getgfs

Release 1.0.0

Jago Strong-Wright

Jul 04, 2022

CONTENTS:

getgfs extracts weather forecast variables from the NOAA GFS forecast in a pure python, no obscure dependencies way. Currently you can:

- "Connect" to a forecast
- Search the variables
- Download variables for any time range, longitude, latitude and altitude
- Download "wind profiles" where you get an interpolation object for the u and v wind components by altitude

CHAPTER

ONE

INSTALLING

Installation is simple with PyPi:

```
`pip install getgfs`
```

CHAPTER

TWO

USAGE

The library is straight forward to use. To get started create a Forecast object by:

```
>>>import getgfs
>>>f=getgfs.Forecast("0p25")
```

You can chose the resolution to be *0p25*, *0p50* or *1p00* and for the *0p25* forecast you can optional specify a shorter timestep by adding *1hr* after.

First to find what variable you are looking for use the search function, for example if I want the wind speed I could search for "wind":

```
>>>f.search("wind")
[('gustsfc', '** surface wind speed (gust) [m/s] ', 100), ('ugrdprs', '** (1000 975 950␣
↪925 900.. 7 5 3 2 1) u-component of wind [m/s] ', 125), ('ugrd_1829m', '** 1829 m␣
↪above mean sea level u-component of wind [m/s] ', 125), ('ugrd_2743m', '** 2743 m␣
↪above mean sea level u-component of wind [m/s] ', 125), ('ugrd_3658m', '** 3658 m␣
↪above mean sea level u-component of wind [m/s] ', 125), ('ugrd10m', '** 10 m above␣
↪ground u-component of wind [m/s] ', 125), ('ugrd20m', '** 20 m above ground u-
↪component of wind [m/s] ', 125), ('ugrd30m', '** 30 m above ground u-component of wind␣
↪[m/s] ', 125), ('ugrd40m', '** 40 m above ground u-component of wind [m/s] ', 125), (
↪'ugrd50m', '** 50 m above ground u-component of wind [m/s] ', 125), ('ugrd80m', '** 80␣
↪m above ground u-component of wind [m/s] ', 125), ('ugrd100m', '** 100 m above ground␣
↪u-component of wind [m/s] ', 125), ('ugrdsig995', '** 0.995 sigma level u-component of␣
↪wind [m/s] ', 125), ('ugrd30_0mb', '** 30-0 mb above ground u-component of wind [m/s]
↪', 125), ('ugrd2pv', '** pv=2e-06 (km^2/kg/s) surface u-component of wind [m/s] ',␣
↪125), ('ugrdneg2pv', '** pv=-2e-06 (km^2/kg/s) surface u-component of wind [m/s] ',␣
↪125), ('ugrdpbl', '** planetary boundary layer u-component of wind [m/s] ', 125), (
↪'ugrdtrop', '** tropopause u-component of wind [m/s] ', 125), ('vgrdprs', '** (1000␣
↪975 950 925 900.. 7 5 3 2 1) v-component of wind [m/s] ', 125), ('vgrd_1829m', '**␣
↪1829 m above mean sea level v-component of wind [m/s] ', 125), ('vgrd_2743m', '** 2743␣
↪m above mean sea level v-component of wind [m/s] ', 125), ('vgrd_3658m', '** 3658 m␣
↪above mean sea level v-component of wind [m/s] ', 125), ('vgrd10m', '** 10 m above␣
↪ground v-component of wind [m/s] ', 125), ('vgrd20m', '** 20 m above ground v-
↪component of wind [m/s] ', 125), ('vgrd30m', '** 30 m above ground v-component of wind␣
↪[m/s] ', 125), ('vgrd40m', '** 40 m above ground v-component of wind [m/s] ', 125), (
↪'vgrd50m', '** 50 m above ground v-component of wind [m/s] ', 125), ('vgrd80m', '** 80␣
↪m above ground v-component of wind [m/s] ', 125), ('vgrd100m', '** 100 m above ground␣
↪v-component of wind [m/s] ', 125), ('vgrdsig995', '** 0.995 sigma level v-component of␣
↪wind [m/s] ', 125), ('vgrd30_0mb', '** 30-0 mb above ground v-component of wind [m/s]
↪', 125), ('vgrd2pv', '** pv=2e-06 (km^2/kg/s) surface v-component of wind [m/s] ',␣
↪125), ('vgrdneg2pv', '** pv=-2e-06 (km^2/kg/s) surface v-component of wind [m/s] ',␣
↪125), ('vgrdpbl', '** planetary boundary layer v-component of wind [m/s] ', 125), (
↪'vgrdtrop', '** tropopause v-component of wind [m/s] ', 125), ('hgtmwl', '** max wind␣
↪geopotential height [gpm] ', 133), ('icahtmwl', '** max wind icao standard atmosphere␣
↪reference height [m] ', 133), ('presmwl', '** max wind pressure [pa] ', 133), ('tmpmwl
↪', '** max wind temperature [k] ', 133), ('ugrdmwl', '** max wind u-component of wind␣
↪[m/s] ', 133), ('vgrdmwl', '** max wind v-component of wind [m/s] ', 133)]
```

