ERA5: compute pressure and geopotential on model levels, geopotential height and geometric height

Last modified on Dec 05, 2023 12:35

Table of Contents

- Introduction
- Pressure on model levels
  - Illustrations of model levels, model half levels and model layers
- Geopotential on model levels
  - Prerequisites to calculating Geopotential on model levels
  - Step 1: Download input data
  - Step 2: Compute geopotential on model levels
- Interpolation of variables from model levels to custom pressure levels
  - Prerequisites for interpolating variables on model levels to custom pressure levels
  - Step 1: Download input data
  - Step 2: Interpolate variables on model levels to custom pressure levels
- Geopotential height
- Geometric height
- Related articles

Introduction

The following procedures describe how to compute the pressure and geopotential on model levels, geopotential height and geometric height.

Pressure on model levels

In ERA5, pressure is provided at the surface, but not on individual model levels. However, an illustration of pressure on model levels (p_ml) is shown in Figure 1, and the pressure can be computed for particular dates and times using the procedure described below.

You will need the following Inputs:

- logarithm of surface pressure (lnsp)

Example to download ERA5 lnsp data for a given area at a regular lat/lon grid in NetCDF format

```
#!/usr/bin/env python
import cdsapi
c = cdsapi.Client()
c.retrieve('reanalysis-era5-complete', { # Requests follow MARS syntax
                                         # Keywords 'expver' and 'class' can be dropped. They are
obsolete
                                         # since their values are imposed by 'reanalysis-era5-complete'
    'date'    : '2013-01-01',            # The hyphens can be omitted
    'levelist': '1',                     # 1 is top level, 137 the lowest model level in ERA5. Use '/'
to separate values.
    'levtype' : 'ml',
    'param'   : '152',                   # Full information at https://apps.ecmwf.int/codes/grib/param-
db/
                                         # The native representation for temperature is spherical
harmonics
    'stream'  : 'oper',                  # Denotes ERA5. Ensemble members are selected by 'enda'
    'time'    : '00/to/23/by/6',         # You can drop :00:00 and use MARS short-hand notation, instead
of '00/06/12/18'
    'type'    : 'an',
    'area'    : '80/-50/-25/0',          # North, West, South, East. Default: global
    'grid'    : '1.0/1.0',               # Latitude/longitude. Default: spherical harmonics or reduced
Gaussian grid
    'format'  : 'netcdf',                # Output needs to be regular lat-lon, so only works in
combination with 'grid'!
}, 'ERA5-ml-lnsp-subarea.nc')            # Output file. Adapt as you wish.
```

- a(n) and b(n) coefficients defining the model levels; these are included in the GRIB header of each model level GRIB message and are also tabulated here.

The model half-level pressure (p_half), illustrated in Figure 2, is given by:

$$\text{p_half} = a + b \ast \text{sp}$$

where sp ($\text{sp} = e^{lnsp}$) is the surface pressure (and lnsp is it's natural logarithm).

The pressure on model levels (p_ml), illustrated in Figure 1, is given by the mean of the pressures on the model half levels immediately above and below (see Figure 2):

$$\text{p_ml} = \frac{\text{p_half_above} + \text{p_half_below}}{2}$$

This means that the pressure on model levels is in the middle of the layers defined by the model half levels (Figure 2).

For more details about the vertical discretisation in the ECMWF Integrated Forecasting System (IFS), please see Part-iii Dynamics and numerical procedures, section 2.2 and the FULL-POS documentation at:

http://www.umr-cnrm.fr/gmapdoc/spip.php?article157

Illustrations of model levels, model half levels and model layers

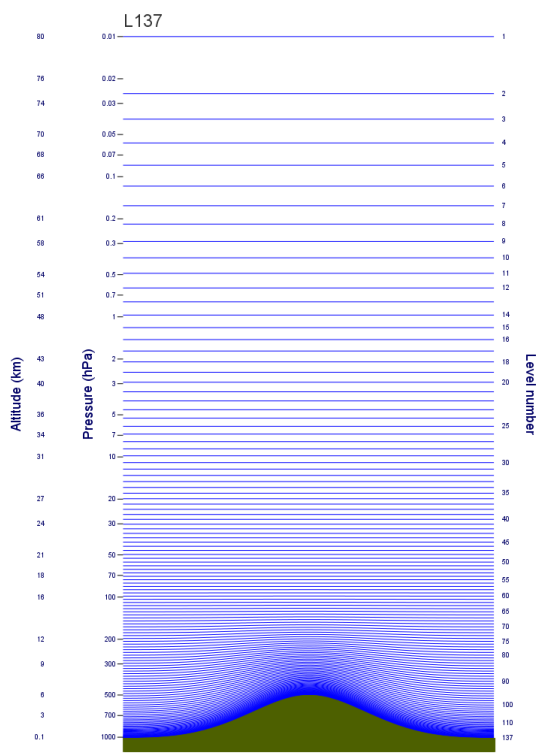

Figure 1. An illustration of IFS model levels, showing how they follow the terrain near the surface of the Earth. Level=1 is near the top of the atmosphere and Level=137 is near the surface of the Earth. The left hand axes are altitude (km) and pressure (hPa), while the right hand axis is level number.

