ERA-Interim: compute geopotential on model levels

Last modified on Nov 06, 2023 15:42

ERA-Interim production stopped on 31st August 2019

For ERA-Interim (1st January 1979 to 31st August 2019) access through the ECMWF Web API **stopped on 01 June 2023**

Its successor ERA5 is available from the Climate Data Store (CDS) (What are the changes from ERA-Interim to ERA5?) and users are strongly advised to migrate to ERA5 (How to download ERA5).

For those users who still need access to ERA-Interim after 01 June 2023 (subject to further notice), they can do so via the Climate Data Store (CDS) API.

Table of Contents

Introduction

Some users are interested on geopotential (z) of the different model levels (ml). ECMWF provides two tools for this, a MetView macro and a Python script, which are the recommended methods, but only work on Linux, and output geopotential as an area, not for a specific location.

One of our customers, Mark Jackson from Cambridge Environmental Research Consultants (CERC), wanted to calculate geopotential and height above the surface for model levels, and this for one particular point location.

So Mark wrote his own script and kindly provided it to us. The script calculates the geopotential in m^2/s^2 on each model level for a single point location. It then also calculates the height in meters by dividing the geopotential by the gravity of Earth (9.80665 m/s^2).

This is a two step process: first you have to obtain the required input data, then you perform the actual geopotential and height calculation.

Notes:

- All data is in NetCDF format
- The computation script requires Python; the input data script requires Python and the ECMWF WebAPI to access ECMWF public datasets
- The script only works correctly for ECMWF ERA-Interim data, do not use it with other datasets
- Input data has to be gridded, not spectral
- In the computation script, paths and other arguments are hard-coded, so you will need to adapt the script to your system

Step1: Get data

The first script downloads ERA-Interim data from ECMWF through the ECMWF Web API:

- Temperature (t) and specific humidity (q), both on each model level, as file 'tq_ml.nc'.
- The log of surface pressure (lnsp) and geopotential (z), both on model level 1, as file 'zlnsp_ml.nc'.

The Python script:

EI_geopotential_on_ml_getdata.py

```
#!/usr/bin/env python
from ecmwfapi import ECMWFDataServer
server = ECMWFDataServer()
server.retrieve({
    "class": "ei",
    "dataset": "interim",
    "expver": "1",
    "levelist": "all",
    "levtype": "ml",
    "param": "t/q",
    "stream": "oper",
    "date": "2015-01-01",    #date: Specify a single date as "2015-08-01" or a period as "2015-08-01/to/2015-08-
31".
    "type": "an",        #type: Use an (analysis) unless you have a particular reason to use fc (forecast).
    "time": "00:00:00",        #time: With type=an, time can be any of "00:00:00/06:00:00/12:00:00/18:00:00".
With type=fc, time can be any of "00:00:00/12:00:00",
    "step": "0",        #step: With type=an, step is always "0". With type=fc, step can be any of "3/6/9/12".
    "grid": "0.75/0.75",    #grid: Only regular lat/lon grids are supported.
    "area": "75/-20/10/60",    #area: N/W/S/E, here we have Europe.
    "format": "netcdf",
    "target": "tq_ml.nc",    #target: the name of the output file.
})
server.retrieve({
    "class": "ei",
    "dataset": "interim",
    "expver": "1",
    "levelist": "1",
    "levtype": "ml",
    "param": "z/lnsp",
    "stream": "oper",
    "date": "2015-01-01",
    "type": "an",
    "time": "00:00:00",
    "step": "0",
    "grid": "0.75/0.75",
    "area": "75/-20/10/60",
    "format": "netcdf",
    "target": "zlnsp_ml.nc",
})
```

Copy the script and save it to your computer.

You can change date, type, step, time, grid and area in the script, but make sure you use the same values in both 'execute' blocks so that the two output files are synchronized. Later the calculation of geopotential will iterate through the date/time/step parameters, calculating values for multiple times.

