pyg2p User Manual

Version	1.3.1
Author	Domenico Nappo (domenico.nappo(at)gmail.com)
Release Date	2014-07-08

Table of Contents

Introduction

pyg2c is the python porting of the grib2pcraster C/C++ application in use at ECMWF and JRC for some EFAS and GLOFAS tasks.

It reads georeferenced variables from GRIB version 1 and 2 files and it produces PCRaster maps, after some manipulation steps. Execution parameters are passed via command line arguments and XML template files.

Performances are comparable to the C/C++ application, except if using scipy based *nearest* and *invdist* interpolation modes, which at current version are not usable for daily operations. Use *grib_nearest* and *grib_invdist* modes for the time being.

Otherwise, for research and/or small grids, you can use them as well and compare results.

Manipulations are completely configurable in execution XML files and are:

- conversion (formulas are to configure in parameters.xml)
- correction
- aggregation (accumulation or average).

The interpolation mode is also configurable and it's needed to resample the input GRIB grid to the PCRaster target grid (which is determined by latitude and longitudes maps).

Usage

To use the application, after the main configuration you followed in the configuration manual, you need to configure a template commands XML for each type of extraction you need to perform.

In this template you configure the variable to extract, conversion, correction, type of aggregation, interpolation parameters, path to the clone PCRaster map for output maps, and their naming.

See the configuration manual to know more about XML tags and attributes to use or later in this manual for specific configurations about interpolation, aggregation, conversion, and correction.

Grabbing information from GRIB files.

To configure the application and compile your XML templates, you might need to know the variable shortName as stored in the input GRIB file you're using or in the geopotential GRIB. Just execute the following GRIB tool command:

```
grib_get -p shortName /path/to/grib
```

Other keys you would to know for configuration or debugging purposes are:

- startStep

- endStep (for instantaneous messages, it can be the same of startStep)
- perturbationNumber (the EPS member number)
- stepType (type of field: instantaneous: 'instant', average: 'avg', cumulated: 'cumul')
- longitudeOfFirstGridPointInDegrees
- longitudeOfLastGridPointInDegrees
- latitudeOfFirstGridPointInDegrees
- latiitudeOfLastGridPointInDegrees
- Ni (it can be missing)
- Nj (it states the resolution: it's the number of points along the meridian)
- numberOfValues
- gridType (e.g.: regular_ll, reduced_gg, rotated_ll)

See Mapping variables section in Appendix B – mapping between grib2pcraster and pyg2p for further information.

Input arguments

The table below describes the command line input arguments to use with pyg2p application. Mandatory arguments are in bold.

Argument name (short and long form)	Details
-i, --inputFile=	The grib file. Use grib tools to get information about it.
-I, --inputFile2=	A second file with messages at differen spatial resolution.
-m, --perturbationNumber	The eps member number.
-c, --commandsFile	Path to the XML template file.
-s, --start	Starting grib timestep. It overwrites the tstart attribute in the execution XML template.
-e, --end	Ending grib timestep. It overwrites the tend attribute in the execution XML template.
-T, --dataTime	Selector for messages with *dataTime* GRIB parameter,.
-o, --outDir	Directory of output maps. Default is current directory (./). The output directory will be created if not existing yet.
-l, --loggerLevel	Level of logging to console. Can be ERROR, INFO, DEBUG. Default is the level for the console logger as configured in the logger-configuration.xml file.
-d, --outLogDir	Logs directory. Default is ./logs. Will be created if not existing.
-g, --addGeopotential	Add a new geopotential grib file (used in correction step) to

	the current configuration: file will be copied into geopotential folder and the XML configuration will be updated.
-t, --test	Path to the test xml configuration file (where to configure paths of grib2pcraster, PCRaster pcrcalc etc.)

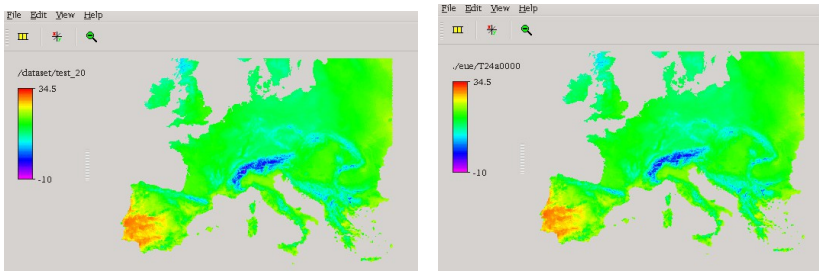
Example usages:

```
./pyg2p.py -c ./exec.xml -i ./input.grib -o /out/dir -s 12 -e 36
./pyg2p.py -c ./exec.xml -i ./input.grib -o /out/dir -m 10 -l INFO -d /var/logs/
./pyg2p.py -c ./exec.xml -i ./input.grib -I /input2ndres.grib -o /out/dir -m 10 -l DEBUG
./pyg2p.py -g /path/to/geopotential/grib/file
./pyg2p.py -t /path/to/test/xml/file
./pyg2p.py -h
```

Check output maps

After the execution, you can check how output maps were written using the PCRaster[1] Aguila viewer. You can also compare maps from grib2pcraster with those from pyg2p you just created:

```
aguila ./eue/T24a0000.001+10 /dataset/test_2013330702/T24aEFAS.001+10
```



Maps will be written in the folder specified by **-o** input argument. If this is missing, you will find maps in the folder where you launched the application (./).

Refer to PCRaster documentation for further information about Aguila.
See also how to configure and run a test suite with pyg2p for automatic comparison between grib2pcraster and pyg2p.

1 http://pcraster.geo.uu.nl/

Interpolation modes

Interpolation is configured in XML execution templates using the tag *Interpolation*. Depending on the interpolation method you use, the number of attributes will differ.