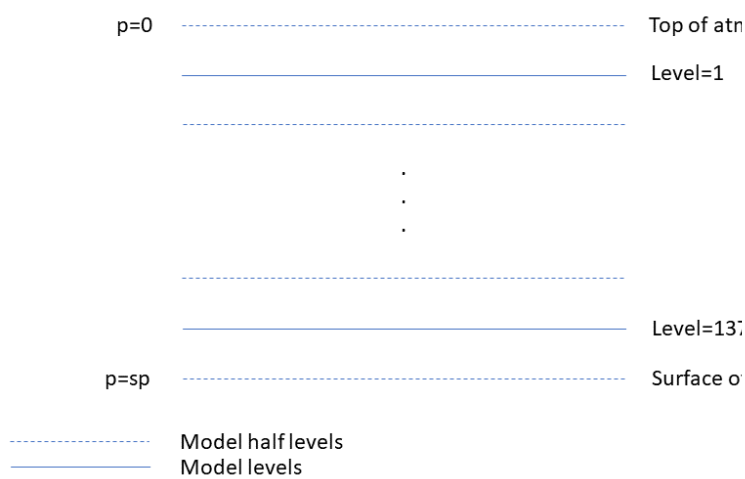

Figure 2. An illustration of IFS model levels, model half-levels and model layers. The on model levels is in the middle of the layers defined by the model half levels immediately above and below. The uppermost layer is adjacent to the top of the atmosphere (where p=0), while the lowest layer is adjacent to the surface of the Earth (where p=sp).

Geopotential on model levels

In ERA5, geopotential (z) is provided at the surface, but not on individual model levels. However, geopotential on model levels can be computed using the procedure described below.

Inputs:

- geopotential at the surface
- logarithm of surface pressure (lnsp)
- temperature and specific humidity on all the model levels

Output: Geopotential for each level, in m^2/s^2.

In the procedure below, the output data is written in GRIB format.

Please note, this procedure is an approximation to the calculation performed in the IFS (which also takes account of the effects of cloud ice and water and rain and snow).

Prerequisites to calculating Geopotential on model levels

You will need:

- A computer running Linux
- Python3
- The CDS API installed; Your computer must be set up for downloading ERA5 model level data (from the 'reanalysis-era5-complete' dataset, stored in ECMWF's MARS catalogue) through the CDS API. For more details, please follow the instructions here (step B).
- The ecCodes library to read and write data.

Step 1: Download input data

First we must retrieve the required ERA5 data. We need:

- Temperature (t) and specific humidity (q) on each model level.
- The logarithm of surface pressure (lnsp) and geopotential (z) on model level 1.

We use a Python script to download the ERA5 data from the MARS catalogue using the CDS API. The procedure is:

1. Copy the script below to a text editor on your computer
2. Edit the date, type, step, time, grid and area in the script to meet your requirements
3. Save the script (for example with the filename as 'get_data_geopotential_on_ml.py')
4. Run the script

