TRANSPORT AND MOBILITY MODELLING COURSEWORK SOLUTIONS

SEMTM0010

2072442

Ye20178@bristol.ac.uk

QUESTION 1 DEMAND MODELLING: 'THE LINE' SAUDI ARABIA

'The Line' – Saudi Arabia, in this question, is modelled to assume that the Line city is organised with its population, $P = 5{,}000{,}000 = 5M$ individuals are

- Only a Working Population (only workplaces and residences along the $L = 170km$ line and everyone has a job)
- Uniformly distributed within localities $(i\ or\ j)$ landmarked by a metro station. The total number of metro stations/localities is N
- Have a typical commute distance $d_{ij} = (1 + |i - j|)\frac{L}{N-1}km.$

Given that the flow of individuals from a locality i to locality j is represented by $f\left(O_i, D_j, d_{ij}\right)$ for $i, j \in \mathbb{Z}^+, [1, N]$ that $f_{ij} = c\frac{O_i D_j}{d_{ij}^2}$ & c is the proportionality constant. Each locality has $O_i = \frac{P}{N}$ workers and $D_i = \frac{P}{N}$ jobs. Therefore, within the model, the total number of workers, $\sum_i^N O_i = P\, workers$, and the total number of jobs $\sum_i^N D_i = P\, jobs.$

A) SHOW THAT NO CHOICE OF C WILL SIMULTANEOUSLY SATISFY ALL OF THE 2N CONSTRAINTS

Proof: No choice of c will simultaneously satisfy all of the 2N constraints of sources (metro station is, $O_i = \sum_j^N f_{ij}$) and sinks (metro station js, $D_j = \sum_i^N f_{ij}$), for

Equation 1-1: Gravity Model Equation

$$f_{ij} = c\frac{O_i D_j}{d_{ij}^2}$$

I : Simplify the equations for the sources and sinks, such that c is the subject of the formula and substitute for $d_{ij} = (1 + |i - j|)\frac{L}{N-1}km, \sum_i^N O_i = P\, workers$ and $\sum_i^N D_i = P$ jobs.

$$O_i = \sum_{j=1}^{N} c\frac{O_i D_j}{d_{ij}^2} = cO_i\sum_{j=1}^{N}\frac{D_j}{d_{ij}^2} = cO_i P\sum_{j=1}^{N}\frac{1}{d_{ij}^2} \Rightarrow$$

$$c_{O_i}{}^{-1} = P\sum_{j=1}^{N}\frac{1}{d_{ij}^2} = P\frac{(N-1)^2}{L^2}\sum_{j=1}^{N}\frac{1}{(1 + |i - j|)^2}$$

Similarly

$$c_{D_j}{}^{-1} = P\sum_{i=1}^{N}\frac{1}{d_{ij}^2} = P\frac{(N-1)^2}{L^2}\sum_{i=1}^{N}\frac{1}{(1 + |i - j|)^2}$$

II : Let's assume that c simultaneously satisfies all the 2N constraints, such that for **all elements** of i **and** j,

$c_{D_j}{}^{-1} = c_{O_i}{}^{-1}$. If **any** element in $i = \{1, \cdots, N\}$ and $j = \{1, \cdots, N\}$ result in $c_{D_j}{}^{-1} \neq c_{O_i}{}^{-1}$, then c **cannot** simultaneously satisfy **all** the 2N constraints.

<u>*Reason:*</u> The statement 'c simultaneously satisfies all the 2N constraints' implies that only one value of c ensures that $O_i = \sum_j^N f_{ij}$ and $D_j = \sum_i^N f_{ij}$, which means that $c_{D_j}{}^{-1} = c^{-1}$ and $c_{O_i}{}^{-1} = c^{-1}$, as they exhibit transitive property, $c_{D_j}{}^{-1} = c_{O_i}{}^{-1}$. Based on the statement, 'all the 2N constraints', imply all elements of i and j in equations O_i and D_j.

III : For $c_{D_j}{}^{-1} = c_{O_i}{}^{-1}$, the set of $M_j = \{1 + |i - j|, j \in [1, N]\}$ must be equal to $M_i = \{1 + |i - j|, i \in [1, N]\}$, based on their equations.

IV: Test for $j = i = 1$, N = 2 & N = 4

For N = 2, $\mathrm{M}_j = \{1 + |1 - j|, j \in [1,2]\} = \{1,2\}, \mathrm{M}_i = \{1 + |1 - j|, j \in [1, N]\} = \{1,2\}$

$$M_j = M_i$$

For N = 4, $\mathrm{M}_j = \{1 + |1 - j|, j \in [1,4]\} = \{\mathbf{1,2,3,4}\}, \mathrm{M}_i = \{1 + |1 - j|, j \in [1,4]\} = \{\mathbf{1,2,3,4}\}$

$$M_j = M_i$$

V: Test for $j \neq i, j = 2, i = 1$, & N = 4

For N = 2, $\mathrm{M}_j = \{1 + |2 - j|, j \in [1,2]\} = \{\mathbf{1,2}\}, \mathrm{M}_i = \{1 + |1 - j|, j \in [1, N]\} = \{\mathbf{1,2}\}$

$$M_j = M_i$$

For N = 4, $\mathrm{M}_j = \{1 + |2 - j|, j \in [1,4]\} = \{\mathbf{1,2,3}\}, \mathrm{M}_i = \{1 + |1 - j|, j \in [1,4]\} = \{\mathbf{1,2,3,4}\}$

$$M_j \neq M_i$$

For case N = 4, $M_j \neq M_i \; for \; j \neq i$, implying that no value of c can simultaneously satisfy all 2N = 8 constraints. Although for case N = 2, $M_j = M_i \; for \; both \; j = i \; and \; j \neq i$.For this scenario there are only two localities on The Line. The trip distribution or the flows from the two localities are easier to predict (there is a value of c that can simultaneously satisfy all 2N = 4 constraints) because there is only one sink and one source. Also, another important factor is the work population density distribution. It is uniform on the Line.

VI: To ensure robustness of the proof, Let's prove that for N = k, $k > 2$ will show that no value of c can simultaneously satisfy all $2k$ constraints.

Test for $j \neq i, j = 2, i = 1$, N = k, $k > 2$

For N = k, $\mathrm{M}_j = \{1 + |2 - j|, j \in [1, k]\} = \{\mathbf{1, 2, \cdots, 1 + |2 - k|}\}, \mathrm{M}_i = \{1 + |1 - j|, j \in [1, N]\} = \{\mathbf{1, 2, \cdots, 1 + |1 - k|}\}$

$$M_j \neq M_i, for \; k > 2$$

∴ No value of c can simultaneously satisfy **all** $2N$ constraints, $O_i = \sum_j^N f_{ij}$, $D_j = \sum_i^N f_{ij}$,for N > 2 on the Line.

B) DOUBLY CONSTRAINED GRAVITY MODEL

I)DERIVATION OF A SYSTEM OF 2N NONLINEAR EQUATIONS TO SOLVE FOR THE CONSTANTS $\mathrm{c}_\mathrm{I}^{(\mathrm{O})}$ AND $\mathrm{c}_j^{(\mathrm{D})}$.

Equation 1-2: Doubly Constrained Gravity Model

$$f_{ij} = \mathrm{c}_\mathrm{i}^{(\mathrm{O})} \, \mathrm{c}_j^{(\mathrm{D})} \, \frac{O_i D_j}{d_{ij}^2}$$

Let $x_{ij} = \frac{O_i D_j}{d_{ij}^2}$ be the **interlocal reaction** variable between two localities $i \; and \; j$, this implies that

$$f_{ij} = \mathrm{c}_\mathrm{i}^{(\mathrm{O})} \, \mathrm{c}_j^{(\mathrm{D})} \, x_{ij}$$

$$O_i = \sum_{j=1}^{N} f_{ij} = \sum_{j=1}^{N} \mathrm{c}_\mathrm{i}^{(\mathrm{O})} \, \mathrm{c}_j^{(\mathrm{D})} \, x_{ij} = \mathrm{c}_\mathrm{i}^{(\mathrm{O})} \sum_{j=1}^{N} \mathrm{c}_j^{(\mathrm{D})} \, x_{ij} =$$

Equation 1-3

$$\mathrm{c}_{\mathrm{i}}^{(\mathrm{O})} = \frac{O_i}{\sum_{j=1}^{N} \mathrm{c}_j^{(\mathrm{D})}\, x_{ij}}$$

Similarly,

Equation 1-4

$$\mathrm{c}_{\mathrm{j}}^{(\mathrm{D})} = \frac{D_j}{\sum_{j=1}^{N} \mathrm{c}_i^{(\mathrm{O})}\, x_{ij}}$$

For,

$$\begin{aligned} x_{ij} &= \frac{O_i D_j}{d_{ij}^2} \\ &= \frac{P^2}{N^2}\frac{(N-1)^2}{L^2}\frac{1}{(1+|i-j|)^2} \end{aligned}$$

II) FIND $\mathrm{C}_{\mathrm{J}}^{(\mathrm{D})}$ *and* $\mathrm{C}_{\mathrm{I}}^{(\mathrm{O})}$, AND EXHIBIT FLOWS

To find $\mathrm{c}_{\mathrm{i}}^{(\mathrm{O})}$ *and* $\mathrm{c}_j^{(D)}$

$$x_{ij} = \frac{O_i D_j}{d_{ij}^2} = \frac{P^2}{N^2}\frac{(N-1)^2}{L^2}\frac{1}{(1+|i-j|)^2}$$

$x_v = \frac{1}{(1+|i-j|)^2}, x_c = \frac{P^2}{N^2}\frac{(N-1)^2}{L^2}, and\ x_{ij} = x_c\, x_v$

Linear programming is a technique used in finding the best solution to a mathematical model or problem for some given constraints or boundaries. To find the best solution for $\mathrm{c}_{\mathrm{i}}^{(\mathrm{O})}$ *and* $\mathrm{c}_j^{(D)}$, a linear programming approach called Factor Estimation algorithm is used. It is a more efficient version of the iterative proportional fitting algorithm (1). The algorithm iterates between $\mathrm{c}_{\mathrm{i}}^{(\mathrm{O})}$ *and* $\mathrm{c}_j^{(D)}$ until convergence such that, for an interation, n :

$$\mathrm{c}_{\mathrm{i}}^{(\mathrm{O})^n} = \frac{O_i}{\sum_{j=1}^{N} \mathrm{c}_j^{(\mathrm{D})^{n-1}}\, x_{ij}}, \mathrm{c}_{\mathrm{j}}^{(\mathrm{D})^0} = 1$$

