pyg2p 2.0

Version	2.0
Author	Domenico Nappo (domenico.nappo(at)gmail.com)
Release Date	2016-01

Table of Contents

Introduction

pyg2p is a tool to convert GRIB files into PCRASTER maps.

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 JSON template files.

Manipulations are completely configurable in execution JSON files and are:

- conversion (formulas are to configure in parameters.json)
- 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).

Installation

With pip tool:

In a python2.7 virtualenv, execute

```
pip install git+https://bitbucket.org/nappodo/pyg2p.git
```

```
Note: Your bitbucket account must have read access to repository.
```

Pip will download python package and install dependencies.

Manual

1. Download last version from bitbucket.org (you have to grant permissions to repository): https://bitbucket.org/nappodo/pyg2p/get/<version>.tar.gz
2. Decompress file in <target directory> using `tar xzvf pyg2p_v.<version>.tar.gz`.
3. Install dependencies (see table below).

Some packages need additional requirements such as C libraries properly installed.

Software Requirement	
Python 2.7	Execution environment

Python packages	
NumPy>=1.10.1	Scientific packages for array operations and reading/writing encoded files. They depends on several C libraries that must exist on your system before to proceed. `$pip install numpy>=1.10.1` `$pip install numexpr` `$pip install scipy>=0.16.0` `$pip install GDAL` `* You may have troubles while installing right GDAL python package. If you have GDAL C/C++ libraries already installed on your system, you might need to install GDAL python matching the C GDAL version.` `** For gribapi python package, please refer to GRIB API documentation.`
Numexpr	
SciPy>=0.16	
gribapi*	
GDAL python**	
dask[array] dask[bag] toolz	Light packages for parallelization (used in interpolation): `$pip install toolz` `$pip install dask[array]` `$pip install dask[bag]`
xmljson (optional)	Used to convert old XML pyg2p v1 to v2 JSON format (-C command line argument): `$pip install xmljson`
memory_profiler (optional)	Memory usage information for test suite (for development). `$pip install memory_profiler`

Migrating from v 1.x

If you migrate from a pyg2p version 1.x, you can convert existing XML configuration files to json (execution command files, parameters.xml and geopotentials.xml) and also convert intertables (only grib api methods)

1. Copy default configuration (from sources) to your .pyg2p/ user folder, with command:
   ```
   $pyg2p.py -P
   ```

2. Copy all your old XML configuration into a folder ./xml_backup. You can create different subfolders for convenience because conversion will be recursive.

3. Convert files (recursively) from XML to JSON with command:
   ```
   $pyg2p.py -C ./xml_backup
   ```

4. Copy parameters.json and geopotentials.json from ./xml_backup into user folder ~/.pyg2p/
5. Copy your geopotential files to ~/.pyg2p/geopotentials/ folder.

You can also convert your existing interpolation tables made with pyg2p version 1.x GRIB_API methods to be compliant with new format. New files will be copied under ~/.pyg2p/intertables/ folder and intertables.json configuration will be updated during conversion.

```
$pyg2p.py -z <path_to_convert>
```

```
Note: Intertables for rotated grids are not valid so they won't be converted.
```

Intertables created with scipy methods cannot be converted but they are generally quite fast to create (minutes).

Configuration

There are several things to configure in pyg2p but default configuration is enough for most of conversion tasks. Since version 2.x, pyg2p read and stores configuration from/in user folder ~/.pyg2p/.

Intertables (for interpolation) are read/write from ~/.pyg2p/intertables/ and geopotentials (for correction) are read from ~/.pyg2p/geopotentials/.
To copy default pyg2p configuration into user configuration folder execute this command:

```
$pyg2p -P
```

The command will copy main configuration and some existing geopotentials file for correction steps.

You have to download default intertables and copy into ~/.pyg2p/intertables from ECMWF ftp server.

