LIFE13 ENV/IT/001254

Dynamap GIS based software for real time noise maps update

LIFE – DYNAMAP

Dynamic Acoustic Mapping – Development of low cost sensors networks for real time noise mapping

Deliverable Number and Title:	**B4 – Dynamap GIS based software for real time noise maps update**
Action Number – Title:	B4 – Software development for dynamic noise mapping
Dissemination Level:	R (Restricted to Beneficiaries)
Status:	Update
Release Date:	25/06/2018
Author(s):	Andrea Cerniglia, Markus Petz, Robert Geberstein
Reviewer(s):	Jochen Schaal, Michael Gillé, Cristina Ferrari
Document code:	**LIFE-DYNAMAP_ACCON_ B4- Dynamap GIS based software for real time noise maps update Update – 25/06/2018**

Contact person:	Andrea Cerniglia
Postal address:	Greifenberg Germany
Telephone:	+49 (0) 8192 99600
Fax:	+49 (0) 8192 996029
E-mail:	andrea.cerniglia@accon.it
Project Website:	www.life-dynamap.eu

KEYWORDS

Dynamap, Noise, End, 2002/49/CE, GIS, Mapping

TABLE OF CONTENTS

TABLE OF PICTURES

1. INTRODUCTION

The LIFE-DYNAMAP project (Dynamic Acoustic Mapping - Development of low cost sensors networks for real time noise mapping) aims at developing an automatic dynamic noise mapping system able to detect and represent in real time the acoustic impact due to road infrastructures. The scope of the project is the European Directive 2002/49/EC relating to the assessment and management of environmental noise (END), referring to the need of updating noise maps every five years, as stated in the END.

The system is composed of low cost sensors measuring the sound pressure levels emitted by the noise sources present in the area to be mapped and of a software tool based on a GIS platform performing real time noise maps updating.

In DYNAMAP, a specific noise monitoring system network was completely developed and the installation was completed both for Rome (May 2017) and Milan (November 2017) areas.
The project also implements an algorithm for automated recognition of anomalous events, which was already completed and right now is under a final tuning phase.

This report refers to Action B4 (Software development for dynamic noise mapping), and more in details to the Task B4.1 – Development of a GIS based software for real time noise maps update. In this action, a detailed investigation on the computational hardware capabilities combined with different operating systems and tools was developed to identify the best compromise between costs and efficiency of the needed configuration. The present report is part of the deliverable related to task B.4. This document describes hardware, software and routines for collecting, storing, managing data for the implementation of dynamic noise maps.
Since the previous document from January 2016, both hardware and software characteristics changed due to special requests and improvements emerged during the development of the whole project, so this document replaces the previous one. Figure 1.1 shows the actual structure of the system.

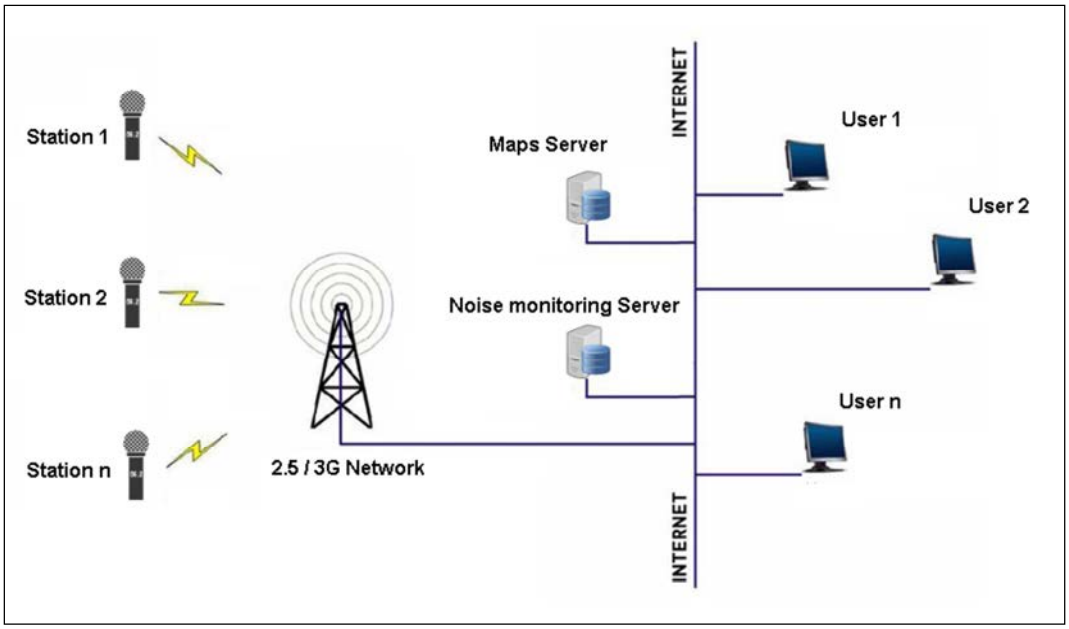

Figure 1.1: Dynamap system structure

More in details, each of two systems (Rome and Milan) systems consists of two physical servers, one for noise data collection and one for map scale and sum. Both servers have also other functions as displayng acquired data, making calculation on them, etcetera. The server relevant for the present report is the Map server. This is better shown in Figure 1.2.

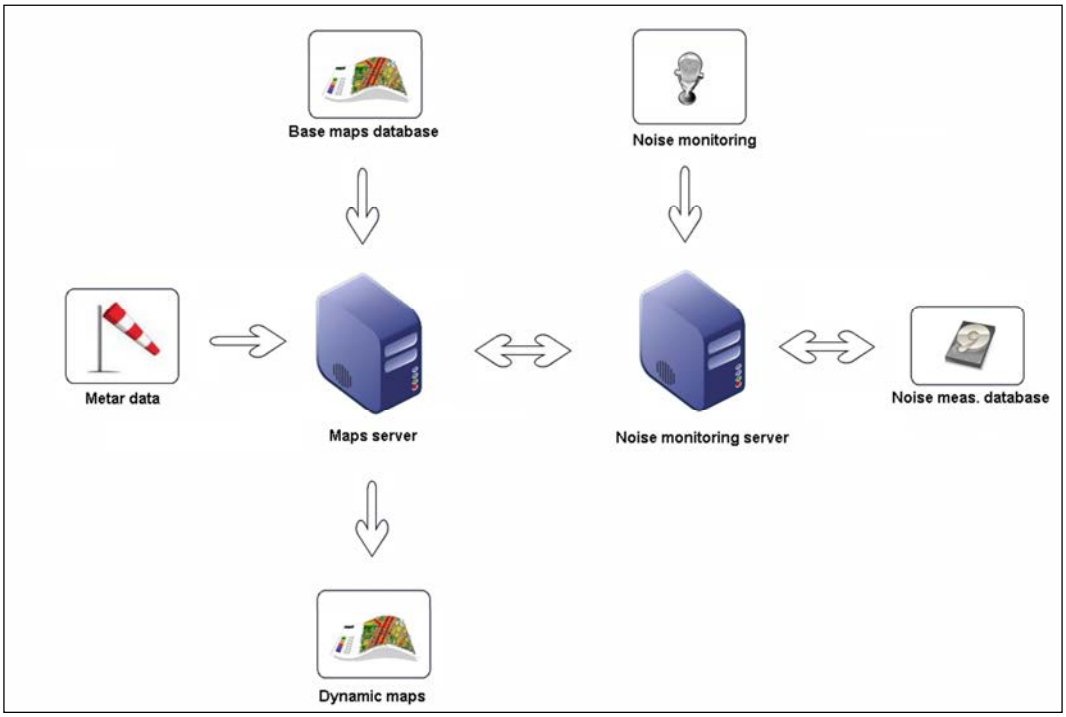

Figure 1.2: Server architecture

Under the domain www.dynamap.eu is located a web page from which is possible to chose the Rome or Milan portal. Figure 1-3 shows the above mantioned main page. The three icons points respectively to the dynamap project page, Rome GIS page and Milan GIS page.

Figure 1-3: Main page

2. HARDWARE

The main idea in the selection process for computational hardware and software was to define a cheap, open, scalable and reliable configuration for Dynamap system implementation. In order to do that, different kinds of tests involving different machines, operating system and tools were checked. In addition, some stress test were managed to check the behavior of the whole configurations under possible critical situations. The final configuration is based on Linux machines with different hardware characteristics for Rome and Milan systems as following:

Rome machine

Microprocessor	Intel® Xeon® E3-1230 v6
Clock	4x 3.5GHz
RAM	32GB (DDR4)
HDD	2xSSD 480 MB

Milan machine

Microprocessor	Intel(R) Pentium(R) CPU J4205
Clock	4x 1.5GHz
RAM	8GB (DDR3)
HDD	1 x 250 MB

The reason for having machines with different hardware characteristics for the two pilot areas is due to the fact that the Rome system needs much more computation power because of the more power-consuming calculation, whereas the Milan system does not need that much power. In fact, the Rome system should be capable of publishing a new noise map every 30 seconds, as sum and scale of 19 base maps, whereas the Milan one should publish a map only every 5 minutes as scale and sum of 6 base maps. For both systems the available bandwidth has a flat rate of 100 MBs/s (the evaluation about possible needs of more band will be possible after the official launch of the portals, according to the average number of visitors).

2.1. Ports

For the correct behaviour of the system and for administration purposes, the following ports should be mantained opened :

- 22 (tcp): server administration
- 80 (tcp): web pages
- 123 (udp) : time synchronisation
- 8080 (tcp): Geoserver
- 3306 (tcp): MySQL admin. (can be normally closed, and opened only when needed)
- 5432 (tcp): Postgres admin. (can be normally closed, and opened only when needed)

3. SOFTWARE CONFIGURATION

The following chapters will describe the software configuration of the systems, in terms of operating systems, databases, scripting languages, etcetera.

3.1. Operating System

In order to have an open and reliable system, the Linux system was chosen. The installed distribution on both machines is the standard distribution Ubuntu, version 5.4.0.6. The choice of an open source OS permits, in case of needs, special requirements or malfunctioning, to go deep in details on the OS routines. Moreover, the Linux system is free and thus the systems can be easily implemented without any additional costs.

3.2. Webserver

The installed webserver is the standard Apache 2 (2.4.18) configured to work with both PHP and Phyton. Apache is a very stable and well-known web server appreciated all around the world.

3.3. Databases

For a better separation of the tasks related to the GIS system from the tasks related to standard operations (web sites administration, data storing, offline calculation, etc.), it was decided to implement two separate databeses. More in details, the GIS operations are based on PostgreSQL database with PostGIS extension, whereas all the other tasks are based on MySQL. In the Dynamap systems PHP interacts together with MySQL, whereas GIS interacts together with PostgreSQL.

3.3.1. PostgreSQL

PostgreSQL is a reliable free database, with a very good support of geographical data through the use of the PostGIS extension. The basic maps for dynamic map generation are stored in PostgreSQL tables. The installed PostgreSQL version is 9.6.4.

3.3.2. MySQL

MySQL is a free general purpose database easily well connectable by PHP. In the system, this database is used to store and retrieve all the information related to website administration, including management of access privileges, and for some off-line calculation; for this last purpose, a copy of the base maps is stored also in the MySQL dBase. The installed MySQL version is 5.7.20.

3.4. Scripting languages used by the system

In order to perform all the needed operation, some procedures are necessary. few of them are automated procedures scheduled with specific time intervals, whereas all the others are procedures triggered by visitors' actions and/or requests.

3.4.1. PHP

PHP is a free general purpose 'C like' scripting language, implemented in most of the Linux distribution. As already mentioned, it has a complete set of instructions to efficiently interact with MySQL, but it is not so powerful for management of complex calculation. In the Dynamap system, PHP is used for all the operations, excluding GIS manipulation. The installed PHP version is 7.0.22.

3.4.1.1. PHP extensions

In order to perform the required Dynamap operations, some PHP extensions are mandatory. These are:

- CURL: it is used to communicate with the noise data server and METAR data server
- GD: it is used for graphics generation in barrier simulation routine
- JSON: it is used for data exchange
- MSQLI: it is used to interface MySQL

3.4.2. Phyton

Phyton is used by the system to communicate with GIS and launch noise map calculation. The installed version is 2.7.12.

3.4.3. Javascript

Javascript is used on several pages for different purposes as event management, cookies management, colors management, etcetera.

3.5. Data exchange format

In order to perform its calculation, the Dynamap machine should retrieve data from different external servers as the measured noise levels and the meteorological data. For data exchange, three different formats are employed.

- CURL: this is used to retrieve noise data from noise measurement server as well METAR data from airport meteorological stations

- JSON: this format is used to manage data for GIS computing

- GeoJSON: this format is used to represent simple geographical features, along with their non-spatial attributes.

3.6. GIS

The GIS sub system takes care of map representation including auxiliary information according to the selected layers. The software installed for this purpose is Geoserver 2.11.0

3.7. GDAL

GDAL is library for reading and writing raster and vector geospatial data format. It is needed by Geoserver.

3.8. JDK and JRE

The Java development kit is the Java platform whereas Java Runtime Environment is the corresponding runtime platform. They are used by Geoserver.

4. DATABASES STRUCTURE

As already mentioned, the system uses two different database engines: PostgreSQL for GIS operations and MySQL for general purpose operations and off-line calculations.

4.1. PostgreSQL

For the management of dynamic map generation, PostgreSql needs several tables as listed below:

4.1.1. Buildings

The database 'buildings' contains all the information about the buildings such as ID, number of inhabitants, noise limits, etcetera. Some information is mandatory (as the above cited), whereas other is intended as a support for possible future use (i.e. name of the street, property, etcetera). Computed results for each building are stored in this table. The total number of columns is 38 and the mandatory fields are:

- gid (integer): ID
- geobid (character varying 15): building ID
- n_occupan (numeric): number of occupants
- lim_g (numeric): day limit
- lim_n (numeric): night limit
- dlgs_now (double precision): computed level for Italian law
- ue_now (double precision): computed level for 2002/49/CE
- altezza_dlgs (double precision): height for max level calculation

4.1.2. locations

The database 'locations' contains all the information about the measuring points such as ID, address, measured level, etcetera. All the values come from the MySQL database and are updated every 30 seconds. The total number of columns is 14 as described below:

- gid (integer): ID
- id (integer): location ID
- ref (character varying 80): number of stations to which it is referred
- station_id (character varying 80): name of the monitoring station
- height (character varying 80): height of the measuring point
- address (character varying 80): address of the monitoring station
- visible (integer): flag according to the visible/not visible status
- status (character varying 80): status of the monitoring station (working/not working)
- base (numeric): base level
- laeq (numeric): measured level L_{Aeq}
- deltalaeq (numeric): difference between base level and L_{Aeq}
- la95 (numeric): measured statistical level L_{A95}
- hi (numeric): harmonica index
- geom (geometry): position of the point

4.1.3. others

The Dynamap system can manage also extra points for customized purposes (managed by PHP/MySQL). Information about these points are stored in the 'other' table in order to represent them in the GIS system. Data about the points coming from the MySQL database and are updated every 30 seconds. The total number of columns for each point is 6 as described below:

- gid (integer): ID
- id (integer): location ID
- address (character varying 80): address
- link (character varying 80): link to a web page with description or other information
- type (integer): flag according to public/private status
- geom (geometry): position of the point

4.1.4. Base maps

The Dynamap system deals with a number of base maps that are different for Rome and Milan implementations. Below there is a description of the tables for the two pilot areas.

4.1.4.1. Rome

Each point of the Rome dynamic map is a combination of 19 base maps, each of them connected with the noise coming from an arch of the road monitored by a noise monitoring station. More in details, the levels represented by the dynamic map are computed as sum of the 19 base maps, each one scaled according to the difference between the predicted level at the measuring point and the measured one. Because of the influence that meteorological conditions can have on the noise propagation and because of the different traffic conditions that can influence the acoustic climate of the area, there is not just a set of 19 base maps but there are several. More in details, the considered number of meteorological conditions is 6 whereas the considered number of traffic conditions is 2, for a total of 12 possible combinations. In Dynamap the horizontal maps are treated as raster, so they are not present in the database but they are imported directly in Geoserver; as for the vertical maps, the total number of sets is doubled because of the different calculation parameters according to the Italian legislation and 2002/49/CE, thus the total number of base map sets is 24, each one consisting in 19 base maps. Before each calculation Dynamap selects the correct set of maps according to the meteorological and traffic conditions. The name of the vertical tables in the database are x_favorable_yy, xx_north_yy, xx_east_yy, xx_south_yy, xx_ west_yy, xx_homogeneous_yy, where xx can be 'wd' for working days or 'we' for the weekend and yy can be 'dl' for Italian calculation method or 'eu' for 2002/49/CE . All the tables have the same structure as described below:

- gid (integer): id
- id (integer): location id
- height (numeric): height
- alt.ter (numeric): terrain height
- piano (character varying 80): floor
- id edif (integer): building id
- utilizzo (character varying 80): type of building
- direzione (character varying 80): direction of the most exposed facade

- V01 (numeric): base level map 1
- ……(numeric): base level map m
- ……(numeric): base level map n
- V19 (numeric): base level map 19
- geom (geometry): position of the point

4.1.4.2. Milan

Milan is more simple than Rome because the meteorological and traffic conditions are not taken into account. The total number of sets for Milan is thus 2 with 6 maps each (1 set for calculation as requested by the Italian legislation, and 1 set for the 2002/49/CE). The structure for the vertical map is the same as for Rome:

- gid (integer): id
- id (integer): location id
- height (numeric): height
- alt.ter (numeric): terrain height
- piano (character varying 80): floor
- id edif (integer): building id
- utilizzo (character varying 80): type of building
- direzione (character varying 80): direction of the most exposed facade
- V01 (numeric): base level map 1
- ……(numeric): base level map m
- ……(numeric): base level map n
- V19 (numeric): base level map 19
- geom (geometry): position of the point

4.2. MySQL

As already written, MySQL is used in Dynamap for general purpose web site management as well as for some non GIS calculation. Even if the purpose of this document is to describe the action B4.1, due to the fact that all the parts of the system are strictly connected together, in the following chapters all the MySQL databases will described for a better understanding of the whole system.

