WRF post-process tutorial

Markel García Díez
garciadm@unican.es
Santander Meteorology Group
Dept Applied Mathematics and Comp. Sci.
Universidad de Cantabria, Santander, Spain

Thanks to:
Jesús Fernández
L. Fita

Index

1. Preparing the session

Preparing the session

Go to **`/home/xubuntu/postprocess`**

There you should find the file postprocess_lecture.tar.gz

Untar it

```
tar xzvf postprocess_lecture.tar.gz
```

And now we are ready.

Index

What do we want:

Main problems

What do we want:

- Group the wrfout files into CF (http://cf-pcmdi.llnl.gov) compliant files for arbitrary periods of time.

- Perform operations over the available fields: change units, de-accumulate, averages, etc.

- Compute derived fields.

- Interpolate from the eta hybrid levels to pressure or height levels.

- Compute integrals along all the levels before filtering them.

- Detect the presence of corrupt files and missing data in the raw files.

Main problems

- Split files by variable, and vertical levels.

- Delete spin up and/or overlapping periods and concatenate the files correctly.

- Format the files to be compliant with more exigent conventions for a given project (e.g. CORDEX)

And we want to do *all this* **efficiently** and **minimizing the probability of bugs and human errors**.

Index

Decission → How many intermediate steps do we use?

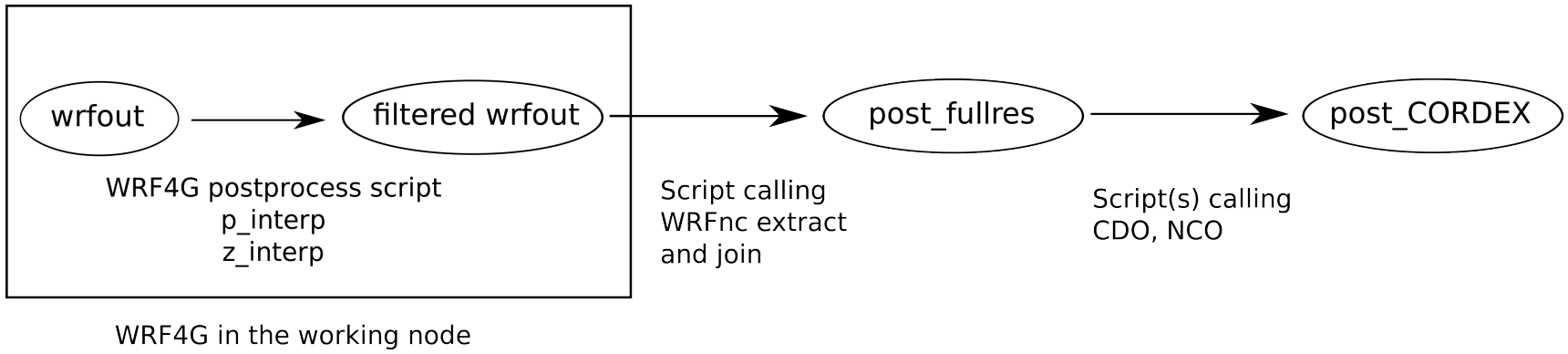
• The large amount of data (~ TB) usually involved suggest designing a process with very few steps. Operating “on the fly” could minimize the hard disk usage.

• But saving intermediate files before computing monthly and seasonal averages helps a lot to detect errors in the first steps of the postprocess and in the raw files.

Our approach

We decided to write several shell (BASH) scripts that call the utilities: p_interp, WRFnc extract and join, CDO, NCO...

WORKFLOW:



The next part of the lecture and the hands-on session is going to be focused in **WRFnc extract and join (XnJ)**, a tool developed by our group.

Index

Modified p_interp

p_interp is a tool written in fortran that reads raw wrfout files and interpolates the data from hybrid eta levels to pressure levels.

There is a version freely available in the WRF users web, but we do use a version which was modified mainly by Lluís Fita, and in less exent by Jesús Fernández and Markel García.

This version is also intended to be freely distributed, but still some of the computations would need to be revised.