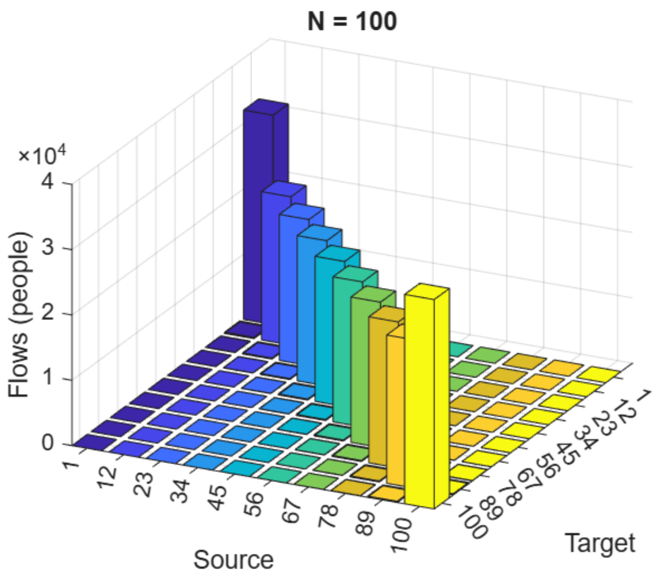
$$\mathrm{c}_{\mathrm{j}}^{(\mathrm{D})^n} = \frac{D_j}{\sum_{j=1}^{N} \mathrm{c}_{\mathrm{i}}^{(\mathrm{O})^n}\, x_{ij}}$$

MATLAB CODE:

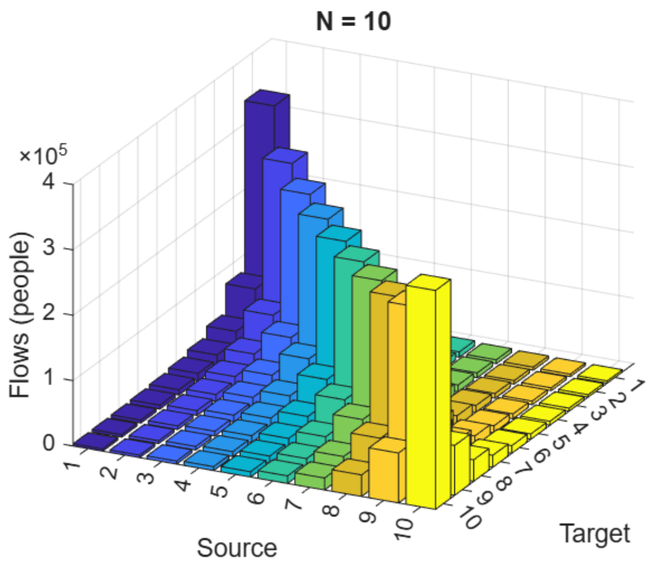
```
function [F,ci,cj,errors,iteration] = The_Line_DCGM_(N,epsilon)
% Based on IPF Version 1.0.0.0 (2.81 KB) by Matthew Roughan (2)
% Input:
%       N - the number of metrostations in the Model
%       epsilon - The error tolerance for convergence

% Outputs:
%       F - the N x N matrix of flows
%       ci - The origin proportionality constant
%       cj - The destination proportionality constant
%       errors – This shows the difference
%       iteration
% Example of usage
%
% N = 10;
% [F, ci, cj, errors, iteration] = The_Line_DCGM_(N, 1e-3);
% disp('For N (Sparse Line Arrangement): 10');
% disp('Flows F:'); disp(F);
% disp('Origin Propotionality Constant ci:'); disp(ci');
```

```
 % disp('Destination Propotionality Constant cj:'); disp(cj');
 %  N = 100;
 % [F_100, ci_100, cj_100, errors_100, iteration_100] = The_Line_DCGM_(N, 1e-3);
 % disp('For N (Dence Line Arrangement): 10');
 % disp('Flows for N = 100, F:'); disp(trimdata(F_100,[10,10]));
 % disp('Origin Propotionality Constant for N = 100, ci:'); disp( trimdata(ci_100',[1,10]));
 % disp('Destination Propotionality Constant for N = 100, cj:'); disp(trimdata(cj_100',[1,10]));
 %
 %

 F = zeros(N, N);
 P = 5e6; %Population
 L = 170; %Length of Line

 Oi = repelem(P/N,N)'; % Define Number of Workers from each Locality
 Dj = repelem(P/N,N)'; % Define Number of Jobs in each Locality

 i = ndgrid(1:N,1:N); % Rows
 j = ndgrid(1:N,1:N)'; % Columns
 % Define Xij, the interlocal reaction vairiable based on parameters P, N, L and distance of the line
 xc = (P^2 * (N-1)^2)/N^2 * L^2; % only the parameters P, N, and L (constants)
 xv = 1./(1 + abs(i-j)).^2; % only distance (array)
 xij = xc.*xv; % Scalar Multiplication
 cj = ones(N,1); % Define the destination proportionality constant
 n = 2*(N^2); % Define the maximum number of iterations

 % Using Factor Estimation
 errors = linspace(1,n,n); % Define the error variable
 iteration = 1; % Initialise the iteration variable
 Oi_F = sum(xij, 2); % Calculate the sum of interlocal reaction vairiables for each origin
 Dj_F = sum(xij, 1)'; % Calculate the sum of interlocal reaction vairiables for each destination
 % Calculate the total error between estimated and actual values
 total_error = norm(Oi_F - Oi) + norm(Dj_F - Dj); % norm is used to measure the euclidean distance between two points
aka the shortest distance between two points. Ther error varaible measures the ‘distance’ between the actual number of
workers and jobs in each locality and the sum of origin and destination flows
 errors(iteration) = total_error; % Store the current total error for convergence tracking
 % Iterate until convergence criteria are met
 while( (total_error > epsilon) && (iteration < n) )

 ci = Oi./ (xij * cj); % The cross product of cj and xij is the sum over j of element wise products of cj and xij
 cj = Dj./ (ci' * xij)';% The cross product of ci' and xij is the sum over i of element wise products of ci and xij

 F = ci .* cj'.* xij; % Dot product multiplication of the proportionaly constants and the -local reaction vairiable

 Oi_F = sum(F, 2); % Sum the origin flows
 Dj_F = sum(F, 1)'; % Sum the destination flows
 iteration = iteration + 1; % add 1 to interation
 errors(iteration) = norm(Oi_F - Oi) + norm(Dj_F - Dj); % Calculcate the error

 end
 F = round(F,0,'decimals'); % F cannot be in decimal form as it represent flows of workers to their jobs. To represent
flows it should be in integer form.
 % Some column or row sums do not equal to Oi, Dj because the factor estimation ensures the flows row and column sums
closely represent the
 % Origin and Destination Matrix closely using real numbers instead of integers. Rounding it each flow to an integer
would cause the difference.
 % If there was integer factor estimation that would have been a better solution
 end
```

Figure 1-1: Flows from one locality to another for N = 10 & N = 100



(Figure 1-1a illustrates the flow of workers from sample of Ns = 10 localities from a source i to a target locality j for N = 100)

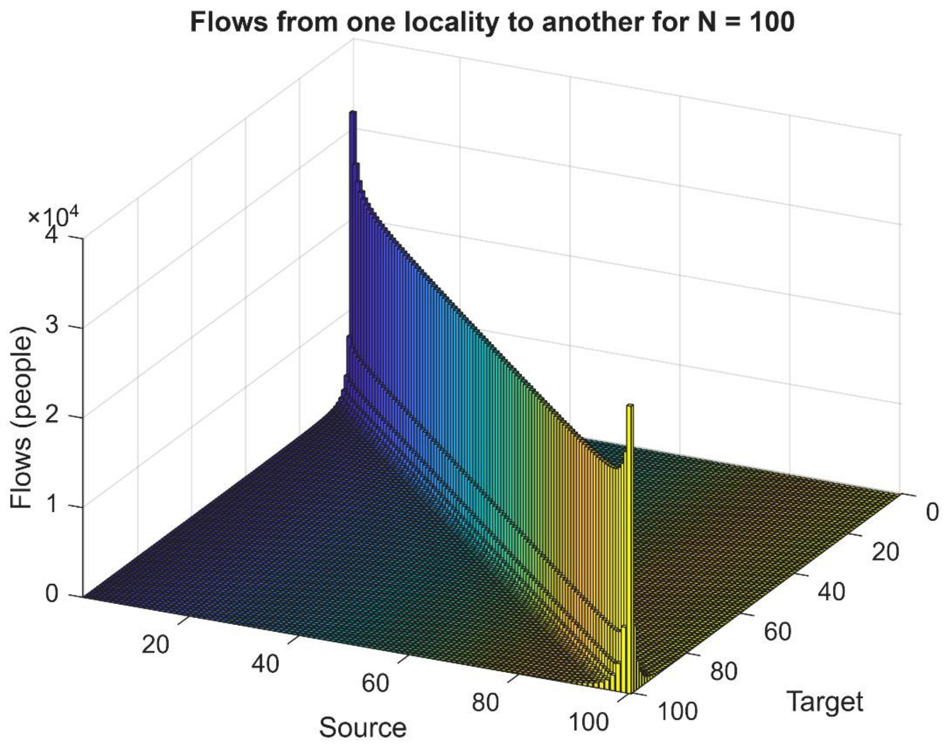


(Figure 1-1b illustrates the flow of workers all localities from a source i to a target locality j for N = 10)

Figure 1-1 exhibits the flows from one locality to another on 'The Line' for the number of station (N) being 100 and 10. In the plot for N = 100, only 10 stations are plotted (1,12,23,34,45,56,67,78,89,100) to make comparisons, Figure 1-3, shows all the nodes.