Advanced configuration

Adding a parameter to parameters.json

If you are extracting a parameter with shortName xy from a grib file that is not configured into ~/.pyg2p/parameters.json file, you need to add an element (a dictionary) to “Parameter” list:

```
{
    "@description": "Variable description",
    "@shortName": "xy",
    "@unit": "unit...",
    "Conversion": [

        {
            "@cutOffNegative": true,
            "@function": "x=x*2",
            "@id": "conversion_id",
            "@unit": "unit..."
        }
    ]
},
```

You can add more than a conversion element with different ids and functions. You will use shortname and conversion id for the execution JSON template.

```
Note: Aware the syntax of conversion functions. They must start with x= followed
by the actual conversion formula where x is the value to convert. Units are only
used for logging.
```

Adding a geopotential for correction

If the input grib file has a geopotential message, pyg2p will use it for correction. Otherwise, it will read the file configured into ~/.pyg2p/geopotentials.json.

To add a geopotential GRIB file to pyg2p configuration, use this command:

```
$pyg2p -g path_to_geopotential_grib
```

This will copy the file to ~/.pyg2p/geopotentials/ folder and will update geopotentials.json with the new item.

Interpolation tables

Interpolation tables are read from *~/.pyg2p/intertables* folder.

If the table is missing, it will create it into same folder for future interpolations.

Depending on source and target grids size, and on interpolation method, table creation can take from minutes to days. To speed up interpolation table creation, use parallel option -X to have up to x6 speed gain.

Execution templates

Execution templates are JSON files that you will use to configure a conversion. You will pass path to the file to pyg2p with command line option'-c'.

Most of options can be both defined in this JSON file and from command line. Note that command line options overwrite JSON template.

If you have a large set of conversions for same parameters, where you change start step, end step and perturbation number parameter filters, you would use one JSON file to define parameter, interpolation and correction and you can change other parameters from command line.

Here some examples of JSON commands files:

```
    {
    "Execution": {
        "@name": "Octahedral test 1",
        "Aggregation": {
            "@step": 24,
            "@type": "average"
        },
        "OutMaps": {
            "@cloneMap": "/dataset/maps/europe5km/dem.map",
            "@ext": 1,
            "@fmap": 1,
            "@namePrefix": "t2",
            "@unitTime": 24,
            "Interpolation": {
                "@latMap": "/dataset/maps/europe5km/lat.map",
                "@lonMap": "/dataset/maps/europe5km/long.map",
                "@mode": "grib_nearest"
            }
        },
        "Parameter": {
            "@applyConversion": "k2c",
            "@correctionFormula": "p+gem-dem*0.0065",
            "@demMap":  "/dataset/maps/europe5km/dem.map",
            "@gem": "(z/9.81)*0.0065",
            "@shortName": "2t"
        }
    }
}
```

There are four sections of configuration.

Aggregation

Defines the aggregation method and step. Method can be *accumulation*, *average* or *instantaneous*.

OutMaps

Here you define interpolation method and paths to coordinates PCRaster maps, output unit time, the clone map etc.

Interpolation

This is a subelement of OutMaps. Here you define interpolation method (see later for details), paths to coordinates maps.

Parameter

In this section, you configure the parameter to select by using its shortName, as stored in GRIB file. You also configure conversion with *applyConversion* property set to a conversion id. Parameter *shortName* must be already configured in ~/.pyg2p/parameters.json along with conversion ids.

If you need to apply correction based on DEM files and geopotentials, you can configure formulas and the path to DEM map.

Path configuration

You can use variables in JSON files to define paths. Variables can be configured in .conf files under ~/.pyg2p/ folder.

