Table of Contents

Reforecast Tutorial

How to get driving model (NCEP) data.

In this example, the publicly available NCEP Reanalysis (run 1) data are going to be used. This data can be dowloaded from ?http://nomad3.ncep.noaa.gov/pub/reanalysis-1/6hr in GRIB format. These are monthly files that get updated each month nearly in real time. Two files are needed for each month, one with the pressure level data, labeled "pgb", and other one with 2D data, labeled "grb2d". extdata_path defined in experiment.wrf4g must point to the folder where these files are located. Alternatively, it is possible to write a preprocessor that downloads the data itself. Note that the file names must be parsed by the preprocessor. In this case, if both files are located into the same folder, and provided the extension **.grb** is appended to them, the default preprocessor will parse them correctly, since it looks for monthly files with year/month (YYYY/mm) into their names. For example, the files for December 2010 should be:

```
[user@mycomputer~]$ ls
grb2d201001.grb
pgb.ft00.201001.grb
```

Creating a WRF experiment

Keeping organized

Before starting to create an experiment, is good practice to create some directories to be tidy. For example, if our project is called **seawind**, you can create the following directory hierarchy.

```
/projects/seawind/submit/exp1
/projects/seawind/submit/exp2
...
/projects/seawind/domains
/projects/seawind/data
/projects/seawind/scripts
/projects/seawind/figures
```

Of course, many other combinations are possible, depending in the organization of the resources available to the user.

The test experiment

Before creating a large experiment, with many chunks and realizations, it is convenient to run a smaller test experiment with exactly the same model configuration. This way you can see that everything is working as you want. Frequently, some attempts are needed before WRF runs, because of mistakes in the configuration files or in the set up of input files.

Go to the **submit** directory and create and define and called **seawind**:

```
[user@mycomputer~]$ cd /projects/seawind/submit
[user@mycomputer~]$ wrf4g exp seawind define
```

Now you can configure our test experiment, following the instructions in experiment.wrf4g. A typical configuration for a reforecast test would be:

```
#           start_date          | end_date            | chunk_size | simulation_interval | simulation_length
date_time = 2009-12-31_18:00:00 | 2010-01-02_06:00:00 | 36 hours   | 24 hours            | 36 hours
```

Once it is finished, you can use the command line interface (CLI) to create and submit our test experiment. Before submitting check in the output of `wrf4g exp seawind create` that the chunks and realizations being created are those that you wanted.

```
[user@mycomputer~]$ wrf4g exp seawind create
[user@mycomputer~]$ wrf4g exp seawind submit
```

Now you can monitor the experiment using `wrf4g exp seawind status`, along with the option `--delay`:

```
[user@mycomputer~]$ wrf4g exp seawind status --delay 2
```

Probably it will fail in the first attempt, so don't worry. If it fails, see `wrf4g exp --help` to look for the errors.

Once the test experiment has run successfully, you should check that the output looks fine and that it contains all the variables that you want. Tools like ?ncview and ?ncdump are very useful for this task. This step is specially important if you are using a postprocessor in WRF4G to filter variables because of disk space limitations.

The experiment

At this point you are done with the most difficult issues, and you are ready to set up the reforecast experiment itself. Then change the experiment_name and extend the end_date to the end date of the full experiment. In this example you will only run one month (January 2010), but it could be many decades. So our dates would be:

```
#           start_date          | end_date            | chunk_size | simulation_interval | simulation_length
date_time = 2009-12-31_18:00:00 | 2010-01-31_06:00:00 | 36 hours   | 24 hours            | 36 hours
```

Finally, update and submit the experiment with WRF4G CLI, as before.

```
[user@mycomputer~]$ wrf4g exp seawind update
[user@mycomputer~]$ wrf4g exp seawind submit
```

Experiment monitoring

With `wrf4g exp seawind submit` user can monitor the state of all the realizations of the experiment. If the experiment is large, you can use `--pattern` or `--rea-status` in combination to filter the list with some criteria. For example:

```
[user@mycomputer~]$ wrf4g exp seawind submit --rea-status FINISHED
```

Returns all the **FINISHED** realizations. Or, more complicated:

```
[user@mycomputer~]$ wrf4g exp seawind submit --rea-status FAILED --pattern *2010012*
```

Returning all the realizations that are currently in some stage of the WRF4G workflow (UNGRIB, METGRID, WRF, etc.)

If `submit` is called again, the realizations in **FAILED** status are re-submitted. This is very useful to re-submit simulations after they crashed because of some problems in the computing infrastructure.

Also, some realizations may fail because particular problems. Unfortunately, there are a lot of things that can fail, and covering them in this tutorial is not possible. After years using WRF we still find new error messages sometimes. In a few realizations, WRF may crash because of some points not filling **cfl** criteria. This numerical instabilities arise when too strong gradients do appear for vertical velocity or some other variable. They can be solved using a lower timestep. However, each WRF4G experiment has a fixed timestep defined. Thus, in this case, a new experiment with a lower **timestep_dxfactor** (e.g. sw_failed_days with timestep_dxfactor = 5) must be created.

