\

ClairCity - Citizen Led Air pollution Reduction in Cities

Deliverable 5.4: Generic city model

February 2020

Contents

Document Details

Authors	Kris Vanherle (TML), Vera Rodrigues, Myriam Lopes, Kevin de Oliveira, Sandra Rafael, Ana Patrícia Fernandes (UAVR), Iason Diafas (PBL), Carlo Trozzi (TECHNE), Angreine Kewo (DTU), Joana Soares (NILU), Willem Himpe (TML)
Creation Date	02/10/19
Date of Last Revision	28/02/20
Version	2.2
Dissemination Level	Confidential (CO)
Description	Deliverable 5.4 – Generic city model

Version History

Version	Updated By	Date	Changes / Comments
v1.0	Kris Vanherle	02/10/2019	Initial outline
v2.0	Kris Vanherle	11/02/2020	Added contributions of PBL, DTU, UA, TECHNE, NILU and redacted
v2.1	Kris Vanherle	18/02/2020	Incorporate comments from DTU & UA
v2.2	Kris Vanherle	28/02/2020	Minor adaptation after comments PBL & TECHNE

Project Involvement

Project Director	Hans Bolsher (Trinomics)
Project Manager	Irati Artola (Trinomics)
Project Technical Director	Enda Hayes (UWE)

WP Manager	Kris Vanherle (TML)

1 Background

This deliverable describes the generic city model that enables any EU city to perform “quick scan” impact analysis of any measure that targets air quality and carbon footprint improvement within the city. The generic model builds on the findings of the 6 cases in the ClairCity project (Bristol, Amsterdam, Sosnowiec, Ljubljana, Liguria and Aveiro).

The generic module aims to capture as much as possible city specific aspects of any city wanting to use the tool for policy support, while maintaining a single, modular, approach for efficiency purposes.

The generic model targets mainly small to medium cities, that lack own capability to assess air quality measures with in-house models and/or expertise and replace this by a low threshold generalized approach that can be replicated with many cities. This generic model is one of the components of the “ClairCity approach” which includes the citizens involvement process (Delphi, game,…) and policy involvement (policy workshop,…)

2 Document structure

The generic model requires users to have a minimum of expertise in air quality assessment and/or modelling. The model does not include simplified user interface, yet a set of building blocks of methodological description and software tools, the users can implement in their own assessment tools and processes. Modules are made available with scripts and full code (R, Python, MatLab) if applicable.

We’ve developed 6 stand-alone modules. This document outlines the generic model for each module in following fashion:

- **Input**: listing the required inputs in as much detail as possible and linking directly to data-sources. These data sources are mostly soft-links to existing public data but can also be offline data. Large data sets are included in *D5.2, the generic city database*: https://claircitydata.cbs.nl/dataset/d5-2-cities-database
- **Method:** the methodology to produce the results. We report on the general rationale of the approach as well as the mathematical formulation.
- **Output:** Description of the output, dimensions and format.
- **Implementation:** practical guidelines for the execution of the module. This includes a step-by-step walk-trough and scripts or code in R, Python, Matlab, SQL,…

3 The Modules

We discuss the 6 modular components, each with a specific objective:

1. Land-use module: to derive (annual) energy consumption and NOx/PM10 emissions of all sectors expect transport in fine spatial detail
2. Transport module: to derive (annual) energy consumption and NOx/PM10 emissions of the transport at road-link level
3. Temporal distribution: to derive time-profiles (intra-day, seasonal) of annual emissions of all sectors expect transport
4. Carbon Footprint assessment: to derive carbon footprint from emission inventories (external data or from output of other modules)

5. Air quality and impact module: to derive from the emission data from the other modules (or external data sources)
6. Health module: to convert exposure and air quality data to health impacts using common indicators

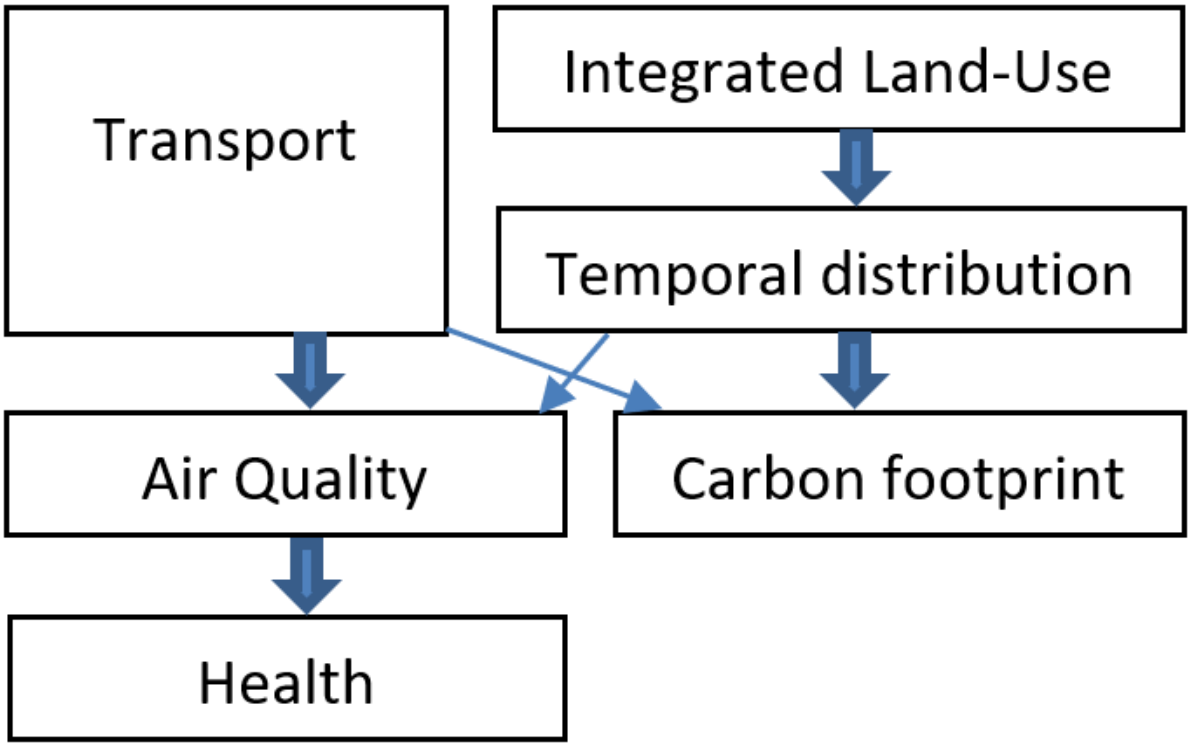


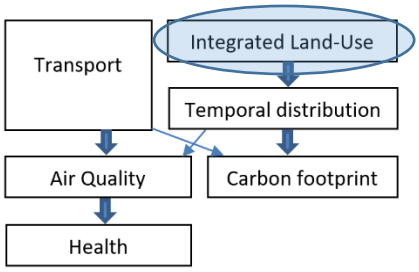
Figure 1: modular structure of the generic model.

All modules can be operated stand-alone, being able to be executed individually from other modules. Users may prefer to use other inputs and only use those modules that are of interest for their specific use case. For example, if an emission inventory is already available but the users has no air quality model, a user could bypass the first 3 modules and directly use own emission data and run the air quality module to understand air quality levels for a specific case.

Vice versa, if a user wants to assess different emission scenario's, with an own air quality modelling tool, the user van single out the transport module to re-assess transport emissions and use it's output as emission input for the air quality module.

We elaborate on each module in the subsequent chapters

3.1 Integrated land-use



In the first step of the ClairCity toolkit, we estimate PM10 and NOX emissions of the residential and commercial sector. This can be replicated for almost all European cities with a population of 50.000 residents and above.

3.1.1 Inputs

For this initial step, we employ two data sources: one pertains to spatial data of estimated emissions, while the other contains land-use data for residential and commercial uses.

With regard to the former, emissions data at a scale sufficiently small to be useful for air quality and health impact assessment for the whole of Europe is not available. At least, not in the sense of actually measured emissions. We therefore make use of an emissions inventory, developed by the Joint Research Centre (Trombetti and Plisoni, 2017), downloaded and stored as resource [1.2] in the generic city database: https://claircitydata.cbs.nl/dataset/d5-2-cities-database. The high-resolution inventory is based on country-level total emission data derived from the Greenhouse Gas and Air Pollution Interactions and Synergies Model (GAINS). The inventory provides emissions for a wide array of sectors and covers NOx, SO2, VOC, PM10, PM2.5 and NH3. It is produced at 1km spatial resolution.

The other data source is the Copernicus Urban Atlas (CUA), which provides pan-European comparable land use, land-cover and population data for Functional Urban Areas (FUA). In its current version it holds data on 785 FUAs covering EU28 + EFTA countries + West Balkans + Turkey. The GIS data can be downloaded, together with a map for each urban area covered and a report with the metadata, from here: https://land.copernicus.eu/local/urban-atlas/urban-atlas-2012?tab=download. Registration is required in order to download the desired data.

3.1.2 Method

The approach to estimate PM10 and NOX emissions for the residential and commercial sector for each city is a rather simple one. It essentially involves combining the two datasets by doing two things:

First, we clip the emission data area by the area covered by the relevant FUA from CUA in order to get an area of emissions that matches geographically the area of the city which we wish to study. Subsequently we 'superimpose' the emissions data on the CUA data, merging in a way the two datasets into one.

Second, we allocate emissions to the FUA areas where the designated CUA land uses are residential and Commercial. Having said this, no distinction is made between residential and commercial land use in terms of the emissions, owing to the fact that the JRC inventory does not distinguish between the two sectors.

The above is done using the R software and the code is described in the attached file.

3.1.3 Outputs

The output is a data set of PM10 and NOX emissions in tons for the FUA areas designated as residential and commercial.

3.1.4 Implementation

Execution of the above procedure is straight-forward. We add a practical example to perform the execution in R:

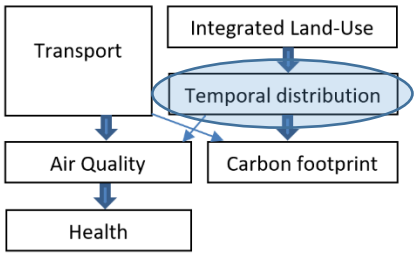
```
#This is an example of how to generate emissions of PM10 for the city of Kavala in Greece. In order to do this for another city,
# you have to a) change the CUA filename in line 6; b) replace'kavala' in the code to the city of your choice;
# if you wish to do the same thing for NOX, just replace 'pm10' with 'NOX' in the code.

setwd("C:/Users/diafasj/OneDrive/PBL/ClairCity/Generic_Cities_Model/emissions1km/PM10/PM10-shapefile/EL008L1_kavalaALA/Shapefiles")
kavalala_copern <- st_read("EL008L1_kavalaALA_UA2012.shp")
the_crs <- st_crs(kavala_copern, asText = TRUE)
setwd("C:/Users/diafasj/OneDrive/PBL/ClairCity/Generic_Cities_Model/emissions1km/PM10/PM10-shapefile")
pm10 <- st_as_sf(st_read("pm10.shp"))
pm10_crs <- st_transform(pm10, crs=the_crs)
pm10_crs_clip <- pm10_crs[kavalaala_copern, ]
thessaloniki_copern_clip <- thessaloniki_copern[pm10_crs_clip, ]
kavala_inters <- st_intersection(kavalaala_copern, pm10_crs_clip)

#apportionment of CUA population to each JRC square km
kavala_inters$apparea <- st_area(kavala_inters)
units(kavala_inters$apparea) <- NULL
kavala_inters$app_pop<-
kavala_inters$Pop2012*(kavala_inters$apparea/kavala_inters$Shape_Area) #apportion population
kavala_inters_PopAggr <- aggregate(app_pop ~ DN, data = kavala_inters, FUN = sum)
#aggregation over square Km

kavala_inters$zone <- as.integer(kavala_inters$zone)
merge <- merge(carlo_pop_emissions, kavala_inters, by='zone')
merge <- data.table(merge[, -c(2:4, 34)])
merge$Hh = merge$app_pop.y/2.7
merge_res <- merge[, -c(2:7, 15:20)]
kavala_res_emissions <- st_as_sf(merge_res)
kavala_res_sum_apparea <- merge_res[, sum(apparea), by=zone]
kavala_res_emissions_sum <- merge(kavala_res_emissions, kavala_res_sum_apparea, by='zone')
kavala_res_emissions_sum <- data.table(kavala_res_emissions_sum)
kavala_res_emissions_sum_percent <- kavala_res_emissions_sum[, area_percent := apparea/V1]
kavala_res_emissions_sum_percent <- kavala_res_emissions_sum_percent %>%
mutate_at(vars(starts_with("res")), funs(.*area_percent))
kavala_res_emissions <- st_as_sf(kavala_res_emissions_sum_percent[, -c(23, 25)])
st_write(kavala_res_emissions, "kavala_residential_emissions1.shp", driver="ESRI Shapefile")
save(kavala_res_emissions, file="kavala_residential_emissions1.Rda")
st_write(kavala_res_emissions, "kavala_residential_emisssions.csv", layer_options = "csv")
```

3.2 Temporal distribution



If emissions at a finer level of temporal detail are required, users can use this module to break the annual emissions from own sources or from the previous step to further break down emission to an hourly resolution, taking into account seasonal variation. This temporal distribution is essential to understand air quality variations between seasons and within a day.
The proposed method consists of four phases: data collection, data pre-processing, data modelling and data clustering.

3.2.1 Inputs

The nature of the datasets in the data-collection process may vary. For reasons of efficiency, we evaluate the input data with the aim of identifying the data that have a significant influence on our goal of generating emission load profiles for regions and cities. Our collected data are in the form of a panel dataset, that is, a cross-sectional data sample at a specific point in time. Therefore, in synthesizing the emission load profiles, we need related datasets dealing with temperature, a national gas supply, a national electricity supply and an emissions area. Data-inputs are:

a. **Monthly gas pattern**: several data sources are possible. One option is the Eurostat database. Alternatively, Joint Organization Data Initiatives (JODI) data provides similar data. Links to datasets are findable on the Dataportal (2.1 & 2.2 https://claircitydata.cbs.nl/dataset/d5-2-cities-database)
b. **Hourly local temperature**: Several open dataset are available from national meteorology offices and private companies such as:
    1. For UK: Metoffice – We used Filton station for the Bristol case
    2. For NL: KNMI – We used Schiphol weather station (ID: 240) for the Amsterdam case

    Several commercial weather service companies provide similar services (e.g. IBM: https://www.ibm.com/weather/industries/cross-industry/graf)
c. **Hourly national electricity load**: https://data.open-power-system-data.org/time_series/ and for the residential sector you can use the Load profile generator (LPG), as explained in the approach above.

3.2.2 Method

Data pre-processing is a required step to support various data sources and formats that suitable to be used in the modelling phase. Depending on the existing data collection, a pre-processing process that consists of some tasks may require. It is due to the incompleteness, noise, ambiguity and inconsistency data in collection. Data corruption, missing values and outliers are the commonest problems in data processing[1].

In general, there four tasks of data pre-processing: Cleaning, transformation, integration and reduction[2] [3]. It is depending on what input data we have, the requirement data format of the tool(s) we will use and the expected output. Cleaning task will fill in the missing values, smooth noise data,

1 Guo Z, Zhou K, Zhang X, Yang S, Shao Z. Data mining based framework for exploring household electricity consumption patterns: A case study in China context. J Clean Prod 2018;195:773–85. doi:https://doi.org/10.1016/j.jclepro.2018.05.254.

2 L'Huillier G, Velásquez JD. Advanced Techniques in Web Intelligence-2. vol. 452. 2013. doi:10.1007/978-3-642-33326-2.

3 Kewo A, Manembu P, Nielsen PS. Data Pre-processing Techniques in the Regional Emissions Load Profile Case. Int Conf Control Decis Inf Technol 2019. doi:https://doi.org/10.1109/CoDIT.2019.8820303.

identify or remove outliers and resolve inconsistencies[4]. The data will be corrected by filling in the missing values into a data warehouse or correct the dataset by using specific a technique, for instance: interpolation[5]. While, transformation task covers normalisation and aggregation. It translates, converts and/or scales the data into the desired formats or units. Furthermore, integration task is utilising multiple databases, data cubes, or files. It also includes combining the various raw dataset into a single dataset[4 6]. Moreover, reduction task will reduce the volume and keep the same analytical results. It includes removal the redundant records and variables[2 4 7]. The following table summarise the common problems and their solutions:

Table 1: Data pre-processing: problems and solutions[4 3]

Task	Problem/issue	Solution/Technique
Cleaning	Missing data	Ignore the record Determine and filling the missing value manually Use an expected value
	Noisy data	Binning methods Clustering Machine learning
	Inconsistent data	External reference Knowledge engineering tools
Transformation	Different format, scale or unit	Normalisation
		Aggregation
		Generalisation
Integration	Different standards among data sources	Combine data into a consistent database
Reduction	Complex analysis or infeasible	Reduce un-necessary observations, variables or values

In general, the regional emissions temporal load profile method is the same for all case cities.

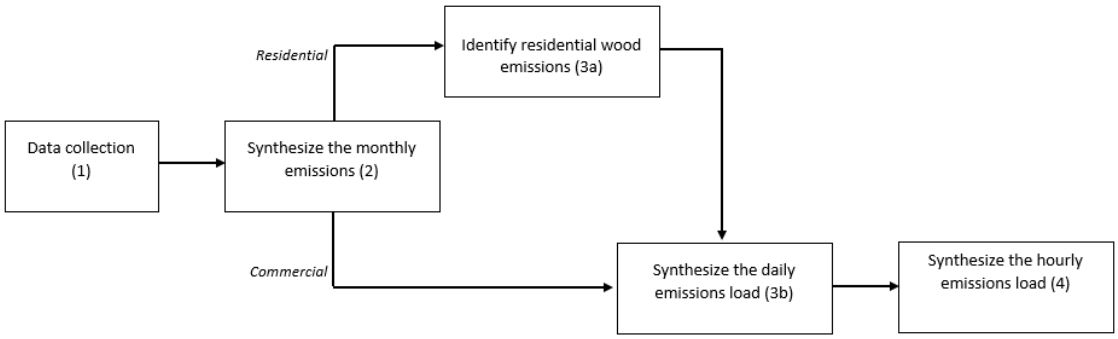


Figure 2: Block diagram for modelling emission load profiles

4 MIT Critical Data. Secondary Analysis of Electronic Health Records. 2016. doi:10.1007/978-3-319-43742-2.

5 Manembu P, Kewo A, Welang B. Missing data solution of electricity consumption based on Lagrange Interpolation case study: IntelligEnSia data monitoring. Proc. - 5th Int. Conf. Electr. Eng. Informatics Bridg. Knowl. between Acad. Ind. Community, ICEEI 2015, 2015. doi:10.1109/ICEEI.2015.7352554.

6 Kim KJ. Lecture Notes in Electrical Engineering 376 Information Science and Applications (ICISA) 2016. 2016. doi:10.1007/978-3-662-46578-3_73.

7 Zulkepli FS, Ibrahim R, Saeed F. Recent Trends in Information and Communication Technology 2018;5. doi:10.1007/978-3-319-59427-9.

The steps of the generic method in the diagram in Figure 2 are the following:

1. Data collection of all required inputs: monthly gas pattern (Gas), hourly local temperature (Temp), hourly national electricity load (El) and share of fuel resources (%), especially wood heaters.
2. Synthesize the monthly emissions load according to the monthly gas pattern.
    a. Residential: identify the residential wood emissions
    b. Commercial and residential: synthesize the daily emissions load according to the daily average temperature pattern.
3. The synthetic daily emissions are distributed from the synthetic total monthly emissions, the wood emissions only being considered for the winter period.
4. Synthesize the hourly emissions load according to the hourly electricity load. The synthetic hourly emissions are distributed from the synthetic total daily emissions.

Residential model

We propose a residential load profiles model by using the weighting proportion of some local parameters to reflect the local characteristic. We combined our model with the Pflugradt's model, which is applied in his tool, namely, the Load Profile Generator (LPG). Pflugrandt's has developed the model with a strong focus on the behavioural aspect. The basic element to model a single household is the desire and expresses the need to do something. The model has specified: weight, threshold and decay time as desire properties[8]. Hence, our model is the determination of selecting some household profiles that are fitted with the weighting proportion.

8 Pflugradt ND. Modellierung von Wasser- und Energieverbräuchen in Haushalten 2016.

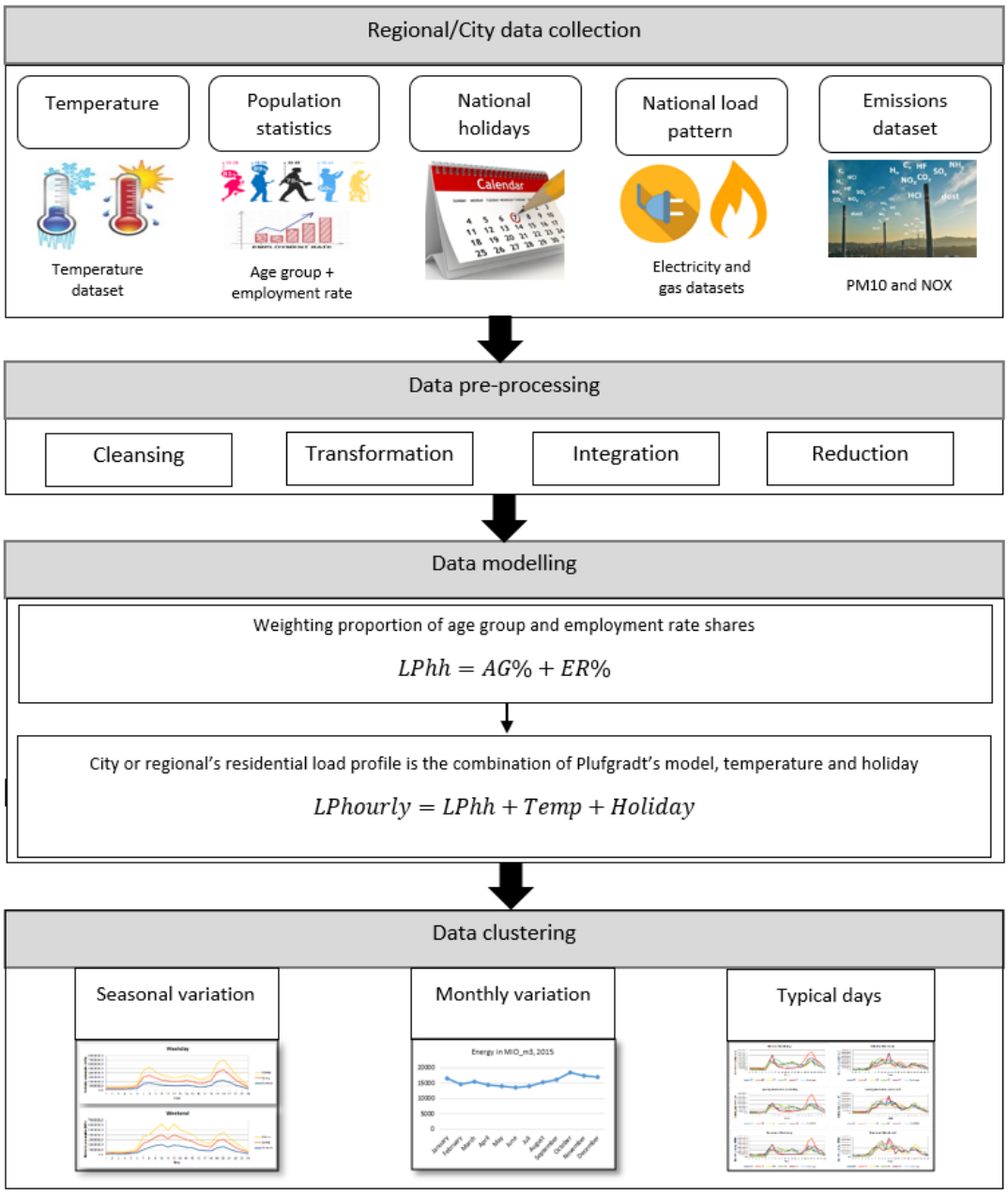


Figure 3: The framework for modelling residential's emission load profiles

The Wepro model is an approach to model the residential load profile at the city level by emphasising on the weighting proportion of some main local parameters. First, it is required to identify the city's age groups (AG), gender (GD) information, and labour force (LF) information. In this case, the total of annual energy consumption is not required, since we only focus on providing the share of hourly energy load profiles. Second, we coupled the share of age groups and labour force, and apply this weighting proportion to the total population, as the main characteristics, which represent the city's profile. In more detail, each age group has gender information. Although, we can also identify the gender information at the higher level of the age groups, which is the total of each gender in the city. In this model, the more detail of gender information of each age group is required.

As mentioned, our model also employed Pflugradt's model and tool, where the early step is to select the household profiles to be modelled by LPG. The fundamental consideration is the selected

household profiles in LPG should represent the city characteristic in term of age groups and labour force, as the focus of our study. Therefore, the weighting proportion of the age groups share and employment share are the fundamental input of the modelling. Consequently, the household load profiles is the combination proportion of age groups share and employment share. The weighting proportion of age groups, gender and labour force is depicted in the picture below.

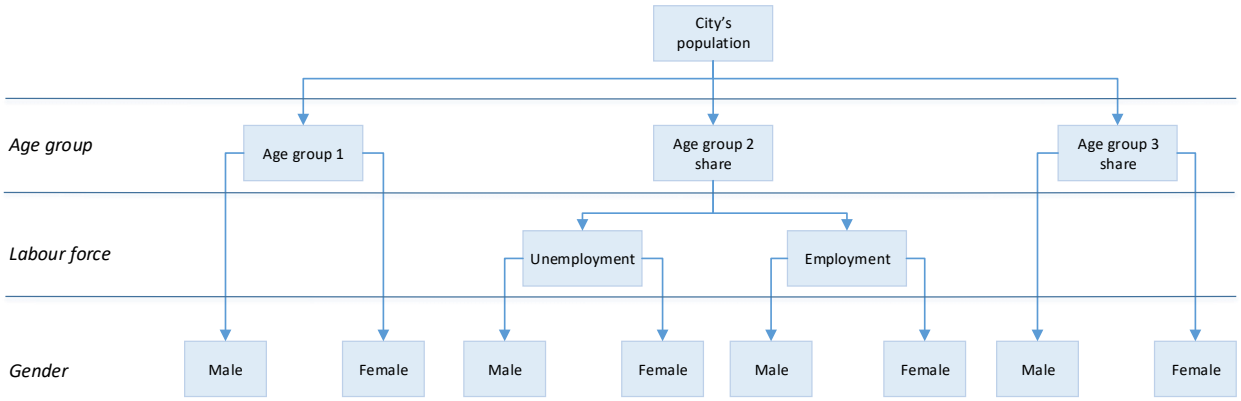


Figure 4: The Weighting proportion structure of residential sector at the city level

3.2.3 Outputs

The outputs are the temporal load profiles, which are clustered in monthly temporal load profiles, daily temporal load profiles and in more specific, the typical days load profiles based on the selected days in hourly resolution. Clustering forms groups that have a common property[9], is commonly used as the technique to characterise the behaviours of consumption and identify the pattern[10]. Since we have developed a model to produce a synthetic load profile, then, the clustering here is focused on the analysis of the load profiles' result, which is the time variations of the emission based on the user energy consumption.

3.2.4 Implementation

The following steps are needed to apply the Weighting proportion (Wepro) model for the residential load profiles and select the representative household profiles of the Wepro model in the Load profile generator. Here are some of the recommended load profile generator:

1. LPG https://www.loadprofilegenerator.de/
2. ALPG https://www.utwente.nl/en/eemcs/energy/profile_generator/ and follow the instructions here to running the ALPG https://github.com/GENETX/alpg

For the data pre-processing and load profiles import Numpy library, Sqlite library and Pandas library

- Numpy library : "NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays"

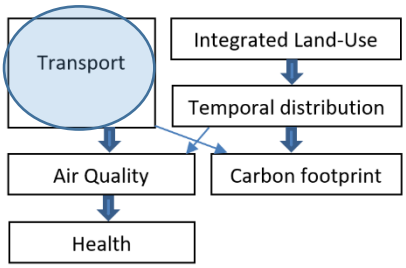
9 Rathod RR, Garg RD. Regional electricity consumption analysis for consumers using data mining techniques and consumer meter reading data. Int J Electr Power Energy Syst 2016;78:368–74. doi:10.1016/j.ijepes.2015.11.110

10 Yang T, Ren M, Zhou K. Identifying household electricity consumption patterns: A case study of Kunshan, China. Renew Sustain Energy Rev 2018;91:861–8. doi:10.1016/j.rser.2018.04.037

- Sqlite library : “SQLite is a relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.”
- Pandas library : “pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license”

The Python code is included in annex.

3.3 Transport



The transport module generates transport emissions at link level, for road transport exclusively. The transport module generates production-attraction from open data to generate transport demand. The tool uses an open assignment algorithm to assign traffic on the network, using OpenStreetMaps (OSM) as a basis for a noded network. Finally, we use existing data to establish modal share and emission factors

3.3.1 Inputs

This module uses a range of open data sources:

- **OpenStreetMaps**: A key source for the network and production attraction to generate transport demand, is OpenStreetMaps (OSM). OSM is fully open and already many applications are building on the open platform. In particular, we use OSMnx functionality to extract data. OSMnx is a Python package that lets you download spatial geometries and model, project, visualize, and analyze street networks from OpenStreetMap's APIs. Users can download and model walkable, drivable, or bikable urban networks with a single line of Python code, and then easily analyze and visualize them. All code is available on GitHub[11].

- **Fleet data**: given that any scenario may affect the fleet composition of passenger cars, we include a dataset with the fleet composition of the passenger car fleet in all European countries for 2016, by age and by type. This data is compiled by TML, using national statistics to feed the MOVEET model. Users can take the fleet composition of the country of the case city to start the assessment. Though regional differences are possible, we found in the 6 ClairCity cases that the city fleet composition is also reflected in the national fleet composition and further detailing to a specific city adds little detail.
  The fleet data is available on the ClairCity data portal in tabular form:
  https://claircitydata.cbs.nl/dataset/d5-2-cities-database/resource/2edbbcc3-717f-4ceb-ba1b-67f2b841e548

- **Modal split**: Various sources are available to establish the modal split of each city. It is likely that locally collected data (for example ad hoc traffic counting data) provides the best insight. However, this type of data is highly fragmented and not centrally stored. The FP7 TRANSPHORM project[12] developed a database with transport indicators at city level. We processed this data to estimate modal split per city.
  The modal share data is available on the ClairCity data portal in tabular form:
  https://claircitydata.cbs.nl/dataset/d5-2-cities-database/resource/c1643866-3324-4373-99b0-e01e24b3f099

- **Emission factors**: For emission factor, we use the common COPERT V methodology[13]. COPERT is the EU standard vehicle emissions calculator. It uses vehicle population, mileage,

[11] https://github.com/gboeing/osmnx
[12] http://www.transphorm.eu/language/en-GB/Home.aspx
[13] https://www.emisia.com/utilities/copert/

speed and other data such as ambient temperature and calculates emissions and energy consumption for a specific country or region.
For this project, we have derived processed emission factors that are relevant for urban air quality modelling; we've selected urban emission factors only and aggregated to the same dimension of fleet data as described above. Data is available in tabular format with the following dimensions:

- Vehicle type: diesel, petrol, CNG, LPG, PHEV (in line with fleet data)
- Pollutant: NOx or PM 3
- Peak/off-peak: (P/OP)
- year of built (YOB): from 1965 to projections up to 2050

The emission factor data in g/km is available on the ClairCity data portal in tabular form: https://claircitydata.cbs.nl/dataset/d5-2-cities-database/resource/a8045108-63cc-4179-baba-9c46abd42067

3.3.2 Method

We focus here on the process to generate transport demand at link level, from OSM as the application of emission factors in a final step is trivial.