Python script to download ERA5 data NOT listed in CDS through CDS API

```
#!/usr/bin/env python
import cdsapi
c = cdsapi.Client()

# data download specifications:
cls     = "ea"         # do not change
expver  = "1"          # do not change
levtype = "ml"         # do not change
stream  = "oper"       # do not change
date    = "2018-01-01" # date: Specify a single date as "2018-01-01" or a period as "2018-08-01/to/2018-01-31".
For periods > 1 month see https://confluence.ecmwf.int/x/l7GqB
tp      = "an"         # type: Use "an" (analysis) unless you have a particular reason to use "fc" (forecast).
time    = "00:00:00"   # time: ERA5 data is hourly. Specify a single time as "00:00:00", or a range as "00:00:00
/01:00:00/02:00:00" or "00:00:00/to/23:00:00/by/1".

c.retrieve('reanalysis-era5-complete', {
    'class'   : cls,
    'date'    : date,
    'expver'  : expver,
    'levelist': '1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19/20/21/22/23/24/25/26/27/28/29/30/31/32/33/34/35
/36/37/38/39/40/41/42/43/44/45/46/47/48/49/50/51/52/53/54/55/56/57/58/59/60/61/62/63/64/65/66/67/68/69/70/71/72
/73/74/75/76/77/78/79/80/81/82/83/84/85/86/87/88/89/90/91/92/93/94/95/96/97/98/99/100/101/102/103/104/105/106
/107/108/109/110/111/112/113/114/115/116/117/118/119/120/121/122/123/124/125/126/127/128/129/130/131/132/133/134
/135/136/137',         # For each of the 137 model levels
    'levtype' : 'ml',
    'param'   : '130/133', # Temperature (t) and specific humidity (q)
    'stream'  : stream,
    'time'    : time,
    'type'    : tp,
        'grid'    : [1.0, 1.0], # Latitude/longitude grid: east-west (longitude) and north-south resolution
(latitude). Default: 0.25 x 0.25
        'area'          : area, #example: [60, -10, 50, 2], # North, West, South, East. Default: global
}, 'tq_ml.grib')


c.retrieve('reanalysis-era5-complete', {
    'class'   : cls,
    'date'    : date,
    'expver'  : expver,
    'levelist': '1',       # Geopotential (z) and Logarithm of surface pressure (lnsp) are 2D fields, archived
as model level 1
    'levtype' : levtype,
    'param'   : '129/152', # Geopotential (z) and Logarithm of surface pressure (lnsp)
    'stream'  : stream,
    'time'    : time,
    'type'    : tp,
        'grid'    : [1.0, 1.0], # Latitude/longitude grid: east-west (longitude) and north-south resolution
(latitude). Default: 0.25 x 0.25
        'area'          : area, #example: [60, -10, 50, 2], # North, West, South, East. Default: global
}, 'zlnsp_ml.grib')
```

Running the script produces two new files in the current working directory:

- 'tq_ml.grib' (a GRIB file containing temperature and specific humidity)
- 'zlnsp_ml.grib' (a GRIB file containing geopotential and log of surface pressure).

Step 2: Compute geopotential on model levels

We then use a Python script to compute geopotential (z) for all model levels:

1. Copy the script below to a text editor
2. Save the script as 'compute_geopotential_on_ml.py'
3. Run the script 'compute_geopotential_on_ml.py' with the correct arguments, i.e. : `python compute_geopotential_on_ml.py tq_ml.grib zlnsp_ml.grib -o z_on_ml.grib`

```
#!/usr/bin/env python3
'''
Copyright 2023 ECMWF.

This software is licensed under the terms of the Apache Licence Version 2.0
which can be obtained at http://www.apache.org/licenses/LICENSE-2.0

In applying this licence, ECMWF does not waive the privileges and immunities
granted to it by virtue of its status as an intergovernmental organisation
nor does it submit to any jurisdiction.

*************************************************************************
Function      : compute_geopotential_on_ml

Author (date) : Cristian Simarro (09/10/2015)
modified:       Cristian Simarro (20/03/2017) - migrated to eccodes
                Xavi Abellan     (03/12/2018) - compatibilty with Python 3
                Xavi Abellan     (27/03/2020) - Corrected steps in output file
                Xavi Abellan     (05/08/2020) - Better handling of levels
                Xavi Abellan     (31/01/2023) - Better handling of steps

Category      : COMPUTATION

OneLineDesc   : Computes geopotential on model levels

Description   : Computes geopotential on model levels.
                Based on code from Nils Wedi, the IFS documentation:
                https://software.ecmwf.int/wiki/display/IFS/CY41R1+Official+IFS+Documentation
                part III. Dynamics and numerical procedures
                optimised implementation by Dominique Lucas.
                ported to Python by Cristian Simarro

Parameters    : tq.grib                - grib file with all the levelist
                                         of t and q
                zlnsp.grib             - grib file with levelist 1 for params
                                         z and lnsp
                -l levelist (optional) - slash '/' separated list of levelist
                                         to store in the output
                -o output   (optional) - name of the output file
                                         (default='z_out.grib')

Return Value  : output (default='z_out.grib')
                A fieldset of geopotential on model levels

Dependencies  : None

Example Usage :
                compute_geopotential_on_ml.py tq.grib zlnsp.grib
'''
from __future__ import print_function
import sys
import argparse
import numpy as np
from eccodes import (codes_index_new_from_file, codes_index_get, codes_get,
                     codes_index_select, codes_new_from_index, codes_set,
                     codes_index_add_file, codes_get_array, codes_get_values,
                     codes_index_release, codes_release, codes_set_values,
                     codes_write)
```

```
R_D = 287.06
R_G = 9.80665


def parse_args():
    ''' Parse program arguments using ArgumentParser'''
    parser = argparse.ArgumentParser(
        description='Python tool to calculate the Z of the model levels')
    parser.add_argument('-l', '--levelist', help='levelist to store',
                        default='all')
    parser.add_argument('-o', '--output', help='name of the output file',
                        default='z_out.grib')
    parser.add_argument('t_q', metavar='tq.grib', type=str,
                        help=('grib file with temperature(t) and humidity(q)'
                              'for the model levels'))
    parser.add_argument('z_lnsp', metavar='zlnsp.grib', type=str,
                        help=('grib file with geopotential(z) and Logarithm'
                              'of surface pressure(lnsp) for the ml=1'))
    args = parser.parse_args()
    # Handle levelist posibilities
    if args.levelist == 'all':
        args.levelist = range(1, 138)
    elif "to" in args.levelist.lower():
        if "by" in args.levelist.lower():
            args.levelist = args.levelist.split('/')
            args.levelist = list(range(int(args.levelist[0]),
                                       int(args.levelist[2]) + 1,
                                       int(args.levelist[4])))
        else:
            args.levelist = args.levelist.split('/')
            args.levelist = list(range(int(args.levelist[0]),
                                       int(args.levelist[2]) + 1))
    else:
        args.levelist = [int(l) for l in args.levelist.split('/')]
    return args


def main():
    '''Main function'''
    args = parse_args()

    print('Arguments: %s' % ", ".join(
        ['%s: %s' % (k, v) for k, v in vars(args).items()]))

    fout = open(args.output, 'wb')
    index_keys = ['date', 'time', 'shortName', 'level', 'step']

    idx = codes_index_new_from_file(args.z_lnsp, index_keys)
    codes_index_add_file(idx, args.t_q)
    if 'u_v' in args:
        codes_index_add_file(idx, args.u_v)
    values = None
    # iterate date
    for date in codes_index_get(idx, 'date'):
        codes_index_select(idx, 'date', date)
        # iterate time
        for time in codes_index_get(idx, 'time'):
            codes_index_select(idx, 'time', time)
            for step in codes_index_get(idx, 'step'):
```

```
                codes_index_select(idx, 'step', step)
                if not values:
                    values = get_initial_values(idx, keep_sample=True)
                if 'height' in args:
                    values['height'] = args.height
                    values['gh'] = args.height * R_G + values['z']
                if 'levelist' in args:
                    values['levelist'] = args.levelist
                    # surface pressure
                    try:
                        values['sp'] = get_surface_pressure(idx)
                        production_step(idx, step, values, fout)
                    except WrongStepError:
                        if step != '0':
                            raise

        try:
            codes_release(values['sample'])
        except KeyError:
            pass

    codes_index_release(idx)
    fout.close()


def get_initial_values(idx, keep_sample=False):
    '''Get the values of surface z, pv and number of levels '''
    codes_index_select(idx, 'level', 1)
    codes_index_select(idx, 'shortName', 'z')
    gid = codes_new_from_index(idx)

    values = {}
    # surface geopotential
    values['z'] = codes_get_values(gid)
    values['pv'] = codes_get_array(gid, 'pv')
    values['nlevels'] = codes_get(gid, 'NV', int) // 2 - 1
    check_max_level(idx, values)
    if keep_sample:
        values['sample'] = gid
    else:
        codes_release(gid)
    return values


def check_max_level(idx, values):
    '''Make sure we have all the levels required'''
    # how many levels are we computing?
    max_level = max(codes_index_get(idx, 'level', int))
    if max_level != values['nlevels']:
        print('%s [WARN] total levels should be: %d but it is %d' %
              (sys.argv[0], values['nlevels'], max_level),
              file=sys.stderr)
        values['nlevels'] = max_level


def get_surface_pressure(idx):
    '''Get the surface pressure for date-time-step'''
    codes_index_select(idx, 'level', 1)
    codes_index_select(idx, 'shortName', 'lnsp')
```

```
    gid = codes_new_from_index(idx)
    if gid is None:
        raise WrongStepError()
    if codes_get(gid, 'gridType', str) == 'sh':
        print('%s [ERROR] fields must be gridded, not spectral' % sys.argv[0],
              file=sys.stderr)
        sys.exit(1)
    # surface pressure
    sfc_p = np.exp(codes_get_values(gid))
    codes_release(gid)
    return sfc_p


def get_ph_levs(values, level):
    '''Return the presure at a given level and the next'''
    a_coef = values['pv'][0:values['nlevels'] + 1]
    b_coef = values['pv'][values['nlevels'] + 1:]
    ph_lev = a_coef[level - 1] + (b_coef[level - 1] * values['sp'])
    ph_levplusone = a_coef[level] + (b_coef[level] * values['sp'])
    return ph_lev, ph_levplusone


def compute_z_level(idx, lev, values, z_h):
    '''Compute z at half & full level for the given level, based on t/q/sp'''
    # select the levelist and retrieve the vaules of t and q
    # t_level: values for t
    # q_level: values for q
    codes_index_select(idx, 'level', lev)
    codes_index_select(idx, 'shortName', 't')
    gid = codes_new_from_index(idx)
    if gid is None:
        raise MissingLevelError('T at level {} missing from input'.format(lev))
    t_level = codes_get_values(gid)
    codes_release(gid)
    codes_index_select(idx, 'shortName', 'q')
    gid = codes_new_from_index(idx)
    if gid is None:
        raise MissingLevelError('Q at level {} missing from input'.format(lev))
    q_level = codes_get_values(gid)
    codes_release(gid)

    # compute moist temperature
    t_level = t_level * (1. + 0.609133 * q_level)

    # compute the pressures (on half-levels)
    ph_lev, ph_levplusone = get_ph_levs(values, lev)

    if lev == 1:
        dlog_p = np.log(ph_levplusone / 0.1)
        alpha = np.log(2)
    else:
        dlog_p = np.log(ph_levplusone / ph_lev)
        alpha = 1. - ((ph_lev / (ph_levplusone - ph_lev)) * dlog_p)

    t_level = t_level * R_D

    # z_f is the geopotential of this full level
    # integrate from previous (lower) half-level z_h to the
    # full level
```

```
    z_f = z_h + (t_level * alpha)

    # z_h is the geopotential of 'half-levels'
    # integrate z_h to next half level
    z_h = z_h + (t_level * dlog_p)

    return z_h, z_f


def production_step(idx, step, values, fout):
    '''Compute z at half & full level for the given level, based on t/q/sp'''
    # We want to integrate up into the atmosphere, starting at the
    # ground so we start at the lowest level (highest number) and
    # keep accumulating the height as we go.
    # See the IFS documentation, part III
    # For speed and file I/O, we perform the computations with
    # numpy vectors instead of fieldsets.

    z_h = values['z']
    codes_set(values['sample'], 'step', int(step))

    for lev in sorted(values['levelist'], reverse=True):
        try:
            z_h, z_f = compute_z_level(idx, lev, values, z_h)
            # store the result (z_f) in a field and add to the output
            codes_set(values['sample'], 'level', lev)
            codes_set_values(values['sample'], z_f)
            codes_write(values['sample'], fout)
        except MissingLevelError as e:
            print('%s [WARN] %s' % (sys.argv[0], e),
                  file=sys.stderr)


class WrongStepError(Exception):
    ''' Exception capturing wrong step'''
    pass


class MissingLevelError(Exception):
    ''' Exception capturing missing levels in input'''
    pass


if __name__ == '__main__':
    main()
```