Run the script.

Outputs: A file 'tq_ml.nc' and a file 'zlnsp_ml.nc', both in the current working directory.

Step2: Compute geopotential on model levels

This script was kindly provided by Mark Jackson from Cambridge Environmental Research Consultants Ltd. **The script is provided 'as is' and is not supported by ECMWF/Copernicus or by CERC.**

The Python script:

EI_geopotential_on_ml_compute.py

```
# Copyright 2016 Cambridge Environmental Research Consultants Ltd.
#
# 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
#
# ******************************************************************
# Function      : compute_geopotential_on_ml_netcdf
#
```

```
# Author (date) : Mark Jackson (1/12/2016)
#
# Category      : COMPUTATION
#
# OneLineDesc   : Computes geopotential and height on model levels using netCDF files
#
# Description   : Computes geopotential on model levels using netCDF files.
#                 Based on the Python script by Cristian Simarro which uses GRIB files:
#                   https://confluence.ecmwf.int/display/GRIB/Compute+geopotential+on+model+levels
#                 Which was based on the code of the Metview function mvl_geopotential_on_ml:
#                   https://confluence.ecmwf.int/metview/mvl_geopotential_on_ml
#                 This in turn was based on code from Nils Wedi, the IFS documentation:
#                   https://confluence.ecmwf.int/display/IFS/CY41R1+Official+IFS+Documentation
#                 part III. Dynamics and numerical procedures
#                 and an optimised implementation by Dominique Lucas.
#                 Ported to Python by Cristian Simarro
#
# Parameters    : FileA.nc  - netCDF file with the levelist of t and q. Does not require all levels
#                             but does require a contiguous set of levels all the way to the bottom.
#                 FileB.nc  - netCDF file with levelist 1 for params z and lnsp
#
# Return Value  : outputs CSV files
#                 with geopotential and height (relative to terrain) on each model level.
#
# Dependencies  : netCDF4, numpy, scipy
#


from __future__ import print_function # make sure print behaves the same in Python 2.7 and 3.x
import netCDF4
from netCDF4 import num2date
import numpy as np
from scipy import interpolate
import datetime
import sys
import io
import math

#Variable names in the netCDF
#(File A - model levels)
# level = model level numbers, possible values 1-60
# t = temperature K
# q = specific humidity kg/kg
#(File B)
# lnsp = log surface pressure
# z = surface geopotential

#arguments
#ONEDAY - read these from the command line
FILE_A_PATH="G:\\MiscProjects\\ERA-Interim-Python\\20160922-InvestigateProfiles-netCDF\\FileA.nc"
FILE_B_PATH="G:\\MiscProjects\\ERA-Interim-Python\\20160922-InvestigateProfiles-netCDF\\FileB.nc"
GRID_LAT=54.75   #(degrees N)
GRID_LONG=-4.5    #(degrees E)
OUT_DIR_PATH="G:\\MiscProjects\\ERA-Interim-Python\\20161101-Check MarkJ geopotentials against ECMWF
geopotentials\\output\\"



#Routine to Read File A data file for t, q values at a particular grid point
def readfa(fileapath):
    #Connect to data file for reading
    print()
    print("================================================")
    print("File A information")
    print("Filename{}".format(fileapath))
    fa = netCDF4.Dataset(fileapath, 'r')

    #Variables as netCDF variable objects
    print()
    print("------------------------------------------------")
    print("Variables")
```

```
    print(fa.variables.keys()) # get all variable names

    fanclongs = fa.variables['longitude']
    print(fanclongs)
    fanclats = fa.variables['latitude']
    print(fanclats)
    fanclevels = fa.variables['level']