From Figure 1-1, the flows of people with N = 10 is 10 times more than the flows in N = 100. The first and last station, for both N =10 and N = 100, have higher intralocal flows as they are the key hubs of the Line. They are on the extreme ends of the line; hence workers would rather work intralocally, i.e the probability working within their locality for $i = 1 \; or \; 10$, is higher that other localities because of the distance travelled would be longer. The longer the distance travelled the less likely for trips to be made. The flows follow the Equation 1-2: Doubly Constrained Gravity Model as the flows for each source matches the origin and destination demand.

Figure 1-2:All the Flows from one locality to another for N = 100



(Figure 1.2 illustrates the flow of workers all localities within a Densely spaced Line from a source i to a target locality j for N = 100)

C) RESULTING FLOW ON EACH LINK OF THE METRO LINE

The resulting flows on each link on the line are calculated by summing up station flows on all the possible sources from a source x to all possible targets for both forward and backward journeys. The equation of link flows for a given source x can be written as

$$Link\ flows\ (forward)_x = \sum_{s=1}^{x} \sum_{t=x+1}^{N} f_{ij}(s,t)$$

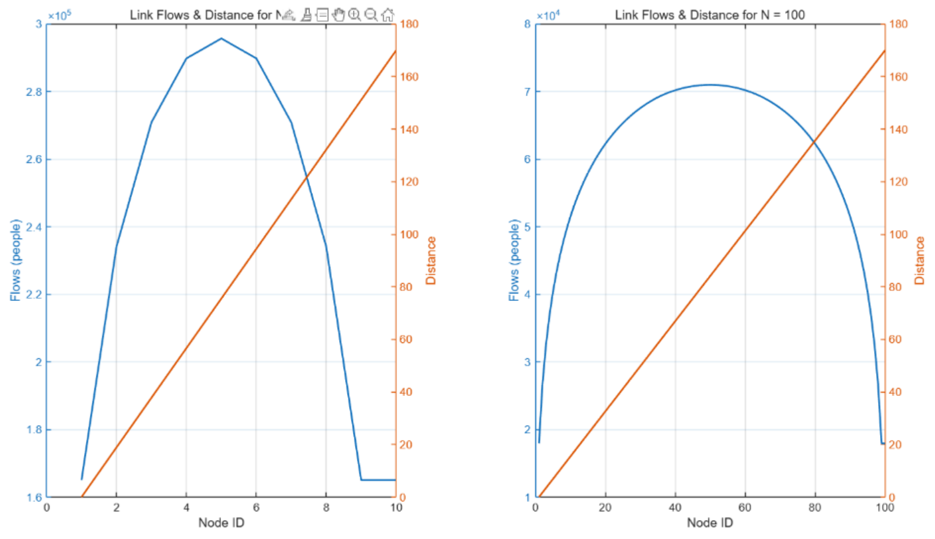
$$Link\ flows\ (backward)_x = \sum_{s=x+1}^{N} \sum_{t=1}^{x} f_{ij}(s,t)$$

MATLAB CODE:

```
function [s,t,d,Flows]= Graph_Line_(N,F)
    % GRAPH_LINE_ creates the sources, sinks and link flows on 'The Line'
    %
    % Input Arguments:
    %     N - number of metrostations or localities on The Line
    %     F - This is OD demand matrix gitten from prevous factor
    %     estimation solution.
    %
    % Output Arguments:
    %     s - sources
    %     t - sinks
    %     d - The distances between nodes or stations
    %     Flows - link flows on each link of the model i.e the number of individuals that will
    %     use link i,j on a given work day, as this is a working population.
    % Dependencies
    %   The_Line_DCGM_() to calculate the flows .
    % Create the source and sink vectors for the line
```

```
     % As it is a double track metrostation, flows can only originate at 1st and
     % the Nth Station.

     so = (1:N-1); % sources for the forward journey i.e from 1 to N-1
     t = (2:N);   % sinks or targets for the forward journey i.e from 2 to N
     s = [so,flip(t)]; % add sources for the backward journey i.e from N to 2
     t = [t,flip(so)]; % add sinks for the backward journey i.e from N-1 to 1
     d = repelem(170/(N-1),(2*N - 2));
     % Ensure flows do not have the intralocal trips, as they do not have
     % intra metrostations and they can be ignored.
     F_nd = F - diag(diag(F));

     Flows = zeros(size(s)); % Initialize the Flows vector with zeros

     for x = 1:numel(so) % x is the number of both forward and backward links for both tracks
         % For a link x, the link flows will be sum of columns
         % (destimations) from the x+1 to N and sum of rows origins from 1
         % to x (Forward Trips). Vice versa for backward trips
         Flows(x) = sum(sum(F_nd(1:x, x+1:N))); % Instance if N = 5, Flows(1) = sum (sum(F_nd(1,2:5))),i.e the sum of
flows from 1,2 + 1,3,+ 1,4 + 1,5
         Flows(N-1+x) = sum(sum(F_nd(x+1:N, 1:x)));
     end
 end
```

Figure 1-3: Graph showing the Link Flows and Distance for The Line Layout for N = 10 and N = 100.



From Figure 1-3, The link flows for each link on the line were plotted with the distance from station 1 to other stations for both N = 10 and N = 100.

Link flows for N = 10 peak when travelling from node 4 to 5. The link flows for N = 100 is much smoother or rounded than for N = 10, this create a scenario that can be called a ‘middle belt bottleneck’. These stations that have similar link flows that are within to 10% of the upper limit of link flows on the Line. If a designer chooses to have a high number of stations or increase the number of stations from N = 10, i.e 10 < N < 100, the ‘middle belt bottleneck’ effect would increase i.e the frequency of ‘middle belt’ stations will increase.

If the localities or stations were spaced on a circular layout, the behaviour of individuals or workers on the network would be different.

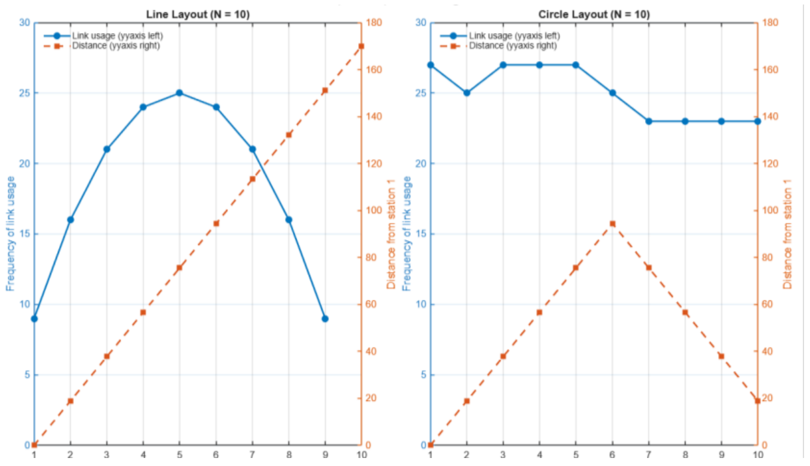


Figure 1-4: Graph comparing Circular Layout and Line Layout, Frequency of Link Usage and Distances for N = 10

Figure 1-4 and Figure 1-5, shows the frequency of link usage and Distance on both the Line and Circular Layout for N = 10 and N = 100.

For both N = 10 and N = 100, there is a correlation between frequency of link usage and link flows in Figure 1-3, Figure 1-4, and Figure 1-5. For a circular layout, the range amongst link usage is lower compared to the line layout.

Would the link flows on a circular layout behave similarly to link usage?

Yes, it would.

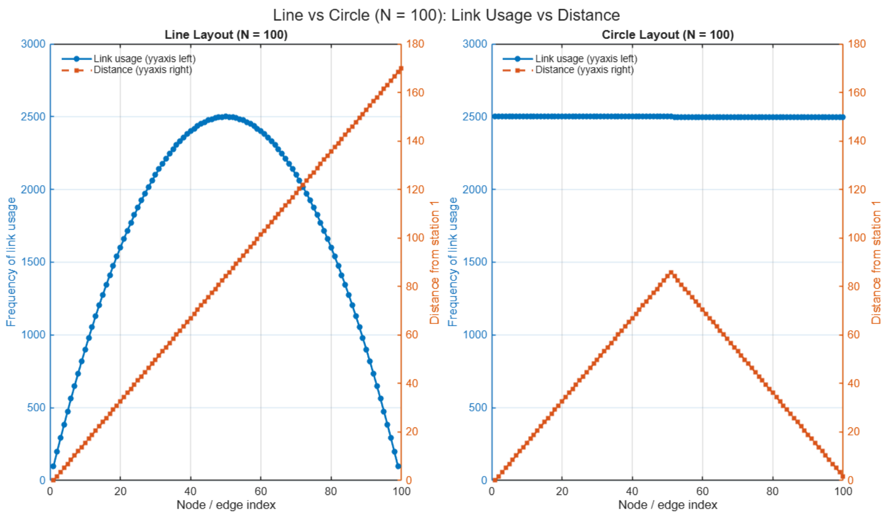


Figure 1-5: Graph comparing Circular Layout and Line Layout, Frequency of Link Usage and Distances for N = 100

The gravity model uses distance as a cost variable to model link flows. For N = 100, the maximum distance has decreased from 170km to $\approx 86km$. Also, the maximum distance happens around N/2 - 1 stops from any source on a circular layout. There is also increased connectivity for far distance stations. This implies individuals would make the trips that they found more costly in the Line model. The flows would increase on those links, while the '*middle belt*' effect would decrease due to the balance created within the model. The middle belt effect can also be seen in the centrality of nodes.

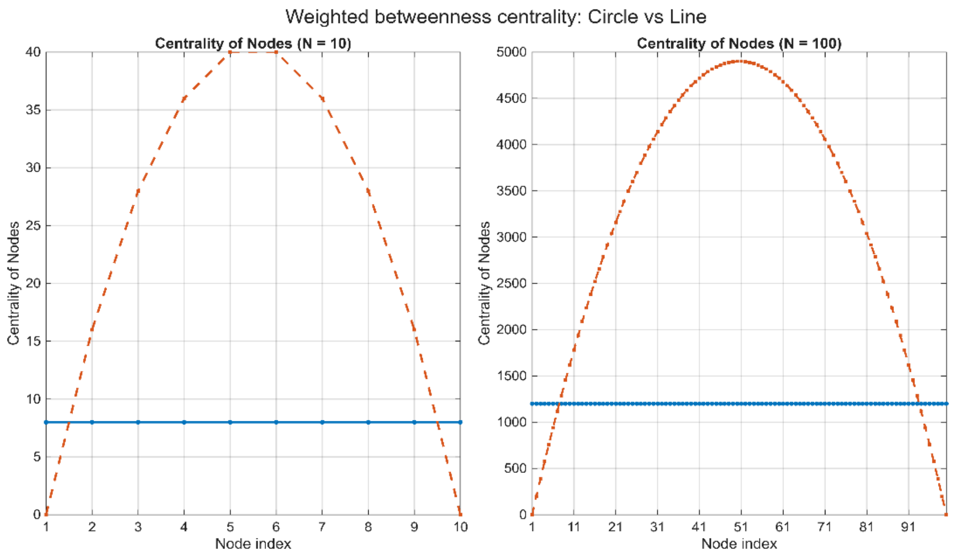


Figure 1-6: The Centrality of nodes on the Circle (blue) and on the Line (orange)

QUESTION 2

A) THE CIRCLE CITY SHORT-CUTS

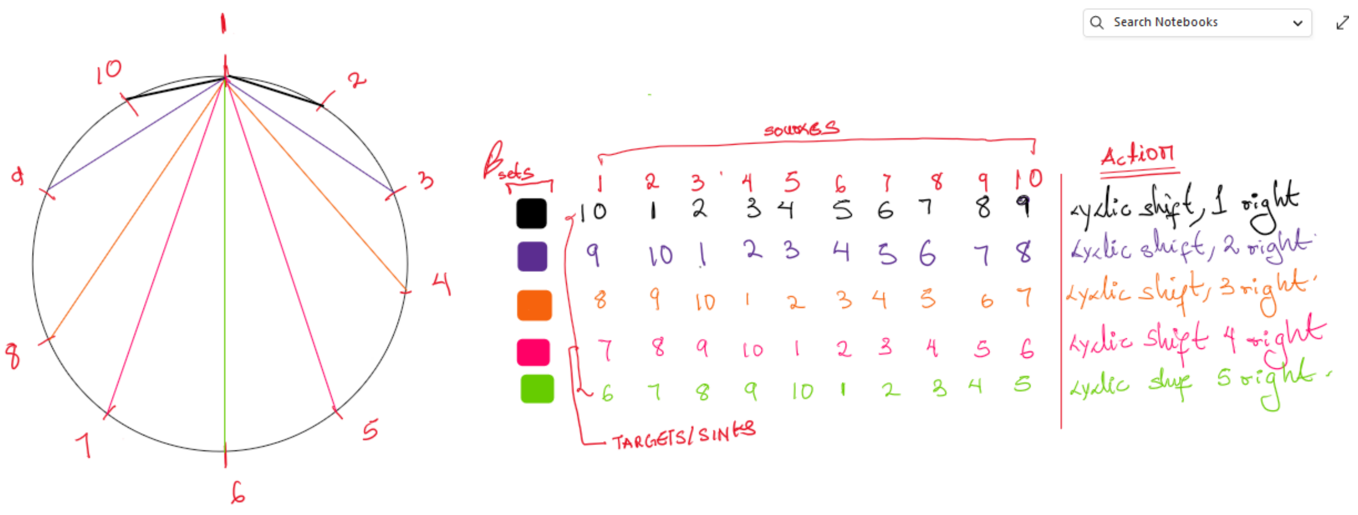
The following are the codes used in implementing the addition of shortcuts to the network model. They follow the workflow:

$$\text{graph\_add\_edge\_circle(N)}(Graph\_Circle\_(N) \rightarrow \text{add\_edge\_circle(N)})$$

Graph Circle, create the sources and sinks for a circular network with node N. Add_edge_circle adds edges to represent the shortcuts and graph_add_edge(N) graphs the network. The number of shortcut sets with the same length = $\frac{N}{2}$. For any shortcut length $\beta L,$ number of shortcuts of that length = N if $\beta L < Diameter\ of\ the\ Circular\ Network$.When $\beta L, = Diameter$, there are $\frac{N}{2}$ lines.

```
function [s,t,d]= Graph_Circle_(N)
% GRAPH_Circle_ Function to create the source and sinks on'The Circle'
    % Example of usage
    % N10 = 10;
    % [sc,tc,dc]= Graph_Circle_(N10)
    % G_Circle_10 = graph(sc,tc,dc)
    % disp('Graph for N = 10 (Circle):');
    %
    % theta = linspace(0, 2*pi, N+1); theta(end) = []; % Parametric circle coordinates
    % Equation of a Circle x = rcosp, y = rsinp, p = [0 to 2*pi)
    % x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
    % plot(G_Circle_10, 'XData', x, 'YData', y);
    % title('Graph Representation for N = 10 (Circle)');
    %
    %
    % Input Arguments:
    %     N - number of metrostations or localities on The Line
    %
    % Output Arguments:
    %     s - sources
    %     t - sinks or targets
    %     d - distance for each edge on the circle

    % Create the source and sink vectors for the line
    % As it is a double track metrostation, flows can originate at 1st and
    % the Nth Station.
s = (1:N); % Create the source nodes representing all metrostation localities on the
Circle that can be used in a graph function on MATLAB
t = circshift(s,-1); % Create the target nodes representing all metrostation localities
on the Circle
% the circshift function shifts the list of nodes to have a represent a
% cyclic flow from 1 to N and N to 1
% if N = 5,  s = [1,2,3,4,5] t = [2,3,4,5,1]
d = repelem(170/(N-1),N); % This repeats the distance 170/(N-1) on an array of length N
end


N10 = 10;
N100 = 100;
```

```
[sc,tc,dc]= Graph_Circle_(N10);
G_Circle_10 = graph(sc,tc,dc);
disp('Graph for N = 10 (Circle):');

theta = linspace(0, 2*pi, N10+1); theta(end) = [];
x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
plot(G_Circle_10, 'XData', x, 'YData', y);
title('Graph Representation for N = 10 (Circle)');

[sc100,tc100,dc100]= Graph_Circle_(N100);
G_Circle_100 = graph(sc100, tc100, dc100);
disp('Graph for N = 100 (Circle):');
theta = linspace(0, 2*pi, N100+1); theta(end) = [];
x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
plot(G_Circle_100, 'XData', x, 'YData', y);

title('Graph Representation for N = 100 (Circle)');
```



sets	1	2	3	4	5	6	7	8	9	10	Action
⬛	10	1	2	3	4	5	6	7	8	9	cyclic shift, 1 right
🟪	9	10	1	2	3	4	5	6	7	8	cyclic shift, 2 right
🟧	8	9	10	1	2	3	4	5	6	7	cyclic shift, 3 right
🟥	7	8	9	10	1	2	3	4	5	6	cyclic shift 4 right
🟩	6	7	8	9	10	1	2	3	4	5	cyclic shif 5 right.

SOURCES

TARGETS/SINKS

Figure 2-1:Handwritten Implementation of the Cyclic Shifter Loop. There are N/2 sets of distinct βL lengths around a circle.

```
function [s, t, CL,B] = add_edge_circle(N)
% add_edge_circle to create the shortcuts on'The Circle'
    % Example of usage
    % [as10, at10 ,CL10 ,B10]= add_edge_circle(10)
    %
    % Input Arguments:
    %     N - number of metrostations or localities on The Line
    %
    % Output Arguments:
    %     s - 1 by 2N by N/2 array:sources
    %     t - 1 by 2N by N/2 array:sinks or targets
```

```
     %     CL -1 by N/2 array:distance or Chord Length for each shortcut edge on the
circle
     %     B - 1 by N/2 array:Beta, the scalling parameter for the length of shortcut
lines
     %     such that BL = length of shortcut line. B_max is the diameter of
     %     the Circle and B_min is the smallest chord between two stations

 % Initialise all ouput Arguments
 s = zeros(1,2*N,N/2);
 t = zeros(size(s));
 CL = zeros(1,N/2);
 B = zeros(1,N/2);

 % If the Number of stations are N
 % The number of shortcut line sets of the same length BL a station can 'create' is  K =
N/2 for even number of N
 % N-1/2
 %
 % starting angle 0 to pi stopping angle on the diameter
 a_rad = linspace(pi/N,pi,N); a_rad((N/2)+1:end) = []; % a_rad are the angles from station
1 to a station N/2 from it

 Chord_Lengths = (170/pi)*sin(a_rad); % Using the formula of chord lengths 2rsin(theta) or
L/2pi(sin(theta)
 Beta = (1/pi)*sin(a_rad); % Beta  = L (The length of the Circle) / Chord_Lengths

 for i = 1:N/2
     % This loop is the bedrock of the code. A handwritten implementation
     % has been attached within this section of the report explaining how
     % the cyclic shift loop works.
     s(1,1:2*N,i) = repmat((1:N),1,2); % Kick of the shortcut edge creation by stating
with the sources
     % declare all the nodes twice; Example for N = 3
     % s  = [1 2 3 1 2 3]
     % shift the source nodes cyclically to the right
     % t1 = [3 1 2]
     % shift the source nodes cyclically to the left
     % t2 = [2 3 1]
     % t = [t1 t2] = [3 1 2 2 3 1]
     % The All Possible Shortcut Edge List
     % 1 to 3
     % 2 to 1
     % 3 to 2
     % 1 to 2
     % 2 to 3
     % 3 to 1

     t(1,1:N,i) = circshift(s(1,1:N,i),i); % Cyclically shift source nodes to the right by
i

     t(1,N+1:2*N,i) = circshift(s(1,N+1:2*N,i),-i); % Cyclically shift source nodes to the
right by i
     % This was done to enable digraph functionality but its later ignored
     % in future codes

     CL(1,i)= Chord_Lengths(i); % Add the Length of the Chord Length
     B(1,i) = Beta(i); % Add Beta to aid functionality of the function

 end
 end

 [as10, at10 ,CL10 ,B10]= add_edge_circle(10);

 function shortest_paths_nodes = shortest_paths_(G, N)
 % shortest_paths_
 %       This function prepares a list of nodes on the shortest path between two
 %       nodes source and target
 % Inputs:
 %       G - The Graph or Network - Must be a a class graph or digraph
 %       N - number of nodes
 %       undirected- can be used if the network is a plan graph
 % Outputs:
 %       shortest_paths_nodes     : N×N cell array, shortest path between all nodes on a
graph

 shortest_paths_nodes = cell(N, N);

 % Create two matrices the shortest path between all nodes can be found
 I_n = repelem([1:N].', 1, N); % repeats each node on each column
 I_s = [repelem([1:N].', 1,  N)].'; % repeats each node on each row
 s = 1:N; % The source node ids
 n = 1:N; % The target node ids

 % Instead of a for Loop - cellfun is used to run the function for each node
 % where  i is used to iterate over all the source nodes
 % j is used to iterate over all the target nodes
 shortest_paths_nodes = cellfun(@(i, j) shortestpath(G, s(i), n(j)), ...
                                    num2cell(I_s), num2cell(I_n), 'UniformOutput',
false);

 end

 function  [EdgeCount, Sp_array, Distances] = Edge_Freq_Shortestpath(G, t, s, N)
     % Edge_Freq_Shortestpath
     %       This function counts the number of times and edge or link is used within a
list of shortest paths on a network
     % Inputs:
     %       G - The Graph
     %       t&s - target and source nodes
     %       N - number of nodes
     % Outputs:
     %        Sp_Array: The Shortest Path Node Sequence
     %        EdgeCount: The frequency of links with all possible shortest paths
     %        between nodes on a network
     %        Distances : The Shortest Path Distance
     % Dependencies:
```

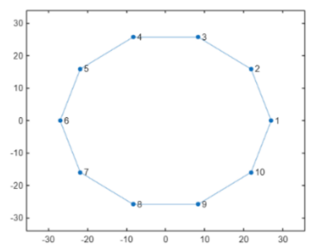
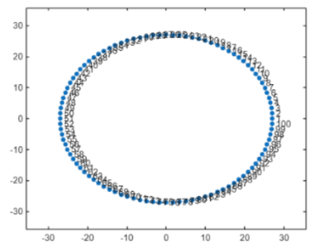
```
    %      1) shortest_paths_: A function with inputs:  G - class: graph or digraph; The
Graph or Network
    %       N - number of nodes  and output: N×N cell array, shortest path between all
nodes on a graph
    %      2) edge_count_per_path : counts the number of times and edge or link is
    %       used within for a given shorthest path node.

    undirected = ~isa(G,'digraph');  % This checks to see what type of network it is

    Edges = G.Edges.EndNodes;
   E = size(Edges,1); % The number of Links or Edges in the Network.
    Sp_array = shortest_paths_(G, N); % This calls the function shortest_paths to create
an array of shortestpaths for all nodes
    counts_per_cell = cellfun(@(pathlink) edge_count_per_path(pathlink, Edges,
undirected), Sp_array, 'UniformOutput', false);

    if isempty(counts_per_cell)
        CountArray = zeros(E,1);
    else
        CountArray = sum(cat(2, counts_per_cell{:}), 2);
    end

    EdgeCount = [EN(:,1) EN(:,2) CountArray];
    Distances = distances(G);
end
function counts = edge_count_per_path(path_nodes, Edges, undirected)
    % edge_count_per_path
    %       This function counts the number of times and edge or link is
    %       used within for a given shorthest path node.
    % Inputs:
    %       Edges - All the edges of the graph
    %       undirected
    %       path_nodes - the sequence of shortest path nodes for a path
    % Output:
    %     counts
    E = size(Edges,1);
    counts = zeros(E,1);

    if numel(path_nodes) < 2
        return; % This ensures the if only one node is called the count would be a zero
as it is not a true link / edge
    end

    links = [path_nodes(1:end-1).'  path_nodes(2:end).']; % This breaks the path into
links such that if the shortest path nodes is [1,2,3,4,5 ] = [1,2;2,3;3,4;4,5]

    if undirected == 1
        links = sort(links,2); % This ensures that in an undirected graph it links are
stored in canonical order to ensure it matches the Edges List.
    end
    [Lia, Locb] = ismember(links, Edges, 'rows'); % This line of code ensures that each
link represented is counted with the position of the link in Edge
```

```
      % Accumarray is used to represent the entire edges list and ensures the edges are
counted such that
      % if link 1,2 is a member of Edges, the position of link 1,2 in Edges would have a
1.
      % if link 1,2 is has been repeated three times in the 'links' of path_nodes,the
position of link 1,2 in Edges would have a  3
     counts = accumarray(Locb(Lia), 1, [E 1]);
 end


 function G_add_edge = graph_add_edge_circle(N)
 % graph_add_edge_circle is used to create the shortcuts on 'The Circle' graph
     % Example of usage
     % G_add_edge10 = graph_add_edge_circle(N10);
      %theta = linspace(0, 2*pi, N10+1); theta(end) = [];
     %x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
      %plot(G_add_edge10{1,1,1}, 'XData', x, 'YData', y);

     %
     % Input Arguments:
     %     N - number of metrostations or localities on The Line
     %
     % Output Argument:
     %    G_add_edge - the cell array of graphs and other variable
     %        G_add_edge{1,1,B_num} - The  Graphs of the Circle with added
     %           shortcuts, B_num = 1 is the plain Circle graph without shortcuts
     %        G_add_edge{1,2,B_num} - EdgeCount: The frequency of links with all possible
shortest paths
     %        G_add_edge{1,3,B_num} - Sp_Array: The Shortest Path Node Sequence
     %        G_add_edge{1,4,B_num} - Distances : The Shortest Path Distance
     %        G_add_edge{1,5,B_num} - Average Shortest Path Length over all
     %           N(N-1)/2 ordered pair nodes
     %        G_add_edge{1,6,B_num} - Beta Value
     %        G_add_edge{1,7,B_num} - Centrality of Nodes
     %
     % Dependencies
     %   1) Graph_Circle_ : to create the source and sinks on'The Circle'
     %   2) add_edge_circle: to create the all possible edges list of source,targets,
     %   chord lengths and Beta
     %   3) Edge_Freq_Shortestpath(): To calculate the
     %   4)

 % Initialise all variables

 [s,t,d] = Graph_Circle_(N); % Create the source and sinks on'The Circle'
 G = graph(s,t,d); % Add them to a graph

 [sae, tae, CL,B] = add_edge_circle(N); %create the all possible edges list of
source,targets, chord lengths and Beta


 G_add_edge = cell(1,7,((N/2)+1)); % Initialize the cell array G
 G_add_edge{1,1,1} = G; % Initialize the plain Circle graph without shortcuts
 G_add_edge{1,5,1} = 0; % Beta = 0

 [G_add_edge{1,2,1},G_add_edge{1,3,1}, G_add_edge{1,4,1}] =
Edge_Freq_Shortestpath(G_add_edge{1,1,1}, t, s, N); % Calculate its link usage, distances
and the sp array

 G_add_edge{1,6,1} = sum(sum(G_add_edge{1,4,1}))/(N*(N-1)); % Sum of all shortest path
lengths or distance/ number of distinct pair of nodes.

 for i = 2:(1 + N/2)
     j = i - 1; % To ensure proper indexing.
     if i == (1 + N/2) % For the Diameter Length BL
         G_add_edge{1,1,i} =
addedge(G_add_edge{1,1,j},sae(:,1:N/2,j),tae(:,1:N/2,j),CL(j)); % Adds all source and edge
nodes with the distance to the graph
         % The definition of addedge for The diameter length BL is different from others
because we only need a single line of BL not two.
         [G_add_edge{1,2,i}, G_add_edge{1,3,i},G_add_edge{1,4,i}] =
Edge_Freq_Shortestpath(G_add_edge{1,1,i}, t, s, N); % Calculate its link usage, distances
and the sp array
         G_add_edge{1,5,i} = B(j);% Calculate its link usage, distances and the sp array
         G_add_edge{1,6,i} = sum(sum(G_add_edge{1,4,i}))/(N*(N-1));% Calculate its link
usage, distances and the sp array
     else % For the other Lengths BL
         G_add_edge{1,1,i} = addedge(G_add_edge{1,1,j},sae(:,1:N,j),tae(:,1:N,j),CL(j)); %
Adds all source and edge nodes with the distance to the graph
         [G_add_edge{1,2,i}, G_add_edge{1,3,i},G_add_edge{1,4,i}] =
Edge_Freq_Shortestpath(G_add_edge{1,1,i}, t, s, N);% Calculate its link usage, distances
and the sp array
         G_add_edge{1,5,i} = B(j);% Calculate its link usage, distances and the sp array
         G_add_edge{1,6,i} = sum(sum(G_add_edge{1,4,i}))/(N*(N-1));% Calculate its link
usage, distances and the sp array
     end
 end
 for i = 1:(1 + N/2)

  G_add_edge{1,7,i} =
centrality(G_add_edge{1,1,i},'betweenness','Cost',G_add_edge{1,1,i}.Edges.Weight);

 end

 end


 G_add_edge10 = graph_add_edge_circle(N10);
 theta = linspace(0, 2*pi, N10+1); theta(end) = [];
 x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
 plot(G_add_edge10{1,1,1}, 'XData', x, 'YData', y);
```

ILLUSTRATION OF GRAPH REPRESENTATION OF THE CIRCLE FOR N = 10 & N =100 WITH ADDITIONAL SHORTCUT LINES

N = 10 **N = 100**

$\boldsymbol{\beta = 0}$

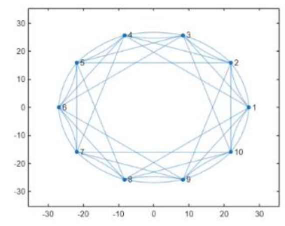
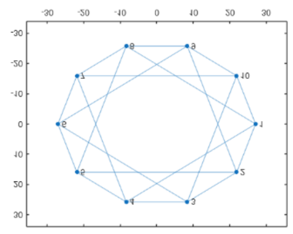
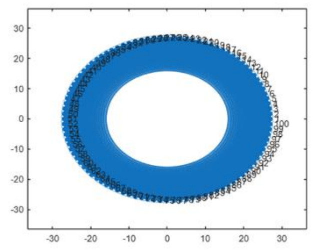
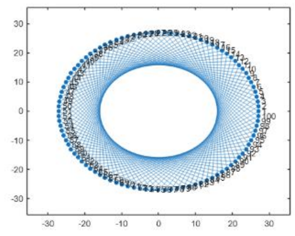
Method 1 Lines Length $\leq \beta L$

Method 2 Lines Length $= \beta L$

Method 1 Lines Length $\leq \beta L$

Method 2 Lines Length $= \beta L$

$\boldsymbol{\beta = 0.3027}$

$\boldsymbol{\beta = 0.3183}$

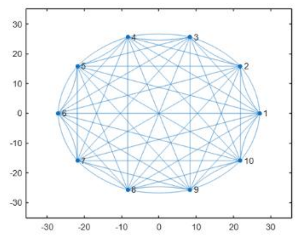
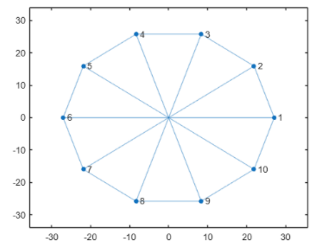
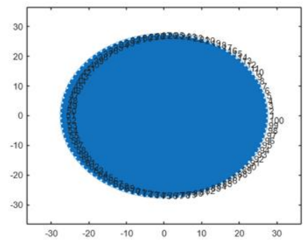
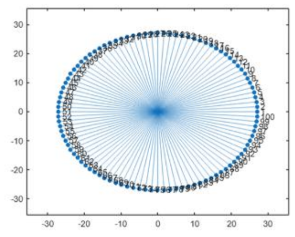





Figure 2-2: Graphical representations of the metro circle design for L = 170km, for N = 10 & N = 100 metro stations. The illustration shows two ways in which additional line could be added to the circle model. Method 1: Add lines with lines Lines Length $\leq \beta L$ and Method 2: Add Lines with Lengths $= \beta L$

The city hopes to build shortcut train tracks to decrease the distance along the metro line. The efficacy of the model is how well the shortcut decreases the distance or the length of travel on the line.

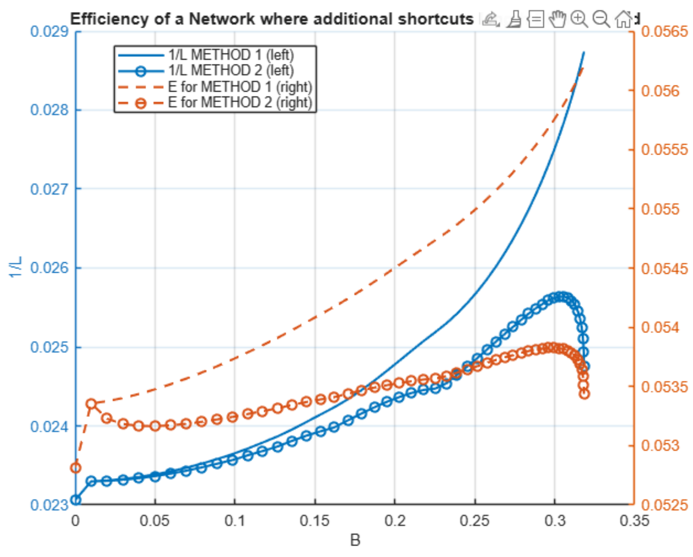
To measure the efficacy of the short cut links, two parameters are used, Inverse average path length and the Efficiency of the network. If the average length decreases, the network is faster to use, hence its inverse can represent how efficient it is to add a shortcut to the line. The Inverse average path length is defined as $\frac{1}{L}$. The Efficiency of the network can be defined as $E = \frac{1}{N(N-1)} \sum_{i \neq j} 1/d_{ij}$, where d is the distance between to nodes I and j. It represents the global efficiency of the network. It is a preferable measure to $\frac{1}{L}$ as it can measure the efficiency of a parallel system (3). Parallel systems are typical in transport networks as people travel concurrently on a network (3).

Both methods 1 and 2 shown in Figure 2-2, were modelled using the length of the added line as the constraining factor adding new lines to the model. Figure 2-3 illustrates the efficacy of the network for both Methods (1 & 2) and Efficiency Measures ($\frac{1}{L}.\, and\ E$).

- As $\boldsymbol{\beta}$ increases, both efficacy measures ($\frac{1}{L}$ and E) increase. This shows that as longer shortcut lines are added to the network. Shorter routes to far distances are available, which increases the likelihood of shortcut lines being used over other longer ones. Method 1 rises faster as there are many alternative routes on the model, allowing for better efficacy (faster routes or shorter path distance)
- A similarity between Methods 1 & 2 is seen on adding a shortcut to the model; there is a sharp rise in the efficiency of the model. This can be seen more clearly in the N =10 graph. This jump illustrates the effect of adding links with a lower weight (shorter distance) between nodes of a network. The original network becomes redundant as it's longer and less preferable to the faster (shorter distance) link.
- Method 2 saturates at $\beta = 0.3$, then the efficiency drops, while Method 1 increases. The method of modelling the shortcuts for Method 1 illustrates the limits of adding longer shortcut routes to the model. Adding longer routes without adding more alternative paths worsens the efficacy of the model.
- Comparing N = 10 and N = 1000

N = 100

N = 10



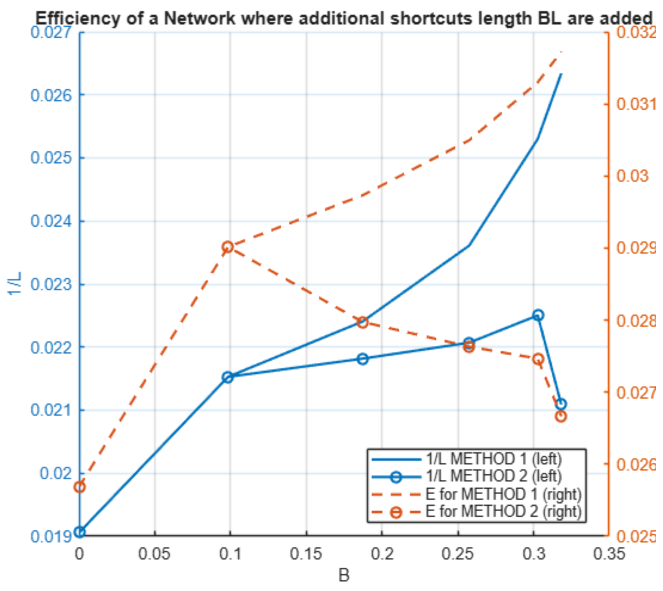
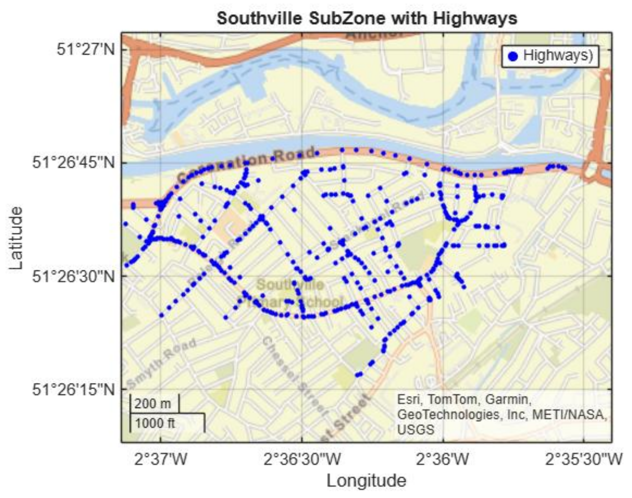
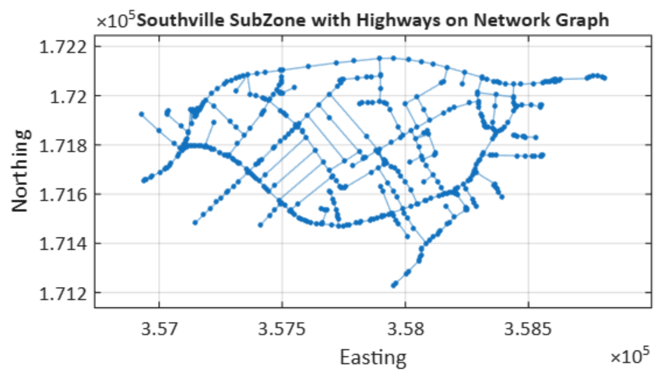


Figure 2-3: The Efficiency of a Network for both Methods for N = 100 and N = 10

Apart from efficiency and the length of additional line on the model another parameter that needs to be considered is the cost. Let's assume the cost of developing a new line is $238.21 million per km [4]. The cost of adding will increase faster with Method 1 compared to Method 2, this is because of the presence of more links in the model. A suitable trade off would be having reduced cost and more alternative routes on the



(a)The map representation of Southville sub-zone showing with all types of Highways



(b) The network representation of Southville sub-zone showing with all types of Highways

Figure 2-4:Map (a) and Graph (b) representation of Southville sub-zone (Details on 0Code Implementation of Southville can be found in the Appendix)

B) SOUTH BRISTOL LIVEABLE NEIGHBOURHOODS – SOUTHVILLE

South Bristol Liveable Neighbourhoods (SBLN) project aims to make routes within the specified regions were to make the local streets "healthier, more accessible, and more pleasant". Some of the general improvements that were suggested including having less traffic and slower speeds. Southville is one of the proposed subzones. The top three challenges that residents of Southville face are, difficulties in parking, fast moving traffic and the lack of crossings. The road brought forward include are Coronation Road, Stackpool Road and Dean Lane.

The interventions that can be analysed with a modelling technique would be increasing the pathways for foot and cycle traffic, which include adding crossing to the model

The steps I would follow to evaluate the schemes impact include:

1. Analysing the types of roads or highways on Coronation Road, Stackpool Road and Dean Lane. The map.xml data from the OpenStreetMap, contains detailed information on the number of lanes, footways, crossings, cycle lanes, type of surface, and the speed limits on those highways. Parameters such as projected distance, and speed limits can be assigned at weight parameters with distance on those links

2. Locate nodes and edges within those streets that (and do not) specify the different types of pedestrians, vehicular, and cycle paths.

