the number of columns in an ASCII input file or grid points if patterns should be classified.
  - `real(kind=8) :: dat(1:nvar,1:nobs)`
    This is a two-dimensional array of the input data in double precision.
  - `integer :: npar`
    This is the number of different data sets (parameters) contained in the `dat` array. It is usually 1, however if the user has given more than one -dat argument it is higher.

  - `integer(kind=1) :: cla(1:nobs)`
    This is a one-dimensional array of one-byte integer numbers. It is allocated but not filled with values, since it is the main result from the method subroutine. Thus you have to store your classification result here. For each of the nobs objects you should store a type number in cla beginning with `1`. The maximum type number allowed is `256`.

- global variables that have to be checked
  - `integer :: tyear(nobs),tmonth(nobs),tday(nobs),thour(nobs)`
    These variables eventually hold the date for each of the nobs objects. You have to check whether they are allocated before you can use it.
  - `real(kind=8) :: minlon(npar),maxlon(npar),diflon(npar)`
    `real(kind=8) :: minlat(npar),maxlat(npar),diflat(npar)`
    These are the grid dimensions for each data set. Not necessarily allocated!
  - `real(kind=8) :: parmean(npar), parsdev(npar)`
    If more than one data set is provided in `dat` each of them is normalized. The corresponding means and standard deviations are stored here.

  You don't have to declare these in your subroutine, you can just use it. Except for the date, coordinate and normalization variables (the latter 4 items) they are all already allocated and filled with values. The others have to be checked whether they are allocated, because the user might have omitted for example to give information about longitudes and latitudes.

- local variables provided by main
  The following variables are not global variables. This means if you need it in your subroutine, you have to provide it as parameters in the subroutine calling function and subroutine
  - `integer :: ncl`
    This is the number of types provided by the option `-ncl`

12.4 Using the input data from datainput.f90

The subroutine datainput() reads in the data from files and performs preprocessing of the data depending on the command line arguments. The following steps are carried out:

1. parsing of the specification strings
2. set up variables / attribute dimension: for each data set the number of variables is calculated to achieve the total number of columns for the data matrix.

3. set up time steps / time dimension: for each data set the number of observations is determined and the maximum is used for allocating RAWDAT.
4. the data sets are read into the RAWDAT array
5. if available the date information for the time steps for each data set is gathered
6. if desired and possible sub grid sections are selected
7. if desired normalization/centering of each observation (map) is done
8. if desired a time filter is applied to each variable time series
9. if desired map sequences are constructed enlarging the number of variables.
10. if desired and possible date selections are done. The (selected) data are stored in the DAT array now.
11. if desired centralization/normalization of each variable of a data set is done (calculation of anomalies)
12. if desired a principal component analysis is done for each data set separately. The data are replaced by the scores. The number of variables are changed.
13. if desired the data sets are normalized as a whole (over columns and rows) and weighted by the given factors for each data set.
14. if desired the data as a whole are compressed by PCA.

12.5 The gnu autotools files

The package is maintained by using gnu autotools. In order to rebuild the configure script run:

```
aclocal
autoconf
automake
```

Only three files are needed for autotools:

- `configure.ac`:

```
AC_PREREQ([2.69])
AC_INIT([cost733class], [1.2], [a.philipp@geo.uni-augsburg.de])
AM_INIT_AUTOMAKE([foreign])

AC_PROG_RANLIB([AC_CHECK_PROG(RANLIB, ranlib, ranlib, :)])
AC_PROG_CC([gcc icc])
AC_LANG_C
```

```
AC_PROG_FC([gfortran ifort])
AC_PROG_F77([gfortran ifort])

AC_ARG_ENABLE([grib],
   [AS_HELP_STRING([--disable-grib  Do not use GRIB/GRIB2 functions])],
   [grib=false],
   [grib=true])

AM_CONDITIONAL([GRIB], test x$grib = xtrue)

AC_ARG_ENABLE([opengl],
   [AS_HELP_STRING([--disable-opengl  Do not use opengl visualisation])],
   [opengl=false],
   [opengl=true])

AM_CONDITIONAL([OPENGL], test x$opengl = xtrue)

if test x$opengl = xtrue; then
  AC_MSG_NOTICE([
    checking for opengl!
  ])
  #AC_CONFIG_SUBDIRS([f03gl])
fi

if test x$grib = xtrue; then
  AC_MSG_NOTICE([
    checking for grib!
  ])
  AC_CONFIG_SUBDIRS([grib_api-1.9.18])
fi

AC_CONFIG_FILES([Makefile f03gl/Makefile src/Makefile ])

dnl makefile ...
AC_OUTPUT
```