Classical transport models are composed of 4 connected and integrated stages:

- Production & Attraction
- Distribution
- Mode choice
- Assignment

Only the first two stages are important for demand generation.

Demand generation

Production factors define the generation of demand for a zone. The factors feed into a function that describes the total amount of trips being generated in a zone. In most cases the trip generation function is a multi-variable regression model based on socio-economic variables such as population density, age distribution, income levels, etc...

The attractiveness of a zone as a trip end is mostly defined by infrastructural/spatial characteristics. The total amount of trips that dissipate in a zone is also described by multi-variable regression model based on number of available workplaces, schools, quantity and quality of shopping locations, availability of leisure activities, etc…

In the second stage total amount of trips generated and dissipated for all zones are combined into a trip matrix connecting each production zones with attraction zones. The relation between zones described in function of the distance or travel times between zones. A gravitational weight function is used to describe decreasing attractiveness of distant zones.

The first two stages lead to a very rough estimation of the number of trips in a network. Further calibration of the trip matrix is required to match the trip matrix with actual demand patterns. This typically based on observations such as link volumes or aggregated trip statistics.

The traditional approach introduces artificial links called connectors to link the demand of a zone to the network. Each zone is represented by a centroid at the center of its shape and a handful of connectors are drawn between this center point and the import roads of the zone.

Figure 5: centroid approach for traffic generation

The main critique is that for small roads and interzonal traffic the placement and weights of the connectors has a non-neglectable effect on the local network. For this reason, this method is not applied here.

To account for distribution of demand within a zone a larger amount of departure or arrival points are required. Therefor a subset of 100+ vertices (crossings) are selected within each zone or within a distance from the centroid.

At the boundary of the case study area additional traditional connectors are required. Depending on the distance from the case study area, the external zones are physical connected with the dead-end vertices of a certain link level.

Mode choice

Instead of a full model to estimate mode choice (car/bus/active travel/…), which could depend on supply, prices and population preferences, we opt for a simplified approach and use observed modal shares from cities, as is available trough public data. We have compiled a databases, based on the TRANSPHORM city database with modal shares for all EU cities, as described in the input section of this chapter

Assignment

The main idea of assigning demand to the network is based on equilibrium principles. These state that drivers will keep on looking for shorter routes until all drivers unilateral perceive the least resistance.

Resistance is defined as a generalized cost which is the sum of link specific cost functions. The cost function used is a BPR curve that is defined in function of free flow travel time, capacity and link flow.

$$Cost = t_0\left(1 + \alpha\left(\frac{Flow}{Capacity}\right)^{\beta}\right)$$

Algorithmically the equilibrium process translates into an iterative process where drivers are repeatedly loaded onto the network until no drivers switches routes. For efficiency reasons drivers originating from the same source are loaded according to similar routes. This allows for an implicit formulation of the route tree rooted in each origin. At every node in the network only the set of efficient inbound links per origin are required. These are the links for which the starting point is

topologically closer to the origin. The efficient links are computed based on a Dijkstra shortest path three evaluated in free flow conditions.

Assignment – initial loading

The assignment is for a full day. Capacities are adjusted accordingly. It is assumed that the maximum hourly road capacity is adjust to a full day and that this factor is a parameter to control for responsiveness of drivers with respect to busy roads. The factor is set to 10 which introduces mild responsiveness and a quick convergence of the algorithm.

The stochastic distribution of flows over the different route alternatives is determined by a parameter which coincides with the mu parameter of logic models. Mu is set to 20. This amounts to a reduction of almost 70% on a route if 5 additional minutes are incurred.

The trip matrix with demand for cars and trucks is combined using a weight factor for trucks. Each truck is counted as 3 vehicles in the assignment.

The demand is distributed per zone according to the following rules:

- 200 nodes are selected within a zone that is part of the case-study area
- Zones that are close to the case-study are mapped to the nearest road of the lowest OSM class (residential, road, living_street or unclassified) that cuts the cordon around the case-study area
- Zones that are further away from the case-study are mapped to the nearest road of all higher OSM classes that cut the cordon around the case-study area
- Zones that are very far away from the case-study are mapped to the nearest road of the highest OSM classes (trunk, trunk_link, motorway, motorway_link) that cut the cordon around the case-study area

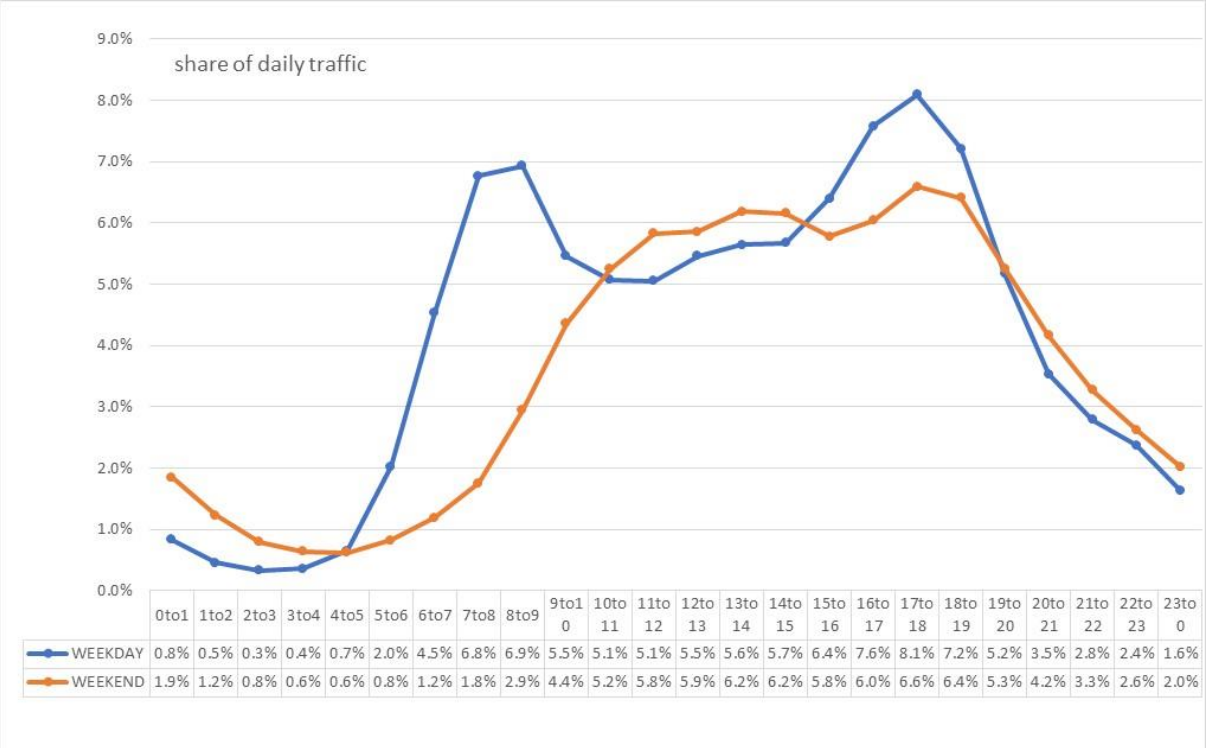
Post-processing

The initial demand generation and assignment need further refinement. The trade-off here is effort vs. accuracy. Users can choose to what extend post-processing is needed for their specific case. We list step-wise refinement options in sequence of importance: low-effort, high impact on accuracy first and high-effort, limited further impact on accuracy last:

- **Refinement 1 - Car & Truck separation**: Keeping all cost of the equilibrium constant, the assignment is rerun but with the separate demand for cars and trucks. This results in link flows for those 2 classes of drivers.
- **Refinement 2 - OD corrections and local road attractiveness**: For some of the origin or destinations in the network a straightforward correction can be applied. All the highest OSM class roads that cut the cordon around the case-study area are origins and destinations in the final trip matrix. This means that a single factor per origin row or destination column can be applied to match the total sum of a row / column with observed averages volumes per day.
  Drivers are reluctant to choose small roads for their daily travels. Therefore, travel costs for roads classified by OSM as “residential”, “road”, “living_street” or “unclassified” are inflated by a factor 4.
  The assignment is rerun with these adjustments and just as in refinement 2 the equilibrium cost are used to assignment the 2 classes of trips.

- **Refinement 3 - Local count corrections**: An estimated selected link assignment is performed based on the shortest path trees emitted upstream and downstream of the observed road. These trees are combined with the observed splitting rate at diverge/merge points to adjust redistribution weights of the deviation with the counts.

Finally, as volumes are estimated for daily totals, a final step is needed to distribute intensity by time of day. This is fairly trivial and can be done using various data that is specific for the local situation. In table and figure below, the estimates we've used, based on observed highway traffic intensity (a good proxy for all roads), making a distinction between weekday and weekend. Note that the sum over all hours is 1 for weekday, but lower for weekend, as traffic generation an assignment is assumed for a weekday with typical peak-profiles.



	0to1	1to2	2to3	3to4	4to5	5to6	6to7	7to8	8to9	9to10	10to11	11to12	12to13	13to14	14to15	15to16	16to17	17to18	18to19	19to20	20to21	21to22	22to23	23to0
WEEKDAY	0.8%	0.5%	0.3%	0.4%	0.7%	2.0%	4.5%	6.8%	6.9%	5.5%	5.1%	5.1%	5.5%	5.6%	5.7%	6.4%	7.6%	8.1%	7.2%	5.2%	3.5%	2.8%	2.4%	1.6%
WEEKEND	1.9%	1.2%	0.8%	0.6%	0.6%	0.8%	1.2%	1.8%	2.9%	4.4%	5.2%	5.8%	5.9%	6.2%	6.2%	5.8%	6.0%	6.6%	6.4%	5.3%	4.2%	3.3%	2.6%	2.0%

Figure 6: share of daily traffic by type of day, compared to a typical weekday

3.3.3 Outputs

The results of the above procedure is a .csv file with MultiLineString information compatible with GIS-software (ArcGis, QGis,…), holding all data from OSM on the network (speed, length, road type, name, class,…) supplemented with the flow estimates (unit = vehicles per day). Further detailing by hour in the day is trivial (see figure above).

Emission factors can readily be applied to the traffic volumes, or can be re-estimated to simulate impact of a transport policy scenario. For example, banning of old cars will remove old car from the fleet, requiring a re-estimation of the average emission factor with the available data on fleet and emission factor by type and YOB.

3.3.4 Implementation

Below is a step-by-step guide how to use the OSMnx functionality, the assignment procedure and the refinement steps. This requires GIS-software (we recommend QGis) and Matlab.

Extract data from OSM:

OSMnx is a Python package that lets you download spatial geometries and model, project, visualize, and analyze street networks from OpenStreetMap's APIs.

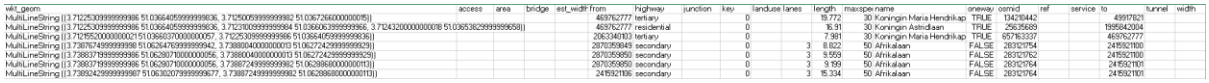
https://github.com/gboeing/osmnx

Extract road network and construct a connected graph for the road network. In such a connected graph routes between locations can easily be computed – example below:

```
import osmnx as ox
G = ox.graph_from_place('Gent, Oost-Vlaanderen, Vlaanderen, België', network_type='drive', simplify=False)
G = ox.simplify_graph(G, strict=False)
ox.plot_graph(ox.project_graph(G))
ox.save_graph_shapefile(G, filename='OSMNX_network-Gent-false', folder='C:/Projects/OSMnx')
```

The previous code will export data for two shapefiles. One with road data & one with node data. Save files as .csv to import into Matlab for the assignment procedures. This can be easily done through QGIS (EPSG:4326)

The result will look like figure below



wkt_geom	highway	osmid	ref
MultiPoint ((3.71225309999999986 51.03664059999999836))		469762777	
MultiPoint ((3.73880040000000013 51.06272429999999929))		2415921100	
MultiPoint ((3.73887249999999982 51.06288680000000113))		2415921101	
MultiPoint ((3.73892429999999987 51.06302079999999677))		2415921106	
MultiPoint ((3.70224710000000012 51.04357009999999661))		50334950	
MultiPoint ((3.70236130000000019 51.04197380000000095))		50334956	
MultiPoint ((3.72886879999999987 51.02583560000000062))		1493177132	
MultiPoint ((3.72937139999999978 51.02561990000000236))		1493177153	
MultiPoint ((3.7290646999999999 51.02589940000000013))		1493177154	

Figure 7: printscreen of resulting .csv file for edges.csv (top) and nods (bottom)

Next, read roads, nodes with matlab scripts:

Matlab program code:

Read_edges.m

```
%% Import data from text file
% Script for importing data from the following text file:
%
%    filename: C:\Projects\IRCEL\edges.csv
%

%% Setup the Import Options
opts = delimitedTextImportOptions("NumVariables", 21);

% Specify range and delimiter
opts.DataLines = [2, Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["wkt_geom", "access", "area", "bridge", "est_width", "from", "highway", "junction",
"key", "landuse", "lanes", "length", "maxspeed", "name", "oneway", "osmid", "ref", "service", "to",
"tunnel", "width"];
opts.VariableTypes = ["string", "string", "string", "string", "string", "double", "categorical", "categorical",
"double", "string", "double", "double", "double", "categorical", "categorical", "double", "string", "string",
"double", "string", "double"];
opts = setvaropts(opts, [1, 2, 3, 4, 5, 10, 17, 18, 20], "WhitespaceRule", "preserve");
opts = setvaropts(opts, [1, 2, 3, 4, 5, 7, 8, 10, 14, 15, 17, 18, 20], "EmptyFieldRule", "auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
edges = readtable("C:\Projects\IRCEL\edges.csv", opts);
edges.co=arrayfun(@(x)[reshape(str2double(split(strrep(erase(erase(x,'MultiLineString ((('),'))'),',','
'))),2,[])';nan(1,2)],edges.wkt_geom,'UniformOutput', false);
edges.oneway(edges.oneway=='TRUE')='True';
edges.oneway(edges.oneway=='FALSE')='False';
%% Clear temporary variables
clear opts
```

Read_nodes.m

```
%% Import data from text file
% Script for importing data from the following text file:
%
%    filename: C:\Projects\IRCEL\nodes.csv
%

%% Setup the Import Options
opts = delimitedTextImportOptions("NumVariables", 4);

% Specify range and delimiter
opts.DataLines = [2, Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["wkt_geom", "highway", "osmid", "ref"];
opts.VariableTypes = ["string", "string", "double", "double"];
opts = setvaropts(opts, [1, 2], "WhitespaceRule", "preserve");
opts = setvaropts(opts, [1, 2], "EmptyFieldRule", "auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
nodes = readtable("C:\Projects\IRCEL\nodes.csv", opts);
nodes.co=arrayfun(@(x)[reshape(str2double(split(erase(erase(x,'MultiPoint
(('),'))')))),2,[])'],nodes.wkt_geom,'UniformOutput', false);

%% Clear temporary variables
clear opts
```

Some data cleaning is advisable, for outliers and missing values:

Key:Highway (road class)
https://wiki.openstreetmap.org/wiki/Key:highway

- Remove unnecessary classes
- Clean-up non existing values

Key:Maxspeed (speed limit)
https://wiki.openstreetmap.org/wiki/Key:maxspeed

- Clean-up nonrealistic values
- Fill in missing value based on most likely speed for that road class

Key:Lanes (number of lanes on a road)
https://wiki.openstreetmap.org/wiki/Key:lanes

- Clean-up nonrealistic values
- Fill in missing values based on most likely number of lanes for that road class

The generation of an OD-matrix from production and attraction is then done as follows:

```
function [OD,skim] = make_OD(zones,nodes,links,pairs,no_zones)

%%
OD = zeros(max(nodes.id));
skim = zeros(max(nodes.id));

totLinks = size(links.toNode,1);
totNodes = size(nodes.x,1);
strN = links.fromNode;
endN = links.toNode;


%% Set the skim matrix
alpha = 0.15;
beta = 4;
travelCosts = calculateCostBPR(alpha,beta,zeros(length(links.freeSpeed),1),links.length,links.freeSpeed,links.capacity);

ref_l = unique(zones.ref_l);
o_l=arrayfun(@(x) any(links.class(x) == [1:2:8]) & nodes.in(nodes.id==links.fromNode(x))==0,[1:length(links.class)]);
o_l=find(o_l);
d_l=arrayfun(@(x) any(links.class(x) == [1:2:8]) & nodes.out(nodes.id==links.toNode(x))==0,[1:length(links.class)]);
d_l=find(d_l);

ref_l = [ref_l;o_l';d_l'];

P = zeros(max(nodes.id),1);
A = zeros(1,max(nodes.id));

[rp,ci,ai]=sparse_to_csr(strN,endN,1:length(links.length),max(nodes.id)); %
a = travelCosts(ai);

h=waitbar(0,'Please wait...');
totZones=length(ref_l);
for o = 1:totZones
   waitbar(o/totZones,h);
   if o<=totZones-(length(d_l)+length(o_l))
      if zones.dist_l(o)> 10^-5
         continue;
      end
      distance = dijkstra_v2(rp,ci,a,links.fromNode(ref_l(o)));
      skim(links.fromNode(ref_l(o)),links.toNode(ref_l))=distance(links.toNode(ref_l));
      P(links.fromNode(ref_l(o)))=P(links.fromNode(ref_l(o)))+zones.population(o);
      A(links.toNode(ref_l(o)))=A(links.toNode(ref_l(o)))+exp((24-zones.class(o))/2);
   elseif o<=totZones-(length(d_l))
      distance = dijkstra_v2(rp,ci,a,links.fromNode(ref_l(o)));
      skim(links.fromNode(ref_l(o)),links.toNode(ref_l))=distance(links.toNode(ref_l));

      P(links.fromNode(ref_l(o)))=P(links.fromNode(ref_l(o)))+1*links.capacity(ref_l(o));
%       P(links.fromNode(ref_l(o)))=P(links.fromNode(ref_l(o)))+0.1*links.capacity(ref_l(o));
   else
      A(links.toNode(ref_l(o)))=A(links.toNode(ref_l(o)))+10*links.capacity(ref_l(o));
   end
end
close(h)
A(no_zones)=0;
P(no_zones)=0;
A = A/sum(A)*sum(P);
if ~isempty(pairs)
   skim(sub2ind(size(skim),pairs(:,1),pairs(:,2)))=0;
end

plotLoadedNodes(nodes,links,A,false,[],1,[]);
plotLoadedNodes(nodes,links,P,false,[],1,[]);

f_temp = exp(-skim*0.1);
val_g_prev = inf;
val_g = sum(sum(abs(f_temp)))*1000000;
it = 0;
h = figure;
semilogy(0,NaN);
start_time = cputime;
while it<50 && val_g < val_g_prev && val_g > 10
   it = it+1;
   val_g_prev = val_g;
   %redistribute production;
   f_temp_new = repmat(P./(sum(f_temp,2)+eps),1,max(nodes.id)).*f_temp;
   %redistribute attraction;
   f_temp_new = repmat(A./(sum(f_temp_new,1)+eps),max(nodes.id),1).*f_temp_new;
   %check convergence
   val_g = sum(sum(abs(f_temp_new-f_temp)));
   f_temp = f_temp_new;
   figure(h)
   hold on
   semilogy(cputime-start_time,val_g,'r.')
end

OD = f_temp;
end
```

Assignment:

The next step is the assignment to the network. Below is a schematic overview of the algorithm

```
%initialisation
Compute dijkstra costs and store in cost-ordered structure

%equilibrium process
While not converged do
   COMPUTE origin-based links flows with Dail's procedure
   MSA UPDATE of link flows
   UPDATE link costs
End while
```

Implementation in Matlab consists of 5 functions

- DFAST.m Main algorithm for finding equilibrium
- calculateCostBPR.m Calculate link costs based on BPR function
- sparse_to_csr.m Compressed row storage for efficient backward node star lookups
- dijkstra_v2.m Compute shortest paths using Dijkstra's algorithm
- Dial_F.m Subroutine with Dail's method for calculating stochastic loading

Given the length of the code, all of the above Matlab-scripts are added in annex

Post-processing:

Data refinement is optional as indicated. In annex, we add an example for a fully elaborated case (for city of Ghent) with some available data. The scripts for initial load and refinement are:

- The initial loading is performed with a matlab script MAIN_GENT_INIT.m
- The first refinement is performed with a matlab script MAIN_GENT_REF1.m
- The second refinement is performed with a matlab script MAIN_GENT_REF2.m
- The third refinement is performed with a matlab script MAIN_GENT_REF3.m and requires additional MatLab functions:
    - double_tree.m: Dijkstra algorithm for finding shortest path trees
    - distribute.m: Distribute unit flow according to flow rates in the double tree

For the example case, for the second refinement the considered highways for which data is available and used are:

- E17N -> Gent
- E17N -> Antwerpen
- E17Z -> Gent
- E17Z -> Kortrijk
- E40E -> Gent
- E40E -> Brussel
- E40W -> Gent
- E40W -> Oostende

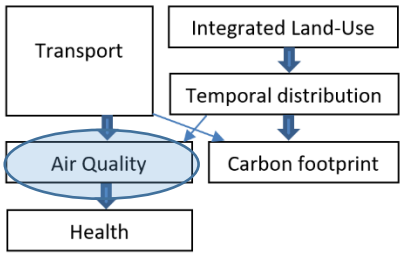
For the example case, for the third refinement the considered additional data points for which data is available and used are:

- Burgravenlaan

- Lange Violettestraat
- Edmond Blockstraat
- Dendermondsesteenweg
- Annonciadestraat
- R4
- R4 binnenring

Application of emission factors is trivial and can be done in a GIS-interface (e.g. add attribute tables in QGis and link via ID)

3.4 Air Quality



The urban air quality numerical model URBAIR was developed by the Department of Environment and Planning of the University of Aveiro. URBAIR is a second generation Gaussian model able to provide air pollutant concentrations at urban scale, with higher spatial resolution than the ones obtained with the mesoscale model system. The air quality patterns are numerically simulated for a given computational domain (with up to about 10 km, depending on grid resolution) and a given time period, e.g. hourly, daily or yearly, for distinct air pollutants, namely PM10, PM2.5 and NO_2. The scheme of the URBAIR modelling system (inputs, model structure and outputs) is presented in Figure 3-1. The detail description of the inputs files and the output file are included on the follow sub-sections.

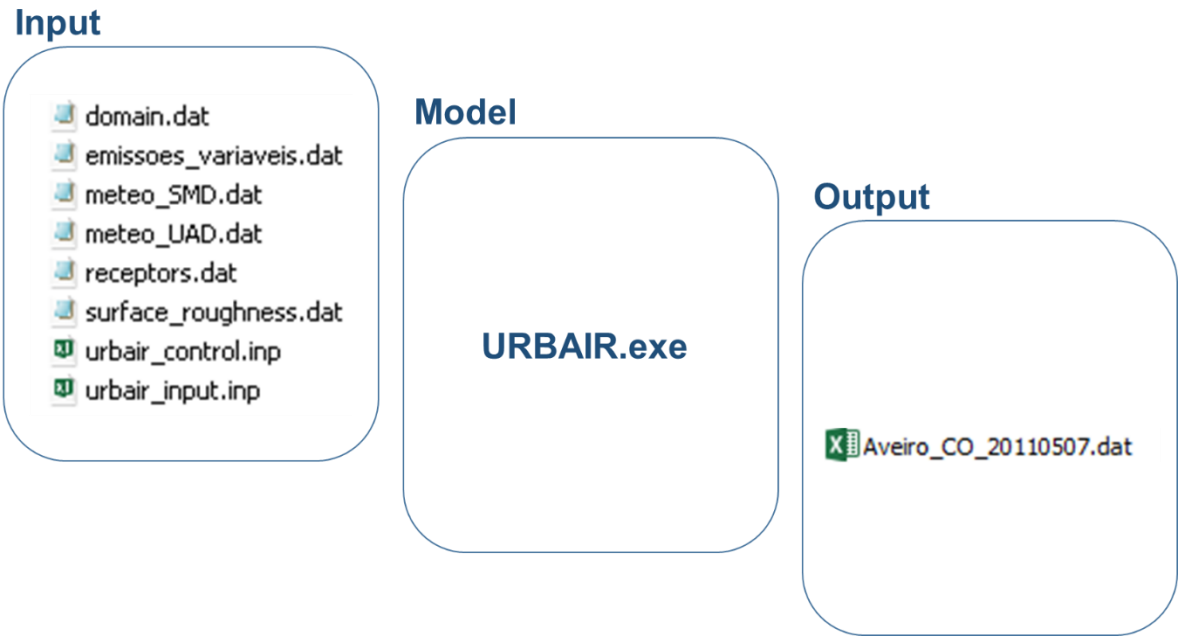


Figure 8: Scheme of the URBAIR modelling system.

3.4.1 Inputs

To run the Urban Air Quality Model system eight input files are required:

- urbair_input.inp
- urbair_control.inp
- meteo_SMD.dat
- meteo_UAD.dat
- surface_roughness.dat
- domain.dat
- receptors.dat
- emissoes_vatiaveis.dat

The detail description of each input files is include below.

Urbair_input.inp

The file Urbair_input.inp intend that the user to define the simulation options. This file brings together the main calculation control options as well as the input and output file names. The follow information is required:

- Name of the project
- Run options
- Time period of the case study (star and end time period)
- Time step
- Geographical characterization: latitude, longitude, time zone
- Simulation domain extreme points
- Meteorological information: number of surface measure levels and number of vertical records per day
- Location of the input files and output file
- Total number of receptors
- Emission source characterization

Urbair_control.inp

The file Urbair_control.inp includes the control parameters for different variables that are considered by URBAIR model. Those parameters are related with control model runner, meteorological data, emissions, topography and concentrations. As a control input, this file is kept constant.

meteo_SMD.dat

The file meteo_SMD.dat aims to characterize the meteorological surface data, being required information for the following variables:

- Simulation period: year/month/day/hour/min/sec
- Meteorological information:
    - ta1 - temperature (^{o}C),
    - vv1 - wind speed (m/s),
    - dv1 - wind direction (º),
    - Pa - Pressure (Pa),
    - hr1 - Relative humidity (%),
    - z0 – roughness length (m),
    - bow – Bowen ratio (-),
    - albed - Albedo (-).

meteo_UAD.dat

The file meteo_SMD.dat aims to include information of meteorological vertical profiles, being required information about:

- Index daily of the vertical profile (k)
- Number of vertical profiles recoded (nr. sounds)
- Day/month/year and hour/min/sec of each vertical profile

- Meteorological data recorded in each vertical profile: level (meters), pressure (Pa), temperature - temp (ºC), dew point temperature – dew point temp (ºC), wind speed (m/s) and wind direction (º)

surface_roughness.dat

The surface_roughness.dat aim to characterize the surface of the study area, being required the following variables:

- year/month/day/hour/min/sec
- Cdz0 - zero displacement length (m)
- Z0 - roughness length (m)
- Bow - Bowen ratio (-)
- alb – Albedo (-)

domain.dat

The domain.dat includes the information of the study domain, being required the following variables:

- orlat/orlong/orzon – latitude/longitude/time zone
- number of the receptors points
- x axis information – initial point, number of the points, delta space
- y axis information – initial point, number of the points, delta space

receptors.dat

The receptors.dat (Figure 3-8) requires information about:

- Total number of discrete receptors points
- Coordinates of each discrete receptor point (x/y/z) in meters (UTM)

emissoes_variaveis.dat

The emissoes_variaveis.dat characterizes the atmospheric emissions under the study area during the study period, being required the following information:

- year
- month
- day
- hour
- min
- sec
- ID – number of each source
- q - flow rate (g/h)
- Vel – emission velocity (m/s)
- Tp - temperature (K);
- P – pressure (Pa)

3.4.2 Method

URBAIR is a second generation Gaussian model able to provide air pollutant concentrations at urban scale, with higher spatial resolution than the ones obtained with the mesoscale model system. The air quality patterns are numerically simulated for a given computational domain (with up to about 10 km, depending on grid resolution) and a given time period, e.g. hourly, daily or yearly, for distinct air pollutants, namely PM10, PM2.5 and NO_2. The simulations for different pollutants need to be made separately, since URBAIR model doesn´t allow to run in parallel the different pollutants. This means that URBAIR inputs need to be prepared individually for each pollutant. URBAIR model considers a simple NO_X chemistry and a method for the simulation of dry and wet deposition mechanisms.

The URBAIR outputs allow three types of analysis: i) assessment of the spatial variation of pollutants concentration (2D concentration fields); ii) assessment of air quality patterns (in an hourly, daily, monthly, yearly basis); and iii) assessment of temporal variation of the pollutant concentration in discrete points. This last feature is particularly useful to model validation (through comparison with air quality measured data, taken in a specific discrete point).

The URBAIR model (URBAIR.exe) is an executable file compiled to be support by Linux System.

3.4.3 Outputs

The output file includes the information of the receptor point localization (x and y coordinates), the concentration values and the corresponding time (hour). The outputs file name are structured using the information about the case study name, the study pollutant and the data information as shown in this example: Aveiro_CO_20150501.dat ([Case study]_[Pollutant]_[data(year/month/day)].dat). The post-processor developed to read this file is described in Section 4.5.

3.4.4 Implementation

In this section we elaborate on the specific execution steps and share (Python) code to run the air quality model.

- urbair_input.inp (created manually)
- urbair_control.inp (kept constant)
- meteo_SMD.dat
- meteo_UAD.dat
- surface_roughness.dat (created manually)
- domain.dat
- receptors.dat
- emissoes_variaveis.dat

Meteorological data

As previously mentioned, two kind of meteorological files are required by URBAIR: meteo_SMD.dat & meteo_UAD.dat. These files can be obtained by two different sources: i) meteorological modelling system outputs or ii) measured data.

The surface measured data can be obtained from meteorological stations included under the study domain, while the vertical information can be obtained from the website http://weather.uwyo.edu/upperair/sounding.html.

