pyPROS Documentation

Servei Meteorològic de Catalunya

Dec 01, 2019

CONTENTS:

Knowledge of surface precipitation type can be critical during snow events at low altitudes or in regions not used to this phenomena. For this purpose, previous studies developed several methodologies to discriminate precipitation types using meteorological surface observations. Some of them are implemented in this package.

CHAPTER

ONE

INSTALLATION

There are several ways to install this package

1.1 setup.py

```
python3 setup.py build
python3 setup.py install
```

1.2 pip

```
pip install -r requirements.txt
pip install pypros
```

1.3 Anaconda

```
conda install -c meteocat pypros
```

CHAPTER

TWO

RAIN OR SNOW

Knowledge of surface precipitation type can be critical during snow events at low altitudes or in regions not used to this phenomena. For this purpose, previous studies developed several methodologies to discriminate precipitation types using meteorological surface observations. Some of them are implemented in this package.

There are different approaches to address this issue:

- Single threshold
- Linear transition
- Koistinen and Saltikoff
- Dual threshold

2.1 Single threshold

A single temperature value is set as a threshold from which precipitation type is discriminated. If temperature is above the threshold, precipitation is classified as rain, otherwise as snow.

2.1.1 Air temperature (TA)

An air temperature (T_a) value is used to discriminate precipitation between rain and snow. If precipitation occurs above the air temperature value considered, rain is assumed. Otherwise, precipitation is classified as snow.

$$T_a <= T_{a_{threshold}} \longrightarrow Snow$$
$$T_a > T_{a_{threshold}} \longrightarrow Rain$$

The best air temperature single threshold may be different depending on the region. For more information on which is the most suitable threshold for your area, see https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5861046/.

2.1.2 Wet bulb temperature (TW)

A wet bulb temperature (T_w) value is used to discriminate precipitation between rain and snow. If precipitation occurs above the air temperature value considered, rain is assumed. Otherwise, precipitation is classified as snow.

$$T_w <= T_{w_{threshold}} \longrightarrow Snow$$
$$T_w > T_{w_{threshold}} \longrightarrow Rain$$

The best wet bulb temperature single threshold may be different depending on the region. Still, it is common to use a wet bulb temperature value of 1.5°C.

2.2 Linear transition

Two threshold values are set to discriminate precipitation type between rain (th_r) and snow (th_r). It can be either used with any meteorological field, but with thresholds properly defined. If a value of the meteorological field is above th_r, precipitation is classified as rain. On the other hand, if the value is below th_s, precipitation is classified as snow. A linear transition is assumed for values between th_s and th_r, then precipitation is classified as a mixed type.

If the meteorological field chosen to discriminate precipitation is air temperature:

$$\begin{aligned} T_a <= T_{snow} &\longrightarrow Snow \\ T_{snow} < T_a < T_{rain} &\longrightarrow Mixed \\ T_a >= T_{rain} &\longrightarrow Rain \end{aligned}$$

2.3 Koistinen and Saltikoff (KS)

The methodology proposed by Koistinen and Saltikoff (1998) provides an empirical formula to calculate the probability of precipitation type using temperature and relative humidity observations. Formally, the formula calculates the probability of rain and two thresholds are set to discriminate between snow, sleet and rain. In our case, the equation is flipped, so probability of snow is determined by (1) which may be expressed as

$$p(snow) = 1 - \frac{1}{1 + e^{22 - 2.7 \cdot T - 0.2 \cdot RH}}$$

where T corresponds to temperature in Celsius and RH to relative humidity in %. If p(snow) obtained values are below 0.33 precipitation is in form of rain, if they are between 0.33 and 0.66 in form of sleet and classified as snow if they are above 0.66.

2.4 Dual thresholds

Two threshold values are set to discriminate precipitation type between rain (th_r) and snow (th_r). It can be either used with any meteorological field, but with thresholds properly defined. If a value of the meteorological field is above th_r, precipitation is classified as rain. On the other hand, if the value is below th_s, precipitation is classified as snow. Finally, if the values are between th_s and th_r, then precipitation is classified as a mixed type.

