Bulgarian Journal of Meteorology and Hydrology

Национален институт по метеорология и хидрология

National Institute of Meteorology and Hydrology

Bul. J. Meteo & Hydro 24/2 (2020) 84-98

On the use of Python programming language in NWP – atmospheric dispersion model coupling

Anton Petrov*

[1]*National Institute of Meteorology and Hydrology, Tsarigradsko shose 66, 1784 Sofia, Bulgaria*

Abstract. This article focuses on the benefits that Python programming language provides in variety of tasks related to model coupling, ranging from functions intended for simple calculations, to automation of data downloads from FTP server and file modifications. The use of Python is demonstrated on a system composed of the NWP model WRF and the Gaussian dispersion model AERMOD. Although, it is impossible to present the scripts in their entirety in this paper, an attempt to do so is made, for the sake of demonstration of Python's capabilities. Being easy to learn, flexible, reliable and fast, Python becomes a language of choice for thousands of developers, engineers, data analysts and scientists around the world.

Keywords: model coupling, python programming language, air pollution forecasting, WRF, AERMOD .

1. INTRODUCTION

Python is a widely used general-purpose, high-level programming language. It was developed mainly for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code. Python was named by its creator after the sketch comedy television show *Monty Python's Flying Circus*, which first aired on the BBC in 1969. The language comes with a large standard library that covers areas such as string processing (regular expressions, Unicode, calculating differences between files), Internet protocols (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming), software engineering (unit testing, logging, profiling, parsing Python code), and operating system interfaces (system calls, filesystems, TCP/IP sockets).

* anton.petrov@meteo.bg

At the very beginning, the programming language Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum (van Rossum, 2009) at the National Research Institute for Mathematics and Computer Science (Centrum Wiskunde & Informatica, CWI) in the Netherlands. Python is a successor to ABC programming language, capable of exception handling and interfacing with the Amoeba operating system (Tanenbaum, 1991). Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, "Benevolent Dictator for Life". However, van Rossum stepped down as leader on July 12, 2018.

Here, follows Python's version releases and developments evolution, according to JournalDev (2020):

- The first version ever, was ***Python 0.9.0***. Being a successor of the ABC language, Python 0.9.0 also came up with the concept of *classes, lists,* and *strings*. More importantly, it included *lambda, map, filter* and *reduce*, which aligned it heavily to functional programming.
- ***Python 1.2*** was the last version developed by the CWI team. Consequently, Van Rossum moved to the Corporation for National Research Initiatives (CNRI) in Reston, Virginia and worked on the project releasing a number of further improvements until ***Python 1.6***.
- In 2000, Guido van Rossum with other developers formed the BeOpen PythonLabs. Noteworthy, the BeOpen team released only one version of the language, the ***Python 2.0***. After this, the Python Software Foundation (PSF), a non-profit organization, came into the scene and took the responsibility of Python Licenses, Fundraising, development, and management of the community as well as the PyCon conferences.
- ***Python 3.0*** was released in the month of December 2008. It included various changes over ***Python 2.0***. The modification of the print statement was most noteworthy. Now it looks like "*print()*". With the release of further versions, Python is always getting bigger and better.

As to date, ***Python 3.8.3*** was released, and this is the version used in the scripts demonstrated further.

2. WIDELY USED PYTHON PACKAGES WITH SCIENTIFICALLY ORIENTED APPLICATIONS

When it comes to processing data files of considerable size, there are many useful Python libraries for data science which are designed to be robust and fast. Among the most widely used libraries are:

- **NumPy**. NumPy (the name comes from the abbreviation of the words "Numeric Python") is the fundamental package for scientific computing with Python, adding support for large, multidimensional arrays and matrices, along with a large library

of high-level mathematical functions to operate on these arrays. NumPy brings the computational power of languages like C and Fortran to Python, which is a language much easier to learn. Being a fundamental package, NumPy is part of many other Python scientific oriented packages.

- **Matplotlib**. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python (Hunter, 2007). Some of the features of Matplotlib are: development of publication quality plots with just a few lines of code; use of interactive figures that can be zoomed, panned, and updated; full control of line styles, font properties, and axes properties; export and embed to a number of file formats and interactive environments, etc.
- **Pandas**. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool. It provides labeled data structures similar to R data.frame objects, statistical functions, and much more.
- **PyNIO, PyNGL and netCDF4**. Both PyNIO and PyNGL are modules developed by the community of National Center for Atmospheric Research, USA (NCAR).
    - **PyNGL** is intended for generating high-quality, 2D visualizations of scientific data. It is built on the top of the same resource model used for the NCAR command language NCL (NCAR, 2019).
    - **PyNIO** is used for reading and writing NetCDF, GRIB, HDF and HDF-EOS files.
    - **NetCDF** module is powerful tool for reading and writing NetCDF files.