- `Makefile.am`:

```
SUBDIRS = netcdf-4.0.1

if GRIB
GRDIR = grib_api-1.9.18
SUBDIRS += $(GRDIR)
endif

if OPENGL
GLDIR = f03gl
SUBDIRS += $(GLDIR)
endif

SUBDIRS += src
```

- `src/Makefile.am`:

```
bin_PROGRAMS = cost733class

cost733class_SOURCES = globvar.f90
cost733class_SOURCES += netcdfcheck.f90
cost733class_SOURCES += netcdfinput.f90
cost733class_SOURCES += writenetcdf.f90
cost733class_LDADD = $(top_srcdir)/netcdf-4.0.1/libsrc/.libs/libnetcdf.a
```

```
AM_FCFLAGS = -I$(top_srcdir)/netcdf-4.0.1/f90

if GRIB
cost733class_SOURCES += gribcheck.f90
cost733class_SOURCES += gribinput.f90
cost733class_LDADD +=
    $(top_srcdir)/grib_api-1.9.18/fortran/.libs/libgrib_api_f90.a
cost733class_LDADD += $(top_srcdir)/grib_api-1.9.18/src/.libs/libgrib_api.a
cost733class_LDADD += -lm -ljasper
AM_FCFLAGS += -I$(top_srcdir)/grib_api-1.9.18/fortran
else
cost733class_SOURCES += gribcheck-dummy.f90
cost733class_SOURCES += gribinput-dummy.f90
endif

if OPENGL
AM_FCFLAGS += -fno-underscoring -I$(top_srcdir)/f03gl -L$(top_srcdir)/f03gl
cost733class_LDADD += -lX11 -ljpeg -lglut -lGLU -lGL
cost733class_LDADD += $(top_srcdir)/f03gl/libf03gl.a
cost733class_SOURCES += GLUT_fonts.c opengl.f90
else
cost733class_SOURCES += opengl_dummy.f90
endif


cost733class_SOURCES += datainput.f90 clasinput.f90 \
days4mon.f90 nobs4dates.f90 scan_matfile.f90 \
finish.f90 subfunctions.f90 gaussfilter.f90 pca.f90 \
distfunc.f90 newseed.f90 sortcla.f90 numday.f90 \
percentile.f90 sort.f90 selectgrid.f90 \
list4dates.f90 coef4pcaproj.f90 geofunctions.f90 listcount.f90 \
centroids.f90 distancevect.f90 dist_ratio.f90 \
arguments.f90 dataviewinit.f90 clasoutput.f90 centoutput.f90 \
binclass.f90 jenkcoll.f90 wlk.f90 lit.f90 prototype.f90 \
tmodpca.f90 tmodpcat.f90 kruiz.f90 pcaxtr.f90 \
erpicum.f90 lund.f90 kirchhofer.f90 \
kmeans.f90 ckmeans.f90 dkmeans.f90 kmedoids.f90 sandra.f90 som.f90
    mixturemodel.f90 hcluster.f90 \
randommedoid.f90 randomclass.f90 \
assign.f90 substitute.f90 ecv.f90 brier.f90 \
aggregate.f90 correlate.f90 frequencies.f90 prognosis.f90 \
cpart.f90 drat.f90 ev_pf.f90 fsil.f90 sil.f90 wsd_cim.f90 \
compare_randindex.f90 aritest.f90 compare_patterns.f90 compare_timeseries.f90 \
plot_pixmap.f90 \
modttest.f90 effectivesamplesize.f90 \
main.f90
```

Bibliography

Beck, C., Jacobeit, J., and Jones, P. (2007). Frequency and within-type variations of large scale circulation types and their effects on low-frequency climate variability in central europe since 1780. *Int. J. Climatology*, 27:473–491.

Beck, C. and Philipp, A. (2010). Evaluation and comparison of circulation type classifications for the european domain. *Physics and Chemistry of the Earth*, 35(9-12):374–387.

Bernaards, C. and Jennrich, R. (2005). Gradient projection algorithms and software for arbitrary rotation criteria in factor analysis. *Educational and Psychological Measurement*, 65/5:676.

Bissolli, P. and Dittmann, E. (2003). Objektive wetterlagenklassen. In *Klimastatusbericht 2003*. DWD.

Blair, D. (1998). The kirchhofer technique of synoptic typing revisited. *Int. J. Climatology*, 18:1625–1635.

Buishand, T. and Brandsma, T. (1997). Comparison of circulation classification schemes for predicting temperature and precipitation in the netherlands. *Int. J. Climatology*, 17:875–889.

Calinski, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Commun. Stat.*, 3:1–27.

Comrie, A. (1996). An all-season synoptic climatology of air pollution in the u.s.-mexico border region. *Professional Geographer*, 48:237–251.