Modified p_interp

Added variables in **p_interp**:
- MSLP (requires a revision)
- MSLPF (filtered version, space averaged)
- RAINTOT (total precipitation)
- CLT (requires a revision)
- column integrated amount of: VIM: moist, VIQ: condensed water, VIQC: cloud condensed water, wind (3D), VIQI: cloud ice
- VIMWIND: column integrated transport of moist

* Computation of mean sea level pressure requires a revision. ECMWF formula should be used?

* Total cloud cover now it is using Sunqvist 1997. Should we computed using max(CLDFRA) ?

* Addition of low (p > 660 hPa), medium (660 < p < 440 hPa), high (p < 440 hPa) cloud cover!

Modified p_interp

Also support for writing files in NETCDF4_CLASSIC format with compression has been added.

Compiling modified p_interp with netCDF4 support:

```
#!/bin/bash
use intel
export NETCDFDIR="/software/netcdf"
export HDF5DIR="/software/hdf5"
ifort -O3 -heap-arrays  p_interp.F90 -o p_interp \
-I$NETCDFDIR/include -I$HDF5DIR/include -Bstatic \
-L$NETCDFDIR/lib -lnetcdf -lnetcdff -L$HDF5DIR/lib \
-lnetcdf -lhdf5_hl -lhdf5 -lz >& compile.log
```

Modified p_interp

namelist.pinterp

```
&io
  path_to_input                  = './',
  input_name                     = 'wrfout.nc',
  path_to_output                 = './',
  fields                         =
'RAINTOT,T2,Q2,PSFC,U10,V10,CLT,T,GHT,SMOIS'
  process                        = 'list',
  debug                          = .FALSE.,
  grid_filt                      = 3,
  ntimes_filt                    = 10,
  output_netCDF4                 = .TRUE.,

&interp_in
  interp_levels                  =
1000.,987.5,975.,962.5,950.,937.5,925.,912.5,900.,887.5,875.,
850.,825.,800.,750.,700.,650.,600.,500.,
  extrapolate                    = 1,
  interp_method                  = 1,
  unstagger_grid                 = .TRUE.,
```

Modified p_interp

namelist.pinterp

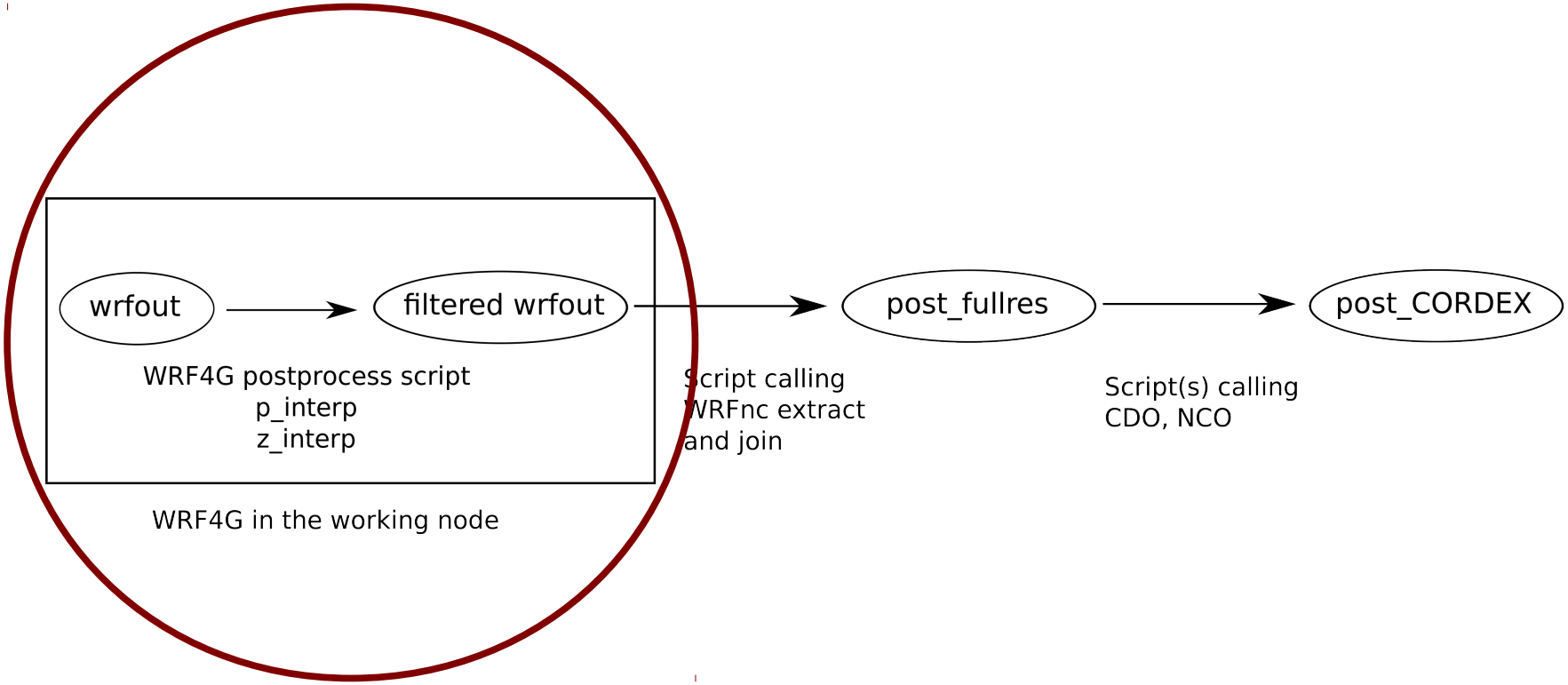
```
&io
  path_to_input                  = './',
  input_name                     = 'wrfout.nc',
  path_to_output                 = './',
  fields                         =
'RAINTOT,T2,Q2,PSFC,U10,V10,CLT,T,GHT,SMOIS
  process                        = 'list',
  debug                          = .FALSE.,
  grid_filt                      = 3,
  ntimes_filt                    = 10,
  output_netCDF4                 = .TRUE.,

&interp_in
  interp_levels                  =
1000.,987.5,975.,962.5,950.,937.5,925.,912.5,900.,887.5,875.,
850.,825.,800.,750.,700.,650.,600.,500.,
  extrapolate                    = 1,
  interp_method                  = 1,
  unstagger_grid                 = .TRUE.,
```

Modified p_interp

It is possible to include **p_interp** in the WRF4G postprocessor.

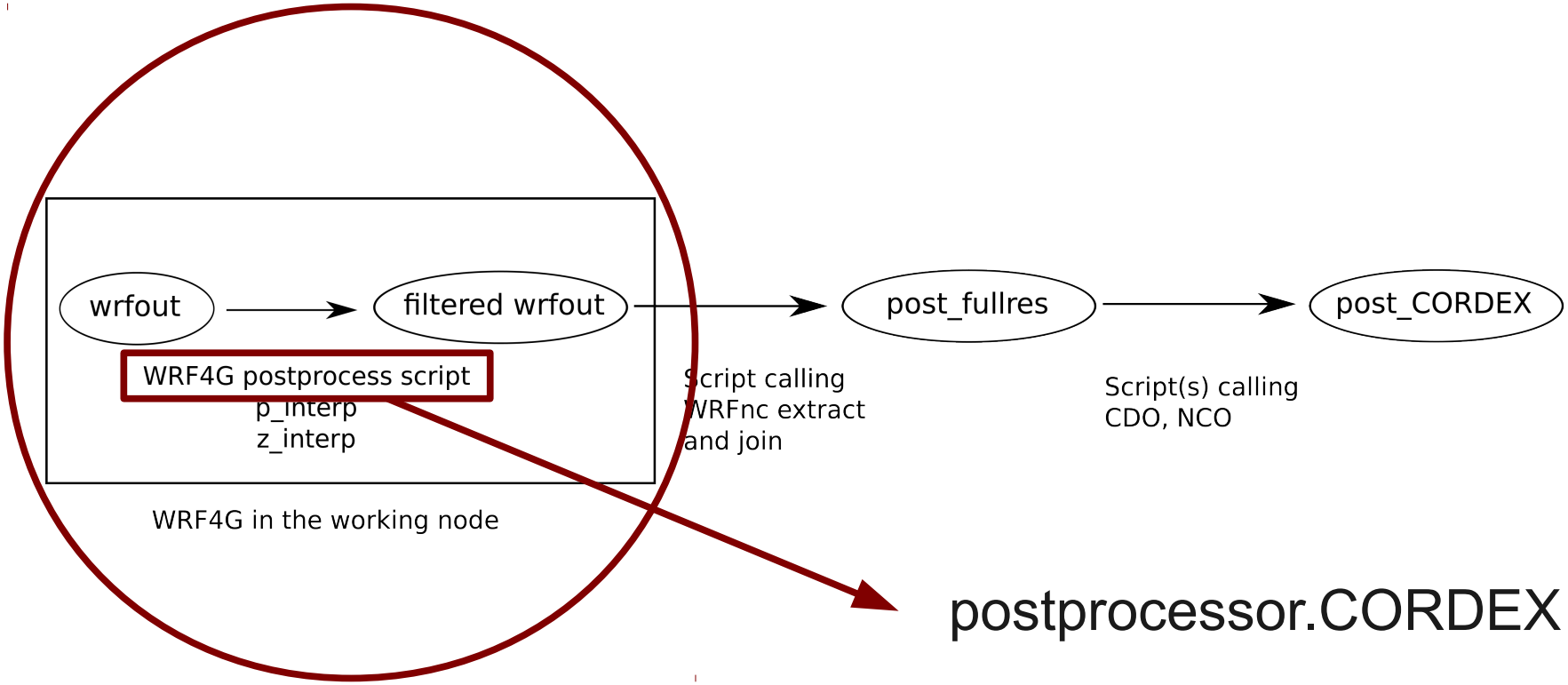
By this way, all the wrfout are automatically filtered and interpolated to pressure levels in the same computing resource where WRF is running.