Some of the features provided by the packages mentioned above are used for the purpose of the exercise described in this article. There are lots of other scientific python packages: ***Astropy*** (collection of packages designed for use in astronomy), ***Biopython*** (tools for computational biology and bioinformatics), ***Bokeh*** (interactive visualization library that targets modern web browsers for presentation), ***Cubes*** (light-weight framework and set of tools for the development of reporting and analytical applications, Online Analytical Processing (OLAP), multidimensional analysis, and browsing of aggregated data), ***DMelt*** (numeric computation, statistics, analysis of large data volumes (Big Data), and scientific visualization), and many more.

A relatively new Python promising collection of tools – ***MetPy*** – is emerging as an attractive package intended for reading, visualizing, and performing calculations with weather data (https://unidata.github.io/MetPy/latest/index.html). As with many other tools, MetPy is binded with the installation of other Python packages, such as *NumPy, Matplotlib, Scipy, Pandas* and *Xarray*.

3. WORKFLOW OF THE NUMERICAL WEATHER PREDICTION – ATMOSPHERIC DISPERSION MODEL SYSTEM

The system proposed here is comprised of the numerical weather prediction (NWP) model WRF v.4.2 (Skamarock et al., 2019), and the atmospheric dispersion model

(ADM) AERMOD v.19191 (Cimorelli et al., 2005). The main purpose of the system is generating short term (72 hour) air pollution forecasts for Sofia, Bulgaria.

In order for the system to be initialized, some preliminary model settings have to be made. For the prognostic model, these refer mainly to the setup of domain configuration and the physics modules used. For the dispersion model, these settings are mostly related to terrain setup, buildings (if applicable), domain configuration and air pollution sources. All the tasks mentioned above, are "one time problems" in sense that after their initial setup, they do not involve any further modifications, unless indispensable. Figure 1 shows the domain configuration generated with the help of the WRF pre-processing system (WPS).

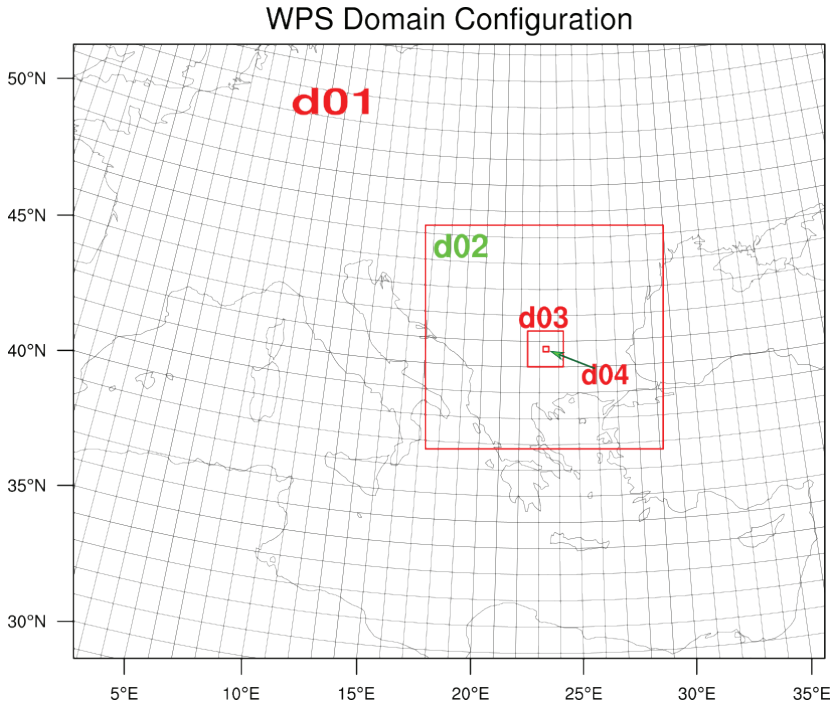

Fig. 1. Configuration of WRF modelling domains and nested subdomains: d01 – Europe, d02 – Balkan Peninsula, d03 – Sofia – region, and d04 – Sofia – city.

The master domain (d01) covers the territories mainly of Central, Southern and South-Eastern Europe, the shores of North Africa, Asia Minor Peninsula, and has a grid resolution of 20 x 20 km. The Balkan Peninsula (d02), Sofia – region (d03) and Sofia – city (d04) subdomains have grid resolutions of 5 x 5 km, 1 x 1 km, and 0.25 x 0.25 km respectively.