This script is from ECMWF's generic article Compute geopotential on model levels .

Alternatively, there is a customer-supplied script (which runs on Microsoft Windows) that computes geopotential on model levels for a specific location. This script was written for the ERA-Interim dataset, but can be adapted to ERA5. Please see the article ERA-Interim: compute geopotential on model levels for details.

For users experienced in Metview, there is a built-in function called mvl_geopotential_on_ml.

Interpolation of variables from model levels to custom pressure levels

The procedure described below is to convert ERA5 model levels data to custom pressure levels data.

Input:

- variable(s) on model levels and related logarithm surface pressure in grib format
- list of custom pressure levels required for interpolation to.

Output: NetCDF file containing variable(s) at each custom pressure level

Prerequisites for interpolating variables on model levels to custom pressure levels

You will need:

- Python3
- The CDS API **installed**. For more details, please follow the instructions here.
- The ecCodes library to read and write data.

Step 1: Download input data

First the required ERA5 variable(s) on model levels data are downloaded. The suggested procedure is:

1. Copy the script below to a text editor on your computer
2. Edit the date, type, step, time and grid in the script to meet your requirements. Note the 'area' keyword can also be used. The output filename can be modified accordingly.
3. Save the script (for example with the filename as 'get_data.py')
4. Run the script i.e. `python3 get_data.py`

get_data

```
# ************************** LICENSE START **********************************
#
# Copyright 2022 ECMWF. This software is distributed under the terms
# of the Apache License version 2.0. In applying this license, ECMWF does not
# waive the privileges and immunities granted to it by virtue of its status as
# an Intergovernmental Organization or submit itself to any jurisdiction.
#
# *************************** LICENSE END ***********************************
import cdsapi

c = cdsapi.Client()

c.retrieve('reanalysis-era5-complete', {

    'class': 'ea',

    'date': '2021-01-01',

    'expver': '1',

    'levelist':'1/to/137',

    'levtype': 'ml',

    'param': '130/152',

    'step': '0',

    'stream': 'oper',

    'time': '00/to/06/by/1',

    'type': 'an',

    'grid': '1.0/1.0'

}, 'output_00_06_130_152_1x1.grib')
```

Running the script produces a file in the current working directory called 'output_00_06_130_152_1x1.grib' (a GRIB file containing the ERA5 variables needed.).

Step 2: Interpolate variables on model levels to custom pressure levels

The suggested procedure to run the Python script to compute the conversion of the variable from model levels to the custom pressure level is:

1. Copy the script below to a text editor
2. Save the script as 'conversion_from_ml_to_pl.py'
3. Run the script 'conversion_from_ml_to_pl.py' with the correct arguments, i.e. : `python3 conversion_from_ml_to_pl.py -p 70000 -o output.nc -i output_00_06_130_152_1x1.grib`