Modified p_interp

It is possible to include p_interp in the WRF4G postprocessor.

By this way, all the wrfout files are automatically filtered and interpolated to pressure levels in the same computing resources where WRF is running.



Modified p_interp

The postprocess script must be called **postprocessor.SOMETHING** and it is set up in the experiment.wrf4g file as:

```
postprocessor="SOMETHING"
```

The script can be located in two places:

- **WRF4G tarball** located in **${WRF4G_LOCATION}/repository/apps** The path into the tarball should be **./bin**, where some example postprocessors are located.

- A folder called **wrf4g_files**, located in the same path where experiment.wrf4g is. Complete path should be **wrf4g_files/WRFV3/run**

Modified p_interp

This script has 1 mandatory argument, the wrfncfile

```
#!/bin/bash
wrfncfile=$1
```

We use BASH, but any language can be used provided it does accept this argument.

A namelist.pinterp file can be easily produced using the EOF construct.

More complex postprocessors can be built. For example: Filtering different variables and interpolating them to different levels (eta, height in meters, pressure, etc.)

Modified p_interp

Sample WRF4G postprocessor

```
#!/bin/bash
wrfncfile=$1

function pintnml(){
  idir=$1
  ifile=$2
  cat << EOF > namelist.pinterp
&io
  path_to_input                 = '${idir}/',
  input_name                    = '${ifile}',
  path_to_output                = '${idir}/',
  etc...
EOF
}

pintnml . ${wrfnc_file}
p_interp
rm namelist.pinterp
mv "${wrfnc_file}_PLEV" "${wrfnc_file}"
```

Index

WRFnc extract and join

WRF NetCDF Extract & Join (wrfncxnj) is a tool written in python that allows the user to generate CF compliant NetCDF files from the WRF raw files.

It has been originally developed at the University of Cantabria by Markel García-Díez, Jesús Fernández and Lluís Fita, and is released under GNU license.

More info and download:

http://www.meteo.unican.es/wiki/cordexwrf/SoftwareTools/WrfncXnj

WRFnc extract and join

The tool is intended to be used from the command line or a Shell script, and offers many options as flags:

- Extract variables and join them across several files

- Split the files by variable or vertical level (in the atmosphere or the soil)

- Select the levels to save and filter dates

- Compute derived variables

- Store the output in CF compliant files. Optionally write them in NetCDF4 compressed format

It does not: Compute monthly or seasonal averages. Check raw data looking for zeros or outliers.

WRFnc extract and join

As python is an interpreted language, **WRFnc extract and join does not need to be compiled or installed**. It is divided in 5 files:

- **wrfncxnj.py**: Main module.