(continues on next page)

(continued from previous page)

So now I can see I might want "gustsfc". Now if I want the wind speed at N70.1 W94.7 at 5:30 on the 27th of Febuary (only forecasts going back around a week are available and furutre times available depend on the forecast - look for f.times) I could do:

```
>>>res=f.get(["gustsfc"],"20210227 5:30", 70.1,-94.7)
>>>res.variables["gustsfc"].data
array([[[18.808477]]])
```

You can get more information (e.g. what is the units of this) by exploring the variables information

```
>>>f.variables["gustsfc"]
{'_FillValue': 9.999e+20, 'missing_value': 9.999e+20, 'long_name': '** surface wind␣
↪speed (gust) [m/s] ', 'level_dependent': False}
```

You can also get multiple variables by including more names in the list or a range of positions by using "'[min_lat:max_lat]'" type strings in place of the position parameters.

CHAPTER

THREE

ABOUT

The incentive to write this library was that the current method to get any variable was to download and extract information from a grib file. This requires you to use the ECMWF's *ecCodes* which doesn't work on Windows. To get around this the OpenDAP version of the forecast is used and a custom decoder reads the downloaded files.

Previous Python projects that attempted this do not fulfil all the requirements, mainly being an importable library. Acknowledgment must be made to albertotb's project get-gfs for providing the first foothold along the way.

3.1 Requirements

The required libraries (installed by PyPi) are:

`scipy, requests, fuzzywuzzy, numpy, python_dateutil, regex`

I have tried to ensure that these are well maintained and work across platforms (as this was the motive for writing this library).

CHAPTER

FOUR

CONTRIBUTING

Please see contributing for more information.

4.1 Todo

- Add historical forecasts from https://www.ncei.noaa.gov/thredds/dodsC/model-gfs-004-files-old/202003/20200328/gfs_4_20200328_1800_384.grb2.das
- Add export to .nc file with netcdf4 (maybe an optional dependency)
- Add purge missing/unreliable (missing/unreliable fill values are provided but have to iterate through data to check probably)

4.1.1 getgfs package

Submodules

getgfs.decode module

Decodes the downloaded files to extract the variables and their coordinates

class getgfs.decode.**Coordinate**(*name*, *values*)

Bases: object

Holds the information and values describing a coordinate

class getgfs.decode.**File**(*text*)

Bases: object

Holds the variables and information from a text file returned by the forecast site

class getgfs.decode.**Variable**(*name*, *coords*, *data*)

Bases: object

Holds the information and data for an extracted variable

getgfs.decode.**replace_val**(*arr*, *val*, *position*)

Inserts a value into a 1 to 4 dimensional numpy array

Note: I am sure there are better ways todo this but I couldn't find any after quite a search

Args:
arr (numpy array): Array to insert into val (float/int): Value to insert position (tuple): Coordinate position in array

Raises:
TypeError: Position invalid ValueError: Dimensionality of array too high

Returns:
[type]: [description]

---

getgfs.getgfs module

getgfs - a library for extracting weather forecast variables from the NOAA GFS forecast in a pure python, no obscure dependencies way

`class getgfs.getgfs.Forecast(resolution='0p25', timestep='')`

Bases: `object`

Object that can be manipulated to get forecast information

`check_avail(forecast_date, forecast_time)`

`datetime_to_forecast(date_time)`

Works out which forecast date/run/time is required for the latest values for a chosen time