    print(fanclevels)
    fanctimes = fa.variables['time']
    print(fanctimes)
    fancts = fa.variables['t']
    print(fancts)
    fancqs = fa.variables['q']
    print(fancqs)

    #Get level values (either model levels or pressure levels) and number of levels
    falevels=fanclevels[:]
    print()
    print("-------------------------------------------------")
    print("Model levels")
    fanlevels=falevels.shape[0]
    print("There are {} levels: {} - {}".format(fanlevels, falevels[0], falevels[fanlevels-1]))

    #Get time values and number of times
    fatimes=fanctimes[:]
    print()
    print("-------------------------------------------------")
    print("File A Times")
    print(fatimes)
    fantimes=fatimes.shape[0]

    #Get python datetime for each time
    fapydts=num2date(fatimes, fanctimes.units)

    #Output first 10 datetimes
    print()
    print("First 10 times as date-time")
    print([pydt.strftime('%Y-%m-%d %H:%M:%S') for pydt in fapydts[:10]])

    #Get lat and long values
    falats = fanclats[:]
    print()
    print("-------------------------------------------------")
    print("File A Latitudes")
    print(falats)
    falongs = fanclongs[:]
    print()
    print("-------------------------------------------------")
    print("File A Longitudes")
    print(falongs)

    #Get index of grid point of interest
    failat=np.where(falats==GRID_LAT)[0]
    failong=np.where(falongs==GRID_LONG)[0]
    print()
    print("=================================================")
    print("Grid point location: latitude and longitude indexes for lat {} and long {}".format(GRID_LAT,
GRID_LONG))
    print(failat)
    print(failong)

    #Get t, q values for specified grid point for all levels (slicing)
    #The result is still a 4D array with 1 latitude and 1 longitude
    print()
    print("=================================================")
    print("Get t, q values for all levels")
    fats=fancts[range(fantimes),range(fanlevels),failat,failong]
    faqs=fancqs[range(fantimes),range(fanlevels),failat,failong]

    return (fats, faqs, falevels, fapydts)
```

```
#Routine to Read File B data file for z, lnsp values at a particular grid point
def readfb(fbfilepath):
    #Connect to file for reading
    print()
    print("================================================")
    print("fb File information")
    print("Filename{}".format(fbfilepath))
    fbf = netCDF4.Dataset(fbfilepath, 'r')

    print(fbf)
    print()

    #Variables as netCDF variable objects
    print()
    print("------------------------------------------------")
    print("Variables")
    print(fbf.variables.keys()) # get all variable names

    fbnclongs = fbf.variables['longitude']
    print(fbnclongs)
    fbnclats = fbf.variables['latitude']
    print(fbnclats)
    fbnctimes = fbf.variables['time']
    print(fbnctimes)
    fbnczs = fbf.variables['z']
    print(fbnczs)
    fbnclnsps = fbf.variables['lnsp']
    print(fbnclnsps)

    #Get time values and number of times
    fbtimes=fbnctimes[:]
    print()
    print("------------------------------------------------")
    print("fb Times")
    print(fbtimes)
    fbntimes=fbtimes.shape[0]

    #Get python datetime for each time
    fbpydts=num2date(fbtimes, fbnctimes.units)

    #Output first 10 datetimes
    print()
    print("fb First 10 times as date-time")
    print([pydt.strftime('%Y-%m-%d %H:%M:%S') for pydt in fbpydts[:10]])

    #Get lat and long values
    fblats = fbnclats[:]
    print()
    print("------------------------------------------------")
    print("fb Latitudes")
    print(fblats)
    fblongs = fbnclongs[:]
    print()
    print("------------------------------------------------")
