Contents

PYWINTER - Python WRF Intermediate files

Read and create WRF-WPS intermediate files with Python

pywinter 2.0.4

Last update: February 2021

1.1. Introduction

Pywinter is a Python3 library designed for handling files in WRF-WPS intermediate file format. Usually you don't need to deal with the intermediate files by your own because that is the function of ungrib.exe, but sometimes you don't have your meteorological data in GRIB format. Pywinter allows to read and create intermediate files by a simple way.

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. READING DATA

For reading the intermediate files information you must utilize the function **rinter**. Once you have read the information you can manipulate the data easily. The results will be a dictionary that contains the variables in the intermediate file, also every key includes general information, geo information, levels, and the data array.

Example:

```
import numpy as np
import pywinter.winter as pyw
import data_example as data

infile = '/home/allyson/Documents/files/FILE:1994-05-18_06'

interfile = pyw.rinter(infile)

print(interfile.keys())
>> dict_keys(['LANDSEA', 'ST', 'SST', 'SOILHGT', 'PSFC', 'HGT', 'SKINTEMP', 'TT',
'PMSL', 'VV', 'SM', 'UU', 'RH', 'TT2M', 'RH2M', 'UU10M', 'VV10M'])

print(interfile['TT'].general)
>> {'VERSION': 5, 'HDATE': '2015-07-27_12:00:00', 'XFCST': 0.0, 'MAP_SOURCE': 'ECMWF',
'FIELD': 'TT', 'UNITS': 'K', 'DESC': 'Temperature', 'XLVL': '1000', 'NX': 441, 'NY': 329,
'EARTH_RADIUS': 6367.47021484375, 'IS_WIND_EARTH_REL': False}

print(interfile['TT'].geoinfo)
>> {'IPROJ': 0, 'PROJ': 'Cylindrical Equidistant (0)', 'STARTLOC': 'SWCORNER',
'STARTLAT': 38.0, 'STARTLON': -130.0, 'DELTALAT': -0.25, 'DELTALON': 0.25}

print(interfile['TT'].level)
>> [100000. 97500. 95000. 92500. 90000. 87500. 85000. 82500. 80000.
  77500. 75000. 70000. 65000. 60000. 55000. 50000. 45000. 40000.
  35000. 30000. 25000. 20000. 17500. 15000. 12500. 10000.  7000.
   5000.  3000.  2000.  1000.]

print(interfile['TT'].val)
>> [[[289.86547852 289.89868164 289.85571289 ... 294.19750977 294.20141602
   294.19360352]
  ...
  [278.23071289 277.93383789 277.69360352 ... 280.58032227 280.63500977
   280.70727539]]
   ...
 [[234.77418518 234.80836487 234.85231018 ... 232.22828674 232.20973206
   232.20582581]
   ...
  [213.70191956 213.95289612 214.19410706 ... 211.46461487 211.38844299
   211.31422424]]]

print(interfile['TT'].val.shape)
>> (31, 441, 329)
```

1.4. WRITING DATA

For writing data pywinter its important to create special objects to manipulate the data easily.

1.4.1. 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)

- stlat: SOUTH-WEST corner latitude of data (degrees north).
- stlon: SOUTH-WEST corner longitude of data (degrees east).
- dlat: Latitude increment (degrees).
- dlon: Longitude increment (degrees).

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

- stlat: SOUTH-WEST corner latitude of data (degrees north).
- stlon: SOUTH-WEST corner longitude of data (degrees east).
- dx: Grid spacing in x (Km).
- dy: Grid spacing in y (Km)
- tlat1: True latitude of projection (degrees north)

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

- stlat: SOUTH-WEST corner latitude of data (degrees north).

- stlon: SOUTH-WEST corner longitude of data (degrees north).
- dx: Grid spacing in x (Km).
- dy: Grid spacing in y (Km).
- 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

- stlat: SOUTH-WEST corner latitude of data (degrees north).
- stlon: SOUTH-WEST corner longitude of data (degrees north).
- nlats: Number of latitudes north of equator (float)
- dlon: Longitude increment (degrees).
- iswin: Earth[False] or source grid[True] rotated winds (Boolean).