/home/domenico/.pyg2p/myconf.conf

```
EUROPE_MAPS=/dataset/maps/europe5km
DEM_MAP=/dataset/maps/dem05.map
EUROPE_DEM=/dataset/maps/europe/dem.map
EUROPE_LAT=/dataset/maps/europe/lat.map
EUROPE_LON=/dataset/maps/europe/long.map
```

Usage in JSON command file:

```
   {  "Execution": {
          "@name": "eue_t24",
          "Aggregation": {
               "@step": 24,
               "@type": "average"
          },
          "OutMaps": {
               "@cloneMap": "{EUROPE_MAPS}/lat.map",
               "@ext": 1,
               "@fmap": 1,
               "@namePrefix": "pT24",
               "@unitTime": 24,
               "Interpolation": {
                    "@latMap": "{EUROPE_MAPS}/lat.map",
                    "@lonMap": "{EUROPE_MAPS}/long.map",
                    "@mode": "grib_nearest"
               }
          },
          "Parameter": {
               "@applyConversion": "k2c",
               "@correctionFormula": "p+gem-dem*0.0065",
               "@demMap": "{DEM_MAP}",
               "@gem": "(z/9.81)*0.0065",
               "@shortName": "2t"
          }
     }
}
```

Note that here you use variables DEM_MAP and EUROPE_MAPS as paths. You can set these variables in a .conf file in ~/.pyg2p/ folder.

Full list of options

Main structure

Attribute/Tag	Details
name	The short name of the parameter, as it is in the grib file. The application use this to select messages.
Parameter	See relative table
Aggregation	See relative table
OutMaps	See relative table

Parameter property

Attribute/Tag	Details
shortName	The parameter short name, as it is in the grib file. Must be configured in the parameters.json file, otherwise the application exits with an error. Main grib selector.
tstart	Optional grib selectors perturbationNumber, tstart, tend, dataDate and dataTime can also be issued via command line arguments (-m, -s, -e, -D, -T), which overwrite the ones in the execution JSON file.
tend	
perturbationNumber	
dataDate	
dataTime	
level	
applyConversion	The conversion id to apply, as in the parameters.json file for the parameter to select. The combination parameter/conversion must be properly configured in parmaters.json file, otherwise the application exits with an error.
correctionFormula[1]	Formula to use for parameter correction with p, gem, dem variables, representing parameter

1 When configuring correction, all properties *gem*, *correctionFormula*, and *demMap* must be present.

	value, converted geopotential to gem, and DEM map value. E.g.: p+gem*0.0065-dem*0.0065
demMap	The dem map used for correction.
gem	Formula for geopotential conversion for correction.

OutMaps property

Attribute/Tag	Details
cloneMap	The clone map with area (must have a REAL cell type and missing values for points outside area of interest. A dem map works fine. A typical area boolean map will not).
unitTime	Time unit in hours of output maps. Tipical value is 24 (daily maps).
namePrefix	Prefix for maps. Default is parameter *shortName.*
fmap	First map number. Default 1.
Interpolation	See relative table.
ext	Extension mode. It's the integer number defining the step numbers to skip when writing maps. Same as old grib2pcraster. Default 1. Refers to User Manual for further information.

Aggregation property

Attribute/Tag	Details
step	Step of aggregation in hours.
type	Type of aggregation (it was Manipulation in grib2pcraster). It can be *average* or *accumulation.*
forceZeroArray	*Optional*. In case of “accumulation”, and only then, if this attribute is set to”y” (or any value different from “false”, “False”, “FALSE”, “no”, “NO”, “No”, “0”), the program will use a zero array as message at step 0 to compute the first map, even if the GRIB file has a step 0 message.

Interpolation property

Attribute/Tag	Details
mode	Interpolation mode. Possible values are: “nearest”, “invdist”, “grib_nearest”, “grib_invdist”
latMap	PCRaster map of target latitudes.
lonMap	PCRaster map of target longitudes.
intertableDir	Alternative home folder for interpolation lookup tables, where pyg2p will load/save intertables. Folder must be existing. If not set, pyg2p will use intertables from ~/.pyg2p/intertables/

Usage

To use the application, after the main configuration you need to configure a template JSON file for each type of extraction you need to perform.