There are four interpolation methods available. Two are using GRIB_API nearest neighbours routines while the other two levereage on Scipy kd_tree module.

When configuring interpolation, you always must set target EFAS coordinate maps, because their metadata will serve to select the right interpolation table.

Interpolation will use precompiled intertables. They will be found in the configuration/intertables folder or a custome folder, using the attribute intertableDir within the Interpolation XML tag.

If interlookup table doesn't exist, the application will create it. Be aware that for certain combination of grid and maps, the creation of the interlookup table (which is a numpy array saved in a binary file) could take several minutes or hours for GRIB interpolation methods. Performances are much better with scipy based interpolation but this option could not be viable for all GRIB inputs.

GRIB API interpolation methods

grib_nearest:

This method uses GRIB API to perform nearest neighbour query.

To configure this method:

```
<Interpolation mode="grib_nearest"
               latMap="/dataset/maps/lat.map"
               lonMap="/dataset/maps/lon.map" />
```

grib_invdist:

It uses GRIB_API to query for four neighbours and relative distances. It applies inverse distance calculation to compute the final value.

To configure this method:

```
<Interpolation mode="grib_invdist"
               latMap="/dataset/maps/lat.map"
               lonMap="/dataset/maps/lon.map" />
```

Scipy interpolation methods

nearest:

It's the same nearest neighbour algorithm of grib_nearest but it uses the *scipy kd_tree*[2] module to obtain neighbours and distances.

```
<Interpolation mode="nearest"
               latMap="/dataset/maps/lat.map"
               lonMap="/dataset/maps/lon.map" />
```

invdist:

It's the inverse distance algorithm with *scipy.kd_tree* , using 8 neighbours.

```
<Interpolation mode="invdist" p="1" leafsize="10" eps="0.1"
               latMap="/dataset/maps/lat.map"
               lonMap="/dataset/maps/lon.map" />
```

Attributes *p*, *leafsize* and *eps* for the kd tree algorithm are default in scipy library:

Attribute	Details
p	2 (Euclidean metric)
eps	0
leafsize	10

OutMaps configuration

Interpolation is configured under the **outMaps** tag. With xml attributes, you also configure attributes for resulting PCRaster maps. Output dir is ./ by default or you can set it via command line using the option -o (--outDir).

Attribute	Details
namePrefix	Prefix name for output map files. Default is the value of shortName key.
unitTime	Unit time in hours for results. This is used during aggregation steps.
fmap	Extension number for the first map. Default 1.
ext	Same as grib2pcraster command line option. Default 1.
cloneMap	Path to a PCRaster clone map, needed by PCRaster libraries to write a new map on disk.

2 http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html

Aggregation

Values from grib files can be aggregated before to write the final PCRaster maps. There are two kind of aggregations available: average and accumulation. The XML configuration in the execution file will look like:

```
<Aggregation type="average" step="24" />
```

Attribute	Details
`type`	Type of aggregation. Accepted values can be: • *average* • *accumulation*
`step`	Step time window in hours.
`forceZeroArray`	Optional. Only for accumulation aggregation, if this option is set to true the program will use a zero array as GRIB message at step 0 in its computations.

To better understand what these two types of aggregations do, the DEBUG output of execution is presented in the following paragraphs.

Average

Temperatures are often extracted as averages on 24 hours or 6 hours. Here's a typical execution configuration and the output of interest:

Configuration:

```
<Execution name = "cosmo_T24" id="2" >
      <Parameter shortName="2t" applyConversion="k2c"
            gem="(z/9.81)*0.0065"
            correctionFormula="p+gem-dem*0.0065"
            demMap="/dataset/maps/cosmo/dem05.map" />

       <Aggregation type="average" step="24" />

      <OutMaps namePrefix="T24a" fmap="1" ext="4" unitTime="24"
             cloneMap="/dataset/maps/cosmo/dem05.map" >
         <Interpolation mode="grib_nearest"
                  latMap="/dataset/maps/cosmo/cosmo05_lat.map"
                  lonMap="/dataset/maps/cosmo/cosmo05_lon.map" />
      </OutMaps>
</Execution>
```

Command:

```
.pyg2p.py -l DEBUG -c /execution_templates/cosmo_t24.xml -i
/dataset/cosmo/2012111912_pf10_t2.grb -o ./cosmo -m 10
```

Output:

```
[2013-07-11 23:23:23,836] : Opening the GRIBReader for
/dataset/cosmo/2012111912_pf10_t2.grb
[2013-07-11 23:23:23,863] : Grib input step 6 [type of step: instant]
[2013-07-11 23:23:23,863] : Gribs from 0 to 132
...
...
[2013-07-11 23:36:13,366] :******** **** MANIPULATION **** **********
[2013-07-11 23:36:13,367] :temp_sum += grib[1] from -> grib[6]
[2013-07-11 23:36:13,373] :temp_sum += grib[2] from -> grib[6]
[2013-07-11 23:36:13,378] :temp_sum += grib[3] from -> grib[6]
[2013-07-11 23:36:13,382] :temp_sum += grib[4] from -> grib[6]
[2013-07-11 23:36:13,387] :temp_sum += grib[5] from -> grib[6]
[2013-07-11 23:36:13,391] :temp_sum += grib[6]
[2013-07-11 23:36:13,396] :temp_sum += grib[7] from -> grib[12]
[2013-07-11 23:36:13,400] :temp_sum += grib[8] from -> grib[12]
...
[2013-07-11 23:36:13,465] :temp_sum += grib[22] from -> grib[24]
[2013-07-11 23:36:13,470] :temp_sum += grib[23] from -> grib[24]
[2013-07-11 23:36:13,475] :temp_sum += grib[24]
[2013-07-11 23:36:13,485] :out[s:0 e:24 res:415 step-lenght:24] = temp_sum / 24
...
...
[2013-07-11 23:36:13,840] :out[s:72 e:96 res:415 step-lenght:24] = temp_sum / 24
[2013-07-11 23:36:13,841] :temp_sum += grib[97] from -> grib[102]
[2013-07-11 23:36:13,845] :temp_sum += grib[98] from -> grib[102]
[2013-07-11 23:36:13,850] :temp_sum += grib[99] from -> grib[102]
[2013-07-11 23:36:13,854] :temp_sum += grib[100] from -> grib[102]
[2013-07-11 23:36:13,859] :temp_sum += grib[101] from -> grib[102]
[2013-07-11 23:36:13,864] :temp_sum += grib[102]
[2013-07-11 23:36:13,868] :temp_sum += grib[103] from -> grib[108]
[2013-07-11 23:36:13,873] :temp_sum += grib[104] from -> grib[108]
...
[2013-07-11 23:36:13,939] :temp_sum += grib[118] from -> grib[120]
[2013-07-11 23:36:13,943] :temp_sum += grib[119] from -> grib[120]
[2013-07-11 23:36:13,948] :temp_sum += grib[120]
[2013-07-11 23:36:13,959] :out[s:96 e:120 res:415 step-lenght:24] = temp_sum / 24
```