If the meteorological field chosen to discriminate precipitation is wet bulb temperature:

$$\begin{aligned} T_w <= T_{snow} &\longrightarrow Snow \\ T_{snow} < T_w < T_{rain} &\longrightarrow Mixed \\ T_w >= T_{rain} &\longrightarrow Rain \end{aligned}$$

CHAPTER

THREE

API

3.1 The PyPROS module

Functions to calculate the precipitation type. For a point or numpy arrays

class pypros.pros.**PyPros**(*variables_file*, *method='ks'*, *threshold=None*, *data_format=None*)

Main project class. Discriminates precipitation type considering different methodologies using surface observations.

__init__(*variables_file*, *method='ks'*, *threshold=None*, *data_format=None*)

Parameters

- **variables_file** (*str, list*) – The file paths containing air temperature, dew point temperature and (digital elevation model) fields.
- **method** (*str*) – The precipitation type discrimination method to use. Defaults to ks.

  Available:

  - ks : Koistinen and Saltikoff method
  - **single_tw: A single wet bulb temperature** threshold
  - single_ta: A single air temperature threshold
  - dual_tw : A dual wet bulb temperature thresholds
  - dual_ta : A dual air temperature threshold
  - **linear_tr: Linear transition between rain** and snow
- **threshold** (*float, list*) – Threshold value(s) to use in the different methods available.

  Defaults to:

  - static_tw: 1.5
  - static_ta: 0.0
  - linear_tr: [0, 3]
- **data_format** (*dict, optional*) – Defaults to None. The order of the variables in the variables files. Defaults to: {'vars_files': ['tair', 'tdew', 'dem']}

Raises **ValueError** – Raised when the method is not valid

__weakref__

list of weak references to the object (if defined)

refl_mask (*refl*)

Calculates the precipitation type masked. The output classification is as follows:

rain

- 1dBZ : 1
- 5dBZ: 2
- 10dBZ : 3
- 15dBZ: 4
- 25dBZ : 5

sleet

- 1dBZ: 6
- 5dBZ : 7
- 10dBZ: 8
- 15dBZ : 9
- 25dBZ: 10

snow

- 1dBZ : 11
- 5dBZ: 12
- 10dBZ : 13
- 15dBZ: 14
- 25dbZ: 15

Parameters **refl** (*numpy.array*) – Array with reflectivity values

Raises **IndexError** – Raised if the types don't match in size or type

Returns The precipitation type classification value

Return type float, numpy array

save_file (*field*, *file_name*)

Saves the calculate field data into a file

Parameters **file_name** (*str*) – The output file path

3.2 Psychrometric calculations

Psychrometric calculations

pypros.psychrometrics.**get_tw_sadeghi** (*tair*, *tdew*, *z*)

Gets the wet bulb temperature from air temperature, dew point temperature and pressure. Formula taken from: https://journals.ametsoc.org/doi/pdf/10.1175/JTECH-D-12-00191.1

Results close to trhp2tw, but computationally efficient

Parameters

- **tair** (*float, numpy array*) – The air temperature in Celsius

- **tdew** (*float, numpy array*) – The dew point temperature in Celsius
- **z** (*float, numpy array*) – The altitude in metres

Returns The wet bulb temperature in Celsius

Return type float, numpy array

pypros.psychrometrics.**hr2td**(*temp*, *r_h*)

Returns the dew point from the relative humidity and the temperature Formula from: https://www.aprweather.com/pages/calc.htm

Both float values or numpy matrices can be passed as input and get as output

Parameters

- **temp** (*float, numpy array*) – The temperature in Celsius
- **r_h** (*float, numpy array*) – The relative humidity in %

Returns The dew point in Celsius

Return type float, numpy array

pypros.psychrometrics.**td2hr**(*temp*, *tempd*)

Returns the relative humidity from the temperature and the dew point Formula from: https://www.aprweather.com/pages/calc.htm

Both float values or numpy matrices can be passed as input and get as output

Parameters

- **temp** (*float, numpy array*) – The temperature in Celsius
- **tempd** (*float, numpy array*) – The dew point in Celsius