On Figure 2, a simple workflow schematics of the WRF-AERMOD system is shown. Almost all processes are managed by Python scripts, except for the nioenv.sh bash

script, which is necessary for loading of the PyNIO environment which is responsible for reading of the information in the downloaded from GFS GRIB2 data files.

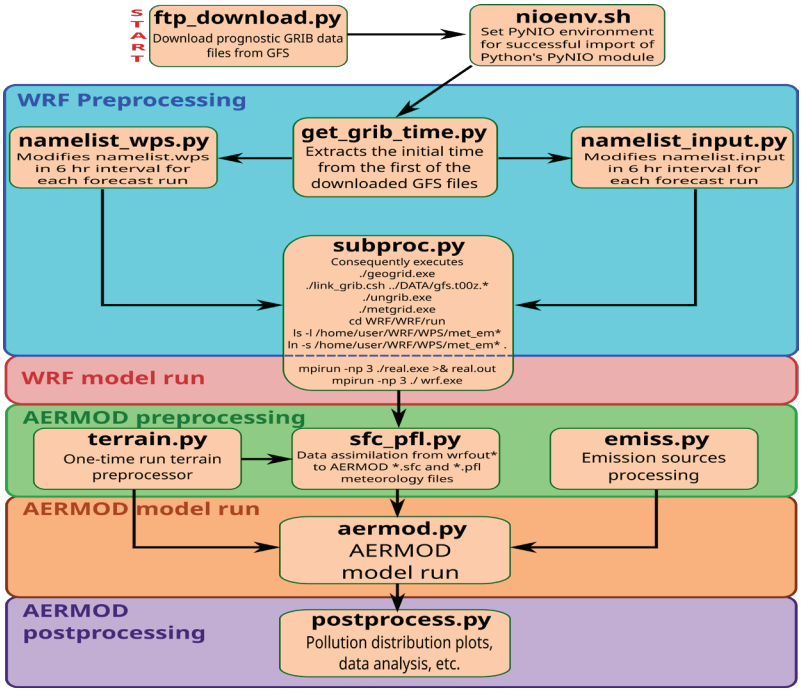

Fig. 2. WRF-AERMOD air pollution forecasting system workflow schematics

The first step in the workflow is downloading of the necessary prognostic data files from the Global Forecasting System (GFS). This is achieved via following Python code (the ftp_download.py script on Figure 2):

```
import os
from datetime import datetime, date, time, timezone
import time
import shutil
import urllib.request as request
import requests
from contextlib import closing

dt = datetime.today()
tt = dt.timetuple()

# Composing the url name with a varying part for the downloadable files
ftpdir = 'ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/'
todaydir = 'gfs.' + str(tt[0]) + str(tt[1]).zfill(2) + str(tt[2]).zfill(2)
+ '/00/'
files = ['gfs.t00z.pgrb2.0p50.f' + str(k).zfill(3) for k in range(6,79,3)]
# Valid for this device! Needs to be changed on another computer
destination_dir = '/home/user/wrf/Build_WRF/DATA/'
```

```
19
20 # Empty the DATA directory from outdated files
21 filelist = os.listdir(destination_dir)
22 print(filelist)
23 if filelist != []:
24     for f in filelist:
25         os.remove(destination_dir + f)
26
27 # Get the beginning time of the download
28 start = time.time()
29
30 for fi in files:
31     url = ftpdir + todaydir + fi
32     destination_path = destination_dir + fi
33
34     # Download the data from the ftp...
35     with closing(request.urlopen(url)) as r:
36         # ...and recording it to the destination folder
37         # by taking the name of the 'fi' variable
38         with open(destination_path, 'wb') as f:
39             sub_start = time.time()
40             print(f'Downloading {fi} from {ftpdir}{todaydir}...')
41             shutil.copyfileobj(r, f)
42             print(f'Downloaded for {round(time.time() - sub_start, 1)} s')
43
44 print(f'Total downloading time: {round(time.time() - start, 1)} s')
```

As it can be seen, with just about 40 lines of code, all the necessary information can be automatically downloaded from the ftp server, any obsolete data – deleted, and the processes can be monitored during execution by showing the time necessary for downloading.

4. VARIABLES AND PARAMETERS NECESSARY FOR THE METEOROLOGICAL INPUT OF THE ADM

In order for the inter-model coupling to be attained in the best possible manner, some conditions have to be satisfied. The most important condition is the availability of sufficient exhaustiveness of the meteorological output data from the NWP, that has to meet the demands of the atmospheric dispersion model. If that condition is not fully satisfied, there has to be enough meteorological information to at least run the ADM, or there has to be an option, such that the variables needed could be derived from the already available ones. In this regard, AERMOD's meteorological data demands can be as minimal as possible, but that would be at the cost of poor performance and more inaccurate results.