Regardless of the source of the meteorological data, meteorological surface information (input of the code program) need to include the following variables:

- Tempo (Data day/month/year Hour:min);

- Albedo (-);
- PSFC - Surface pressure (Pa);
- RAINC - Accumulated total cumulus precipitation (mm);
- RH - Relative Humidity (%);
- T2C - Temperature (^{o}C);
- Rugosidade - Roughness (-);
- U10 - wind component at 10 meters (-);
- V10 - wind component at 10 meters (-)

2D_WRFOUT_2010_CIRA_1.csv

```
tempo; Albedo; PSFC; RAINC; RH; T2C; Rugosidade; U10; V10
01/01/2010 00:00;0.17;100315;0;77.2498;9.55844;1.5;9.87859;1.24377
01/01/2010 01:00;0.166203;100470;0;77.6718;10.0878;1.5;4.44076;2.10753
01/01/2010 02:00;0.166203;100319;0.499201;68.83;11.6695;1.5;6.94309;1.53842
01/01/2010 03:00;0.166203;100315;1.40916;72.6167;11.5784;1.5;7.78886;1.90752
01/01/2010 04:00;0.166203;100338;2.17843;71.8808;11.7586;1.5;8.53637;1.96846
01/01/2010 05:00;0.166203;100371;2.79895;71.2353;11.8627;1.5;9.22461;2.21551
01/01/2010 06:00;0.166203;100433;3.43108;75.074;11.6699;1.5;9.62241;1.77805
01/01/2010 07:00;0.166203;100467;3.43108;72.3164;12.113;1.5;9.02455;1.08105
01/01/2010 08:00;0.166203;100503;4.23599;79.749;11.5475;1.5;9.69432;0.592557
01/01/2010 09:00;0.166203;100675;5.03546;74.4702;12.0568;1.5;8.11075;-2.0504
01/01/2010 10:00;0.166203;100791;5.03546;68.24;12.8105;1.5;7.51506;-1.95868
01/01/2010 11:00;0.166203;100909;5.03546;59.8833;13.2495;1.5;8.72037;-2.93217
01/01/2010 12:00;0.166203;101021;5.03546;62.2681;13.3705;1.5;8.13984;-2.22324
01/01/2010 13:00;0.166203;101032;5.03546;57.6862;13.5264;1.5;6.97152;-2.49471
01/01/2010 14:00;0.166203;101058;5.03546;56.6547;13.6298;1.5;6.78473;-2.7138
01/01/2010 15:00;0.166203;101045;5.03546;54.3765;13.4902;1.5;6.51607;-2.22441
01/01/2010 16:00;0.166203;101083;5.03546;59.7792;12.8542;1.5;4.65762;-1.66746
01/01/2010 17:00;0.166203;101173;5.03546;78.1658;8.63635;1.5;3.03455;-1.03783
01/01/2010 18:00;0.166203;101194;5.03546;77.8967;8.09152;1.5;3.16317;-0.123678
01/01/2010 19:00;0.166203;101219;5.03546;78.6501;7.88867;1.5;3.15002;0.133505
```

Figure 9: data format for horizontal meteo data

The meteorological vertical information (input of the code program) need to include the following variables:

- Tempo – (Data day/month/year Hour:min);
- Nivel – number of each level;
- P - Pressure (Pa);
- H - Relative Humidity (%);
- TC - Temperature (ºC);
- U – wind component at each level (-);
- V – wind component at each level (-)

3D_WRFOUT_2010_CIRA_1.csv

```
tempo; nivel; P; H; TC; U; V
01/01/2010 00:00;1;99979.8;27.95874405;10.1433;9.84494;1.24745
01/01/2010 00:00;2;99166;96.05757523;9.08807;9.86604;1.24508
01/01/2010 00:00;3;98065.4;188.8219376;8.45825;10.8197;1.4931
01/01/2010 00:00;4;96678.3;306.9489365;7.80545;12.7041;1.87138
01/01/2010 00:00;5;94956.4;455.4155579;6.4776;14.9767;2.21201
01/01/2010 00:00;6;92804.2;643.821106;4.74988;17.1716;2.16673
01/01/2010 00:00;7;90222.3;874.1262512;2.84064;18.3643;1.60447
01/01/2010 00:00;8;86530.7;1213.411621;0.276154;19.1879;0.694413
01/01/2010 00:00;9;81922.8;1651.437195;-2.77612;19.9679;-0.345157
01/01/2010 00:00;10;77318.9;2108.853943;-6.0174;20.1994;-1.11373
01/01/2010 00:00;11;72717.9;2587.932007;-9.39029;19.9667;-1.37959
01/01/2010 00:00;12;66441.9;3288.956055;-14.2272;19.8842;-1.60704
01/01/2010 00:00;13;58863.5;4198.877319;-20.8894;20.9251;-2.65244
01/01/2010 00:00;14;52002;5107.909912;-26.3569;26.2345;-5.39219
01/01/2010 00:00;15;45801.2;6021.676514;-30.2229;37.0325;-9.46299
01/01/2010 00:00;16;40207.6;6946.24585;-33.0543;49.2192;-15.8124
01/01/2010 00:00;17;35171.1;7881.727051;-37.5953;58.7114;-22.5134
01/01/2010 00:00;18;30646.1;8820.92334;-44.6363;60.2117;-22.2001
01/01/2010 00:00;19;26590.2;9761.654297;-50.7635;57.8792;-17.5572
01/01/2010 00:00;20;22963.8;10712.10352;-54.5425;54.9719;-12.1234
```

Figure 10: data format for vertical meteo data

These files should be used as input of the program code made available below, to create the meteorological URBAIR input meteo_SMD.dat file and meteo_UAD.dat file.

The Python code is added in annex

Domain file

The domain file are processed using information from grid Label input data in DBF format with information about the Id, X and Y coordinates in meters (UTM). This information is obtained using one Geographic Information System (GIS) (e.g. ArcGis). Using the program code showed below, it creates the domain URBAIR input file.

The Python code is added in annex

Receptors file

The receptors file are processed using information from grid Label input data in csv format with information about the Id, X and Y coordinates. This information is obtained using one Geographic Information System (GIS). Using the program code made available below, it creates the receptor URBAIR input file.

Python program code:

```
"""
The script creates the receptor.dat input file
"""

############################################################
#MODULES
from dbfread import DBF
import mpl_toolkits.basemap.pyproj as pyproj # Import the pyproj module
import numpy as np

############################################################
```

```
WORK_PATH = "PATH”
OUTPUT_PATH = WORK_PATH + "PATH"
INPUT_PATH = WORK_PATH + "PATH”
INPUT_FILE_NAME = "FILE_NAME.csv"

Id = []
POINT_X = []
POINT_Y = []

PATH_DOMAIN = INPUT_PATH + INPUT_FILE_NAME

if PATH_DOMAIN.split("/")[-1].split(".")[-1] == "csv":
   OPEN_DOMAIN_FILE = open(PATH_DOMAIN,'r')
   lines = OPEN_DOMAIN_FILE.readlines()
   OPEN_DOMAIN_FILE.close()
   Id_DOMAIN = []
   POINT_X = []
   POINT_Y = []
   for line in lines[1:]:
      Id.append(float(line.split(";")[0]))
      POINT_X.append(float(line.split(";")[1]))
      POINT_Y.append(float(line.split(";")[2]))

max_x = float(max(POINT_X)) + abs((POINT_X[0]-POINT_X[1])/2)
min_x = float(min(POINT_X)) - abs((POINT_X[0]-POINT_X[1])/2)
max_y = float(max(POINT_Y)) + abs((POINT_X[0]-POINT_X[1])/2)
min_y = float(min(POINT_Y)) - abs((POINT_X[0]-POINT_X[1])/2)

CP_x = (max_x + min_x)/2
CP_y = (max_y + min_y)/2

print 'Coordenadas central x,y ', CP_x, CP_y

xrecgci = int(min_x - CP_x)
yrecgci = int(min_y - CP_y)
xrecgci_MAX = int(max_x - CP_x)
yrecgci_MAX = int(max_y - CP_y)
xrecgcn = int((max_x - min_x)/abs((POINT_X[0]-POINT_X[1])))
yrecgcn = int((max_y - min_y)/abs((POINT_X[0]-POINT_X[1])))
xrecgcdl = int(abs((POINT_X[0]-POINT_X[1])))/2
yrecgcdl = int(abs((POINT_X[0]-POINT_X[1])))/2

Z = 1.0 # coordinate z from the terrain surface level

POINT_X_v1 = []
POINT_Y_v1 = []

for a in range(0,len(Id)):
   POINT_X_v1.append(float(POINT_X[a])-CP_x)
   POINT_Y_v1.append(float(POINT_Y[a])-CP_y)

RECEPTORS_NUMBER = len(POINT_X_v1)

print RECEPTORS_NUMBER
```

```
f = open(OUTPUT_PATH + 'domain.dat', 'w')
for i in range(0, len(POINT_X_v1)):
   f.write(str('RE DISCCART  ')+str(POINT_X_v1[i]) + '  ' + str(POINT_Y_v1[i]) +  '\n')
f.close()
```

Emissions data

Emission data are the key input for the air quality model. In this model, we differentiate between different emission sources, each having a different data-format. As such, each sector from the ClairCity emission database requires a preprocessing tool to create the input files to be used by the URBAIR model. Therefore, a preprocessor exists for the (1) **road transport**, one for the (2) **residential and commercial sector** and one for (3) **other area emission sectors**, as shipping and industrial area. The industrial sector emissions, on ClairCity database divides into point sources and area sources, when emission values are above 100 Mg/year is considered an (4) **industry point source** and when bellow it is an area source.

For each preprocessor the input data requires a main file, which consists of the emission value for each cell. In order to create this file a GIS software is required to manipulate de emissions provided with a shapefile (".shp"). This procedure requires two shapefiles: one with the model grid and another with the polygons that contain the emission data. Using the potentialities of the GIS software it is necessary to calculate the area of each polygon, then by using the tool "intersect" it will divide the polygons according to the grid, it is then required to calculate the area of the smaller resultant polygons. The emissions will then be multiplied by a factor; the factor is obtained by dividing the smaller area by the bigger area of the polygon. Lastly, the "dissolve" tool is applied by the field "ID cell" and using the statistic type "SUM".

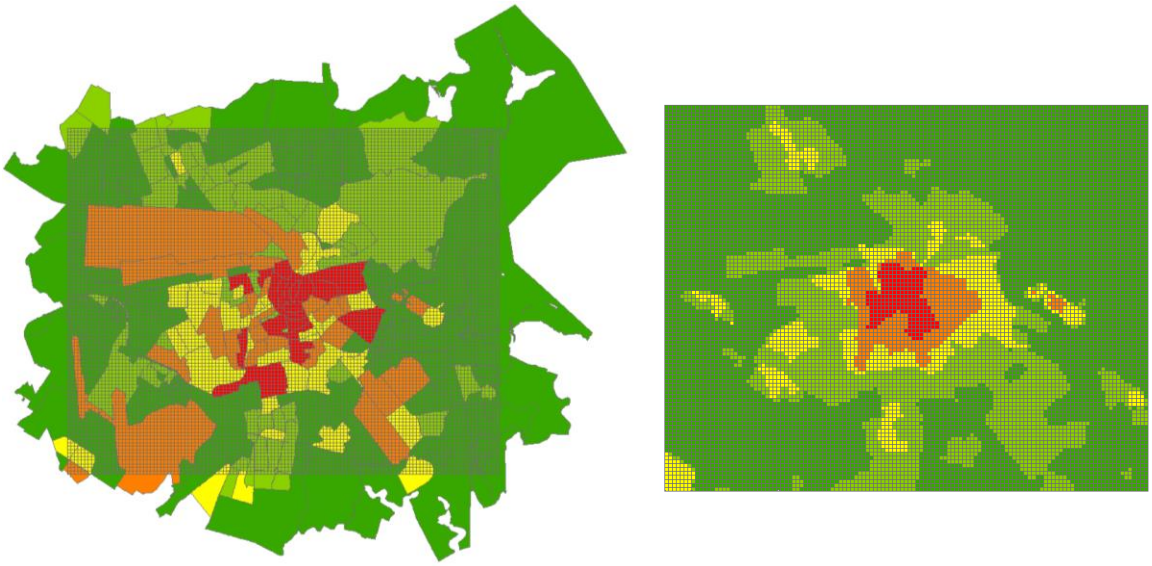
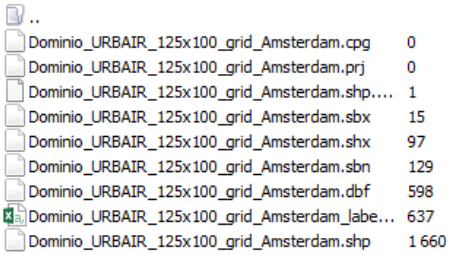


Figure 11: Representation of the residential emission shapefile overlay with the URBAIR grid (L) and Example of the output after using a GIS software (R)

1. Road Transport

The road transport sector preprocessor is used to prepare the road transport emissions to be read into the URBAIR model. It requires as input:

- a domain folder containing a shapefile of the grid and a .csv file containing the ID, X and Y coordinates of each cell (see above – “domain file”)
- an excel file containing initialization parameters values, such as engine exhaust temperature (K), atmospheric pressure (Pa), Angle of rotation of the domain in relation to the north, z coordinate of the source and the velocity (m/s)
- emission files divided by weekend and weekday for each pollutant, where each file links hourly emission values to each cell:



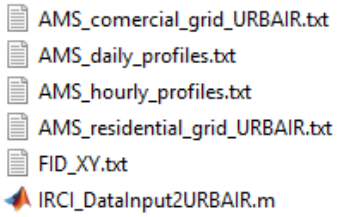
The preprocessor will read all the files and create monthly emission files and two files containing the location of the grid cells with emission values to be used as input for the URBAIR model.

The Matlab code is added in Annex

2. Residential and commercial

A preprocessor was developed for the residential and commercial to prepare the input files for the URBAIR model. It requires as input:

- annual emission values for each cell for the residential and commercial sector,
- daily and hourly profiles and
- .txt file containing the ID, X and Y coordinates for each cell:

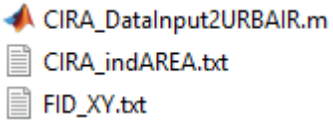


The preprocessor will read the daily profiles file and split the total annual emission for each cell per day of the year and then it will read the hourly profiles file and distribute the daily emissions per weekday and weekend for winter and summer. Finally, the last step consists on reordering the consecutive day for the whole year and create monthly files to be used as input to the URBAIR model.

The Matlab code is added in Annex

3. Other area emission sources

A preprocessor was developed for other emission sources, as the shipping sector and area industrial sources. This preprocessor requires two files as input: a .txt file containing a ID, X and Y coordinates for each cell of the domain and an emission file with annual emission values for each cell:



The preprocessor will split the annual emission values evenly by each hour and then create monthly files to be used as input to the URBAIR model.

The Matlab code is added in Annex

Post-processors

The outputs provided by the URBAIR model require some further data operations.

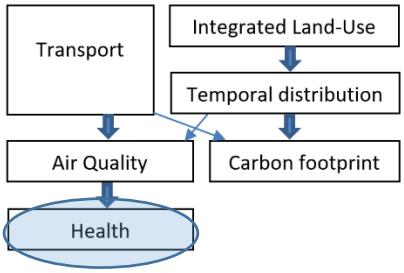
The script is divided mainly into two parts. The first part reads the data from each month and for each sector and combines it with generated spatial coordinates resulting into a .mat file that contains the concentration value for each coordinate. This file will be later used by the second part of the pre-processor, which will apply a correction factor to the results based on the concentration values measured by an air quality monitoring station.

The adjusted concentration values are then used to create concentration maps for the total and by sector (for example, transport, shipping, residential and commercial, and industrial). The script will also prepare an analysis regarding sector contribution, location of maximum values, analysis of sector concentration values and contribution for each location of the monitoring station. The post-processor is also used to provide other maps as for example, maps with concentration values above EU limits and WHO limits for each pollutant.

The URBAIR model (URBAIR.exe) is an executable file compiled to be support by Linux system.

The Matlab code is added in annex

3.5 Health impact



This module estimates health impacts, using air quality estimates and population exposure on a set of common output indices such as premature death and years of life lost.

3.5.1 Inputs

To estimate health outcomes due to exposure to air pollution of the population, gridded ambient air quality data and population density data are used to quantify the relative risk in a population based on concentration-response functions, and the estimation of mortality endpoints is based on the relation between relative risk and demographic data. The detailed input data is described below:

- Ambient air concentrations maps: the product of an air quality model, as presented in the previous section. Here we use the annual mean concentrations for NO_2, PM_{10} and $PM_{2.5}$.
- Population density maps: the number of people living in a specific area. Population maps should be described on the same spatial resolution as the ambient air concentration maps
- Concentration response functions (CRF): the estimated effect of a pollutant on the health outcome based on health studies and is typically described as change in incidence per unit concentration (UC) for those at risk. Table x shows the recommended CRFs for mortality with 95% confidence interval (CI), including the baseline concentration taken into consideration when calculating the health outcomes for each air pollutant. The baseline concentration is a concentration below which no health effects are expected.
- Crude death rates (CDR): the total number of deaths in a country per year per 1,000 people broken down by age cohort and sex. Mortality data is available from WHO website: http://apps.who.int/healthinfo/statistics/mortality/causeofdeath_query/start.php
- Population pyramids data: the number of inhabitants at country level broken down by age cohort and sex. Data published by from the United Nations (UN): https://population.un.org/wpp/Download/Standard/Population/ Tables F15-2 and F15-3
- Life expectancy at exact age: the time a person has lived broken down by age cohort and sex, and country. Data published by from the UN: https://population.un.org/wpp/Download/Standard/Mortality/ Tables F16-2 and F16-3
- Average number of years lived: broken down by age cohort and sex, and country. Data published by from the UN: https://population.un.org/wpp/Download/Standard/Mortality/ Tables F17-2 and F17-3

Table 2: concentration-response coefficients for mortality with 95% confidence interval (CI)

Pollutant	**Risk ratios for mortality**		
	Value [per 10 µg/m^3]	**Type**	**Reference**
$PM_{2.5}$	1.062 (95 % CI 1.040-1.083) No threshold	All-cause (natural) mortality in ages above 30 (ICD-10 codes A00-R99).	WHO (2013)

PM_{10}	1.04 (95% CI, 1-1.09) No threshold	All-cause (natural) mortality in ages above 30 (ICD-10 codes A00-R99).	Beelen et al. (2014)
NO_2	1.055 (95 % CI 1.031-1.08%) Threshold 10 µg/m^3	All-cause (natural) mortality in ages above 30 (ICD-10 codes A00-R99).	WHO (2013)

3.5.2 Method

The burden of disease associated with ambient air pollution is estimated by relating air concentrations to health outcomes. Hence, the relative risk (RR) at a given exposure level can be specified as follows:

$$RR = \exp\left[\ \beta * \ (C_i - C_0)\ \right] \qquad [3.5.1]$$

where, C_i is the concentration level the population is exposed to in grid cell *i*, C_0 is the baseline concentration, and β is the coefficient of the CRF. Assuming an exponential behaviour, β can be estimated based on the CRF:

$$CRF\ per\ UC = \exp(\beta * UC) \qquad [3.5.2]$$

where CRFs applied are described in Table 3.5.1 and UC = 10 ug/m^3.

The contribution of a risk factor to a disease or a death can be estimated by means of population attributable fraction (PAF). PAF is defined as the proportional reduction in population disease or mortality that would occur if exposure to a risk factor were reduced to an alternative ideal exposure scenario (e.g. concentrations equal to C_0). If the population is exposed to a single concentration level over the assessed period, PAF can be calculated from the relative risk, for every grid cell *i*, as follows:

$$PAF_i = \frac{RR-1}{RR} \qquad [3.5.3]$$

The health impact assessment focuses on estimating PD and YLL as mortality-related health outcomes. PD are deaths that occur before a person reaches an expected age, thus considered to be preventable if their cause is eliminated. The so-called expected age is usually the life expectancy for a country typically stratified by sex and age. the PD metric is estimated assuming the baseline incidence as the crude death rates by sex (s) and age (a) $CDR_{a,s}$:

$$PD = \sum_{a,s} CDR_{a,s} * PAF * \ Pop_{\ i} \qquad [3.5.4]$$

Where $Pop_{\ i}$ is the concentration at grid cell *i*.

To estimate the CDR for the population aged above 30, the following steps were taken:

1) Population pyramids data is available for 5-year age intervals from 0-4 to 100+ years old. Total population ($Tpop$) for a country is calculated by summing Population pyramids data for females and males.
2) All-cause mortality data is available for each country and sex in 5-year age interval, from 0-4 to 95+. For the current calculations, the data for the country should be collected between age interval 30-34 to 95+ interval for each sex. Because only natural causes are taken into consideration, only ICD codes A00-R99 should be included in the data. The CDR for each age interval *a* and sex *s* will be then re-calculated by:

$$CDR_{a,s} = \frac{CDR_{a,s}}{Tpop/1000} \qquad [3.5.5]$$

YLL is defined as the years of potential life loss due to premature death. It is an estimate of the average number of years that a person would have lived if the person would not have died prematurely. YLL considers the age at which deaths occur

and is greater for deaths at a younger age and lower for deaths at an older age (Murray, 1996). It gives, therefore, more nuanced information than the number of PD alone. YLL is determined by relating PD with life expectancy (*LE*) by sex and age, for every grid cell *i*.

$$YLL = \sum_{a,s,i} PD \cdot LE_{a,s} \qquad [3.5.6]$$

$LE_{a,s}$ is the average time a person is expected to live, based on the year of their birth, their current age and sex. This statistical measure is typically available from demographic datasets.

Life expectancy data is required for estimating YLL (Equation 3.5.6). Life expectancy by age and sex must be calculated based on 'Life expectancy at exact age' and 'Average number of years lived' for the population over 30 years old. The following steps were taken:

1) Life expectancy at exact age data is available every 5 years from age 5 up to 100+ per country; each table contains sex-specific data. The data for the country should be collected between age 30 and 100 for each sex.
2) Average number of years lived data is available for 5-years interval from interval 5-9 to 80-84, and an 85-100 interval per country; each table contains sex-specific data. The data for the country should be collected between age interval 30-34 to 85-100 interval for each sex.
3) Average age at death data for 5-years interval is calculated by summing the lower limit of the age interval ($ageMin_{a,s,c}$) and UN estimate for the average number of years lived ($AvgY_{a,s,c}$) at the correspondent age interval a*,* sex *s* and country *c* by applying the following WHO recommendations (WHO, 2020) :

$$ageD_{a,s,c} = AvgY_{a,s,c} + ageMin_{a,s,c} \quad [3.5.7]$$

4) Average life expectancy corresponding to the average age of death for age interval *a*, for sex *s* and country *c* ($LEcal_{i,s,c}$), is determined by the following formulation:

$$LEcal_{a,s,c} = LE_a + \frac{(ageMax_{a,s,c} - ageD_{a,s,c}) * (LE_{a,s,c} - LE_{a+1,s,c})}{ageDMax_{a+1,s,c} - ageDMax_{a,s,c}} \quad [3.5.8]$$

where higher limit of the age interval ($ageMax_{a,s,c}$), $LE_{a,s,c}$ is the Life expectancy at exact age a*,* sex *s* and country *c.*

3.5.3 Outputs

The output of the health impact assessment presented here is a value for premature deaths and years of life for the population affected by air pollution living within the study area, per emission scenario.

3.5.4 Implementation

The user first must make sure to set up an input file with health data related to the country/area of the study. An excel file is provided, where the user only must add data to the tab named "Population statistics". The data for Life expectancy at exact age, Average number of years lived, Population pyramids, and all-cause mortality, per sex and age interval ("type/sex" is signalled with green), as shown in figure below.

		type/sex	Females																				type/sex
Demographic data	Life expectancy at exact age, 2010-2015, UN data (https://esa.un.org/unpd/wpp/Download/Standard/Mortal	Age range	50	55	60	65	70	75	80	85	90	95	100										Age range
		Value	34.846372	30.213541	25.661612	21.212798	16.915404	12.907976	9.3353927	6.5083918	4.495818	3.0724361	2.1696578										Value
	Average number of years lived a(x,n), 2010-2015, UN data (https://esa.un.org/unpd/wpp/Download/Standard/Mortal	Age range	50	55	60	65	70	75	80	85	90	95	100										Age range
		Value	2.64	2.64	2.66	2.70	2.73	2.74	2.70	6.51													Value
	Average age at death (calculated) (e.g. D2+D5)	Age range	50-54	55-59	60-64	65-69	70-74	75-79	80-84	85+													Age range
		Value	52.6383	57.6378	62.6598	67.6990	72.7259	77.7423	82.6956	91.5084													Value
	YLL = Life expectancy at death (calculated e.g. D4+(D3-D8)*(D4-E4)/(E3-D3)	Age range	50-54	55-59	60-64	65-69	70-74	75-79	80-84	85+													Age range
		Value	32.36	27.76	23.22	18.76	14.53	10.61	7.44	4.07													Value
	Population pyramids, 2010-2015, UN data	Age range	20-24	25-29	30-34	35-39	40-44	45-49	50-54	55-59	60-64	65-69	70-74	75-79	80+	80-84	85-89	90-94	95-99	100+			Age range
	(https://esa.un.org/unpd/wpp/Download/Standard/Popula	Value (thousands)	274.90	291.01	340.58	399.78	424.35	394.13	398.00	369.25	348.45	327.82	281.71	253.042	...	208.87	120.20	48.84	10.93	1.50			Value (thousands)
	Total population (calculated)	Value																					Value
Baseline health data	Natural all cause mortality (2015), ICD codes A00-R99, WHO data (http://apps.who.int/healthinfo/statistics/mortality/causeofdeath_query/start.php)	Age range	50-54	55-59	60-64	65-69	70-74	75-79	80-84	85-89	90-94	95+											Age range
		Value	1090	1417	2266	3208	5906	9697	11918	8722	3637	1											Value
		Crude death rate per 1000 persons (in total population) - calculated	0.11	0.14	0.23	0.33	0.60	0.99	1.22	0.89	0.37	0.00											Crude death rate per 1000 persons (in total population) - calculated
		total crude death rate per 1000 persons (in total population) - ages 30+ - calculated																					total crude death rate per 1000 persons (in total population) - ages 30+ - calculated

Figure 12: Population statistics tab in the health data file

Currently the file is named "example_health_analysis.xlsx", please substitute "example" by e.g. area/country name as stated on the air quality modelling output files.

Edit the script example_HIA.py, and make sure the section is edited according to the case study. The variables to edited are described below:

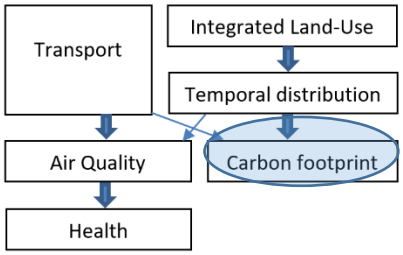
- area: area of study as stated on the health_analysis.xlsx file
- areaLong: area of study as stated on the air quality model output filenames.
- varCnc: NO2, PM10, or PM2 (=PM2.5), depending on the assessed air contaminant. The variable can only assume one of the three possibilities. This should match the air contaminant stated on the air quality model output filenames.
- aq_dir: qhere the air quality model data is stored. All the files should be stored here, the script is codded so that the data is read taking no subdirectories into account.
- health_dir: where the health data is located.
- save_dir: where to store the health impact assessment
- scenarios: name of the scenarios available to run. These are the same as stated on the air quality model output filenames.
- year: years of the future scenarios

Python program code:

To run the script, the user needs to have a python environment installed. The environment this script was created to run on Anaconda 2019.10 (Anaconda3) with Python 3.7.4. For a successful completion of the run, several packages are needed to be installed, and the following dependencies are installed: pandas 0.25.1, numpy 1.16.5, scipy 1.3.1, matplotlib 3.1.1, and seaborn 0.9.0.

The Python code is added in annex

3.6 Carbon footprint



For carbon footprint using the so-called "emission inventory" approach. The "emission inventory" approach includes generating GHG emissions activities that occur inside the city/region boundary as well as outside the city/region boundary. Considering the general goals of the activities and to delimitate the efforts in data retrieval, we select scope 2 approach considering emissions that (1) physically occur within the city/region i.e. from sources located within the city boundary and (2) occur from the use of electricity, steam, and/or heating/cooling supplied by grids which may or may not cross city/region boundaries.

As the project is finalized to produce strategies for the cities the carbon footprint evaluation must be conducted following an emission inventory approach similar to the approach followed in the Covenant of Mayors[14] and using both:

- "Standard" emission factors (emissions will be evaluated using methodologies and emission factors from 2006 IPCC Guidelines for National Greenhouse Gas Inventories[15] and specific activity level); the methodology cover all the CO_2 emissions that occur due to energy consumption within the territory of the city/region, either directly due to fuel combustion within the city/region or indirectly via fuel combustion associated with electricity and heat/cold usage within their area; the standard emission factors are based on the carbon content of each fuel, like in national greenhouse gas inventories in the context of the UNFCCC and the Kyoto protocol; in the standard approach, the CO_2 emissions from the sustainable use of biomass/biofuels, as well as emissions of certified green electricity, are considered to be zero; emission are reported as:
    - CO_2 only emissions, the most important greenhouse gas,
    - CO_2 equivalent emissions, including calculation of the emissions of CH_4 and N_2O with emission factor from 2006 IPCC Guidelines for National Greenhouse Gas Inventories[16] and reported as CO_2 using the Global Warming Potential (GWP) with 100 years - time horizon[17]:

$$1 \text{ Mg } CO_2 = 1 \text{ Mg } CO_2\text{-eq}$$

$$1 \text{ t CH4} = 21 \text{ Mg CO2-eq}$$

$$1 \text{ t N2O} = 310 \text{ Mg CO2-eq;}$$

- LCA (Life Cycle Assessment) emission factors, which take into consideration the overall life cycle of the energy carrier; this approach includes not only the emissions of the final combustion, but also all emissions of the supply chain; it includes emissions from exploitation, transport and processing (e.g. refinery) steps in addition to the final combustion; this hence includes also emissions that take place outside the location where the fuel is used; in this approach, the GHG emissions from the use of biomass/biofuels, as well as emissions of certified green electricity, are higher than zero; in the case of this approach, other greenhouse gases than CO_2 may play an important role, therefore the LCA approach will report emissions as CO_2 equivalent; as a default will be used the LCA emission factors given in Covenant of Mayors guidelines, based on JRC European Reference Life Cycle

[14] Covenant of Mayors (2010), How to develop a Sustainable Energy Action Plan (SEAP) – Guidebook Part II, Baseline emissions inventory
[15] 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 2 Energy
[16] 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 2 Energy
[17] IPCC, 1995. Contribution of Working Group I to the Second Assessment of the Intergovernmental Panel on Climate Change

Database; specific national emission factors will be investigates.
Eventual local electricity production is not included in the model while district heating is included where data are available.