Returns The relative humidity in %

Return type float, numpy array

pypros.psychrometrics.**trhp2tw**(*temp*, *rh*, *z*)

Gets the wet bulb temperature from the temperature, relative humidity and pressure. Formula taken from: https://www.weather.gov/epz/wxcalc_wetbulb (Brice and Hall, 2003)

Parameters

- **temp** (*float, numpy array*) – The temperature in Celsius
- **rh** (*float, numpy array*) – The relative humidity in [0,1]
- **z** (*float, numpy array*) – The altitude in metres

Returns The wet bulb temperature in Celsius

Return type float, numpy array

pypros.psychrometrics.**ttd2tw**(*temp*, *tempd*)

Gets the wet bulb temperature from the temperature and the dew point Formula taken from: https://journals.ametsoc.org/doi/full/10.1175/JAMC-D-11-0143.1

TODO: Take altitude in account (should change algorithm)

Parameters

- **temp** (*float, numpy array*) – The temperature in Celsius
- **tempd** (*float, numpy array*) – The dew point in Celsius

Returns The wet bulb temperature in Celsius

Return type float, numpy array

3.3 Rain or snow methodologies

Implements several rain or snow methodologies.

pypros.ros_methods.**calculate_dual_threshold**(*field*, *th_s*, *th_r*)

Calculates the precipitation type based on two threshold values, one for rain and one for snow. If value >= th_r –> rain –> 0 If value <= th_s –> snow –> 1 If th_s < value < th_r –> mixed –> 0.5

Parameters

- **field** (*float, numpy array*) – Meteorological variable field
- **th_s** (*float*) – Snow threshold. Values below this threshold classified as snow.
- **th_r** (*float*) – Rain threshold. Values above this threshold classified as rain.

Raises **ValueError** – Raised if th_r is smaller than th_s.

Returns Precipitation type field

Return type float, numpy array

pypros.ros_methods.**calculate_koistinen_saltikoff**(*temp*, *tempd*)

Returns the Koistinen-Saltikoff value.

Koistinen J., Saltikoff E. (1998): Experience of customer products of accumulated snow, sleet and rain, COST 75 Final Seminar on Advanced Weather Radar Systems, Locarno, Switzerland. EUR 18567 EN, 397-406.

The formula values are
- prob < 0.3 –> rain
- 0.3 < prob < 0.7 –> sleet
- prob > 0.7 –> snow

Both float values or numpy matrices can be passed as input and get as output

Parameters

- **temp** (*float, numpy array*) – The temperature in Celsius
- **tempd** (*float, numpy array*) – The dew point in Celsius

Returns The Koistinen J., Saltikoff E. formula value

Return type float, numpy array

pypros.ros_methods.**calculate_linear_transition**(*field*, *th_s*, *th_r*)

Calculates the probability of precipitation type based on two threshold values, one for rain and one for snow. Assumes a linear transition between them. If value >= th_r –> rain –> 0 If value <= th_s –> snow –> 1 If th_s < value < th_r –> mixed –> (0, 1)

Parameters

- **field** (*float, numpy array*) – Meteorological variable field
- **th_s** (*float*) – Snow threshold. Values below this threshold classified as snow.
- **th_r** (*float*) – Rain threshold. Values above this threshold classified as rain.

Raises **ValueError** – Raised if th_r is smaller than th_s.

Returns Probability of precipitation type field

Return type float, numpy array

pypros.ros_methods.**calculate_single_threshold**(*field*, *th*)

Calculates the precipitation type based on a threshold value. If value > threshold –> rain –> 0 If value <= threshold –> snow –> 1

Parameters

- **field** (*float, numpy array*) – Meteorological variable field
- **th** (*float*) – Threshold from which precipitation type is discriminated

Returns Precipitation type field

Return type float, numpy array

CHAPTER

FOUR

EXAMPLES

This section contains explanations and examples of the PyPros class applications.

4.1 PyPros class

PyPros is the main class of this library as it implements the different methodologies available to discriminate the surface precipitation type using surface observations.

In this notebook we'll cover the parameters of PyPros class and their format depending on the rain or snow methodology.

