PyGrib Documentation

– by

Arulalan.T

arulalant@gmail.com

Project Associate,

Centre for Atmospheric Sciences ,

Indian Institute of Technology Delhi.

03.05.2011

Pygrib Installation in CDAT path procedure

Now we are going to see the procedure how to install PyGrib [1] inside the CDAT [2] installed path in our system.

[1] http://code.google.com/p/pygrib/
[2] http://www2-pcmdi.llnl.gov/cdat

Let us assume we have installed CDAT in our system and its working path is /usr/local/cdat5.2/bin/ .

Make sure that the path working fine as like below….

```
$ /usr/local/cdat5.2//bin/python
Executing /usr/local/cdat5.2//bin/python
Python 2.5.2 (r252:60911, Feb 2 2011, 15:29:23)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To install PyGrib in your normal python path, kindly follow the link

http://code.google.com/p/pygrib/wiki/LinuxMacInstallation

First Install the following dependencies for pygrib in our ubuntu.

```
$ sudo apt-get install libjasper-dev libjasper-runtime .
```

Note : Do Not install libjasper-java, jasper and openjpeg-tools.

If you installed these above 3 packages, then it should crash our ubuntu login and its X.

These are all stuff tested in ubuntu 10.10.

libjasper-dev and libjasper-runtime are working fine without making any problem.

Download grib_api-1.9.5.tar.gz from http://www.ecmwf.int/products/data/software/download/grib_api.html

Download pyproj-1.8.8.tar.gz from http://code.google.com/p/pyproj/downloads/list

Download pygrib-1.8.0.tar.gz from http://code.google.com/p/pygrib/downloads/list

Now we need to move the three downloaded packages to the cdat path as like below …

```
$ mv grib_api-1.9.5.tar.gz /usr/local/cdat5.2/bin/

$ mv pyproj-1.8.8.tar.gz /usr/local/cdat5.2/bin/

$ mv pygrib-1.8.0.tar.gz /usr/local/cdat5.2/bin/
```

Now change the current working directory into cdat .