3.6.1 Inputs

The input are energy vectors (fuels and electricity) consumptions in the domain taken into consideration, from the residential, industrial and commercial sector and transport, in totals per year. These are the outputs from the Land-use module and the transport module discussed earlier.

3.6.2 Method

The Carbon Footprint tool evaluates emissions at most detailed administrative territorial units' level and uses emission factors from Covenant of Mayors guidelines. Emissions are calculated as:

$$E_k = A_{ij} F_{ik}$$

where:

- A_{ij} is the indicator of the activity i in the territorial unit j
- F_{ik} is the emission factor for different Carbon footprint indicators k for activity i (expressed in grams per unit of activity);
- k is the carbon footprint indicator used: CO_2, CO_{2eq}, $CO_{2eq,LCA}$

In Table 3 standard emission factors are reported for different energy vectors while in **Error! Reference source not found.** electricity consumptions national emission factors[18] are reported to use for CO_2 indirect emissions from electricity consumptions. EFs from JRC are used to have a comparable set of data for all the cities.

Table 3: CO2 Emission Factors

Fuel	Standard Emission Factors[19] [Mg CO2/MWh]	Standard Emission Factors[20] [Mg CO2eq/MWh]	LCA Emission Factors[21] [Mg CO2-eq/MWh]
Motor Gasoline	0.249	0.250	0.299
Gas oil, diesel	0.267	0.268	0.305
Residual Fuel Oil	0.279	0.279	0.310
Anthracite	0.354	0.356	0.393
Other Bituminous Coal	0.341	0.342	0.380
Sub-Bituminous Coal	0.346	0.348	0.385
Lignite	0.364	0.365	0.375

[18] Koffi, Brigitte; Cerutti, Alessandro; Duerr, Marlene; Iancu, Andreea; Kona, Albana; Janssens-Maenhout, Greet (2017): CoM Default Emission Factors for the Member States of the European Union - Version 2017, European Commission, Joint Research Centre (JRC)
[19] 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 2 Energy
[20] 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 2 Energy
[21] ELCD (2015), European Reference Life Cycle Database (ELCD), Release 3.2. LCA data sets of key energy carriers, materials, waste and transport services of European scope

Natural Gas	0.202	0.202	0.237
LPG	0.227	0.227	0.281
Municipal Wastes (non-biomass fraction)	0.330	0.337	0.330
Municipal Wastes (biomass fraction)	0	0.007	0.106
Industrial Wastes	0.515	0.522	0.522
Wood°	0 – 0.403	0.007 – 0.410	0.017 – 0.416
Plant oil°	0 – 0.287	0.001 – 0.302	0.182 – 0.484
Biodiesel°	0 – 0.255	0.001 – 0.256	0.156 – 0.411
Biogas/Greengas°	0-0.197	0-0.197	0,087-0.284
Solar°°	0	0	0.04
Geothermal°°	0	0	0,05
Hydroelectric°°°	0	0	0,006
Wind°°°	0	0	0,01

° Lower value if fuel meet carbon neutrality criteria, higher otherwise

°° Default EC/JRC Emission factors[22]

°°° Default EC/JRC Emission factors for local electricity production[9]

Table 4: CO2 National Electricity Emission Factors

Fuel	Standard Emission Factors [Mg CO2/MWh]	Standard Emission Factors [Mg CO2eq/MWh]	LCA Emission Factors [Mg CO2-eq/MWh]
United Kingston	0,515	0,517	0,589
Netherland	0,429	0,430	0,486
Slovenia	0,399	0,401	0,424
Poland	1,013	1,017	1,09
Italy	0,343	0,344	0,424
Portugal	0,314	0,316	0,368
Austria	0,17	0,17	0,211

[22] Koffi, Brigitte; Cerutti, Alessandro; Duerr, Marlene; Iancu, Andreea; Kona, Albana; Janssens-Maenhout, Greet (2017): CoM Default Emission Factors for the Member States of the European Union - Version 2017, European Commission, Joint Research Centre (JRC)

Belgium	0,198	0,199	0,239
Bulgaria	0,791	0,795	0,824
Croatia	0,204	0,205	0,228
Cyprus	0,707	0,709	0,817
Czech Republic Estonia	0,783	0,787	0,85
Denmark	0,331	0,333	0,38
Estonia	1,977	1,986	2,017
Finland	0,155	0,156	0,206
France	0,082	0,083	0,093
Germany	0,587	0,589	0,658
Greece	0,757	0,76	0,81
Hungary	0,254	0,255	0,297
Ireland	0,464	0,465	0,523
Italy	0,343	0,344	0,424
Latvia	0,121	0,121	0,183
Lithuania	0,096	0,096	0,128
Luxembourg	0,091	0,091	0,108
Malta	0,871	0,874	1,002
Netherlands	0,429	0,43	0,486
Poland	1,013	1,017	1,09
Portugal	0,314	0,316	0,368
Romania	0,502	0,504	0,532
Slovak Republic	0,199	0,199	0,241
Slovenia	0,399	0,401	0,424
Spain	0,297	0,298	0,343
Sweden	0,015	0,016	0,038

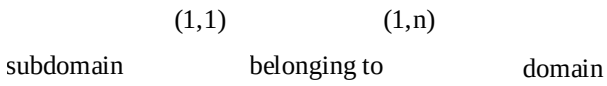
United-Kingdom	0,515	0,517	0,589
EU-28	0,391	0,393	0,444

3.6.3 Implementation

The data management procedures for baseline are developed in MS Access environment and are similar to the procedure for IRCI sector.

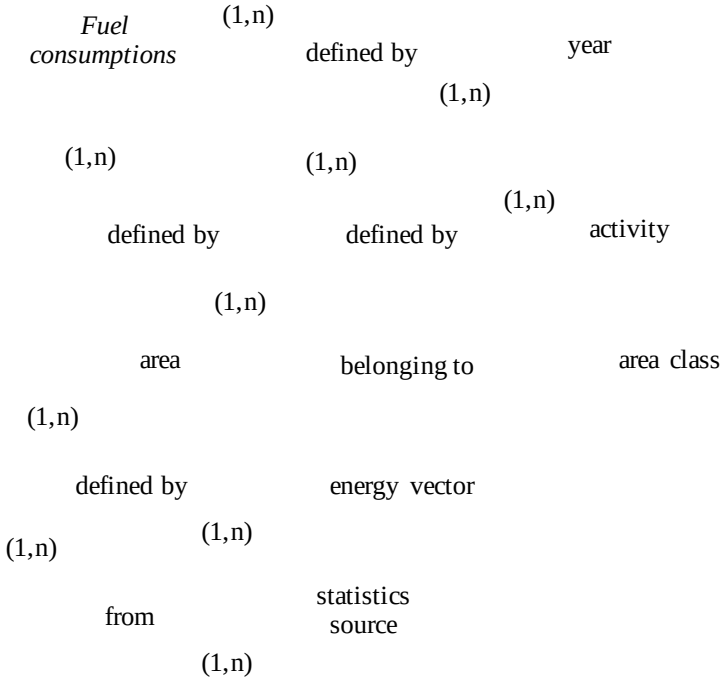
Data structure used for the estimate of current and future carbon footprints is described below.

Carbon footprints are evaluated at level of subdomains of the specific city/region domain. In the database, a geographical structure is defined as:

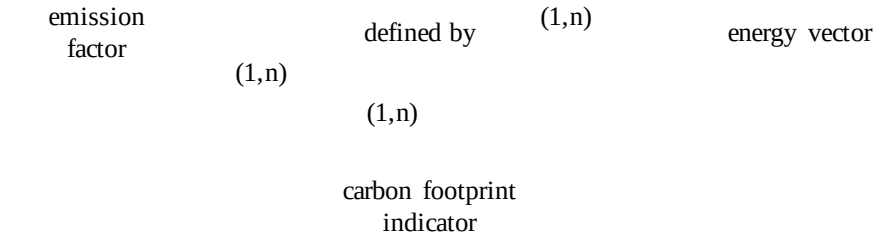


This structure is applied to the definition of subdomains inside the area of model.

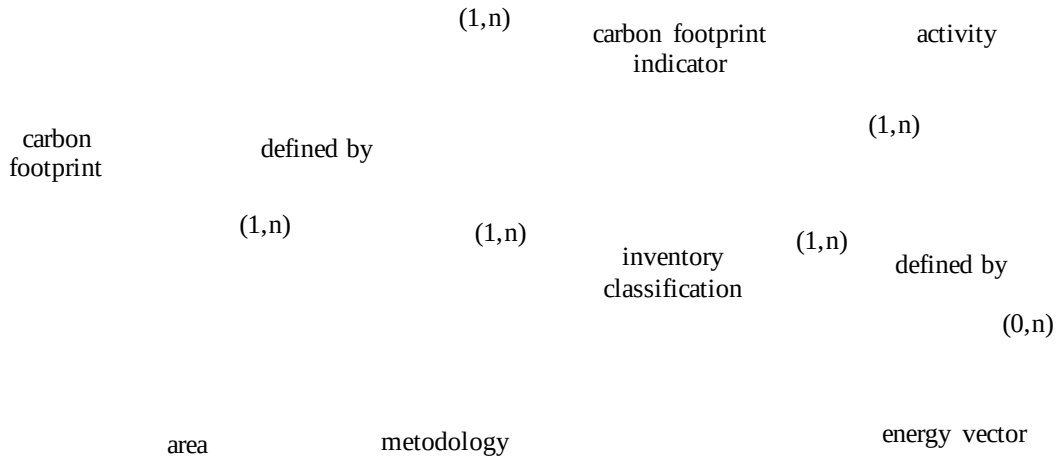
Energy vector consumptions data are related to an activity for an area, and are time (year) dependent. The fuel consumptions data are an ***input*** of the model.



Furthermore, the emission factors are defined, whose conceptual scheme, in which the links with activities, energy source and pollutants are highlighted:



The carbon footprint levels are expressed for each activity in the activity nomenclature, for each zone or point, per year and, where relevant, by energy source:



For Carbon footprints projection the Techne Consulting ***Projection*** model is used.

4 Annex

4.1 Temporal module – Python code

```
# Temperature data pre-processing, to find the location of the missing values
# The goals are:
# 1. To find the missing values' location,
# 2. To identify how many batch of the missing values, and
# 3. To identify how many row of each batch of the missing values
# Input : The pre-processed temperature data in .csv form.
# Output :
# 1. Missing values' location
# 2. Number of the missing values' batch
# 3. Number of the rows of each batch
#
# We can use this output information to decide what kind of technique to be used
# for batch that only contain 1,2 or 3 mv, we applied linear interpolation
# for batch contain longer row mv, we applied machine learning technique
################################################################################
import numpy as np
import pandas as pd
import time
# Path to .csv file location
dat = pd.read_csv('../5.Liguaria/LiguFinal/hasil/hourly-liguria15.csv', sep=",")
mv =  dat['temp'].isnull().sum()
print('Number of missing values:',mv)
Number of missing values: 30
In [2]:
## Filtering the dataframe, select only mv row
# buffer format [row_number, values, hourly_form]
start_time = time.time()
#buffer for mv
buffer = list()
#helper variables
ff, gg, ind = 0, 0, 0
seq, idx = [], []
# Part 1 : buffer preparation
for i in range (0,8760):
   for ii in range (0, 24): # ini for iterate jam
      cc = pd.isnull(dat["temp"])[i]
      if (cc == True):
         if (i%24 == ii): # brarti jam 0, dst
            dd = ( i, dat['temp'][i],ii)
            ee = list(dd)
            buffer.append(ee)
#part 2 : loop for Missing values Eyes
for n in range (0, mv):
   if (buffer[n][0] - buffer[n-1][0] == 1):
```

```
        ff +=1
    else:
        seq.append(ff)
        idx.append(buffer[n][0])
        ff = 1
        gg +=1
        ind +=1
seq.append(ff)
elapsed_time = time.time() - start_time
print ("Time elapsed: {} seconds".format(elapsed_time))
print()
print('Number of batch         :', len(seq)-1)
print('Number of data each batch:', seq[1:])
print('Start from (Row Number)  :',idx)
```

```
Time elapsed: 21.069633960723877 seconds
Number of batch         : 5
Number of data each batch: [3, 8, 2, 3, 14]
Start from (Row Number)  : [537, 2851, 4146, 7078, 8341]
```

In [3]:

```
print(buffer)
```

```
[[537, nan, 9], [538, nan, 10], [539, nan, 11], [2851, nan, 19], [2852, nan, 20], [2853, nan, 21], [2854, nan, 22], [2855, nan, 23], [2856, nan, 0], [2857, nan, 1], [2858, nan, 2], [4146, nan, 18], [4147, nan, 19], [7078, nan, 22], [7079, nan, 23], [7080, nan, 0], [8341, nan, 13], [8342, nan, 14], [8343, nan, 15], [8344, nan, 16], [8345, nan, 17], [8346, nan, 18], [8347, nan, 19], [8348, nan, 20], [8349, nan, 21], [8350, nan, 22], [8351, nan, 23], [8352, nan, 0], [8353, nan, 1], [8354, nan, 2]]
```

In [ ]:

```
# Temperature data pre-processing, to check the missing values.
#
# Input : Temperature data in .sqlite form.
# Output : 1. Hourly temperature in .csv form.
#         2. Daily temperature in .csv form.

import sqlite3
import pandas as pd
import datetime
import time

## Path to .sqlite file location
conn = sqlite3.connect('../data/db/liguria.sqlite')
c = conn.cursor()
c.execute('PRAGMA TABLE_INFO({})'.format('liguria'))

print('Preview of Data Parameters from database:')
print(pd.DataFrame(c.fetchall()))
```

```
Preview of Data Parameters from database:
  0       1          2 3    4 5
0 0   ogc_fid      INTEGER  0  None  1
```

```
1  1  datetime  VARCHAR(255)  0  None  0
2  2    valore        FLOAT  0  None  0
```

In [2]:

```
# Data Preview
query = 'SELECT * FROM liguria'
c.execute(query)
nox = c.fetchall()
df2 = pd.DataFrame(nox)
print('Preview of values from database:')
print (df2.head())
```

```
Preview of values from database:
  0                 1    2
0  1  2015-01-01 00:00:00 -5.5
1  2  2015-01-01 01:00:00 -5.4
2  3  2015-01-01 02:00:00 -5.2
3  4  2015-01-01 03:00:00 -5.4
4  5  2015-01-01 04:00:00 -5.3
```

In [3]:

```
##Generation of hourly resolution data
import sqlite3
import pandas as pd
import requests
import json
import time
import datetime

start_time = time.time()
## To select specific date, these parameters below can be adjusted
# start week
mingst = 1
# end week
mingak = 53
# Year
ath = 2015
bth = ath
# Variable and initial values needed in for loop
perjam = list()
lines = 0
for bul in range(12):
    abu = bul+1
    bbu = abu
    # number of day within months
    if  abu == 1:
        jbul = 31
    elif abu == 2:
        jbul = 28
    elif abu == 3:
```

```
    jbul = 31
elif abu == 4:
    jbul = 30
elif abu == 5:
    jbul = 31
elif abu == 6:
    jbul = 30
elif abu == 7:
    jbul = 31
elif abu == 8:
    jbul = 31
elif abu == 9:
    jbul = 30
elif abu == 10:
    jbul = 31
elif abu == 11:
    jbul = 30
elif abu == 12:
    jbul = 31
for tgl in range(jbul): ## days iteratoe
    atg = tgl+1
    btg = atg
    mingx = datetime.date(ath, abu, atg).isocalendar()[1]
    if mingx >= mingst and mingx <= mingak:
        for jam in range(24): ## hours iterator
            aja = jam
            bja=jam+1
            if jam >= 23:
                bja = 0
                btg = atg+1
                if btg > jbul:
                    btg = 1
                    bbu = abu+1
                    if bbu > 12:
                        bbu = 1
                        bth = ath+1
            aw = str(datetime.datetime(ath, abu, atg, aja, 00, 00))
            ak = str(datetime.datetime(bth, bbu, btg, bja, 00, 00))
            cc = "SELECT avg(valore) FROM liguria WHERE datetime between \"%s\" and \"%s\" " % (aw,ak)
            sql = sqlite3.connect('../data/db/liguria.sqlite')
            c = sql.cursor()
            c.execute(cc)
            roww = c.fetchone()
            sql.close()
            dff = list(roww)
            dff.insert(0, aw)
            perjam.append(dff)
```

```
            lines += 1
print("")
print("LAst datetime : "+str(aw)+", Num_data: "+str(len(perjam)))
elapsed_time = time.time() - start_time
print ("Time elapsed: {} seconds".format(elapsed_time))
print ("Read {} lines".format(lines))
```

```
LAst datetime : 2015-12-31 23:00:00, Num_data: 8760
Time elapsed: 8.72475290298462 seconds
Read 8760 lines
```

In [4]:

```
# Write hourly data to .csv form.
dataMenit = pd.DataFrame(perjam)
dataMenit.to_csv('../hasil/hourly-liguria15Xx.csv', index=False, header=[ ' date','temp'])
# Preview Missing Values
print('Number of missing valus:', dataMenit[1].isnull().sum())
# Preview last 5 row data
print('Preview last 5 rows')
print(dataMenit.tail())
```

```
Number of missing values: 30
Preview last 5 rows
                    0     1
8755  2015-12-31 19:00:00  0.05
8756  2015-12-31 20:00:00  0.00
8757  2015-12-31 21:00:00 -0.05
8758  2015-12-31 22:00:00 -0.20
8759  2015-12-31 23:00:00 -0.10
```

In [5]:

```
##Generation of daily resolution data
import sqlite3
import pandas as pd
import time
import datetime
start_time = time.time()
ath = 2015
bth = ath
perjam = list()
lines = 0
for bul in range(0, 12):
    abu = bul+1
    bbu = abu
    if  abu == 1:
        jbul = 31
    elif abu == 2:
        jbul = 28
    elif abu == 3:
        jbul = 31
    elif abu == 4:
```

```
        jbul = 30
    elif abu == 5:
        jbul = 31
    elif abu == 6:
        jbul = 30
    elif abu == 7:
        jbul = 31
    elif abu == 8:
        jbul = 31
    elif abu == 9:
        jbul = 30
    elif abu == 10:
        jbul = 31
    elif abu == 11:
        jbul = 30
    elif abu == 12:
        jbul = 31
    for tgl in range(jbul): ## date
        atg = tgl+1
        btg = atg+1
        if btg > jbul:
            btg = 1
            bbu = abu+1 ## change month
            if bbu > 12:
                bbu = 1
                bth = ath+1 ## change year
        aw = str(datetime.datetime(ath, abu, atg, 00, 00, 00))
        ak = str(datetime.datetime(bth, bbu, btg, 00, 00, 00))
        cc = "SELECT avg(valore) FROM liguria WHERE datetime between \"%s\" and \"%s\" " % (aw,ak)
        sql = sqlite3.connect('../data/db/liguria.sqlite')
        c = sql.cursor()
        c.execute(cc)
        roww = c.fetchone()
        sql.close()
        dff = list(roww)
        dff.insert(0, aw)
        perjam.append(dff)
        lines += 1
print("")
print("LAst date : "+str(aw)+", Num_data: "+str(len(perjam)))
elapsed_time = time.time() - start_time
print ("Time elapsed: {} seconds".format(elapsed_time))
print ("Read {} lines".format(lines))
```

```
LAst date : 2015-12-31 00:00:00, Num_data: 365
Time elapsed: 0.3376741409301758 seconds
Read 365 lines
```

In [6]:

```
# Write daily data to .csv form.
dataMenit = pd.DataFrame(perjam)
dataMenit.to_csv('../hasil/daily-liguria15Xx.csv', index=False, header=[ ' date','temp'])
# Preview last 5 row data
print('Preview last 5 rows')
print(dataMenit.tail())
```

```
Preview last 5 rows
                    0      1
360  2015-12-27 00:00:00  6.9240
361  2015-12-28 00:00:00  6.7360
362  2015-12-29 00:00:00  6.4120
363  2015-12-30 00:00:00  4.9800
364  2015-12-31 00:00:00  0.8125
```

In [ ]:

```
#####################################################################
# Table of recaps
# Input : data from techne in .sqlite form.
# Output : Total emission of PM10 and NoX in various category #
import sqlite3
import pandas as pd
#Path to .sqlite file location
conn = sqlite3.connect('../data/db/area_emissions_liguria.sqlite')
c = conn.cursor()
c.execute('PRAGMA TABLE_INFO({})'.format('area_emissions_liguria')) # hoba table namen
print('Data Information:')
print()
print(pd.DataFrame(c.fetchall()))
```

```
Data Information:
  0           1           2 3    4 5
0 0      ogc_fid      INTEGER 0  None 1
1 1         year      INTEGER 0  None 0
2 2         city  VARCHAR(255) 0  None 0
3 3         zone       BIGINT 0  None 0
4 4   codvariable  VARCHAR(255) 0  None 0
5 5  namevariable  VARCHAR(255) 0  None 0
6 6    pollutant  VARCHAR(255) 0  None 0
7 7     emission        FLOAT 0  None 0
8 8         unit  VARCHAR(255) 0  None 0
```

In [2]:

```
## Preview variables name
query = 'SELECT distinct(namevariable) FROM area_emissions_liguria'
c.execute(query)
nox = c.fetchall()
df2 = pd.DataFrame(nox)
print('Preview of Data Parameters from database:')
#df2.head()
```

nox

Preview of Data Parameters from database:

Out[2]:

```
[('Commercial Combustion plants (boilers) (3220 - LPG)',),
 ('Commercial Combustion plants (boilers) (3260 - Gas/Diesel Oil )',),
 ('Commercial Combustion plants (boilers) (4100 - Natural gas)',),
 ('Residential Combustion plants (boilers) (3220 - LPG)',),
 ('Residential Combustion plants (boilers) (3260 - Gas/Diesel Oil )',),
 ('Residential Combustion plants (boilers) (4100 - Natural gas)',),
 ('Residential Fireplaces (5541 - Solid biomass)',),
 ('Residential Advanced Fireplaces (5541 - Solid biomass)',),
 ('Residential Conventional Stoves (5541 - Solid biomass)',),
 ('Residential Advanced Stoves (5541 - Solid biomass)',)]
```

In [3]:

```
## Selection on PM10
#  'namevariable' can be changed, depends on the parameters above
query = 'SELECT zone, codvariable, sum(emission) FROM area_emissions_liguria WHERE pollutant LIKE"pm10" and
namevariable LIKE"Commercial Combustion plants (boilers) (4100 - Natural gas%"'
c.execute(query)
nox = c.fetchall()
df2 = pd.DataFrame(nox)
df2.head()
```

Out[3]:

	0	1	2
0	100010000053	020131M1	0.034995

In [4]:

```
## Selection on NOX
#  'namevariable' can be changed, depends on the parameters above
query = 'SELECT zone, codvariable, sum(emission) FROM area_emissions_liguria WHERE pollutant LIKE"nox" and
namevariable LIKE"Commercial Combustion plants (boilers) (4100 - Natural gas%" '
c.execute(query)
nox = c.fetchall()
df2 = pd.DataFrame(nox)
df2.head()
```

Out[4]:

	0	1	2
0	100010000053	020131M1	7.348859

In [ ]:

4.2 Transport module – Matlab code

DFAST.m

```
function [flows,originFlows] = DFAST_v2(odmatrix,nodes,links,theta,final_travel_costs)
%Method of successive averages for calculating stochastic user equilibrium
%For fixed ordering of nodes based on free flow conditions

%SYNTAX
%   [flows] = DFAST(odmatrix,nodes,links)
%
%DESCRIPTION
%   returns the flow on each link in the stochastic user equilibrium as
%   calculated by the method of successive averages
%
%INPUTS
%   odmatrix: static origin/destination matrix
%   nodes: table with all the nodes in the network.
%   links: table with all the links in the network
%   theta: stochastic distribution parameter (related to the value of time)

%setup the output figure

%initilization
totLinks = size(links.toNode,1);
totNodes = size(nodes.x,1);
strN = links.fromNode;
endN = links.toNode;

%initialize the travel cost
%note that the total flow on a link is the sum of the origin based flow
%on that link
alpha = 0.15;
beta = 4;
travelCosts =
calculateCostBPR(alpha,beta,zeros(size(links.freeSpeed)),links.length,links.freeSpeed,links.capacity);

%Compute dijkstra costs and store them in a cost-ordered structure
% netCostMatrix=sparse(strN,endN,travelCosts,totNodes,totNodes);
[rp,ci,ai]=sparse_to_csr(strN,endN,1:length(links.length),max(nodes.id)); %
a = travelCosts(ai);
orderN = zeros(totNodes,length(odmatrix));
for origin = 1:size(odmatrix,1)
   if sum(odmatrix(origin,:)) > 0
      distance = dijkstra_v2(rp,ci,a,origin);
      costTree = [nodes.id,distance];
      %order costTree on column 2
      costTree = sortrows(costTree,2);
      orderN(:,origin) = costTree(:,1);
   end
end
```

```
%Initialize the iteration numbering
it = 0;

%initialize the gap function
gap = inf;

%MAIN LOOP: iterate until convergence is reached or maximum number of
%iterations is reached

%Compute new flows via the implicit routing scheme of Dail (1971)
originFlows = Dial_F(odmatrix,nodes,links,orderN,final_travel_costs,theta);

%Return the total flow for every linnk (sum over all origines)
flows = sum(originFlows,2);

end
```

calculateCostBPR.m

```
function costs = calculateCostBPR(alpha,beta,flows,lengths,speeds,caps)
%Calculates the costs on a network according to the BPR curve
%
%SYNTAX
%   [costs] = calculateCostBPR(alpha,beta,flows,lengths,speeds,caps)
%
%DESCRIPTION
%   returns the costs on a network according to the BPR curve. Note that
%   the free flow travel time is calculated within the function.
%
%INPUTS
%   alpha: parameter that captures the additional travel time at capacity
%   beta: parameter that handles the slope of the increase in travel time
%   flows: total flow over each link
%   lengths: length of each link
%   speeds: maximum speed of each link
%   caps: capacity of each link

costs = lengths./speeds*60.*(1+alpha.*(flows./caps).^beta);
```

sparse_to_csr.m

```
function [rp ci ai m invm ncol]=sparse_to_csr(A,varargin)
% SPARSE_TO_CSR Convert a sparse matrix into compressed row storage arrays
%
% [rp ci ai] = sparse_to_csr(A) returns the row pointer (rp), column index
% (ci) and value index (ai) arrays of a compressed sparse representation of
% the matrix A.
%
% [rp ci ai] = sparse_to_csr(i,j,v,n) returns a csr representation of the
% index sets i,j,v with n rows.
%
% Example:
%   A=sparse(6,6); A(1,1)=5; A(1,5)=2; A(2,3)=-1; A(4,1)=1; A(5,6)=1;
%   [rp ci ai]=sparse_to_csr(A)
%
% See also CSR_TO_SPARSE, SPARSE

% David F. Gleich
% Copyright, Stanford University, 2008-2009

% History
% 2008-04-07: Initial version
% 2008-04-24: Added triple array input
% 2009-05-01: Added ncol output
% 2009-05-15: Fixed triplet input

% narginchk(1, 5, nargin, 'struct')
retc = nargout>1; reta = nargout>2;

if nargin>1
   if nargin>4, ncol = varargin{4}; end
   nzi = A; nzj = varargin{1};
   if reta && length(varargin) > 2, nzv = varargin{2}; end
   if nargin<4, n=max(nzi); else n=varargin{3}; end
   nz = length(A);
   if length(nzi) ~= length(nzj), error('gaimc:invalidInput',...
         'length of nzi (%i) not equal to length of nzj (%i)', nz, ...
         length(nzj));
   end
   if reta && length(varargin) < 3, error('gaimc:invalidInput',...
         'no value array passed for triplet input, see usage');
   end
   if ~isscalar(n), error('gaimc:invalidInput',...
         ['the 4th input to sparse_to_csr with triple input was not ' ...
          'a scalar']);
   end
   if nargin < 5, ncol = max(nzj);
   elseif ~isscalar(ncol), error('gaimc:invalidInput',...
         ['the 5th input to sparse_to_csr with triple input was not ' ...
          'a scalar']);
```

```
   end
else
   n = size(A,1); nz = nnz(A); ncol = size(A,2);
   retc = nargout>1; reta = nargout>2;
   if reta,     [nzi nzj nzv] = find(A);
   else         [nzi nzj] = find(A);
   end
end

m = zeros(nz,1);
invm = zeros(nz,1);

if retc, ci = zeros(nz,1); end
if reta, ai = zeros(nz,1); end
rp = zeros(n+1,1);
for i=1:nz
   rp(nzi(i)+1)=rp(nzi(i)+1)+1;
end
rp=cumsum(rp);
if ~retc && ~reta, rp=rp+1; return; end
for i=1:nz
   if reta, ai(rp(nzi(i))+1)=nzv(i); m(rp(nzi(i))+1)=i; invm(i)=rp(nzi(i))+1; end %map(rp(nzi(i))+1)=i;
   ci(rp(nzi(i))+1)=nzj(i);
   rp(nzi(i))=rp(nzi(i))+1;
end
for i=n:-1:1
   rp(i+1)=rp(i);
end
rp(1)=0;
rp=rp+1;
```

dijkstra_v2.m

```
function [d,pred,list]=dijkstra_v2(rp,ci,ai,u)
% DIJKSTRA Compute shortest paths using Dijkstra's algorithm
%
% d=dijkstra(A,u) computes the shortest path from vertex u to all nodes
% reachable from vertex u using Dijkstra's algorithm for the problem.
% The graph is given by the weighted sparse matrix A, where A(i,j) is
% the distance between vertex i and j.  In the output vector d,
% the entry d(v) is the minimum distance between vertex u and vertex v.
% A vertex w unreachable from u has d(w)=Inf.
%
% [d pred]=dijkstra(A,u) also returns the predecessor tree to generate
% the actual shorest paths.  In the predecessor tree pred(v) is the
% vertex preceeding v in the shortest path and pred(u)=0.  Any
% unreachable vertex has pred(w)=0 as well.
%
% If your network is unweighted, then use bfs instead.
%
% See also BFS
%
% Example:
%   % Find the minimum travel time between Los Angeles (LAX) and
%   % Rochester Minnesota (RST).
%   load_gaimc_graph('airports')
%   A = -A; % fix funny encoding of airport data
%   lax=247; rst=355;
%   [d pred] = dijkstra(A,lax);
%   fprintf('Minimum time: %g\n',d(rst));
%   % Print the path
%   fprintf('Path:\n');
%   path =[]; u = rst; while (u ~= lax) path=[u path]; u=pred(u); end
%   fprintf('%s',labels{lax});
%   for i=path; fprintf(' --> %s', labels{i}); end, fprintf('\n');

% David F. Gleich
% Copyright, Stanford University, 2008-2009

% History
% 2008-04-09: Initial coding
% 2009-05-15: Documentation


% [rp,ci,ai]=sparse_to_csr(in,out,val,max(max([in;out]))); check=1;
%
% if check && any(ai)<0, error('gaimc:dijkstra', ...
%       'dijkstra''s algorithm cannot handle negative edge weights.'); end

n=length(rp)-1;
d=Inf(n,1); T=zeros(n,1); L=zeros(n,1);
pred=zeros(1,length(rp)-1);
```

```
list=zeros(length(rp)-1,1);
xout=true(length(rp)-1,1);
l_it=1;

n=1; T(n)=u; L(u)=n; % oops, n is now the size of the heap

% enter the main dijkstra loop
d(u) = 0;
while n>0
   v=T(1); ntop=T(n); T(1)=ntop; L(ntop)=1; n=n-1; % pop the head off the heap
   list(l_it)=v;
   xout(v)=false;
   l_it=l_it+1;

   k=1; kt=ntop;              % move element T(1) down the heap
   while 1,
      i=2*k;
      if i>n, break; end       % end of heap
      if i==n, it=T(i);        % only one child, so skip
      else                     % pick the smallest child
         lc=T(i); rc=T(i+1); it=lc;
         if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
      end
      if d(kt)<d(it), break;    % at correct place, so end
      else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
      end
   end                         % end heap down

   % for each vertex adjacent to v, relax it
   for ei=rp(v):rp(v+1)-1         % ei is the edge index
      w=ci(ei); ew=ai(ei);        % w is the target, ew is the edge weight
      % relax edge (v,w,ew)
      if d(w)>d(v)+ew
         d(w)=d(v)+ew; pred(w)=v;
         % check if w is in the heap
         k=L(w); onlyup=0;
         if k==0
            % element not in heap, only move the element up the heap
            n=n+1; T(n)=w; L(w)=n; k=n; kt=w; onlyup=1;
         else kt=T(k);
         end
         % update the heap, move the element down in the heap
         while 1 && ~onlyup,
            i=2*k;
            if i>n, break; end       % end of heap
            if i==n, it=T(i);        % only one child, so skip
            else                     % pick the smallest child
               lc=T(i); rc=T(i+1); it=lc;
               if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
            end
            if d(kt)<d(it), break;    % at correct place, so end
```

```
            else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
            end
          end
          % move the element up the heap
          j=k; tj=T(j);
          while j>1,                  % j==1 => element at top of heap
            j2=floor(j/2); tj2=T(j2);    % parent element
            if d(tj2)<d(tj), break;      % parent is smaller, so done
            else                    % parent is larger, so swap
              T(j2)=tj; L(tj)=j2; T(j)=tj2; L(tj2)=j; j=j2;
            end
          end
        end
      end
end
list(l_it:end)=find(xout);
% figure;plot(N(1:sl_it));
end
```