Grabbing information from GRIB files.

To configure the application and compile your JSON 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

If you run ./pyg2p.py without arguments, it will show help of all input arguments.

```
usage: pyg2p.py [-h] [-c json_file] [-o out_dir] [-i input_file]
                [-I input_file_2nd] [-s tstart] [-e tend] [-m eps_member]
                [-T data_time] [-D data_date] [-f fmap] [-x extension_step]
                [-n outfiles_prefix] [-l log_level] [-N intertable_dir] [-B]
                [-X] [-t cmds_file] [-g geopotential] [-C path] [-z path] [-P]

Execute the grib to pcraster conversion using parameters from the input json
configuration. Read user and configuration manuals

optional arguments:
  -h, --help            show this help message and exit
  -c json_file, --commandsFile json_file
                        Path to json command file
  -o out_dir, --outDir out_dir
                        Path where output maps will be created.
  -i input_file, --inputFile input_file
                        Path to input grib.
  -I input_file_2nd, --inputFile2 input_file_2nd
                        Path to 2nd resolution input grib.
  -s tstart, --start tstart
                        Grib timestep start. It overwrites the tstart in json
                        execution file.
  -e tend, --end tend   Grib timestep end. It overwrites the tend in json
                        execution file.
  -m eps_member, --perturbationNumber eps_member
                        eps member number
  -T data_time, --dataTime data_time
                        To select messages by dataTime key value
  -D data_date, --dataDate data_date
                        <YYYYMMDD> to select messages by dataDate key value
  -f fmap, --fmap fmap  First map number
  -x extension_step, --ext extension_step
                        Extension number step
  -n outfiles_prefix, --namePrefix outfiles_prefix
                        Prefix name for maps
```

```
 -l log_level, --loggerLevel log_level
                       Console logging level
 -N intertable_dir, --intertableDir intertable_dir
                       interpolation tables dir
 -B, --createIntertable
                       create intertable file
 -X, --interpolationParallel
                       Use parallelization tools to make interpolation
                       faster.If -B option is not passed or intertable
                       already exists it does not have any effect.
 -t cmds_file, --test cmds_file
                       Path to a text file containing list of commands,
                       defining a battery of tests. Then it will create diff
                       pcraster maps and log alerts if differences are higher
                       than a threshold (edit configuration in test.json)
 -g geopotential, --addGeopotential geopotential
                       Add the file to geopotentials.json configuration file,
                       to use for correction. The file will be copied into
                       the right folder (configuration/geopotentials) Note:
                       shortName of geopotential must be "fis" or "z"
 -C path, --convertConf path
                       Convert old xml configuration to new json format
 -z path, --convertIntertables path
                       Convert old pyg2p intertables to new version and copy
                       to user folders
 -P, --copyConf        Copy configuration from source to user folder (except
                       intertables)
```

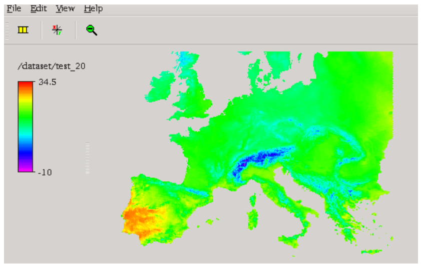
Example usages:

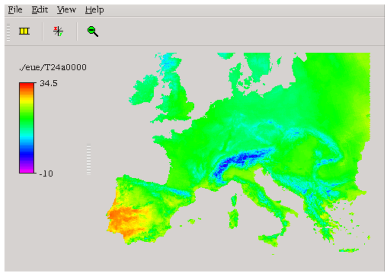
```
./pyg2p.py -c ./exec1.json -i ./input.grib -o /out/dir -s 12 -e 36
./pyg2p.py -c ./exec2.json -i ./input.grib -o /out/dir -m 10 -l INFO
./pyg2p.py -c ./exec3.json -i ./input.grib -I /input2ndres.grib -o /out/dir -m 10 -l DEBUG
./pyg2p.py -g /path/to/geopotential/grib/file  # add geopotential to configuration
./pyg2p.py -t /path/to/test/commands.txt
./pyg2p.py -h
```

Check output maps

After the execution, you can check how output maps were written using the PCRaster[2] 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.

Interpolation modes

Interpolation is configured in JSON execution templates using the *Interpolation* attribute.

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

```
Note: GRIB_API does not implement nearest neighbours routing for rotated grids.
You have to use scipy methods and regular target grids (i.e.: latitudes and
longitudes PCRaster maps).
```

Intertable creation

Interpolation will use precompiled intertables. They will be found in the .pyg2p/intertables folder under user home directory or a custom folder, using the attribute intertableDir within the *Interpolation* attribute or passing -N option on command line.

If interlookup table doesn't exist, the application will create it only if -B option is passed otherwise program will exit.

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 hours or even days for GRIB interpolation methods. To have better performances (up to x6 of gain) you can pass -X option to enable parallel processing.

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

Performances are much better with scipy based interpolation (minutes) but this option could not be viable for all GRIB inputs.

GRIB API interpolation methods

To configure the interpolation method, set the @mode attribute in Execution/OutMaps/Interpolation property.

grib_nearest:

This method uses GRIB API to perform nearest neighbour query.

To configure this method, define:

```
"Interpolation": {
     "@latMap": "/dataset/maps/europe5km/lat.map",
     "@lonMap": "/dataset/maps/europe5km/long.map",
     "@mode": "grib_nearest"
}
```

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": {
     "@latMap": "/dataset/maps/europe5km/lat.map",
     "@lonMap": "/dataset/maps/europe5km/long.map",
     "@mode": "grib_invdist"
}
```

Scipy interpolation methods

nearest:

It's the same nearest neighbour algorithm of grib_nearest but it uses the *scipy kd_tree*[3] module to obtain neighbours and distances.

```
"Interpolation": {
     "@latMap": "/dataset/maps/europe5km/lat.map",
     "@lonMap": "/dataset/maps/europe5km/long.map",
     "@mode": "nearest"
}
```

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

invdist:

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

```
"Interpolation": {
    "@latMap": "/dataset/maps/europe5km/lat.map",
    "@lonMap": "/dataset/maps/europe5km/long.map",
    "@mode": "invdist"
}
```

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

```
Note: Building intertable is much faster with 'nearest' and 'invdist' methods.
These use KDTree scipy package. Methods 'grib_nearest' and 'nearest' produce
identical results in most of cases while there can be some differences between
'grib_invdist' and 'invdist' on 'edges' and around 'peeks'. If you have to use
GRIB_API nearest routines, try with -X command line option to enable parallel
processing. Consider that -X option can be enabled for scipy interpolation as
well.
```

OutMaps configuration

Interpolation is configured under the **outMaps** tag. With additional attributes, you also configure 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.

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 JSON configuration in the execution file will look like:

```
"Aggregation": {
    "@type": "average"
},
```

To better understand what these two types of aggregations do, the DEBUG output of execution is presented later 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 cosmo_t24.json:

```
{
    "Execution": {
        "@name": "cosmo_T24",
        "Aggregation": {
            "@step": 24,
            "@type": "average"
        },
        "OutMaps": {
            "@cloneMap": "/dataset/maps/europe/dem.map",
            "@ext": 4,
            "@fmap": 1,
            "@namePrefix": "pT24",
            "@unitTime": 24,
            "Interpolation": {
                "@latMap": "/dataset/maps/europe/lat.map",
                "@lonMap": "/dataset/maps/europe/lon.map",
                "@mode": "nearest"
            }
        },
        "Parameter": {
            "@applyConversion": "k2c",
            "@correctionFormula": "p+gem-dem*0.0065",
            "@demMap": "/dataset/maps/europe/dem.map",
            "@gem": "(z/9.81)*0.0065",
            "@shortName": "2t"
        }
    }
}
```

Command:

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

ext parameter

In configuration, ext value will affect numbering of output maps.

