```python
#!/usr/bin/env python
import datetime as dt
import numpy as np
import os.path

from python.profile_database import ProfileDatabase
from python.vertical_profile import VerticalProfile
from python.plot_profile import plot_profile
import python.datasets.stations_list as stations_list
import python.datasets.util as util

#####################################################
# TEST


# parameters:

# no of radiosonde:
wmoId = 17095
station = stations_list.stations[wmoId]
stations = [ station ]
minh = 16000
maxh = 26000

# pick compared datasets:
# 3/10/2018 ECMWF EUROPE z levels from 16 km
# TODO: comparing sondes won't work, since they have variable size
model_label = "ECMWF" # WRF; TODO: ECMWF doesn't work yet
sonde_label = "LORES" # LORES or HIRES

# TODO: only those params are currently available:
params = ["wvel_knt", "wdir_deg", "u_knt", "v_knt"]
# param = "wdir_deg"

# date ranges:
start_date = dt.datetime(2016, 4, 1,12,00)
end_date   = dt.datetime(2016, 4, 30,12,00)
str_H=end_date.strftime("%H")
# output figs dir:
outdir='/home/sigalit/Loon/v_Oct_2018/machon-winds-master/'+str(model_label)+str(wmoId)+'_'+sond

if not os.path.exists(outdir):
    os.makedirs(outdir)


# fetch data handles:
db = ProfileDatabase()
model_ds = db.get_dataset(model_label, minh, maxh, params)
sonde_ds = db.get_dataset(sonde_label, minh, maxh, params)


# prepare arrays for statistics:
heights = db.get_heights( minh, maxh)

rad_dir="/home/sigalit/sigdata/Loon_radiosondes/"


#####################################################
# caching all relevant data:

profiles = {}
print("Caching profiles...")
for (heights, model, sonde, curr_date) in db.iterator(model_ds, sonde_ds, heights, station, star
    print("Extracting %s..." % curr_date)
    profiles[curr_date] = (heights, model, sonde, curr_date)
```

```
#######################################################
# preparing arrays for statistics:


bias = {}
mae = {}
rmse = {}

model_mean = {}
sonde_mean = {}

model_var = {}
sonde_var = {}
model_var2 = {}
sonde_var2 = {}
count = {}  # at each z level number of events  when model and measurement exist


for param in params:

    bias[param] = np.zeros((len(heights)))
    mae[param] = np.zeros((len(heights)))
    rmse[param] = np.zeros((len(heights)))
    model_mean[param] = np.zeros((len(heights)))
    sonde_mean[param] = np.zeros((len(heights)))

    model_var[param] = np.zeros((len(heights)))
    sonde_var[param] = np.zeros((len(heights)))
    model_var2[param] = np.zeros((len(heights)))
    sonde_var2[param] = np.zeros((len(heights)))

    count[param] = np.zeros((len(heights)))



count_mod={}
count_sonde={}
model_delta={}
sonde_delta={}

for param in params:
    count_mod[param]=np.zeros((len(heights)))
    count_sonde[param]=np.zeros((len(heights)))
    model_delta[param]=np.zeros((len(heights)))
    sonde_delta[param]=np.zeros((len(heights)))
    model_delta[param][:]=np.nan
    sonde_delta[param][:]=np.nan

for (heights, model, sonde, curr_date) in profiles.values():
    for param in params:
        for ix in range(len(heights)):
            # data was found and this is not a missing value
            # for wind dir nan for missing data or when sp < 2 knt. wind dir values - are wrong
            # have to set correct values according to u v
             if (model.values[param] is not None and not np.isnan(model.values[param][ix])): cou
             if (sonde.values[param] is not None and not np.isnan(sonde.values[param][ix])): cou

#######################################################
#  params = ["wvel_knt", "wdir_deg", "u_knt", "v_knt", "pres_hpa"]
#   1 kt = 0.51444444444 mps
```

```
print("Calculating statistics...")