conversion_from_ml_to_pl.py

```
# ***************************** LICENSE START ************************************
#
# Copyright 2022 ECMWF. This software is distributed under the terms
# of the Apache License version 2.0. In applying this license, ECMWF does not
# waive the privileges and immunities granted to it by virtue of its status as
# an Intergovernmental Organization or submit itself to any jurisdiction.
#
# ***************************** LICENSE END ************************************


import cfgrib
import xarray as xr
import numpy as np
from eccodes import *
import matplotlib.pyplot as plt
import argparse
import sys
import os

def parse_args():
    ''' Parse program arguments using ArgumentParser'''
    parser = argparse.ArgumentParser(description ="Python tool to calculate the model level variable at a given
pressure level and write data to a netCDF file")
    parser.add_argument('-p', '--pressure', required=True, nargs='+',type=float,
                        help='Pressure levels (Pa) to calculate the variable')
    parser.add_argument('-o', '--output', required=False, help='name of the output file (default "output.nc"')
    parser.add_argument('-i', '--input', required=True, metavar='input.grib', type=str,
                        help=('grib file with required variable(s) on model level and surface pressure fields',
                              'the model levels'))
    args = parser.parse_args()
    if not args.output:
        args.output = 'output.nc'
    return args

def get_input_variable_list(fin):
    f = open(fin)
    var_list = []
    while 1:
        gid = codes_grib_new_from_file(f)
        if gid is None:
            break
        keys = ('dataDate', 'dataTime', 'shortName')
        for key in keys:
            if key == 'shortName':
              var_list.append(codes_get(gid, key))
        codes_release(gid)
    var_list_unique = list(set(var_list))
    f.close()
    if 'lnsp' not in var_list_unique:
      print("Error - lnsp variable missing from input file -exiting")
      sys.exit()
    if len(var_list_unique) < 2:
      print("Error - Data variable missing from input file -exiting")
      sys.exit()
    return var_list_unique

def check_requested_levels(plevs):
    check_lev = True
    if len(plevs) > 1:
        error_msg = "Error - only specify 1 input pressure level to interpolate to"
    else:
        for lev in plevs:
           if lev < 0 or lev > 110000 :
```

```
            check_lev = False
            error_msg = "Error - negative values and large positive values for pressure are not allowed -
exiting"
    if check_lev == False:
        print(error_msg)
        sys.exit()
    return check_lev

def check_in_range(data_array,requested_levels):
    amin = data_array.minimum()
    amax = data_array.maximum()
    print("min max ",amin,amax)

def vertical_interpolate(vcoord_data, interp_var, interp_level):
    """A function to interpolate sounding data from each station to
    every millibar. Assumes a log-linear relationship.

    Input
    -----
    vcoord_data : A 1D array of vertical level values (e.g. from ERA5 pressure at model levels at a point)
    interp_var : A 1D array of the variable to be interpolated to the  pressure level
    interp_level : A 1D array containing the vertical level to interpolate to

    Return
    ------
    interp_data : A 1D array that contains the interpolated variable on the interp_level
    """

    l_count = 0
    for l in interp_level:
      if l < np.min(vcoord_data) or l > np.max(vcoord_data):
        ip = [np.NAN]
      else:
    # Make vertical coordinate data and grid level log variables
        lnp = np.log(vcoord_data)
        lnp_interval = [np.log(x) for x in interp_level]
    # Use numpy to interpolate from observed levels to grid levels
        ip = np.interp(lnp_interval, lnp, interp_var)
    return ip[0]

  def calculate_pressure_on_model_levels(ds_var,ds_lnsp):
    # Get the number of model levels in the input variable
    nlevs=ds_var.sizes['hybrid']
    # Get the a and b coefficients from the pv array to calculate the model level pressure
    pv_coeff = np.array(ds_var.GRIB_pv)
    pv_coeff=pv_coeff.reshape(2,nlevs+1)
    a_coeff=pv_coeff[0,:]
    b_coeff=pv_coeff[1,:]
    # get the surface pressure in hPa
    sp = np.exp(ds_lnsp)
    p_half=[]
    for i in range(len(a_coeff)):
        p_half.append(a_coeff[i] + b_coeff[i] * sp)
    p_ml=[]
    for hybrid in range(len(p_half) - 1):
        p_ml.append((p_half[hybrid + 1] + p_half[hybrid]) / 2.0)
    ds_p_ml = xr.concat(p_ml, 'hybrid')
    return ds_p_ml

def plot_profile(var_ml,press_ml, var_int_press,var_int_plevs,tstep,lat,lon):

    var_v= var_ml.sel(time = var_ml.time[tstep],longitude=lon, latitude=lat, method='nearest')
    var_v_values = var_v.values
    var_p= press_ml.sel(time = var_ml.time[tstep],longitude=lon, latitude=lat, method='nearest')
    var_p_values = var_p.values
    var_ip= var_int_press.sel(time = var_ml.time[tstep],longitude=lon, latitude=lat, method='nearest')
    var_ip_values = var_ip.values
    var_ip_p = var_ip.pressure
    var_ip_p_values = var_ip_p.values
    plt.axis([min(var_v_values), max(var_v_values), max(var_p_values), min(var_p_values)])
    plt.plot(var_v_values,var_p_values, 'o', color = 'black')
```

```
    plt.plot(var_ip_values,var_ip_p_values,'o', color = 'red')
    plt.show()
