# Introduction

This note is intended to lay out the structure of the Mobile Network Data (MND) processing scripts written for the Norwich Western Link project and then adapted for both the Wisley and A259 projects. All of these projects use a set of Telefonica MND and have a similar overall structure, but each has slightly different adaptations due to calibration and validation adjustments that were put in place late on in the processing period.

Note that although I’ve set out a lot of the information you might need in this readme document, there should be fairly good docstrings for both the individual modules and all the functions and objects within the modules if you are experienced enough to just be able to use them directly.

# Python Setup

In order to be able to run the scripts you’ll need a Python install with the relevant libraries.

I would recommend grabbing the [latest Anaconda version](https://www.anaconda.com/products/individual). If you don’t want all of the extras that come with normal Anaconda, you can get [Miniconda](https://docs.conda.io/en/latest/miniconda.html), just be aware that this means you’ll need to install all of the extras that you want on your own (such as editors). This isn’t difficult, it’s just something to be aware of and it’s easier to get Anaconda if you don’t know what you might want.

It’s possible to look at the code in something like Notepad++, however I would recommend using a more powerful editor such as Spyder, Pycharm, or VS.Code for editing (Spyder and Pycharm are included in Anaconda). This helps to tell you when you’ve made mistakes in the code and highlights certain parts to make it easier to read.

You’ll need the following modules installed. Some of these come by default with Anaconda, but others you may need to install yourself. If you get an error when running a script to do with one of the Import statements at the top of the script where it tells you it’s missing one of modules, you need to either open the environment editor in Anaconda and install the relevant package, or open up the conda terminal and type in ‘conda install packagename’ where packagename is the name of the modules you want.

Libraries required:

* pandas;
* numpy;
* jsonpickle.

I believe all other modules referred to in import statements are either part of the standard Python modules or are custom modules created for this process.

# General Structure

There are 3 main modules involved in the process, with one associated module:

* Matrix Class (with Matrix Decorators);
* MND Import; and
* MND Processing.

Matrix Class defines some objects that are used throughout the MND import and processing. This contains definitions of a MatrixStack, Matrix, and Zones objects. I’ll cover these in more detail in the Matrix Class section.

MND Import then contains functions that can be used to import the mobile network data in a variety of forms and filtering certain information. Some of the imports / exports in this script are for viewing the data in configurations for verification rather than for use in processing.

MND Processing contains the actual processing used to convert the MND (and associated information such as synthetic matrices) into the prior matrices in CSV format.

There are a set of batch and key files in the ‘SaturnConvert’ folder which allow easy conversion of the CSV matrices into UFM format, as this is not part of the python script (although running the batch files could be incorporated easily enough if you wanted to do that).

# Matrix Class

There is no need to fully understand the matrix class in order to use the MND scripts. The objects that are defined are used a lot but an understanding of how this works isn’t necessary to just use them. It’s probably easiest to just inspect the MND import and processing script and see how the objects are used rather than delving into the specifics of each one before doing this. A summary of the objects is provided below. The matrix class script also forms the backbone of the functionality that is used for the MXI tool.

## Zones

A zones object contains data about the zone references and X and Y coordinates, along with any associated data the user wishes to feed in. A Zone object is required for a matrix to be defined, and can be used to do a number of things such as generate a blank matrix, create a crow-flies distance matrix, create a dataframe with all origin to destination pairs listed, or summarise the zone system down to something more aggregate (which is used for sectoring operations).

## Matrix

A matrix is defined as a square matrix of values. It doesn’t have to be a trip matrix, but could also represent distances, times, or some other data. The matrix has a set of properties set on initialisation, such as the vehicle type, value type, level (user class number in SATURN terminology), and purpose. These don’t have to be defined and other than level are not used in any specific way. The definition as square is inherent, and it has to be the same size as the zone system that is attached to it. Matrix data can be imported from record-format files or square-format files. Operations (addition etc) can be carried out between matrices and the object will handle these operations as you might expect. Matrices have methods to allow disaggregation, redistribution, and aggregation (e.g. sectoring) operations, among other more niche tasks.

## MatrixStack

A matrix stack is intended to contain a stack of matrices. Many of the operations that can be carried out on a matrix can also be applied to a matrix stack, and the stack will handle applying that operation to each matrix that forms part of the stack individually. Matrix stacks also handle operations such as addition in the way you would expect when handling a stacked matrix.

Matrices and stacks can be exported or imported using save and save\_csv methods for exporting, and the load\_matrix function for importing an object that has had save called. jsonpickle handles the save and load functionality to fully export the objects intact and then import them back again without any loss of data.

# MND Import

MND import has a number of hard-coded definitions of the MND set at the top of the script. These constants are used to define what the MND looks like and how it will therefore be imported. Note that these definitions are not exhaustive and there are some hard-coded elements within the functions themselves in certain cases. The constants defined may vary per project. The lists included in these constants are order-sensitive, so care should be taken when editing these or producing a new version of the script for a new job to make sure the ordering is correct. Python is a caps-sensitive language as well, so be aware that capitalisation is important.

The main function to use to get a summary of all of the MND and export it to a set of files is get\_all\_mnd. Most of the rest of the functions in the script are either helpers for this function or are for slicing the MND in a particular way for verification testing (getting aggregated daily trips, getting just journey to work trips, etc).

# MND Processing

The MND processing outline is described in the docstring at the top of the module and follows what is described in the LMVRs for the different projects. Note that the A259 script is likely the ‘tidiest from a documentation point of view as it is the most recently developed, although as noted in the introduction, each of the project scripts differs a little due to specific adjustments or adaptations introduced for each.

The primary tasks are:

* Disaggregation of zone system from MND to model;
* Splitting of purposes (HB Other -> HB Other + HB Employers Business);
* Road -> Car / LGV / Bus splitting;
* Rail -> Car access/egress trips + Remainder;
* Person to vehicle conversions where relevant;
* PCU scaling; and
* Short distance infill (using Synthetic join).

The main process for each input element (Road, HGV, Rail) is dealt with by a dedicated function (‘road\_matrix\_process’ or equivalent). The doc strings for these indicate what steps of the process are being carried out. The primary steps of the process outline above are dealt with by dedicated functions, while intermediate adjustments (simple scaling, splitting, saving, and discarding of matrices) are contained within the process functions.

Following the main processing, the ‘Highway\_matrices\_join’ function combines the car, rail-related car, LGV, and HGV outputs into one matrix stack for easier exporting. There’s an equivalent public transport joining system where required (Wisley does not have anything to do with exporting PT matrices as the model was purely highway).

## Running the script

When the ‘matrix\_processing’ script is run it should run through the block at the very bottom of the script (following the “if \_\_name\_\_ == ‘\_\_main\_\_’:” statement). This runs the processes for each time period. The scripts should report to the console the current operations they’re doing. You might find that, if you don’t have the right constants set at the top of the script to do with folder locations, that the script will throw up a load of errors because it either can’t find files to import, or it can’t find folders to export to as the script doesn’t create folders when exporting, it expects them to already exist.

The two variables ‘load\_raw’ and ‘refresh’ can be used to force the script to reimport the base data from scratch rather than loading what has already been done so far. It’s recommended to keep these set to True as it doesn’t take that long to run any of the processing scripts from start to finish. The slowest part of the scripts is reading and writing files.