ext parameter

In configuration, the effect of setting the *ext* value is clear when looking on how the output maps are numbered:

```
[2013-07-12 00:06:18,545] :./cosmo/T24a0000.001 written!
[2013-07-12 00:06:18,811] :./cosmo/T24a0000.005 written!
[2013-07-12 00:06:19,079] :./cosmo/T24a0000.009 written!
[2013-07-12 00:06:19,349] :./cosmo/T24a0000.013 written!
[2013-07-12 00:06:19,620] :./cosmo/T24a0000.017 written!
```

This is needed because we performed 24 hours average over 6 hourly steps.

Accumulation

For precipitation values, accumulation over 6 or 24 hours is often performed. Here's an example of configuration and execution output in DEBUG mode.

Configuration:

```
<Execution name = "dwd_rain_gsp" id="4" >
   <Parameter shortName="rain_gsp" tstart="6" tend="48"/>
   <Aggregation type="accumulation" step="6" forceZeroArray="true" />

   <OutMaps namePrefix="R06a" fmap="1"
         cloneMap="/dataset/maps/dwd/dem05.map" unitTime="24">

      <Interpolation mode="grib_invdist" intertableDir="./alternativeCosmoGrids"
               latMap="/dataset/maps/dwd/dwdlme05_lat.map"
               lonMap="/dataset/maps/dwd/dwdlme05_lon.map" />
   </OutMaps>
</Execution>
```

Command:

```
.pyg2p.py -l DEBUG -c /execution_templates/dwd_r06.xml -i
/dataset/dwd/2012111912_pf10_tp.grb -o ./cosmo -m 10
```

Output:

```
[2013-07-11 23:33:19,646] : Opening the GRIBReader for
/dataset/dwd/grib/dwd_grib1_ispra_LME_2012111900
```

```
[2013-07-11 23:33:19,859] : Grib input step 1 [type of step: accum]
[2013-07-11 23:33:19,859] : Gribs from 0 to 78
...
...
[2013-07-11 23:33:20,299] : ******** **** MANIPULATION **** ************
[2013-07-11 23:33:20,299] : Accumulation at resolution: 657
[2013-07-11 23:33:20,300] : out[s:6 e:12 res:657 step-lenght:6] = grib:12 - grib:6  * (24/6))
[2013-07-11 23:33:20,316] : out[s:12 e:18 res:657 step-lenght:6] = grib:18 - grib:12  * (24/6))
```

Note: If you want to perform **accumulation** from Ts to Te with an aggregation step Ta, and **Ts-Ta=0** (e.g. Ts=6h, Te=48h, Ta=6h), the program will select the first message at step 0 if present in the GRIB file, while you would use a zero values message instead.

To use a zero values array, set the attribute *forceZeroArray* to "true" or "yes" in the Aggregation configuration element.

For some DWD[3] and COSMO[4] accumulated precipitation files, the first zero message is an instant precipitation (not clear what it is) and the decision at EFAS was to use a zero message, as it happens for UKMO[5] extractions, where input GRIB files don't have a first zero step message.

```
$ grib_get -p units,name,stepRange,shortName,stepType 2012111912_pf10_tp.grb

kg m**-2 Total Precipitation 0       tp instant
kg m**-2 Total Precipitation 0-6     tp accum
kg m**-2 Total Precipitation 0-12    tp accum
kg m**-2 Total Precipitation 0-18    tp accum
...
kg m**-2 Total Precipitation 0-48 tp accum
```

3 http://www.dwd.de/
4 http://www.cosmo-model.org/
5 http://www.metoffice.gov.uk/

Correction

Values from grib files can be corrected with respect to their altitude coordinate (Lapse rate formulas). Formulas will use also a geopotential value (to read from a GRIB file, see later in this chapter for configuration).

Correction has to be configured in the Parameter tag, with three mandatory attributes.

- correctionFormula (the formula used for correction, with input variables parameter value (p), gem, and dem value.
- gem (the formula to obtain gem value from geopotential z value)
- demMap (path to the DEM PCRaster map)

Tested configurations are only for temperature and are specified as follows:

Temperature correction:

```
<Parameter shortName="2t" applyConversion="k2c"
             gem="(z/9.81)*0.0065"
          correctionFormula="p+gem-dem*0.0065"
          demMap="/dataset/maps/cosmo/dem05.map" />
```

Evotranspiration correction:

```
<Parameter shortName="2t" applyConversion="k2c"
          gem="(10**((-0.159)*(z/9.81)/1000))"
          correctionFormula="p/gem*(10**((-0.159)*dem/1000))"
          demMap="/dataset/maps/europe5km/dem.map"/>
```

How to write formulas

z is the geopotential value as read from the grib file
gem is the value resulting from the formula specified in gem attribute
*I.e.: (gem="(10**((-0.159)*(z/9.81)/1000)))"*
dem is the dem value as read from the PCRaster map

Be aware that if your dem map has directions values, those will be replicated in the final map.