```
$ cd /usr/local/cdat5.2/bin/

$ pwd
/usr/local/cdat5.2/bin/

$ su – root
```

Installing grib-api

Following these instructions should be safe on an average linux machine:

- mkdir grib_api_dir
- tar zxvf grib_api-1.9.5.tar.gz
- cd grib_api-1.9.5
- export CFLAGS=”-O2 -fPIC”
- ./configure –prefix= `pwd`/../grib_api_dir
- make
- make check
- make install
- cd ..
- pwd
  /usr/local/cdat5.2/bin/

Installing pyproj

This one is very simple, just like the normal python install :

- tar zxvf pyproj-1.8.8.tar.gz
- cd pyproj-1.8.8
- /usr/local/cdat5.2/bin//python setup.py install
- cd ..
- pwd
  /usr/local/cdat5.2/bin/

Installing pygrib

Installing pygrib inside cdat path.

- export GRIBAPI_DIR=`pwd`/grib_api_dir
- export JASPER_DIR=/usr
- tar zxvf pygrib-1.8.0.tar.gz
- cd pygrib-1.8.0
- /usr/local/cdat5.2/bin//python setup.py install
- /usr/local/cdat5.2/bin//python test.py
- pwd
  /usr/local/cdat5.2/bin/pygrib

Checking pygrib in CDAT path

open a new terminal as a normal user

- $ /usr/local/cdat5.2//bin/python**Executing /usr/local/cdat5.2//bin/python**
  Python 2.5.2 (r252:60911, Feb 2 2011, 15:29:23)
  [GCC 4.4.5] on linux2
  Type “help”, “copyright”, “credits” or “license” for more information.
  >>> import pygrib
  >>>

  Thats it. We have successfully installed pygrib in our cdat path and imported it.

Pygrib Usage

- from the python interpreter prompt, import the package:
  ```
  >>> import pygrib
  ```

- open a GRIB file, create a grib message iterator:
  ```
  >>> grbs = pygrib.open('sampledata/flux.grb')
  ```

- pygrib open instances behave like regular python file objects,
  with seek, tell, read, readline and close methods, except that offsets are measured in grib messages instead of bytes:
  ```
  >>> grbs.seek(2)
  >>> grbs.tell()
  2
  >>> grb = grbs.read(1)[0] # read returns a list with the next N (N=1 in this case) messages.
  >>> grb # printing a grib message object displays summary info
  3:Maximum temperature:K (instant):regular_gg:heightAboveGround:level 2 m:fcst time 108-120:from 200402291200
  >>> grbs.tell()
  3
  ```

- print an inventory of the file:
  ```
  >>> grbs.seek(0)
  >>> for grb in grbs:
  >>>     grb
  1:Precipitation rate:kg m**-2 s**-1 (avg):regular_gg:surface:level 0:fcst time 108-120:from 200402291200
  2:Surface pressure:Pa (instant):regular_gg:surface:level 0:fcst time 120:from 200402291200
  3:Maximum temperature:K (instant):regular_gg:heightAboveGround:level 2 m:fcst time 108-120:from 200402291200
  4:Minimum temperature:K (instant):regular_gg:heightAboveGround:level 2 m:fcst time 108-120:from 200402291200
  ```

- find the first grib message with a matching name:
  ```
  >>> grb = grbs.select(name='Maximum temperature')[0]
  ```

- extract the data values using the 'values' key (grb.keys() will return a list of the available keys):
  ```
  # The data is returned as a numpy array, or if missing values or a bitmap
  # are present, a numpy masked array.  Reduced lat/lon or gaussian grid
  # data is automatically expanded to a regular grid. Details of the internal
  # representation of the grib data (such as the scanning mode) are handled
  # automatically.
  >>> data = grb.values # same as grb['values']
  >>> data.shape, data.min(), data.max()
  (94, 192) 223.7 319.9
  ```

- get the latitudes and longitudes of the grid:
  ```
  >>> lats, lons = grb.latlons()
  >>> lats.shape, lats.min(), lats.max(), lons.shape, lons.min(), lons.max()
  (94, 192) -88.5419501373 88.5419501373  0.0 358.125
  ```

- get the second grib message:

  ```
  >>> grb = grbs.message(2) # same as grbs.seek(1); grb=grbs.readline()
  >>> grb
  2:Surface pressure:Pa (instant):regular_gg:surface:level 0:fcst time 120:from 200402291200
  ```

- modify the values associated with existing keys (either via attribute or dictionary access):

  ```
  >>> grb['forecastTime'] = 240
  >>> grb.dataDate = 20100101
  ```

- get the binary string associated with the coded message:

  ```
    >>> msg = grb.tostring()
  >>> grbs.close() # close the grib file.
  ```

- write the modified message to a new GRIB file:

  ```
  >>> grbout = open('test.grb','wb')
  >>> grbout.write(msg)
  >>> grbout.close()
  >>> pygrib.open('test.grb').readline()
  1:Surface pressure:Pa (instant):regular_gg:surface:level 0:fcst time 240:from 201001011200
  ```

Few More examples

- To open

  ```
  >>> grbs = pygrib.open('gdas1.t00z.grbanl')
  ```

- To get the filename

  ```
  >>> grbs.name
  'gdas1.t00z.grbanl'
  ```

- To get how many messages/lines it has

  ```
  >>> grbs.messages
  361
  ```

- To get message of particular line no

  ```
  >>> grbs[5]
  5:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 900:fcst time  0:from 201005250000
  >>> grbs.message(250)
  250:V component of wind:m s**-1 (instant):regular_ll:pressureFromGroundLayer:lev el None:fcst time 0:from 201005250000
  ```

  Note : grb message numbers start at 1

- To get the next message from current position

  ```
  >>> grbs.next()
  251:V component of wind:m s**-1 (instant):regular_ll:pressureFromGroundLayer:lev el None:fcst time 0:from 201005250000
  ```

- To move the file pointer to the first position

  ```
  >>> grbs.seek(0)
  >>> grbs.next()
  1:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 1000:fcst tim e 0:from 201005250000
  ```

- To get the first message