First of all, we'll import PyPros class.

```
from pypros.pros import PyPros
```

`PyPros` class receives four parameters:

- variables_files: A list of the files paths containing the fields of required variables
- method: The surface precipitation type method to use
- threshold: The value of the threshold(s) to be used by the chosen method
- data_format: A dictionary containing the order of the fields in variables_files

4.1.1 Variables_files

There are two mandatory fields to include: air temperature and dew point temperature. Both fields allow to use all the implemented methodologies of surface precipitation type discrimination.

Digital Elevation Model (DEM) is an optional field which allows to calculate accurately the wet bulb temperature (if this method is selected) by using altitude values. Otherwise, wet bulb temperature is derived from air and dew point temperature fields only.

First, we'll define the paths to each field and we'll set `variables_file` with all of them.

```
tair_file = '../sample-data/INT_TAIR_20170325_0030.tif'
tdew_file = '../sample-data/INT_TDEW_20170325_0030.tif'
dem_file = '../sample-data/DEM_CAT.tif'

variables_files = [tair_file, tdew_file, dem_file]
```

4.1.2 Method and threshold

The method is an optional parameter defaults to Koistinen and Saltikoff method, which must be passed as 'ks'. The following table illustrates the different methodologies available, how they must be introduced in `PyPros` class and the kind of threshold required. If no threshold is set, it assumes the default one.

Method	Name	Threshold	Default
Koistinen and Saltikoff	`'ks'`	`None`	`None`
Air temperature single threshold	`'single_ta'`	`float`	`0.0`
Wet bulb temperature single threshold	`'single_tw'`	`float`	`1.5`
Air temperature linear transition	`'linear_tr'`	`[th_l, th_u]`	`[0, 3]`

Now, as an example, we'll define wet bulb temperature single threshold as the method to use and set threshold to 1.3°C.

```
method = 'single_tw'
threshold = 1.3
```

Data format

This parameter is a dictionary containing a key, `vars_files` providing the order of the fields in `variables_files`. The name of the variables are the following ones:

Field	Name
Air temperature	`'tair'`
Dew point temperature	`'tdew'`
Digital Elevation Model	`'dem'`

Then, we'll set `data_format` parameter following the `variables_files` order:

```
data_format = {'vars_files': ['tair', 'tdew', 'dem']}
```

Now we're ready to call PyPros class and obtain a surface precipitation type field.

```
single_tw = PyPros(variables_files, method, threshold, data_format)
```

Once we've called the class, now we can obtain the surface precipitation type field, apply the reflectivity mask available and save both in a raster file.

To obtain the result, we must get the `result` attribute of the class.

```
single_tw_field = single_tw.result
```

And if we want to apply the reflectivity mask, we have to call `refl_mask` function from the PyPros class, which requires the reflectivity field as a parameter. So before calling `refl_mask`, we have to prepare the reflectivity field.

First of all, as it's a .tif file, we'll import `gdal` library.

```
from osgeo import gdal
refl_file = '../sample-data/CAPPI_XRAD_20170325_0030.tif'
refl_array = gdal.Open(refl_file).ReadAsArray()
```

In this case we used gdal because we have the reflectivity field stored in a .tif file, but for the `refl_mask` only an array is needed. So any format can be used, as long as it is transformed into a numpy array.

```
single_tw_masked = single_tw.refl_mask(refl_array)
```

Now, we've obtained two fields that we can save in raster files using `save_result` function from PyPros class. This function receives two parameters: the field matrix we want to save and the file path destination.

```
single_tw.save_file(single_tw_field, '../sample-data/output/single_tw.tif')
single_tw.save_file(single_tw_masked, '../sample-data/output/single_tw_masked.tif')
```

We can have a look at `single_tw` result by plotting it with imshow:

```
import matplotlib.pyplot as plt

plt.imshow(single_tw.result)
plt.colorbar()
plt.show()
```

We have finished the introduction to PyPros class! Change the threshold values

and methods and see how the snow level varies!

4.2 pypros_run script

If you want to run PyPros form terminal directly, you can use `pypros_run` script. Now we'll how it must be called.