for (heights, model, sonde, curr_date) in profiles.values():
    for param in params:
        model_values = model.values[param]  # type: np array
        sonde_values = sonde.values[param]  # type: np array
        #print param
        #print model_values
        #print sonde_values
        if model_values is None or sonde_values is None:
            continue
        delta=np.zeros((len(heights)))
        delta[:] = np.nan
        if param == "wdir_deg":
            for ix in range(len(heights)):
                if model_values[ix] is not np.nan and sonde_values[ix] is not np.nan:
                    delta[ix] = util.to_degrees(sonde.values["u_knt"][ix], sonde.values["v_knt"]
                    - \
                    util.to_degrees(model.values["u_knt"][ix], model.values["v_knt"][ix])

                    delta[ix] = util.wrap_degrees(delta[ix])
        else:
            for ix in range(len(heights)):
                if model_values[ix] is not np.nan and sonde_values[ix] is not np.nan:
                    delta[ix] = sonde_values[ix] - model_values[ix]

        for ix in range(len(heights)):
            if (not np.isnan(delta[ix])): count[param][ix] += 1
            if (np.isnan(delta[ix])): delta[ix] = 0  # delta = 0 , do not increase number of eve
        #print count[param]

        bias[param] += delta



        model_mean[param] += model_values

        for ix in range(len(heights)):
            if not np.isnan(sonde_values[ix])   :
                sonde_mean[param][ix] += sonde_values[ix]

        mae[param] += abs(delta)
        rmse[param] += delta**2



for param in params:
    if param != "wdir_deg":
        for ix in range(len(heights)):
            if count_mod[param][ix] != 0 :
                model_mean[param][ix] /= count_mod[param][ix]
            if count_sonde[param][ix] != 0 :
                sonde_mean[param][ix] /= count_sonde[param][ix]

    #print sonde_mean[param]
    for ix in range(len(heights)):
        if count[param][ix] != 0 :
            bias[param][ix] /= count[param][ix]
            mae[param][ix] /= count[param][ix]
            rmse[param][ix] = (rmse[param][ix] / count[param][ix])**0.5


# completed mean bias ame rmse calculations
```

```
model_mean["wdir_deg"] = util.to_degrees(model_mean["u_knt"],model_mean["v_knt"])


for ix in range(len(heights)):
    if(not np.isnan(sonde_mean["u_knt"][ix])  ):
        sonde_mean["wdir_deg"][ix] = util.to_degrees(sonde_mean["u_knt"][ix],sonde_mean["v_knt"]
#print sonde_mean["wdir_deg"]


#########################################################
# second pass: calculate variance




for (heights, model, sonde, curr_date) in profiles.values():

    for param in params:

        model_values = model.values[param]  # type: np array
        sonde_values = sonde.values[param]  # type: np array
        if model_values is None or sonde_values is None:
            continue

        if param == "wdir_deg": # define dir values from u v  again  model_values, sonde_values
            for ix in range(len(heights)):
                if model_mean[param][ix] is not np.nan and model_values[ix] is not np.nan:
                    model_delta[param][ix] = util.to_degrees(model_mean["u_knt"][ix], model_mean
                    -   util.to_degrees(model.values["u_knt"][ix], model.values["v_knt"][ix])


            for ix in range(len(heights)):
                if sonde_mean[param][ix] is not np.nan and sonde_values[ix] is not np.nan:
                    sonde_delta[param][ix] = util.to_degrees(sonde_mean["u_knt"][ix], sonde_mean
                    -   util.to_degrees(sonde.values["u_knt"][ix], sonde.values["v_knt"][ix])
                #print sonde_delta[param][ix]

            for ix in range(len(heights)):
                if np.isnan(model_delta[param][ix]) :
                    # skip assigment,
                    # in case of a single day, we will get 0 error- which means - no data , so o
                    # match between model and rad or no data case