4.2.1. Dynamap_Access_log

This database contains a table named 'log' that collects information about the website visits. The structure of 'log' is the following:

- ID (int 11): autoincrement ID
- timestamp (int 11): unix timestamp (date of visit)
- IP (varchar 50): IP address of the visitor
- user (varchar 100): user name (if logged)
- page (varchar 100): visited page
- address (varchar 100): address of the visited page
- agent (varchar 100): user agent
- refer (varchar (100): refer page

4.2.2. Dynamap_Airatt

Dynamap has many utilities for both unregistered and advanced users, and some of them require database for their functions. AirAtt contains 10 tables, all with the same structures and named from Ur_010 to Ur_100, which store the attenuation of sound in the air at various frequencies and temperatures, for air humidity between 10% and 100%. The data stored in each table is in terms of dBs/100m and lines store temperatures from -10 °C to 40 °C in 5° steps (11 lines/table). This is the structure of each of the ten tables:

- id (int 11): ID
- temp (int 11): temperature
- f_50 (char 4): attenuation for 50 Hz band
- f_63 (char 4): attenuation for 63 Hz band
- ……(char 4): attenuation for m band
- ……(char 4): attenuation for n band
- f_12500 (char 4): attenuation for 12500 Hz band

4.2.3. Dynamap_Basic_maps

The Dynamap_Basic_Maps database contains the tables with the sets of the horizontal basic maps. In MySQL these sets are used for off-line calculation of the user-definable points.

4.2.3.1. Rome

The database Rome.Dynamap_Basic_maps contains the 12 sets of the 19 horizontal base maps according to the various meteorological and traffic conditions. The sets are named xx_FAVORABLE, xx_FAVORABLE NORTH, xx_FAVORABLE EAST, xx_FAVORABLE SOUTH, xx_FAVORABLE WEST, xx_HOMOGENEOUS, where xx can be WD for working days or WE for the weekend. All the tables have the same structure as described below:

- ID (int 8): ID
- Point_ID (int 8): Point ID
- Lon (int 7): Latitude
- Lat (int 7): Longitude
- Height (int 6): Height
- V01 (int 8): Power value for arch 1
- V02 (int 8): Power value for arch 2
- .. (int 8): Power value for arch m
- .. (int 8): Power value for arch n
- V19 (int 8): Power value for arch 19

4.2.3.2. Milan

The database Milan.Dynamap_Basic_maps contains one set with the 6 horizontal base maps, all with the same structure as the Rome maps.

4.2.4. Dynamap_Configuration

The Dynamap configuration database contains four tables for configuring some calculation parameters, colours for Harmonica and LAeq scale, registered users.

4.2.4.1. 'Table calc'

This table contains a single line with some relevant parameters:

- ID (int 11): ID
- Delay (int 5): delay between actual time and requested measured time
- LA95CalcTime (int 5): time period for LA95 calculation

4.2.4.2. Table 'HarmonicaScales'

Table HarmonicaScales contains 4 lines; line 1 is related to HI colours for day period, line 2 is related to Hi colours for night period, whereas line 101 and 102 contain the default values for the Harmonica Index. The data structure is the following:

- ID (int 11): ID
- C00 (varchar 6): RRGGBB colour for step 1
- C01 (varchar 6): RRGGBB colour for step 2
- ….. (varchar 6): RRGGBB colour for step n
- C11 (varchar 6): RRGGBB colour for step 11

4.2.4.3. Table 'LAeqScales'

Table LAeqScales contains 2 lines; line 1 is the LAeq colour scale, whereas line 101 contains the default values for the above. The data structure is the following:

- ID (int 11): ID
- C00 (varchar 6): RRGGBB colour for step 1
- ….. (varchar 6): RRGGBB colour for step m
- ….. (varchar 6): RRGGBB colour for step n
- C11 (varchar 6): RRGGBB colour for step 11

4.2.4.4. Table 'user'

The table users contains as many lines as the number of registered users. The structure of the table is the following:

- ID (int 4): ID
- user (varchar 30): user
- pass (varchar 10): password
- Name (varchar 15): Name of the user
- Surname (varchar 15): Surname of the user
- email (varchar 50): email of the user
- created (timestamp): date of creation
- permission (tinyint): mask for management of permission (8 possible levels)
- active e (char): flag for the status (active/no active)

In order to grant different privileges to different users, the permission field value can assume each value between 0 and 255. The value 0 means that the user can only see the public pages plus some tools in the reserved page, whereas values greater than 0 grants different privileges as below better specified. More in details, the meaning of the eight bits of the mask are tehe following:

- Bit 0 (weight 1): user with access only to real time meteorological and noise data
- Bit 1 (weight 2): user with access only to historical meteorological and noise data
- Bit 2 (weight 4): user with access only to real time L_{Aeq} noise maps
- Bit 3 (weight 8): user with access only to historical L_{Aeq} noise maps
- Bit 4 (weight 16): user with access only to administration pages
- Bit 5 (weight 32): user with access only to Server status and log data
- Bit 6 (weight 64): reserved for future use
- Bit 7 (weight 128): reserved for future use

In other words, an user with permission equals to 3 can see only real-time data (meteorological and noise) and real-time L_{Aeq} map. Another user with permission equal to 10 can see only historical data (meteorological and noise) and historical maps. The page that grants privileges can efficiently manage users privileges through a graphical interface as in the following picture. The system permits to have only one administrator which is not deletable.

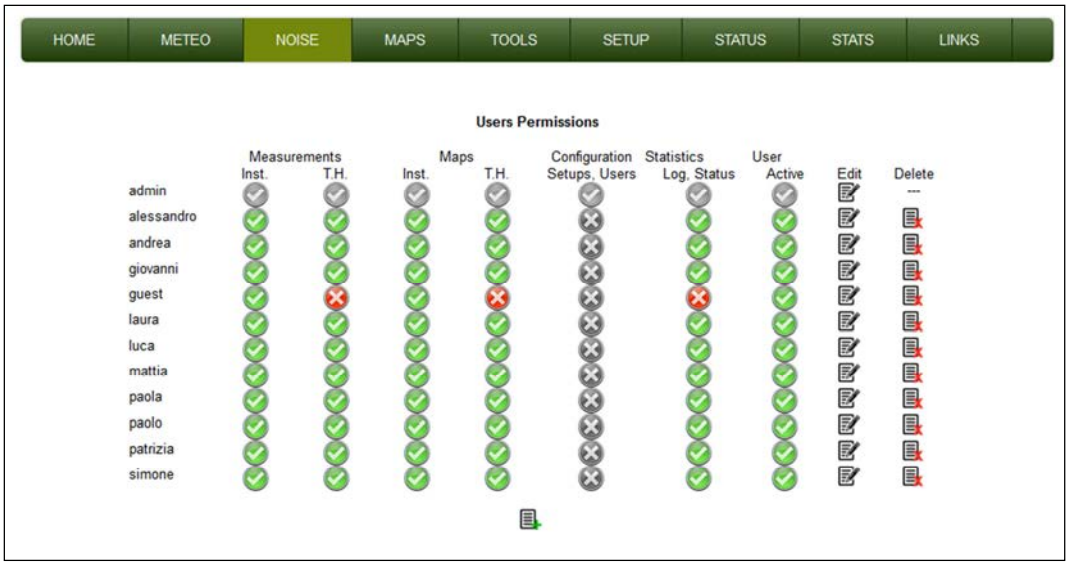

Figure 4.1: User administration page

4.2.5. Dynamap_Glossary

Dynamap_Glossary dabase consists on two tables named 'english' and 'italian' with some explanation of acoustical definitions for unregistered users. Each table contains three columns as following:

- ID (int 11): ID
- def (varchar 50): definition
- meaning (longtext): explanation (stored as html)

4.2.6. Dynamap_Maps

Dynamap_Maps is the database containing the table with input data for the calculation of the static maps over an user-defined period. The process is different for the Rome and Milan systems.

4.2.6.1. Rome static map calculation

For the requested period (date interval and period of the day), the Dynamap system computes -for each of the 12 possible sets of base maps- the number of occurrences of each set and the corresponding averaged scaling factors for the 19 maps included in each set. Each point P of the final static map over the selected period is thus computed according to the following formula:

$$P = 10log\left(\sum_{i=1}^{19} 10^{P_i/10}\right)$$

Where P_i is the level of each of the 19 contributions coming from the 19 arches of the road and it is equals to:

$$P_i = 10log\left(\sum_{j=1}^{12} c_j 10^{\frac{S_{ij}}{10}}\right)$$

where

S_{ij} is the scaled level for the point *i* and cluster *j*

$$c_i = \frac{N_j}{N_{tot}}$$

N_j = number of occurrences per cluster

N_{tot} = total number of occurrences

Data in this table is used from the calculation script in order to compute the requested total maps over the selected period.

4.2.6.2. Milan static map calculation

The process for Milan is more simple because there are just 6 maps to be scaled and summed together, regardless the meteorological or traffic conditions. Each of the 6 maps is related to a group of roads with similar characteristics according the algorithm developed by Università Milano Bicocca.

Each point P of the final static map over the selected period is computed according to the following formula:

$$P = 10log\left(\sum_{i=1}^{6} 10^{\frac{P_i+\delta_i}{10}}\right)$$

where, in the first implementation of the system

P_i is the computed L_{Aeq} at the considered point due to the noise related to group *i*

δ_i is the average of the arithmetic differences between the measured and the computed level for the four monitoring stations related to the group *i*.

4.2.7. Dynamap_Meteo

Dynamap_meteo contains one table named 'Log' which stores METAR data from the Ciampino airport (Rome system) or the Linate airport (Milan system). METAR data is relevant for the Rome calculation, whereas, for Milan, they are acquired but are not used in calculation. The table

contains one line for each collected METAR data (normally one line every 30 minutes). The structure of the table is the following:

- ID (int 11): ID
- timestamp (inty 11): store time
- upd (varchar 40): UTC sample time
- weekend (char 1): weekend flag (0=working, 1=weekend)
- sunrise (time): UTC sunrise time for the relevant location (Rome or Milan)
- sunset (time): UTC sunrise time for the relevant location (Rome or Milan)
- pressure (varchar 6): air pressure in mbar
- temperature (varchar 6): temperature in °C
- humidity (varchar 5): %UR
- speed (varchar 5): wind speed in ms^{-1}
- direction (varchar 50): wind direction in degrees
- clouds (varchar 3): OK, FEW, SCT, OVC or BKN (according METER decode)
- map (varchar 30): relevant base map according to METAR (available only in Rome)
- metar (text): METAR string
- htmlmetar (blob): METAR html

4.2.8. Dynamap_Monitoring

Dynamap_Monitoring contains tables with default values to be used in case of fault for one or more monitoring stations, and one table with relevant data for the installed monitoring stations.

4.2.8.1. Default values

The tables are divided according to period of the year and day of week. More in details, the year was divided in five periods as following:

- Period 1: from 07/01 to 15/05
- Period 2: form 16/05 to 31/07
- Period 3: from 01/08 to 31/08

- Period 4: from 01/09 to 23/12
- Period 5: from 24/12 to 06/12

The reason for this classification is due to the fact that, at least in Italy and in the considered areas, traffic noise is strictly connected with the above periods (i.e. most of the people go on holiday in August, etcetera). The default values are being defined on statistical basis, according to the previous acquired data. The table are thus named Px_default_Yyy where x is the period number (1 to 5) and Yyy is the day of the week (Mon, Thu, Wed, Tue, Fri, Sat, Sun); following this convention, P2_default_Mon is the table valid for a generic Monday in the period between 16/05 and 31/07. The total number of tables is thus 35 (days in a week multiplied by periods of the year). The number of lines in each table is 24 (one for each hour of the day). The structure of each table is the following:

- hour (text): hour of the day (12 means between 12:00:00 and 12:59:59)
- 01_LAeq (int 11): default L_{Aeq} level for station 01
- 01_LA95 (int 11): default L_{A95}level for station 01
- 02_LAeq (int 11): default L_{Aeq} level for station 02
- 02_LA95 (int 11): default L_{A95}level for station 02
- nn_LAeq (int 11): default L_{Aeq} level for station nn
- nn_LA95 (int 11): default L_{A95}level for station nn

The number of columns is 19 for the Rome system (19 monitoring stations) and 24 for the Milan system (24 monitoring stations).

4.2.8.2. Table locations

The table 'locations' contains all the relevant data for the monitoring stations that are 19 for the Rome system and 24 for the Milan system. The structure is the following:

- ID (int 3): ID
- Ref (varchar 6): arch to which the monitoring station refers
- Station_ID (varchar 9): name of the monitoring station
- Lat (varchar 9): latitude of the monitoring station

- Lon (varchar 9): longitude of the monitoring station
- Height (varchar 9): height of the monitoring station
- Address (varchar 50): address of the monitoring station
- Visible (char 1): flag (1 visible, 2 hidden)
- Conf (varchar 10): configuration of the monitoring station
- Firmware (varchar 10): firmware of the monitoring station
- Serial (varchar 10): serial number
- Inst_Date (varchar 8): date of installation
- Cal_Date (varchar 8): date of calibration
- Note (varchar 100): note
- Conn_ID (varchar 16): info about the connection
- Status (Char 1): status (0 ok, 1 fail)
- Base (float): base level at measuring point
- LAeq (float): last measured L_{Aeq} level
- DeltaLAeq (float): difference between base level and measured one
- LA95 (float): last measured L_{Aeq} level
- HI (float): Harmonica Index

Most of the columns have the same name and values as in the corresponding table in the postgreSQL database.

For Milan system the table contains 6 lines more than the number of the noise monitoring stations. These lines, named 101 to 106, store the data for the scaling process of the six maps, and are computed starting from the monitoring data and the base maps themselves, as already above described.

4.2.8.3. Dynamap_Others

Dynamap system permits to put hypertext pointers on the map, for example to identify some relevant information related to specific places. This information is stored in the Dynamap_Others database that has a table named 'locations'. The number of lines in the table is the same as the pointers that should be placed on the map. The table structure is the following:

- ID (int 11): ID
- Lat (varchar 8): latitude
- Lon (varchar 8): longitude
- Address (varchar 50): address
- Link (varchar 50): url
- Type (int 1): flag (1= public, 2=private)

4.2.8.4. Dynamap_Points

The Dynamap system permits to make calculation on a specific point in terms of time history over user-definable half an hour period of the acquired data. In addition to the above, it is also possible to compute the behaviour of several points over one day in terms of 48 half an hour values. The Dynamap Points database contains information about the point that should be calculated. The number of lines is equal to the number of points, whereas the structure of the database is the following:

- ID (int 11): ID
- Lat (varchar 8): latitude
- Lon (varchar 8): longitude
- Address (varchar 50): address
- LAeq (float): computed L_{Aeq}
- LA95 (float): computed L_{A95}

5. SCRIPTS

Dynamap scripts can be divided into two main categories : scripts for automated operation and scripts for manual operation.

5.1. Scripts for automated operation

The system should automatically trigger some operations every defined time intervals. These operations are automatically managed by the use of Linux crontab deamon. The following picture shows the crontab configuration.