    print("fb Longitudes")
    print(fblongs)


    #Get index of grid point of interest
    fbilat=np.where(fblats==GRID_LAT)[0]
    fbilong=np.where(fblongs==GRID_LONG)[0]
    print()
    print("================================================")
    print("Grid point location: fb latitude and longitude indexes for lat {} and long {}".format(GRID_LAT,
GRID_LONG))
    print(fbilat)
    print(fbilong)

    #Get z, lnsp values for specified grid point (slicing)
    #The result is still a 3D array with 1 latitude and 1 longitude
    print()
```

```
    print("================================================")
    print("Get z, lnsp values ")
    fbzs=fbnczs[range(fbntimes),fbilat,fbilong]
    print('shape of z slice: %s' % repr(fbzs.shape))
    fblnsps=fbnclnsps[range(fbntimes),fbilat,fbilong]

    return (fbzs, fblnsps, fbpydts)


#Read File A file
fats, faqs, falevels, fapydts = readfa(FILE_A_PATH)
fanlevels=falevels.shape[0]
fantimes=fapydts.shape[0]

#Read File B file
fbzs, fblnsps, fbpydts = readfb(FILE_B_PATH)


print()
print("================================================")
print("Running...")

#Calculation of geopotential and height
def calculategeoh(z, lnsp, ts, qs, levels):
    heighttoreturn=np.full(ts.shape[0], -999, np.double)
    geotoreturn=np.full(ts.shape[0], -999, np.double)

    Rd = 287.06

    z_h = 0

    #surface pressure
    sp = math.exp(lnsp)

    # A and B parameters to calculate pressures for model levels,
    #  extracted from an ECMWF ERA-Interim GRIB file and then hardcoded here
    pv =  [
      0.0000000000e+000, 2.0000000000e+001, 3.8425338745e+001, 6.3647796631e+001, 9.5636962891e+001,
      1.3448330688e+002, 1.8058435059e+002, 2.3477905273e+002, 2.9849584961e+002, 3.7397192383e+002,
      4.6461816406e+002, 5.7565112305e+002, 7.1321801758e+002, 8.8366040039e+002, 1.0948347168e+003,
      1.3564746094e+003, 1.6806403809e+003, 2.0822739258e+003, 2.5798886719e+003, 3.1964216309e+003,
      3.9602915039e+003, 4.9067070313e+003, 6.0180195313e+003, 7.3066328125e+003, 8.7650546875e+003,
      1.0376125000e+004, 1.2077445313e+004, 1.3775324219e+004, 1.5379804688e+004, 1.6819472656e+004,
      1.8045183594e+004, 1.9027695313e+004, 1.9755109375e+004, 2.0222203125e+004, 2.0429863281e+004,
      2.0384480469e+004, 2.0097402344e+004, 1.9584328125e+004, 1.8864750000e+004, 1.7961359375e+004,
      1.6899468750e+004, 1.5706449219e+004, 1.4411125000e+004, 1.3043218750e+004, 1.1632757813e+004,
      1.0209500000e+004, 8.8023554688e+003, 7.4388046875e+003, 6.1443164063e+003, 4.9417773438e+003,
      3.8509133301e+003, 2.8876965332e+003, 2.0637797852e+003, 1.3859125977e+003, 8.5536181641e+002,
      4.6733349609e+002, 2.1039389038e+002, 6.5889236450e+001, 7.3677425385e+000, 0.0000000000e+000,
      0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000,
      0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000,
      0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000,
      0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000,
      0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000, 0.0000000000e+000,
      7.5823496445e-005, 4.6139489859e-004, 1.8151560798e-003, 5.0811171532e-003, 1.1142909527e-002,
      2.0677875727e-002, 3.4121163189e-002, 5.1690407097e-002, 7.3533833027e-002, 9.9674701691e-002,
      1.3002252579e-001, 1.6438430548e-001, 2.0247590542e-001, 2.4393314123e-001, 2.8832298517e-001,
      3.3515489101e-001, 3.8389211893e-001, 4.3396294117e-001, 4.8477154970e-001, 5.3570991755e-001,
      5.8616840839e-001, 6.3554745913e-001, 6.8326860666e-001, 7.2878581285e-001, 7.7159661055e-001,
      8.1125342846e-001, 8.4737491608e-001, 8.7965691090e-001, 9.0788388252e-001, 9.3194031715e-001,