  ```
  >>> g=grbs[1]
  >>> g.messagenumber
  1
  ```

- To get the keys of the message

```
>>> g.keys()
['parametersVersion', 'definitionFilesVersion', 'two', 'three', 'truncateDegrees', 'offset', 'count', 'countTotal', 'unitsFactor', 'unitsBias', 'eps', 'globalDomain', 'libraryVersion',
'kindOfProdu ct', 'GRIBEditionNumber', 'offsetSection0', 'section0Length', 'totalLength', 'editionNumber', 'WMO', 'productionStatusOfProcessedData', 'section1Length', 'table2Version',
'centre', 'generatingProce ssIdentifier', 'gridDefinition', 'indicatorOfParameter', 'parameterName', 'parameterUnits', 'indicatorOfTypeOfLevel', 'pressureUnits', 'typeOfLevel', 'level',
'yearOfCentury', 'month', 'day', 'hour ', 'minute', 'second', 'unitOfTimeRange', 'P1', 'P2', 'timeRangeIndicator', 'numberIncludedInAverage','numberMissingFromAveragesOrAccumulations',
'centuryOfReferenceTimeOfData', 'subCentre', 'para mIdECMF', 'paramId', 'cfName', 'units', 'name', 'decimalScaleFactor', 'setLocalDefinition', 'dataDate', 'year', 'dataTime', 'julianDay',
'stepUnits', 'stepType', 'stepRange', 'startStep', 'endStep' , 'marsParam', 'validityDate', 'validityTime', 'wrongPadding', 'deleteLocalDefinition', 'localUsePresent', 'shortNameECMF',
'shortName', 'ifsParam', 'gridDescriptionSectionPresent', 'bitmapPresent' , 'section2Length', 'radius', 'shapeOfTheEarth', 'numberOfVerticalCoordinateValues', 'neitherPresent',
'pvlLocation', 'dataRepresentationType', 'gridDefinitionTemplateNumber', 'Ni', 'Nj', 'latitude OfFirstGridPoint', 'latitudeOfFirstGridPointInDegrees', 'longitudeOfFirstGridPoint',
'longitudeOfFirstGridPointInDegrees', 'resolutionAndComponentFlags', 'ijDirectionIncrementGiven', 'earthIsOblate ', 'resolutionAndComponentFlags3', 'resolutionAndComponentFlags4',
'uvRelativeToGrid', 'resolutionAndComponentFlags6', 'resolutionAndComponentFlags7', 'resolutionAndComponentFlags8', 'latitudeOfLas tGridPoint', 'latitudeOfLastGridPointInDegrees',
'longitudeOfLastGridPoint', 'longitudeOfLastGridPointInDegrees', 'iDirectionIncrement', 'jDirectionIncrement', 'scanningMode', 'iScansNegatively', ' jScansPositively',
'jPointsAreConsecutive', 'alternativeRowScanning', 'iScansPositively', 'scanningMode4', 'scanningMode5', 'scanningMode6', 'scanningMode7', 'scanningMode8',
'jDirectionIncrementInDegrees', 'iDirectionIncrementInDegrees', 'numberOfDataPoints', 'numberOfValues', 'latLonValues', 'latitudes', 'longitudes', 'distinctLatitudes', 'distinctLongitudes',
'PVPresent', 'PLPresent', 'le ngthOfHeaders', 'missingValue', 'tableReference', 'section4Length', 'halfByte', 'dataFlag', 'binaryScaleFactor', 'referenceValue', 'bitsPerValue',
'referenceValueError', 'sphericalHarmonics', 'comp lexPacking', 'integerPointValues', 'additionalFlagPresent', 'packingType', 'bitMapIndicator', 'values', 'numberOfCodedValues',
'packingError', 'unpackedError', 'maximum', 'minimum', 'average', 'num berOfMissing', 'standardDeviation', 'skewness', 'kurtosis', 'isConstant', 'dataLength', 'changeDecimalPrecision',
'decimalPrecision', 'bitsPerValueAndRepack', 'scaleValuesBy', 'offsetValuesBy', 'gr idType', 'getNumberOfValues', 'section5Length']
```

From the above keys we can choose any one and pass it to the g var as shown like below

- To get the data of the message