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# run-parts

*/5  * * * * root php /var/www/html/auto/meteosave.php
*    * * * * root php /var/www/html/auto/M00.php
*    * * * * root sleep 30; php /var/www/html/auto/M00.php
```

More in details, the first line schedules, every five minutes, the script for meteorological data download (in fact meteorological data is published every thirty minutes, but the operation is scheduled every five minutes in order to avoid data lacks due to temporary unavailable service; the system avoid to store data with the same time information in it).
The second and third lines launch the script for collecting data from the measuring station server every thirty seconds (the third line is equal to the second one, but with a 30 second delay as first commad.

5.1.1. meteosave.php – script to collect meteorological data

The script, located in /var/www/html/auto/ , takes care of collecting meteorological data, computing the correct map that should be used (only for Rome) and storing results on the MySQL Dynamap_Meteo database. The relevant parts of the script are commented and the whole script is reported in appendix 'A'.

5.1.2. M00.php – script to collect noise data

The script, located in /var/www/html/auto/ , takes care of collecting noise data from the noise monitoring server, computing the scaling factor for each map and storing results on the MySQL Dynamap_Monitoring database. In case of missing data from one or more stations, the script collects proper data from the default data tables, according to period of the year, day of the week,

hour of the day and off-line monitoring station(s). The Milan script is different from the Rome one because of the different update times for the two portals (30 seconds between 06:00 and 22:00 and 300 seconds between 22:00 and 06:00 for Rome; 5 minutes between 07:00 and 21:00, 15 minutes between 21:00 and 01:00 and 60 minutes between 01:00 and 07:00 for Milan). The relevant parts of the scripts (Rome and Milan) are commented and the whole scripts are reported in appendix 'B' (Rome) and 'C' (Milan).

5.1.3. Script for horizontal map calculation

For the update of the dynamic horizontal map are used two separate Phyton scripts, one for the Harmonica Index map and the other for L_{Aeq} map. As the horizontal maps must be computed over the entire area of interest with a constant spatial resolution, rasters were identified as the best solution to handle this kind of data.

5.1.3.1. dynamapCore_public.py

The task of this script, executed every 30 seconds using the 'watch –n 30" command, is to calculate and publish the public Harmonica Index map. More in details the script do the following jobs:

- Retrieve the live adjusting parameters and, only for Rome, the set of basemap to use from the MySQL database (Milan has a single set)
- Read the corresponding basemaps as TIF files
- Multiply the basemaps with the coefficients in order to determine the Harmonica maps
- Save the output maps to a TIF file, ready to be processed by the geoserver to create tiles

The script, located in /home , also takes care of updating the sensors value and Syncronize the MySQL and Postgres databases. The relevant parts of the scripts (Rome and Milan) are commented and the whole scripts are reported in appendix 'D' (Rome) and 'G' (Milan).

5.1.3.2. dynamapCore.py

The task of this script, again executed every 30 seconds using the 'watch –n 30" command, is to calculate and publish the private L_{Aeq} map. More in details the script do the following jobs:

- Retrieve the live adjusting parameters and basemap from MySQL database
- Read the basemap
- Compute the LAeq map based on the basemap and the scaling coefficients
- Save the map to a tif file
- Extract the isolines

The script is located in /home , the relevant part are commented and the whole scripts are reported in appendix 'E' (Rome) and 'H' (Milan).

5.1.4. dynamapCoreVertical.py – script for vertical map calculation

For the update of the dynamic vertical map is used a single Phyton script for both IT and END maps, executed every thirty seconds. Due to the heterogeneity and sparsity of the input dataset (buildings), measures have been stored in vectors, where each footprint must be considered with its own parameters derived from the basemap. More in details, the script take care of the following steps:

- Determine the period of the day to select among day or night
- Collect the adjusting coefficients and basemap from MySQL db
- Combine the coefficients with the basemap. Measures must be computed for each single building
- Update the db table according to the computed values

The script is located in /home , the relevant part are commented and the whole scripts are reported in appendix 'F' (Rome) and 'I' (Milan).

5.2. Scripts for manual operations

In addition to the above, many other scripts provide the management of the perations for auxiliary calculations as well as the management of the Dynamap web site. The table below lists the main scripts and their related functions.

File	Type	Category	Use	Task
/var/www/html/@alogin.php	php	-	private	Script to create session
/var/www/html/@alogin_form.php	php	-	private	Login form, for registered users
/var/www/html/@alogout.php	php	-	private	Logout script for registered users
/var/www/html/_inc_menu.php	php	-	system	Unregistered visitor menu, common to most pages (php include)
/var/www/html/ActionPlan.php	php	info	public	Script to browse information about Action Plan
/var/www/html/ANAS_Strategy.php	php	info	public	Script to browse information about ANAS strategy to reduce noise (Rome)
/var/www/html/audio_a.php	php	learn	public	Script to play amplitude mp3 sounds examples
/var/www/html/audio_f.php	php	learn	public	Script to play frequecies mp3 sound examples
/var/www/html/Contribute.php	php	info	public	Script to browse informationn about how to contribute to reduce noise
/var/www/html/cookiechoices.js	javascript	-	system	Cookies management
/var/www/html/cookies.html	html	general	public	Explanation about cookies
/var/www/html/glossary.php	php	learn	public	Script to browse a glossary about noise (dBase based)
/var/www/html/harmonica_style.xml	xml	-	system	xml file with styles for Harmonica representation
/var/www/html/index.php	php	Pub. entr.	public	Index page (entrance of the site)
/var/www/html/leaflet-src.map	javascript	-	system	map manager
/var/www/html/leaflet-image.js	javascript	-	system	map manager
/var/www/html/map.php	php	map	public	Unregistered users Harmonica Index Noise Map browser
/var/www/html/sensors.json	json	-	system	Labels file for monitoring stations
/var/www/html/visitor.css	css	-	system	CSS for unregisterd visitor pages
/var/www/html/admin/@alogout.php	php	-	private	Logout script for registered users
/var/www/html/admin/_inc_log.php	php	-	system	Script for logging visits (recalled by most of the pages)
/var/www/html/admin/_inc_menu.php	php	-	system	Registered visitior menu, common to all other pages (php include)
/var/www/html/admin/_time_search.php	php	-	system	Script for date selection (used by other scripts)
/var/www/html/admin/admin.css	css	-	system	CSS for registerd visitor pages
/var/www/html/admin/AirAtt.php	php	tools	private	Script to browse air attenuation
/var/www/html/admin/Aree_critiche_GRA_new.geojson	geojson	-	system	File with geographical information about Critical Areas
/var/www/html/admin/A-Weighting.php	php	tools	private	Script to browse A-weigthing curve
/var/www/html/admin/barriere_GRA_new.geojson	geojson	-	system	File with geographical infprmation about Noise Barriers
/var/www/html/admin/baseMap.php	php	calc	private	Script to select the day for base map calculation
/var/www/html/admin/baseMap_day_select.php	php	results	private	Script to calculate base map for a given date
/var/www/html/admin/calculation_setup.php	php	setup	private	Script to set up some calculation parameters
/var/www/html/admin/coord_Deg2UTM.php	php	tools	private	Script to convert coordinates from LatLon to UTM
/var/www/html/admin/coord_UTM2Deg.php	php	tools	private	Script to convert coordinates from UTM to LatLon
/var/www/html/admin/custom_map.php	php	results	private	Script to launch the calculation of a map over a given period
/var/www/html/admin/dBLden.php	php	tools	private	Script to calculate Lden
/var/www/html/admin/dBLeq.php	php	tools	private	Script to calcolate Leq

/var/www/html/admin/dBsub.php	php	tools	private	Script to calcolate dB Sub
/var/www/html/admin/dBsum.php	php	tools	private	Script to calcolate dB Sum
/var/www/html/admin/dewpoint.php	php	tools	private	Script to calculate dew point
/var/www/html/admin/divergence.php	php	tools	private	Script to calculate geometric divergence
/var/www/html/admin/edit_locations_edit.php	php	setup	private	Script to edit data fot the measuring locations
/var/www/html/admin/glossary.php	php	tools	private	Script to show a glossary
/var/www/html/admin/gPoint.php	php	-	system	Script for coordinate conversion (used by other scripts)
/var/www/html/admin/jcolor.js	javascript	-	system	Javascript for colo selection
/var/www/html/admin/laeq_map.php	php	results	private	LAeq map browser
/var/www/html/admin/LAeq_select_period.php	php	results	private	Script to select the period for static map calculation
/var/www/html/admin/location_chvis.php	php	setup	private	Script to change the status (visible/not visible) of a measureing point
/var/www/html/admin/location_edit.php	php	setup	private	Script to edit measuring locations data
/var/www/html/admin/locations.php	php	setup	private	Script to manage monitoring ststion info
/var/www/html/admin/log.php	php	status	private	Script to view log file
/var/www/html/admin/main.php	php	Priv. entr.	private	Main private page
/var/www/html/admin/map.php	php	results	private	Harmonica map browser
/var/www/html/admin/map_extract_data.php	php	results	private	Script to extract data for static map calculation
/var/www/html/admin/map_select_begin.php	php	results	private	Script to define begin period for static map
/var/www/html/admin/map_select_end.php	php	tresults	private	Script to select end period fo static map
/var/www/html/admin/map_select_period.php	php	results	private	Scipt to select day, night, den, etc., for static map calc,
/var/www/html/admin/maps_color_set.php	php	setup	private	Script to set colors
/var/www/html/admin/maps_setup.php	php	setup	private	Script to set maps colors
/var/www/html/admin/maps_setup_reset.php	php	setup	private	Script to set maps colors to default values
/var/www/html/admin/measurements.php	php	results	private	Script to show real time measured data
/var/www/html/admin/meteo.php	php	results	private	Script to show real time meteo data
/var/www/html/admin/meteo_hist_complete.php	php	results	private	Script to browse meteo history (full)
/var/www/html/admin/meteo_log.php	php	results	private	Script to browse historical meteo data
/var/www/html/admin/meteo_metar.php	php	results	private	Script to browse metar data
/var/www/html/admin/meteo_metar_log_save.php	php	results	private	Script to save metar data (complete)
/var/www/html/admin/meteo_metar_save.php	php	results	private	Script to save metar data (short)
/var/www/html/admin/meteo_select.php	php	results	private	Script to select metar data (short)
/var/www/html/admin/meteo_select_complete.php	php	results	private	Script to select matar data (complete)
/var/www/html/admin/mysql2shp_others.py	phyton	-	system	Script to convert data form my sql to shape file
/var/www/html/admin/mysql_inf.php	php	tools	private	Script to show MySQL info
/var/www/html/admin/other_locations_add.php	php	setup	private	Script to add location for non noise parameters
/var/www/html/admin/other_locations_delete.php	php	setup	private	Script to delete location for non noise parameters
/var/www/html/admin/other_locations_edit.php	php	setup	private	Script to add location for non noise parameters
/var/www/html/admin/other_locations_setup.php	php	setup	private	Script to manage location for non noise parameters
/var/www/html/admin/pavimentazioni_GRA.geojson	geojson	-	system	File with geographical infprmation about road surfaces
/var/www/html/admin/phpinfo.php	php	tools	private	Script to show php info
/var/www/html/admin/postgres_info.php	php	tools	private	Script to show postgreSQL info
/var/www/html/admin/pyton_info.php	php	tools	private	Script to show pyton info
/var/www/html/admin/selected_locations_add.php	php	setup	private	Script to add custom location for noise calculation
/var/www/html/admin/selected_locations_delete.php	php	setup	private	Script to delete custom location for noise calculation
/var/www/html/admin/selected_locations_edit.php	php	setup	private	Script to edit custom locations for noise calculation
/var/www/html/admin/selected_locations_hist_LAeq.php	php	results	private	Script to show custom locations for LAeq calc

/var/www/html/admin/selected_locations_hist_point.php	php	results	private	Script to select custom location for noise calculation
/var/www/html/admin/selected_locations_hist_sel.php	php	results	private	Script to select day for custom location noise calculation
/var/www/html/admin/selected_locations_LAeq.php	php	setup	private	Script to show custom points computed values
/var/www/html/admin/selected_locations_setup.php	php	setup	private	Script to define custom points for which calculation is required
/var/www/html/admin/server_status.php	php	status	private	Script to show machine status
/var/www/html/admin/speed_knt2ms.php	php	tools	private	Script to convert speed from knots to m/s to
/var/www/html/admin/speed_ms2knt.php	php	tools	private	Script to convert speed from m/s to knots
/var/www/html/admin/ss_wgs_ricettori.geojson	geojson	-	system	File with geographical information about receivers
/var/www/html/admin/temp_C2F.php	php	tools	private	Script to convert temperature from Celsius to fahrenheit
/var/www/html/admin/temp_F2C.php	php	tools	private	Script to convert temperature from fahrenheit to Celsius
/var/www/html/admin/users.php	php	setup	private	Script to show users
/var/www/html/admin/users_add.php	php	setup	private	Script to add users
/var/www/html/admin/users_cp.php	php	setup	private	Script to change tha status (active/no active) of an user
/var/www/html/admin/users_delete.php	php	setup	private	Script to delete users
/var/www/html/admin/users_edit.php	php	setup	private	Script to edit users data
/home/dynamapCore_public.py	phyton	core	system	Script to calculate HI map
/home/dynamapCore.py	phyton	core	system	Script to calculate LAeq map
/home/dynamapCoreVertical.py	phyton	core	system	Script to calculate Vertical maps
/home/dynamapCoreCustom.py	phyton	core	system	Script to calculate LAeq map over specific period
/home/dynamapCoreVertical_custom.py	phyton	core	system	Script to calculate Vertical maps over specific period
/var/www/html/auto/M00.php	php	-	system	Script to dwnld and store measured noise data (crontab)
/var/www/html/auto/meteosave.php	php	-	system	Script to dnwld and store measured meteo data (crontab)
/var/www/html/barrier/audio.php	php	learn	public	Barrier simulation
/var/www/html/barrier/graphics.php	php	learn	public	Barrier simulation
/var/www/html/barrier/inc_log.php	php	learn	public	Barrier simulation
/var/www/html/barrier/index.php	php	learn	public	Barrier simulation
/var/www/html/barrier/main.php	php	learn	public	Barrier simulation
/var/www/html/barrier/mostra_sezione.php	php	learn	public	Barrier simulation
/var/www/html/barrier/style.php	php	-	public	Barrier simulation

6. SUGGESTED SOFTWARE FOR REMOTE ADMINISTRATION FROM WINDOWS

In order to administrate the system remotely, some packages are preferred. The following chapters discuss the suggested software, once again by keeping in mind the purpose of using free packages.

6.1. Putty

PuTTY is a free open-source terminal emulator that supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. In the Dynamap project, it can be used to access the Linux console through an SSH connection. Figure **1.1**: Dynamap system structureshows a typical Linux console through SSH using Windows OS.

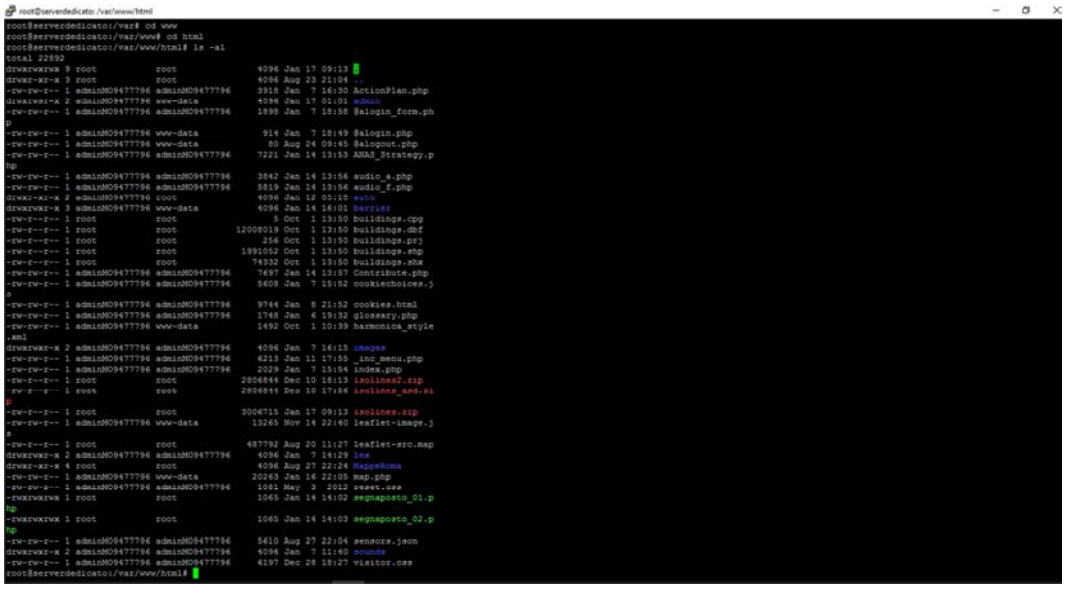

Figure 6.1: Putty screenshot

6.2. WinSCP

WinSCP is a free open-source SFTP, FTP, WebDAV, S3 and SCP client for Microsoft Windows. Its main function is to manage a secure file transfer between a local and a remote computer. Once again, it uses SSH for transferring files, and it can be used to upload files from local PC to the Dynamap machine. In combination with Notepad++ (see below), it makes an efficient way to administrate/modify scripts. Figure 6.2 shows a WinSCP session with the two windows (local window on the left and remote window on the right)

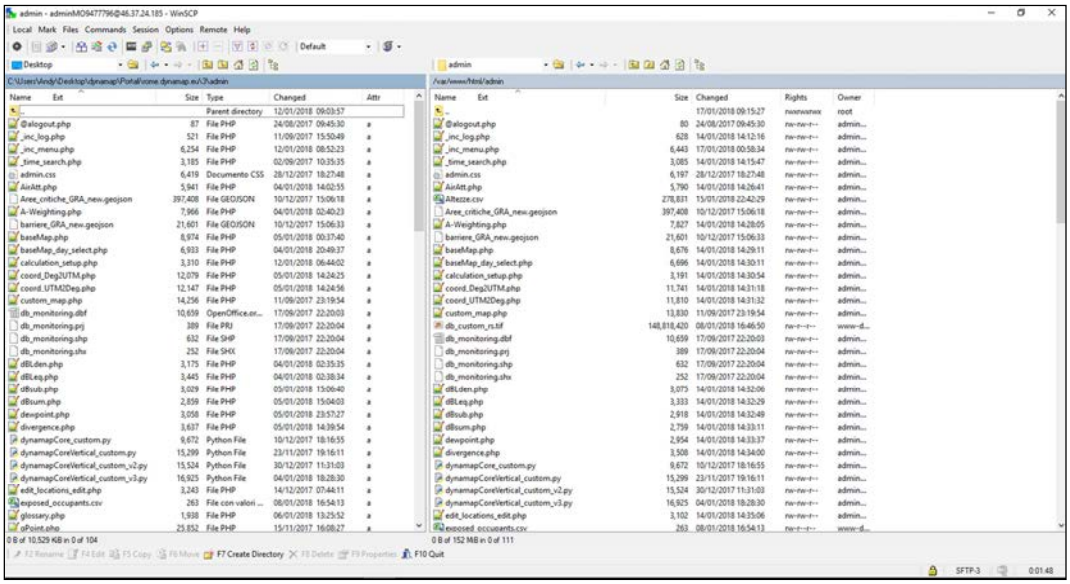

Figure 6.2: WinSCP screenshot

6.3. Notepad++

Notepad++ is a free text and source code editor for use with Microsoft Windows. It supports tabbed editing, which allows working with multiple open files in a single window. This software is a must for script editing and automatically recognises most languages, helping the developer write a clear code and identify some possible mistakes. Figure 6.3 shows the editing of a PHP page, where the tool highlights keywords, comments, etecetra.

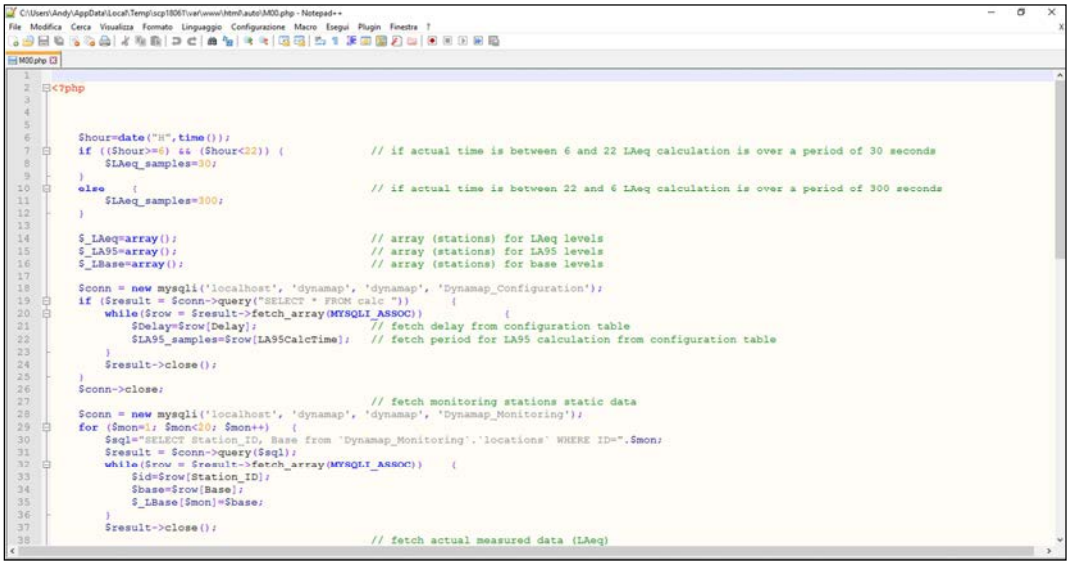

Figure 6.3: Notepad++ screenshot

6.4. HeidiSQL

HeidiSQL is a free open-source administration tool for MySQL and others databases. It can be used to interact directly with MySQL. In fact, the software can work also with PostgreeSQL, but pgAdmin is preferred for this task because it is purposely devoted to it. Figure 6.4 shows a screenshot of HeidiSQL.

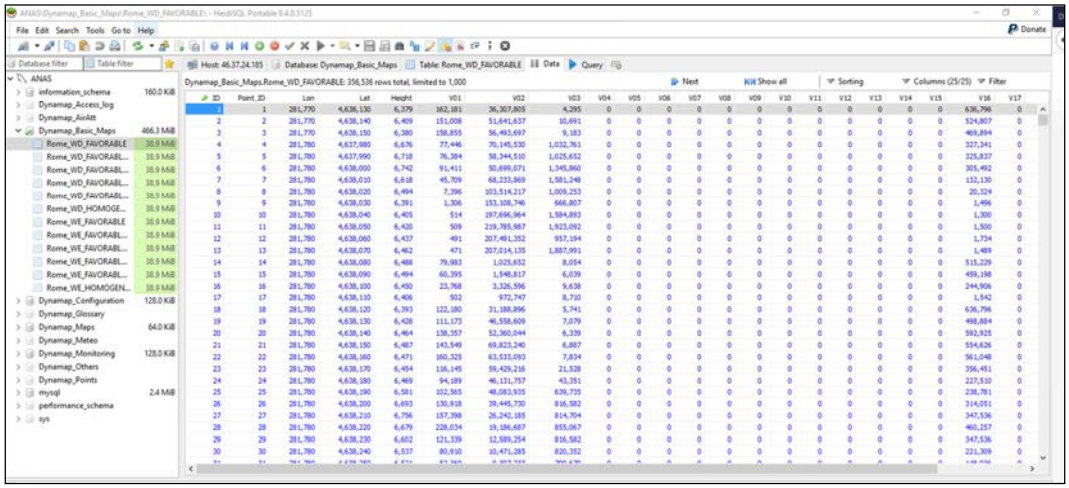

Figure 6.4: HeidySQL screenshot

6.5. pgAdmin

The pgAdmin package is a free open source graphical user interface administration tool for PostgreSQL, which is available on many computer platforms. In the project, it can be used to interact with PostgreSQL, and its powerful tools also permit to efficiently monitor the work load of the database. Figure 6.5 shows a screenshot of pgAdmin tool for the Rome system, where in the top right window (transaction per second) it is possible to recognise two peaks repeated about every 30 seconds: the highest peak is related to the vertical map update (a little bit less than 30 seconds) and the lowest one for the horizontal map update (30 seconds).

Figure 6.5: pgAdmin screenshoot

7. PACKAGES INSTALLATION

Assuming that the starting point is a LAMP machine (Linux, Apache, MySQL, PHP), following chapters will report the necessary steps to install all the required additional software. If versions of the LAMP configuration is different for the one here considered, related additional packages can have their names according the relevant versions, so the installation commands can slightly differ from the ones here reported.

7.1. GD Extension