Dial_F.m

```
function [originFlows] = Dial_F(odmatrix, nodes, links, orderN,travelCosts, theta)
%Dails method for calculating a stochastic network loading
%
%
%SYNTAX
%   [originFlows] = Dial(ODmatrix, nodes, links, costs, theta)
%
%DESCRIPTION
%   Returns the flow on the network as a bush for each origin.
%   Flow is assigned to each link of the network according to Dial's
%   methods.
%
%INPUTS
%   ODmatrix: static origin destination matrix
%   nodes: list of all the nodes in the network.
%   Each entry of the list represents one node. Each node is a structure that
%   has at least a node ID and an x and y coordinate of the node
%   links: list of all the links in the network
%   Each entry of the list represents one link. Each link is a structure that
%   has at least a link ID and an upstream and downstream node.

totNodes = size(nodes.x,1);
maxNodes = max(nodes.id);
totLinks = size(links.toNode,1);

strN=links.fromNode;
endN=links.toNode;

originFlows = zeros(totLinks,size(odmatrix,1));

% linkPosition=sparse(strN,endN,1:length(travelCosts),maxNodes,maxNodes);
[rp_,ci_,ai_]=sparse_to_csr(endN,strN,1:length(links.length),max(nodes.id)); %
% [rp,ci,ai]=sparse_to_csr(strN,endN,1:length(links.length),max(nodes.id)); %

for origin = 1:size(odmatrix,1)
   if sum(odmatrix(origin,:)) > 0
      nodeWeights = zeros(maxNodes,1);
      linkWeights = zeros(totLinks,1);
      nodeFlows = zeros(maxNodes,1);
      nodeWeights(origin) = 1;

      costTree = orderN(:,origin);

      %visit nodes in costTree starting from k=2
      for k = 2:totNodes
         nodeInd = costTree(k,1);
         %get which links enter node k
         nodeWeights(nodeInd) = 0; %?
         for ei = rp_(nodeInd):rp_(nodeInd+1)-1
```

```
            l=ai_(ei);
            upNodeInd = strN(l); %get node id
            linkWeights(l) = nodeWeights(upNodeInd) * exp(-travelCosts(l)*theta);
            nodeWeights(nodeInd) = nodeWeights(nodeInd) + linkWeights(l);
         end
      end

      %Build flows vector, starting from destination(s) and backwards to the
      %origin (also nodeflows vector is used for dial)
      for destination = 1:size(odmatrix,2)
         if origin~=destination
            nodeFlows(destination) = odmatrix(origin,destination);
         end
      end

      %beginning from n
      for  k = totNodes:-1:2
         nodeInd = costTree(k,1); %get node Id
         %get which links enter node k
         for ei = rp_(nodeInd):rp_(nodeInd+1)-1
            l=ai_(ei);
            upNodeInd = strN(l); %get node id
            if ( nodeWeights(nodeInd) == 0 )
               continue;
            end
            originFlows(l,origin) = originFlows(l,origin) +
nodeFlows(nodeInd)*linkWeights(l)/nodeWeights(nodeInd);
            nodeFlows(upNodeInd) = nodeFlows(upNodeInd) +
nodeFlows(nodeInd)*linkWeights(l)/nodeWeights(nodeInd);
         end
         k = k-1;
      end
   end
end
end
```

MAIN_GENT_INIT.m

```
load('..\MATLAB codes\gent_processed.mat')
load('..\MATLAB codes\hb_matrix_gent.mat')
links = edges;
tmp_l = links(links.ind==1642,:);
tmp_l.from = 2861346839;
tmp_l.from_ind = 12824;
tmp_l.to = 247107913;
tmp_l.to_ind = 875;
tmp_l.ind = height(links)+1;
links = [links;tmp_l];
con_mat(end+1,:)=[12824,875,height(links),2];

links=links(con_mat(:,3),:);
links.from_ind=con_mat(:,1);
links.to_ind=con_mat(:,2);
links.lanes=con_mat(:,4);

links.fromNode = links.from_ind;
links.toNode = links.to_ind;
links.freeSpeed = links.maxspeed;
links.length = links.length/1000;

links.class('trunk'==links.highway)=1;
links.class('trunk_link'==links.highway)=2;
links.class('motorway'==links.highway)=3;
links.class('motorway_link'==links.highway)=4;
links.class('primary'==links.highway)=5;
links.class('primary_link'==links.highway)=6;
links.class('secondary'==links.highway)=7;
links.class('secondary_link'==links.highway)=8;
links.class('tertiary'==links.highway)=9;
links.class('tertiary_link'==links.highway)=10;
links.class('residential'==links.highway)=11;
links.class('road'==links.highway)=12;
links.class('living_street'==links.highway)=13;
links.class('unclassified'==links.highway)=14;

links.capacity = links.lanes*1800;

%% Nodes

nodes.id = [1:height(nodes)]';
coor = cell2mat(nodes.co);
nodes.xco = coor(:,1);
nodes.yco = coor(:,2);
nodes.x = nodes.xco;
nodes.y = nodes.yco;

%% Zones
```

```
zones = centroids;
zones.id = [1:height(zones)]';
coor = cell2mat(zones.co);
zones.xco = coor(:,1);
zones.yco = coor(:,2);


%% Plot network
figure;
plot(nodes.xco,nodes.yco,'k.');
hold on;
co = cell2mat(links.co);
plot(co(:,1),co(:,2),'k');
% co = cell2mat(links.co(links.class<=4));
% plot(co(:,1),co(:,2),'b');
selected_communities = [44021,44040,44013,44043,44012,44064];
selected_zones = arrayfun(@(x) any(selected_communities==x), zones.GEMEENTE);
plot(zones.xco,zones.yco,'bo');
plot(zones.xco(selected_zones),zones.yco(selected_zones),'go');

cl_x = cellfun(@(x) mean(x(1:end-1,1)),links.co);
cl_y = cellfun(@(x) mean(x(1:end-1,2)),links.co);

% cl_x = (xco(1:3:end)+xco(2:3:end))/2;
% cl_y = (yco(1:3:end)+yco(2:3:end))/2;

%% Map zones for car
% zones.ref_l = arrayfun(@(x,y) find(min([cl_x-x].^2+[cl_y-y].^2 + 10^10*[links.class<5])==[cl_x-
x].^2+[cl_y-y].^2 + 10^10*[links.class<5],1,'First'),zones.xco,zones.yco);
% zones.dist_l = arrayfun(@(x,y) min([cl_x-x].^2+[cl_y-y].^2 +
10^10*[links.class<5]),zones.xco,zones.yco);

od = HB_matrix_car + 3*HB_matrix_truck(1:end-1,1:end-1);
od1 = od;
od2 = od;
od1(~selected_zones,:)=0;
od2(:,~selected_zones)=0;
od = od1 + od2;
od(selected_zones,selected_zones)=od(selected_zones,selected_zones)/2;

% [o,d,val]=find(od);
[o,d,val]=find(od);

%
od_outside=sum(sum(HB_matrix_car(zones.ZONENUMMER,:)))+sum(sum(HB_matrix_car(:,zones.ZO
NENUMMER)))-2*sum(val);

% od_outside/sum(sum(od))
% f1=plot(nodes.xco(1),nodes.yco(1),'r.');
% f2=plot(nodes.xco(2),nodes.yco(2),'b.');
```

```
lowest_function_out = arrayfun(@(x)
max(links.class(links.from_ind==x)),nodes.id,'UniformOutput',false);
lowest_function_in = arrayfun(@(x) max(links.class(links.to_ind==x)),nodes.id,'UniformOutput',false);
non_used_nodes_out = find(cellfun(@(x) isempty(x), lowest_function_out));
non_used_nodes_in = find(cellfun(@(x) isempty(x), lowest_function_in));
for n=non_used_nodes_out'
   lowest_function_out{n} = -1;
end
for n=non_used_nodes_in'
   lowest_function_in{n} = -1;
end
lowest_function_out = cell2mat(lowest_function_out);
lowest_function_in = cell2mat(lowest_function_in);

nodes.n_in = cellfun(@(x) numel(x),nodes.bw_str);
nodes.n_out = cellfun(@(x) numel(x),nodes.fw_str);

active_nodes_v1_in = (lowest_function_out <= 4 & lowest_function_out ~= -1 & lowest_function_in
== -1);
active_nodes_v2_in =  active_nodes_v1_in | (lowest_function_out <= 10 & lowest_function_out > 0
& nodes.n_in+nodes.n_out <= 2 & ~active_nodes_v1_in);
active_nodes_v3_in = (lowest_function_out > 10 & nodes.n_in+nodes.n_out <= 2 &
~active_nodes_v1_in & ~active_nodes_v2_in);
%
% figure;
% % plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(active_nodes_v1_in),nodes.yco(active_nodes_v1_in),'r.');
% plot(nodes.xco(active_nodes_v2_in),nodes.yco(active_nodes_v2_in),'b.');
% plot(nodes.xco(active_nodes_v3_in),nodes.yco(active_nodes_v3_in),'g.');

active_nodes_v1_out = (lowest_function_in <= 4 & lowest_function_in ~= -1 & lowest_function_out
== -1);
active_nodes_v2_out =  active_nodes_v1_out | (lowest_function_in <= 10 & lowest_function_in > 0
& nodes.n_in+nodes.n_out <= 2 & ~active_nodes_v1_out);
active_nodes_v3_out = (lowest_function_in > 10 & nodes.n_in+nodes.n_out <= 2 &
~active_nodes_v1_out & ~active_nodes_v2_out);

% figure;
% % plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(active_nodes_v1_out),nodes.yco(active_nodes_v1_out),'r.');
% plot(nodes.xco(active_nodes_v2_out),nodes.yco(active_nodes_v2_out),'b.');
% plot(nodes.xco(active_nodes_v3_out),nodes.yco(active_nodes_v3_out),'g.');

tic
for i=1:height(zones)
```

```
   n1_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 <= min((nodes.xco-
zones.xco(i)).^2 + (nodes.yco-zones.yco(i)).^2 + 0.00025) & lowest_function_in > 8 &
lowest_function_out > 8,200,'First');
end
toc
tic
for i=1:height(zones)
   nout_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
10*(~active_nodes_v1_out) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
10*(~active_nodes_v1_out)),1,'First');
   nout_bis{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v2_out) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v2_out)),1,'First');
   nout_tris{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v3_out) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v3_out)),1,'First');
end
toc

figure;
% plot(nodes.xco,nodes.yco,'k.');
hold on;
co = cell2mat(links.co);
plot(co(:,1),co(:,2),'k');
plot(nodes.xco(cell2mat(nout_)),nodes.yco(cell2mat(nout_)),'r.');
plot(nodes.xco(cell2mat(nout_bis)),nodes.yco(cell2mat(nout_bis)),'b.');
plot(nodes.xco(cell2mat(nout_tris)),nodes.yco(cell2mat(nout_tris)),'g.');


tic
for i=1:height(zones)
   nin_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  + 10*(~active_nodes_v1_in)
== ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
10*(~active_nodes_v1_in)),1,'First');
   nin_bis{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v2_in) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v2_in)),1,'First');
   nin_tris{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v3_in) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v3_in)),1,'First');
end
toc

z_xco_m = mean(zones.xco(selected_zones));
z_yco_m = mean(zones.yco(selected_zones));
```

```
OD = zeros(max(nodes.id));
tic
for i=1:length(val)
   if val(i)==0
      continue;
   elseif o(i)~=d(i) && selected_zones(o(i)) && selected_zones(d(i))
      n1=n1_{o(i)};
      n2=n1_{d(i)};
      OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
   elseif o(i)~=d(i) && selected_zones(o(i))
      if sqrt((zones.xco(d(i))-z_xco_m)^2 + (zones.yco(d(i))-z_yco_m)^2) > 0.25
         n1=n1_{o(i)};
         n2=nout_{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      elseif sqrt((zones.xco(d(i))-z_xco_m)^2 + (zones.yco(d(i))-z_yco_m)^2) > 0.15
         n1=n1_{o(i)};
         n2=nout_bis{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      else
         n1=n1_{o(i)};
         n2=nout_tris{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      end
   elseif o(i)~=d(i) && selected_zones(d(i))
      if sqrt((zones.xco(o(i))-z_xco_m)^2 + (zones.yco(o(i))-z_yco_m)^2) > 0.25
         n1=nin_{o(i)};
         n2=n1_{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      elseif sqrt((zones.xco(o(i))-z_xco_m)^2 + (zones.yco(o(i))-z_yco_m)^2) > 0.15
         n1=nin_bis{o(i)};
         n2=n1_{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      else
         n1=nin_tris{o(i)};
         n2=n1_{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      end
   elseif o(i)~=d(i)
      display('bb');%
   end
end
%
% figure;
% plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(n1),nodes.yco(n1),'ro');
% plot(nodes.xco(n2),nodes.yco(n2),'bo');
% plot(zones.xco(selected_zones),zones.yco(selected_zones),'go');
```

```
% plot(zones.xco(o(i)),zones.yco(o(i)),'rx')
% plot(zones.xco(d(i)),zones.yco(d(i)),'rx')

OD=OD-diag(diag(OD));

OD_car=OD;
toc

%% Map zones for truck
% zones.ref_l = arrayfun(@(x,y) find(min([cl_x-x].^2+[cl_y-y].^2 + 10^10*[links.class<5])==[cl_x-
x].^2+[cl_y-y].^2 + 10^10*[links.class<5],1,'First'),zones.xco,zones.yco);
% zones.dist_l = arrayfun(@(x,y) min([cl_x-x].^2+[cl_y-y].^2 +
10^10*[links.class<5]),zones.xco,zones.yco);

od = HB_matrix_truck(1:end-1,1:end-1);
od1 = od;
od2 = od;
od1(~selected_zones,:)=0;
od2(:,~selected_zones)=0;
od = od1 + od2;
od(selected_zones,selected_zones)=od(selected_zones,selected_zones)/2;

% [o,d,val]=find(od);
[o,d,val]=find(od);

%
od_outside=sum(sum(HB_matrix_car(zones.ZONENUMMER,:)))+sum(sum(HB_matrix_car(:,zones.ZO
NENUMMER)))-2*sum(val);

% od_outside/sum(sum(od))
% f1=plot(nodes.xco(1),nodes.yco(1),'r.');
% f2=plot(nodes.xco(2),nodes.yco(2),'b.');

lowest_function_out = arrayfun(@(x)
max(links.class(links.from_ind==x)),nodes.id,'UniformOutput',false);
lowest_function_in = arrayfun(@(x) max(links.class(links.to_ind==x)),nodes.id,'UniformOutput',false);
non_used_nodes_out = find(cellfun(@(x) isempty(x), lowest_function_out));
non_used_nodes_in = find(cellfun(@(x) isempty(x), lowest_function_in));
for n=non_used_nodes_out'
   lowest_function_out{n} = -1;
end
for n=non_used_nodes_in'
   lowest_function_in{n} = -1;
end
lowest_function_out = cell2mat(lowest_function_out);
lowest_function_in = cell2mat(lowest_function_in);

nodes.n_in = cellfun(@(x) numel(x),nodes.bw_str);
nodes.n_out = cellfun(@(x) numel(x),nodes.fw_str);
```

```
active_nodes_v1_in = (lowest_function_out <= 4 & lowest_function_out ~= -1 & lowest_function_in
== -1);
active_nodes_v2_in =  active_nodes_v1_in | (lowest_function_out <= 10 & lowest_function_out > 0
& nodes.n_in+nodes.n_out <= 2 & ~active_nodes_v1_in);
active_nodes_v3_in = (lowest_function_out > 10 & nodes.n_in+nodes.n_out <= 2 &
~active_nodes_v1_in & ~active_nodes_v2_in);
%
% figure;
% % plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(active_nodes_v1_in),nodes.yco(active_nodes_v1_in),'r.');
% plot(nodes.xco(active_nodes_v2_in),nodes.yco(active_nodes_v2_in),'b.');
% plot(nodes.xco(active_nodes_v3_in),nodes.yco(active_nodes_v3_in),'g.');

active_nodes_v1_out = (lowest_function_in <= 4 & lowest_function_in ~= -1 & lowest_function_out
== -1);
active_nodes_v2_out =  active_nodes_v1_out | (lowest_function_in <= 10 & lowest_function_in > 0
& nodes.n_in+nodes.n_out <= 2 & ~active_nodes_v1_out);
active_nodes_v3_out = (lowest_function_in > 10 & nodes.n_in+nodes.n_out <= 2 &
~active_nodes_v1_out & ~active_nodes_v2_out);

% figure;
% % plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(active_nodes_v1_out),nodes.yco(active_nodes_v1_out),'r.');
% plot(nodes.xco(active_nodes_v2_out),nodes.yco(active_nodes_v2_out),'b.');
% plot(nodes.xco(active_nodes_v3_out),nodes.yco(active_nodes_v3_out),'g.');

tic
for i=1:height(zones)
   n1_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 <= min((nodes.xco-
zones.xco(i)).^2 + (nodes.yco-zones.yco(i)).^2 + 0.00025) & lowest_function_in > 8 &
lowest_function_out > 8,200,'First');
end
toc
tic
for i=1:height(zones)
   nout_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
10*(~active_nodes_v1_out) == ...
         min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
10*(~active_nodes_v1_out)),1,'First');
   nout_bis{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v2_out) == ...
         min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v2_out)),1,'First');
   nout_tris{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v3_out) == ...
```

```
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2 +
0.1*(~active_nodes_v3_out)),1,'First');
end
toc

figure;
% plot(nodes.xco,nodes.yco,'k.');
hold on;
co = cell2mat(links.co);
plot(co(:,1),co(:,2),'k');
plot(nodes.xco(cell2mat(nout_)),nodes.yco(cell2mat(nout_)),'r.');
plot(nodes.xco(cell2mat(nout_bis)),nodes.yco(cell2mat(nout_bis)),'b.');
plot(nodes.xco(cell2mat(nout_tris)),nodes.yco(cell2mat(nout_tris)),'g.');


tic
for i=1:height(zones)
   nin_{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  + 10*(~active_nodes_v1_in)
== ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
10*(~active_nodes_v1_in)),1,'First');
   nin_bis{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v2_in) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v2_in)),1,'First');
   nin_tris{i,1}=find((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v3_in) == ...
        min((nodes.xco-zones.xco(i)).^2+(nodes.yco-zones.yco(i)).^2  +
0.1*(~active_nodes_v3_in)),1,'First');
end
toc

z_xco_m = mean(zones.xco(selected_zones));
z_yco_m = mean(zones.yco(selected_zones));

OD = zeros(max(nodes.id));
tic
for i=1:length(val)
   if val(i)==0
      continue;
   elseif o(i)~=d(i) && selected_zones(o(i)) && selected_zones(d(i))
      n1=n1_{o(i)};
      n2=n1_{d(i)};
      OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
   elseif o(i)~=d(i) && selected_zones(o(i))
      if sqrt((zones.xco(d(i))-z_xco_m)^2 + (zones.yco(d(i))-z_yco_m)^2) > 0.25
         n1=n1_{o(i)};
         n2=nout_{d(i)};
         OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      elseif sqrt((zones.xco(d(i))-z_xco_m)^2 + (zones.yco(d(i))-z_yco_m)^2) > 0.15
         n1=n1_{o(i)};
```

```
        n2=nout_bis{d(i)};
        OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      else
        n1=n1_{o(i)};
        n2=nout_tris{d(i)};
        OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      end
    elseif o(i)~=d(i) && selected_zones(d(i))
      if sqrt((zones.xco(o(i))-z_xco_m)^2 + (zones.yco(o(i))-z_yco_m)^2) > 0.25
        n1=nin_{o(i)};
        n2=n1_{d(i)};
        OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      elseif sqrt((zones.xco(o(i))-z_xco_m)^2 + (zones.yco(o(i))-z_yco_m)^2) > 0.15
        n1=nin_bis{o(i)};
        n2=n1_{d(i)};
        OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      else
        n1=nin_tris{o(i)};
        n2=n1_{d(i)};
        OD(n1,n2)=OD(n1,n2)+val(i)/(length(n1)*length(n2)-numel(intersect(n1,n2)));
      end
    elseif o(i)~=d(i)
      display('bb');%
    end
end
%
% figure;
% plot(nodes.xco,nodes.yco,'k.');
% hold on;
% co = cell2mat(links.co);
% plot(co(:,1),co(:,2),'k');
% plot(nodes.xco(n1),nodes.yco(n1),'ro');
% plot(nodes.xco(n2),nodes.yco(n2),'bo');
% plot(zones.xco(selected_zones),zones.yco(selected_zones),'go');
% plot(zones.xco(o(i)),zones.yco(o(i)),'rx')
% plot(zones.xco(d(i)),zones.yco(d(i)),'rx')

OD=OD-diag(diag(OD));

OD_truck=OD;
toc

%% Assign network
%stochastic assignment&

oFlw_Stoch =DFAST((OD_car+3*OD_truck)/100,nodes,links,20);
flows = sum(oFlw_Stoch,2)*100;

%visualize the result
travel_costs =
calculateCostBPR(alpha,beta,sum(oFlw_Stoch,2),links.length,links.freeSpeed,links.capacity);
```

MAIN_GENT_REF1.m

```
%do again for cars & trucks
oFlw_Stoch_car =DFAST_v2(OD_car,nodes,links,20,travel_costs);
flows_car = sum(oFlw_Stoch_car,2);
oFlw_Stoch_truck =DFAST_v2(OD_truck,nodes,links,20,travel_costs);
flows_truck = sum(oFlw_Stoch_truck,2);

%visualize the result
plotLoadedLinks(nodes,links,flows,false,[],[],[]);
alpha = 0.15;
beta = 4;
travel_costs = calculateCostBPR(alpha,beta,flows/24,links.length,links.freeSpeed,links.capacity);
plotLoadedLinks(nodes,links,travel_costs,false,[],[],[]);

links.flows=flows;
links.flows_car=flows_car;
links.flows_truck=flows_truck;
writetable(links(:,[1:21,23:end]),'gent_flow.csv');
```

MAIN_GENT_REF2.m

```
%% Additional calibration of borders

%E17N -> Gent
ind = unique(links.from_ind(find(links.from==30959309)));
val_model=sum(OD_car(ind,:));
val_obs=49069;
OD_car(ind,:)=OD_car(ind,:)*val_obs/val_model;
val_model=sum(OD_truck(ind,:));
val_obs=11714;
OD_truck(ind,:)=OD_truck(ind,:)*val_obs/val_model;
%E17N -> Antwerpen
ind = unique(links.to_ind(find(links.to==2790121907)));
val_model=sum(OD_car(:,ind));
val_obs=48300;
OD_car(:,ind)=OD_car(:,ind)*val_obs/val_model;
val_model=sum(OD_truck(:,ind));
val_obs=11652;
OD_truck(:,ind)=OD_truck(:,ind)*val_obs/val_model;

%E17Z -> Gent
ind = unique(links.from_ind(find(links.from==3886348353)));
val_model=sum(OD_car(ind,:));
val_obs=47609;
OD_car(ind,:)=OD_car(ind,:)*val_obs/val_model;
val_model=sum(OD_truck(ind,:));
val_obs=9509;
OD_truck(ind,:)=OD_truck(ind,:)*val_obs/val_model;
%E17Z -> Kortrijk
ind = unique(links.to_ind(find(links.to==3157371031)));
val_model=sum(OD_car(:,ind));
val_obs=41473;
OD_car(:,ind)=OD_car(:,ind)*val_obs/val_model;
val_model=sum(OD_truck(:,ind));
val_obs=10445;
OD_truck(:,ind)=OD_truck(:,ind)*val_obs/val_model;

%E40E -> Gent
ind = unique(links.from_ind(find(links.from==3134536767)));
val_model=sum(OD_car(ind,:));
val_obs=61305;
OD_car(ind,:)=OD_car(ind,:)*val_obs/val_model;
val_model=sum(OD_truck(ind,:));
val_obs=7986;
OD_truck(ind,:)=OD_truck(ind,:)*val_obs/val_model;
%E40E -> Brussel
ind = unique(links.to_ind(find(links.to==5382838611)));
val_model=sum(OD_car(:,ind));
val_obs=62135;
OD_car(:,ind)=OD_car(:,ind)*val_obs/val_model;
```

```
val_model=sum(OD_truck(:,ind));
val_obs=8112;
OD_truck(:,ind)=OD_truck(:,ind)*val_obs/val_model;

%E40W -> Gent
ind = unique(links.from_ind(find(links.from==676859936)));
val_model=sum(OD_car(ind,:));
val_obs=45088;
OD_car(ind,:)=OD_car(ind,:)*val_obs/val_model;
val_model=sum(OD_truck(ind,:));
val_obs=8167;
OD_truck(ind,:)=OD_truck(ind,:)*val_obs/val_model;
%E40W -> Oostende
ind = unique(links.to_ind(find(links.to==676859969)));
val_model=sum(OD_car(:,ind));
val_obs=46843;
OD_car(:,ind)=OD_car(:,ind)*val_obs/val_model;
val_model=sum(OD_truck(:,ind));
val_obs=8125;
OD_truck(:,ind)=OD_truck(:,ind)*val_obs/val_model;

%% Assign network again
%stochastic assignment

%ADJUST FREESPEED TO ATTRACT MORE CARS TO MAIN ROADS
links.freeSpeed(links.class>10) = links.freeSpeed(links.class>10)*0.25;

oFlw_Stoch =DFAST((OD_car+3*OD_truck)/100,nodes,links,20);
flows = sum(oFlw_Stoch,2)*100;

%visualize the result
plotLoadedLinks(nodes,links,flows,false,[],[],[]);
alpha = 0.15;
beta = 4;
travel_costs = calculateCostBPR(alpha,beta,flows/24,links.length,links.freeSpeed,links.capacity);
plotLoadedLinks(nodes,links,travel_costs,false,[],[],[]);

%do again for cars & trucks
oFlw_Stoch_car =DFAST_v2(OD_car,nodes,links,20,travel_costs);
flows_car = sum(oFlw_Stoch_car,2);
oFlw_Stoch_truck =DFAST_v2(OD_truck,nodes,links,20,travel_costs);
flows_truck = sum(oFlw_Stoch_truck,2);

links.flows=flows;
links.flows_car=flows_car;
plotLoadedLinks(nodes,links,flows_car,false,[],[],[]);
links.flows_truck=flows_truck;
plotLoadedLinks(nodes,links,flows_truck,false,[],[],[]);
writetable(links(:,[1:21,23:end]),'gent_flow_v2_final.csv');
```