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

- lats: SOUTH-WEST corner latitude of data (degrees north).
- lons: SOUTH-WEST corner longitude of data (degrees north).
- dx: Grid spacing in x (Km).
- dy: Grid spacing in y (Km).
- 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)
```

```
lat = data.variables['Latitude'][:] # degrees north
lon = data.variables['Longitude'][:] # degrees east

dlat = np.abs(lat[1] - lat[0])
dlon = np.abs(lon[1] - lon[0])

# create winter Geo-information for cylindrical equidistant projection
winter_geo = pwy.Geo0(lat[0],lon[0],dlat,dlon)
```

1.4.2. 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 avalaible
SPECHUMD	$kgkg^{-1}$	2m specific humidity	Not needed if RH is avalaible
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:

```
import numpy as np
import pywinter.winter as pyw
import data_example as data

# Read 2D data (2m temperature, 10m U wind, 10m V wind)
tp2m = data.variables['T2'][:,:]
u10m = data.variables['U10'][:,:]
v10m = data.variables['V10'][:,:]

# Create winter 2D fields
winter_t2m = pyw.V2d('TT',tp2m)
winter_u10 = pyw.V2d('UU',u10m)
winter_v10 = pyw.V2d('VV',u10m)
```

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 for some WRF applications, for doing this 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:

```
import numpy as np
```

```
import pywinter.winter as pyw
import data_example as data

# Read 2D data
pmaxw = data.variables['pmax'][:,:]

# Create winter 2D fields
winter_new = pyw.V2d('PMAXW',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 avalaible
SPECHUMD	$kgkg^{-1}$	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:

```
import numpy as np
import pywinter.winter as pyw
import data_example as data

```

```
# Read 3D non-isobaric data (Pressure, U wind, V wind)
press = data.variables['P'][:,:,:]
uwind = data.variables['U'][:,:,:]
vwind = data.variables['V'][:,:,:]

# Create winter 3D non-isobaric fields
winter_p = pyw.V3d('PRESSURE',press)
winter_u = pyw.V3d('UU',u10m)
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 (Pa).

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 avalaible
RH	%	3D relative humidity	
SPECHUMD	$kgkg^{-1}$	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:

```
import numpy as np
import pywinter.winter as pyw
import data_example as data

# Read 3D isobaric data (temperature, U wind, V wind)
temp = data.variables['T'][:,:,:]
uwind = data.variables['U'][:,:,:]
```

```
vwind = data.variables['V'][:,:,:]

# Read 3D pressure levels (hPa)
plevs = data.variables['PLEV'][:]

# Create winter 3D isobaric fields
winter_t = pyw.V3dp('TT',tp2m,plevs)
winter_u = pyw.V3dp('UU',u10m,plevs)
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 [*bbbttt*] (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:

```
import numpy as np
import pywinter.winter as pyw
import data_example as data
```

```
4
5  # Read 3D soil data (temperature)
6  soilt_lay = data.variables['SLTY'][:,:,:]
7  soilt_lev = 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_soilt_layer = pyw.Vsl('ST',soilt_lay,sl_layer)
15 winter_soitl_level = pyw.Vsl('SOILT',soilt_lev,sl_level)
```

1.4.3. 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(filen,date,geoinfo,varias,rout)

- filen: 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  dlat = np.abs(lat[1] - lat[0])
9  dlon = np.abs(lon[1] - lon[0])
10 # Read 2D data
11 tp2m = data.variables['T2'][:,:]
```

```
# Read 3D data
temp = data.variables['T'][:,:,:]
plevs = data.variables['PLEV'][:]
# Read 3D soil data
soilt = data.variables['SLTY'][:,:,:]
sl_layer = ['000010','010040','040100','100200']

# Create winter fields
winter_geo = pyw.Geo0(lat[0],lon[0],dlat,dlon)

winter_t2m = pyw.V2d('TT',tp2m)
winter_t = pyw.V3dp('TT',tp2m,plevs)
winter_soilt_layer = pyw.Vsl('ST',soilt,sl_layer)

# Listing fields
total_fields = [winter_t2m,
                winter_t,
                winter_soilt_layer]

# Out path
path_out = '/home/Documents/intermediate_files/'

# Write intermediate file
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.

1.5. 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

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

```
...
..
.
================================================
FIELD = RH
UNITS = % DESCRIPTION = Relative Humidity
DATE = 06-01_00:00:0000000 FCST = 0.000000
SOURCE = INTERMF
LEVEL = 1000.000000
I,J DIMS = 288, 192
IPROJ = 0 PROJECTION = LAT LON
  REF_X, REF_Y = 1.000000, 1.000000
  REF_LAT, REF_LON = -90.000000, 0.000000
  DLAT, DLON = 0.942408, 1.250000
  EARTH_RADIUS = 6367.470215
DATA(1,1)=34.307831

SUCCESSFUL COMPLETION OF PROGRAM RD_INTERMEDIATE
```

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

```
...
..
.
 Reading Field, Level: TT, 20000
 Reading Field, Level: TT, 16800
```

```
Reading Field, Level: TT, 13600
Reading Field, Level: TT, 10400
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