    return

def calculate_interpolated_pressure_field(data_var_on_ml, data_p_on_ml,plevs):
    nlevs = len(data_var_on_ml.hybrid)
    p_array = np.stack(data_p_on_ml, axis=2).flatten()
    # Flatten the data array to enable faster processing
    var_array = np.stack(data_var_on_ml, axis=2).flatten()
    no_grid_points =  int(len(var_array)/nlevs)
    interpolated_var = np.empty((len(plevs), no_grid_points))
    ds_shape = data_var_on_ml.shape
    nlats_values = data_var_on_ml.coords['latitude']
    nlons_values = data_var_on_ml.coords['longitude']
    nlats = len(nlats_values)
    nlons = len(nlons_values)

#     Iterate over the data, selecting one vertical profile at a time
    count = 0
    profile_count = 0
    interpolated_values=[]
    for point in range(no_grid_points):
        offset =  count*nlevs
        var_profile = var_array[offset:offset+nlevs]
        p_profile = p_array[offset:offset+nlevs]
        interpolated_values.append(vertical_interpolate(p_profile, var_profile, plevs))
        profile_count += len(p_profile)
        count = count + 1
    interpolated_field=np.asarray(interpolated_values).reshape(len(plevs),nlats,nlons)
    return interpolated_field

def check_data_cube(dc):
    checks = True
    for var_name in dc.variables:
        if var_name in ['time','step','hybrid','latitude','longitude','valid_time']:
            continue
        if var_name == 'lnsp':
            lnsp_dims = ['time','latitude','longitude']
            if all(value in lnsp_dims for value in dc.variables[var_name].dims):
                continue
            else:
                print("Not all required lnsp dimensions found -exiting ", dc.variables[var_name].dims)
                checks = False
        else:
            var_dims = ['time','hybrid','latitude','longitude']
            if all(value in var_dims for value in dc.variables[var_name].dims):
                continue
            else:
                print("Not all required variable dimensions found -exiting ",dc.variables[var_name].dims)
                checks = False
            continue
    return checks

def main():
    '''Main function'''
    print("-p <pressure level (Pa) > -o <output_file> -i <input grib file>")
    print("e.g. to process a grib file containing 6 hours of lnsp and temperature data to the 500 hPa level:")
    print("python3 script.py -o output_press.nc -p 50000  -i output_00_06_130_152_1x1.grib`n")
    args = parse_args()

    print('Arguments: %s' % ", ".join(
        ['%s: %s' % (k, v) for k, v in vars(args).items()]))

    plevels = args.pressure
    plevels.sort(reverse = True)

    check_requested_levels(plevels)

    input_fname = args.input
    output_fname = args.output
    if not os.path.isfile(input_fname):
```

```
        print("Input file does not exist - exiting")
        sys.exit()
    variable_list = get_input_variable_list(input_fname)
    # Create a data object to hold the input and derived data
    data_cube = xr.merge(cfgrib.open_datasets(input_fname, backend_kwargs={'read_keys': ['pv']}),
combine_attrs='override')
    if not check_data_cube(data_cube):
        sys.exit()
   # Get the ln surface pressure
    lnsp = data_cube['lnsp']
    for var in variable_list:
      if var == 'lnsp':
          continue
      else:
          data_cube['pml']=data_cube[var].copy()
          break
    for var in variable_list:
        if var == 'lnsp' :
            continue
        data_pressure_on_model_levels_list =[]
        for time_step in range(len(data_cube[var].time)):
            data_slice_var=data_cube[var][time_step,:,:,:]
            data_slice_lnsp=data_cube['lnsp'][time_step,:,:]
#   Get the pressure field on model levels for each timestep
            data_cube['pml'][time_step,:,:,:] = calculate_pressure_on_model_levels(data_slice_var,
data_slice_lnsp)
    data_cube['pml'].attrs = {'units' :
'Pa','long_name':'pressure','standard_name':'air_pressure','positive':'down'}
    all_interpolated_var_fields_list = []
    for var in variable_list:
        if var == 'lnsp' or var == 'pml':
            continue
        interpolated_var_field = data_cube[var].copy()
        interpolated_var_field = interpolated_var_field[:,0:len(plevels),:,:]
        interpolated_var_field = interpolated_var_field.rename({'hybrid':'pressure'})
        interpolated_var_field['pressure'] = plevels
        for time_step in range(len(data_cube[var].time)):
            var_on_ml = data_cube[var][time_step,:,:,:]
            p_on_ml = data_cube['pml'][time_step,:,:,:]
            interpolated_var_field[time_step,:,:,:] = calculate_interpolated_pressure_field(var_on_ml,p_on_ml,
plevels)
        all_interpolated_var_fields_list.append(interpolated_var_field)
    all_interpolated_var_fields = xr.merge(all_interpolated_var_fields_list)
    all_interpolated_var_fields['pressure'].attrs = {'units' :
'Pa','long_name':'pressure','standard_name':'air_pressure','positive':'down'}
    all_interpolated_var_fields.to_netcdf(output_fname)
#   Write interpolated data variable to output filename
    PLOT_DATA = False
    if PLOT_DATA:
        latitude = 45.0
        longitude = 0
        tstep =0
        plot_profile(data_cube[var],data_cube['pml'],interpolated_var_field,plevels,tstep,latitude,longitude)
    print("Finished interpolation of variables to pressure level")

if __name__ == '__main__':
    main()
```