MAIN_GENT_REF3.m

```
%% Adding telraam data

%fromNode - toNode - Quantity Cars - Quantity Trucks
telraam_data = [427,     10673,   1502*1.30,   302*1.30;%Burgravenlaan
        10673,     427,   1503*1.30,   301*1.30;%Burgravenlaan
        2902,     11903,   2000*1.30,   301*1.30;%Lange Violettestraat
        11903,     2902,   916*1.30,   100*1.30;%Lange Violettestraat
        2430,     8691,   2810*1.30,   312*1.30;%Edmond Blockstraat
        8691,     2430,   2810*1.30,   350*1.30;%Edmond Blockstraat
        10565,     14895,   2119*1.30,   505*1.30;%Dendermondsesteenweg
        14895,     10565,   2115*1.30,   504*1.30;%Dendermondsesteenweg
        998,     1513,   3573*1.30,   160*1.30;%Annonciadestraat
        1513,     998,   1572*1.30,   169*1.30;%Annonciadestraat
        5198,     12317,     26337,     4963;%R4
        12316,     5196,     26982,     4543;%R4
        12692,     68,   8273*1.30,   1832*1.30;%R4 binnenring
        ];

%setup structure
[rp,ci,ai]=sparse_to_csr(links.toNode,links.fromNode,1:height(links),max(nodes.id)); %
a = links.length(ai);
[rp_,ci_,ai_]=sparse_to_csr(links.fromNode,links.toNode,1:height(links),max(nodes.id)); %
a_ = links.length(ai_);

result_u_flow = zeros(height(links),size(telraam_data,1));
result_u_flow_car = zeros(height(links),size(telraam_data,1));
result_u_flow_truck = zeros(height(links),size(telraam_data,1));

flows_car_adj = flows_car;
flows_truck_adj = flows_truck;
for i=1:size(telraam_data,1)
   %make a double tree
   [d,pred,list,dir]=double_tree(rp,ci,a,telraam_data(i,1),rp_,ci_,a_,telraam_data(i,2),7.5);
   % d_tmp=d;
   % d_tmp(isinf(d))=0;
   % d_tmp(dir==1)=0;
   % d_tmp(d_tmp>0)=d_tmp(d_tmp>0)+1;
   %plotLoadedNodes(nodes,links,d_tmp,false,[],10,[])

   %distribute unit accorinding to flow in the double tree
   result_u_flow(:,i)=distribute(links,pred,list,dir,flows);

   obs_car_flow = telraam_data(i,3);
   model_car_flow = flows_car_adj(links.fromNode == telraam_data(i,1) & links.toNode ==
telraam_data(i,2));
   if (obs_car_flow - model_car_flow) > 0
      flows_car_adj = flows_car_adj + result_u_flow(:,i) * (obs_car_flow - model_car_flow);
   else
```

```
      flows_car_adj = max(flows_car/10,flows_car_adj + result_u_flow(:,i) * (obs_car_flow -
model_car_flow));
   end
   obs_truck_flow = telraam_data(i,4);
   model_truck_flow = flows_truck_adj(links.fromNode == telraam_data(i,1) & links.toNode ==
telraam_data(i,2));
   if (obs_truck_flow - model_truck_flow) > 0
      flows_truck_adj = flows_truck_adj + result_u_flow(:,i) * (obs_truck_flow - model_truck_flow);
   else
      flows_truck_adj = max(flows_truck/10,flows_truck_adj + result_u_flow(:,i) * (obs_truck_flow -
model_truck_flow));
   end
%     plotLoadedLinks(nodes,links,abs(result_u_flow(:,i) * (obs_car_flow -
model_car_flow)),false,[],[],[]);
%     plotLoadedLinks(nodes,links,abs(flows_car_adj),false,[],[],[]);
end

flows_car_v3 = flows_car_adj;
flows_truck_v3 = flows_truck_adj;
% for i=1:size(telraam_data,1)
%     flows_car_v3=(1+result_u_flow_car(:,i)).*flows_car_v3;
%     flows_truck_v3=(1+result_u_flow_truck(:,i)).*flows_truck_v3;
% end
flows_v3 = flows_car_v3 + 3 * flows_truck_v3;

links.flows=flows_v3;
links.flows_car=flows_car_v3;
links.flows_truck=flows_truck_v3;
% writetable(links(:,[1:21,23:end]),'gent_flow_v3.csv');
```

double_tree.m

```
function [d,pred,list,dir]=double_tree(rp,ci,ai,u,rp_,ci_,ai_,u_,cut_off_ai)

%My function to redistribute traffic
%willem.himpe@kuleuven.be

%dir = [1 , 0, -1] %[forward, unassigned, backward]

n=length(rp)-1;
d=Inf(n,1); T=zeros(n,1); L=zeros(n,1);dir=zeros(n,1);
pred=zeros(1,length(rp)-1);
list=zeros(length(rp)-1,1);
xout=true(length(rp)-1,1);
l_it=1;

n=1; T(n)=u; L(u)=n; % oops, n is now the size of the heap
n=2; T(n)=u_; L(u_)=n;

dir(u)=-1;
dir(u_)=1;
pred(u) = u_;
pred(u_) = u;
d(u) = 0;
d(u_) = 0;

% enter the main loop
while n>0
   v=T(1);ntop=T(n); T(1)=ntop; L(ntop)=1; n=n-1; % pop the head off the heap
   list(l_it)=v;
   xout(v)=false;
   l_it=l_it+1;

   k=1; kt=ntop;              % move element T(1) down the heap
   while 1,
      i=2*k;
      if i>n, break; end       % end of heap
      if i==n, it=T(i);        % only one child, so skip
      else                     % pick the smallest child
         lc=T(i); rc=T(i+1); it=lc;
         if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
      end
      if d(kt)<d(it), break;    % at correct place, so end
      else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
      end
   end                         % end heap down

   % for each vertex adjacent to v, relax it
   if dir(v)==-1
      for ei=rp(v):rp(v+1)-1          % ei is the edge index
         w=ci(ei); ew=ai(ei);         % w is the target, ew is the edge weight
```

```
      % relax edge (v,w,ew) if it is not already on different side
      if d(w)>d(v)+ew && dir(w)~=1 && d(v)+ew < cut_off_ai
        d(w)=d(v)+ew; pred(w)=v;
        dir(w)=-1;
        % check if w is in the heap
        k=L(w); onlyup=0;
        if k==0
          % element not in heap, only move the element up the heap
          n=n+1; T(n)=w; L(w)=n; k=n; kt=w; onlyup=1;
        else kt=T(k);
        end
        % update the heap, move the element down in the heap
        while 1 && ~onlyup,
          i=2*k;
          if i>n, break; end        % end of heap
          if i==n, it=T(i);         % only one child, so skip
          else                      % pick the smallest child
            lc=T(i); rc=T(i+1); it=lc;
            if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
          end
          if d(kt)<d(it), break;    % at correct place, so end
          else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
          end
        end
        % move the element up the heap
        j=k; tj=T(j);
        while j>1,                  % j==1 => element at top of heap
          j2=floor(j/2); tj2=T(j2);   % parent element
          if d(tj2)<d(tj), break;    % parent is smaller, so done
          else                      % parent is larger, so swap
            T(j2)=tj; L(tj)=j2; T(j)=tj2; L(tj2)=j; j=j2;
          end
        end
      end
    end
  elseif dir(v)==1
    for ei=rp_(v):rp_(v+1)-1        % ei is the edge index
      w=ci_(ei); ew=ai_(ei);        % w is the target, ew is the edge weight
      % relax edge (v,w,ew) if it is not already on different side
      if d(w)>d(v)+ew && dir(w)~=-1 && d(v)+ew < cut_off_ai
        d(w)=d(v)+ew; pred(w)=v;
        dir(w)=1;
        % check if w is in the heap
        k=L(w); onlyup=0;
        if k==0
          % element not in heap, only move the element up the heap
          n=n+1; T(n)=w; L(w)=n; k=n; kt=w; onlyup=1;
        else kt=T(k);
        end
        % update the heap, move the element down in the heap
        while 1 && ~onlyup,
```

```
            i=2*k;
            if i>n, break; end          % end of heap
            if i==n, it=T(i);           % only one child, so skip
            else                        % pick the smallest child
               lc=T(i); rc=T(i+1); it=lc;
               if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
            end
            if d(kt)<d(it), break;      % at correct place, so end
            else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
            end
          end
          % move the element up the heap
          j=k; tj=T(j);
          while j>1,                    % j==1 => element at top of heap
            j2=floor(j/2); tj2=T(j2);   % parent element
            if d(tj2)<d(tj), break;     % parent is smaller, so done
            else                        % parent is larger, so swap
               T(j2)=tj; L(tj)=j2; T(j)=tj2; L(tj2)=j; j=j2;
            end
          end
        end
      end
    else
      display('problem');
    end
end
list(l_it:end)=find(xout);
% figure;plot(N(1:sl_it));
end
```

distribute.m

```
function distr=distribute(links,pred,list,dir,flow)

distr = zeros(size(flow));


%backward pass
list_bw=list(dir(list)==-1);
pred_bw=pred(list_bw)';


distr(links.fromNode ==list_bw(1) & links.toNode ==pred_bw(1))=1;
selected_l = cell2mat(arrayfun(@(x,y) find(links.fromNode == x & links.toNode ==
y),list_bw,pred_bw,'UniformOutput',false));

for n_ind=2:length(list_bw)
   list_l = selected_l(links.toNode(selected_l)==pred_bw(n_ind));
   [~,ia1]=intersect(links.fromNode(list_l),list_bw);
   list_l = list_l(ia1);
   ia1=find(links.fromNode(list_l)==list_bw(n_ind));
   distr(list_l(ia1))=sum(distr(links.fromNode ==
pred_bw(n_ind)))*flow(list_l(ia1))/(sum(flow(list_l))+eps);
end

%forward pass
list_fw=list(dir(list)==1);
pred_fw=pred(list_fw)';

distr(links.toNode ==list_fw(1) & links.fromNode ==pred_fw(1))=1;
selected_l = cell2mat(arrayfun(@(x,y) find(links.toNode == x & links.fromNode ==
y),list_fw,pred_fw,'UniformOutput',false));

for n_ind=2:length(list_fw)
   list_l = selected_l(links.fromNode(selected_l)==pred_fw(n_ind));
   [~,ia1]=intersect(links.toNode(list_l),list_fw);
   list_l = list_l(ia1);
   ia1=find(links.toNode(list_l)==list_fw(n_ind));
   distr(list_l(ia1))=sum(distr(links.toNode ==
pred_fw(n_ind)))*flow(list_l(ia1))/(sum(flow(list_l))+eps);
end

end
```

4.3 Air Quality module – Python & Matlab code

Meteo file - Python program code:

```
# -*- coding: utf-8 -*-
import numpy as np

######### name list ################

PATH = "PATH"
n_level=29 #number of vertical levels
n_wrf_cells=88 #number of files from wrf

############### read WRF 2D #######

#tempo   Albedo   PSFC      RAINC    RH        T2C       Rugosidade        U10       V10
year=[]
month=[]
day=[]
hour=[]
Albedo=[]
PSFC=[]
RAINC=[]
RH=[]
T2C=[]
Rugosidade=[]
U10=[]
V10=[]

for i in range(1,n_wrf_cells+1,1):

   ficheiroin2D= '2D_WRFOUT_2010_CIRA_'+str(i)+'.csv' #name of WRF file with information 2D
   print ficheiroin2D

   OPEN_DOMAIN_FILE2D = open(PATH + ficheiroin2D,'r')
   lines = OPEN_DOMAIN_FILE2D.readlines()

   #tempo          Albedo   PSFC      RAINC    RH        T2C       Rugosidade        U10       V10

   for line in lines[1:]:
      #print line
      day.append(int(line.split(";")[0][0:2])) #01/01/2010 00:00
      month.append(int(line.split(";")[0][3:5]))
      year.append(int(line.split(";")[0][6:10])+5)
      hour.append(int(line.split(";")[0][11:13]))
            Albedo.append(float(line.split(";")[1]))
            PSFC.append(float(line.split(";")[2]))
            RAINC.append(float(line.split(";")[3]))
            RH.append(float(line.split(";")[4]))
            T2C.append(float(line.split(";")[5]))
            Rugosidade.append(float(line.split(";")[6]))
            U10.append(float(line.split(";")[7]))
            V10.append(float(line.split(";")[8]))

print len(Albedo), len(PSFC), len(RAINC), len(RH), len(T2C), len(Rugosidade), len(U10), len(V10)
```

```
Albedo_f=[]
PSFC_f=[]
RAINC_f=[]
RH_f=[]
T2C_f=[]
Rugosidade_f=[]
U_1f=[]
V_1f=[]

for jj in range(0,(365*24),1):
    Albedo_1=[]
    PSFC_1=[]
    RAINC_1=[]
    RH_1=[]
    T2C_1=[]
    Rugosidade_1=[]
    U_1=[]
    V_1=[]
    for ii in range(0,len(Albedo),(365*24)):
        Albedo_1.append(Albedo[ii+jj])
        PSFC_1.append(PSFC[ii+jj])
        RAINC_1.append(RAINC[ii+jj])
        RH_1.append(RH[ii+jj])
        T2C_1.append(T2C[ii+jj])
        Rugosidade_1.append(Rugosidade[ii+jj])
        U_1.append(U10[ii+jj])
        V_1.append(V10[ii+jj])
    #print len(Albedo_1)
    Albedo_f.append(sum(Albedo_1)/len(Albedo_1))
    PSFC_f.append(sum(PSFC_1)/len(PSFC_1))
    RAINC_f.append(sum(RAINC_1)/len(RAINC_1))
    RH_f.append(sum(RH_1)/len(RH_1))
    T2C_f.append(sum(T2C_1)/len(T2C_1))
    Rugosidade_f.append(sum(Rugosidade_1)/len(Rugosidade_1))
    U_1f.append(sum(U_1)/len(U_1))
    V_1f.append(sum(V_1)/len(V_1))

#print len(Albedo_f)
print 'read files wrf 2d finished'

WS_1f=[]
WD_1f=[]
for ij in range(len(U_1f)):
    WS_1f.append(np.sqrt(float(U_1f[ij])**2+float(V_1f[ij])**2))
    WD_1f.append(57.3*np.arctan2(U_1f[ij],V_1f[ij])+180)

############### read WRF 3D #######

year2=[]
month2=[]
day2=[]
nivel=[]
P=[]
altura=[]
TC=[]
```

```
U=[]
V=[]

for ij in range(1,n_wrf_cells+1,1):

   ficheiroin3D= '3D_WRFOUT_2010_CIRA_'+str(ij)+'.csv' #name of WRF file with information 3D
   print ficheiroin3D

   OPEN_DOMAIN_FILE3D = open(PATH + ficheiroin3D,'r')
   lines = OPEN_DOMAIN_FILE3D.readlines()
   #tempo          nivel      P          TC          WS          WD
   for li in lines[1:21171]:
      #print li
      day2.append(int(li.split(";")[0][0:2])) #01/01/2010 00:00
      month2.append(int(li.split(";")[0][3:5]))
      year2.append(int(li.split(";")[0][6:10])+5)
            nivel.append(float(li.split(";")[1]))
            P.append(float(li.split(";")[2]))
            altura.append(float(li.split(";")[3]))
            TC.append(float(li.split(";")[4]))
            U.append(float(li.split(";")[5]))
            V.append(float(li.split(";")[6]))

print len(nivel), len(P), len(TC), len(U10), len(V10)

nivel_f=[]
P_f=[]
altura_f=[]
TC_f=[]
U_2f=[]
V_2f=[]

for jj in range(0,(365*2*29),1):
   nivel_1=[]
   P_1=[]
   altura_1=[]
   TC_1=[]
   U_1=[]
   V_1=[]
   for ii in range(0,len(nivel),(365*2*29)):
      nivel_1.append(nivel[ii+jj])
      P_1.append(P[ii+jj])
      altura_1.append(altura[ii+jj])
      TC_1.append(TC[ii+jj])
      U_1.append(U[ii+jj])
      V_1.append(V[ii+jj])

   nivel_f.append(sum(nivel_1)/len(nivel_1))
   P_f.append(sum(P_1)/len(P_1))
   altura_f.append(sum(altura_1)/len(altura_1))
   TC_f.append(sum(TC_1)/len(TC_1))
   U_2f.append(sum(U_1)/len(U_1))
   V_2f.append(sum(V_1)/len(V_1))

WS_2f=[]
WD_2f=[]
```

```
for ij in range(len(U_2f)):
   WS_2f.append(np.sqrt((U_2f[ij])**2+(V_2f[ij])**2))
   WD_2f.append(57.3*np.arctan2(U_2f[ij],V_2f[ij])+180)

############## write file  meteo_SMD #######

print 'write file meteo_SMD'

#mes=['jan','fev','mar','abr','mai','jun','jul','ago','set','out','nov','dez']
#ndiasmes=[31,28,31,30,31,30,31,31,30,31,30,31]
#ndias=0
#for ms in range(len(mes)):

out1 = open(PATH+'meteo_SMD_CIRA.dat', 'w') #name of URBAIR input meteo SMD

out1.write('*****************************************************************************
*****************************************************************************
*****************************************************************************
*****************************************************************************
************************************************************************\n')
out1.write('URBAIR          SURFACE DATA    INPUT    FILE\n')
out1.write('*****************************************************************************
*****************************************************************************
*****************************************************************************
*****************************************************************************
************************************************************************\n')
out1.write('-----------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------\n')
out1.write('year   month   day     hour    minut   sec     ta0     ta1     ta2     vv1     vv2     dv1
          dv2     stdV1   stdV2   stdH1   stdH2   Pa      hr0     hr1     rad     nuv     ceilH   Prec
          cdPrec  Rn      corio   Rib     roar    Cpar    Cpas    clas    CBSB    L       ustr    wstr
          Tpstr   Hsf     smh     GVTP    zim     zic     zi      cdz0    z0      bow     albed   ang
          acrit   hns     hps     mflag\n')
out1.write('1      2       3       4       5       6       7       8       9       10      11      12
          13      14      15      16      17      18      19      20      21      22      23      24
          25      26      27      28      29      30      31      32      33      34      35      36
          37      38      39      40      41      42      43      44      45      46      47      48
          49      50      51      52\n')
out1.write('-----------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------\n')
#print ndias
for ijj in range(0,len(WS_1f),1):
   out1.writelines('%s      %s      %s      %s      %s      %s      %s      ' %
(year[ijj],month[ijj],day[ijj],hour[ijj],0,0,-999))
   out1.writelines('%f      %s      %f      %s      %s      %s      %s      ' % (T2C_f[ijj],-999,WS_1f[ijj],-
999,WD_1f[ijj],-999,0))
   out1.writelines('%s      %s      %s      %f      %s      %f      %s      ' % (-999,-999,-999,PSFC_f[ijj],-
999,RH_f[ijj],-999))
   out1.writelines('%s      %s      %s      %s      %s      %s      %s      ' % (-999,-999,0,0,-9999,-999,-
999))
   out1.writelines('%s      %s      %s      %s      %s      %s      %s      ' % (-999,-999,-999,-999,-999,-
99999,-999))
   out1.writelines('%s      %s      %s      %s      %s      %s      %s      ' % (-999,-999,-99999,-999,-
999,-999,-999))
   out1.writelines('%s      %s      %f      %f      %f      %s      %s      ' % (-999,-
999,1.5,0.5,Albedo_f[ijj],-999,-999))
```

```
   out1.writelines('%s        %s        %s' % (-999,-999,-999))
   out1.write("\n")

out1.close()

############### write file meteo_UAD #######

print 'write file meteo_UAD'

out2 = open(PATH +'meteo_UAD_CIRA.dat', 'w') #name of URBAIR input meteo UAD

out2.write('*******************************************************************************\n')
out2.write('URBAIR          UPPER   air         data:\n')
out2.write('-------------------------------------------------------------------------------\n')
out2.write('k       /       nr.       sounds  /       day/month/year  /       hour/mint/sec/certarad\n')
out2.write('-------------------------------------------------------------------------------\n')
out2.write('level  /        pressure /        temp    /       dew     point   temp    /       wind
        speed   /       wind    direction\n')
out2.write('*******************************************************************************\n')
print len(WS_2f)
for ijjj in range(0,len(WS_2f),(n_level*2)):
   out2.write('-------------------------------------------------------------------------------\n')
   out2.write('1    '+str(n_level)+'    '+str(day2[ijjj])+'  '+str(month2[ijjj])+'          '+str(year2[ijjj])+' 0          0
        0          4\n')
   out2.write('-------------------------------------------------------------------------------\n')
   for ji in range(0,n_level,1):
      out2.writelines('%s    %s        %s        %s        %s        %s        %s        ' %
(int(altura_f[ijjj+ji]),int(P_f[ijjj+ji]),int(TC_f[ijjj+ji]),9.8,-999,int(WS_2f[ijjj+ji]),int(WD_2f[ijjj+ji])))
      out2.writelines('\n')
   out2.write('-------------------------------------------------------------------------------\n')
   out2.write('1    '+str(n_level)+'    '+str(day2[ijjj])+'  '+str(month2[ijjj])+'          '+str(year2[ijjj])+' 12         0
        0          2\n')
   out2.write('-------------------------------------------------------------------------------\n')
   for ji in range(0,n_level,1):
      out2.writelines('%s    %s        %s        %s        %s        %s        %s        ' %
(int(altura_f[ijjj+ji+n_level]),int(P_f[ijjj+ji+n_level]),int(TC_f[ijjj+ji+n_level]),9.8,-
999,int(WS_2f[ijjj+ji+n_level]),int(WD_2f[ijjj+ji+n_level])))
      out2.writelines('\n')
out2.close()
```

Domain – python code

Python program code:

```
"""
The script creates the domain.dat input file
"""

############################################################
#MODULES
from dbfread import DBF
import mpl_toolkits.basemap.pyproj as pyproj # Import the pyproj module
import numpy as np

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

WORK_PATH = "PATH"
INPUT_PATH = WORK_PATH + "PATH"
INPUT_FILE_NAME = "FILE_NAME.dbf"
OUTPUT_PATH = WORK_PATH + "PATH"

PRESENCE_BUILDINGS = 1 # presence of buildings (0-No/1-Yes)
GRIDTYPE = 1 #gridtype/nomerecgrid
NAMERECGRID = "malha1" #gridtype/nomerecgrid
orzon = 8 # time zone

Id = []
POINT_X = []
POINT_Y = []

for record in DBF(INPUT_PATH + INPUT_FILE_NAME):
  Id.append(record["Id"])
  POINT_X.append(record["POINT_X"])
  POINT_Y.append(record["POINT_Y"])

max_x = float(max(POINT_X)) + abs((POINT_X[0]-POINT_X[1])/2)
min_x = float(min(POINT_X)) - abs((POINT_X[0]-POINT_X[1])/2)
max_y = float(max(POINT_Y)) + abs((POINT_X[0]-POINT_X[1])/2)
min_y = float(min(POINT_Y)) - abs((POINT_X[0]-POINT_X[1])/2)

CP_x = (max_x + min_x)/2
CP_y = (max_y + min_y)/2
xrecgci = int(min_x - CP_x)
yrecgci = int(min_y - CP_y)
xrecgci_MAX = int(max_x - CP_x)
yrecgci_MAX = int(max_y - CP_y)
xrecgcn = int((max_x - min_x)/abs((POINT_X[0]-POINT_X[1])))
yrecgcn = int((max_y - min_y)/abs((POINT_X[0]-POINT_X[1])))
xrecgcdl = int(abs((POINT_X[0]-POINT_X[1])))
yrecgcdl = int(abs((POINT_X[0]-POINT_X[1])))

nrgrid = xrecgcn * yrecgcn

# Define some common projections using EPSG codes
wgs84=pyproj.Proj("+init=EPSG:4326") # LatLon with WGS84 datum used by GPS units and Google Earth
# Define a projection with Proj4 notation: WRF_CAMX simulations
```

```
"""
Supplied by NYC
Lambert Conformal Conic:
   Standard Parallel (lat_1): 30.98
   Standard Parallel (lat_2): 30.98
   Longitude of Central Meridian (lon_0): 107.97
   Latitude of Projection Origin (lat_0): 30.98
   False Easting (x_0): 0.000000
   False Northing (y_0): 0.000000
"""
lat_1 = 38.716 # It necessary change
lat_2 = 38.716 # It necessary change
lon_0 = -9.084 # It necessary change
lat_0 = 38.716 # It necessary change
x_0 = 0 # It necessary change
y_0 = 0 # It necessary change

lcc = "+proj=lcc +lat_1=" + str(lat_1) + " +lat_2=" + str(lat_2) + " +lat_0=" + str(lat_0) + " +lon_0=" + str(lon_0) +
" +x_0=" + str(x_0) +  " +y_0=" + str(y_0) + "+no_defs +a=6378137 +rf=298.257222101 +to_meter=1"
WRF_CAMx_lcc = pyproj.Proj(str(lcc))
xx, yy = pyproj.transform(WRF_CAMx_lcc, wgs84, min_x, min_y)

orlong = round(np.array(xx),3)
orlat = round(np.array(yy),3)

f = open(OUTPUT_PATH + 'domain.dat', 'w')
f.write("**********************************************************************************
**********\n")
f.write(" URBAIR MODEL - STUDY DOMAIN TERRAIN DATA, RECEPTORS & SourCES DATA\n")

f.write("**********************************************************************************
**********\n")
f.write(str(PRESENCE_BUILDINGS) +"                                ynedif                - presence of buildings (0-
No/1-Yes)\n")
f.write(str(orlat) + "          " + str(orlong) + "  " + str(orzon) +  "orlat/orlong/orzon - Latitude(+east)/Longitude
(+north)/time zone\n")
f.write("##################################################################################################
################\n")
f.write("              RECEPTORS LOCATION DEFINITION\n")
f.write("##################################################################################################
################\n")
f.write(str(nrgrid) +"                        nrgrid - number of receptors grids\n")
f.write("1 malha1gridtype/nomerecgrid\n")
f.write(str(GRIDTYPE) + " " + str(NAMERECGRID) + "    gridtype/nomerecgrid\n")
f.write("**********************************************************************************
**********\n")
f.write(str(xrecgci) + " " + str(xrecgcn)+ " " +  str(xrecgcdl)+ "    xrecgci/xrecgcn/xrecgcdl ---- x-axix - \
initial point, number of points, delta space\n")
f.write(str(yrecgci) + " " + str(yrecgcn)+ " " +  str(yrecgcdl)+ "    yrecgci/yrecgcn/yrecgcdl ---- y-axix - \
initial point, number of points, delta space\n")
f.write("**********************************************************************************
**********\n")
f.close()

f = open(OUTPUT_PATH + 'domain_to_URBAIR_INPUT.dat', 'w')
f.write("--------------------------------------------------------------------------------\n")
```

```
f.write("CaseStudyName\n")
f.write(str(orlat) + "            " + str(orlong) + "  " + str(orzon) + " 0
        latitude/longitude/time zone/rotation from true north\n")
f.write(str(xrecgci) + " " + str(yrecgci) + "  " + str(xrecgci_MAX) + " " + str(yrecgci_MAX) + "
        simulation domain extreme points\n")
f.write("--------------------------------------------------------------------------------\n")
f.close()
```