Which geopotential file will be read?

Geopotential GRIB files are under configuration/geopotentials folder and their filenames are configured in geopotentials.xml file. The application will select the right file according the geodetic attributes of the input GRIB file.

Those attributes compose the key *id* in the XML configuration:
Note the $ delimiter.

```
longitudeOfFirstGridPointInDegrees$longitudeOfLastGridPointInDegrees$Ni$Nj$numberOfValues$gridType
```

If you want to add another geopotential file to the configuration, just execute the command:

```
./pyg2p.py -g /path/to/geopotential/grib/file
```

The application will copy the geopotential GRIB file into *configuration/geopotentials* folder and will also add the proper XML configuration to geopotentials.xml file.

Notes about formulas

Previous correction formulas were derived directly from grib2pcraster C code. The need to set up two formulas – one for the gem value and one for the final corrected value (which uses the previously formulated gem value) – exists because the correction formulas are based on DEM and geopotential values. DEM values are in PCRaster format while geopotential data is a GRIB message, which needs to be interpolated before to put it into the correction formula. That means that the correction process is applied in two steps with a split formula as configured in XML.

Grib2pcraster correction process:

In this paragraph, some C code is shown from the original grib2pcraster C application.

Geopotential value is read from a grib file as (note the correction):

```
//correction formula is applied here (it's the gem in pyg2p)
if (intCorrectionFlag == 1)//temperature correction
	{
		for (i = 0; i < values_len; i++) {
			pgeopotential[i] = (pgeopotential[i] / 9.81) * 0.0065;
		}
```

```
        } else if (intCorrectionFlag == 2)//evap correction
        {
                for (i = 0; i < values_len; i++) {
                        pgeopotential[i] = pgeopotential[i] / 9.81;
                        pgeopotential[i] = pow(10, ((-0.159) * (pgeopotential[i]) / 1000));
                }
        }
```

Then, the actual correction is made during the interpolation process (some not relevant code was removed and comments added to the real code):

```
//interpolation
float tmpValue = (pGRIBValues[grid[i].indexInGRIBArrNode1]) * (grid[i].CoefficientNode1) +
(pGRIBValues[grid[i].indexInGRIBArrNode2]) * (grid[i].CoefficientNode2) +
(pGRIBValues[grid[i].indexInGRIBArrNode3]) * (grid[i].CoefficientNode3)
                                + (pGRIBValues[grid[i].indexInGRIBArrNode4]) *
(grid[i].CoefficientNode4);


if (intCorrectionFlag != 0) {
  //interpolation of geopotential as read before
  float tmpGeo = (geopotential[grid[i].indexInGRIBArrNode1]) * (grid[i].CoefficientNode1) +
(geopotential[grid[i].indexInGRIBArrNode2]) * (grid[i].CoefficientNode2) +
(geopotential[grid[i].indexInGRIBArrNode3]) * (grid[i].CoefficientNode3)
+ (geopotential[grid[i].indexInGRIBArrNode4]) * (grid[i].CoefficientNode4);

   //correction formula is applied here (the correctionFormula in pyg2p)
   if (intCorrectionFlag == 1)//temperature correction
                tmpValue = (tmpValue + tmpGeo) - (grid[i].demValue * 0.0065);
   if (intCorrectionFlag == 2)//evap correction
                  tmpValue = (tmpValue / tmpGeo) * (pow(10, ((-0.159) * (grid[i].demValue) /
1000)));
}

//store value in the output array
pPCRasterValues[grid[i].indexInPCRasterArr] = tmpValue;
```

pyg2p correction process

The process in pyg2p is almost the same, that means:

- read geopotential from grib, and apply the correction formula as configured in XML (gem="(z/9.81)*0.0065")
- interpolate the gem values so obtained to the PCRaster output grid
- read grib values and interpolate them
- apply the final correction formula as configured (correctionFormula="p+gem-dem*0.0065")

The only difference is that in pyg2p formulas are configurable, as explained in this chapter (read Configuration manual as well), while in grib2pcraster you can choose, with a binary flag in parameters.xml file, between two hard coded formulas (one for temperature and one for evaporation).

Conversion

Values from GRIB files can be converted before to write final output maps. Conversions are configured in the *parameters.xml* file, on a parameter basis. The right conversion formula will be selected using the id specified in the *applyConversion* attribute, and the *shortName* attribute of the parameter that is going to be extracted and you want to convert.

In the *Parameter* tag of the XML template you will configure only the id of the conversion:

```
<Parameter shortName="2t" tstart="0" tend="12" applyConversion="k2c"
            correctionFormula="p+gem*0.0065-dem*0.0065"
                  gem = "(z/9.81)*0.0065"
            demMap="/dataset/maps/efasLatLon/europe5km/dem.map"
/>
```

The conversion formula is configured in *parameters.xml* configuration file, under the relative parameter configuration, identified by *shortName*:

```
<Parameter shortName="2t" description="2 meters Temperature" unit="K">
  <Conversion id="k2c" unit="C" function="x=x-273.15" cutOffNegative="False"/>
</Parameter>
```

Note that formulas must be written in the form *var=f(var)*.

The configuration file parameters.xml is needed only for conversion functionality. Tags *description* and *unit*s serve for logging purposes.

Logging

File-loggers are configured in *loggers-configuration.xml* (see relative chapter in the configuration manual). File writing can be disabled using the following configuration for the root XML *Loggers* element:

```
<Loggers enabled="false">
```

You can use any string between false, False, no, NO, No to deactivate file logging. Any other string is evaluated as True. The element is optional and default value is True.