                    model_delta[param][ix] = 0
                else:
                    model_delta[param][ix] = np.abs(util.wrap_degrees(model_delta[param][ix]))

                if  np.isnan(sonde_delta[param][ix]) :
                    sonde_delta[param][ix] = 0
                else:
                    sonde_delta[param][ix] = np.abs(util.wrap_degrees(sonde_delta[param][ix]))


        else:

            for ix in range(len(heights)):
                if not np.isnan(model_values[ix]) and not np.isnan(model_mean[param][ix]):
                    model_delta[param][ix] = model_mean[param][ix] - model_values[ix]
                else:
                    model_delta[param][ix] = 0

            for ix in range(len(heights)):
                if not np.isnan(sonde_values[ix]) and not np.isnan(sonde_mean[param][ix]):
                    sonde_delta[param][ix] = sonde_mean[param][ix] - sonde_values[ix]
                else:
```

```
                    sonde_delta[param][ix] = 0



# second term for wind dir variance calculations:
        model_var2[param] += model_delta[param]
        sonde_var2[param] += sonde_delta[param]
        model_var[param] += model_delta[param]**2
        sonde_var[param] += sonde_delta[param]**2


# finalize computation:
# calculate variance:
for param in params:
     if param == "wdir_deg":
         for ix in range(len(heights)):
             if count_mod[param][ix] != 0. :
                 model_var[param][ix]= (model_var[param][ix]/ count_mod[param][ix]  - (model_var
             if count_sonde[param][ix] != 0 :
                 sonde_var[param][ix] = (sonde_var[param][ix] / count_sonde[param][ix]  - (sonde
     else:

         for ix in range(len(heights)):
             if count_mod[param][ix] != 0. :
                 model_var[param][ix] = (model_var[param][ix] / count_mod[param][ix]  )**0.5
             if count_sonde[param][ix] != 0 :
                 sonde_var[param][ix] = (sonde_var[param][ix] / count_sonde[param][ix]  )**0.5



# print number of events :
print 'number of days [wdir] = ',count["wdir_deg"],count_mod["wdir_deg"],count_sonde["wdir_deg"]

# print results, screen and file:
ofile = outdir+'output_statistics_wind'


of = open( ofile, 'w')
for idx in range(len(bias["wvel_knt"])):
    of.writelines("%6dm : bias:%3.3f mae:%3.3f rmse:%3.3f \n" % (heights[idx], bias["wvel_knt"][
    print("%6dm : bias:%3.3f mae:%3.3f rmse:%3.3f" % (heights[idx], bias["wvel_knt"][idx], mae["

##########################################

# dont display edges of interpolation results
    # suffer from low number of points
    # radiosonde dont have a lot of points at the higher levels 20-24 km
    # therefore nf=len(heights)-2
    # draw from index 2 , eg mae[draw_param][2: nf]
    #
for draw_param in ["wdir_deg","wvel_knt","u_knt","v_knt"]:
    title = "%s Errors for %s to %s , station %s" % (model_label, start_date, end_date , station
    nf=len(heights)-2
    plot_profile(
        VerticalProfile(heights[2:nf],{
          draw_param + " MAE": mae[draw_param][2: nf],
          draw_param + " RMSE": rmse[draw_param][2: nf]
        }, station),
        VerticalProfile(heights[2: nf], {
          draw_param + " bias": bias[draw_param][2: nf]
        }, station),
        outdir, title)
    # save figure
```

```
title = "%s Means for %s to %s, station %s" % (model_label,start_date, end_date, station.wmo
plot_profile(
    VerticalProfile(heights[2: nf], {
       draw_param + " model mean": model_mean[draw_param][2: nf],
       draw_param + " sonde mean": sonde_mean[draw_param][2: nf]
    }, station),
    None, outdir, title)

title = "%s Variance for %s to %s, station %s" % (model_label, start_date, end_date, station
plot_profile(
    VerticalProfile(heights[2: nf], {
       draw_param + " sonde variance": sonde_var[draw_param][2: nf],
       draw_param + " model variance": model_var[draw_param][2: nf]
    }, station),
    None, outdir, title)
```