This produces a netCDF file called 'output.nc' in the current directory containing the interpolated data.

Geopotential height

In ERA5 (and in the IFS), and often in meteorology, heights (the height of the land and sea surface, or specific heights in the atmosphere) are not represented as geometric height, or altitude (in metres above the spheroid), but as geopotential height (in geopotential metres above the geoid). Note, that ECMWF usually archives the geopotential (in m^2/s^2), not the geopotential height.

To obtain the geopotential height (h) in (geopotential) metres (of the land and sea surface or at particular heights in the atmosphere), simply divide the geopotential by the Earth's gravitational acceleration, which has a fixed value of 9.80665 m/s^2 in the IFS. This geopotential height is relative to the geoid (over ocean, mean sea level is assumed to be coincident with the geoid) - for more information see ERA5: data documentation - spatial reference systems.

Geometric height

Geometric height is not represented in ERA5 (nor in the IFS). However, the geometric height or altitude (alt) can be approximated by the following formula (neglecting horizontal variations in the Earth's gravitational acceleration):

$$alt = Re \ast h/(Re-h)$$

where Re is the radius of the Earth, and h is the Geopotential height. This geometric height is relative to the geoid (over ocean, mean sea level is assumed to be coincident with the geoid) and it is assumed that the Earth is a perfect sphere - for more information see ERA5: data documentation - spatial reference systems.

Related articles

- ERA5: data documentation
- ERA5: How to calculate wind speed and wind direction from u and v components of the wind?
- Parameters valid at the specified time
- Convective and large-scale precipitation
- Model grid box and time step