USER EQUILIBRIUM SOLUTION IN TRAFFIC ASSIGNMENT USING FRANK-WOLFE ALGORITHM

Team:

1) Amogha | 150040102
2) Priyharsh | 150040089
3) Ashwini K | 150040040
4) Ayush | 150040087

Introduction: Traffic assignment or trip assignment is the last step of classical four step model of travel demand modelling.In this step we allocate a given set of trip interchanges to specified transportation system. The fundamental aim of this process is to reproduce on the transportation system, the pattern of vehicular movements which would be observed when the travel demand represented by the trip matrix, or matrices, to be assigned is satisfied.

There are different types of traffic assignment models such as are all-or-nothing assignment **(AON)**, incremental assignment, capacity restraint assignment, deterministic user equilibrium assignment **(UE)**, stochastic user equilibrium assignment **(SUE)**, system optimum assignment **(SO)**, etc

In our project we have focused on **Deterministic User Equilibrium (UE)** method which is a widely used model among various available trip assignment models.

Deterministic User Equilibrium Model - Deterministic User Equilibrium model of traffic assignment is based on the fact that humans choose a route so as to minimize his / her travel time and on the assumption that such a behaviour on the individual level creates an equilibrium at the system (or network) level.
In terms of Wardrop's first principle - Flows on the links(whose travel time vary with the flow) are said to be in equilibrium when no driver/trip maker can unilaterally reduce his/her travel costs by shifting to another route.
Since this is a deterministic model it assumes the fact that drivers are completely rational and have knowledge of all travel cost and travel time on a given link is a function of flow on that link only.

Mathematical Formulation -

Minimize Z = $\sum_a \int_0^{x_a} t_a x_a \, dx$

Subjected to: $\sum_k f_k^{rs} = q_{rs}$: $\forall\ r, s$

$$x_a = \sum_r \sum_s \sum_k \delta_{a,k}^{rs} f_k^{rs} \ : \forall a$$

$$f_k^{rs} \geq 0 \quad \forall\ k, r, s$$

$$x_a \geq 0 \quad : a \,\varepsilon\, A$$

where: x_a – equilibrium flows in link a,

t_a – travel time on link a , $t = t_0 \left\{ 1 + \alpha \left(\frac{x}{k}\right)^{\beta} \right\}$, where α *and* β are specific to type of link and needs to be calibrated from field data.

f_k^{rs} – flow on path k connecting O-D pair

q_{rs} – Trip rate between r and s

$\delta_{a,k}^{rs} = 1$ *if link 'a' belongs to path 'k'*

0 *otherwise*

The equation is simply the flow conservation equation and non negativity constraint respectively. These constraints naturally hold the point that minimizes the objective function.The path connecting O-D pair can be divided into two categories : those carrying the flow and those not carrying the flow on which the travel time is greater than (or equal to) the minimum O-D travel time. If the flow pattern satisfies these equations no motorist can better off by unilaterally changing routes. All other routes have either equal or heavy travel times. The user equilibrium criteria is thus met for every O-D pair.

This problem is convex optimization problem because the link travel time functions are monotonically increasing function, and the link travel time a particular link is independent of the flow and other links of the networks.

To solve such problem we have used Frank Wolfe Algorithm -

Frank-Wolfe Algorithm is used for solving quadratic programming problem with linear constraints.It is applicable to nonlinear programming problems with convex objective functions.

The algorithm involves two key implementation steps which are -

1) **Descent direction search :** Maximize the *drop* (product of the rate of descent in a given direction and the length of feasible move in that direction); popular methods include linear programming and Dijkstra's shortest path algorithm.

2) **Line search :** Since the bounding of the move size is naturally accomplished in the descent direction search process, simple interval reduction methods could be used to find the optimal step size.

The above steps are implemented repeatedly until a convergence is reached

We have used Python to replicate the process described above in the following steps (for more details, please refer to the comments in the code):

1) Use array data structures to represent the network and given information;

Input Datasets	Python Variable	Definition	Data Structure	Size
Network Information: Link & Node	n, k	n: # of links k: # of nodes	Python number	1
Network Information: Link & Node	Link Node	Link-node Incidence matrix (adjacency)	Python array	nparray(n, k)
OD flow demand (q ij)	Q	Travel flow demand between each OD pair	Python array	n parray(n,n)
Link travel time information: free Flow time,parameters for the BPR function,link capacity	coeff	Includes two columns, column 1 is for free flow travel time on each link (t0); column 2 is for link capacity (ca), other parameters used in the BPR function (e.g., alpha,	Python array	nparray(n,2)