AERMOD assimilates the meteorological data through two specific input files. The first must contain surface observational data and calculated (by the preprocessor AERMET, or by the NWP model, or by means of a script as in this particular exercise) variables, which describe the meteorological conditions that influence the nature of air

pollutants dispersion – friction velocity, convective velocity scale , Monin-Obukhov length *L*. The second file contains hourly data with wind and temperature vertical profiles.

Some of the variables presented in the WRF prognostic output files (from here on, referenced as wrfout* files) were taken as they are, but some needed preliminary processing. A list of these variables follow with some comments.

- **Wind speed, U_{spd} (m s^{-1}).**

In the wrfout* files there are x, y, z components of wind available (respectively u, v, w). The wind speed (U_{spd}) is calculated as:

$$U_{\text{spd}} = \sqrt{u^2 + v^2 + w^2}\ [\text{ms}^{-1}] \tag{1}$$

- **Wind direction (degrees).**

AERMOD accepts meteorological wind direction as an input, i.e. the *direction from which the wind is blowing*. For the calculation, the zonal velocity (*u)* and the meridional velocity (*v)* wind components are used. Since 360 degree continuous scale is in use, some additional conditions have to be satisfied in order to calculate the wind direction. The following example function written in Python shows a solution to this problem:

```
1 import numpy as np
2
3 def wind_dir (u, v):
4  """Calculates the METEOROLOGICAL wind direction, given the u and v
5  components of wind. Not to be confused with the WIND VECTOR AZIMUTH!"""
6  d = 90 - np.arctan2(-v, -u) * 180/np.pi
7  if d < 0:
8     return d + 360
9  return d
```

As seen on line 6 in the script above, the first step performed is coordinate system rotation in 90º counterclockwise direction, so that the 0º (360º) direction can coincide with the direction of the *y*-axis (northward). Since in the interval [270º, 360º] the expression on line 6 returns values between -90º and 0º, an additional condition is applied, so that if the variable $d < 0$º, a value of $d + 360$º should be returned.

- **Elevation of a WRF level over earth's surface, *z* (m).**

The heights of the different layers over the surface (in this case the layers are 32) with the calculated from the NWP model variables, are expressed by WRF either by the so called *eta-levels* (in the range between 0 and 1), or in units of geopotential (m^2 s^{-2}). However, the heights attached to each level in the vertical profiles of wind and temperature assimilated by AERMOD should be presented in meters. The conversion of geopotential to elevation above the ground is done by using the following relation:

$$z = \frac{\left(\Phi_B^{(k)} + \Phi_B^{(k+1)} + \Phi_P^{(k)} + \Phi_P^{(k+1)}\right)}{2g} - h\,[m] \quad (2)$$

Here, z is the height of interest (m), $\Phi_B^{(k)}$ and $\Phi_B^{(k+1)}$ are the geopotentials and their perturbations $\Phi_P^{(k)}$ and $\Phi_P^{(k+1)}$ ($m^2\ s^{-2}$), on levels k and $k+1$ accordingly, g = 9.81 m s^{-2}, and h is terrain elevation (m). Expressed as a Python code, the expression above looks like this:

```
from netCDF4 import Dataset
import numpy as np

ncfile = 'wrfout_d02_2020-11-06_01:00:00'
f = Dataset(ncfile, 'r')

z = (((f.variables['PHB'][0, :-1, 83, 101] +
    f.variables['PHB'][0, 1:, 83, 101])/2 +
    (f.variables['PH'][0, :-1, 83, 101] +
     f.variables['PH'][0, 1:, 83, 101])/2)/9.81 -
     f.variables['HGT'][0, 83, 101])
z = np.array([int(z[i]) for i in range(len(z))])
```

- **Atmospheric pressure, p (hPa, mb).**

As similar as the geopotential, the pressure is presented by WRF output in two parts – basic and perturbational. For each of the 32 calculated levels the following formula is valid:

$$p^{(k)} = p_B^{(k)} + p_P^{(k)}\ [\text{mb, hPa}], \quad (3)$$

where p is the pressure at level k, p_B is the base pressure and p_P is the pressure perturbation.

- **Temperature, T (K, ºC).**

Variables available for "direct use" from the wrfout* files (without any additional calculations) are the temperature (T_2) at 2 m height and the potential temperature (θ_2) at 2 m height. Vertical profiles are available for θ only. The vertical profile of T can be calculated using:

$$\theta = T \ln\left(\frac{p_0}{p}\right)^{(R/c_p)}\ [K] \quad (4)$$

Here, R/c_p = 0.286 is the Poisson constant, where $R \approx 8.3145$ $JK^{-1}mol^{-1}$ is the universal gas constant, and $c_p \approx 29.14$ $JK^{-1}mol^{-1}$ is the specific heat capacity of air at constant pressure p_0 = 1000 hPa.