3. Measure the following parameters:
   (1) **Degree of Nodes** within the same highway type& different highway types that represent more than one class.
   (2) **Betweenness Centrality** of the Network in terms of the connections between the nodes of the same highway type& different highway types
4. Adding & Removing nodes and edges, and measuring the same parameters

5. Add cost variables as weights on the edges for adding or removing nodes. The cost variables could be terrible pavements on the pathways, delay because of the added node or additional distance or the cost of adding a new line.

The efficacy of the model can be identified when there is balance or trade-off between the degree, connectivity and cost.

QUESTION 3

A) STACKELBERG GAME

Defined Parameters of the Stackelberg game

I. **Demand** d – This represents the number of trips that need to be made. It is split into two components
   i. $d_A = \propto d$ – the demand autonomous vehicles have on the model, aimed at minimising the total system cost for all users.
   ii. $d_L = (1{-}\propto)d$ – the demand legacy vehicles have on the model aimed at minimising owns costs i.e. user equilibrium
   $d_A + d_L = d\ ,\ d \leq 0$
II. **Link Flow** – The distribution of the demand across the network or the number of vehicles on a link of the two parallel link system
   a. $x_1\ = x_1^{(A)} +\ x_1^{(L)}$– flows on a link 1 where $x_1^{(A)}$ for autonomous vehicles is the $x_1^{(L)}$ for legacy vehicles. $x_1 \leq 0$
   b. $x_2 = x_2^{(A)} +\ x_2^{(L)}$– flows on a link 2. $x_1 \leq 0$
III. **Link Cost** – Price of using a link
   a. $f_1(x_1) = a + x_1,\ 0 \leq a \leq 1$
   b. $f_2(x_2) = 1 + bx_2,\ 0 \leq b \leq 1$
IV. **Constants**
   a. a – constants of the $f_1(x_1)$ function. Shifts the intercept of the link cost function $f_1(x_1)$ to value
   b. $b - f_2(x_2)\ constant.$ Manipulates the slope of the link cost function $f_1(x_1)$
   c. $\propto$ (alpha) - proportion of autonomous vehicles on the two parallel link model $0\ \leq\ \alpha \leq 1.\ \alpha = 1 - is\ system\ optimal\ case\ , \alpha = 0 - user\ equilibrium$

Stackelberg Game

The Stackelberg game models a mixed-flow system where both autonomous and legacy vehicles make a choice on what link to take on the transport network. The decisions made by both players are route or trip assignments i.e. whether they choose link 1 or 2 on the model. The parameter that reflects the trip assignments in the model is the link flows. At the start of the game, we assume equal link flows on both $lanes\ 1\ \&\ 2, x_1 = x_2)$the model. The autonomous vehicles stick with their previous or initialised link (link flows for autonomous vehicles remains the same). Legacy vehicles make the first move motivated by 'selfish interest' to minimise their own costs. The autonomous vehicles act after the legacy vehicles have made their new choice with the goal of ensuring the model becomes system optimal i.e. the cost of using the net travel cost of the network. During this stage the legacy vehicles stick to their original choice. The choices are made by both players until the system converges i.e. link flows are constant regardless of any further decision made.

Code Workflow

1. **Stackelberg_Game(d)** – starts the game by setting global parameters based on the demand d
2. **Step 0 functions**
- system_optimiser_step0 – broadcasts the a, b, and d constants to quad_prog_solver_step0 for $0 \leq a, b \leq 1$
- quad_prog_solver_step0 – minimises the total system cost function using quadprog.

$$F = x_1 f_1(x_1) + x_2 f_2(x_2)$$
$$= x_1^2 + bx_2^2 + ax_1 + x_2$$

This form equation can be solved using quadprog, by using the following:

1. H- which holds the quadratic objective terms
2. f - which holds the linear objective terms
3. A (Matrix)– Linear inequality constraint
4. b (vector)-Linear inequality constraint
5. Aeq (Matrix)- Linear equality constraint
6. $b$eq (vector) - Linear equality constraint
7. lb – the lower bounds
8. ub – the upper bounds

3. Step n functions

- User Equilibrium
    - user_equilibrium_step_n:
    - quad_prog_solver_ue_stepn: Minimises the Beckmann transformation

Using Beckmann formulation,

$$\hat{f} = \int_0^{x_1^L} f(x_1^L)\,dx_1^L + \int_0^{x_2^L} f(x_2^L)\,dx_2^L$$
$$\hat{f} = \int_0^{x_1^L} (a + x_1^A + x_1^L)\,dx_1^L + \int_0^{x_2^L} (1 + b(\,x_2^A + x_2^L)\,dx_2^L$$
$$\hat{f} = ax_1^L + \frac{1}{2}(x_1^L)^2 + x_1^A x_1^L + x_2^L + b\frac{1}{2}(x_2^L)^2 + bx_2^A x_2^L$$

- System Optimal
    - system_optimiser_step_n
    - quad_prog_solver_so_stepn

$$F = x_1 f_1(x_1) + x_2 f_2(x_2)$$
$$= (x_1^{(A)} + x_1^{(L)})^2 + a(x_1^{(A)} + x_1^{(L)}) + b(x_2^{(A)} + x_2^{(L)})^2 + x_2^{(A)} + x_2^{(L)}$$
$$F = F_0(x) + F_c(x)$$

Objective Function

$$F_0(x) = \left(x_1^{(A)}\right)^2 + 2x_1^{(A)}x_1^{(L)} + ax_1^{(A)} + b\left(x_2^{(A)}\right)^2 + 2bx_2^{(A)}x_2^{(L)}$$

Objective Constant

$$F_C = \left(x_2^{(L)}\right)^2 + ax_1^{(L)} + b\left(x_2^{(L)}\right)^2 + x_2^{(L)}$$

4. plot_F_Cost- Plots the costs for a given demand and cost output

Table 3-1; Showing the list of objective terms, constraints and bounds using in minimising the system cost on the two link model.

Parameters for quadprog	**H**	$\boldsymbol{f}$	$\boldsymbol{A}$	$\boldsymbol{b}$	$\boldsymbol{Aeq}$	$\boldsymbol{beq}$	$\boldsymbol{lb}$	$\boldsymbol{ub}$
System Optimal Step 0- quad_prog_solver_step0	$\begin{bmatrix} 2 & 0 \\ 0 & 2b \end{bmatrix}$	$[a; \quad 1]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$[d]$	$[0 \quad 0]$	$[d \quad d]$
User Equilibrium Step n – quad_prog_solver_ue_stepn	$\begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$	$[a + x_1^A; \quad 1 + bx_2^A]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$[(1-\alpha)d]$	$[0 \quad 0]$	$[(1-\alpha)d \quad (1-\alpha)d]$
System Optimal Step n- quad_prog_solver_so_stepn	$\begin{bmatrix} 2 & 0 \\ 0 & 2b \end{bmatrix}$	$[a + 2x_1^L; \quad 1 + 2bx_2^L]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$(\alpha)d$	$[0 \quad 0]$	$[\alpha d \quad \alpha d]$

