Welcome to the step-by-step guide on how to use our full model!

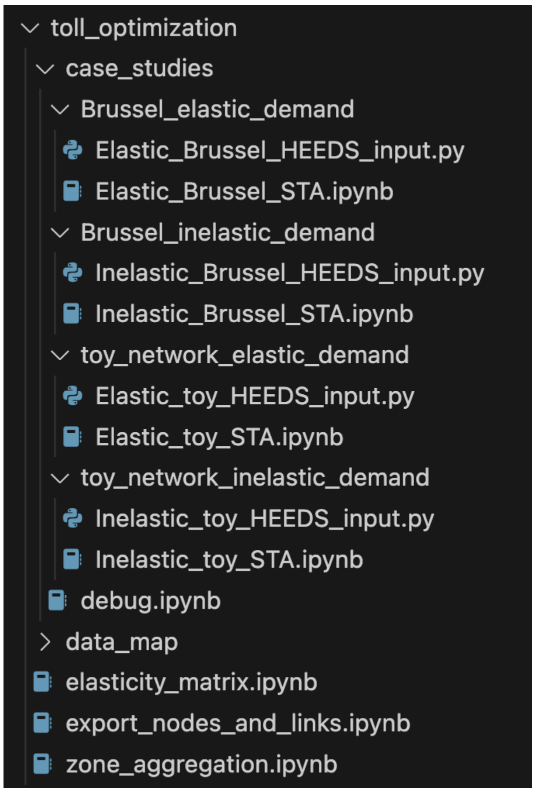
The goal of this guide is to make you understand all the steps we had to take to construct the scaled-up toll implementation model on the Brussels network for the course 'Integrated Practicum 2' of the MSCE master. Our project is based on the dyntapy toolbox by the ITS research team of KU Leuven. For further reference, look at the included paper and dyntapy documentation.

INTRODUCTION

Getting started:

- Clone [Github repository]()
    - This repository is built on dyntapy v0.2.3

Folders and file structure



```
toll_optimization
  case_studies
    Brussel_elastic_demand
      Elastic_Brussel_HEEDS_input.py
      Elastic_Brussel_STA.ipynb
    Brussel_inelastic_demand
      Inelastic_Brussel_HEEDS_input.py
      Inelastic_Brussel_STA.ipynb
    toy_network_elastic_demand
      Elastic_toy_HEEDS_input.py
      Elastic_toy_STA.ipynb
    toy_network_inelastic_demand
      Inelastic_toy_HEEDS_input.py
      Inelastic_toy_STA.ipynb
    debug.ipynb
  data_map
  elasticity_matrix.ipynb
  export_nodes_and_links.ipynb
  zone_aggregation.ipynb
```

The only folders you might have to run are depicted on the left side, which are located in folder "toll optimization". You can see that there are four main folders: Brussel (in)elastic demand and toy_network (in)elastic demand. In the continuation of this guide, the toy_networks will be discussed first, and afterwards the more complex Brussel files. The data map will be discussed in this last topic. The two files at the bottom are only relevant for real cases, so they will be elaborated on in the Brussels part of the guide. files for the elastic Brussels file, so these are also elaborated on in the last part of the guide.

Each folder contains both a python notebook covering the STA for the specific case, as well as a python script that can be used as input in Heeds for optimization. All of these files are extensively documented, such that every step of the process is explained in detail. This allows the guide to give a more high-level overview.

STEP-BY-STEP GUIDE

General dyntapy functions

Dyntapy was used as the core of our model throughout the development of the full toll optimization. Therefore, there are some functions that were implemented by other researchers. However, having a basic understanding of how to use dyntapy is extremely important for the ease of use. Hence, this paragraph provides an explanation for the most used functions. All of them are located in the "dyntapy" folder. If anything is unclear, you can always right-click on a function to go to its definition.