```
apt-get install php5-gd
/etc/init.d/apache2 restart
```

7.2. JDK and JRE

```
sudo apt-get update
sudo apt-get install default-jre
sudo apt-get install default-jdk
```

7.3. PostgreeSQL and PostGIS

The commands are intended for Rome installation. For Milan please substitute 'rome' with 'milan' on lines 7 and 11

```
sudo add-apt-repository "deb http://apt.postgresql.org/pub/repos/apt/ xenial-pgdg main"
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
sudo apt update
sudo apt install postgresql-9.6 postgresql-contrib-9.6
psql --version
sudo -u postgres createuser -P admin
sudo -u postgres createdb -O admin rome
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
sudo apt update
sudo apt install postgis postgresql-9.6-postgis-2.3
sudo -u postgres psql -c "CREATE EXTENSION postgis; CREATE EXTENSION postgis_topology;" rome
```

7.4. Geoserver

Download the package form http://docs.geoserver.org/stable/en/user/installation/linux.html

```
unpack in /usr/share/geoserver-2.11.0: unzip geoserver-2.11.0 -d /usr/share
sudo chown -R USER_NAME /usr/share/geoserver-2.11.0/
export ENABLE_JSONP=true
sh /usr/share/geoserver-2.11.0/bin/startup.sh
```

8. RESOURCES ON THE WEB

8.1. Packages

- Ubuntu https://www.ubuntu.com/
- Apache https://httpd.apache.org/
- MySQL https://www.mysql.com
- PHP http://www.php.net/
- PostgreSQL https://www.postgresql.org/
- PostGIS http://www.postgis.net/
- GeoServer http://geoserver.org/

8.2. Administration tools

- Putty http://www.putty.org/
- WinSCP https://winscp.net
- Notepad++ https://notepad-plus-plus.org/download/
- HeidySQL http://www.heidisql.com/
- pgAdmin https://www.pgadmin.org/

8.3. Instruction manuals and tutorials

- Linux man http://linux.die.net/man/
- Apache docs https://httpd.apache.org/docs/
- PHP manual http://php.net/manual
- PHP Prog. http://www.infoap.utcluj.ro/multi/programming_PHP.pdf
- MySQL Man http://dev.mysql.com/doc/refman/5.7/en/
- PostgreSQL https://www.postgresql.org/docs/manuals/
- Putty Manual http://ftp.icm.edu.pl/packages/putty/putty-0.59/htmldoc/
- HeidiSQL http://www.heidisql.com/forum.php?t=522
- WinSCP https://winscp.net/eng/docs/start

Appendix A – meteosave.php

```
<?php
        set_time_limit (30);
        //  >>>> connect Metar server
        $curl = curl_init('http://www.aviationweather.gov/adds/tafs/?station_ids=LIRA&std_trans=translated&submit_both=Get+TAFs+and+METARs');
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, TRUE);
        // >>>> fetch data
        $page = curl_exec($curl);
        sleep(2);
        curl_close($curl);
        // >>>> extract data
        $page=substr($page,strpos($page,'METAR text'));
        $page=substr($page,0,strpos($page,'Forecast'));
        $metar=$page;
        $time=substr($page,strpos($page,'observed'),45);
        $time=substr($time,9);
        $time = html_entity_decode ($time);
        $time = preg_replace('/[\x00-\x1F\x7F-\xFF]/', ' ', $time);
        // >>>> temperature
        $temperature=substr($page,strpos($page,'Temperature:'),88);
        $temperature=substr($temperature,strpos($temperature,'&deg;C')-9,9);
        $temperature = html_entity_decode ($temperature);
        // >>>> wind direction
         if (strpos($page,'variable')==0)                        {
                        $direction=substr($page,strpos($page,'Winds:'),99);
                        $direction=substr($direction,strpos($direction,'(')+1,20);
                        $direction=substr($direction,0,strpos($direction,' degrees'));
        }
        if (strpos($page,'variable')!=0)          $direction='  ';
        $direction=substr($direction,0,3);
        // >>>> wind speed
        $speed=substr($page,strpos($page,'Winds:'),150);
        $speed=substr($speed,strpos($speed,' m&#47;s')-4,4);
        // >>>> air pressure
        $pressure=substr($page,strpos($page,'Pressure'),135);
        $pressure=substr($pressure,strpos($pressure,' mb')-10,10);
        $pressure = html_entity_decode ($pressure);
        $pressure=str_replace('(',",$pressure);
        // >>>> humidity
        $humidity=substr($page,strpos($page,'Dewpoint:'),121);
        $humidity=substr($humidity,strpos($humidity,'RH')+15,3);
        $humidity=html_entity_decode ($humidity);
        $humidity=str_replace('&',",$humidity);
        // >>>>>> clouds
        $CC='---';
        if (strpos($page,'CAVOK')>0) $CC='OK ';
        if (strpos($page,'NSC')>0) $CC='NSC';
        if (strpos($page,'FEW')>0) $CC='FEW';
        if (strpos($page,'SCT')>0) $CC='SCT';
        if (strpos($page,'BKN')>0) $CC='BKN';
        if (strpos($page,'OVC')>0) $CC='OVC';
```

```
// >>>> Compute map that should be used (Only fo Rome)
$we=0;
$dow=date('D');
if (($dow=='Sat')||($dow=='Sun')) $we=1;
$LocalTime=date('H:i');
// >>>> Sunrise time
$UTCSunriseTime=date_sunrise(time(), SUNFUNCS_RET_STRING, 41.9, 12.5, 90, 0);
// >>>> Sunset time
$UTCSunsetTime=date_sunset(time(), SUNFUNCS_RET_STRING, 41.9, 12.5, 90, 0);
$OC=$LocalTime;
$OA=date_sunrise(time(), SUNFUNCS_RET_STRING, 41.9, 12.5, 90, date('P'));
$OT=date_sunset(time(), SUNFUNCS_RET_STRING, 41.9, 12.5, 90, date('P'));
$VV=$speed;
$DV=$direction;
$timestamp=time();
$OC=substr($OC,0,2)*3600+substr($OC,3,2)*60;
$OA=substr($OA,0,2)*3600+substr($OA,3,2)*60;
$OT=substr($OT,0,2)*3600+substr($OT,3,2)*60;
$OA_M_30=differenza("00:30",$OA,":");
$OA_P_30=somma("00:30",$OA, ":");
$OT_M_30=differenza("00:30",$OT,":");
$OT_P_30=somma("00:30",$OT, ":");
// >>>> case 'close' to sunrise
if (($OC>=$OA_M_30)&&($OC<=$OA_P_30)&&($VV!=0)&&($DV>=45)&&($DV<135))  $map='FAVORABLE EAST';
if (($OC>=$OA_M_30)&&($OC<=$OA_P_30)&&($VV!=0)&&($DV>=135)&&($DV<225)) $map= 'FAVORABLE SOUTH';
if (($OC>=$OA_M_30)&&($OC<=$OA_P_30)&&($VV!=0)&&($DV>=225)&&($DV<315)) $map= 'FAVORABLE WEST';
if (($OC>=$OA_M_30)&&($OC<=$OA_P_30)&&($VV!=0)&&(($DV>=315)||($DV<45))) $map= 'FAVORABLE NORTH';
if (($OC>=$OA_M_30)&&($OC<=$OA_P_30)&&($VV==0)) $map= 'HOMOGENEOUS';
// >>>>        case 'close' to sunset
if (($OC>=$OT_M_30)&&($OC<=$OT_P_30)&&($VV!=0)&&($DV>=45)&&($DV<135)) $map= 'FAVORABLE EAST';
if (($OC>=$OT_M_30)&&($OC<=$OT_P_30)&&($VV!=0)&&($DV>=135)&&($DV<225)) $map='FAVORABLE SOUTH';
if (($OC>=$OT_M_30)&&($OC<=$OT_P_30)&&($VV!=0)&&($DV>=225)&&($DV<315)) $map='FAVORABLE WEST';
if (($OC>=$OT_M_30)&&($OC<=$OT_P_30)&&($VV!=0)&&(($DV>=315)||($DV<45))) $map='FAVORABLE NORTH';
if (($OC>=$OT_M_30)&&($OC<=$OT_P_30)&&($VV==0)) $map='HOMOGENEOUS';
//              >>>> day
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC=='OVC')&&($VV!=0)&&($DV>=45)&&($DV<135))  $map='FAVORABLE EAST';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC=='OVC')&&($VV!=0)&&($DV>=135)&&($DV<225)) $map='FAVORABLE SOUTH';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC=='OVC')&&($VV!=0)&&($DV>=225)&&($DV<315)) $map='FAVORABLE WEST';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC=='OVC')&&($VV!=0)&&(($DV>=315)||($DV<45))) $map='FAVORABLE NORTH';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC=='OVC')&&($VV==0)) $map='HOMOGENEOUS';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC!='OVC')&&($VV<=3))  $map='HOMOGENEOUS';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC!='OVC')&&($VV>3)&&($DV>=45)&&($DV<135))  $map='FAVORABLE EAST';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC!='OVC')&&($VV>3)&&($DV>=135)&&($DV<225))  $map='FAVORABLE SOUTH';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC!='OVC')&&($VV>3)&&($DV>=225)&&($DV<315))  $map='FAVORABLE WEST';
if (($OC>=$OA_P_30)&&($OC<=$OT_M_30)&&($CC!='OVC')&&($VV>3)&&(($DV>=315)||($DV<45)))  $map='FAVORABLE NORTH';
// >>>> Night
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV==0))  $map='FAVORABLE';
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV!=0)&&($CC!='BKN'))  $map='FAVORABLE';
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV!=0)&&($CC=='BKN')&&($DV>=45)&&($DV<135))  $map='FAVORABLE EAST';
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV!=0)&&($CC=='BKN')&&($DV>=135)&&($DV<225))  $map='FAVORABLE SOUTH';
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV!=0)&&($CC=='BKN')&&($DV>=225)&&($DV<315))  $map='FAVORABLE WEST';
if ((($OC>$OT_P_30)||($OC<$OA_M_30))&&($VV!=0)&&($CC=='BKN')&&(($DV>=315)||($DV<45))) $map='FAVORABLE NORTH';
if ($we==1) $map='WE_'.$map;
if ($we==0) $map='WD_'.$map;
$htmlmetar=$metar;
$metar=substr($metar,strpos($metar,'LIRA')-1,1024);
$metar=substr($metar,1,strpos($metar,'<')-1);
$conn = new mysqli('localhost', 'dynamap', 'dynamap', 'Dynamap_Meteo');
$sql="select * from Log WHERE upd='$time'";
```

```
        // >>>> Store data
        if ($result = $conn->query($sql))                    {
        if($result->num_rows == 0)                           {
                                $sql="INSERT INTO Log (timestamp, upd, weekend, sunrise, sunset, pressure, temperature, humidity, speed, direction, clouds,
map, metar, htmlmetar) VALUES ('$timestamp', '$time', '$we', '$UTCSunriseTime', '$UTCSunsetTime', '$pressure', '$temperature',  '$humidity', '$speed', '$direction',
'$CC', '$map', '$metar', '$htmlmetar')";
                                $conn->query($sql);
                        }
        }
        $result->close();
        $conn->close;