After some rearrangements and taking under consideration some specifics in the way the potential temperature is presented by WRF, for T at any given level k we have:

$$\mathrm{T}^{(k)} = \left(\theta^{(k)} + 300\right) \ln\left(\frac{p^{(k)}}{p_0}\right)^{0.286} [K] \tag{5}$$

where $p^{(k)}$ is the pressure at level k, and $\theta^{(k)} + 300$ is the total potential temperature at level k, with base part of value *300* K and perturbation part θ.

- **Friction velocity (m s^{-1}).**

As described in "AERMOD: Description of model formulation" (AERMOD, 2004), the calculation of the friction velocity is not a trivial task, and is done by iteration procedures until convergence is reached. However, u_* is readily available in the wrfout* files.

- **Sensible (H) and latent (λE) heat fluxes (W m^{-2}).**

The values of H and λE are required for the calculation of the Bowen ratio

$$B = \frac{H}{\lambda \mathrm{E}}, \tag{6}$$

which determines what part of the heat is transferred from earth's surface to the atmosphere by means of direct heating (diffusive and/or turbulent), and what part – by indirect (latent, hidden) heating (spent for evaporation and/or transpiration and later – when meeting certain conditions – released through condensation). Thus, B is also an indicator of the type of the underlying surface. According to AERMOD's formulation, H can be calculated as:

$$H = \frac{0.9R_n}{\left(1 + \frac{1}{B}\right)} \left[\mathrm{Wm}^{-2}\right], \tag{7}$$

where R_n is the net radiation (W m^{-2}).

The sensible heat flux plays great role in determining the Monin-Obukhov's length:

$$L = -\frac{\rho c_p \mathrm{T} u_*^3}{\kappa g H} [\mathrm{m}] \tag{8}$$

L is one of the recommended for use variables by AERMOD. H, λE andare readily available in WRF prognostic output files, T can be easily derived as shown in (5) and $k = 0.4$ is the von Kármán constant. However, the density of the air ρ has to be calculated separately. This problem can be solved by using the formula:

$$\rho^{(k)} = \frac{p^{(k)}}{R_S T^{(k)}} \frac{1 + w^{(k)}}{1 + 1.609 w^{(k)}} \left[\mathrm{kgm}^{-3}\right], \tag{9}$$

Here, $R_s = 287.058$ Jkg^{-1}K^{-1} is the specific gas constant for dry air and $w^{(k)}$ is the water mixing ratio in the k^{th} vertical layer (kg kg^{-1}).

- **Convective velocity scale *(m s^{-1})*.**

Unlikeis not available in the wrfout* files. Therefore, it has to be calculated using the relation employed by the meteorological pre-processor AERMET:

$$w_* = \left(\frac{gHz_{ic}}{\rho c_p T}\right)^{\frac{1}{3}} \quad (10)$$

where c_p = 0.001005 Jkg^{-1}K^{-1} is the specific heat capacity at constant pressure, and z_{ic} is the height of the convective boundary layer (CBL).

- **Cloud cover (tenths).**

The variable named "CLDFRA" in the WRF output contains information about the cloudiness in each cell on each level in the calculation grid. This quantity is presented by a coefficient in the range between 0 and 1, where 1 corresponds to fully clouded cell, and 0 – to absence of clouds. Therefore, in order to be determined the cloud cover over an observational point situated on the ground, the total sky area visible from that point has to be considered. However, the choice of an appropriate area size is what makes this task difficult. In many cases the cloud cover occupies not one, but several levels, so the distance between the observer and the low level clouds situated at a point over the horizon, is much shorter than the distance to the high level clouds at the same point. In this exercise's particular case, a simple algorithm for determining the total cloud cover is applied:

```
1 cells_all_lvl = f.variables['CLDFRA'][0, :, 77:88, 87:112].flatten()
2 cells_one_lvl = f.variables['CLDFRA'][0, 0, 77:88, 87:112].flatten()
3
4 for i in range(32):
5     for j in range(len(cells_one_lvl)):
6         if cells_one_lvl[j] < cells_all_lvl[i * j]:
7             cells_one_lvl[j] = cells_all_lvl[i * j]
8
9 clfrac = int((np.sum(cells_one_lvl) / len(cells_one_lvl)) * 10)
```

On the first line of the script, the values of all grid cells on all the 32 levels covering Sofia valley and the surrounding mountains are assigned to the variable cells_all_lvl. The values from the lowest level are assigned to cells_one_lvl. In the following loop (lines 4 to 7) the values of the bottom cells are compared to the ones lying over them. If values greater than 0 exist in one or several cells in a vertical column, the greatest of them is assigned to the cell at the bottom level. On line 9, the sum of all updated bottom level values is divided to the number of cells in that level, and then multiplied by 10 to get the cloud cover in tenths.

An issue with the accuracy of the proposed algorithm could be the fact, that the calculations are made for cells arranged in vertical columns. This means, that if we use the point of view of each of the bottom level cells, looking in vertical direction only

(Fig. 3, left), in most of the cases, the calculated cloud cover would be different from that received by the observer's point of view (Fig. 3, right).

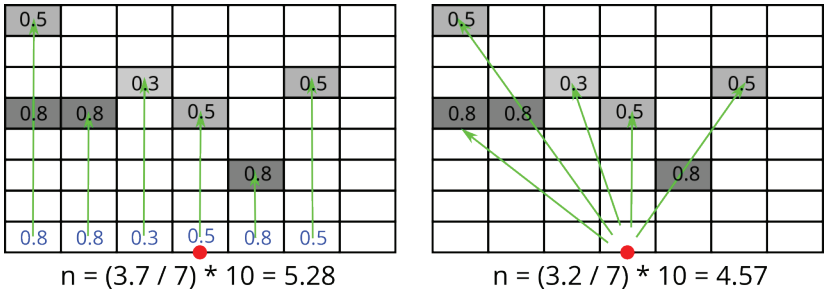

Fig. 3. An example of cloud cover calculation. *Left*: using bottom cells' points of view; *right*: using the observer's (red dot) point of view.

On the other hand, AERMOD accepts meteorological data from just one point, which is deemed representative for the entire modelling domain. In this particular case, an averaged value of cloud cover *n* over the whole territory of the Sofia valley (as shown in fig. 3, left) is used, which guarantees representativeness for any point of choice, at least in regard to that meteorological element.

5. RESULTS

The forecast of PM_{10} pollution distribution by the WRF-AERMOD system, on the territory of Sofia municipality, for the 11th, 12th, and 13th of December 2020 is shown on Figure 4. The domain area covered by AERMOD is 950 km^2 (38 x 25 km). The affected areas in square kilometers, for different threshold values of PM_{10}, as well as the percentages of these areas from the entire domain area are presented on Table 1.

Table 1. WRF-AERMOD system forecast for PM_{10} pollution affected areas (km^2, %) on the territory of Sofia municipality, for values over 100, 75, 50, 25, 10 and 5 μgm^{-1}, on the 11th, 12th, and 13th of December 2020. The values in brackets show the part of affected areas (in percent) from the whole domain area. C_{PM10} denotes the PM_{10} concentration.

	Affected area (km^2), (%)		
C_{PM10} (μgm^{-1})	**11.12.2020**	**12.12.2020**	**13.12.2020**
> 100	0.25 (0.03%)	0.0 (0.00%)	0.0 (0.00%)
> 75	1.25 (0.13%)	0.25 (0.03%)	0.0 (0.00%)
> 50	6.5 (0.68%)	4.5 (0.47%)	0.25 (0.03%)
> 25	32.0 (3.37%)	22.5 (2.37%)	9.25 (0.97%)
> 10	140.75 (14.82%)	110.5 (11.63%)	55.75 (5.87%)
> 5	277.75 (29.24%)	205.75 (21.66%)	149.0 (15.68%)