LOCATION	**FUNCTIONALITY**	**EXPLANATION**
STA → assignments.py	StaticAssignment()	This performs a Static Traffic Assignment on the given network and the provided OD graph with the demand flows.
	assignment.run()	This runs the STA based on the method provided between brackets. Dial-B is always used. You can find the code for it in STA → dial_b.py.
→ supply_data.py	get_toy_network()	This function retrieves a toy network, which can be used as a simple road network.
	relabel_graph()	This relabels links and nodes counting from 0. Refer to dyntapy documentation to see why relabelling is required.
→ demand_data.py	add_centroids()	This function adds the centroids as artificial nodes to the graph, on which eventually the demand data will be loaded into the network. This automatically also creates connectors, such that the flows can get transferred from the centroids to the actual network.
→ visualisation.py	show_network()	This shows the road network in the notebook as a graph.
	show_demand()	This shows the demand data in the notebook as a graph.
→ toll.py	create_toll_object()	This creates the toll object that you will need to give as an argument to the STA function. You should provide a network, a tolling method ('single', 'zone' or 'cordon'), the link(s) to toll and the toll value(s) to charge. **Note:** This is not a function in the publicly available dyntapy, but was added in our project.

Toy networks

To get a grasp on the concepts that will be used in a scaled-up version of the model, please start with the toy network implementations. This gives you the opportunity to understand all the steps towards the end result better.

Toy network with inelastic demand

The first file you should look at is the notebook “Inelastic_toy_STA.ipynb”. The vast documentation can be found in the notebook itself, but the steps taken are:

1. Import packages and files
2. Build network and OD graph
    a. Create or retrieve a network
    b. Create and add centroids to the network → adds connectors automatically
    c. Create or load an OD matrix (with the demand) onto the centroids
    d. Optional - Visualise network and demand
3. Run STA
    a. Decide on method
    b. Optional - Visualise loaded network
4. Add toll object to STA
    a. Explore different tolling methods by defining method, links to be tolled and toll value(s)
5. Prepare for Heeds outer loop

The second file you should look at is the python file “Inelastic_toy_HEEDS_input.py”. The steps are mainly the same as the previous file, but slightly adapted to be used as input for Heeds. The only addition is the objective function, which is required in Heeds to optimize. You should modify steps 2, 3 and 6 of this script according to your own files and needs.

1. Import packages and load your files
2. Create Heeds input
    a. Toll value(s)
    b. Toll ID’s
    c. Toll method
3. Retrieve paths → *warning*: hardcoded!
    a. Network
    b. OD matrix
4. Create a toll object
5. Run STA with toll
6. Create an objective function for Heeds output

Toy network with elastic demand

The first file you should look at is the notebook “Elastic_toy_STA.ipynb”. The vast documentation can be found in the notebook itself (and an explanation of elastic demand in our report), but the steps taken are:

1. Import packages and files
2. Build network and OD graph
    - Create or retrieve a network
    - Create and add centroids to the network → adds connectors automatically
    - Create or load an OD matrix (with the demand) onto the centroids
    - Optional - Visualise network and demand
3. Estimating A matrix
    - Choosing a random B matrix (in a realistic case we would estimate elasticities, but here a random value is chosen)
    - Run STA without toll
    - Derive A based on inverse demand function and B.
4. Run a loop of STAs with toll
    - Decide on tolling method, recall different tolling schemes from “Inelastic_toy_STA.ipynb”
    - After each iteration, OD-matrix is modified (until convergence)
    - Optional - Visualise loaded network
5. Prepare for Heeds outer loop

The second file you should look at is the python file “Elastic_toy_HEEDS_input.py”.

The steps that are needed here are the same as in “Inelastic_toy_HEEDS_input.py”, with some minor adaptations to account for elastic demand. With the explanation about this given above, the code should be clear to the user.

Brussels network

By now, you should understand the main functionality of our project. This allows us to take it to the next step: a bigger and more realistic network. In our project, this is a network of Brussels with a radius of 40 km around the centre.

As a starting point, it is assumed that the following necessary files are available:

- OD matrix data for the region of interest
- Shapefile of the region of interest

These can be loaded by running STA_prep/STA_prep_script.py (or ~.ipynb) as follows:

```
import STA
place = "BRUSSEL" # CAPS
buffer_N = 40
buffer_D = buffer_N
buffer_transit = 25
D_sup=1 # kms, use in the range 0-1 (res:0.1) if you want to include some boundary zones
time_of_day = 9
ext  = 0  # 0 for only internal, 1 for taking external demand w/o transit, 2 for external
with transit
STA.STA_initial_setup(place, buffer_D,buffer_N,buffer_transit, D_sup, time_of_day, ext)
```

Our code stores the relevant files (from which we start our notebooks etc.) elsewhere. The following table shows where STA_initial_setup(...) stores its output files and the files from which we start our notebooks. These files should be the same.