        // >>>> Function to compute difference between 2 hours (for periods across sunset/sunrise)

        function differenza($ora1, $ora2, $sep){
                        $part = explode($sep, $ora1);
                        $arr = explode($sep, $ora2);
                        $diff= mktime($arr[0], $arr[1]) - mktime($part[0], $part[1]);
                        $ore=floor($diff / (60*60));
                        $minuti=($diff / 60) % 60;
                        $ore = str_pad($ore,2,0,STR_PAD_LEFT);
                        $minuti = str_pad($minuti,2,0,STR_PAD_LEFT);
                        $risultato = $ore.":".$minuti;
                        return $risultato;
        }

        //  Function to compute sum of 2 hours              (for periods across sunset/sunrise)

        function somma($ora1,$ora2, $sep){
                        $ora1 = explode($sep,$ora1);
                        $ora2 = explode($sep,$ora2);
                        $ore     = $ora1[0] + $ora2[0];
                        $minuti  = $ora1[1] + $ora2[1];
                        if ($minuti  > 59){
                                $minuti  = $minuti - 60;
                                $ore +=1;
                        }
                        $ore     = str_pad($ore,2,0,STR_PAD_LEFT);
                        $minuti  = str_pad($minuti,2,0,STR_PAD_LEFT);
                        $risultato = $ore.":".$minuti;
                        return $risultato;
        }

?>
```

Appendix B – M00.php (Rome)

```
<?php

	$hour=date("H",time());
	// if actual time is between 6 and 22 LAeq calculation is over a period of 30 seconds
	if (($hour>=6) && ($hour<22)) {
		$LAeq_samples=30;
	}
	// if actual time is between 22 and 6 LAeq calculation is over a period of 300 seconds
	else	{
		$LAeq_samples=300;
	}
	$_LAeq=array();			// array (stations) for LAeq levels
	$_LA95=array();			// array (stations) for LA95 levels
	$_LBase=array();		// array (stations) for base levels
	$conn = new mysqli('localhost', 'dynamap', 'dynamap', 'Dynamap_Configuration');
	if ($result = $conn->query("SELECT * FROM calc "))		{
		while($row = $result->fetch_array(MYSQLI_ASSOC))			{
			$Delay=$row[Delay];			// fetch delay from configuration table
			$LA95_samples=$row[LA95CalcTime];		// fetch period for LA95 calculation from configuration table
		}
		$result->close();
	}
	$conn->close;
								// fetch monitoring stations static data
	$conn = new mysqli('localhost', 'dynamap', 'dynamap', 'Dynamap_Monitoring');
	for ($mon=1; $mon<20; $mon++)	{
		$sql="SELECT Station_ID, Base from `Dynamap_Monitoring`.`locations` WHERE ID=".$mon;
		$result = $conn->query($sql);
		while($row = $result->fetch_array(MYSQLI_ASSOC))		{
			$id=$row[Station_ID];
			$base=$row[Base];
			$_LBase[$mon]=$base;
		}
		$result->close();
		// fetch actual measured data (LAeq)
		$query=urlencode("select meandb(leqA) from noise where time > now() - ".($Delay+$LAeq_samples)."s and time < now() - ".$Delay."s and
id='$id'");
		$ch = curl_init();
		curl_setopt($ch, CURLOPT_TIMEOUT, 1);
		curl_setopt($ch, CURLOPT_URL,"http://roma.noisemote.com:8086/query?db=noise&epoch=s&q=$query");
		curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
		$res =  curl_exec ($ch);
		curl_close ($ch);
		$res=substr($res,strpos($res,'values')+21, strlen($res));
		$LAeq=round((substr($res,0,5))*10)/10;
		// fetch actual measured data (LA95)
		$query = urlencode("select percentile(leqA,5) from noise where time > now() - ".($Delay+$LAeq_samples)."s and time < now() - ".$Delay."s
and id='$id'");
		$ch = curl_init();
		curl_setopt($ch, CURLOPT_TIMEOUT, 1);
		curl_setopt($ch, CURLOPT_URL,"http://roma.noisemote.com:8086/query?db=noise&epoch=s&q=$query");
		curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
		$res =  curl_exec ($ch);
		curl_close ($ch);
		$res=substr($res,strpos($res,'values')+21, strlen($res));
		$LA95=round((substr($res,0,5))*10)/10;
		$_LAeq[$mon]=$LAeq;
		$_LA95[$mon]=$LA95;
	}
```

```
	// in case of missing data form one or more stations, fetch default data from a database according station, day, hour
	$day=date("l",time());
	$hour=date("H",time());
	$z=date("z",time());
	if (($z<7)||($z>357))	$period=5;
	if (($z>6)&&($z<136))	$period=1;
	if (($z>135)&&($z<213)) $period=2;
	if (($z>212)&&($z<244)) $period=3;
	if (($z>243)&&($z<358)) $period=4;
	$table='P'.$period.'_default_'.substr($day,0,3);	for ($mon=1; $mon<20; $mon++)	{
		$Status=0;
		if ($_LAeq[$mon]==0)		{
			$st=$mon;
			if (strlen($st)<2) $st='0'.$st;
			$parLAeq=$st.'_LAeq';
			$parLA95=$st.'_LA95';
			$sql="SELECT $parLAeq, $parLA95 from Dynamap_Monitoring.$table WHERE hour=$hour";
			$result = $conn->query($sql);
			while($row = $result->fetch_array(MYSQLI_ASSOC))		{
				$_LAeq[$mon]=$row[$parLAeq]/10;
				$_LA95[$mon]=$row[$parLA95]/10;
			}
			$result->close();
			$Status=1;
		}
		// harmonica index calculation and delta calculation
		$LAeq=$_LAeq[$mon];
		$LA95=$_LA95[$mon];
		$HI=round((0.25*$LAeq-0.05*$LA95-6)*10)/10;
		$DeltaLAeq=$LAeq-$_LBase[$mon];
		// save data on real time table (Monitoring station status, LAeq, LA95, delta, Harmonica)
		$sql="UPDATE `Dynamap_Monitoring`.`locations` SET Status='$Status', LAeq='$LAeq', LA95='$LA95', DeltaLAeq='$DeltaLAeq', HI='$HI'
WHERE ID=".$mon;
		$conn->query($sql);
		}
	$conn->close;
?>
```

Appendix C – M00.php (Milan)

```
<?php

	$hour=date("H",time());
	if (($hour>=7) && ($hour<21)) {		// if actual time is between 07 and 21 LAeq calculation is over a period of 300 seconds
		$LAeq_samples=300;
	}
	elseif (($hour>=1) && ($hour<7))	{	// if actual time is between 01 and 07 LAeq calculation is over a period of 3600 seconds
		$LAeq_samples=3600;
	}
	else	{
		$LAeq_samples=900;		// if actual time is between 21 and 01 LAeq calculation is over a period of 900 seconds
	}
	$_LAeq=array();			// array (stations) for LAeq levels
	$_LA95=array();			// array (stations) for LA95 levels
	$_LBase=array();		// array (stations) for base levels
	$conn = new mysqli('localhost', 'dynamap', 'dynamap', 'Dynamap_Configuration');
	if ($result = $conn->query("SELECT * FROM calc "))		{
		while($row = $result->fetch_array(MYSQLI_ASSOC))		{
			$Delay=$row[Delay];		// fetch delay from configuration table
			$LA95_samples=$row[LA95CalcTime];	// fetch period for LA95 calculation from configuration table
		}
		$result->close();
	}
	$conn->close;
							// fetch monitoring stations static data
	$conn = new mysqli('localhost', 'dynamap', 'dynamap', 'Dynamap_Monitoring');
	for ($mon=1; $mon<20; $mon++)	{
		$sql="SELECT Station_ID, Base from `Dynamap_Monitoring`.`locations` WHERE ID=".$mon;
		$result = $conn->query($sql);
		while($row = $result->fetch_array(MYSQLI_ASSOC))	{
			$id=$row[Station_ID];
			$base=$row[Base];
			$_LBase[$mon]=$base;
		}
		$result->close();
		// fetch actual measured data (LAeq)
		$query=urlencode("select meandb(leqA) from noise where time > now() - ".($Delay+$LAeq_samples)."s and time < now() - ".$Delay."s
and id='$id'");
		$ch = curl_init();
		curl_setopt($ch, CURLOPT_TIMEOUT, 1);
		curl_setopt($ch, CURLOPT_URL,"http://milano.noisemote.com:8086/query?db=noise&epoch=s&q=$query");
		curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
		$res =  curl_exec ($ch);
		curl_close ($ch);
		$res=substr($res,strpos($res,'values')+21, strlen($res));
		$LAeq=round((substr($res,0,5))*10)/10;
		// fetch actual measured data (LA95)
		$query = urlencode("select percentile(leqA,5) from noise where time > now() - ".($Delay+$LAeq_samples)."s and time < now() -
".$Delay."s and id='$id'");
		$ch = curl_init();
		curl_setopt($ch, CURLOPT_TIMEOUT, 1);
		curl_setopt($ch, CURLOPT_URL,"http://milano.noisemote.com:8086/query?db=noise&epoch=s&q=$query");
		curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
		$res =  curl_exec ($ch);
		curl_close ($ch);
		$res=substr($res,strpos($res,'values')+21, strlen($res));
		$LA95=round((substr($res,0,5))*10)/10;
		$_LAeq[$mon]=$LAeq;
		$_LA95[$mon]=$LA95;
	}
```

```
        // in case of missing data form one or more stations, fetch default data from a database according station, day, hour
        $day=date("l",time());
        $hour=date("H",time());
        $z=date("z",time());
        if (($z<7)||($z>357))       $period=5;
        if (($z>6)&&($z<136))     $period=1;
        if (($z>135)&&($z<213)) $period=2;
        if (($z>212)&&($z<244)) $period=3;
        if (($z>243)&&($z<358)) $period=4;
        $table='P'.$period.'_default_'.substr($day,0,3);
        for ($mon=1; $mon<20; $mon++)     {
                $Status=0;
                if ($_LAeq[$mon]==0)                    {
                        $st=$mon;
                        if (strlen($st)<2) $st='0'.$st;
                        $parLAeq=$st.'_LAeq';
                        $parLA95=$st.'_LA95';
                        $sql="SELECT $parLAeq, $parLA95 from Dynamap_Monitoring.$table WHERE hour=$hour";
                        $result = $conn->query($sql);
                        while($row = $result->fetch_array(MYSQLI_ASSOC))          {
                                $_LAeq[$mon]=$row[$parLAeq]/10;
                                $_LA95[$mon]=$row[$parLA95]/10;
                        }
                        $result->close();
                        $Status=1;
                }
                // harmonica index calculation and delta calculation
                $LAeq=$_LAeq[$mon];
                $LA95=$_LA95[$mon];
                $HI=round((0.25*$LAeq-0.05*$LA95-6)*10)/10;
                $DeltaLAeq=$LAeq-$_LBase[$mon];
                // save data on real time table (Monitoring station status, LAeq, LA95, delta, Harmonica)
                $sql="UPDATE `Dynamap_Monitoring`.`locations` SET Status='$Status', LAeq='$LAeq', LA95='$LA95', DeltaLAeq='$DeltaLAeq', HI='$HI'
WHERE ID=".$mon;
                $conn->query($sql);
                }
        $conn->close;
?>
```

Appendix D – dynamapCore_public.py (Rome)

```
#!/usr/bin/python

import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import MySQLdb
import time
import pdb

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"

def main():
        arg = args()
        if arg.rs:
                enable_resampling = True
                resampling_res = float(arg.rs[0])
        else:
                enable_resampling = False
                resampling_res = 0.0
        if arg.epsg_code:
                epsg_code = int(arg.epsg_code[0])
        else:
                epsg_code = 4326

        base_dir = arg.base_dir

        dynamapCore_public(base_dir,epsg_code,enable_resampling,resampling_res)

def args():
        parser = argparse.ArgumentParser(description='dynamapCore')
        parser.add_argument("--rs", default=False, nargs=1, help="Enable resampling using cubic spline. Use --rs <value>")
        parser.add_argument("--epsg_code", nargs=1, help="EPSG code for projection")
        parser.add_argument("base_dir", help="Base directory for files")
        args = parser.parse_args()
        return args


def dynamapCore_public(base_dir,epsg_code,enable_resampling,resampling_res):

        start_time = time.time()
        #base_dir = '/usr/share/geoserver-2.11.0/data_dir/data/rome/'
        if os.path.isfile(base_dir + 'harmonica.tif'):
                os.remove(base_dir + 'harmonica.tif')

        #Read input parameters from MySQL DB
        #Read basemap
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Meteo")
        cur = db.cursor()
        cur.execute('SELECT map FROM `Log` WHERE id=(SELECT MAX(id) FROM `Log`);')
        param = cur.fetchone()
        base_split = param[0].replace(" ","_")
        base_split = base_split.split('_')
        if len(base_split) == 3:
                base = base_split[0] + '_' + base_split[2]
        else:
                base = param[0]
        basemap = '/var/www/html/MappeRoma/Orizzontali/' + base + '.tif'
        db.close()
        print basemap
```

```
	#Read coefficients
	db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
	cur = db.cursor()
	delta_laeq = []
	la95 = []
	loc_list = []
	loc = {}
	oth_list = []
	oth = {}
	#cur.execute('SELECT Base,DeltaLAeq,LA95 FROM locations')
	cur.execute('SELECT ID,Base,LAeq,DeltaLAeq,LA95,HI FROM locations ORDER BY ID')
	for row in cur.fetchall():
		loc = {"ID":row[0],"Base":row[1],"LAeq":row[2],"DeltaLAeq":row[3],"LA95":row[4],"HI":row[5]}
		#print loc
		loc_list.append(loc)
		delta_laeq.append(row[3])
		la95.append(row[4]-row[1])
	db.close()
	#coeff = [np.power(10,c/10) for c in coeff]
	delta_laeq = [np.power(10,c/10) for c in delta_laeq]
	la95 = [np.power(10,c/10) for c in la95]
	print delta_laeq
	print la95

	db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Others")
	cur = db.cursor()
	cur.execute('SELECT ID,Lon,Lat,Address,Link,Type FROM locations ORDER BY ID')
	for row in cur.fetchall():
		print row[1],row[2]
		if (row[1] is not None) and (row[2] is not None):
			oth = {"ID":row[0],"Address":row[3],"Link":row[4],"Type":row[5],"Lon":row[1],"Lat":row[2]}
			oth_list.append(oth)


	db.close()

	#Update PostGIS table with new values
	connectionString = "PG:dbname='%s' host='%s' port='%s' user='%s' password='%s'" % ('rome','localhost','5432','admin','dynamap')
	driver_pg = osgeo.ogr.GetDriverByName("PostgreSQL")
	infile_locations = driver_pg.Open(connectionString,1)
	inlayer_locations = infile_locations.GetLayer('locations')
	#infeature_locations = inlayer_locations.GetNextFeature()
	nfeatures = inlayer_locations.GetFeatureCount()
	print nfeatures

	for j in range(0,nfeatures):
		print '{} of {}'.format(str(j+1),nfeatures)
		infeature = inlayer_locations.GetFeature(j+1)
		if infeature.GetField('ID') == loc_list[j]['ID']:
			sql_upd = 'UPDATE locations SET Base={},LAeq={},DeltaLAeq={},LA95={},HI={} WHERE CAST(ID as
INT)={};'.format(loc_list[j]['Base'],loc_list[j]['LAeq'],loc_list[j]['DeltaLAeq'],loc_list[j]['LA95'],loc_list[j]['HI'],loc_list[j]['ID'])
			infile_locations.ExecuteSQL(sql_upd)

	#Update PostGIS 'others' table with new values
	inlayer_others = infile_locations.GetLayer('others')
	featureDefOthers = inlayer_others.GetLayerDefn()
	nfeatures = inlayer_others.GetFeatureCount()
	print 'N features others {}'.format(nfeatures)
	infeature = inlayer_others.GetNextFeature()
	while infeature:
		inlayer_others.DeleteFeature(infeature.GetFID())
		infeature = inlayer_others.GetNextFeature()
```

```
	for i in range(0,len(oth_list)):
		#pdb.set_trace()
		#feature = inlayer_others.GetFeature(i+1)
		feature = osgeo.ogr.Feature(featureDefOthers)
		point = osgeo.ogr.Geometry(osgeo.ogr.wkbPoint)
		point.AddPoint(float(oth_list[i]['Lon']),float(oth_list[i]['Lat']))
		feature.SetGeometry(point)
		point.Destroy()
		'''
		if (feature_tb.GetField('ID') != oth_list[i]['ID']):
			point = osgeo.ogr.Geometry(osgeo.ogr.wkbPoint)
			point.AddPoint(float(oth_list[i]['Lon']),float(oth_list[i]['Lat']))
			feature.SetGeometry(point)
			point.Destroy()
		'''
		feature.SetField('ID',int(oth_list[i]['ID']))
		feature.SetField('Address',oth_list[i]['Address'])
		feature.SetField('Link',oth_list[i]['Link'])
		feature.SetField('Type',oth_list[i]['Type'])

		inlayer_others.CreateFeature(feature)
		#inlayer_others.SetFeature(feature)

		feature.Destroy()

	#ds.Destroy()
	#LAeq: convert in linear
	#Scale
	#Harmonica= 0.25LAeq - 0.05LA95 - 6