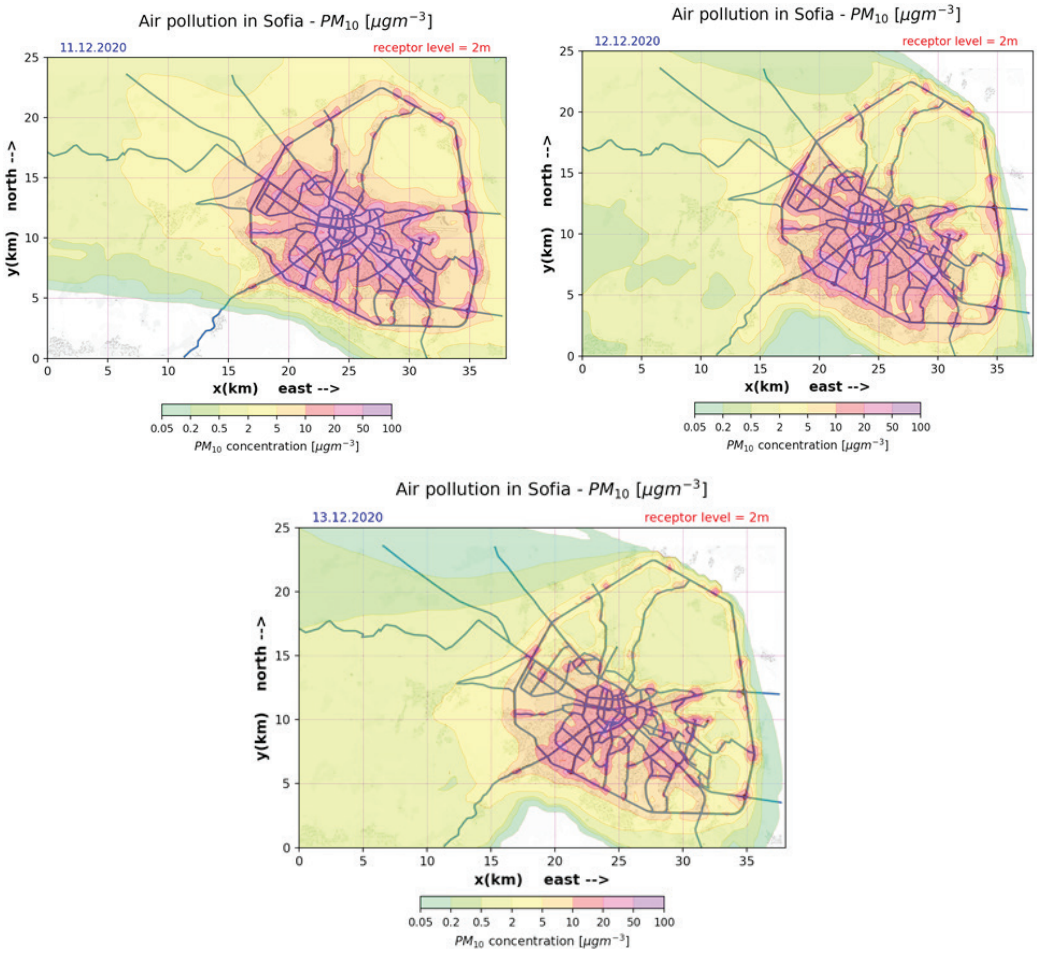

Fig. 4 WRF-AERMOD PM_{10} pollution forecast for 11and 12.12.2020 (top), and 13.12.2020 (bottom).

As seen on Table 1 and in Figure 4, the highest forecasted PM_{10} concentrations are observed on 11.12.2020, and are gradually decreasing in the next two days. Since the emissions from the sources were set to be a constant value in time, the change in meteorological conditions dictated by WRF remains the only reason for the change in the concentrations of the pollutant.