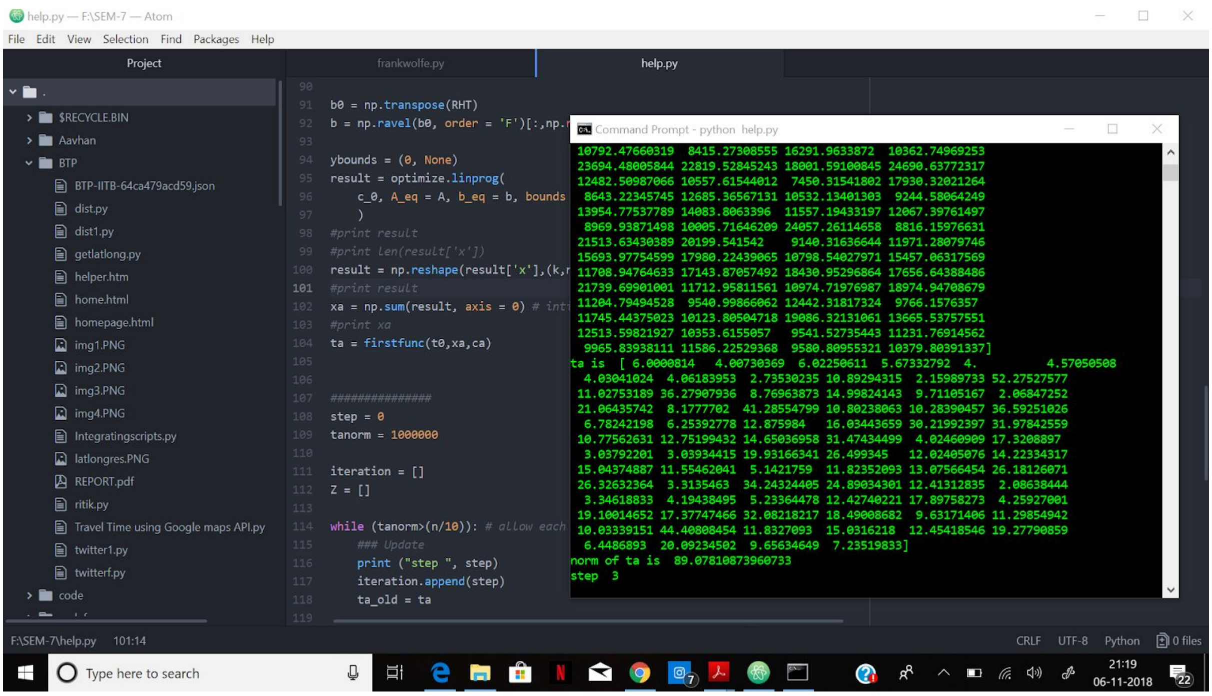
		beta) are constants		

2) Conduct the descent direction search by forming a Linear Programming problem and solve it;
   - **Initialization:** carry out the All or Nothing flow assignment (find xa) based on the free flow travel time on each link (t0), then update the travel time on each.

   - **Iteration (direction search):** Formulate the objective function (Z) and constraints for the LP program, then use the *optimize.linprog* function in Python package to solve the problem and obtain updated link flow (ya)

3) Find the optimal step size using the Golden section interval reduction method for the one-dimensional search;
   - **Iteration (move):** formulate the line search function and use the *minimize_scalar* function in Python package to solve the step size alpha

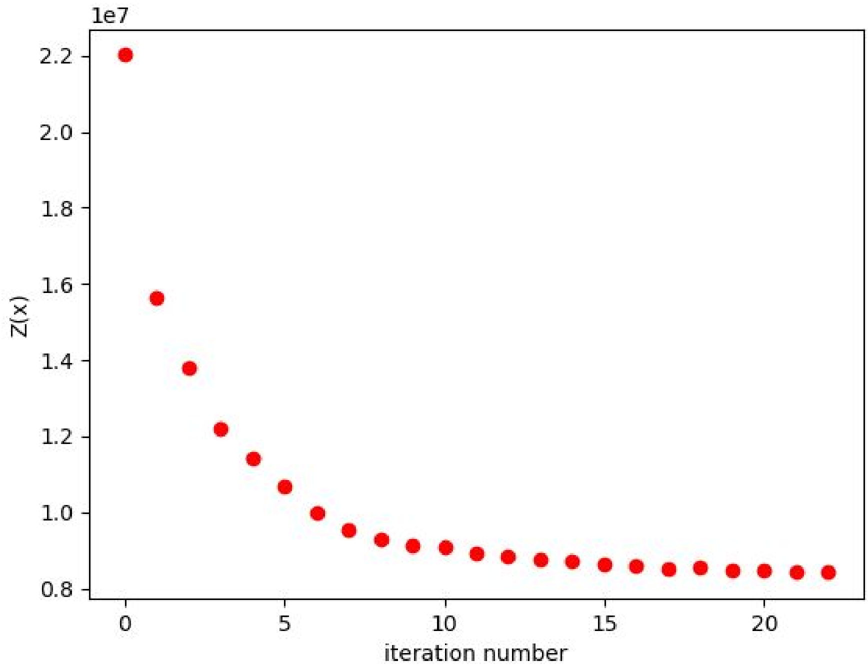
4) Update the network and check for convergence.
   - **Iteration (update):** update link flow (x_a) using y_a and alpha, then use x_a to update t_a.
   - **Iteration (check convergence):** Calculate the norm of (iteration n) and ta (iteration n-1), when Norm < 0.1*n (on average the link flow changes 0.1), the iteration stops and the algorithm is converged



Results of the Implementation-

→ Two csv files are generated as results.

→ The graph of number of iterations and the objective function value is being attached here.



The above algorithm is deterministic in nature assumes that drivers are completely rational and identical and have complete knowledge of flows and networks This is usually not the case in reality and this is the reason why these models fall short in giving a proper estimate for any arbitrary network.