      9.5182150602e-001, 9.6764522791e-001, 9.7966271639e-001, 9.8827010393e-001, 9.9401944876e-001,
      9.9763011932e-001, 1.0000000000e+000 ]
    levelSize=60
    A = pv[0:levelSize+1]
    B = pv[levelSize+1:]

    Ph_levplusone = A[levelSize] + (B[levelSize]*sp)

    #Get a list of level numbers in reverse order
    reversedlevels=np.full(levels.shape[0], -999, np.int32)
    for iLev in list(reversed(range(levels.shape[0]))):
```

```
        reversedlevels[levels.shape[0] - 1 - iLev] = levels[iLev]

    #Integrate up into the atmosphere from lowest level
    for lev in reversedlevels:
        #lev is the level number 1-60, we need a corresponding index into ts and qs
        ilevel=np.where(levels==lev)[0]
        t_level=ts[ilevel]
        q_level=qs[ilevel]

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

        #compute the pressures (on half-levels)
        Ph_lev = A[lev-1] + (B[lev-1] * sp)

        if lev == 1:
            dlogP = math.log(Ph_levplusone/0.1)
            alpha = math.log(2)
        else:
            dlogP = math.log(Ph_levplusone/Ph_lev)
            dP    = Ph_levplusone-Ph_lev
            alpha = 1. - ((Ph_lev/dP)*dlogP)

        TRd = t_level*Rd

        # 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 + (TRd*alpha)

        #Convert geopotential to height
        heighttoreturn[ilevel] = z_f / 9.80665

        #Geopotential (add in surface geopotential)
        geotoreturn[ilevel] = z_f + z

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

        Ph_levplusone = Ph_lev

    return geotoreturn, heighttoreturn

#Output all the values for the specified grid point
#Iterate over all the times and write out the data
for itime in range(fantimes):
    if fapydts[itime]!=fbpydts[itime]:
        print("ERROR! Mismatching times in the files. Time number {}: File A = {} and File B = {}".format(
            itime, fapydts[itime].strftime('%Y-%m-%d %H:%M:%S'), fbpydts[itime].strftime('%Y-%m-%d %H:%M:%S')))
        sys.exit()

    pydt=fapydts[itime]
    print(" Processing {}/{} : {} ...".format(itime, fantimes, pydt.strftime('%Y-%m-%d %H:%M:%S')))
    z, lnsp = fbzs[itime,0,0], fblnsps[itime,0,0]
    sline="{}, {}, {}".format(pydt.strftime('%Y-%m-%d %H:%M:%S'), z, lnsp)

    #Calculate geopotentials and heights for the model levels
    geo, h = calculategeoh(z, lnsp, fats[itime,range(fanlevels),0,0],
        faqs[itime,range(fanlevels),0,0], falevels)

    with io.open(OUT_DIR_PATH + pydt.strftime('%Y%m%d-%H') + ".csv", 'w', newline='\r\n') as writer:
        sheader1 = "lat, long, geopotential, h\n"
        writer.write(unicode(sheader1))

        #Iterate over levels for this time
        for ilevel in range(fanlevels):
            sline="{}, {}, {}, {}".format(
                GRID_LAT, GRID_LONG, geo[ilevel], h[ilevel])
            sline+="\n"
            writer.write(unicode(sline))
```

```
print("Finished")
```

Copy the script and save it to your computer.

You have to adapt:

- line 58: specify your file containing t and q ('tq_ml.nc')
- line 59: specify your file containing z and lnsp ('tzlnsp.nc')
- line 60/61: specify lat/long of your point of interest. These must be multiples of the grid resolution of your input files.
- line 62: specify an output directory.

Run the script.

Outputs: CSV files (one per timestamp) with geopotential and height (relative to terrain) on each model level.

Related articles

- Transformation or regridding of ECMWF Reanalyses data
- ECMWF Model Documentation
- Model grid box and time step
- ERA-Interim: documentation
- ERA5: compute pressure and geopotential on model levels, geopotential height and geometric height