`pypros_run` receives up to six arguments, since two of them are optional. The arguments and their order are the following ones:

Order	Argument	Description	Mandatory
1	tair	Air temperature field file path	
2	tdew	Dew point temperature field file path	
3	config_file	Configuration file path	
4	out_file	Digital Elevation Model file path	
5	dem	Digital Elevation Model file path	

The configuration file is a .json including the following parameters:

```
{
 "method": "single_tw",
 "threshold": 1.0,
 "data_format": {"vars_files": ["tair", "tdew", "dem"]},
 "refl_masked": "True"
}
```

For more information about the pypros_run script configuration parameters, see PyPros Class.

In order to execute the script you must have pyPROS package installed, see Documentation.

A configuration file and sample fields for air temperature, dew point temperature, digital elevation model and radar reflectivity are available in `../sample-data/` directory. We'll introduce two examples of how `pypros_run` script is run.

4.2.1 Air temperature single threshold

The configuration file must look like the following one. We'll set the threshold to 1.0°C.

```
{
 "method": "single_ta",
 "threshold": 1.0,
 "data_format": {"vars_files": ["tair", "tdew"]},
 "refl_masked": "False"
}
```

Since we set `refl_masked` to `False` we do not have to import any radar reflectivity field. We would execute the script this way:

```
> pypros_run [path to air temperature field] [path to dew point temperature field]␣
↪[path to configuration file] [output path]
```

4.2.2 Wet bulb temperature threshold

The configuration file should include the following parameters. We'll set the threshold to 1.5°C.

```
{
 "method": "single_tw",
 "threshold": 1.5,
 "data_format": {"vars_files": ["tair", "tdew", "dem"]},
 "refl_masked": "True"
}
```

Since we set `refl_masked` to `True` we have to include the radar reflectivity field in the configuration file and as an script argument. In addition, we have also included `dem` in order to take into account altitude when calculating wet bulb temperature. We would execute the script this way:

```
> pypros_run [path to air temperature field] [path to dew point temperature field]␣
↪[path to configuration file] [output path] --dem [path to dem] --refl [path to␣
↪radar reflectivity file]
```

4.3 Single threshold

A single meteorological variable value is set as a threshold from which precipitation type is discriminated. If the meteorological variable value is above the threshold, precipitation is classified as rain, otherwise as snow.

If air temperature (T_a) is chosen as meteorological variable:

$T_a <= T_{a_{threshold}} \longrightarrow Snow$
$T_a > T_{a_{threshold}} \longrightarrow Rain$

In the following example we'll show how PyPROS classifies precipitation considering the single threshold methodology.

First of all, we'll import the required libraries.

```
from pypros.pros import PyPros
```

As an example, we'll get the precipitation type classification from different methodologies for Catalonia on 2017-03-25 00.30 UTC. For this purpose we'll use an air temperature, dew point temperature, digital elevation model (DEM) and reflectivity fields.

Those fields can be found in notebooks/data directory and we'll keep the path for all of them:

```
tair_file = '../sample-data/INT_TAIR_20170325_0030.tif'
tdew_file = '../sample-data/INT_TDEW_20170325_0030.tif'
dem_file = '../sample-data/DEM_CAT.tif'
```

Now, we'll define those parameters that PyPros class uses and are the same whether the methodology changes or not. These parameters are: `variables_files` and `data_format`. For more information on this class, see PyPros Class notebook.

```
variables_files = [tair_file,
                   tdew_file,
                   dem_file]
data_format = {'vars_files':['tair', 'tdew', 'dem']}
```

4.3.1 Air temperature threshold

Since we want to apply a single air temperature threshold, first we'll define `method` PyPros parameter as `'single_ta'` and then we'll set the `threshold` parameter to 1.0°C.

```
method = 'single_ta'
threshold = 1.0
```

Now, we're ready to call PyPros class!

```
single_ta = PyPros(variables_files, method, threshold, data_format)
```

We can get a quicklook of the obtained field using `plot_pros` function:

```
import matplotlib.pyplot as plt
plt.imshow(single_ta.result)
plt.show()
```