Road transport pre-processor in Matlab

```
"""
It prepares the road transport emissions for the CLAIRCITY project
"""
WORK_PATH = '/mnt/porao/GEMAC/Kevin_Oliveira/ROAD_TRANSPORT_EMISSIONS/'
DOMAIN_NAME = "Dominio_URBAIR_200x200_grid_LJU_label.csv"
DATA_BEGIN = 20150101
DATA_END = 20160101
UTM = 0 # (the values can range between -12 and 12)
POLLUTANTS = ["PM","NOX"]
RESOLUTION = 200
Tp2F = 1 # Location (1-urban/2-peri-urban/3-rural/4-industrial)
Tp3F = 1 # Emission type (0-continuous/1-variable)

'''
Modules
'''
import datetime as dt  # Python standard library datetime  module
from xlrd import open_workbook
import pandas as pd

"""
Functions
"""
def weekDay(year, month, day):
   offset = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]
   week   = ['sunday',
             'monday',
             'tuesday',
             'wednesday',
             'thursday',
             'friday',
             'saturday']
```

```
    afterFeb = 1
    if month > 2: afterFeb = 0
    aux = year - 1700 - afterFeb
    # dayOfWeek for 1700/1/1 = 5, Friday
    dayOfWeek  = 5
    # partial sum of days betweem current date and 1700/1/1
    dayOfWeek += (aux + afterFeb) * 365
    # leap year correction
    dayOfWeek += aux / 4 - aux / 100 + (aux + 100) / 400
    # sum monthly and day offsets
    dayOfWeek += offset[month - 1] + (day - 1)
    dayOfWeek %= 7
    return dayOfWeek, week[dayOfWeek]

"""
SCRIPT
"""

"""
Treatment of dates
"""
yybeg = str(DATA_BEGIN)[0:4]
mmbeg = str(DATA_BEGIN)[4:6]
ddbeg = str(DATA_BEGIN)[6:8]
datbeg_hour = dt.datetime(int(yybeg),int(mmbeg),int(ddbeg))
datbeg = datbeg_hour.date()

yyend = str(DATA_END)[0:4]
mmend = str(DATA_END)[4:6]
ddend = str(DATA_END)[6:8]
datend_hour = dt.datetime(int(yyend),int(mmend),int(ddend))
datend = datend_hour.date()
number_of_days = (datend - datbeg).days
```

```
EMISSIONS_days = []
EMISSIONS_days_V2 = []
EMISSIONS_days_UTM = []
datbeg_hour_UTM = (datbeg_hour + dt.timedelta(hours=int(UTM)))

EMISSIONS_days = []
SELECT_MONTHS = []
for i in range(0,int(number_of_days)):
    idt = (datbeg + dt.timedelta(days=i)).strftime("%Y%m%d")
    EMISSIONS_days.append(idt)
    SELECT_MONTHS.append(int(idt[4:6]))

SELECT_MONTHS_V2 =list(set(SELECT_MONTHS))

for i in range(0,int(number_of_days)*24):
    idt = (datbeg_hour + dt.timedelta(hours=i)).strftime("%Y%m%d%H")
    idt_UTM = (datbeg_hour_UTM + dt.timedelta(hours=i)).strftime("%Y%m%d%H")
    EMISSIONS_days_V2.append(idt)
    EMISSIONS_days_UTM.append(idt_UTM)

year = int(yybeg)
year_split_digit = str(year)
year_split_digit = year_split_digit[2] + year_split_digit[3]

if year % 400 == 0:
    month_days = (31,29,31,30,31,30,31,31,30, 31,30,31)
    February                                                                  =
["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20
","21","22","23","24","25","26","27","28","29"]
elif year % 4 == 0:
    if year % 100 != 0:
        month_days = (31,29,31,30,31,30,31,31,30, 31,30,31)
```

```
        February                                                                                    =
["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20
","21","22","23","24","25","26","27","28","29"]
    else:
        month_days = (31,29,31,30,31,30,31,31,30, 31,30,31)
        February                                                                                    =
["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20
","21","22","23","24","25","26","27","28","29"]
else:
    month_days = (31,28,31,30,31,30,31,31,30, 31,30,31)
    February                                                                                        =
["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20
","21","22","23","24","25","26","27","28"]

"""
TO CALCULATE THE EMISSIONS
"""
wb = open_workbook(WORK_PATH + '/INPUT/PARAMETERS_EMISSIONS_BY_AREA.xlsx')
sheet = wb.sheet_by_name('PARAMETERS')
head = []
for col in range(sheet.ncols):
    head.append(sheet.cell_value(0,col))
position = head.index(str("URBair Manual"))

values = []
for row in range(sheet.nrows):
    values.append(sheet.cell_value(row,position))

ptnF_AREA = values[1] # Temperature [k]
dpnF_AREA = values[2] # Pressure [Pa]
Q_AREA = values[3] # Flow [g/m2.s]
vtnF_AREA = values[4] # Velocity [m/s]
Z0f_AREA  = values[5] # Z coordinate of the source
angF_AREA = values[6] # Angle of rotation in relation to the north
```

```
for a in range(0,len(POLLUTANTS)):

   WEEKDAY_PROFILE = open(WORK_PATH + "/INPUT/" + str(POLLUTANTS[a]) + "_WEEKDAY.csv", "r")
   WEEKDAY_PROFILE_READ = WEEKDAY_PROFILE.readlines()

   WEEKEND_PROFILE = open(WORK_PATH + "/INPUT/" + str(POLLUTANTS[a]) + "_WEEKEND.csv", "r")
   WEEKEND_PROFILE_READ = WEEKEND_PROFILE.readlines()

   for b in range(1,len(SELECT_MONTHS_V2)+1):
      SELECT_DAYS = []

      for c in range(0,len(EMISSIONS_days)):

         if int(EMISSIONS_days[c][4:6]) == int(b):
            SELECT_DAYS.append(EMISSIONS_days[c])

      FILE = open(WORK_PATH +
'/OUTPUT/emissoes_variaveis_'+str(POLLUTANTS[a])+'_'+str(b)+'.dat', 'w')

FILE.write('*************************************************************************
*******************\n')

      FILE.write(' URBAIR VARIABLE          EMISSIONS       INPUT   DATA    FILE\n')


FILE.write('*************************************************************************
*******************\n')

      FILE.write('-------------------------------------------------------------------------------------\n')

      FILE.write('year    month  day    hour    min    sec    ID1    ID2    q        vel      Tp
         P\n')
      FILE.write('-------------------------------------------------------------------------------------\n')

      for d in range(0,len(SELECT_DAYS)):
```

```
        WEEKDAY_v2 = weekDay(int(SELECT_DAYS[d][0:4]),int(SELECT_DAYS[d][4:6]),int(SELECT_DAYS[d][6:8]))

        if WEEKDAY_v2[1] == "monday" or  WEEKDAY_v2[1] == "tuesday" or WEEKDAY_v2[1] == "wednesday" or WEEKDAY_v2[1] == "thursday" or WEEKDAY_v2[1] == "friday":
            temp = WEEKDAY_PROFILE_READ
        if WEEKDAY_v2[1] == "saturday" or  WEEKDAY_v2[1] == "sunday":
            temp = WEEKEND_PROFILE_READ

        for e in range(0,24):
            SELECT_ID = []
            ID = 0
            for f in range(1,len(temp)):

                YEAR_v2 = str(int(SELECT_DAYS[d][0:4]))
                MONTH_v2 = str(int(SELECT_DAYS[d][4:6]))
                DAY_v2 = str(int(SELECT_DAYS[d][6:8]))
                HOUR_v2 = str(int(e))
                MINUTE_v2 = str(0)
                SECOND_v2 = str(0)

                AREA_for_EACH_GRID = RESOLUTION*RESOLUTION
                EMISSION_HOUR_2 = float("{0:.15f}".format(float(temp[f].split(";")[e+1].replace('\n', ''))*(1/60.0)*(1/60.0)*(1.0/AREA_for_EACH_GRID)))

                if float(EMISSION_HOUR_2) > 0:

                    #ID1_v2 = str(temp[e].split(";")[0]) + ' '
                    #ID2_2 = str(temp[e].split(";")[0]) + ' '
                    ID = ID + 1

                    SELECT_ID.append(temp[f].split(";")[0])

                    print (str(YEAR_v2) + ' ' + str(MONTH_v2) + ' ' + str(DAY_v2) + ' ' + str(HOUR_v2) + 'h WRITE EMISSIONS FILE')
```

```
            VEL_VAR = vtnF_AREA
            TP_VAR = ptnF_AREA
            P_VAR = int(dpnF_AREA)

            FILE.write(str(YEAR_v2) + ' ' + str(MONTH_v2) + ' ' + str(DAY_v2)+ ' ' +\
            str(HOUR_v2) + ' '+ str(MINUTE_v2) + ' ' + str(SECOND_v2) + ' ' + str(ID) + ' '+ \
            str(ID) + ' ' +str(EMISSION_HOUR_2) + ' ' + str(VEL_VAR) + \
            ' ' + str(int(TP_VAR)) + ' ' + str(P_VAR) + '\n')

    FILE.write('\n')

FILE.write('**********************************************************************
**********')
    FILE.close()

    # CENTRAL POINT OF DOMAIN
    df = pd.read_csv(WORK_PATH+"/INPUT/DOMAIN/"+ str(DOMAIN_NAME), sep=';', skiprows=0)
    df.columns=['Id','ID_CELL','ORI_FID','POINT_X','POINT_Y']

    max_x = float(max(df.POINT_X)) + abs((df.POINT_X[0]-df.POINT_X[1])/2)
    min_x = float(min(df.POINT_X)) - abs((df.POINT_X[0]-df.POINT_X[1])/2)
    max_y = float(max(df.POINT_Y)) + abs((df.POINT_X[0]-df.POINT_X[1])/2)
    min_y = float(min(df.POINT_Y)) - abs((df.POINT_X[0]-df.POINT_X[1])/2)

    max_x_URBAIR = float(max(df.POINT_X)) - abs((df.POINT_X[0]-df.POINT_X[1])/2)
    max_y_URBAIR = float(max(df.POINT_X)) - abs((df.POINT_X[0]-df.POINT_X[1])/2)

    CP_x = (max_x + min_x)/2
    CP_y = (max_y + min_y)/2

    EACH_GRID = max(abs(int(df.POINT_X[0]-df.POINT_X[1])),abs(int(df.POINT_Y[0]-
df.POINT_Y[1])))
    AREA_for_EACH_GRID = EACH_GRID*EACH_GRID
```

```
# READ PARAMETERS EMISSIONS BY AREA
ID_AREA = range(1,len(df.Id)+1)
POINT_X_AREA = []
POINT_Y_AREA = []
for g in range(0,len(SELECT_ID)):

   temp = df.loc[(df['ORI_FID'] == int(SELECT_ID[g]))]

   POINT_X_AREA.append(float(temp.POINT_X) - CP_x - abs((EACH_GRID/2.0)))
   POINT_Y_AREA.append(float(temp.POINT_Y) - CP_y - abs((EACH_GRID/2.0)))

dxF_AREA = EACH_GRID
dyF_AREA = EACH_GRID

FILE = open(WORK_PATH + '/OUTPUT/EMISSIONS_AREA_RECTANGULAR_urbair_input_'+str(POLLUTANTS[a])+'.dat', 'w')
FILE.write('The number of emissions AREA grid are: ' + str(len(SELECT_ID)) +'\n')
for h in range(0,len(SELECT_ID)):

   ID1 =  h + 1
   ID2 =  h + 1
   Tp1F = 2

   FILE.write(str(ID1) + ' ' + str(Tp1F) + ' ' + str(int(Tp2F))+' '+ \
   str(int(Tp3F)) +' '+ str(float("{0:.1f}".format(POINT_X_AREA[h]))) +' '+\
   str(float("{0:.1f}".format(POINT_Y_AREA[h]))) +\
   ' ' + str(Z0f_AREA) + ' ' + str(dxF_AREA) + ' ' + str(dyF_AREA) +\
   ' ' + str(angF_AREA) + ' ' +  str(float("{0:.3f}".format(Q_AREA))) + ' ' \
   + str(float("{0:.2f}".format(vtnF_AREA))) + ' ' + \
   str(int(ptnF_AREA)) + ' ' + str(int(dpnF_AREA)) + '\n')
FILE.close()
```

Residential and commercial preprocessor in Matlab:

```
%% Add Path to Other m-files
addpath('.\functions\');
addpath('.\functions\DataFrame\');

%% Input Parameters
nDays = [31 28 31 30 31 30 31 31 30 31 30 31];
nDays_wd = [21 20 23 22 21 22 22 22 22 21 22 23];
nDays_we = [10 8 8 8 10 8 9 9 8 10 8 8];
nMonths = numel(nDays);

%% Generate Spatial Coordinates
x = -12400:200:12400;
y = -7400:200:7400;
[Y,X] = meshgrid(x,y);
nSpatialPoints = numel(x)*numel(y);
nDataPointsPerDay = nSpatialPoints*24;

%% Read Total Annual Average emissions NOx + PM10
% dissagregated at the URBAIR grid level
nCells = 9375;

% residential
fid = fopen('GEN_residential_grid_URBAIR.txt');
DATA = textscan(fid,'%f32%f32%f32','Headerlines',1);
fclose(fid);

FIDres = DATA{1};
NOxres = DATA{2};
PMres = DATA{3};

% commercial
fid = fopen('GEN_comercial_grid_URBAIR.txt');
```

```
DATA = textscan(fid,'%f32%f32%f32','Headerlines',1);
fclose(fid);

FIDcom = DATA{1};
NOxcom = DATA{2};
PMcom = DATA{3};

%
NOx = NOxres + NOxcom;
PM = PMres + PMcom;

%% Read ID with corresponding grid cells coordinates X and Y
fid = fopen('FID_XY.txt');
DATA = textscan(fid,'%f32%f32%f32','Headerlines',1);
fclose(fid);

FID = DATA{1};
Xid = DATA{2};
Yid = DATA{3};
%% Create a ID for all the grid cells with emission (emission > 0)
th = prctile(NOxres,30);
idx_keep_NOx = find(NOxres > th);
NOxresN = NOxres(idx_keep_NOx);

th = prctile(PMres,30);
idx_keep_PM = find(PMres > th);
PMresN = PMres(idx_keep_PM);

th = prctile(NOxcom,30);
idx_keep_NOxcom = find(NOxcom > th);
NOxcomN = NOxcom(idx_keep_NOxcom);

th = prctile(PMcom,30);
```

```
idx_keep_PMcom = find(PMcom > th);
PMcomN = PMcom(idx_keep_PMcom);

%%
% NOx Res + Com
th = 0.0079;
idx_keep_NOxtotal = find(NOx > th);
NOxN = NOx(idx_keep_NOxtotal);

% PM Res + Com
th = 0.0047;
idx_keep_PMtotal = find(PM > th);
PMN = PM(idx_keep_PMtotal);

%% Read Temporal Profiles
% Daily profiles - Distribution of the total annual emissions per each day of the year
fid = fopen('GEN_daily_profiles.txt');
DATA = textscan(fid,'%s%f32%f32%f32','Headerlines',1);
fclose(fid);

date2015 = DATA{1};
shareCom = DATA{2};
shareResNOx = DATA{3};
shareResPM = DATA{4};

for n = 1:numel(date2015)
%    date2015{n}(date2015{n}=='/') = '-';
   dd = date2015{n}(1:2);
   mm = date2015{n}(4:5);
   date2015{n}(1:2) = mm;
   date2015{n}(4:5) = dd;
end
[dayNum2015,dayName2015] = weekday(date2015);
```

```
date2010 = date2015;
for n = 1:numel(date2015)
   date2010{n}(end)= '0';
end
[dayNum2010,dayName2010] = weekday(date2010);

shareCom2010 = circshift(shareCom,-1);
shareResNOx2010 = circshift(shareResNOx,-1);
shareResPM2010 = circshift(shareResPM,-1);

%% Post-processing data
% 1st: To compute the daily emission for each FID cell
for d = 1:sum(sum(nDays))
   for c = 1:nCells
      NOx_com_grid_daily (d,c) = NOxcom(c).*shareCom2010(d)./100;
      PM_com_grid_daily (d,c) = PMcom(c).*shareCom2010(d)./100;
      NOx_res_grid_daily (d,c) = NOxres(c).*shareResNOx2010(d)./100;
      PM_res_grid_daily (d,c) = PMres(c).*shareResPM2010(d)./100;
   end
end

%%
% Hourly profiles - for the typical days
fid = fopen('GEN_hourly_profiles.txt');
DATA = textscan(fid,'%s%f32%f32%f32%f32%f32%f32%f32%f32%f32%f32%f32%f32',...
   'Headerlines',1);
fclose(fid);

%residential NOx
HHNOx_res_winter_wd = DATA{2};
HHNOx_res_winter_we = DATA{5};
HHNOx_res_summer_wd = DATA{8};
```

```
HHNOx_res_summer_we = DATA{11};

totalHHNOx_res_winter_wd = sum(HHNOx_res_winter_wd);
totalHHNOx_res_winter_we = sum(HHNOx_res_winter_we);
totalHHNOx_res_summer_wd = sum(HHNOx_res_summer_wd);
totalHHNOx_res_summer_we = sum(HHNOx_res_summer_we);

%residential PM
HHPM_res_winter_wd = DATA{3};
HHPM_res_winter_we = DATA{6};
HHPM_res_summer_wd = DATA{9};
HHPM_res_summer_we = DATA{12};

totalHHPM_res_winter_wd = sum(HHPM_res_winter_wd);
totalHHPM_res_winter_we = sum(HHPM_res_winter_we);
totalHHPM_res_summer_wd = sum(HHPM_res_summer_wd);
totalHHPM_res_summer_we = sum(HHPM_res_summer_we);

%commercial
HHCom_winter_wd = DATA{4};
HHCom_winter_we = DATA{7};
HHCom_summer_wd = DATA{10};
HHCom_summer_we = DATA{13};

totalHHCom_winter_wd = sum(HHCom_winter_wd);
totalHHCom_winter_we = sum(HHCom_winter_we);
totalHHCom_summer_wd = sum(HHCom_summer_wd);
totalHHCom_summer_we = sum(HHCom_summer_we);

%% share per hour per typical day normalized 100%
shareHourly_NOx_res_winter_wd = HHNOx_res_winter_wd .*100 ./ ...
   totalHHNOx_res_winter_wd;
shareHourly_NOx_res_winter_we = HHNOx_res_winter_we .*100 ./ ...
```

```
   totalHHNOx_res_winter_we;
shareHourly_NOx_res_summer_wd = HHNOx_res_summer_wd .*100 ./ ...
   totalHHNOx_res_summer_wd;
shareHourly_NOx_res_summer_we = HHNOx_res_summer_we .*100 ./ ...
   totalHHNOx_res_summer_we;

shareHourly_PM_res_winter_wd = HHPM_res_winter_wd .*100 ./ ...
   totalHHPM_res_winter_wd;
shareHourly_PM_res_winter_we = HHPM_res_winter_we .*100 ./ ...
   totalHHPM_res_winter_we;
shareHourly_PM_res_summer_wd = HHPM_res_summer_wd .*100 ./ ...
   totalHHPM_res_summer_wd;
shareHourly_PM_res_summer_we = HHPM_res_summer_we .*100 ./ ...
   totalHHPM_res_summer_we;

shareHourly_Com_winter_wd = HHCom_winter_wd .*100 ./ ...
   totalHHCom_winter_wd;
shareHourly_Com_winter_we = HHCom_winter_we .*100 ./ ...
   totalHHCom_winter_we;
shareHourly_Com_summer_wd = HHCom_summer_wd .*100 ./ ...
   totalHHCom_summer_wd;
shareHourly_Com_summer_we = HHCom_summer_we .*100 ./ ...
   totalHHCom_summer_we;

%%
% residential NOx
shareResNOx2010_rep = repmat(shareResNOx2010,1,24);
shareResNOx2010_per = permute(shareResNOx2010_rep,[2,1]);

shareHourly_NOx_res_winter_wd_rep = ...
   repmat(shareHourly_NOx_res_winter_wd,1,365);
shareHH_DD_NOxres_winter_wd = shareHourly_NOx_res_winter_wd_rep ./100 ...
   .* shareResNOx2010_per;
```

```
shareHourly_NOx_res_winter_we_rep = ...
   repmat(shareHourly_NOx_res_winter_we,1,365);
shareHH_DD_NOxres_winter_we = shareHourly_NOx_res_winter_we_rep ./100 ...
   .* shareResNOx2010_per;

shareHourly_NOx_res_summer_wd_rep = ...
   repmat(shareHourly_NOx_res_summer_wd,1,365);
shareHH_DD_NOxres_summer_wd = shareHourly_NOx_res_summer_wd_rep ./100 ...
   .* shareResNOx2010_per;

shareHourly_NOx_res_summer_we_rep = ...
   repmat(shareHourly_NOx_res_summer_we,1,365);
shareHH_DD_NOxres_summer_we = shareHourly_NOx_res_summer_we_rep ./100 ...
   .* shareResNOx2010_per;

% residential PM
shareResPM2010_rep = repmat(shareResPM2010,1,24);
shareResPM2010_per = permute(shareResPM2010_rep,[2,1]);

shareHourly_PM_res_winter_wd_rep = ...
   repmat(shareHourly_PM_res_winter_wd,1,365);
shareHH_DD_PMres_winter_wd = shareHourly_PM_res_winter_wd_rep ./100 ...
   .* shareResPM2010_per;

shareHourly_PM_res_winter_we_rep = ...
   repmat(shareHourly_PM_res_winter_we,1,365);
shareHH_DD_PMres_winter_we = shareHourly_PM_res_winter_we_rep ./100 ...
   .* shareResPM2010_per;

shareHourly_PM_res_summer_wd_rep = ...
   repmat(shareHourly_PM_res_summer_wd,1,365);
shareHH_DD_PMres_summer_wd = shareHourly_PM_res_summer_wd_rep ./100 ...
```

```
   .* shareResPM2010_per;

shareHourly_PM_res_summer_we_rep = ...
   repmat(shareHourly_PM_res_summer_we,1,365);
shareHH_DD_PMres_summer_we = shareHourly_PM_res_summer_we_rep ./100 ...
   .* shareResPM2010_per;

% commercial
shareCom2010_rep = repmat(shareCom2010,1,24);
shareCom2010_per = permute(shareCom2010_rep,[2,1]);

shareHourly_Com_winter_wd_rep = ...
   repmat(shareHourly_Com_winter_wd,1,365);
shareHH_DD_Com_winter_wd = shareHourly_Com_winter_wd_rep ./100 ...
   .* shareCom2010_per;

shareHourly_Com_winter_we_rep = ...
   repmat(shareHourly_Com_winter_we,1,365);
shareHH_DD_Com_winter_we = shareHourly_Com_winter_we_rep ./100 ...
   .* shareCom2010_per;

shareHourly_Com_summer_wd_rep = ...
   repmat(shareHourly_Com_summer_wd,1,365);
shareHH_DD_Com_summer_wd = shareHourly_Com_summer_wd_rep ./100 ...
   .* shareCom2010_per;

shareHourly_Com_summer_we_rep = ...
   repmat(shareHourly_Com_summer_we,1,365);
shareHH_DD_Com_summer_we = shareHourly_Com_summer_we_rep ./100 ...
   .* shareCom2010_per;

%%
% filter winter days from 2010:
```

```
k = 1;
for n = 1:365
   mm(n) = str2double(date2010{n}(1:2));
%    if mm>=10 | mm <=3
%        winterDays{k} = date2010{n};
%        k = k+1;
%    end
end
% [dayNumWinter2010,dayNameWinter2010] = weekday(winterDays);

cond_wd = dayNum2010 >=2 & dayNum2010 <=6;
cond_we = dayNum2010 <=1 | dayNum2010 >=7;

cond_winter = mm>=10 | mm <=3;
idx_winter_wd = find(cond_wd & cond_winter');
idx_winter_we = find(cond_we & cond_winter');

cond_summer = mm>=4 & mm <=9;
idx_summer_wd = find(cond_wd & cond_summer');
idx_summer_we = find(cond_we & cond_summer');

%% Daily To distribute emissions per week day and weekend for winter and summer
NOx_res_winter_wd = NOx_res_grid_daily(idx_winter_wd,:);
NOx_res_winter_we = NOx_res_grid_daily(idx_winter_we,:);
NOx_res_summer_wd = NOx_res_grid_daily(idx_summer_wd,:);
NOx_res_summer_we = NOx_res_grid_daily(idx_summer_we,:);

% Sum up of sum(sum(NOx_res_winter_wd)) + sum(sum(NOx_res_winter_we))
% sum(sum(NOx_res_summer_wd)) + sum(sum(NOx_res_summer_we))
% it's equal to sum(NOxres) - total annual value

PM_res_winter_wd = PM_res_grid_daily(idx_winter_wd,:);
PM_res_winter_we = PM_res_grid_daily(idx_winter_we,:);
```

```
PM_res_summer_wd = PM_res_grid_daily(idx_summer_wd,:);
PM_res_summer_we = PM_res_grid_daily(idx_summer_we,:);

NOx_com_winter_wd = NOx_com_grid_daily(idx_winter_wd,:);
NOx_com_winter_we = NOx_com_grid_daily(idx_winter_we,:);
NOx_com_summer_wd = NOx_com_grid_daily(idx_summer_wd,:);
NOx_com_summer_we = NOx_com_grid_daily(idx_summer_we,:);

PM_com_winter_wd = PM_com_grid_daily(idx_winter_wd,:);
PM_com_winter_we = PM_com_grid_daily(idx_winter_we,:);
PM_com_summer_wd = PM_com_grid_daily(idx_summer_wd,:);
PM_com_summer_we = PM_com_grid_daily(idx_summer_we,:);

%% Share of each hour for each day of the year depending on the
% classification winter/ summer wd/we
share_NOxres_winter_wd = shareHH_DD_NOxres_winter_wd (:,idx_winter_wd);
share_NOxres_winter_we = shareHH_DD_NOxres_winter_we (:,idx_winter_we);
share_NOxres_summer_wd = shareHH_DD_NOxres_summer_wd (:,idx_summer_wd);
share_NOxres_summer_we = shareHH_DD_NOxres_summer_we (:,idx_summer_we);

share_PMres_winter_wd = shareHH_DD_PMres_winter_wd (:,idx_winter_wd);
share_PMres_winter_we = shareHH_DD_PMres_winter_we (:,idx_winter_we);
share_PMres_summer_wd = shareHH_DD_PMres_summer_wd (:,idx_summer_wd);
share_PMres_summer_we = shareHH_DD_PMres_summer_we (:,idx_summer_we);

share_NOxcom_winter_wd = shareHH_DD_Com_winter_wd (:,idx_winter_wd);
share_NOxcom_winter_we = shareHH_DD_Com_winter_we (:,idx_winter_we);
share_NOxcom_summer_wd = shareHH_DD_Com_summer_wd (:,idx_summer_wd);
share_NOxcom_summer_we = shareHH_DD_Com_summer_we (:,idx_summer_we);

share_PMcom_winter_wd = shareHH_DD_Com_winter_wd (:,idx_winter_wd);
share_PMcom_winter_we = shareHH_DD_Com_winter_we (:,idx_winter_we);
share_PMcom_summer_wd = shareHH_DD_Com_summer_wd (:,idx_summer_wd);
```

```
share_PMcom_summer_we = shareHH_DD_Com_summer_we (:,idx_summer_we);

%% hourly To distribute emissions per week day and weekend for winter and summer
% NOx residential
NOx_res_winter_wd_H = repmat(NOx_res_winter_wd,1,1,24);
NOx_res_winter_wd_H = permute(NOx_res_winter_wd_H,[3,1,2]);

share_NOxres_winter_wd                                                    =
share_NOxres_winter_wd./repmat(sum(share_NOxres_winter_wd,1),24,1);
shareHourly_NOx_res_winter_wd_H = ...
   repmat(share_NOxres_winter_wd,1,1,nCells);
NOx_res_winter_wd_HH = NOx_res_winter_wd_H .* ...
   shareHourly_NOx_res_winter_wd_H;

NOx_res_winter_we_H = repmat(NOx_res_winter_we,1,1,24);
NOx_res_winter_we_H = permute(NOx_res_winter_we_H,[3,1,2]);
share_NOxres_winter_we                                                    =
share_NOxres_winter_we./repmat(sum(share_NOxres_winter_we,1),24,1);
shareHourly_NOx_res_winter_we_H = ...
   repmat(share_NOxres_winter_we,1,1,nCells);
NOx_res_winter_we_HH = NOx_res_winter_we_H .* ...
   shareHourly_NOx_res_winter_we_H;

NOx_res_summer_wd_H = repmat(NOx_res_summer_wd,1,1,24);
NOx_res_summer_wd_H = permute(NOx_res_summer_wd_H,[3,1,2]);
share_NOxres_summer_wd                                                    =
share_NOxres_summer_wd./repmat(sum(share_NOxres_summer_wd,1),24,1);
shareHourly_NOx_res_summer_wd_H = ...
   repmat(share_NOxres_summer_wd,1,1,nCells);
NOx_res_summer_wd_HH = NOx_res_summer_wd_H .* ...
   shareHourly_NOx_res_summer_wd_H;

NOx_res_summer_we_H = repmat(NOx_res_summer_we,1,1,24);
NOx_res_summer_we_H = permute(NOx_res_summer_we_H,[3,1,2]);
```

```
share_NOxres_summer_we =
share_NOxres_summer_we./repmat(sum(share_NOxres_summer_we,1),24,1);
shareHourly_NOx_res_summer_we_H = ...
   repmat(share_NOxres_summer_we,1,1,nCells);
NOx_res_summer_we_HH = NOx_res_summer_we_H .* ...
   shareHourly_NOx_res_summer_we_H;

% Final ordering per consecutive day of the year
% *1e6 to transform Mg into g
% /3600 to transform g/h into g/s
% *200*200 to transform g/s into an emission flux in g/(m2.s)
NOx_res = zeros(24,365,nCells);
NOx_res(:,idx_summer_wd,:) = NOx_res_summer_wd_HH .*1e6/(3600*200*200);
NOx_res(:,idx_summer_we,:) = NOx_res_summer_we_HH .*1e6/(3600*200*200);
NOx_res(:,idx_winter_wd,:) = NOx_res_winter_wd_HH .*1e6/(3600*200*200);
NOx_res(:,idx_winter_we,:) = NOx_res_winter_we_HH .*1e6/(3600*200*200);

% PM residential
PM_res_winter_wd_H = repmat(PM_res_winter_wd,1,1,24);
PM_res_winter_wd_H = permute(PM_res_winter_wd_H,[3,1,2]);

share_PMres_winter_wd =
share_PMres_winter_wd./repmat(sum(share_PMres_winter_wd,1),24,1);
shareHourly_PM_res_winter_wd_H = ...
   repmat(share_PMres_winter_wd,1,1,nCells);
PM_res_winter_wd_HH = PM_res_winter_wd_H .* ...
   shareHourly_PM_res_winter_wd_H;

PM_res_winter_we_H = repmat(PM_res_winter_we,1,1,24);
PM_res_winter_we_H = permute(PM_res_winter_we_H,[3,1,2]);
share_PMres_winter_we =
share_PMres_winter_we./repmat(sum(share_PMres_winter_we,1),24,1);
shareHourly_PM_res_winter_we_H = ...
   repmat(share_PMres_winter_we,1,1,nCells);
```

```
PM_res_winter_we_HH = PM_res_winter_we_H .* ...
   shareHourly_PM_res_winter_we_H;

PM_res_summer_wd_H = repmat(PM_res_summer_wd,1,1,24);
PM_res_summer_wd_H = permute(PM_res_summer_wd_H,[3,1,2]);
share_PMres_summer_wd                                                    =
share_PMres_summer_wd./repmat(sum(share_PMres_summer_wd,1),24,1);
shareHourly_PM_res_summer_wd_H = ...
   repmat(share_PMres_summer_wd,1,1,nCells);
PM_res_summer_wd_HH = PM_res_summer_wd_H .* ...
   shareHourly_PM_res_summer_wd_H;

PM_res_summer_we_H = repmat(PM_res_summer_we,1,1,24);
PM_res_summer_we_H = permute(PM_res_summer_we_H,[3,1,2]);
share_PMres_summer_we                                                    =
share_PMres_summer_we./repmat(sum(share_PMres_summer_we,1),24,1);
shareHourly_PM_res_summer_we_H = ...
   repmat(share_PMres_summer_we,1,1,nCells);
PM_res_summer_we_HH = PM_res_summer_we_H .* ...
   shareHourly_PM_res_summer_we_H;

% Final ordering per consecutive day of the year
PM_res = zeros(24,365,nCells);
PM_res(:,idx_summer_wd,:) = PM_res_summer_wd_HH .*1e6/(3600*200*200);
PM_res(:,idx_summer_we,:) = PM_res_summer_we_HH .*1e6/(3600*200*200);
PM_res(:,idx_winter_wd,:) = PM_res_winter_wd_HH .*1e6/(3600*200*200);
PM_res(:,idx_winter_we,:) = PM_res_winter_we_HH .*1e6/(3600*200*200);

%% Commercial
%hourly To distribute emissions per week day and weekend for winter and summer
% NOx

NOx_com_winter_wd_H = repmat(NOx_com_winter_wd,1,1,24);
NOx_com_winter_wd_H = permute(NOx_com_winter_wd_H,[3,1,2]);
```

```
share_NOxcom_winter_wd                                                    =
share_NOxcom_winter_wd./repmat(sum(share_NOxcom_winter_wd,1),24,1);
shareHourly_NOx_com_winter_wd_H = ...
   repmat(share_NOxcom_winter_wd,1,1,nCells);
NOx_com_winter_wd_HH = NOx_com_winter_wd_H .* ...
   shareHourly_NOx_com_winter_wd_H;

NOx_com_winter_we_H = repmat(NOx_com_winter_we,1,1,24);
NOx_com_winter_we_H = permute(NOx_com_winter_we_H,[3,1,2]);

share_NOxcom_winter_we                                                    =
share_NOxcom_winter_we./repmat(sum(share_NOxcom_winter_we,1),24,1);
shareHourly_NOx_com_winter_we_H = ...
   repmat(share_NOxcom_winter_we,1,1,nCells);
NOx_com_winter_we_HH = NOx_com_winter_we_H .* ...
   shareHourly_NOx_com_winter_we_H;

NOx_com_summer_wd_H = repmat(NOx_com_summer_wd,1,1,24);
NOx_com_summer_wd_H = permute(NOx_com_summer_wd_H,[3,1,2]);

share_NOxcom_summer_wd                                                    =
share_NOxcom_summer_wd./repmat(sum(share_NOxcom_summer_wd,1),24,1);
shareHourly_NOx_com_summer_wd_H = ...
   repmat(share_NOxcom_summer_wd,1,1,nCells);
NOx_com_summer_wd_HH = NOx_com_summer_wd_H .* ...
   shareHourly_NOx_com_summer_wd_H;

NOx_com_summer_we_H = repmat(NOx_com_summer_we,1,1,24);
NOx_com_summer_we_H = permute(NOx_com_summer_we_H,[3,1,2]);

share_NOxcom_summer_we                                                    =
share_NOxcom_summer_we./repmat(sum(share_NOxcom_summer_we,1),24,1);
shareHourly_NOx_com_summer_we_H = ...
   repmat(share_NOxcom_summer_we,1,1,nCells);
```

```
NOx_com_summer_we_HH = NOx_com_summer_we_H .* ...
   shareHourly_NOx_com_summer_we_H;

% Final ordering per consecutive day of the year
% *1e6 to transform Mg into g
% /3600 to transform g/h into g/s
% *200*200 to transform g/s into an emission flux in g/(m2.s)
% 200 is the horizontal grid resolution
NOx_com = zeros(24,365,nCells);
NOx_com(:,idx_summer_wd,:) = NOx_com_summer_wd_HH .*1e6/(3600*200*200);
NOx_com(:,idx_summer_we,:) = NOx_com_summer_we_HH .*1e6/(3600*200*200);
NOx_com(:,idx_winter_wd,:) = NOx_com_winter_wd_HH .*1e6/(3600*200*200);
NOx_com(:,idx_winter_we,:) = NOx_com_winter_we_HH .*1e6/(3600*200*200);

% PM residential
PM_com_winter_wd_H = repmat(PM_com_winter_wd,1,1,24);
PM_com_winter_wd_H = permute(PM_com_winter_wd_H,[3,1,2]);

share_PMcom_winter_wd                                                    =
share_PMcom_winter_wd./repmat(sum(share_PMcom_winter_wd,1),24,1);
shareHourly_PM_com_winter_wd_H = ...
   repmat(share_PMcom_winter_wd,1,1,nCells);
PM_com_winter_wd_HH = PM_com_winter_wd_H .* ...
   shareHourly_PM_com_winter_wd_H;

PM_com_winter_we_H = repmat(PM_com_winter_we,1,1,24);
PM_com_winter_we_H = permute(PM_com_winter_we_H,[3,1,2]);

share_PMcom_winter_we                                                    =
share_PMcom_winter_we./repmat(sum(share_PMcom_winter_we,1),24,1);
shareHourly_PM_com_winter_we_H = ...
   repmat(share_PMcom_winter_we,1,1,nCells);
PM_com_winter_we_HH = PM_com_winter_we_H .* ...
   shareHourly_PM_com_winter_we_H;
```

```
PM_com_summer_wd_H = repmat(PM_com_summer_wd,1,1,24);
PM_com_summer_wd_H = permute(PM_com_summer_wd_H,[3,1,2]);

share_PMcom_summer_wd                                                    =
share_PMcom_summer_wd./repmat(sum(share_PMcom_summer_wd,1),24,1);
shareHourly_PM_com_summer_wd_H = ...
   repmat(share_PMcom_summer_wd,1,1,nCells);
PM_com_summer_wd_HH = PM_com_summer_wd_H .* ...
   shareHourly_PM_com_summer_wd_H;

PM_com_summer_we_H = repmat(PM_com_summer_we,1,1,24);
PM_com_summer_we_H = permute(PM_com_summer_we_H,[3,1,2]);

share_PMcom_summer_we                                                    =
share_PMcom_summer_we./repmat(sum(share_PMcom_summer_we,1),24,1);
shareHourly_PM_com_summer_we_H = ...
   repmat(share_PMcom_summer_we,1,1,nCells);
PM_com_summer_we_HH = PM_com_summer_we_H .* ...
   shareHourly_PM_com_summer_we_H;

% Final ordering per consecutive day of the year
PM_com = zeros(24,365,nCells);
PM_com(:,idx_summer_wd,:) = PM_com_summer_wd_HH .*1e6/(3600*200*200);
PM_com(:,idx_summer_we,:) = PM_com_summer_we_HH .*1e6/(3600*200*200);
PM_com(:,idx_winter_wd,:) = PM_com_winter_wd_HH .*1e6/(3600*200*200);
PM_com(:,idx_winter_we,:) = PM_com_winter_we_HH .*1e6/(3600*200*200);

%% To write the URBAIR input file - "urbair_input.inp"
% ID  Tp1F  Tp2F  Tp3F  X0F  y0F  z0F  dxF  dyF  angF  qF  velF  TpF  pF
% ID  '2'  '1'  '1'  x0F  y0F  '0.0'  '200.0'  '200.0'  '0.0'  '1.0'  '0.1'
%  '298'  '101325'

% 1 file for residential NOx
```

```
% 1 file for residential PM
% 1 file for commercial NOx
% 1 file for commercial NOx

% remove positions below the threshold:
Xid_NOx = Xid(idx_keep_NOxtotal);
Yid_NOx = Yid(idx_keep_NOxtotal);
fileID = fopen('urbair_input_NOx.txt','w');
for n = 1:numel(idx_keep_NOxtotal)
   fprintf(fileID, '%d %s %s %s %1.1f %1.1f %s %s %s %s %s %s %s %s \n',...
      n,'2','1','1',Xid_NOx(n),Yid_NOx(n),...
      '0.0','200.0','200.0',...
      '0.0','1.0','0.1','298','101325');
end
fclose(fileID);

Xid_PM = Xid(idx_keep_PMtotal);
Yid_PM = Yid(idx_keep_PMtotal);
fileID = fopen('urbair_input_PM.txt','w');
for n = 1:numel(idx_keep_PMtotal)
   fprintf(fileID, '%d %s %s %s %1.1f %1.1f %s %s %s %s %s %s %s %s \n',...
      n,'2','1','1',Xid_PM(n),Yid_PM(n),...
      '0.0','200.0','200.0',...
      '0.0','1.0','0.1','298','101325');
end
fclose(fileID);

%% To write the URBAIR input file - "emissoes_variaveis.dat"
%year month day hour min sec ID1 ID2 q vel Tp P
%'2015' month day hour '0' '0' ID1 ID1 q '0.1' '298' '101325.0'
nDays_sum = cumsum(nDays);
nDays_sum = [0 nDays_sum];
```

```
for m = 1:nMonths
   NOxtotal = NOx_com(:,nDays_sum(m)+1:nDays_sum(m+1),:) + ...
     NOx_res(:,nDays_sum(m)+1:nDays_sum(m+1),:);
   fileID = fopen(['IRCI_NOx',num2str(m),'.txt'],'w');

fprintf(fileID,'**********************************************************************
**********************\n');
   fprintf(fileID,' URBAIR VARIABLE          EMISSIONS       INPUT   DATA    FILE\n');

fprintf(fileID,'**********************************************************************
**********************\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------\n');
   fprintf(fileID,'year    month  day    hour    min    sec    ID1    ID2    q        vel      Tp
         P\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------\n');
   for d = 1:nDays(m)
      for h = 0:23
         for ID = 1:numel(idx_keep_NOxtotal)
            fprintf(fileID, '%s %d %d %d %s %s %d %d %15e %s %s %s \n','2015',m,d,h,...
               '0','0',ID,ID, ...
               NOxtotal(h+1,d,idx_keep_NOxtotal(ID)),'0.1','298','101325.0');
         end
      end
   end
   fprintf(fileID,'\n')

fprintf(fileID,'**********************************************************************
**********************\n')
   fclose(fileID);
end

for m = 1:nMonths
   PMtotal = PM_com(:,nDays_sum(m)+1:nDays_sum(m+1),:) + ...
     PM_res(:,nDays_sum(m)+1:nDays_sum(m+1),:);
   fileID = fopen(['IRCI_PM',num2str(m),'.txt'],'w');
```

```
fprintf(fileID,'*****************************************************************************
**********************\n');
   fprintf(fileID,' URBAIR VARIABLE          EMISSIONS        INPUT   DATA    FILE\n');

fprintf(fileID,'*****************************************************************************
**********************\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------------\n');
   fprintf(fileID,'year     month  day     hour     min      sec      ID1      ID2      q          vel        Tp
         P\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------------\n');
   for d = 1:nDays(m)
      for h = 0:23
         for ID = 1:numel(idx_keep_PMtotal)
            fprintf(fileID, '%s %d %d %d %s %s %d %d %15e %s %s %s \n','2015',m,d,h,...
               '0','0',ID,ID,PMtotal(h+1,d,idx_keep_PMtotal(ID)),'0.1','298','101325.0');
         end
      end
   end
   fprintf(fileID,'\n')

fprintf(fileID,'*****************************************************************************
**********************\n')
   fclose(fileID);
end
```

