compute_geopotential_on_ml.py

```
#!/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()
```