	#Compute new layer
	new_val_delta = []
	new_val_la95 = []
	band_list = read_image(basemap,np.float32,0)
	#pdb.set_trace()
	rows,cols,nbands,geo_transform,projection = read_image_parameters(basemap)
	for b in range(0,19):
		print 'Band: ' + str(b)
		new_val_delta.append(band_list[b]*delta_laeq[b])
		new_val_la95.append(band_list[b]*la95[b])
	#new_mat = np.round(100*np.log10(np.sum(np.array(new_val_list),axis=0)))
	new_val_delta_masked = np.ma.masked_equal(np.sum(np.array(new_val_delta),axis=0),0.0)
	new_val_la95_masked = np.ma.masked_equal(np.sum(np.array(new_val_la95),axis=0),0.0)

	#new_mat_delta = np.round(10*np.log10(np.sum(np.array(new_val_delta),axis=0)))
	new_mat_delta = np.round(10*np.log10(new_val_delta_masked))
	#new_mat_la95 = np.round(10*np.log10(np.sum(np.array(new_val_la95),axis=0)))
	new_mat_la95 = np.round(10*np.log10(new_val_la95_masked))
	hi = 0.25*new_mat_delta - 0.05*new_mat_la95 - 6
	hi[np.less(hi,0)] = 0.0
	hi[np.greater(hi,10)] = 10.0
	hi[new_mat_delta.mask] = 99  #custom nodata value

	write_image([hi],np.float32,0,base_dir+'harmonica.tif',rows,cols,geo_transform,projection)

	if os.path.isfile(base_dir+'harmonica_rs.tif'):
		os.remove(base_dir+'harmonica_rs.tif')

	#Resample raster
	if enable_resampling == True:
		command = 'gdalwarp -tr {} {} -srcnodata 99 -r cubicspline {} {}'.format(resampling_res,resampling_res,base_dir + 'harmonica.tif',base_dir +
'harmonica_rs.tif')
		print command
		proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
		for line in iter(proc.readline, ''): print line
		os.remove(base_dir+'harmonica.tif')
```

```
		#Isolines generation
		#command = 'gdal_contour -i 1.0 -f "ESRI Shapefile" {} {}'.format(base_dir+'harmonica_rs.tif',base_dir+'isolines.shp')
		#print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout

		#Zip output
		#command = 'zip -r {} {}'.format('/var/www/html/isolines.zip',base_dir+'isolines.*')
		#print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout

		end_time = time.time()
		print 'Total time: ' + str(end_time-start_time)

def read_image(input_raster,data_type,band_selection):

		'''Read raster using GDAL

		:param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
		:param data_type: numpy type used to read the image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
		:param band_selection: number associated with the band to extract (0: all bands, 1: blue, 2: green, 3:red, 4:infrared) (integer).
		:returns:  a list containing the desired bands as ndarrays (list of arrays).
		:raises: AttributeError, KeyError

		'''

		band_list = []

		if data_type == 0: #most of the images (MR and HR) can be read as uint16
			data_type = np.uint16

		inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
		cols=inputimg.RasterXSize
		rows=inputimg.RasterYSize
		nbands=inputimg.RasterCount

		if band_selection == 0:
			#read all the bands
			for i in range(1,nbands+1):
				inband = inputimg.GetRasterBand(i)
				inband.SetNoDataValue(0)
				#mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
				mat_data = inband.ReadAsArray().astype(data_type)
				mat_data = np.ma.masked_equal(mat_data, 0.0)
				band_list.append(mat_data)
		else:
			#read the single band
			inband = inputimg.GetRasterBand(band_selection)
			mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
			print inband.GetNoDataValue()
			band_list.append(mat_data)

		inputimg = None
		return band_list
```

```
def read_image_parameters(input_raster):

        '''Read raster parameters using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :returns:  a list containing rows, columns, number of bands, geo-transformation matrix and projection.
        :raises: AttributeError, KeyError

        '''

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount
        geo_transform = inputimg.GetGeoTransform()
        projection = inputimg.GetProjection()

        inputimg = None
        return rows,cols,nbands,geo_transform,projection

def write_image(band_list,data_type,band_selection,output_raster,rows,cols,geo_transform,projection):

        '''Write array to file as raster using GDAL

        :param band_list: list of arrays containing the different bands to write (list of arrays).
        :param data_type: numpy data type of the output image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type)
        :param band_selection: number associated with the band to write (0: all, 1: blue, 2: green, 3: red, 4: infrared) (integer)
        :param output_raster: path and name of the output raster to create (*.TIF, *.tiff) (string)
        :param rows: rows of the output raster (integer)
        :param cols: columns of the output raster (integer)
        :param geo_transform: geo-transformation matrix containing coordinates and resolution of the output (array of 6 elements, float)
        :param projection: projection of the output image (string)
        :returns: An output file is created
        :raises: AttributeError, KeyError

        '''

        if data_type == 0:
                gdal_data_type = GDT_UInt16 #default data type
        else:
                gdal_data_type = data_type2gdal_data_type(data_type)

        driver = osgeo.gdal.GetDriverByName('GTiff')

        if band_selection == 0:
                nbands = len(band_list)
        else:
                nbands = 1
        outDs = driver.Create(output_raster, cols, rows,nbands, gdal_data_type)
        if outDs is None:
                print 'Could not create output file'
                sys.exit(1)

        if band_selection == 0:
                #write all the bands to file
                for i in range(0,nbands):
                        outBand = outDs.GetRasterBand(i+1)
                        outBand.SetNoDataValue(99)
                        outBand.WriteArray(band_list[i], 0, 0)
        else:
                #write the specified band to file
                outBand = outDs.GetRasterBand(1)
                outBand.WriteArray(band_list[band_selection-1], 0, 0)
        #assign geomatrix and projection
        outDs.SetGeoTransform(geo_transform)
        outDs.SetProjection(projection)
        outDs = None
```

```
def data_type2gdal_data_type(data_type):

        '''Conversion from numpy data type to GDAL data type

        :param data_type: numpy type (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :returns: corresponding GDAL data type
        :raises: AttributeError, KeyError

        '''
        #Function needed when it is necessary to write an output file
        if data_type == np.uint16:
                return GDT_UInt16
        if data_type == np.uint8:
                return GDT_Byte
        if data_type == np.int32:
                return GDT_Int32
        if data_type == np.float32:
                return GDT_Float32
        if data_type == np.float64:
                return GDT_Float64

if __name__ == '__main__':
        main()
```

Appendix E – dynamapCore.py (Rome)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import MySQLdb
import zipfile

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"

def main():
        arg = args()
        if arg.rs:
                enable_resampling = True
                resampling_res = float(arg.rs[0])
        else:
                enable_resampling = False
                resampling_res = 0.0
        if arg.epsg_code:
                epsg_code = int(arg.epsg_code[0])
        else:
                epsg_code = 4326

        base_dir = arg.base_dir

        dynamapCore(base_dir,epsg_code,enable_resampling,resampling_res)


def args():
        parser = argparse.ArgumentParser(description='dynamapCore')
        parser.add_argument("--rs", default=False, nargs=1, help="Enable resampling using cubic spline. Use --rs <value>")
        parser.add_argument("--epsg_code", nargs=1, help="EPSG code for projection")
        parser.add_argument("base_dir", help="Base directory for files")
        args = parser.parse_args()
        return args


def dynamapCore(base_dir,epsg_code,enable_resampling,resampling_res):

        #base_dir = '/usr/share/geoserver-2.11.0/data_dir/data/rome/'
        if os.path.isfile(base_dir + 'db.tif'):
                os.remove(base_dir + 'db.tif')

        #Read input parameters from MySQL DB
        #Read basemap
        db =  MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Meteo")
        cur = db.cursor()
        cur.execute('SELECT map FROM `Log` WHERE id=(SELECT MAX(id) FROM `Log`);')
        param = cur.fetchone()
        base_split = param[0].replace(" ","_")
        base_split = base_split.split('_')
        if len(base_split) == 3:
                base = base_split[0] + '_' + base_split[2]
        else:
                base = param[0]
        basemap = '/var/www/html/MappeRoma/Orizzontali/' + base + '.tif'
        db.close()
        print basemap
```

```
		#Read coefficients
		db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
		cur = db.cursor()
		laeq = []

		cur.execute('SELECT DeltaLAeq FROM locations')
		for row in cur.fetchall():
			laeq.append(row[0])

		print laeq
		coeff = [np.power(10,c/10) for c in laeq]
		print coeff
		#LAeq: convert to power
		#Scale
		#Compute new layer
		new_val_list = []
		band_list = read_image(basemap,np.float32,0)
		rows,cols,nbands,geo_transform,projection = read_image_parameters(basemap)
		for b in range(0,19):
			print 'Band: ' + str(b)
			new_val_list.append(band_list[b]*coeff[b])
		new_mat = np.round(100*np.log10(np.sum(np.array(new_val_list),axis=0)))
		write_image([new_mat],np.uint16,0,base_dir+'db.tif',rows,cols,geo_transform,projection)

		if os.path.isfile(base_dir+'db_rs.tif'):
			os.remove(base_dir+'db_rs.tif')

		#Resample raster
		if enable_resampling == True:
			command = 'gdalwarp -tr {} {} -srcnodata 0.0 -r cubicspline {} {}'.format(resampling_res,resampling_res,base_dir + 'db.tif',base_dir + 'db_rs.tif')
			print command
			proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
			for line in iter(proc.readline, ''): print line
			os.remove(base_dir+'db.tif')

		#Isolines generation
		if os.path.isfile(base_dir+'isolines.shp'):
			os.remove(base_dir+'isolines.shp')
		if os.path.isfile(base_dir+'isolines.shx'):
			os.remove(base_dir+'isolines.shx')
		if os.path.isfile(base_dir+'isolines.dbf'):
			os.remove(base_dir+'isolines.dbf')
		if os.path.isfile(base_dir+'isolines.prj'):
			os.remove(base_dir+'isolines.prj')

		command = 'gdal_contour -i 50.0 -f "ESRI Shapefile" {} {}'.format(base_dir+'db_rs.tif',base_dir+'isolines.shp')
		print command
		#proc = subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
		os.system(command)
		#Zip output
		#command = 'zip -r {} {}'.format('/var/www/html/isolines.zip',base_dir+'isolines.*')
		#print command
		#proc = subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
		if os.path.isfile('/var/www/html/isolines.zip'):
			os.remove('/var/www/html/isolines.zip')
		zf = zipfile.ZipFile('/var/www/html/isolines.zip', mode='w')
		zf.write(base_dir+'isolines.shp','isolines.shp')
		zf.write(base_dir+'isolines.shx','isolines.shx')
		zf.write(base_dir+'isolines.dbf','isolines.dbf')
		zf.write(base_dir+'isolines.prj','isolines.prj')
		zf.close()
```

```
def read_image(input_raster,data_type,band_selection):

        '''Read raster using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :param data_type: numpy type used to read the image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :param band_selection: number associated with the band to extract (0: all bands, 1: blue, 2: green, 3:red, 4:infrared) (integer).
        :returns:  a list containing the desired bands as ndarrays (list of arrays).
        :raises: AttributeError, KeyError

        '''

        band_list = []

        if data_type == 0: #most of the images (MR and HR) can be read as uint16
                data_type = np.uint16

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount

        if band_selection == 0:
                #read all the bands
                for i in range(1,nbands+1):
                        inband = inputimg.GetRasterBand(i)
                        #mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
                        mat_data = inband.ReadAsArray().astype(data_type)
                        band_list.append(mat_data)
        else:
                #read the single band
                inband = inputimg.GetRasterBand(band_selection)
                mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
                band_list.append(mat_data)

        inputimg = None
        return band_list


def read_image_parameters(input_raster):

        '''Read raster parameters using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :returns:  a list containing rows, columns, number of bands, geo-transformation matrix and projection.
        :raises: AttributeError, KeyError

        '''

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount
        geo_transform = inputimg.GetGeoTransform()
        projection = inputimg.GetProjection()

        inputimg = None
        return rows,cols,nbands,geo_transform,projection
```

```
def write_image(band_list,data_type,band_selection,output_raster,rows,cols,geo_transform,projection):

        '''Write array to file as raster using GDAL

        :param band_list: list of arrays containing the different bands to write (list of arrays).
        :param data_type: numpy data type of the output image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type)
        :param band_selection: number associated with the band to write (0: all, 1: blue, 2: green, 3: red, 4: infrared) (integer)
        :param output_raster: path and name of the output raster to create (*.TIF, *.tiff) (string)
        :param rows: rows of the output raster (integer)
        :param cols: columns of the output raster (integer)
        :param geo_transform: geo-transformation matrix containing coordinates and resolution of the output (array of 6 elements, float)
        :param projection: projection of the output image (string)
        :returns: An output file is created
        :raises: AttributeError, KeyError

        '''

        if data_type == 0:
                gdal_data_type = GDT_UInt16 #default data type
        else:
                gdal_data_type = data_type2gdal_data_type(data_type)

        driver = osgeo.gdal.GetDriverByName('GTiff')

        if band_selection == 0:
                nbands = len(band_list)
        else:
                nbands = 1
        outDs = driver.Create(output_raster, cols, rows,nbands, gdal_data_type)
        if outDs is None:
                print 'Could not create output file'
                sys.exit(1)

        if band_selection == 0:
                #write all the bands to file
                for i in range(0,nbands):
                        outBand = outDs.GetRasterBand(i+1)
                        outBand.SetNoDataValue(0.0)
                        outBand.WriteArray(band_list[i], 0, 0)
        else:
                #write the specified band to file
                outBand = outDs.GetRasterBand(1)
                outBand.WriteArray(band_list[band_selection-1], 0, 0)
        #assign geomatrix and projection
        outDs.SetGeoTransform(geo_transform)
        outDs.SetProjection(projection)
        outDs = None

def data_type2gdal_data_type(data_type):

        '''Conversion from numpy data type to GDAL data type

        :param data_type: numpy type (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :returns: corresponding GDAL data type
        :raises: AttributeError, KeyError

        '''
        #Function needed when it is necessary to write an output file
        if data_type == np.uint16:
                return GDT_UInt16
        if data_type == np.uint8:
                return GDT_Byte
        if data_type == np.int32:
                return GDT_Int32
        if data_type == np.float32:
                return GDT_Float32
        if data_type == np.float64:
                return GDT_Float64

if __name__ == '__main__':
        main()
```