The loggers directory is *./logs* by default. This can be changed by using the **-d** option (or **–outLogDir**).

Console logger level is INFO by default and can be optionally set by using **-l** (or **–loggerLevel**) input argument.

Possible logger level values are ERROR, INFO, DEBUG, in increasing order of verbosity .

The output console will be logged in the file *pyg2p_out.log* as well.

Comparing Test Suite

Since version 1.1, pyg2p comes with a little test utility which helps to compare a new version release with the grib2pcraster C application. This functionality will produce and check PCRaster *diff* maps, and print aguila commands on screen; it is being activated with the command:

```
./pyg2p.py -t test.xml
```

The program will warn if the number of produced maps differs for a certain test.

For convenience, the output will present the aguila commands for visualizing grib2pcraster, pyg2p and diff maps.

Sample output:

```
====> Producing pcraster diff maps. Copy and paste aguila commands to compare them.
aguila /dataset/testdiffmaps/cosmoT24/diff.013 /dataset/testdiffmaps/cosmoT24/gT240000.013
/dataset/testdiffmaps/cosmoT24/pT240000.013
[GOOD] values are good!
...
```

Writing test.xml file

In *test.xml* file, you configure:

- path to **grib2pcraster** executable
- path to a *commands.txt* file containing the list of input arguments for both **grib2pcraster** and **pyg2p**
- path to the **pcrcalc** executable, for producing *diff* maps
- the error tolerance when checking diff maps

An example of *text.xml*:

```
<TestConfiguration atol="1e-02"
commands="/home/dominik/ECMWF_grib2pcraster/pyg2p_workingcopy/configuration/tests/commands.txt">
   <g2p exec="/home/dominik/ECMWF_grib2pcraster/grib2pcraster/grib2pcraster" />

   <!--pcrcalc diff.001 = mapEFAS.001 - mappyg2p.001-->
   <PcRasterDiff exec="/home/dominik/devapps/PCRaster-3.0/bin/pcrcalc"/>
</TestConfiguration>
```

An example of *commands.txt* file:

```
g1@ -p 228.128 -n gR24 -t 24 --fmap=1 -i EpsN320-2013063000.grb -o ./eueR24 -m 10
p1@ -l ERROR -c eue_r24.xml -i /EpsN320-2013063000.grb -o ./eueR24 -m 10

g2@ -p 182.128 -n gE24 -t 24 --fmap=1 -m 10 -i EpsN320-2013063000.grb -o ./eueE24
p2@ -l ERROR -c eue_e24.xml -i EpsN320-2013063000.grb -o ./eueE24 -m 10

#glofas commands. grib2pcraster needs two executions for each pyg2p command
g3@ -i 20130325_en0to10.grib -o ./multiresSRO -p 8.128 -m 50 -n gsro --date=20130325 -s 0
-e 240 --unit=mm -t 24 --interlookup=intertable_2011_medium.txt
g3@ -i 20130325_en11to15.grib -o ./multiresSRO -p 8.128 -m 50 -n gsro --date=20130325 -s
264 --fmap=11 -e 360 --unit=mm -t 24 --interlookup=intertable_2011_low.txt
p3@ -c glofas_sro.xml -i 20130325_en0to10.grib -I 20130325_en11to15.grib -o ./multiresSRO
-m 50

#only pyg2p tests (these are not reproducible with grib2pcraster 1.07)
p4@ -i UKMO_20121218.grib -o ./UKMO/ -c UKMO_t24_LA.xml -m 0
p5@ -i EUE_2013073100_cv.grb -o ./UKMO/ -c EUE_RainAnim_CV.xml -m 0
#scipy comparison tests
p6@ -i UKMO_20121218.grib -o ./UKMO/ -c UKMO_t24_LA.xml -m 0
z6@ -i EUE_2013073100_cv.grb -o ./UKMO/ -c EUE_RainAnim_CV_scipy.xml -m 0
```

See the following paragraph or the Configuration manual for further details.

Writing the commands.txt file

When writing the commands.txt file, please follow these rules:

- Each test is composed by both the list of arguments for **grib2pcraster** and **pyg2p**. Prefix each list of arguments with 'g' (for **grib2pcraster**) and 'p' (for **pyg2p**) (or 'p' for **pyg2p with grib api interpolation** and 'z' for **pyg2p with scipy interpolation** or any alternative execution configuration to compare the two)
- and use the same id to tie them.
    - If in a test case you need to compare two executions of grib2pcraster with one of pyg2p (i.e. for multiresolution grib files), just write the two list of grib2pcraster arguments using the same id.
    - If in a test case you want to execute only the pyg2p command, just write the respective list

of argument as pN@ .

- Output directories of each test must be the same for the two executions. This limitation happens because of how the **pcrcalc** command works.
- Maps prefixes of each test must be 'g' for **grib2pcraster** output maps (or pyg2p alternative output maps - e.g. with scipy interpolation) and 'p' for **pyg2p** output maps (to configure in xml).
- Don't use –report=2 option for grib2pcraster commands and use -l ERROR for pyg2p commands to avoid pollution of output console.