In addition, we can save the precipitation type field in a raster file using `save_file` function:

```
single_ta.save_file(single_ta.result, '../sample-data/output/single_ta.tif')
```

If we have a reflectivity field, we can also apply it as a mask by using `refl_mask` function and save it as a raster file. However, we'll have to read first the reflectivity field. For this purpose we need to import gdal.

```
from osgeo import gdal

refl_file = '../sample-data/CAPPI_XRAD_20170325_0030.tif'
refl_array = gdal.Open(refl_file).ReadAsArray()
```

Once we've read the `refl_field` we can call the `refl_mask` function.

```
single_ta_masked = single_ta.refl_mask()

single_ta.save_file(single_ta_masked, '../sample-data/output/single_ta_masked.tif')
```

4.3.2 Wet bulb temperature threshold

We want to apply a single wet bulb temperature threshold, so first we'll define `method` PyPros parameter as `'single_tw'` and then we'll set the `threshold` parameter to 1.5°C.

```
method = 'single_tw'
threshold = 1.5
```

Now, we're ready to call PyPros class!

```
single_tw = PyPros(variables_files, method, threshold, data_format)
```

We can get a quicklook of the obtained field using `plot_pros` function:

```
import matplotlib.pyplot as plt
plt.imshow(single_tw.result)
plt.show()
```

In addition, we can save the precipitation type field in a raster file using `save_file` function:

```
single_tw.save_file(single_tw.result, '../sample-data/output/single_tw.tif')
```

If we have a reflectivity field, we can also apply it as a mask by using `refl_mask` function and save it as a raster file. However, we'll have to read first the reflectivity field. For this purpose we need to import gdal.

```
from osgeo import gdal

refl_file = '../sample-data/CAPPI_XRAD_20170325_0030.tif'
refl_array = gdal.Open(refl_file).ReadAsArray()
```

Once we've read the `refl_file` we can call the `refl_mask` function.

```
single_tw_masked = single_tw.refl_mask(refl_array)

single_tw.save_file(single_tw_masked, '../sample-data/output/single_tw_masked.tif')
```

4.4 Linear transition

Two threshold values are set to discriminate precipitation type between rain (th_{rain}) and snow (th_{snow}). It can be either used with any meteorological field, but with thresholds properly defined. If a value of the meteorological field is above th_{rain}, precipitation is classified as rain. On the other hand, if the values is below th_{snow}, precipitation is classified as snow. A linear transition is assumed for values between th_{snow} and th_{rain}, then precipitation is classified as a mixed type.

If the meteorological field chosen to discriminate air is air temperature:

$T_a <= T_{snow} \longrightarrow Snow$
$T_{snow} < T_a < T_{rain} \longrightarrow Mixed$
$T_a >= T_{rain} \longrightarrow Rain$

In the following example we'll show how PyPROS classifies precipitation considering the linear transition methodology.

First of all, we'll import the required libraries.

```
from pypros.pros import PyPros
```

As an example, we'll get the precipitation type classification from different methodologies for Catalonia on 2017-03-25 00.30 UTC. For this purpose we'll use an air temperature, dew point temperature, digital elevation model (DEM) and reflectivity fields.

Those fields can be found in notebooks/data directory and we'll keep the path for all of them:

```
tair_file = '../sample-data/INT_TAIR_20170325_0030.tif'
tdew_file = '../sample-data/INT_TDEW_20170325_0030.tif'
dem_file = '../sample-data/DEM_CAT.tif'
```

Now, we'll define those parameters that PyPros class uses and are the same whether the methodology changes or not. These parameters are: `variables_files` and `data_format`. For more information on this class, see PyPros Class notebook.

```
variables_files = [tair_file,
                   tdew_file,
                   dem_file]
data_format = {'vars_files':['tair', 'tdew', 'dem']}
```

4.4.1 Air temperature transition

Since we want to apply an air temperature linear transition, first we'll define `method` PyPros parameter as `'linear_tr'` and then we'll set the `threshold` parameter to [0, 3] (°C).

```
method = 'linear_tr'
threshold = [0, 3]
```