Parameters
date_time (*string*) – The date and time of the desired forecast, parser is used so any format is valid e.g. 20210205 11pm

Raises
ValueError – The date time requested is not available from the NOAA at this time

Returns
forecast date string: forecast run string: forecast query time (the appropriate timestep within the forecast)

Return type
string

`get(variables, date_time, lat, lon)`

Returns the latest forecast available for the requested date and time

---

Note:

- "raw" since you have to put indexes in rather than coordinates and it returns a file object rather than a processed file
- If a variable has level dependance, you get all the levels - it seems extremely hard to impliment otherwise

Args:
variables (list): list of required variables by short name date_time (string): datetime requested (parser used so any format fine) lat (string or number): latitude in the format "[min:max]" or a single value lon (string or number): longitude in the format "[min:max]" or a single value

Raises:
ValueError: Invalid variable choice ValueError: Level dependance needs to be specified for chosen variable Exception: Unknown failure to download the file

Returns:
File Object: File object with the downloaded variable data (see File documentation)

`get_windprofile`(*date_time*, *lat*, *lon*)

Finds the verticle wind profile for a location. I wrote this since it is what I require in another program. The U/V compoents of wind with sigma do not go down to the surface so the surface components are also included as well as a pressure altitude change of x- variable.

Parameters

- **date_time** (*string*) – datetime requested (parser used so any format fine)
- **lat** (*string or number*) – Latitude for data
- **lon** (*string or number*) – Longitude for data

Returns
U component of wind interpolater by altitude interpolation object: V component of wind interpolater by altitude

Return type
interpolation object

`search`(*variable*, *sensetivity=80*)

The short names of the forecast variables are nonsence so this can be used to find the short name (used for the rest of the forecast) that you are looking for

---

Note: Will not work if fuzzywuzzy is not installed

Args:
variable (string): Search terms for the variable (e.g. U-Component wind) sensetivity (int, optional): The search sensitivity, for common words a higher sensitivity may be required. Defaults to 80.

Raises:
RuntimeError: Fuzzywuzzy not installed

Returns:
list: List of possible matches sorted by ratio, short name (what you need) and long name also given

---

`value_input_to_index`(*coord*, *inpt*)

Turns a chosen value of a coordinate/coordinate range to the index in the forecast array

Parameters

- **coord** (*string*) – The short name of the coordinate to convert
- **inpt** (*float/str*) – The value or range requested, for a range a string in the format [min_val:max_val] is required

Raises
ValueError – Incorrect inpt format

Returns
Index of coordinate value(s) as the forecast requires it (i.e. within [])

Return type
str

value_to_index(*coord*, *value*)

Turns a coordinate value into the index in the forecast array

Parameters

- **coord** (*string*) – The short name of the coordinate to convert
- **inpt** (*float/str*) – The value requested

Returns
Index in array

Return type
int

getgfs.getgfs.**extract_line**(*possibles*, *line*)

Works out what is being refered to by a line in the das and dds pages for the forecast

Parameters

- **possibles** (*list*) – Possible titles
- **line** (*string*) – Line to search

Returns
Index of the attribute

Return type
int

getgfs.getgfs.**get_attributes**(*res*, *step*)

Finds the available variables and coordinates for a given forecast

Parameters

- **res** (*str, optional*) – The forecast resulution, choices are 1p00, 0p50 and 0p25. Defaults to "0p25".
- **step** (*str, optional*) – The timestep of the forecast to use, most do not have a choice but 0p25 can be 3hr (default) or 1hr. Defaults to "".

Raises

- **Exception** – Failed to download the requested resolution and forecast
- **RuntimeError** – Failed to download the other attributes

Returns
Time attributes (the number of timesteps and the size of the timesteps) dict: Coordinates for the forecast with their short name, number of steps, min, max, resolution dict: Variables with all the information about them

Return type
dict

getgfs.getgfs.**hour_round**(*t*)

Rounds to the nearest hour for a datetime object

Parameters
t (*datetime*) – Datetime to round

Returns
Rounded datetime

Return type
datetime

getgfs.test module

class getgfs.test.**Decode**(*methodName='runTest'*)

Bases: TestCase

test_data()

test_variables()

class getgfs.test.**TestBasics**(*methodName='runTest'*)

Bases: TestCase

test_attribute()

test_folders()

Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX