

Contents

1	PYWINTER - Python WRF Intermediate Files	2
1.1	Introduction	2
1.2	Intermediate file format	2
1.3	WINTER	2
1.3.1	About winter	2
1.3.2	Geo-information	3
1.3.3	Variables	5
1.3.4	Create intermediate files	9
1.4	Final notes	11

1. PYWINTER - Python WRF Intermediate Files

PYWINTER - Python WRF Intermediate files

Create WPS intermediate files with Python

pywinter 1.2.1

Last update: December 2020

1.1. Introduction

Pywinter is a Python3 library designed to create files in WPS intermediate file format (one step back to execute metgrid.exe). Usually you don't need to create the intermediate files by your own because that is the function of ungrib.exe, but sometimes you don't have your meteorological data in GRIB1 or GRIB2 format, it is the case of NetCDF data. Pywinter allows to create intermediate files by a simple way and this data can be in any format, the only prerequisite is read the data with python and next you can convert it to intermediate file format with pywinter.

1.2. Intermediate file format

The intermediate files contains data written as 2-dimensional horizontal slabs of data. Each horizontal slab contains a single level of a single variable, Any number of horizontal slabs may be written to a single file, each file contains data for a single time.

The files are written as unformatted Fortran records. For each horizontal data slab a number of records are written. Typical information in these records is:

- A version number
- Information common to all types of gridded data
- Information specific to the particular grid type represented
- A 2-dimensional slab of data

For more information you can visit the WRF Tutorial in the official web site. This is just a resume of that that source.

1.3. WINTER

1.3.1. About winter

Pywinter is Oriented Object library, you must create special objects to manipulate data easily.

1.3.2. Geo-information

This data is used to locate the intermediate file information in the space. There are several kind of geo-info, it depends on projection of the original data.

- 0: Cylindrical Equidistant (Lat/lon)
- 1: Mercator projection
- 3: Lambert conformal conic
- 4: Gaussian [global only] (Transverse mercator)
- 5: Polar-stereographic projection

GEO OBJETS

For every projection exists a different object:

Geo0(lats,lons): Cylindrical Equidistant (Lat/lon)

- lats: 1D array of latitudes (degrees north).
- lons: 1D array of longitudes (degrees east).

Geo1(lats,lons,dx,dy,tlat1): Mercator projection

- lats: 1D array of latitudes (degrees north).
- lons: 1D array of longitudes (degrees east).
- dx: Grid spacing in x (m).
- dy: Grid spacing in y (m)
- tlat1: True latitude of projection (degrees north)

Geo3(lats,lons,dx,dy,xloc,tlat1,tlat2,iswin): Lambert conformal conic

- lats: 1D array of latitudes (degrees north).
- lons: 1D array of longitudes (degrees east).
- dx: Grid spacing in x (m).
- dy: Grid spacing in y (m).

- xloc: Center longitude of the projection (degrees east).
- tlat1: True latitude 1 of projection (degrees north)
- tlat2: True latitude 2 of projection (degrees north)
- iswin: Earth[False] or source grid[True] rotated winds (Boolean).

Geo4(lats,lons,nlats,iswin): Gaussian

- lats: 1D array of latitudes (degrees north).
- lons: 1D array of longitude (degrees east).
- nlats: Number of latitudes north of equator (float)
- iswin: Earth[False] or source grid[True] rotated winds (Boolean).

Geo5(lats,lons,dx,dy,xloc,tlat1,iswin): Polar-stereographic

- lats: 1D array of latitudes (degrees north).
- lons: 1D array of longitudes (degrees east).
- dx: Grid spacing in x (m).
- dy: Grid spacing in y (m).
- xloc: Center longitude of the projection (degrees east).
- tlat1: True latitude 1 of projection (degrees north)
- iswin: Earth[False] or source grid[True] rotated winds (Boolean).

Example:

```
1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read Geo-data (Latitudes and longitudes)
6 lat = data.variables['Latitude'][:] # degrees north
7 lon = data.variables['Longitude'][:] # degrees east
8
9 # create winter Geo-information for cylindrical equidistant projection
10 winter_geo = pyw.Geo0(lat,lon)
```

1.3.3. Variables

There are several kind of variables or fields:

- 2D field: It is the surface data.
- 3D field: It is vertical atmospheric data.
- Soil field: It is data in some layers of soil

2D FIELD OBJECT

V2d(name,field)

- name: WPS field name.
- field: 2D numpy array.

Available 2D name fields: PSFC, PMSL, SKINTEMP, SOILHGT, TT, RH, SPECHUMD, UU, VV, LANDSEA, SST, SEAICE, SNOW, TAVGSFC.

Field name	Units	Description	Notes
PSFC	Pa	Surface pressure	
PMSL	Pa	Mean sea-level pressure	
SKINTEMP	K	Skin temperature	
SOILHGT	m	Soil height	
TT	K	2m air temperature	
RH	%	2m relative humidity	Not needed if SPECHUMD is available
SPECHUMD	$kgkg^{-1}$	2m specific humidity	Not needed if RH is available
UU	ms^{-1}	10m wind u component	
VV	ms^{-1}	10m wind v component	
LANDSEA	fraction	Land-sea mask	0=water, 1=land
SST	K	Sea surface temperature	
SEAICE	fraction	Sea-ice fraction	
SNOW	kgm^{-2}	Water equivalent snow depth	
TAVGSFC	K	Daily mean of surface air temperature	

Table 1.1: Available 2D fields

Some 2D fields are masked fields, so you must make sure to convert the missing values to numpy nan before create the pywinter 2d field.

Example:

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read 2D data (2m temperature, 10m U wind, 10m V wind)
6 tp2m = data.variables['T2'][:, :]
7 u10m = data.variables['U10'][:, :]
8 v10m = data.variables['V10'][:, :]
9
10 # Create winter 2D fields
11 winter_t2m = pyw.V2d('TT', tp2m)
12 winter_u10 = pyw.V2d('UU', u10m)
13 winter_v10 = pyw.V2d('VV', v10m)

```

The 2D fields listed in the Table 1.1 are the most important, however exists more fields that are less common but can be useful to some WRF applications, for this reason it is also possible to add your own variables to pywinter. It is just necessary give to the function additional information about the field description, units and pressure level:

Example:

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read 2D data
6 pmaxw = data.variables['pmax'][:, :]
7
8 # Create winter 2D fields
9 winter_new = pyw.V2d('PMAW', pmaxw, 'Pressure at max wind level', 'Pa', '200100')

```

You must use 200100 for surface data and 201300 for sea level pressure data. Also it is very important to remember that all the fields you add must be in accordance with the information processed by metgrid.exe, you can get more information about this in the METGRID.TBL file or in the VTABLE files.

3D NON-ISOBARIC FIELD OBJECT

V3d(name,field)

- name: WPS field name.
- field: 3D array.

Available 2D name fields: TT, RH, SPECHUMD, UU, VV, GHT, PRESSURE.

Field name	Units	Description	Notes
TT	K	3D air temperature	
RH	%	3D relative humidity	No needed if SPECHUMD is available
SPECHUMD	$kg\,kg^{-1}$	3D specific humidity	
UU	ms^{-1}	3D wind u component	
VV	ms^{-1}	3D wind v component	
GHT	m	3D geopotential height	
PRESSURE	Pa	3D pressure	Only needed for non-isobaric datasets

Table 1.2: Available 3D non-isobaric fields

Some 3D fields has missing values, so you must make sure to convert it to numpy nan before create the pywinter 3d field.

Example:

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read 3D non-isobaric data (Pressure, U wind, V wind)
6 press = data.variables['P'][:, :, :]
7 uwind = data.variables['U'][:, :, :]
8 vwind = data.variables['V'][:, :, :]
9
10 # Create winter 3D non-isobaric fields
11 winter_p = pyw.V3d('PRESSURE', press)
12 winter_u = pyw.V3d('UU', u10m)
13 winter_v = pyw.V3d('VV', u10m)

```

3D ISOBARIC FIELD OBJECT

V3dp(name, field, plevs)

- name: WPS field name.
- field: 3D array [plev, lat, lon].
- plevs: 1D vector of pressure levels in hectopascals (hPa).

Available 3D isobaric fields: TT, RH, SPECHUMD, UU, VV, GHT.

Field name	Units	Description	Notes
TT	K	3D air temperature	No needed if SPECHUMD is available
RH	%	3D relative humidity	
SPECHUMD	$kg\,kg^{-1}$	3D specific humidity	
UU	ms^{-1}	3D wind u component	
VV	ms^{-1}	3D wind v component	
GHT	m	3D geopotential height	

Table 1.3: Available 3D isobaric fields

Some 3D fields has missing values, so you must make sure to convert it to numpy nan before create the pywinter 3d field.

Example:

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read 3D isobaric data (temperature, U wind, V wind)
6 temp = data.variables['T'][:, :, :]
7 uwind = data.variables['U'][:, :, :]
8 vwind = data.variables['V'][:, :, :]
9
10 # Read 3D pressure levels (hPa)
11 plevs = data.variables['PLEV'][:]
12
13 # Create winter 3D isobaric fields
14 winter_t = pyw.V3dp('TT', tp2m, plevs)
15 winter_u = pyw.V3dp('UU', u10m, plevs)
16 winter_v = pyw.V3dp('VV', u10m, plevs)

```

SOIL FIELD OBJECT

Vsl(name,field,levs)

- name: WPS field name.
- field: 3D array [lev,lat,lon].
- levs: 1D list of string soil levels or layers in cm.

Available 3D soil fields: ST, SM, SOILT, SOILM.

Field name	Units	Description	Notes
SM	m^3m^{-3}	Soil moisture	'ttt' is layer top, 'bbb' is layer bottom
ST	K	Soil temperature	'ttt' is layer top, 'bbb' is layer bottom
SOILM	kg^3m^{-3}	Soil moisture	'mmm' is tthe level depth Not needed if SM avalaible
SOILT	K	Soil temperature	'mmm' is tthe level depth. Not needed if ST avalaible

Table 1.4: Available soil fields

Is important to know that if you have ST or SM, you must indicate the layer as `[bbbtth]` (top layer - bottom layer in cm) and if you have SOILT or SOILM, you must indicate the level as `[mmm]` (level depth in cm).

Soil fields are used to be masked fields, so you must make sure to convert the missing values to numpy nan before create the pywinter soil field.

Example:

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read 3D soil data (temperature)
6 soil_t_layer = data.variables['SLTY'][:, :, :]
7 soil_t_level = data.variables['SLTL'][:, :, :]
8
9 # 3D soil layers and levels
10 sl_layer = ['000010', '010040', '040100', '100200']
11 sl_level = ['010', '040', '100', '200']
12
13 # Create winter 3D soil fields
14 winter_soil_t_layer = pyw.Vsl('ST', soil_t_layer, sl_layer)
15 winter_soil_t_level = pyw.Vsl('SOILT', soil_t_level, sl_level)

```

1.3.4. Create intermediate files

When you have all the pywinter meteorological variables, now is time to create the intermediate files, you must use **cinter** function and it is very easy.

cinter(file, date, geoinfo, varias, rout)

- file: File name or prefix.
- date: Datetime of the file

- geoinfo: pywinter Geo object.
- varias: List of pywinter fields.
- rout: Path where files will be created. ([optional] if not rout, files will be created in current folder)

Actually you can write just one file per time step. For creating several files you can use a loop, just remember that parameters filen, geoinfo, and rout will be static.

Final example