Again, the Python programming language is used for the preparation of the maps in Figure 4. An example code with clarifying comments is presented below:

```
1 from pylab import *
2 import matplotlib.pyplot as plt
3 import matplotlib.colors as col
4 import numpy as np
```

```
import os
import scipy.ndimage
from dataprocessor import DataProcessor
from hex2dec_norm import rgba_nor

# List the AERMOD output in the directory
path = 'output/24hr_averages/'
filelist = [f for f in os.listdir(path) if f.endswith('.dat')]
filelist.sort()

# pollution levels to be plotted:
levels = [0.05, 0.2, 0.5, 2, 5, 10, 20, 50, 100]

for i in range(len(filelist)):
  # Reshaping data for contour plot:
  grid = DataProcessor(path + filelist[i]).contour()

  # Interpolating/smoothing the grid data:
  grid = scipy.ndimage.zoom(grid,
                  order = 1,
                  zoom = [1.1, 1.1],
                  mode='constant',
                  cval = 0.12)

  # Assign labels to x and y axis:
  xlabel(u'x(km)   east -->', fontsize=12, weight=600)
  ylabel(u'y(km)   north -->', fontsize=12, weight=600)

  # An easy way to adjust the figure to the required domain size:
  plt.xlim(0,38000)
  plt.ylim(0,25000)

  # Background map of Sofia:
  im = plt.imread('sofia_base.jpg')

  # Background adjustment:
  implot = plt.imshow(im,
              extent=(-3100,38400,-1400,25200),
              alpha=1.0,
              label='')

  # Plot the contour lines:
  CP = plt.contour(grid,
            levels,
            extent=(0,38000,0,25000),
            colors='red',
            linewidths=0.1)

  # Fill the contours
  CP1 = plt.contourf(grid,
              levels,
              extent=(0,38000,0,25000),
              #RGBA colors: (Red,Green,Blue,Transparency)
              colors=[(rgba_norm('33ff77',1)),
                  (rgba_norm('66ff00',1)),
                  (rgba_norm('ccff00',1)),
                  (rgba_norm('ffff00',1)),
```

```
                    (rgba_norm('ffaa00',1)),
                    (rgba_norm('ff0000',1)),
                    (rgba_norm('ff00aa',1)),
                    (rgba_norm('9900ff',1))], alpha=0.3)

# Set x and y axis to km instead of meters:
plt.xticks(np.arange(0, 38000, 5000), np.arange(0, 38, 5))
plt.yticks(np.arange(0, 26000, 5000), np.arange(0, 26, 5))

# Plot a grid on the map:
plt.grid(which='both',
      color='#aa33aa',
      linestyle='-',
      linewidth = 0.2)

# Plot the colorbar:
cbar = plt.colorbar(orientation='horizontal',
            fraction=0.03,
            pad=0.12,
            shrink=0.8,
            ticks=levels)

# Set colorbar labels and ticks:
cbar.set_label(u'$PM_{10}$ concentration $[\mu g m^{-3}]$')
cbar.ax.set_xticklabels([str(levels[k]) for k\
                in range(len(levels))])

# Set the font size of the colorbar ticklabels
for t in cbar.ax.get_xticklabels():
  t.set_fontsize(9)

# Plot the roadmap on top of the other layers:
roads = plt.imread('sofia_roads.png')
roadplot = plt.imshow(roads,
            extent=(-100,37700,0,23700),
            alpha=1.0,
            label='',
            zorder = 0)

 # Set text decorations:
 timetext = filelist[i][0:2] + '.' +\
        filelist[i][3:5] + '.' +\
        filelist[i][6:10]
 text(7500,27500, u'Air pollution in Sofia - $PM_{10}$\
    $[\mu g m^{-3}]$', fontsize=14)
 text(1000,25500, timetext, color='b')
 text(27000, 25500, u'receptor level = 2m', color='r')

 # Get the current figure, set its size, and save it to a file:
 figure = plt.gcf()
 figure.set_size_inches(7, 6)
 figure.savefig(path+'images/'+filelist[i][:-4]+'.png', dpi=192)

 # Clears current figure, otherwise it plots over the next one:
 plt.clf()
```

6. CONCLUDING REMARKS

Within the last few years, Python has grown to new heights and today, it is truly one of the most utilized program languages available. As to date, it is tying with Java as the second most popular programming language behind JavaScript. Python has been rising across several programming language popularity indexes, including Tiobe and IEEE Spectrum. The Python programming language owes its attractiveness to the features it possesses, and they are:

- Python is simple to execute and easy to learn;
- Availability of quality libraries, which boost Python's development capabilities;
- Efficiency, speed, reliability, and flexibility;
- Suitable for **software automation**, which is one of the main reasons Python to be chosen as programming language for this exercise;
- It has support from large experienced community and by big sponsors.

ACKNOWLEDGEMENTS

This study is made with the financial support of the National Program "Young Scientists and Postdoctoral Students" of the Ministry of Education and Science of the Republic of Bulgaria, under contract №: **ЧР-04-20 /12.05.2020**.

REFERENCES

AERMOD (2004). AERMOD: Description of model formulation, EPA-454/R-03-004, September 2004, 91 pp.

Cimorelli, A. J., Perry, S. G., Venkatram, A., Weil, J. C., Paine, R. J., Wilson, R. B., Lee,R. F., Peters, W. D., and Brode, R. W. (2005). AERMOD: A Dispresion Model for Industrial Source Applications. Part I: General Model Formulation and Boundary Layer Characterization, Journal of Applied Meteorology and Climatology, Volume 44: Issue 5 (01 May 2005), pages 682-693.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

JournalDev, (2020), https://www.journaldev.com/34415/history-of-python-programming-language#history-of-python.

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Liu, Z., Berner, J., Wang W., Powers, J. G., Duda, M. G., Barker, D. M. and Huang, X.-Y., (2019). A Description of the Advanced Research WRF Version 4. NCAR Tech. Note NCAR/TN-556+STR, 145 pp. doi:10.5065/1dfh-6p97.

Tanenbaum, A. S., Kaashoek, M. F., Van Renesse R., Bal, H. E. (1991). The Amoeba distributed operating system – A status report, Computer Communications, Volume 14, Issue 6, 1991, Pages 324-335, ISSN 0140-3664, https://doi.org/10.1016/0140-3664(91)90058-9.

The NCAR Command Language (Version 6.6.2) [Software]. (2019). Boulder, Colorado: UCAR/NCAR/CISL/TDD, http://dx.doi.org/10.5065/D6WD3XH5.

Van Rossum, G., (2009). A Brief Timeline of Python. http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html; Retrieved November 29, 2020.