Appendix F – dynamapCoreVertical.py (Rome)

```
#!/usr/bin/python

import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import time
import datetime
import MySQLdb
import csv
import psycopg2
import pdb

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"


def main():
        arg = args()
        dynamapCoreVertical()


def args():
        parser = argparse.ArgumentParser(description='dynamapCoreVertical')
        args = parser.parse_args()
        return args


def dynamapCoreVertical():

        start_time = time.time()

        #Get current hour
        now = datetime.datetime.now()
        if now.hour >= 6 and now.hour <= 22:
                interval = 'day'
        else:
                interval = 'night'

        #Read basemap
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Meteo")
        cur = db.cursor()
        cur.execute('SELECT map FROM `Log` WHERE id=(SELECT MAX(id) FROM `Log`);')
        param = cur.fetchone()
        base_split = param[0].replace(" ","_")
        base_split = base_split.split('_')
        if len(base_split) == 3:
                base = base_split[0] + '_' + base_split[2]
        else:
                base = param[0]

        basemap = base.lower() + '_dl'
        if base.lower() != 'we_favorable':
                basemap_ue = base.lower() + '_ue'
        else:
                basemap_ue = 'we_homogeneous_ue'
        db.close()
        print basemap
```

```
#Read coefficients
db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
cur = db.cursor()
laeq = []

cur.execute('SELECT DeltaLAeq FROM locations')
for row in cur.fetchall():
        laeq.append(row[0])

print laeq
coeff = [np.power(10,c/10) for c in laeq]
print coeff

conn = psycopg2.connect("dbname=rome host=localhost user=admin password=dynamap")

# Update buildings in DLGS and UE tables
#Initialize counters for exposed people
count_dlgs = {"40":0,"45":0,"50":0,"55":0,"60":0,"65":0,"70":0,"75":0,"80":0,"85":0}
count_ue = {"40":0,"45":0,"50":0,"55":0,"60":0,"65":0,"70":0,"75":0,"80":0,"85":0}

#Generate attribute names for specific basemap
query_basemap = ['{}.v{}'.format(basemap,str(i).zfill(2)) for i in range(1,20)]
query_basemap_ue = ['{}.v{}'.format(basemap_ue,str(i).zfill(2)) for i in range(1,20)]

#Update only buildings available in DLGS and UE basemaps
sql = 'SELECT buildings.geodbid,buildings.lim_g,buildings.lim_n,buildings.n_occupan,{},{} FROM buildings INNER JOIN {} ON CAST(buildings.geodbid as
INT) = {}."id edif" INNER JOIN {} ON CAST (buildings.geodbid as INT) = {}."id edif" ORDER BY
buildings.geodbid'.format(','.join(query_basemap),','.join(query_basemap_ue),basemap,basemap,basemap_ue,basemap_ue)
count_dlgs,count_ue = update_db(conn,sql,count_dlgs,count_ue,1,coeff,interval)

#Update buildings missing from UE but available on DLGS basemap
sql = 'SELECT buildings.geodbid,buildings.lim_g,buildings.lim_n,buildings.n_occupan,{} FROM buildings INNER JOIN {} ON CAST(buildings.geodbid as
INT) = {}."id edif" AND buildings.geodbid NOT IN (SELECT buildings.geodbid FROM buildings INNER JOIN {} ON CAST(buildings.geodbid as INT) = {}."id edif" INNER
JOIN {} ON CAST (buildings.geodbid as INT) = {}."id edif") ORDER BY
1'.format(','.join(query_basemap),basemap,basemap,basemap,basemap,basemap_ue,basemap_ue)
count_dlgs,count_ue = update_db(conn,sql,count_dlgs,count_ue,2,coeff,interval)

#Update buildings missing from DLGS but available on UE basemap
sql = 'SELECT buildings.geodbid,buildings.lim_g,buildings.lim_n,buildings.n_occupan,{} FROM buildings INNER JOIN {} ON CAST(buildings.geodbid as
INT) = {}."id edif" AND buildings.geodbid NOT IN (SELECT buildings.geodbid FROM buildings INNER JOIN {} ON CAST(buildings.geodbid as INT) = {}."id edif" INNER
JOIN {} ON CAST (buildings.geodbid as INT) = {}."id edif") ORDER BY
1'.format(','.join(query_basemap_ue),basemap_ue,basemap_ue,basemap,basemap,basemap_ue,basemap_ue)
count_dlgs,count_ue = update_db(conn,sql,count_dlgs,count_ue,3,coeff,interval)

#Delete CSV file if existing
if os.path.isfile('/var/www/html/admin/exposed_occupants_now.csv'):
        os.remove('/var/www/html/admin/exposed_occupants_now.csv')

#Write counters to CSV file
with open('/var/www/html/admin/exposed_occupants_now.csv', 'wb') as csvfile:
        field_names = ['Interval','Occupants_DLGS','Occupants_UE']
        writer = csv.DictWriter(csvfile, fieldnames=field_names)
        writer.writeheader()
        writer.writerow({'Interval':'40-44.9','Occupants_DLGS':count_dlgs["40"],'Occupants_UE':count_ue["40"]})
        writer.writerow({'Interval':'45-49.9','Occupants_DLGS':count_dlgs["45"],'Occupants_UE':count_ue["45"]})
        writer.writerow({'Interval':'50-54.9','Occupants_DLGS':count_dlgs["50"],'Occupants_UE':count_ue["50"]})
        writer.writerow({'Interval':'55-59.9','Occupants_DLGS':count_dlgs["55"],'Occupants_UE':count_ue["55"]})
        writer.writerow({'Interval':'60-64.9','Occupants_DLGS':count_dlgs["60"],'Occupants_UE':count_ue["60"]})
        writer.writerow({'Interval':'65-69.9','Occupants_DLGS':count_dlgs["65"],'Occupants_UE':count_ue["65"]})
        writer.writerow({'Interval':'70-74.9','Occupants_DLGS':count_dlgs["70"],'Occupants_UE':count_ue["70"]})
        writer.writerow({'Interval':'75-79.9','Occupants_DLGS':count_dlgs["75"],'Occupants_UE':count_ue["75"]})
        writer.writerow({'Interval':'80-84.9','Occupants_DLGS':count_dlgs["80"],'Occupants_UE':count_ue["80"]})
        writer.writerow({'Interval':'85-89.9','Occupants_DLGS':count_dlgs["85"],'Occupants_UE':count_ue["85"]})

end_time = time.time()
print 'Total time: ' + str(end_time-start_time)
```

```
def update_db(conn,sql,count_dlgs,count_ue,opt,coeff,interval):

	cur = conn.cursor()
	cur.execute(sql)
	fcount = cur.rowcount
	print 'Total: {}'.format(fcount)

	c = 1
	#Loop on features to update
	for row in cur.fetchall():
		id_build = int(row[0])
		#sys.stdout.write('%s of %s \r' % (str(c),fcount))
		#sys.stdout.flush()

		if opt == 1: #Update DLGS and UE
			param = np.array(row[4:23]).astype('float')
			param_ue = np.array(row[23:42]).astype('float')

			comb = np.sum(param*coeff)
			comb_ue = np.sum(param_ue*coeff)
			if comb !=0:
				comb = np.round([10*np.log10(comb)],1)
				comb = comb[0]
			else:
				comb = 0
			if comb_ue !=0:
				comb_ue = np.round([10*np.log10(comb_ue)],1)
				comb_ue = comb_ue[0]
			else:
				comb_ue = 0

			if interval == 'day':
				diff = comb - float(row[1])
			elif interval == 'night':
				diff = comb - float(row[2])
			if diff < 0:
				diff = 0
			sql_upd = 'UPDATE "buildings" SET "dlgs_now"={}, "ue_now"={}, "conflitto_now"={} WHERE CAST(geodbid as
INT)={};'.format(float(comb),float(comb_ue),float(diff),int(id_build))
			cur.execute(sql_upd)
			conn.commit()

		if opt == 2: #Update only DLGS
			param = np.array(row[4:23]).astype('float')
			comb = np.sum(param*coeff)
			if comb !=0:
				comb = np.round([10*np.log10(comb)],1)
				comb = comb[0]
			else:
				comb = 0
			if interval == 'day':
				diff = comb - float(row[1])
			elif interval == 'night':
				diff = comb - float(row[2])
			if diff < 0:
				diff = 0
			sql_upd = 'UPDATE "buildings" SET "dlgs_now"={}, "conflitto_now"={} WHERE CAST(geodbid as
INT)={};'.format(float(comb),float(diff),int(id_build))
			cur.execute(sql_upd)
			conn.commit()
```

```
                if opt == 3: #Update only UE
                        param_ue = np.array(row[4:23]).astype('float')
                        comb_ue = np.sum(param_ue*coeff)
                        if comb_ue !=0:
                                comb_ue = np.round([10*np.log10(comb_ue)],1)
                                comb_ue = comb_ue[0]
                        else:
                                comb_ue = 0
                        sql_upd = 'UPDATE "buildings" SET "ue_now"={} WHERE CAST(geodbid as INT)={};'.format(float(comb_ue),int(id_build))
                        cur.execute(sql_upd)
                        conn.commit()

                if opt == 1 or opt == 2: #Update counters for DLGS
                        if comb >= 40 and comb < 45:
                                count_dlgs["40"] = count_dlgs["40"] + float(row[3])
                        elif comb >= 45 and comb < 50:
                                count_dlgs["45"] = count_dlgs["45"] + float(row[3])
                        elif comb >= 50 and comb < 55:
                                count_dlgs["50"] = count_dlgs["50"] + float(row[3])
                        elif comb >= 55 and comb < 60:
                                count_dlgs["55"] = count_dlgs["55"] + float(row[3])
                        elif comb >= 60 and comb < 65:
                                count_dlgs["60"] = count_dlgs["60"] + float(row[3])
                        elif comb >= 65 and comb < 70:
                                count_dlgs["65"] = count_dlgs["65"] + float(row[3])
                        elif comb >= 70 and comb < 75:
                                count_dlgs["70"] = count_dlgs["70"] + float(row[3])
                        elif comb >= 75 and comb < 80:
                                count_dlgs["75"] = count_dlgs["75"] + float(row[3])
                        elif comb >= 80 and comb < 85:
                                count_dlgs["80"] = count_dlgs["80"] + float(row[3])
                        elif comb >= 85 and comb < 90:
                                count_dlgs["85"] = count_dlgs["85"] + float(row[3])
                if opt == 1 or opt == 3: #Update counter for UE
                        if comb_ue >= 40 and comb_ue < 45:
                                count_ue["40"] = count_ue["40"] + float(row[3])
                        elif comb_ue >= 45 and comb_ue < 50:
                                count_ue["45"] = count_ue["45"] + float(row[3])
                        elif comb_ue >= 50 and comb_ue < 55:
                                count_ue["50"] = count_ue["50"] + float(row[3])
                        elif comb_ue >= 55 and comb_ue < 60:
                                count_ue["55"] = count_ue["55"] + float(row[3])
                        elif comb_ue >= 60 and comb_ue < 65:
                                count_ue["60"] = count_ue["60"] + float(row[3])
                        elif comb_ue >= 65 and comb_ue < 70:
                                count_ue["65"] = count_ue["65"] + float(row[3])
                        elif comb_ue >= 70 and comb_ue < 75:
                                count_ue["70"] = count_ue["70"] + float(row[3])
                        elif comb_ue >= 75 and comb_ue < 80:
                                count_ue["75"] = count_ue["75"] + float(row[3])
                        elif comb_ue >= 80 and comb_ue < 85:
                                count_ue["80"] = count_ue["80"] + float(row[3])
                        elif comb_ue >= 85 and comb_ue < 90:
                                count_ue["85"] = count_ue["85"] + float(row[3])
                c+=1

        return count_dlgs,count_ue

if __name__ == '__main__':
        main()
```

Appendix G – dynamapCore_public.py (Milan)

```
#!/usr/bin/python

import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import MySQLdb
import time
import pdb

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"


def main():
        arg = args()
        if arg.rs:
                enable_resampling = True
                resampling_res = float(arg.rs[0])
        else:
                enable_resampling = False
                resampling_res = 0.0
        if arg.epsg_code:
                epsg_code = int(arg.epsg_code[0])
        else:
                epsg_code = 4326

        base_dir = arg.base_dir
        #input_txt = arg.input_txt
        #db_name = arg.db_name
        #table_name = arg.table_name
        #relevant_column = arg.relevant_column

        dynamapCore_public(base_dir,epsg_code,enable_resampling,resampling_res)


def args():
        parser = argparse.ArgumentParser(description='dynamapCore')
        parser.add_argument("--rs", default=False, nargs=1, help="Enable resampling using cubic spline. Use --rs <value>")
        parser.add_argument("--epsg_code", nargs=1, help="EPSG code for projection")
        parser.add_argument("base_dir", help="Base directory for files")
        args = parser.parse_args()
        return args
```

```
def dynamapCore_public(base_dir,epsg_code,enable_resampling,resampling_res):

        start_time = time.time()
        #base_dir = '/usr/share/geoserver-2.11.0/data_dir/data/rome/'
        if os.path.isfile(base_dir + 'harmonica.tif'):
                os.remove(base_dir + 'harmonica.tif')

        #Read input parameters from MySQL DB
        #Read basemap
        '''
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Meteo")
        cur = db.cursor()
        cur.execute('SELECT map FROM `Log` WHERE id=(SELECT MAX(id) FROM `Log`);')
        param = cur.fetchone()
        #Fix WD_FAVORABLE_EAST to WD_EAST
        base_split = param[0].replace(" ","_")
        base_split = base_split.split('_')
        if len(base_split) == 3:
                base = base_split[0] + '_' + base_split[2]
        else:
                base = param[0]
        '''
        base = 'milan'
        basemap = '/var/www/html/MappeMilano/' + base + '.tif'
        #db.close()
        print basemap

        #Read coefficients
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
        cur = db.cursor()

        delta_laeq = []
        la95 = []
        loc_list = []
        loc = {}
        #cur.execute('SELECT Base,DeltaLAeq,LA95 FROM locations')
        cur.execute('SELECT ID,Base,LAeq,DeltaLAeq,LA95,HI FROM locations WHERE ID>100 ORDER BY ID')
        for row in cur.fetchall():
                print row
                if row[1] is not None:
                        loc = {"ID":row[0],"Base":row[1],"LAeq":row[2],"DeltaLAeq":row[3],"LA95":row[4],"HI":row[5]}
                        print loc
                        loc_list.append(loc)
                        delta_laeq.append(row[3])
                        la95.append(row[4]-row[1])
        db.close()
        #coeff = [np.power(10,c/10) for c in coeff]
        delta_laeq = [np.power(10,c/10) for c in delta_laeq]
        la95 = [np.power(10,c/10) for c in la95]
        print delta_laeq
        print la95
```

```
		#LAeq convert to linear
		#scale
		#Harmonica Index= 0.25LAeq - 0.05LA95 – 6

		#Compute new layer
		new_val_delta = []
		new_val_la95 = []
		band_list = read_image(basemap,np.float32,0)
		#pdb.set_trace()
		rows,cols,nbands,geo_transform,projection = read_image_parameters(basemap)
		for b in range(0,6):
			print 'Band: ' + str(b)
			new_val_delta.append(band_list[b]*delta_laeq[b])
			new_val_la95.append(band_list[b]*la95[b])
		new_val_delta_masked = np.ma.masked_equal(np.sum(np.array(new_val_delta),axis=0),0.0)
		new_val_la95_masked = np.ma.masked_equal(np.sum(np.array(new_val_la95),axis=0),0.0)
		new_mat_delta = np.round(10*np.log10(new_val_delta_masked))
		new_mat_la95 = np.round(10*np.log10(new_val_la95_masked))
		hi = 0.25*new_mat_delta - 0.05*new_mat_la95 - 6
		hi[np.less(hi,0)] = 0.0
		hi[np.greater(hi,10)] = 10.0
		hi[new_mat_delta.mask] = 99  #custom nodata value

		write_image([hi],np.float32,0,base_dir+'harmonica.tif',rows,cols,geo_transform,projection)

		if os.path.isfile(base_dir+'harmonica_rs.tif'):
			os.remove(base_dir+'harmonica_rs.tif')

		#Resample raster
		if enable_resampling == True:
			command = 'gdalwarp -tr {} {} -srcnodata 99 -r cubicspline {} {}'.format(resampling_res,resampling_res,base_dir + 'harmonica.tif',base_dir +
'harmonica_rs.tif')
			print command
			proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
			for line in iter(proc.readline, ''): print line
			os.remove(base_dir+'harmonica.tif')

		#Isolines generation
		#command = 'gdal_contour -i 1.0 -f "ESRI Shapefile" {} {}'.format(base_dir+'harmonica_rs.tif',base_dir+'isolines.shp')
		#print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout

		#Zip output
		#command = 'zip -r {} {}'.format('/var/www/html/isolines.zip',base_dir+'isolines.*')
		#print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout

		end_time = time.time()
		print 'Total time: ' + str(end_time-start_time)
```

```
def read_image(input_raster,data_type,band_selection):

        '''Read raster using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :param data_type: numpy type used to read the image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :param band_selection: number associated with the band to extract (0: all bands, 1: blue, 2: green, 3:red, 4:infrared) (integer).
        :returns:  a list containing the desired bands as ndarrays (list of arrays).
        :raises: AttributeError, KeyError

        '''

        band_list = []

        if data_type == 0: #most of the images (MR and HR) can be read as uint16
                data_type = np.uint16

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount

        if band_selection == 0:
                #read all the bands
                for i in range(1,nbands+1):
                        inband = inputimg.GetRasterBand(i)
                        inband.SetNoDataValue(0)
                        mat_data = inband.ReadAsArray().astype(data_type)
                        mat_data = np.ma.masked_equal(mat_data, 0.0)
                        band_list.append(mat_data)
        else:
                #read the single band
                inband = inputimg.GetRasterBand(band_selection)
                mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
                print inband.GetNoDataValue()
                band_list.append(mat_data)

        inputimg = None
        return band_list


def read_image_parameters(input_raster):

        '''Read raster parameters using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :returns:  a list containing rows, columns, number of bands, geo-transformation matrix and projection.
        :raises: AttributeError, KeyError

        '''

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount
        geo_transform = inputimg.GetGeoTransform()
        projection = inputimg.GetProjection()

        inputimg = None
        return rows,cols,nbands,geo_transform,projection
```

```
def write_image(band_list,data_type,band_selection,output_raster,rows,cols,geo_transform,projection):

        '''Write array to file as raster using GDAL

        :param band_list: list of arrays containing the different bands to write (list of arrays).
        :param data_type: numpy data type of the output image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type)
        :param band_selection: number associated with the band to write (0: all, 1: blue, 2: green, 3: red, 4: infrared) (integer)
        :param output_raster: path and name of the output raster to create (*.TIF, *.tiff) (string)
        :param rows: rows of the output raster (integer)
        :param cols: columns of the output raster (integer)
        :param geo_transform: geo-transformation matrix containing coordinates and resolution of the output (array of 6 elements, float)
        :param projection: projection of the output image (string)
        :returns: An output file is created
        :raises: AttributeError, KeyError

        '''

        if data_type == 0:
                gdal_data_type = GDT_UInt16 #default data type
        else:
                gdal_data_type = data_type2gdal_data_type(data_type)

        driver = osgeo.gdal.GetDriverByName('GTiff')

        if band_selection == 0:
                nbands = len(band_list)
        else:
                nbands = 1
        outDs = driver.Create(output_raster, cols, rows,nbands, gdal_data_type)
        if outDs is None:
                print 'Could not create output file'
                sys.exit(1)

        if band_selection == 0:
                #write all the bands to file
                for i in range(0,nbands):
                        outBand = outDs.GetRasterBand(i+1)
                        outBand.SetNoDataValue(99)
                        outBand.WriteArray(band_list[i], 0, 0)
        else:
                #write the specified band to file
                outBand = outDs.GetRasterBand(1)
                outBand.WriteArray(band_list[band_selection-1], 0, 0)
        #assign geomatrix and projection
        outDs.SetGeoTransform(geo_transform)
        outDs.SetProjection(projection)
        outDs = None

def data_type2gdal_data_type(data_type):

        '''Conversion from numpy data type to GDAL data type

        :param data_type: numpy type (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :returns: corresponding GDAL data type
        :raises: AttributeError, KeyError

        '''
        #Function needed when it is necessary to write an output file
        if data_type == np.uint16:
                return GDT_UInt16
        if data_type == np.uint8:
                return GDT_Byte
        if data_type == np.int32:
                return GDT_Int32
        if data_type == np.float32:
                return GDT_Float32
        if data_type == np.float64:
                return GDT_Float64

if __name__ == '__main__':
        main()
```