```
>>> g['values']
array([[ 289.1,  289.1,  289.1, ...,  289.1,  289.1,  289.1],
 [ 300.8,  300.8,  300.8, ...,  301. ,  300.9,  300.9],
 [ 293.7,  293.6,  293.6, ...,  293.7,  293.7,  293.7],
 ...,
 [ 231.2,  231.3,  231.4, ...,  231. ,  231.1,  231.1],
 [ 232.2,  232.3,  232.3, ...,  232.1,  232.1,  232.2],
 [ 233. ,  233. ,  233. , ...,  233. ,  233. ,  233. ]])
```

- To get the missing Value of the message

```
>>> g['missingValue']
9999
```

- To get the level of the message

```
>>> g['level']
1000
```

- To get the latitudes as numpy array

```
>>> g['distinctLatitudes']
array([-90. , -89.5, -89. , -88.5, -88. , -87.5, -87. , -86.5, -86. ,
 ..., ..., ...,
85.5,  86. ,  86.5,  87. ,  87.5,  88. ,  88.5,  89. ,  89.5,  90. ])
```

- To get the longitudes as numpy array

```
>>> g['distinctLongitudes']
array([   0. ,    0.5,    1. ,    1.5,    2. ,    2.5,    3. ,    3.5,
..., ..., ...,
356. ,  356.5,  357. ,  357.5,  358. ,  358.5,  359. ,  359.5])
```

- To get both latitudes and longitudes

```
>>> g.latlons()
(array([[-90. , -90. , -90. , ..., -90. , -90. , -90. ],
...,
[ 90. ,  90. ,  90. , ...,  90. ,  90. ,  90. ]]), array([[   0. ,    0.5,    1. , ...,  358.5,  359. ,  359.5],
 ...,
 [   0. ,    0.5,    1. , ...,  358.5,  359. ,  359.5]]))
```

- Using for loop

```
>>> for i in grbs:
                print i
1:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 1000:fcst time 0:from 201005250000
2:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 975:fcst time 0:from 201005250000
3:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 950:fcst time 0:from 201005250000
…
…
360:V component of wind:m s**-1 (instant):regular_ll:maxWind:level 0:fcst time 0:from 201005250000
361:V component of wind:m s**-1 (instant):regular_ll:sigma:level 9950:fcst time 0:from 201005250000
```

- **To get data and other properties belongs to one gribfile, variableName, typeOfLevel, and level by fastest way using the file index**

```
>>> file_name = 'gdas1.t00z.grbanl'
>>> fileidx = pygrib.index(file_name,'name','typeOfLevel','level')
>>> g = fileidx.select(name = "Geopotential Height", typeOfLevel = "isobaricInhPa", level = 1000)
>>> g
[1:Geopotential Height:gpm (instant):regular_ll:isobaricInhPa:level 1000:fcst time 0:from 201005250000]
>>> g[0]['values']
array([[ 289.1,  289.1,  289.1, ...,  289.1,  289.1,  289.1],
[ 300.8,  300.8,  300.8, ...,  301. ,  300.9,  300.9],
 ...,
[ 233. ,  233. ,  233. , ...,  233. ,  233. ,  233. ]])
```

Notes:

1. Here this 'fileidx' belongs to the file_name = 'gdas1.t00z.grbanl'
2. fileindex is more faster than pygrib.open() method
3. In fileindex we can set the arguments maximum only the following three options ... 'name','typeOfLevel','level'

Links :

- Home Page : http://code.google.com/p/pygrib/
- Documentation : http://pygrib.googlecode.com/svn/trunk/docs/index.html
- Installation in System Python Path : http://code.google.com/p/pygrib/wiki/LinuxMacInstallation
- Installation in CDAT Path : http://tuxcoder.wordpress.com/2011/05/03/how-to-install-pygrib-inside-our-cdat-path/