```
[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 dwd_r06.json:

```
{
    {"Execution": {
        "@name": "dwd_rain_gsp",
        "Aggregation": {
            "@step": 6,
            "@type": "accumulation"
        },
        "OutMaps": {
            "@cloneMap": "/dataset/maps/europe/dem.map",
            "@fmap": 1,
            "@namePrefix": "pR06",
            "@unitTime": 24,
            "Interpolation": {
                "@latMap": "/dataset/maps/europe/lat.map",
                "@lonMap": "/dataset/maps/europe/lon.map",
                "@mode": "nearest"
            }
        },
        "Parameter": {
            "@shortName": "RAIN_GSP",
            "@tend": 18,
            "@tstart": 12
        }
    }
}
```

Command:

```
.pyg2p.py -l DEBUG -c /execution_templates/dwd_r06.json -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[4] and COSMO[5] accumulated precipitation files, the first zero message is an instant precipitation and the decision at EFAS was to use a zero message, as it happens for UKMO[6] 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
```

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

```
...
kg m**-2 Total Precipitation 0-48 tp accum
```

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": {
     "@applyConversion": "k2c",
    "@correctionFormula": "p+gem-dem*0.0065",
    "@demMap": "/dataset/maps/europe/dem.map",
    "@gem": "(z/9.81)*0.0065",
    "@shortName": "2t"
}
```

Evotranspiration correction:

```
"Parameter": {
     "@applyConversion": "k2c",
    "@correctionFormula": "p/gem*(10**((-0.159)*dem/1000))",
    "@demMap": "/dataset/maps/europe/dem.map",
    "@gem": "(10**((-0.159)*(z/9.81)/1000))",
    "@shortName": "2t"
}
```

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 is used?

The application will try to find a geopotential message in input GRIB file. If a geopotential message is not stored, pyg2p will select a geopotential file from *~/.pyg2p/geopotentials* according the geodetic attributes of the input GRIB file. If it doesn't find any suitable grib file, application will exit with an error message.

Geodetic attributes compose the key *id* in the JSON 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 *.pyg2p/geopotentials* folder (under user home directory) and will also add the proper JSON configuration to geopotentials.json file.

Developer 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.

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 json execution file (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.json* file for each parameter (ie. shortName). 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 converted.

In the *Parameter* section of execution template you will configure only the id of the conversion:

```
"Parameter": {
    "@applyConversion": "tommd",
    "@shortName": "alhfl_s"
}
```

Conversion formulas are configured in *parameters.json* configuration file, under the relative parameter configuration, identified by *shortName*. Here the item relative to `alhfl_s` parameter:

```
{
    "@description": "latent heat flux",
    "@shortName": "alhfl_s",
    "@unit": "W/(m^2)",
    "Conversion": [
        {
            "@cutOffNegative": true,
            "@function": "x=x*(-0.0353)",
            "@id": "tommd",
            "@unit": "mm/d"
```

Conversion to apply:

```
        },
        {
            "@cutOffNegative": true,
            "@function": "x=x*(-0.00147)",
            "@id": "tommh",
            "@unit": "mm/h"
        }
    ]
},
```

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

The configuration file parameters.json is needed only for conversion functionality. Tags *description* and *units* serve for logging purposes.

Logging

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, WARN, INFO, DEBUG, in increasing order of verbosity .

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.json')
>>>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.json -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.json -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 JSON files Examples

This paragraph will explain typical execution json configurations.

Example 1: Correction with dem and geopotentials

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

example1.json