Sample Output

```
 ======================> Running Test 5
        - pyg2p comm -l ERROR -c dwd_e06.xml -i dwd_grib1_ispra_LME_2012111900 -o
./dwdE06
        - g2p comms [u'/grib2pcraster/grib2pcraster  -p 121.2 -n gE06 -t 06 --fmap=1 -s 12 -e 18 -i
dwd_grib1_ispra_LME_2012111900 -o ./dwdE06']
        - out dir ./dwdE06/

Running grib2pcraster...
Running pyg2p...

====> Producing pcraster diff maps. Copy and paste aguila commands to compare them.
aguila ./dwdE06/diff.002 ./dwdE06/gE060000.002 ./dwdE06/pE060000.002
[GOOD] values are good!
aguila ./dwdE06/diff.001./dwdE06/gE060000.001 ./dwdE06/pE060000.001
[GOOD] values are good!

=========== SUMMARY ==============
pyg2p test 5 executed in 0:00:01.005321
pyg2p memory usage: max 205.64MB, avg 117.13MB
grib2pcraster test 5 executed in 0:00:00.586653
Difference: 0:00:00.418668
============== END ================
```

pyg2p API

From version 1.3, pyg2p comes with a simple API to import and use from other python programs (e.g. pyEfas).

The pyg2p API is intended to mimic the pyg2p.py script execution from command line so it provides a Command class with methods to set input parameters and a *run_command(cmd)* module level function to execute pyg2p.

Setting execution parameters

First, create a pyg2p command:

```
>>> import pyg2p
>>> command = pyg2p.command()
```

Then, set up the execution parameters using a chain of methods or single calls:

```
>>>command.with_cmdpath('a.xml')
>>>command.with_inputfile('0.grb')
>>>command.with_log_level('ERROR')
>>>command.with_outdir('/dataout/').with_tstart('6').with_tend('24').with_eps('10').with_fmap('1')
>>>command..with_ext('4')
>>> print(str(command))
pyg2p.py -c a.xml -e 240 -f 1 -i 0.grb -l ERROR -m 10 -o /dataout/test -s 6 -x 4
```

You can also create a command object using the input arguments as you would do when execute pyg2p from shell:

```
>>>args_string = '-l ERROR -c /pyg2p_git/execution_templates_devel/eue_t24.xml -i
/dataset/test_2013330702/EpsN320-2013063000.grb -o /dataset/testdiffmaps/eueT24 -m 10'
 >>>command2 = pyg2p.command(args_string)
```

Execute

Use the *run_command* function from pyg2p module. This will delegate the main method, without shell execution.

```
>>>ret = pyg2p.run_command(command)
```

The function returns the same value pyg2p returns if executed from shell (0 for correct executions, included those for which messages are not found).

Adding geopotential file to configuration

You can add a geopotential file to configuration from pyg2p API as well, using the *addGeo* function:

```
>>>pyg2p.addGeo('/dataset/maps/fredrik/T3999.gph.grb')
```

The result will be the same of ./pyg2p -g /dataset/maps/fredrik/T3999.gph.grb

Appendix A - Execution XML files Examples

This paragraph will explain typical execution xml configurations.

Example 1: Correction with dem and geopotentials

./pyg2p.py -c example1.xml -i /dataset/cosmo/2012111912_pf2_t2.grb -o ./out_1

```
<Execution name = "cosmo5km_2t_corrected_invdist" id="1">
      <Parameter shortName="2t" tstart="0" tend="12"  applyConversion="k2c"
                  gem="(z/9.81)*0.0065"
                  correctionFormula="p+gem-dem*0.0065"
                  demMap="/dataset/maps/cosmo/dem05.map" />
      <OutMaps namePrefix="t2" fmap="001"
            cloneMap="/dataset/maps/efasLatLon/europe5km/dem.map" >
        <Interpolation mode="grib_invdist"
                    latMap="/dataset/maps/cosmo/cosmo05_lat.map"
                    lonMap="/dataset/maps/cosmo/cosmo05_lon.map" />
      </OutMaps>
   </Execution>
```

This configuration, will select the 2t parameter from time step 0 to 12, out of a cosmo t2 file. Values will be corrected using the dem map and a geopotential file as in geopotentials.xml configuration.

You can use another well known correction whose formulas are configured as below:

```
gem="(z/9.81)*0.0065"
correctionFormula="p+gem-dem*0.0065"
```

Maps will be written under ./out_1 folder (the folder will be created if not existing yet). The clone map is set as same as dem.map.

The original values will be converted using the conversion “k2c”. This conversion must be configured in the parameters.xml file for the variable which is being extracted (2t). See Parameter XML tag configuration paragraph in Configuration Manual or in the Conversionchapter in this manual.

The interpolation method is *grib_invdist*. For grib interpolation methods (*grib_nearest* and *grib_invdist*), no additional parameters are needed. Latitudes and longitudes values will be used only if the interpolation lookup table (intertable) hasn't be created yet but it's mandatory to set latMap and lonMap because the application uses their metadata raster attributes to select the right intertable.

The table filename to be read and used for interpolation is automatically found by the application, so there is no need to specify it in configuration. However, lat and lon maps are mandatory configuration attributes.

Example 2: Dealing with multiresolution files

./pyg2p.py -c example1.xml -i 20130325_en0to10.grib -I 20130325_en11to15.grib -o ./out_2

Performs accumulation 24 hours out of sro values of two input grib files having different vertical resolutions. You can also feed pyg2p with a single multiresolution file.

./pyg2p.py -c example1.xml -i 20130325_sro_0to15.grib o ./out_2 -m 0

```
<Execution name = "test_multi_glob_inv" id="6">
      <Aggregation step="24" type="accumulation" />
<Parameter shortName="sro" tstart="192" tend="360"
         applyConversion="m2mm" />

   <OutMaps namePrefix="sro" fmap="001" cloneMap="/dataset/maps/global/dem.map">
      <Interpolation mode="grib_nearest" latMap="/dataset/maps/global/lat.map"
                lonMap="/dataset/maps/global/lon.map" />
   </OutMaps>
</Execution>
```

This execution configuration will extract global overlapping messages sro (perturbation number 0) from two files at different resolution.

Values will be converted using “tomm” conversion and maps (interpolation used here is grib_nearest) will be written under ./out_6 folder.