Now, we're ready to call PyPros class!

```
linear_tr = PyPros(variables_files, method, threshold, data_format)
```

We can get a quicklook of the obtained field using `plot_pros` function:

```
import matplotlib.pyplot as plt
plt.imshow(linear_tr.result)
plt.show()
```

In addition, we can save the precipitation type field in a raster file using `save_file` function:

```
linear_tr.save_file(linear_tr.result, '../sample-data/output/linear_tr.tif')
```

If we have a reflectivity field, we can also apply it as a mask by using `refl_mask` function and save it as a raster file. However, we'll have to read first the reflectivity field. For this purpose we need to import gdal.

```
from osgeo import gdal

refl_file = '../sample-data/CAPPI_XRAD_20170325_0030.tif'
refl_array = gdal.Open(refl_file).ReadAsArray()
```

Once we've read the `refl_field` we can call the `refl_mask` function.

```
linear_tr_masked = linear_tr.refl_mask(refl_array)

linear_tr.save_file(linear_tr_masked, '../sample-data/output/linear_tr_masked.tif')
```

4.5 Koistinen-Saltikoff

The methodology proposed by Koistinen and Saltikoff (1998) provides an empirical formula to calculate the probability of precipitation type using temperature and relative humidity observations. Formally, the formula calculates the probability of rain and two thresholds are set to discriminate between snow, sleet and rain. In our case, the equation is flipped, so probability of snow is determined by (1) which may be expressed as

$$p(snow) = 1 - \frac{1}{1 + e^{22 - 2.7 \cdot T - 0.2 \cdot RH}}$$

where T corresponds to temperature in Celsius and RH to relative humidity in %. If p(snow) obtained values are below 0.33 precipitation is in form of rain, if they are between 0.33 and 0.66 in form of sleet and classified as snow if they are above 0.66.

In the following example we'll show how PyPROS classifies precipitation considering the Koistinen-Saltikoff methodology.

First of all, we'll import the required libraries.

```
from pypros.pros import PyPros
```

As an example, we'll get the precipitation type classification from different methodologies for Catalonia on 2017-03-25 00.30 UTC. For this purpose we'll use an air temperature, dew point temperature, digital elevation model (DEM) and reflectivity fields.

Those fields can be found in notebooks/data directory and we'll keep the path for all of them:

```
tair_file = '../sample-data/INT_TAIR_20170325_0030.tif'
tdew_file = '../sample-data/INT_TDEW_20170325_0030.tif'
dem_file = '../sample-data/DEM_CAT.tif'
```

Now, we'll define those parameters that PyPros class uses and are the same whether the methodology changes or not. These parameters are: `variables_files` and `data_format`. For more information on this class, see PyPros Class notebook.

```
variables_files = [tair_file,
                   tdew_file,
                   dem_file]
data_format = {'vars_files':['tair', 'tdew', 'dem']}
```

Since we want to apply the Koistinen-Saltikoff methodology, first we'll define `method` PyPros parameter as `'ks'` and then we'll set the `threshold` parameter to `None`.

```
method = 'ks'
threshold = None
```

Now, we're ready to call PyPros class!

```
ks = PyPros(variables_files, method, threshold, data_format)
```

We can get a quicklook of the obtained field using `plot_pros` function:

```
import matplotlib.pyplot as plt
plt.imshow(ks.result)
plt.show()
```

In addition, we can save the precipitation type field in a raster file using `save_file` function:

```
ks.save_file(ks.result, '../sample-data/output/ks.tif')
```

If we have a reflectivity field, we can also apply it as a mask by using `refl_mask` function and save it as a raster file. However, we'll have to read first the reflectivity field. For this purpose we need to import gdal.

```
from osgeo import gdal

refl_file = '../sample-data/CAPPI_XRAD_20170325_0030.tif'
refl_array = gdal.Open(refl_file).ReadAsArray()
```

Once we've read the `refl_field` we can call the `refl_mask` function.

```
ks_masked = ks.refl_mask(refl_array)

ks.save_file(ks_masked, '../sample-data/output/ks_masked.tif')
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

C

G

H

P

R

S

T