- **wrfncxnj_base.py**: File containing the classes and the functions.

- **wrfncxnj_fun.py**: File containing the functions defined to compute specific derived variables.

- **wrfncxnj_cli.py**: File containing the definition of the flags.

- **wrfncxnj.table**: ASCII table with the information of the attributes that wrfncxnj needs to write CF compliant files.

It has a few dependencies. Most of them are available in the main linux repositories: **python**, **numpy**, **netCDF** libraries with netCDF4 enabled and **python netCDF4**.

Example: A CF compliant file generated by WRFnc extract and join:

```
ncdump -h example1.nc
```

As a command line calling XnJ can became very long, its more efficient to write shell scripts that call it. For example see **xnj_script_example1.sh** This script:

- Extracts many variables.

- Splits them is separated files.

- Selects three pressure levels and also splits them in different files.

- The name of the output files is specified with --output-pattern

- Runs in a temporary folder and copies the output to $OUTPUTDIR when finished.

WRFnc extract and join

Basic usage:

```
python wrfncxnj.py [options] [files to process]
```

To obtain a description of all the options available type:

```
python wrfncxnj.py -h
```

Essential elements:

```
python wrfncxnj.py [flags] \
--split-variables \
--output-pattern=EXPERIMENT_[varcf].nc \
-v "VAR1,VAR2,VAR3" \
wrfout1.nc wrfout2.nc ... wrfoutN.nc
```

WRFnc extract and join

Main flags of wrfncxnj:

--output-pattern Output pattern to use if the option --split-in-variables is activated. Patterns recognized are currently of the form **[varcf]_[varwrf]_[firsttime]_[lasttime]_experiment.nc**. Firsttime and lasttime are replaced by datetimes of the form YYYYmmddHH.

-o to directly specify an output file name.

-v VAR1[,VAR2,...], --variables=VAR1[,VAR2,...] Variables to extract. They need to be defined in wrfncxnj.table

-r YYYY-MM-DD_hh:mm:ss, --reference-date=YYYY-MM-DD_hh:mm:ss Reference date for the time axis in the output files.

-a attributes.file, --attributes=attributes.file Table for setting the global attributes of the file.

wrfncxnj.table

The wrfncxnj.table file is an ASCII column file with the semicolon ";" as field separator. It provides CF-conforming metadata for each variable. Also, derived variables can be implemented here, using formulas and functions defined in wrfncxnj_base.py and wrfncxnj_fun.py

Wrfname; AMIP; Long name; Standard name; Units; Function/formula; Vertical axis

Example entry:

```
T2; tas; Surface air temperature; air_temperature; K;
screenvar_at_2m;
```

```
float tas(time, height, rlat, rlon) ;
    tas:long_name = "Surface air temperature" ;
    tas:standard_name = "air_temperature" ;
    tas:units = "K" ;
    tas:coordinates = "lat lon" ;
    tas:grid_mapping = "Rotated_Pole" ;
```

Vertical axis and p_interp:

• Most of the projects ask for data in some pressure levels (850 hPa, 500 hPa, etc.)

• Currently wrfncxnj does not interpolate, this **interpolation must be carried out in a previous stage**.

• We saw how **p_interp** is a useful tool able to do this task.

• Variables in pressure levels (with a “p” in the 7th column of wrfncxnj.table) need to be previously interpolated through p_interp.

• It is possible to incorporate this step into the WRF4G workflow, calling p_interp from the postprocessor script.

Index

Index

Example 1: Processing 1 WRF variable

```
python wrfncxnj/wrfncxnj.py \
-o cf/EXPERIMENT_tas.nc \
-g raw/geo_em.d01_ecdx044.nc \
-v T2 \
raw/wrfout_d01*
```

Example 2: Processing pressure levels WRF variable

```
python wrfncxnj/wrfncxnj.py  \
--split-variables --split-levels \
--plevs-filter=850,700,500 \
--output-pattern=cf/EXPERIMENT_[varcf]_[level].nc \
-g raw/geo_em.d01_ecdx044.nc \
-v QVAPOR,TEMP \
raw/wrfout_d01*
```

Commands start to grow large, and defining a variable can help:

```
xnj="python wrfncxnj/wrfncxnj.py \
--split-variables \
--output-pattern cf/EXPERIMENT_[varcf]_[level].nc \
-g raw/geo_em.d01_ecdx044.nc -r 1940-01-01_00:00:00 \
-a wrfncxnj/wrfnc_extract_and_join.gattr_EUROCORDEX"
```

We add “-r” to define a reference date and “-a” to define global attributes.