Example 3

./pyg2p.py -c example1.xml -i dwd_grib_ispra_gme_2006_2_168_i384_2012111500.grib2 -o ./out_13

```
<Execution name = "dwd_grib2_10u" id="13" >

      <Parameter shortName="10u" tstart="12" tend="15" />

      <OutMaps namePrefix="10u" fmap="001" cloneMap="/dataset/maps/dem05.map">
         <Interpolation mode="grib_nearest"  latMap="/dataset/maps/dwd/dwdgmi05_lat.map"
                lonMap="/dataset/maps/dwd/dwdgmi05_lon.map" />
```

```
    </OutMaps>
  </Execution>
```

Performing this execution will extract 10u messages (from timestep 12 to 15) from a dwd GRIB2 file.
For resampling, dwd gmi maps file are used.
Again, if there is a precompiled interlookup table, this will be used instead of performing an expensive resampling using GRIB API nearest functions.

Example 4: Average 24 hours

./pyg2p.py -c execution_file_examples/execution_6.xml -o ./out_6 -i /dataset/dwd/grib/dwd_grib1_ispra_LME_2012111900

```
<?xml version="1.0" encoding="utf-8"?>

  <Execution name = "dwd_T24a" id="6" >
    <Parameter shortName="2t" tstart="30" tend="35" applyConversion="k2c"/>
    <Aggregation type="average" step="24" />

    <OutMaps outDir="./out/dwd/T24a" namePrefix="T24a" fmap="001"
          cloneMap="/dataset/maps/dwd/dem05.map" unitTime="24">
       <Interpolation mode="grib_nearest" intertableDir="./myTables"
                latMap="/dataset/maps/dwd/dwdlme05_lat.map"
                lonMap="/dataset/maps/dwd/dwdlme05_lon.map" />
    </OutMaps>
  </Execution>
```

Note the intertableDir specifing an alternative folder where to read and store interlookup tables.

Example 5: Accumulation 24 hours

./pyg2p.py -i /dataset/eue/EpsN320-2012112000.grb -o ./out_eue -c execution_file_examples/execution_9.xml

```
<?xml version="1.0" encoding="utf-8"?>

  <Execution name = "eue_tp" id="9" >
    <Parameter shortName="tp" tstart="0" tend="240" applyConversion="tomm"
perturbationNumber="50"/>
    <Aggregation type="accumulation" step="24" />

    <OutMaps namePrefix="R24a" fmap="1"
          cloneMap="/dataset/maps/europe5km/dem.map" unitTime="24">
```

```
      <Interpolation mode="grib_invdist"
                latMap="/dataset/maps/europe5km/lat.map"
                lonMap="/dataset/maps/europe5km/long.map" />
   </OutMaps>
</Execution>>
</Execution>
```

Appendix B – mapping between grib2pcraster and pyg2p

This appendix aims to help to migrate from grib2pcraster to pyg2p.

Differences in glossary

- The Manipulation as intended in grib2pcraster becomes Aggregation in pyg2c; if you read the term manipulation in the pyg2c documentation, that means a more generic term, including correction and conversion. Anyway, if you read the python code, the python class which performs aggregation is called Manipulator itself.
- What it's called TypeOfField in grib2pcraster becomes stepType in pyg2p (as for the GRIB standard). It defines the aggregated nature of the variable which can be instant, averaged, or cumulated.
- In grib2pcraster, the variable to extract is called parameter and it's referenced by a couple of numbers: **x.y**, where:
  - **x** is the value of the *indicatorOfParameter* key as found in the GRIB message
  - **y** is an internal number identifying the TableVersion as found in the parameters.xml file of grib2pcraster.
- In pyg2p, while the Parameter term was retained, it is identified by the *shortName* key as found in the GRIB message. This is an independent version key and it's only related to the variable itself and not to the *generating centre* of the file, since this is already coded in GRIB API tables. That is, in pyg2p the concepts of generating centre and table parameter versions don't need to exist.

Mapping variables

To “translate” the parameter code **x.y** used in grib2pcraster to the ***shortName*** form of pyg2c, just follow these simple steps:

1. Identify the parameter in grib2pcraster's parameters.xml file using **y** to find the *gribTablesVersionNo* and **x** to find the single item *Param*.
2. Take note of abbreviation and field description strings of the Param above.

3. Use some basic GRIB tools (and meteorological) skills to find the ***shortName*** value of the grib:

```
grib_get -p shortName,parameterName /path/to/grib/file.grib2
```

Known parameters mapping with conversions

Variable	grib2pcraster selectors (indicatorOfParameter.gribTablesVersionNo)	pyg2p.py shortName	Configured conversions in grib2pcraster
2 meters Dew-point	• 17.2 • 168.128	• 2D	• x-273.15
2 meters Temperature	• 11.2 • 500011.0 • 167.128	• t_2m • td_2m • 2t	• x-273.15
Total precipitation	• 61.2 • 228.128	• tp	• Only cut off • x*1000 + cut off
Latent heat flux	• 500086.0 • 121.2	• alhfl_s	• **x*(-0.0353) + cut off [default]** • x*(-0.00147) + cut off
Surface Latent heat flux	• 147.128	• slhf	No conversion
Surface Runoff	• 8.230 • 8.128	• sro	• x*1000 + cut off
Sub-Surface Runoff	• 9.230 • 9.128	• ssro	• x*1000 + cut off
Large scale precipitation	• 142.128	• lsp	• x*1000 + cut off
Convective precipitation	• 143.128	*Not tested. It needs to be added to parameters.xml if used.*	• x*1000 + cut off
Evaporation	• 182.128	• e	• x*(-1000)
Runoff	• 205.128	*Not tested. It needs to be added to parameters.xml if used.*	• x*1000 + cut off *(not as default, it needs to be set by command line with --unit parameter)*
Surface precipitation amount, rain, grid scale	• 102.201	• rain_gsp	• Only cut off
Surface precipitation amount, rain, convective	• 113.201	• rain_con	• Only cut off
Large scale rain rate	• 500134.0	*Not tested. It needs to be added to parameters.xml if used.*	• Only cut off