	Datapaths output STA_prep_script	Datapaths input of our notebooks
OD matrix	STA_prep/od_matrix_data/BRUSSEL_40_9_.xlsx	toll_optimization/data_map/STA/OD_matrix/BRUSSEL_40_9_.xlsx
Shapefile (zoning)	STA_prep/network_data/BRUSSEL_40.osm	**Inelastic notebook:** toll_optimization/data_map/STA/shapefiles/BRUSSEL_40_10/BRUSSEL_40_10.shp **Elastic notebook:** toll_optimization/data_map/QGIS/BRUSSEL_40_10.shp
Road network	STA_prep/network_data/BRUSSEL_40.osm	**Not used**: We run road_network_from_place in our notebooks, which basically retrieves the same. Alternatively, you can load the .osm file.

However, due to the large size of both the road network and the OD matrix, it is crucial that measures to mitigate complexity are implemented. For this, you first need the "zone_aggregation.ipynb" file. The steps are mostly done using QGIS, for which we refer to the additional paper.

"zone_aggregation.ipynb":

1. Import packages and files
2. Relabel zones
3. Split the shapefile using QGIS
4. Cluster and keep track of which original zones lie within each cluster
5. Update the centroids and OD matrix
   - Add cluster column to unclustered files
   - Combine shapefiles
6. Retrieve all files needed

What might also be useful is the "Export_nodes_and_links.ipynb" notebook. This notebook exports the links and nodes of your network to shapefiles. This allows you to easily select links in a cordon or zone using QGIS or other programs.

"Export_nodes_and_links.ipynb":

1. Import packages and files
2. Get the correct file paths
3. Build network and OD graph
   - Create or retrieve a network
   - Create and add centroids to the network → adds connectors automatically
4. Exports to QGIS
5. Perform desired steps (see additional report)

Brussels network with inelastic demand

The first file you should look at is the notebook "Inelastic_Brussel_STA.ipynb".

0. Make sure you start with the correct files by running:
   a. "STA_prep_script.py"
   b. "zone_aggregation.ipynb"
1. Import packages and files
2. Retrieve paths to data and zoning
   a. Dynamically coded: specify the city of interest and buffer (what you used in STA_prep)
3. Load network and OD
   a. Project centroids to the correct coordinate reference system
   b. Add centroids (and connectors) to network
4. Run STA with toll
5. Prepare for Heeds outer loop
6. Export network to QGIS with ""Export_nodes_and_links.ipynb"" to select links to toll in Heeds

The second file to interpret is the python file "Inelastic_Brussel_Heeds_input.py".

1. Import packages and files
2. Create Heeds input
   - Toll value(s)
   - Toll ID's
   - Toll method
3. Retrieve paths → *warning*: hardcoded!
   - Network
   - OD matrix
4. Create a toll object
5. Run STA with toll
6. Create an objective function for Heeds output

Brussels network with elastic demand

In order to use the following notebook, you first need an elasticity matrix. The matrix we constructed can be obtained by running "elasticity_matrix.ipynb":

1. Group zones based on slices from QGIS
2. Initialise slice elasticity matrix
3. Index actual elasticity values in slice matrix→ *warning*: hardcoded!
4. Make the actual zone elasticity matrix
    - Initialise matrix
    - Loop to fill in values
5. Determine path

The first file you should look at is the notebook "Elastic_Brussel_STA.ipynb".

0. Make sure you start with the correct files by running:
    - "STA_prep_script.py"
    - "zone_aggregation.ipynb"
    - "elasticity_matrix.ipynb"
1. Import packages and files
2. Retrieve paths to data and zoning
    - Indicate buffer used for both network and OD matrix delineation
    - The city used as the centre
3. Load network and OD
    - Adapt to "zoning"
    - Project centroids to the correct coordinate reference system
    - Add centroids (and connectors) to network
4. Find A matrix
    - Run STA without toll
    - Import B matrix
    - Run inverse demand function with skims
5. Run STA with toll (see "tolling" folder)
    - Decide on method
    - Loop over changed OD graphs
    - Modify zero-rows and columns
    - Optional - Visualise loaded network
6. Prepare for Heeds outer loop

The second file to interpret is the python file "Elastic_Brussel_Heeds_input.py".

The steps that are needed here are again more or less the same as in "Inelastic_Brussel_Heeds_input.py", with some minor adaptations to account for elastic demand. With all the explanations about this given above, the code should be clear to the user.