```

1 import numpy as np
2 import pywinter.winter as pyw
3 import data_example as data
4
5 # Read Geo-data
6 lat = data.variables['Latitude'][:]
7 lon = data.variables['Longitude'][:]
8 # Read 2D data
9 tp2m = data.variables['T2'][:, :]
10 # Read 3D data
11 temp = data.variables['T'][:, :, :]
12 plevs = data.variables['PLEV'][:]
13 # Read 3D soil data
14 soilt = data.variables['SLTY'][:, :, :]
15 sl_layer = ['000010', '010040', '040100', '100200']
16
17 # Create winter fields
18 winter_geo = pyw.Geo0(lat, lon)
19
20 winter_t2m = pyw.V2d('TT', tp2m)
21 winter_t = pyw.V3dp('TT', tp2m, plevs)
22 winter_soilt_layer = pyw.Vsl('ST', soilt, sl_layer)
23
24 # Listing fields
25 total_fields = [winter_t2m,
26                 winter_t,
27                 winter_soilt_layer]
28
29 # Out path
30 path_out = '/home/Documents/intermediate_files/'
31
32 # Write intermediate file
33 pwy.cinter('FILE', '1994-05-18_06', winter_geo, total_fields, path_out)

```

*****WARNING***:** You can choose any prefix for the intermediate files though usually are called

FILE, however the datetime must be chosen carefully, the files will be created with success but metgrid.exe sometimes can't read them:

- If your original data has an exact hourly time resolution, for example: [**05:00:00, 06:00:00**], your intermediate files will be [**FILE:YYYY-MM-DD_05, FILE:YYYY-MM-DD_06**]. **Minutes and seconds are not needed**, if you put them, it is probably metgrid.exe can't find the intermediate files.
- If your original data has less than one hour time resolution, for example: [**05:00:00, 05:30:00**], your intermediate files will be [**FILE:YYYY-MM-DD_05:00, FILE:YYYY-MM-DD_05:30**]. **Seconds are not needed**, if you put them, it is probably metgrid.exe can't find the intermediate files.
- If your original data has a not-exact hourly time resolution, for example: [**05:30:00, 06:30:00, 07:30:00**], metgrid.exe won't find the intermediate files. You can trick the program if you set the **interval_seconds** namelist parameter as **3601** and your intermediate files in this case will be [**FILE:YYYY-MM-DD_05:30:00, FILE:YYYY-MM-DD_06:30:01, FILE:YYYY-MM-DD_07:30:02**]. This just works for short runs, because you must remember that in 60 hours the delay will be 1 minute with respect to your original data. This could be a bug or a simple restriction from metgrid.exe.

1.4. Final notes

Actually pywinter can't check if you write the file with all the necessary fields to run WRF, neither cannot check if your information is consistent, therefore is important you make sure the data is well before you create the files, you can get more information of how to create good files in the WRF User guide and WRF web tutorial. additionally I recommend the text 'Intermediate_file_notes.pdf' where there are some notes about intermediate files. Also you can check the intermediate files with WPS util programs:

- .../WRF/WPS/util/rd_intermediate.exe: It reads fields into intermediate files and show them.
- .../WRF/WPS/util/int2nc.exe: it converts intermediate files to netCDF format.

Example

```
1 mylinux@User:~/Documents/intermediate_files$  
/home/User/WRF/WPS/util/rd_intermediate.exe IFILE:1994-05-18_06:00:00
```

```
1 ...  
2 ..  
3 .  
4 =====  
5 FIELD = RH  
6 UNITS = % DESCRIPTION = Relative Humidity
```

```
7 DATE = 06-01_00:00:0000000 FCST = 0.000000
8 SOURCE = INTERMF
9 LEVEL = 1000.000000
10 I,J DIMS = 288, 192
11 IPROJ = 0 PROJECTION = LAT LON
12 REF_X, REF_Y = 1.000000, 1.000000
13 REF_LAT, REF_LON = -90.000000, 0.000000
14 DLAT, DLON = 0.942408, 1.250000
15 EARTH_RADIUS = 6367.470215
16 DATA(1,1)=34.307831
17
18 SUCCESSFUL COMPLETION OF PROGRAM RD_INTERMEDIATE
```

```
1 mylinux@User:~/Documents/intermediate_files$
  /home/User/WRF/WPS/util/int2nc.exe IFILE:1994-05-18_06:00:00
```

```
1 ...
2 ..
3 .
4 Reading Field, Level: TT, 20000
5 Reading Field, Level: TT, 16800
6 Reading Field, Level: TT, 13600
7 Reading Field, Level: TT, 10400
8 SUCCESSFUL COMPLETION OF PROGRAM INT2NC, IFILE:1994-05-18_06:00:00.nc WRITTEN.
```

Contact:

Pywinter have been tested with success for many cases but its yet an early version and it can be improved. if you have any problem our suggestion please contact to dniloash@gmail.com