Snow fall	• 144.128	*Not tested. It needs to be added to parameters.xml if used.*	• x*1000
Snow depth	• 141.128	*Not tested. It needs to be added to parameters.xml if used.*	• X*1000 (not as default, it needs to be set by command line with --unit parameter)

Execution parameters mapping

Argument	**grib2pcraster**	**pyg2p.py**
Input file	-i	-i
Variable	-p x.y	Xml[6] (shortName)
Output directory	-o	-o
EPS perturbationNumber	-m	-m
Variable level	--level	Xml
Aggregation type	Default in parameters.xml. Overwriting with --ma	Xml (type)
Aggregation step	-t	Xml (step)
Start step	-s	-s or Xml (tstart)
End step	-e	-e or Xml (tend)
Output map unittime	--unittime	Xml (unitTime)
Interpolation	-r,-l,-a or –intertable	Xml (see relative section in this manual).
Conversion	parameters.xml	parameters.xml and Xml (applyConversion)
Correction	Two formulas hardcoded and configured by default in parameters.xml or with --temp and --evap.	Xml (attributes correctionFormula, gem, dem)
Map extension naming	--fmap, --ext	Xml
Map prefix name	-n	Xml
Logging	--report. Loggers directory is under program installation folder	Configurable in logger-configuration.xml. Console logger level overwritable with -l (default is INFO). Default loggers dir ./logs overwritable with -d.

6 In this table, “Xml” means that the corresponding parameter configuration is in the execution xml template where you configure your extraction and pass to the pyg2p with the -c input argument. If not indicated otherwise, the xml tag/attribute name is the same as the grib2pcraster input argument (e.g.: --level=10 becomes the xml attribute level=”10”).

Difference in dealing with multiresolution

Multiresolution extraction can be made in one step with pyg2p, using both -i and -I for input files or even a single multiresolution file. The grib2pcraster application needs to perform two executions, instead.

Differences in Correction

There is a sort of bug in grib2pcraster when come to correction, since the dem values it uses are stored in its interlookup tables and, if the DEM has changed for some reasons, the corrected values will be wrong unless the interlookup table is not regenerated using the new DEM map.

However, here the main difference is that the correction formulas are editable directly by the user in pyg2p.

Differences in Aggregation

The type of aggregation and its step window must be specified in pyg2p, using the ***Aggregation*** XML tag.

In grib2pcraster, instead, the type of aggregation for the specific variable is being extract (i.e. manipulation) is read from *ManipulationFlag* attribute in *parameters.xml* file: “1” for average and “2” for accumulation. The step window is 24 hours by default and it can be overwritten with -t parameter.

Differences in Interpolation

Resampling of grids in **grib2pcraster** is made using GRIB API. Available interpolation methods are:

- nearest neighbours
- inverse distance

The interpolation table file is declared with the input argument –intertable, or otherwise by configuration, using one of the tables in *interpolation.xml*.

In **pyg2p** there are two additional resampling modes (see the relative Interpolation modes paragraph in this document). Results are very similar to GRIB_API based interpolation methods but interlookup table is being created in seconds rather than hours (in our tests, interlookup table creation for high resolution or global scale grids take almost two days with GRIB_API methods while it takes one minute with scipy based methods).

They are *nearest* and *invdist* modes, performed using scipy routines to find neighbours and distances.

Interlookup tables are read/written by default using $PYG2PHOME/configuration/intertables as folder where to load existing intertables or save new ones. An alternative folder can be specified using the intertableDir attribute in the Interpolation XML tag of the xml template configuration.

Appendix C – Glossary

Brief description of terms you find in this manual, with their synonims.

Term	Synonims	Description
Interpolation lookup table	Intertable, interlookup table	It's a numpy binary file containing indexes correspondences and coefficients/distances. Each combination of **GRIB grid**/PCRaster target coordinates maps/Interpolation mode has its own intertable. Since there are four interpolation modes using intertables, there can be four different intertables for each combination of **GRIB grid**/PCRaster lat/lon maps.
Aggregation		Average or accumulation
Interpolation	Resampling	Resampling between the original **GRIB grid** and the target raster lat/lon coordinates.
Execution XML template file	Execution template, commands file, execution file, xml template, xml configuration	The XML file where most of **execution parameters** are configured. It's being passed by the mandatory CLI argument ***-c***.
Parameter	Variable	The meteorological variable stored in the input grib file.
Command line Input argument	Input argument, CLI argument, option	Input arguments to the application passed via command line. They are ***-i***,*-I*,*-o*,*-m*,*-l*,*-d*,***-c***
Execution parameter	Parameter, input argument	All configured options (via CLI or XML files) that define the execution context.
GRIB grid	Grid	It's the set of geodetic attributes identifying the coordinate system and the specific projected area. The set includes **gridType**, first and last values of latitude and longitudes, number of points along the meridian.
GRIB message's gridType	Grid type, grid	It's the specific grid type of the **GRIB message** and can be regular_rr, regular_ll, rotated_rr, rotated_ll, reduced_ll and so on.
GRIB message	Grib	It's a single message extracted from a **GRIB file**.
GRIB API		C library (and its wrappers) from ECMWF to deal with **grib files**.
GRIB file	Grib	An input file encoded using GRIB format.
Geopotential GRIB values	Geopotential	Geopotential values encoded in grib files, used for correction.
Gem value	Gem	In correction, the **gem value** is being obtained using the formula configured with *gem* XML attribute, where *z* is the geopotential value. E.g. gem = “(z/9.81)*0.0065”