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:

```
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", we 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 we can see that everything is working as we 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" folder and create another folder called "sw_test":

```
cd projects/seawind/submit
mkdir sw_test
cd sw_test
```

Now we need to copy here the templates of experiment.wrf4g and resources.wrf4g.

```
cp $WRF4G_LOCATION/experiments/wrfuc_single_serial/experiment.wrf4g .
cp $WRF4G_LOCATION/etc/resources.wrf4g .
```

Now we can configure our test experiment, following the instructions in WRF4Gexperiment_wrf4g and WRF4Gresources_wrf4g. Note that, as it is a reforecast, we need to use the multiple dates configuration variables. A typical configuration for a reforecast test would be:

```
start_date="2009-12-31_18:00:00"
end_date="2010-01-02_06:00:00"
chunk_size_h=36
multiple_dates=1
 simulation_interval_h=24
 simulation_length_h=36
```

Once we are finished, we can use the command line interface (CPI) to prepare and submit our test experiment. Before submitting check in the output of wrf4g_prepare that the chunks and realizations being created are those that we wanted.

```
wrf4g_prepare
wrf4g_submit -e sw_test
```

Now we can monitor the experiment using wrf4g_status, along with the "watch" command.

```
watch wrf4g_status -l -e sw_test
```

Probably it will fail in the first attempts, so don't worry. If it fails, see how to manage WRF4G errors to look for the error.

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

The experiment

At this point we are done with the most difficult issues, and we are ready to set up the reforecast experiment itself. Simply create another folder inside "submit" named "sw_ncep", and copy there the configuration files of the test experiment.

```
cd ..
mkdir sw_ncep
cp -r ../sw_test/* .
```

Then change the experiment_name and extend the end_date to the end date of the full experiment. In this example we will only run one month (January 2010), but it could be many decades. So our dates would be:

```
start_date="2009-12-31_18:00:00"
end_date="2010-01-31_06:00:00"
```

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

```
wrf4g_prepare
wrf4g_submit -e sw_test
```

Experiment monitoring

With wrf4g_status user can monitor the state of all the realizations of the experiment. If the experiment is large, wrf4g_status can be used in combination with shell tools as grep or awk to filter the list with some criteria. For example:

```
wrf4g_status -l -e sw_test | grep Finished
```

Returns all the Finished realizations. Or, more complicated

```
wrf4g_status -l -e sw_test | grep -v Finished | grep -v Submitted | grep -v Failed
```

Returns all the realizations that are currently in some stage of the WRF4G workflow (Down. Bin., ungrib, metgrid..., etc.)

If wrf4g_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. There is a page in this wiki called WRFKnownProblems where many of them are commented. 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.

Other more complicated situations can occur. For example, if there is a blackout, the jobs can fail before they can send the "Failed" signal to the database. In that case, they can appear to be indefinitely in "WRF" status. These kind of problems need a less confortable monitoring, such as entering to the working nodes and use the commands "top" or "ps -ef" to see it WRF is really running.

When a realization has failed but does not appear with the "Failed" status, it can be resubmitted using the --force flag of wrf4g_submit. Note also that wrf4g_submit can be used refering only to one realization or chunk, using the flags, -r or -c.

The monitoring and maintenance of an experiment can be shared my different users, provided they connect their WRF4G to the same database.

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 we can filter a time interval with the --filter-times flag. If we want to average the jumps between realizations, we to retain one extra hour for each day.

What do we need now it to write an small shell script that will call this python script with the proper arguments. First, we 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 we can easily load these variables. A sample script would be:

```
#!/bin/bash
use python

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/curso01/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, we now have much more friendly files, so this step can be carried out with many of the netCDF supporting languages available. We 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
                       jumpyness
```