Example 3: De-accumulate

Functions “deaccumulate_flux” or “deaccumulate” must be in the entry of wrfncxnj.table:

```
ACLWUPT; rlut; Outgoing LW radiation at top of atmosphere;
toa_outgoing_longwave_flux; W m-2  ;deaccumulate_flux
```

```
${xnj} -v ACLWUPT raw/wrfout_d01*
```

Using ncview we can see that the first time step is full of zeros. We don't want this.

```
${xnj} -v ACLWUPT \
--previous-file=raw/wrfout_d01_20010701T000000Z \
raw/wrfout_d01_20010702T000000Z.nc \
raw/wrfout_d01_20010703T000000Z.nc
```

Then the first time step is correct (but now is the first step of July 2nd).

If we want to deaccumulate “forward” then the equivalent flag would be –next-file e.g. RAINFORWARD.

Example 4: Derived variables

```
${xnj} -v TDPS raw/wrfout_d01*
```

A function called compute_TDPS is defined in wrfncxnj_fun.py

```
def compute_TDPS(varobj, onc, wnfiles, wntimes):
    t2 =    wnfiles.current.variables["T2"][:]
    q2 =    wnfiles.current.variables["Q2"][:]
    psfc =  wnfiles.current.variables["PSFC"]
    es = 10.* Constants.es_base_bolton *
    np.exp(Constants.es_Abolton*(t2-Constants.tkelvin)/(t2-
Constants.tkelvin+Constants.es_Bbolton))
    rh = 0.01*psfc[:]/es*(q2/(Constants.RdRv+q2))
    ...etc
```

Example 5: fullfile

Some derived variables need fixed fields that are not available in the geo_em file. Then we use –fullfile to point to a original raw file, not passed through p_interp.

```
${xnj} -v MRSO raw/wrfout_d01*
```

Gives an error. To overcome it we use --fullfile:

```
${xnj} --fullfile=raw/wrffull_eurocordex044.nc \
-v MRSO raw/wrfout_d01*
```

Example 6: fixed fields

Using –single-record it is possible to extract the fixed fields from the geo_em, without any wrfout being involved.

```
${xnj} --single-record -v  HGT_M,LANDMASK
```

Example 7: Filter dates

```
python wrfncxnj/wrfncxnj.py \
-o cf/EXPERIMENT_tas.nc \
-g raw/geo_em.d01_ecdx044.nc \
--filter-times=2001070112,2001070312 \
--temp-dir=xnj.$(date '+%Y%m%d%H%M%S')\
-v T2 \
raw/wrfout_d01*
```

WRFnc extract and join limitations

- There are still some features missing to be fully CF compliant: time_bonds, cell_method, etc.

- Detection of gaps and missing data can be improved. User must check the output files (e.g. with ncview), to be sure that all the data is present.

- It does not complete the whole postprocess, since it does not compute monthly and seasonal statistics.

WRFnc extract and join limitations

- There are still some features missing to be fully CF compliant: time_bonds, cell_method, etc.

- Detection of gaps and missing data can be improved. User must check the output files (e.g. with ncview), to be sure that all the data is present.

- It does not complete the whole postprocess, since it does not compute monthly and seasonal statistics.

Ideas to reach the final processed files

CDO and NCO are very useful. However, they have some limitations:

- They need to be called from shell scripts, but writing and debugging complex programs in BASH can be annoying.

- The CDO does inevitably delete some attributes when processing files, which need to be restored.

- Also, writing this programs in python, or other high level language, would imply a lot of system calls.

- Finally, the best approach may be combining shell with tools written in other languages, as wrfncxnj. Each group must make its own decision, depending on its needs. Currently we do use only CDO and NCO.

Thank you !!

Questions to garciadm@unican.es