When a realization has failed, you can obtain its log by using `log` subcommand :

```
[user@mycomputer~]$ wrf4g rea seawind-default-20091231T180000 log 1
```

And it can be resubmitted using the `--rerun`:

- Re-submit all failed realizations:

  ```
  [user@mycomputer~]$ wrf4g exp seawind submit --rea-status FAILED --rerun
  ```

  Re-submit all realizations with the pattern **2010012**:

  ```
  [user@mycomputer~]$ wrf4g exp seawind submit --pattern *2010012* --rerun
  ```

Postprocess

If the computing resources and the model configuration are stable enough, once running the experiment the monitoring is not time consuming. However, if the experiment is large (e.g. 30 year of daily reforecasts), the postprocessing of the output produced by WRF4G can be a complicated and error-prone task. Here we provide some guidelines to follow to successfully postprocess a reforecast like experiment like this. In this tutorial case it should not be too complicated, since it is only one month.

1st step: Generate daily CF-compliant files.

To generate CF-compliant files from WRF4G raw data should not be complicated, since it is applying a dictionary to variable name and attributes. However, in fact, we want to do many other things:

- Merge the files apart from traducing them to CF.compliant netCDF.
- Perform operations over the available fields: change units, de-accumulate, averages, etc.
- Compute derived fields.
- Split files by variable, and vertical levels.
- Delete spin up and/or overlapping periods and concatenate the files correctly.

To deal with them, we created a tool written in python called ?WRFnc extract and join, published with a GNU open license. It is documented in that web.

In the last versions, `wrfncxnj` can even remove the spin up period, as you can filter a time interval with the `--filter-times` flag. If you want to average the jumps between realizations, you retain one extra hour for each day.

What do you need now it to write an small shell script that will call this python script with the proper arguments. First, you need to define the paths of the different files. An useful practice is to write a small file called "dirs" with these paths, something like:

```
BASEDIR="/home/users/curso01"
EXPDIR="${BASEDIR}/data/raw"
DATADIR="${BASEDIR}/data"
SCRIPTDIR="${BASEDIR}/scripts"
POSTDIR="${BASEDIR}/data/post_fullres"
POST2DIR="${BASEDIR}/data/post_ih"
```

Now writing "source dirs" in our postprocess script you can easily load these variables. A sample post-process script would be:

```
#!/bin/bash
set -e

year=2010
exp="seawind"
fullexp="SEAWIND_NCEP"
varlist_xtrm="U10MEANER,V10MEANER"

mkdir -p ${POSTDIR}/${year} || exit
cd ${POSTDIR}/post_fullres

for dir in ${EXPDIR}/${exp}/${exp}__${year}*
do
 read expname dates <<< ${dir//__/ }
 read datei datef <<< ${dates//_/ }
 fdatei=$(date -u '+%Y%m%d%H' -d "${datei:0:8} ${datei:8:2} 12 hour")
 fdatef=$(date -u '+%Y%m%d%H' -d "${datei:0:8} ${datei:8:2} 36 hour")
 expname=$(basename ${expname})
 geofile="/home/users/user/domains/Europe_30k/geo_em.d01.nc"
 xnj="python \
   / \
   -r 1940-01-01_00:00:00 \
   -g ${geofile} -a ${SCRIPTDIR}/xnj_East_Anglia.attr \
   --fullfile=${SCRIPTDIR}/wrffull_${dom}.nc \
   --split-variables --split-levels --output-format=NETCDF4_CLASSIC \
   --temp-dir=/localtmp/xnj.$(date '+%Y%m%d%H%M%S') \
   --filter-times=${fdatei},${fdatef} \
   --output-pattern=${POSTDIR}/post_fullres/${year}/[varcf]_[level]_${fullexp}_$(dom2dom ${dom})__[firsttime]_[lasttime].n
```

```
 #
 # xtrm
 #
 filesx=$(find ${dir}/output -name 'wrfxtrm_'${dom}'_*.nc' | sort)
 if test $(echo ${filesx} | wc -w) -ne 7 ; then
   echo "Wrong number of files on $datei"
   continue
 fi
 ${xnj} -a wrfnc_extract_and_join.gattr_SEAWIND \
 -t ${SCRIPTDIR} -v ${varlist_xtrm} ${filesx}
done
```

These scripts can be sent to the queue with the proper command: qsub or msub. Either they can be run in interactive mode. Final files need to be checked to correct holes or incorrect values.

2nd step: Concatenate the daily realizations

After the first step, you now have much more friendly files, so this step can be carried out with many of the netCDF supporting languages available. You have carried out this task with ?Climate Data Operators (CDO) or with ?netcdf4-python package.

A python script we created called `py_netcdf_merge_and_average.py` can very much simplify this task.

```
Usage: py_netcdf_merge_and_average.py [options]

Options:
 -h, --help            show this help message and exit
 -f                    Overwrite the output file if exists.
 --selyear=YEAR        Filter time steps outside this year
 --bdy_points=BDY_POINTS
                       Number of points to delete from the boundaries
 --average-jumps       If True, averages the repeated timesteps found when
                       concatenating the input files to try to smooth
                       jumpiness
```