```
{
     "Execution": {
          "@name": "eue_t24",
          "Aggregation": {
               "@step": 24,
               "@type": "average"
          },
          "OutMaps": {
               "@cloneMap": "{EUROPE_MAPS}/lat.map",
               "@ext": 1,
               "@fmap": 1,
               "@namePrefix": "pT24",
               "@unitTime": 24,
               "Interpolation": {
                    "@latMap": "{EUROPE_MAPS}/lat.map",
                    "@lonMap": "{EUROPE_MAPS}/long.map",
                    "@mode": "grib_nearest"
               }
          },
          "Parameter": {
               "@applyConversion": "k2c",
               "@correctionFormula": "p+gem-dem*0.0065",
               "@demMap": "{DEM_MAP}",
               "@gem": "(z/9.81)*0.0065",
               "@shortName": "2t"
          }
     }
}
```

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.json configuration.

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. Note that paths to maps uses variables `EUROPE_MAPS` and `DEM_MAP`. You will set these variables in myconf.conf file under ~/.pyg2p/ folder.

The original values will be converted using the conversion “k2c”. This conversion must be configured in the parameters.json file for the variable which is being extracted (2t). See Parameter property configuration at Parameter.

The interpolation method is *grib_nearest*. 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.json -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.json -i 20130325_sro_0to15.grib o ./out_2 -m 0

```
    {"Execution": {
        "@name": "multi_sro",
        "Aggregation": {
            "@step": 24,
            "@type": "accumulation"
        },
        "OutMaps": {
            "@cloneMap": "/dataset/maps/global/dem.map",
            "@fmap": 1,
            "@namePrefix": "psro",
            "@unitTime": 24,
            "Interpolation": {
                "@latMap": "/dataset/maps/global/lat.map",
                "@lonMap": "/dataset/maps/global/lon.map",
                "@mode": "grib_nearest"
            }
        },
        "Parameter": {
            "@applyConversion": "m2mm",
            "@shortName": "sro",
            "@tend": 360,
            "@tstart": 0
        }
    }
}
```

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: Accumulation 24 hours

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

```
  {  "Execution": {
          "@name": "eue_tp",
          "Aggregation": {
               "@step": 24,
               "@type": "accumulation"
          },
          "OutMaps": {
               "@cloneMap": "/dataset/maps/europe5km/lat.map",
               "@fmap": 1,
               "@namePrefix": "pR24",
               "@unitTime": 24,
               "Interpolation": {
                    "@latMap": "/dataset/maps/europe5km/lat.map",
                    "@lonMap": "/dataset/maps/europe5km/long.map",
                    "@mode": "grib_nearest"
               }
          },
          "Parameter": {
               "@applyConversion": "tomm",
               "@shortName": "tp"
          }
     }
}
```

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,	• 113.201	• rain_con	• Only cut off

convective			
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	json[7] (shortName)
Output directory	-o	-o
EPS perturbationNumber	-m	-m
Variable level	--level	json
Aggregation type	Default in parameters.xml. Overwriting with --ma	json (type)
Aggregation step	-t	json (step)
Start step	-s	-s or json (tstart)
End step	-e	-e or json (tend)
Output map unittime	--unittime	json (unitTime)
Interpolation	-r,-l,-a or –intertable	json (see relative section in this manual).
Conversion	parameters.xml	parameters.json and json command file (applyConversion)
Correction	Two formulas hardcoded and configured	json (attributes

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

	by default in parameters.xml or with --temp and --evap.	correctionFormula, gem, dem)
Map extension naming	--fmap, --ext	json
Map prefix name	-n	json
Logging	--report. Loggers directory is under program installation folder	Console logger level overwritable with -l (default is INFO).

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*** attribute.

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 ~/.pyg2p/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* property of the json 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 JSON template file	Execution template, commands file, execution file, json template, json configuration	The JSON 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 json 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.

Term	Synonims	Description
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* property, where *z* is the geopotential value. E.g. gem = “(z/9.81)*0.0065”