Other area emission sources preprocessor in Matlab:

```
%% Add Path to Other m-files
addpath('.\functions\');
addpath('.\functions\DataFrame\');

%% Input Parameters
nDays = [31 28 31 30 31 30 31 31 30 31 30 31];
nDays_wd = [21 20 23 22 21 22 22 22 22 21 22 23];
nDays_we = [10 8 8 8 10 8 9 9 8 10 8 8];
nMonths = numel(nDays);
% industrial AREA
fid = fopen('CIRA_indAREA.txt');
DATA = textscan(fid,'%f32%f32%f32','Headerlines',1);
fclose(fid);

FIDres = DATA{1};
NOxres = DATA{2};
PMres = DATA{3};

%% Read ID with corresponding grid cells coordinates X and Y
fid = fopen('FID_XY.txt');
DATA = textscan(fid,'%f32%f32%f32','Headerlines',1);
fclose(fid);

FID = DATA{1};
Xid = DATA{2};
Yid = DATA{3};
%% Create a ID for all the grid cells with emission (emission > 0)
th = 0;
idx_keep_NOx = find(NOxres > th);
NOxresN = NOxres(idx_keep_NOx);

th = 0;
```

```
idx_keep_PM = find(PMres > th);
PMresN = PMres(idx_keep_PM);

 %% To write the URBAIR input file - "urbair_input.inp"
% ID  Tp1F  Tp2F  Tp3F  X0F  y0F  z0F  dxF  dyF  angF  qF  velF  TpF  pF
% ID  '2'  '1'  '1'  x0F  y0F  '0.0'  '200.0'  '200.0'  '0.0'  '1.0'  '0.1'
%  '298'  '101325'



% % remove positions below 0:
Xid_NOx_res = Xid(idx_keep_NOx);
Yid_NOx_res = Yid(idx_keep_NOx);
FID_NOx_res = 1:numel(idx_keep_NOx);
fileID = fopen('urbair_input_NOx.txt','w');
for n = 1:numel(FID_NOx_res)
   fprintf(fileID, '%d %s %s %s %1.1f %1.1f %s %s %s %s %s %s %s %s \n',...
      FID_NOx_res(n),'2','1','1',Xid_NOx_res(n),Yid_NOx_res(n),...
      '100.0','200.0','200.0',...
      '0.0','1.0','0.1','298','101325');
end
fclose(fileID);

Xid_PM_res = Xid(idx_keep_PM);
Yid_PM_res = Yid(idx_keep_PM);
FID_PM_res = 1:numel(idx_keep_PM);
fileID = fopen('urbair_input_PM.txt','w');
for n = 1:numel(FID_PM_res)
   fprintf(fileID, '%d %s %s %s %1.1f %1.1f %s %s %s %s %s %s %s %s \n',...
      FID_PM_res(n),'2','1','1',Xid_PM_res(n),Yid_PM_res(n),...
      '100.0','200.0','200.0',...
      '0.0','1.0','0.1','298','101325');
end
fclose(fileID);
```

```
%% To write the URBAIR input file - "emissoes_variaveis.dat"
%year  month  day  hour  min  sec  ID1  ID2  q  vel  Tp  P
%'2015' month day hour '0' '0' ID1 ID1 q '0.1' '298' '101325.0'
nDays_sum = cumsum(nDays);
nDays_sum = [0 nDays_sum];

NOxR = repmat(NOxres,1,365,24);
NOxR = permute(NOxR,[3,2,1]);
NOxRarea = NOxR/(400*400);



for m = 1:nMonths
   fileID = fopen(['indAREA_NOx',num2str(m),'.txt'],'w');

fprintf(fileID,'**********************************************************************
**********************\n');
   fprintf(fileID,' URBAIR VARIABLE          EMISSIONS        INPUT   DATA    FILE\n');

fprintf(fileID,'**********************************************************************
**********************\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------\n');
   fprintf(fileID,'year    month  day    hour    min    sec    ID1    ID2    q          vel        Tp
         P\n');
   fprintf(fileID,'-------------------------------------------------------------------------------------\n');
   for d = 1:nDays(m)
      for h = 0:23
         for ID = 1:numel(FID_NOx_res)
            fprintf(fileID, '%s %d %d %d %s %s %d %d %15e %s %s %s \n','2015',m,d,h,...
               '0','0',ID,ID,NOxRarea(h+1,d,idx_keep_NOx(ID)),'0.1','298','101325.0');
         end
      end
   end
    fprintf(fileID,'\n')
```

```
fprintf(fileID,'*****************************************************************************
**********************\n');
    fclose(fileID);
end

PMR = repmat(PMres,1,365,24);
PMR = permute(PMR,[3,2,1]);
PMRarea = PMR/(400*400);

for m = 1:nMonths
  fileID = fopen(['indAREA_PM',num2str(m),'.txt'],'w');

fprintf(fileID,'*****************************************************************************
**********************\n');
  fprintf(fileID,' URBAIR VARIABLE          EMISSIONS        INPUT   DATA    FILE\n');

fprintf(fileID,'*****************************************************************************
**********************\n');
  fprintf(fileID,'-------------------------------------------------------------------------------------------\n');
  fprintf(fileID,'year    month  day    hour    min    sec    ID1    ID2    q          vel        Tp
        P\n');
  fprintf(fileID,'-------------------------------------------------------------------------------------------\n');
  for d = 1:nDays(m)
    for h = 0:23
      for ID = 1:numel(FID_PM_res)
        fprintf(fileID, '%s %d %d %d %s %s %d %d %15e %s %s %s \n','2015',m,d,h,...

'0','0',FID_PM_res(ID),FID_PM_res(ID),PMRarea(h+1,d,idx_keep_PM(ID)),'0.1','298','101325.0');
      end
    end
  end
  fprintf(fileID,'\n')

fprintf(fileID,'*****************************************************************************
**********************\n')
```

```
    fclose(fileID);
end
```

Post processor in Matlab:

```
clear all;
close all;
clc;

%% Add Path to Other m-files
addpath('.\functions\');

%% Input Parameters
nDays = [31 28 31 30 31 30 31 31 30 31 30 31];
nDays_wd = [21 20 23 22 21 22 22 22 22 21 22 23];
nDays_we = [10 8 8 8 10 8 9 9 8 10 8 8];
nMonths = numel(nDays);

%% Generate Spatial Coordinates
x = -20000:400:19600;
y = -27600:400:27200;
[X,Y] = meshgrid(x,y);
nSpatialPoints = numel(x)*numel(y);
nDataPointsPerDay = nSpatialPoints*24;

%% Read Data for Each Month
for n = 1:12 %1:nMonths
   % from Industrial/Area Sector:
   C{n}.indArea = readMonthlyData(['\_matDATAindArea',...
      num2str(n,'%02d'),'\'],n,nDays(n),nSpatialPoints);
%    % from Industrial/Point:
   C{n}.indPoint = readMonthlyData(['\_matDATAindPoint',...
      num2str(n,'%02d'),'\'],n,nDays(n),nSpatialPoints);
   % from residential and commercial sector:
   C{n}.irci = readMonthlyData([...
      '\_matDATAirci',num2str(n,'%02d'),'\'],n,...
      nDays(n),nSpatialPoints);
```

```
   % from transport Sector week days:
   C{n}.transp = readMonthlyData([...
      '\_matDATAtransp',num2str(n,'%02d'),'\'],n,...
      nDays(n),nSpatialPoints);
   % from transport Sector weekend :
end

%% Save Data
save('CIRA_NOx_bysector','C','X','Y','x','y','nDays');
```

The representation post-processor code from matlab:

```
%% Load Data - change for each case study
fileName = 'CIRA_NOx_bysector';
load([fileDir fileName]);

S = shaperead('CIRA_D3.shp');
x = 25200:400:64800;
y = 187700:400:242500;

[X,Y] = meshgrid(x,y);
%% Concentration
for m = 1:12 %1:numel(nDays)
    meanC_XY(m,:,:) = squeeze(mean(mean(C{m}.indArea + C{m}.indPoint + ...
       C{m}.irci + C{m}.transp,1),2));
   meanC_indArea(m,:,:) = squeeze(mean(mean(C{m}.indArea,1),2));
   meanC_indPoint(m,:,:) = squeeze(mean(mean(C{m}.indPoint,1),2));
   meanC_irci(m,:,:) = squeeze(mean(mean(C{m}.irci,1),2));
   meanC_transp(m,:,:) = squeeze(mean(mean(C{m}.transp,1),2));
end
% Calculates the annual mean
CYY = squeeze(mean(meanC_XY,1));
CYY_trs = squeeze(mean(meanC_transp,1));
```

```
CYia = squeeze(mean(meanC_indArea,1));
CYip = squeeze(mean(meanC_indPoint,1));
CYY_irci = squeeze(mean(meanC_irci,1));

CYY_ind = CYia + CYip;

%% Load data - Change for each case study
% Background concentrations
Cback = 0.1;
C_back = repmat(Cback,100,138);
Ctotal = CYY + C_back;

%%All measured data
fid = fopen('CIRA_NO2_2015.txt');
DATA = textscan(fid,'%f32%f32%f32%f32','Headerlines',1);
X_UTM = DATA{1};
Y_UTM = DATA{2};
Cpt = DATA{3};
fclose(fid);

%% Model adjustment procedure
%%All AQS (no distinction by type): slope1
for n=1:numel(Cpt)
   [~,idx_x] = min(abs(x-X_UTM(n)));
   X_UTM2(n) = x(idx_x);
   [~,idx_y] = min(abs(y-Y_UTM(n)));
   Y_UTM2(n) = y(idx_y);
   idx_x_sta(n)=idx_x;
   idx_y_sta(n)=idx_y;
   x_sta = x(idx_x_sta);
   y_sta = y(idx_y_sta);
   Cbackground(n) = C_back(idx_x,idx_y);
   Cmodel(n) = Ctotal(idx_x,idx_y);
```

```
   Cmonitor(n) = Cpt(n) - Cbackground(n);
end
slope1 = Cmodel(:)\Cmonitor(:);

%%Adjusted concentrations + Background
%Ajusted with unique slope
Cind_adj_slope1 = CYY_ind*slope1;
Ctrs_adj_slope1 = CYY_trs*slope1;
Circi_adj_slope1 = CYY_irci*slope1;
Cadjusted_slope1_woutback = Cind_adj_slope1 + Ctrs_adj_slope1 + Circi_adj_slope1;
Cadjusted_slope1 = Cadjusted_slope1_woutback + C_back;

%% Results analysis - contributions, min/max,mean, station location analysis, etc...
%Max,Min and Mean for each approach
max_domain = max(max(Cadjusted_slope1));
min_domain = min(min(Cadjusted_slope1));
mean_domain = mean(mean(Cadjusted_slope1));

%Sector contribution for whole domain
Ctr_slope1_t= mean(mean(Ctrs_adj_slope1))/mean(mean(Cadjusted_slope1_woutback))*100;
Ctr_slope1_i= mean(mean(Cind_adj_slope1))/mean(mean(Cadjusted_slope1_woutback))*100;
Ctr_slope1_irci= mean(mean(Circi_adj_slope1))/mean(mean(Cadjusted_slope1_woutback))*100;
%Sector contribution for max value
[i_max,j_max] = find(Cadjusted_slope1 == max(max(Cadjusted_slope1)));
Ctr_TRS_max= Ctrs_adj_slope1(i_max,j_max)/Cadjusted_slope1_woutback(i_max,j_max)*100;
Ctr_IND_max= Cind_adj_slope1(i_max,j_max)/Cadjusted_slope1_woutback(i_max,j_max)*100;
Ctr_IRCI_max= Circi_adj_slope1(i_max,j_max)/Cadjusted_slope1_woutback(i_max,j_max)*100;
%Sector contribution for min value
[i_min,j_min] = find(Cadjusted_slope1 == min(min(Cadjusted_slope1)));
Ctr_TRS_min= Ctrs_adj_slope1(i_min,j_min)/Cadjusted_slope1_woutback(i_min,j_min)*100;
Ctr_IND_min= Cind_adj_slope1(i_min,j_min)/Cadjusted_slope1_woutback(i_min,j_min)*100;
Ctr_IRCI_min= Circi_adj_slope1(i_min,j_min)/Cadjusted_slope1_woutback(i_min,j_min)*100;
%Max value for each sector
```

```
[i_max_t,j_max_t] = find(Ctrs_adj_slope1 == max(max(Ctrs_adj_slope1)));
[i_min_t,j_min_t] = find(Ctrs_adj_slope1 == min(min(Ctrs_adj_slope1)));
x_t = x(i_max_t);
y_t = y(j_max_t);
[i_max_i,j_max_i] = find(Cind_adj_slope1 == max(max(Cind_adj_slope1)));
[i_min_i,j_min_i] = find(Cind_adj_slope1 == min(min(Cind_adj_slope1)));
x_i = x(i_max_i);
y_i = y(j_max_i);
[i_max_rc,j_max_rc] = find(Circi_adj_slope1 == max(max(Circi_adj_slope1)));
[i_min_rc,j_min_rc] = find(Circi_adj_slope1 == min(min(Circi_adj_slope1)));
x_rc = x(i_max_rc);
y_rc = y(j_max_rc);

for n=1:numel(Cpt)
   Ctr_TRS_sta(n) = Ctrs_adj_slope1(idx_x_sta(n),idx_y_sta(n))/Cadjusted_slope1_woutback(idx_x_sta(n),idx_y_sta(n))*100;
   Ctr_IND_sta(n) = Cind_adj_slope1(idx_x_sta(n),idx_y_sta(n))/Cadjusted_slope1_woutback(idx_x_sta(n),idx_y_sta(n))*100;
   Ctr_IRCI_sta(n) = Circi_adj_slope1(idx_x_sta(n),idx_y_sta(n))/Cadjusted_slope1_woutback(idx_x_sta(n),idx_y_sta(n))*100;
   Conc_slope1_sta(n) = Cadjusted_slope1(idx_x_sta(n),idx_y_sta(n));

end

%% Air quality contours
%Load Color Map
temp = load('cmap');
myColorMap = temp.cMap;

for n = 1:numel(S)
   S(n).X = S(n).X * 1e-3;
   S(n).Y = S(n).Y * 1e-3;
```

```
end

% Coordinates conversion
x = x * 1e-3;
y = y * 1e-3;
X_UTM = X_UTM * 1e-3;
Y_UTM = Y_UTM * 1e-3;

% Contour
hFig1 = figure(1);
mapshow(S,'FaceColor', [1 1 1]);
hold on;
scatter(x_sta.*1e-3,y_sta.*1e-3,70,Cpt,'o','filled','MarkerEdgeColor',[0 0 0],'LineWidth',1)
% hplot = imagesc(x,y,(Cadjusted_slope1'));
% set(hplot,'alphadata',0.8);
%%Plot maximum by sector
% plot(x(i_max_t),y(j_max_t),'ok','MarkerSize',7)
% plot(x(i_max_s),y(j_max_s),'+k','MarkerSize',7)
% plot(x(i_max_i),y(j_max_i),'sk','MarkerSize',7)
% plot(x(i_max_rc),y(j_max_rc),'*k','MarkerSize',7)
% legend('Transport','Shipping','Industrial','Com and Res')
%%Plot das célula máxima e da mínima
% plot(x(i_max),y(j_max),'ok','LineWidth',2,'MarkerSize',6)
% plot(x(i_min),y(j_min),'om','LineWidth',2,'MarkerSize',6)
% legend('Max','Min')
% legend('Max')
%%Plot AQ Stations
% plot(x(idx_x_sta),y(idx_y_sta),'ok','Markersize',6)
%
text(x_sta,y_sta,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','Color','black','FontSize',6)

cMap = myColorMap;
colormap(cMap);
```

```
caxis([0 40]);
h = colorbar;
xlabel('West-East (km)','Interpreter','Latex','fontsize',12);
ylabel('South-North (km)','Interpreter','Latex','fontsize',12);
set(get(h,'label'),'string','NO$_2$ concentrations ($\mu$g.m$^{-3}$)',...
   'Interpreter','Latex','fontsize',12);
set(get(gca,'xaxis'),'TickLabelInterpreter','latex', ...
   'TickLabelFormat','%.0f','fontsize',12,'exponent',0);
set(get(gca,'yaxis'),'TickLabelInterpreter','latex', ...
   'TickLabelFormat','%.0f','fontsize',12,'exponent',0);
set(gca,'xTick',min(x):4:max(x));
set(gca,'yTick',min(y):4:max(y));
set(h,'TickLabelInterpreter','latex','fontsize',12);
set(h,'Limits',[0 40]);
set(gca,'Box','on')
axis([min(x) max(x) min(y) max(y)]);
hold off;


% Export Figure
figFolder = '.\_figures\';
figName = 'CIRA_NO2_2015_station';
set(hFig1,'color','none');
export_fig(hFig1,[figFolder figName],'-png','-r600');
```

4.4 Health module Pyhton code

```
# -*- coding: utf-8 -*-
"""
Script to calculate impacts associated with NO2 from:
   - Ljubljana
   - baseline file

"""
import os
import pandas as pd
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import seaborn as sns
from math import log

#
# USER NEEDS TO CHANGE THESE VARIABLES ACCORDINGLY"
#

# name of the city/area included in the name of the health data file: *area*_health_analysis.xlsx
# t will also be used for creating output filename
area = 'CIRA'

# name of the city/area included in the name of the AQ model data files
areaLong = 'CIRA'

# variable name in file name; it will also be used for creating output filename
varCnc = 'NO2'

# AQ data directory
aq_dir = r'c:\Users\jos\projects\ClairCity\data\AQ\\'+ areaLong + '\\'
# Health statistis data
health_dir = r'c:\Users\jos\projects\ClairCity\data\health_stats\\'

# Output directory
save_dir  = r'C:\Users\jos\projects\ClairCity\results\\'+ areaLong + '\\'

scenarios = ['baseline','BAU','FUPS' 'high','low']
years = ['25', '35', '50']

#
# USER DOES NOT NEED TO CHANGE ANYTHING AFTER THIS LINE UNLESS INPUT DATA STRUCTURE
HAS CHANGED
#
```

```
# Define here the name of the scenarios to be analysed and define dataframes to store results in
prem_death_df = pd.DataFrame()
year_life_lost_df = pd.DataFrame()

# ug /m3 threshold above which there is an effect
if (varCnc == 'NO2'):
   threshold = 10
elif (varCnc == 'PM10'):
   threshold = 0
elif (varCnc == 'PM2'):
   threshold = 0

## Upload excel file ----------------------------------------------------------
filename = area + '_health analysis.xlsx'

xls_file = health_dir + filename
xls = pd.read_excel(xls_file,
                sheet_name = 'Summary_population statistics',
                usecols = 'B:Z',
                index_col= 0)
data = xls.loc[['Crude death rate per 1000 persons (in total population)', 'Value'],:].copy()
data.columns = xls.iloc[1,:]
data.index = ['CDR','LE']

selection_mort = ['m30-34', 'm35-39', 'm40-44',
            'm45-49', 'm50-54', 'm55-59',
            'm60-64', 'm65-69', 'm70-74',
            'm75-79', 'm80-84', 'm85+',
            'f30-34', 'f35-39', 'f40-44',
            'f45-49', 'f50-54', 'f55-59',
            'f60-64', 'f65-69', 'f70-74',
            'f75-79', 'f80-84', 'f85+' ]

data_mort = data.loc[:, selection_mort].copy()
#print(data_mort)

## upload data ----------------------------------------------------------------
back_file = area + '_' + varCnc + '_baseline.mat'

# Upload the background matlab file
mat_file = aq_dir + back_file

mat_back = sio.loadmat(mat_file,
            variable_names=[varCnc,
                        'ninhab',
                        'X',
                        'Y'],
```

```
            matlab_compatible=False)

# Coordinate system
X = pd.DataFrame(mat_back['X'])
Y = pd.DataFrame(mat_back['Y'])

# Number of inhabitants
ninhab = pd.DataFrame(mat_back['ninhab'],
                index = X.iloc[0,:],
                columns = Y.iloc[:,0])

# Get population per thousand
ninhab_norm = ninhab / 1000

## Loop over all scenarios and years in files ------------------------------

# Make empty list of source files available in directory
mat_files = []

data_dir = aq_dir

fnm = area + '_' + varCnc

# Create list of files in data directory
for filename in os.listdir(data_dir):
   if filename.startswith(fnm) and filename.endswith(".mat"):
     mat_files.append(filename)
   else:
      continue

for mf in mat_files:

   # Get scenario of current mat file
   scenario = mf.split('.')[0].split('_')[2]
   print('fileNm', mf,'scenario',scenario)

   # Generate list of variables to be uploaded
   var = []
   if 'baseline' in scenario:
      var.append(varCnc)
   else:
      for yr in years:
         var.append(varCnc + '_' + scenario + yr)

   mat_file = data_dir + '\\' + mf

   print(var)
```

```
print(mat_file)

# Upload data from mat file
mat_tmp = sio.loadmat(mat_file,
        variable_names=var,
        matlab_compatible=False)

for v in var:

    # Get scenario year
    year = '20' + years[var.index(v)]

    # Get annual mean concentration
    Cmean = pd.DataFrame(mat_tmp[v],
            index = X.iloc[0,:],
            columns = Y.iloc[:,0])

    # Get total. Optional you can add values here.
    Cmean_tot = Cmean.copy()

    # Calculate relative risk and population attributable fraction
    # Concentration-Response Functions
    if (varCnc == 'NO2'):
       CRF = 1.055
    elif (varCnc == 'PM10'):
        CRF = 1.04
    elif (varCnc == 'PM2'):
        CRF = 1.62

    B = log(CRF)/10
    # making sure we take only the values above the Co= hreshold, otherwise there will be negative
values
    Cmean_thres = Cmean_tot[Cmean_tot >= threshold].copy()

    # risk ratio and population attributable fraction calculation
    RR = np.exp(B * (Cmean_thres - threshold))
    PAF = (RR - 1)/ RR

    # Calculate premature death and years of life lost and store in cube

    # Prepare cubes
    PD = np.zeros((X.shape[1],Y.shape[0],len(data_mort.columns)))
    YLL = np.zeros((X.shape[1],Y.shape[0],len(data_mort.columns)))

    # Loop over all relevant age, gender categories
    for col,row in data_mort.iteritems():
       prem_death = data_mort.loc['CDR', col] * PAF * ninhab_norm
```

```
        year_life_lost = prem_death * data_mort.loc['LE',col]

        i = selection_mort.index(col)

        PD[:,:,i] = prem_death
        YLL[:,:,i] = year_life_lost

        del prem_death, year_life_lost

    # Sum data over 3rd dimension of cubes. These results could be used
    # to generate maps when saved.
    PD_sum = np.nansum(PD,axis=2)
    PD_sum[PD_sum == 0] = np.nan
    YLL_sum = np.nansum(YLL, axis=2)
    YLL_sum[YLL_sum == 0] = np.nan

    # Store results in dataframes
    if 'baseline' in scenario:
        prem_death_df.at[scenario, 'baseline'] = np.nansum(PD)
        year_life_lost_df.at[scenario, 'baseline'] = np.nansum(YLL)
    else:
        prem_death_df.at[scenario, year] = np.nansum(PD)
        year_life_lost_df.at[scenario, year] = np.nansum(YLL)

    # Print out totals for entire grid and all age groups
    print('Annual premature deaths in scenario {}, year {}: {}'.format(
          scenario, year, np.nansum(PD)))

    print('Annual years of life lost in scenario {}, year {}: {}'.format(
          scenario, year, np.nansum(YLL)))

## Save results ----------------------------------------------------------------
prem_death_df.to_csv(save_dir +  area + '_prem_death_' + varCnc + '_scenarios_exp.csv')
year_life_lost_df.to_csv(save_dir + area +'_yll_' + varCnc + '_scenarios_exp.csv')
```