Appendix H – dynamapCore.py (Milan)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import MySQLdb
import zipfile

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"


def main():
        arg = args()
        if arg.rs:
                enable_resampling = True
                resampling_res = float(arg.rs[0])
        else:
                enable_resampling = False
                resampling_res = 0.0
        if arg.epsg_code:
                epsg_code = int(arg.epsg_code[0])
        else:
                epsg_code = 4326
        base_dir = arg.base_dir

        dynamapCore(base_dir,epsg_code,enable_resampling,resampling_res)

def args():
        parser = argparse.ArgumentParser(description='dynamapCore')
        parser.add_argument("--rs", default=False, nargs=1, help="Enable resampling using cubic spline. Use --rs <value>")
        parser.add_argument("--epsg_code", nargs=1, help="EPSG code for projection")
        parser.add_argument("base_dir", help="Base directory for files")
        args = parser.parse_args()
        return args

def dynamapCore(base_dir,epsg_code,enable_resampling,resampling_res):

        if os.path.isfile(base_dir + 'db.tif'):
                os.remove(base_dir + 'db.tif')

        base = 'milan'
        basemap = '/var/www/html/MappeMilano/' + base + '.tif'
        print basemap

        #Read coefficients
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
        cur = db.cursor()
        laeq = []

        cur.execute('SELECT DeltaLAeq FROM locations WHERE ID>100 ORDER BY ID')
        for row in cur.fetchall():
                laeq.append(row[0])

        #print laeq
        coeff = [np.power(10,c/10) for c in laeq]
        #print coeff
        #LAeq conv in linear
        #Scale
        #Compute new layer
```

```
		new_val_list = []
		band_list = read_image(basemap,np.float32,0)
		rows,cols,nbands,geo_transform,projection = read_image_parameters(basemap)
		for b in range(0,6):
			print 'Band: ' + str(b)
			new_val_list.append(band_list[b]*coeff[b])
		new_mat = np.round(100*np.log10(np.sum(np.array(new_val_list),axis=0)))
		write_image([new_mat],np.uint16,0,base_dir+'db.tif',rows,cols,geo_transform,projection)

		if os.path.isfile(base_dir+'db_rs.tif'):
			os.remove(base_dir+'db_rs.tif')

		#Resample raster
		if enable_resampling == True:
			command = 'gdalwarp -tr {} {} -srcnodata 0.0 -r cubicspline {} {}'.format(resampling_res,resampling_res,base_dir + 'db.tif',base_dir + 'db_rs.tif')
			print command
			proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
			for line in iter(proc.readline, ''): print line
			os.remove(base_dir+'db.tif')

		#Isolines generation
		if os.path.isfile(base_dir+'isolines.shp'):
			os.remove(base_dir+'isolines.shp')
		if os.path.isfile(base_dir+'isolines.shx'):
			os.remove(base_dir+'isolines.shx')
		if os.path.isfile(base_dir+'isolines.dbf'):
			os.remove(base_dir+'isolines.dbf')
		if os.path.isfile(base_dir+'isolines.prj'):
			os.remove(base_dir+'isolines.prj')

		command = 'gdal_contour -i 50.0 -f "ESRI Shapefile" {} {}'.format(base_dir+'db_rs.tif',base_dir+'isolines.shp')
		print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
		os.system(command)
		#Zip output
		#command = 'zip -r {} {}'.format('/var/www/html/isolines.zip',base_dir+'isolines.*')
		#print command
		#proc =
subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stdin=subprocess.PIPE,stderr=subprocess.STDOUT,universal_newlines=True,).stdout
		if os.path.isfile('/var/www/html/isolines.zip'):
			os.remove('/var/www/html/isolines.zip')
		zf = zipfile.ZipFile('/var/www/html/isolines.zip', mode='w')
		zf.write(base_dir+'isolines.shp','isolines.shp')
		zf.write(base_dir+'isolines.shx','isolines.shx')
		zf.write(base_dir+'isolines.dbf','isolines.dbf')
		zf.write(base_dir+'isolines.prj','isolines.prj')
		zf.close()
```

```
def read_image(input_raster,data_type,band_selection):

        '''Read raster using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :param data_type: numpy type used to read the image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :param band_selection: number associated with the band to extract (0: all bands, 1: blue, 2: green, 3:red, 4:infrared) (integer).
        :returns:  a list containing the desired bands as ndarrays (list of arrays).
        :raises: AttributeError, KeyError

        '''

        band_list = []

        if data_type == 0: #most of the images (MR and HR) can be read as uint16
                data_type = np.uint16

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount

        if band_selection == 0:
                #read all the bands
                for i in range(1,nbands+1):
                        inband = inputimg.GetRasterBand(i)
                        mat_data = inband.ReadAsArray().astype(data_type)
                        band_list.append(mat_data)
        else:
                #read the single band
                inband = inputimg.GetRasterBand(band_selection)
                mat_data = inband.ReadAsArray(0,0,cols,rows).astype(data_type)
                band_list.append(mat_data)

        inputimg = None
        return band_list


def read_image_parameters(input_raster):

        '''Read raster parameters using GDAL

        :param input_raster: path and name of the input raster file (*.TIF,*.tiff) (string).
        :returns:  a list containing rows, columns, number of bands, geo-transformation matrix and projection.
        :raises: AttributeError, KeyError

        '''

        inputimg = osgeo.gdal.Open(input_raster, GA_ReadOnly)
        cols=inputimg.RasterXSize
        rows=inputimg.RasterYSize
        nbands=inputimg.RasterCount
        geo_transform = inputimg.GetGeoTransform()
        projection = inputimg.GetProjection()

        inputimg = None
        return rows,cols,nbands,geo_transform,projection
```

```
def write_image(band_list,data_type,band_selection,output_raster,rows,cols,geo_transform,projection):

        '''Write array to file as raster using GDAL

        :param band_list: list of arrays containing the different bands to write (list of arrays).
        :param data_type: numpy data type of the output image (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type)
        :param band_selection: number associated with the band to write (0: all, 1: blue, 2: green, 3: red, 4: infrared) (integer)
        :param output_raster: path and name of the output raster to create (*.TIF, *.tiff) (string)
        :param rows: rows of the output raster (integer)
        :param cols: columns of the output raster (integer)
        :param geo_transform: geo-transformation matrix containing coordinates and resolution of the output (array of 6 elements, float)
        :param projection: projection of the output image (string)
        :returns: An output file is created
        :raises: AttributeError, KeyError

        '''

        if data_type == 0:
                gdal_data_type = GDT_UInt16 #default data type
        else:
                gdal_data_type = data_type2gdal_data_type(data_type)

        driver = osgeo.gdal.GetDriverByName('GTiff')

        if band_selection == 0:
                nbands = len(band_list)
        else:
                nbands = 1
        outDs = driver.Create(output_raster, cols, rows,nbands, gdal_data_type)
        if outDs is None:
                print 'Could not create output file'
                sys.exit(1)

        if band_selection == 0:
                #write all the bands to file
                for i in range(0,nbands):
                        outBand = outDs.GetRasterBand(i+1)
                        outBand.SetNoDataValue(0.0)
                        outBand.WriteArray(band_list[i], 0, 0)
        else:
                #write the specified band to file
                outBand = outDs.GetRasterBand(1)
                outBand.WriteArray(band_list[band_selection-1], 0, 0)
        #assign geomatrix and projection
        outDs.SetGeoTransform(geo_transform)
        outDs.SetProjection(projection)
        outDs = None

def data_type2gdal_data_type(data_type):

        '''Conversion from numpy data type to GDAL data type

        :param data_type: numpy type (e.g. np.uint8, np.int32; 0 for default: np.uint16) (numpy type).
        :returns: corresponding GDAL data type
        :raises: AttributeError, KeyError

        '''
        #Function needed when it is necessary to write an output file
        if data_type == np.uint16:
                return GDT_UInt16
        if data_type == np.uint8:
                return GDT_Byte
        if data_type == np.int32:
                return GDT_Int32
        if data_type == np.float32:
                return GDT_Float32
        if data_type == np.float64:
                return GDT_Float64

if __name__ == '__main__':
        main()
```

Appendix I – dynamapCoreVertical.py (Milan)

```
#!/usr/bin/python

import json
import osgeo.gdal, osgeo.ogr, osgeo.osr
from osgeo.gdalconst import *
import subprocess
import argparse
import os, sys
import numpy as np
import random
import time
import datetime
import MySQLdb
import csv
import psycopg2
import pdb

sys.path = ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages']
os.environ['PATH'] = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"


def main():
        arg = args()
        dynamapCoreVertical()


def args():
        parser = argparse.ArgumentParser(description='dynamapCoreVertical')
        args = parser.parse_args()
        return args


def dynamapCoreVertical():

        start_time = time.time()

        #Get current hour
        now = datetime.datetime.now()
        if now.hour >= 6 and now.hour <= 22:
                interval = 'day'
        else:
                interval = 'night'

        #Read basemap
        base = 'milan'
        basemap = '/var/www/html/MappeMilano/' + base + '.tif'
        print basemap

        #Read coefficients
        db = MySQLdb.connect(host="localhost",user="dynamap",passwd="dynamap",db="Dynamap_Monitoring")
        cur = db.cursor()
        laeq = []

        cur.execute('SELECT DeltaLAeq FROM locations WHERE ID>100 ORDER BY ID')
        for row in cur.fetchall():
                laeq.append(row[0])

        print laeq
        coeff = [np.power(10,c/10) for c in laeq]
        print coeff

        conn = psycopg2.connect("dbname=milan host=localhost user=admin password=dynamap")
```

```
		## Update buildings in DLGS and UE tables
		#Initialize counters for exposed people
		count_dlgs = {"40":0,"45":0,"50":0,"55":0,"60":0,"65":0,"70":0,"75":0,"80":0,"85":0}
		count_ue = {"40":0,"45":0,"50":0,"55":0,"60":0,"65":0,"70":0,"75":0,"80":0,"85":0}

		#Generate attribute names for specific basemap
		query_basemap = ['{}.v{}'.format('ed_maxlevel_ITA',str(i).zfill(2)) for i in range(1,20)]
		query_basemap_ue = ['{}.v{}'.format('ed_maxlevel_END',str(i).zfill(2)) for i in range(1,20)]

		#Update DLGS and UE
		sql = 'SELECT * FROM residential ORDER BY ed_id'
		count_dlgs,count_ue = update_db(conn,sql,count_dlgs,count_ue,coeff,interval)

		#Delete CSV file if existing
		if os.path.isfile('/var/www/html/admin/exposed_occupants_now.csv'):
			os.remove('/var/www/html/admin/exposed_occupants_now.csv')

		#Write counters to CSV file
		with open('/var/www/html/admin/exposed_occupants_now.csv', 'wb') as csvfile:
			field_names = ['Interval','Occupants_DLGS','Occupants_UE']
			writer = csv.DictWriter(csvfile, fieldnames=field_names)
			writer.writeheader()
			writer.writerow({'Interval':'40-44.9','Occupants_DLGS':count_dlgs["40"],'Occupants_UE':count_ue["40"]})
			writer.writerow({'Interval':'45-49.9','Occupants_DLGS':count_dlgs["45"],'Occupants_UE':count_ue["45"]})
			writer.writerow({'Interval':'50-54.9','Occupants_DLGS':count_dlgs["50"],'Occupants_UE':count_ue["50"]})
			writer.writerow({'Interval':'55-59.9','Occupants_DLGS':count_dlgs["55"],'Occupants_UE':count_ue["55"]})
			writer.writerow({'Interval':'60-64.9','Occupants_DLGS':count_dlgs["60"],'Occupants_UE':count_ue["60"]})
			writer.writerow({'Interval':'65-69.9','Occupants_DLGS':count_dlgs["65"],'Occupants_UE':count_ue["65"]})
			writer.writerow({'Interval':'70-74.9','Occupants_DLGS':count_dlgs["70"],'Occupants_UE':count_ue["70"]})
			writer.writerow({'Interval':'75-79.9','Occupants_DLGS':count_dlgs["75"],'Occupants_UE':count_ue["75"]})
			writer.writerow({'Interval':'80-84.9','Occupants_DLGS':count_dlgs["80"],'Occupants_UE':count_ue["80"]})
			writer.writerow({'Interval':'85-89.9','Occupants_DLGS':count_dlgs["85"],'Occupants_UE':count_ue["85"]})

		end_time = time.time()
		print 'Total time: ' + str(end_time-start_time)

def update_db(conn,sql,count_dlgs,count_ue,coeff,interval):

		cur = conn.cursor()
		cur.execute(sql)
		fcount = cur.rowcount
		print 'Total: {}'.format(fcount)

		c = 1
		#Loop on features to update
		for row in cur.fetchall():
			id_build = str(row[0])
			#sys.stdout.write('%s of %s \r' % (str(c),fcount))
			#sys.stdout.flush()
			#print id_build

			param = np.array(row[4:10]).astype('float')
			param_ue = np.array(row[10:16]).astype('float')
			#print param
			param = np.power(10,param/10)
			param_ue = np.power(10,param_ue/10)
			#print coeff
			comb = np.sum(param*coeff)
			comb_ue = np.sum(param_ue*coeff)

			if comb !=0:
				comb = np.round([10*np.log10(comb)],1)
				comb = comb[0]
			else:
				comb = 0
```

```
                        if comb_ue !=0:
                                comb_ue = np.round([10*np.log10(comb_ue)],1)
                                comb_ue = comb_ue[0]
                        else:
                                comb_ue = 0

                        if interval == 'day':
                                diff = comb - float(row[1])
                        elif interval == 'night':
                                diff = comb - float(row[2])
                        if diff < 0:
                                diff = 0

                        #print comb,comb_ue
                        if np.isnan(comb):
                                comb = 0.0
                        if np.isnan(comb_ue):
                                comb_ue = 0.0
                        if np.isnan(diff):
                                diff = 0.0
                        sql_upd = '"UPDATE "residential" SET "dlgs_now"={}, "ue_now"={}, "conflitto_now"={} WHERE
"ed_id"='{}';"'.format(float(comb),float(comb_ue),float(diff),str(id_build))
                        cur.execute(sql_upd)
                        conn.commit()
                        if comb >= 40 and comb < 45:
                                count_dlgs["40"] = count_dlgs["40"] + float(row[3])
                        elif comb >= 45 and comb < 50:
                                count_dlgs["45"] = count_dlgs["45"] + float(row[3])
                        elif comb >= 50 and comb < 55:
                                count_dlgs["50"] = count_dlgs["50"] + float(row[3])
                        elif comb >= 55 and comb < 60:
                                count_dlgs["55"] = count_dlgs["55"] + float(row[3])
                        elif comb >= 60 and comb < 65:
                                count_dlgs["60"] = count_dlgs["60"] + float(row[3])
                        elif comb >= 65 and comb < 70:
                                count_dlgs["65"] = count_dlgs["65"] + float(row[3])
                        elif comb >= 70 and comb < 75:
                                count_dlgs["70"] = count_dlgs["70"] + float(row[3])
                        elif comb >= 75 and comb < 80:
                                count_dlgs["75"] = count_dlgs["75"] + float(row[3])
                        elif comb >= 80 and comb < 85:
                                count_dlgs["80"] = count_dlgs["80"] + float(row[3])
                        elif comb >= 85 and comb < 90:
                                count_dlgs["85"] = count_dlgs["85"] + float(row[3])

                        if comb_ue >= 40 and comb_ue < 45:
                                count_ue["40"] = count_ue["40"] + float(row[3])
                        elif comb_ue >= 45 and comb_ue < 50:
                                count_ue["45"] = count_ue["45"] + float(row[3])
                        elif comb_ue >= 50 and comb_ue < 55:
                                count_ue["50"] = count_ue["50"] + float(row[3])
                        elif comb_ue >= 55 and comb_ue < 60:
                                count_ue["55"] = count_ue["55"] + float(row[3])
                        elif comb_ue >= 60 and comb_ue < 65:
                                count_ue["60"] = count_ue["60"] + float(row[3])
                        elif comb_ue >= 65 and comb_ue < 70:
                                count_ue["65"] = count_ue["65"] + float(row[3])
                        elif comb_ue >= 70 and comb_ue < 75:
                                count_ue["70"] = count_ue["70"] + float(row[3])
                        elif comb_ue >= 75 and comb_ue < 80:
                                count_ue["75"] = count_ue["75"] + float(row[3])
                        elif comb_ue >= 80 and comb_ue < 85:
                                count_ue["80"] = count_ue["80"] + float(row[3])
                        elif comb_ue >= 85 and comb_ue < 90:
                                count_ue["85"] = count_ue["85"] + float(row[3])
                        c+=1
        return count_dlgs,count_ue
if __name__ == '__main__':
        main()
```