Dittmann, E., Barth, S., Lang, J., and Mueller-Westermeier, G. (1995). Objektive wetterlagenklassifikation. *Ber. Dt. Wetterd.*, 197.

Ekstroem, M., Joensson, P., and Baerring, L. (2002). Synoptic pressure patterns associated with major wind erosion events in southern sweden 1973 1991. *Climate Research*, 23:51–66.

Enke, W., Schneider, F., and Deutschländer, T. (2005). A novel scheme to derive optimized circulation pattern classifications for downscaling and forecast purposes. *Theor. Appl. Climatol.*, 82:51–63.

Enke, W. and Spekat, A. (1997). Downscaling climate model outputs into local and regional weather elements by classification and regression. *Climate Research*, 8:195–207.

Hartigan, J. (1975). *Clustering Algorithms.* Wiley Series in probability and mathematical statistics. John Wiley & Sons.

Hartigan, J. A. and Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

Hewitson, B. and Crane, R. (1992). Regional climates in the giss global circulation model and synoptic-scale circulation. *Journal of Climate*, 5:1002–1011.

Huth, R. (1996). An intercomparison of computer-assisted circulation classification methods. *Int. J. Climatol.*, 16:893–922.

Huth, R. (2000). A circulation classification scheme applicable in gcm studies. *Theor. Appl. Climatol.*, 67:1–18.

Jenkinson, A. and Collison, B. (1977). An initial climatology of gales over the north sea. In *Synop. Climatol. Branch, Memo. 62.* Meteorological Office.

Jones, P., Hulme, M., and Briffa, K. (1993). A comparison of lamb circulation types with an objective classification scheme. *International Journal of Climatology*, 13:655–663.

Kalkstein, L. S., Tan, G., and Skindlov, J. A. (1987). An evaluation of three clustering procedures for use in synoptic climatological classification. *J. Appl. Meteor.*, 26:717–730.

Kalnay, E., Kanamitsua, M., Kistlera, R., Collinsa, W., Deavena, D., Gandina, L., Iredella, M., Sahaa, S., Whitea, G., Woollena, J., Zhua, Y., Leetmaaa, A., Reynoldsa, R., Chelliahb, M., Ebisuzakib, W., Higginsb, W., Janowiakb, J., Mob, K., Ropelewskib, C., Wangb, J., Jennec, R., and Josephc, D. (1996). The ncep/ncar 40-year reanalysis project. *Bull. Amer. Meteor. Soc.*, 77:437–470.

Kirchhofer, W. (1974). Classification of european 500 mb patterns. In *Arbeitsbericht der Schweizerischen Meteorologischen Zentralanstalt*, volume 43, page 16. Bundesamt fuer Meteorologie und Klimatologie MeteoSchweiz.

Kruizinga, S. (1979). Objective classification of daily 500 mbar patterns. In *Preprints Sixth Conference on Probability and Statistics in Atmospheric Sciences, 9-12 October 1979 Banff, Alberta*, pages 126–129, Boston, MA. American Meteorological Society.

Kuncheva, L. and Hadjitodorov, S. T. (2004). Using diversity in cluster ensembles. *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2:1214–1219.

L.Hubert and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2:193–218.

Litynski, J. (1969). A numerical classification of circulation patterns and weather types in poland. (in polish). *Prace Panstwowego Instytutu Hydrologiczno-Meteorologicznego*, 97:3?15.

Lund, I. (1963). Map-pattern classification by statistical methods. *J. Appl. Meteorol.*, 2:56–65.

Milligan, G. and Cooper, M. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179.

Philipp, A., Bartholy, J., Beck, C., Erpicum, M., Esteban, P., Fettweis, X., Huth, R., James, P., Jourdain, S., Kreienkamp, F., Krennert, T., Lykoudis, S., Michalides, S. C., Pianko-Kluczynska, K., Post, P., Alvarez, D. R., Schiemann, R., Spekat, A., and Tymvios, F. S. (2010). Cost733cat - a database of weather and circulation type classifications. *Physics and Chemistry of the Earth*, 35(9-12):360–373.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *J. Amer. Stat. Assoc.*, 66:846–850.

Rousseeuw, P. (1987). Rousseeuw, p., 1987: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. journal of computational and applied mathematics, 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65.

Southwood, T. R. E. (1978). *Southwood, T. R. E., 1978: Ecological Methods, 2nd edn. London: Chapman and Hall.* Chapman and Hall.

Strehl, A. and Gosh, J. (2002). Cluster ensembles - a knowledge reuse framework for combining partitions. *Journal of Machine Learning Research*, 3:583–617.

Tang, L., Chen, D., Karlsson, P.-E., Gu, Y., and Ou, T. (2008). Synoptic circulation and its influence on spring and summer surface ozone concentrations in southern sweden. *Boreal Env. Res.*, 14:889–902.