[] – implies empty

```
function [x12_d_cay,xa_d_cay,xl_d_cay] = system_optimiser_step0(d)
    % system_optimiser_step0()- This initialises the game by performing the following
    %  - Sets the propotion of legacy and autonomous vehicles to d/2 and equal
    %  - Calculates the system optimal costs on the network by minimising
    %  the Total System Cost equation using quadprog.
    % Input Arguments:
    %   d - The total number of trips needed to be made
    %
    %
    % Output Arguments:
    %   x12_d_cay -
    %   xa_d_cay -
    %   xl_d_cay -
    % Dependencies
    %  quad_prog_solver_step0 - Takes each constant for each link and returns the link
    %  flows on links 1 & e

    % Initialise the constatnts of the cost
    a = linspace(0,1,11);
    J_b = repmat((1:11)', 1, 11);
    b = linspace(0,1,11);
    I_a = repmat(1:11, 11, 1);

    % Calls quadprog to solve for all constants a,b
    x12_d_cay = cellfun(@(i, j) quad_prog_solver_step0(a(i),b(j),d), ...
                                num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);
    alpha = 0.5;
    xa_d_cay  = cellfun(@(i) i * alpha  , x12_d_cay, 'UniformOutput', false);

    xl_d_cay = xa_d_cay ;
end

function x_d_ar = quad_prog_solver_step0(a,b,d)
%  quad_prog_solver_step0(a,b,d): Minimises the cost function
%
% Input Arguements:
%   a - constant for link flow x1 equation
%   b - constant for link flow x2 equation
%   d - trip demand on the network
%
% Output Arguements
%   x_d_ar - This is the link flows on both links of the network


H = [2,0
    0,2*b];


f = [a; 1];

options = optimoptions('quadprog','Display','off');
```

```
 Aeq = [1, 1];
 beq = d;
 lb = [0 0]; % Set the upper bounds of te
 ub = [d d];

 x_d_ar = quadprog(H, f, [], [], Aeq, beq,lb,ub,[],options); % Minimises the cost function
 end

 function [xa_now,xl_now,F,error, iteration]= Stackelberg_Game(d)
   % STACKELBERG_GAME - Runs the Stackelberg game until convergence starts the game by
setting
   % global parameters based on the demand d
   %
   % Input arguments:
   %     d - the total demand
   %
   % Output arguments:
   %     xa_now, xl_now - final autonomous and legacy link flows
   %     C -
   %     error - maximum error from previous and present iterations
   %     iteration - iteration count

 % Initialise the variables & the convergence limits
 Max_Iteration = 110;
 Min_error = 1e-6;

 % Start the stackelbeg game by running the equation until convergence
 [x12_d_cay,xa_d_cay,xl_d_cay] = system_optimiser_step0(d);
 iteration = 1;
 error = Inf;

 while ((error > Min_error)  && (iteration < Max_Iteration))

 % First Iteration n = 1
 if iteration == 1
     % Compute user equilibrium from initial system optimiser outputs
     x_l_d_ar_1 = user_equilibrium_step_n(x12_d_cay,xa_d_cay,xl_d_cay,d,iteration);

      % Optimise the system to reduce system costs
     [xa_fin_d_ar_1,xl_fin_d_ar_1,F] =
system_optimiser_step_n(x12_d_cay,x_l_d_ar_1,xa_d_cay,d,iteration);

 else
     % Use previous iteration outputs to compute next user equilibrium and
     % system optimal responses
     xa_prev = xa_fin_d_ar_1;
     xl_prev = xl_fin_d_ar_1;
     x_l_d_ar_1 = user_equilibrium_step_n(x12_d_cay,xa_prev,xl_prev,d,iteration);
    [xa_fin_d_ar_1,xl_fin_d_ar_1,F] =
system_optimiser_step_n(x12_d_cay,x_l_d_ar_1,xa_prev,d,iteration);
 end
 xa_now = xa_fin_d_ar_1;
 xl_now = xl_fin_d_ar_1;
 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);

 % Calculate the Errors on the model.
 if iteration == 1

     error_a = max(cell2mat(cellfun(@(i, j)
norm((cell2mat(xa_fin_d_ar_1{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput',
false)));
     error_l = max(cell2mat(cellfun(@(i, j)
norm((cell2mat(xl_fin_d_ar_1{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput',
false)));
 else

     error_a = max(cell2mat(cellfun(@(i, j) norm((cell2mat(xa_prev{i,j})-
cell2mat(xa_now{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput', false)));
     error_l = max(cell2mat(cellfun(@(i, j) norm((cell2mat(xl_prev{i,j})-
cell2mat(xl_now{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput', false)));
 end
 error = error_a + error_l; % Calculate total error on the model
 error = max(error); % Find its maximum
 iteration = iteration+1;
 xa_now = xa_now;
 end
 end



 function x_l_d_ar = user_equilibrium_step_n(x12_d_cay,xa_d_cay,xl_d_cay,d,n)
 % user_equilibrium_step_n: broadcast a and b on the quadprog solver
 %
 %
 % Input Arguements
 %   x12_d_cay - the link flows for the system
 %   xa_d_cay - the autonomous vehicle link flows
 %   xl_d_cay - the legacy vehicle link flows
 %   d - the demand
 %   n - iteration number
 % Output Arguements
 %   x_ld_ar - the minimised beckmann formulation link flows for the model
 % Dependencies
 %   quad_prog_solver_ue_stepn
 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed
 a = linspace(0,1,11);
 I_a = repmat(1:11, 11, 1);
 b = linspace(0,1,11);
 J_b = repmat((1:11)', 1, 11);
 if n == 1
     x_l_d_ar = cellfun(@(i, j)
quad_prog_solver_ue_stepn(a(i),b(j),d,x12_d_cay{i,j},xa_d_cay{i,j},xl_d_cay{i,j},n), ...
```

```
                                     num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);

 else
     x_l_d_ar = cellfun(@(i, j)
quad_prog_solver_ue_stepn(a(i),b(j),d,x12_d_cay{i,j},cell2mat(xa_d_cay{i,j}),
cell2mat(xl_d_cay{i,j}),n), ...
                                     num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);

 end
 end
 function x_l_ar_aplha = quad_prog_solver_ue_stepn(a,b,d,x12,xa,xl,n)
 % quad_prog_solver_ue_stepn - Minimises the Beckmann Formulation for a
 % certain a, b, and d
 % Input Arguements
 %   a,b - the constants for link flows
 %   d - the demand
 %   x12 - The link
 %   xa - autonomous link flows
 %   xl - leagacy link flows
 %   n -iteration count

 % Output Arguements
 %  x_l_ar_aplha - the legacy link flows

 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed
 x1 = x12(1);
 x2 = x12(2);

 % Objective Function Parameters
 H = [1,0
     0,b];

 lb = [0 0];

 Aeq = [1, 1];
 alpha = linspace(0,1,11);
 I = 1:11;
 options = optimoptions('quadprog','Display','off');

 if n == 1
     x1_a = xa(1);
     x2_a = xa(2);
     xl_0 = xl;

     f = [a+x1_a; 1+(b*x2_a)];
     x_l_ar_aplha = cellfun(@(i) quadprog(H, f, [], [], Aeq, [(1-alpha(i))*d],lb,[(1-
alpha(i))*d (1-alpha(i))*d],xl_0,options), ...
                                     num2cell(I), 'UniformOutput', false);
 else
     x1_a = xa(1,:);
```

```
    x2_a = xa(2,:);
    xl_0 = xl;

    %f = [a+x1_a; 1+(b*x2_a)];
    x_l_ar_aplha = cellfun(@(i) quadprog(H, [a+x1_a(i); 1+(b*x2_a(i))], [], [], Aeq, [(1-
alpha(i))*d],lb,[(1-alpha(i))*d (1-alpha(i))*d],xl_0(:,i),options), ...
                                    num2cell(I), 'UniformOutput', false);



 end
 end

 function [xa_fin_d_ar,xl_fin_d_ar,Fval] =
system_optimiser_step_n(x12_d_cay,x_l_d_ar,xa_d_cay,d,n)
 % system_optimiser_step_n() - broadcast a, b on to be used as variables for
 % the minimised Total Cost Function
 % Input Arguements
 %  x12_d_cay - The link flows 1 and 2
 %  x_l_d_ar - legacy vehicle link flows
 %  xa_d_cay - autonomous vehicle link flows
 %  d - delay
 %  n - iteration
 %
 %  Output Arguements
 %   Fval - The Total System Cost
 %   xl_fin_d_ar - legacy link flows
 %   xa_fin_d_ar - autonomous link flows
 %
 %   Dependencies
 %       quad_prog_solver_so_stepn
 %
 %



 % n -iteration count
 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed



 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed
    a = linspace(0,1,11);
    J_b =repmat((1:11)', 1, 11);
    b = linspace(0,1,11);
    I_a = repmat(1:11, 11, 1);

 if n == 1
 [xa_fin_d_ar,Fval] = cellfun(@(i, j)
quad_prog_solver_so_stepn(a(i),b(j),d,x12_d_cay{i,j},cell2mat(x_l_d_ar{i,j}),xa_d_cay{i,j}
,n), ...
                                    num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);
 else
 [xa_fin_d_ar,Fval] = cellfun(@(i, j) quad_prog_solver_so_stepn(a(i), b(j), d,
x12_d_cay{i,j}, cell2mat(x_l_d_ar{i,j}), cell2mat(xa_d_cay{i,j}),n), ...
                                     num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);
 end
 xl_fin_d_ar = x_l_d_ar;

 end


 function [x_a_ar_aplha,F] = quad_prog_solver_so_stepn(a,b,d,x12,xl,xa,n)
 % quad_prog_solver_so_stepn : Minimises the cost function for
 % certain a, b, and d
 % Input Arguements
 %   a,b - the constants for link flows
 %   d - the demand
 %   x12 - The link
 %   xa - autonomous link flows
 %   xl - leagacy link flows
 %   n -iteration count
 % x1_l & x2_l:- are fixed
 % x1_a & x2_a:- are being optimised
 x1 = x12(1);
 x2 = x12(2);
 x1_l = xl(1,:);
 x2_l = xl(2,:);
 xa_0 = xa;


 alpha = linspace(0,1,11);
 I = 1:11;

 H = [2,0
      0,2*b];


 Aeq = [1, 1];
 %beq = [(alpha)*d];
 lb = [0 0];
 %ub = [(alpha)*d (alpha)*d];

  options = optimoptions('quadprog','Display','off');
 if n == 1

     [x_a_ar_aplha, Fval] = cellfun(@(i) quadprog(H, [a+ (2*x1_l(i)); 1+(2*b*x2_l(i))],
[], [], Aeq, [(alpha(i))*d],lb,[(alpha(i))*d (alpha(i))*d],xa_0,options), ...
                                     num2cell(I), 'UniformOutput', false);

 else

    [x_a_ar_aplha, Fval] = cellfun(@(i) quadprog(H, [a+ (2*x1_l(i)); 1+(2*b*x2_l(i))], [],
[], Aeq, [(alpha(i))*d],lb,[(alpha(i))*d (alpha(i))*d],xa_0(:,i),options), ...
```

```
                                    num2cell(I), 'UniformOutput', false);

end

C = x2_l.^2 + (a.*x1_l) + (b.*x2_l.^2) + x2_l; % F_c
C =  num2cell(C);

F = cellfun(@(fval, c) fval + c, Fval, C, 'UniformOutput', false);




end

function plot_F_Cost = plot_F_(Total_System_Cost,d)
Za = [];
Zb = [];
alpha_b = [];
alpha_a = [];
Total_System_Cost_plot_a = [];
Total_System_Cost_plot_b = [];
A = linspace(0,1,11);
B = linspace(0,1,11);
ALPHA = linspace(0,1,11);
a = linspace(0,1,11);
b = linspace(0,1,11);

I_a = repmat(1:11, 11, 1);
J_b = repmat((1:11)', 1, 11);

Lab_fig = [];Lab_figa = [];
Total_System_Cost_plot_a = Total_System_Cost;
Total_System_Cost_plot_b = Total_System_Cost;
figure('Name', ['Total System Cost Plot FOR d = ' num2str(d)], 'NumberTitle',
'off','Position', [100, 100, 1000, 600]);
subplot(1,2,1);
[ALAL, BB] = meshgrid(ALPHA, b); % expand into 2D grids
colors = parula(11);
for b = 1:11
    Zb = vertcat(Total_System_Cost_plot_b{:,b});
    alpha_b = surf(ALAL, BB, Zb);
    hold on
    alpha_b.DisplayName = ['b = '  num2str((b-1)/10)];
    Lab_fig = [Lab_fig alpha_b];
    alpha_b.FaceColor = colors(b,:);
    alpha_b.DataTipTemplate.DataTipRows(1).Label = '\alpha';
    alpha_b.DataTipTemplate.DataTipRows(2).Label = 'a';
    alpha_b.DataTipTemplate.DataTipRows(3).Label = 'Total Cost';
end

legend(Lab_fig)
hold off
```

```
 xlabel('\alpha', 'FontSize', 12, 'FontWeight', 'bold');
 ylabel('a', 'FontSize', 12, 'FontWeight', 'bold');
 zlabel('Total System Cost', 'FontSize', 12, 'FontWeight', 'bold');
 title('Total System Cost Plot against \alpha & b', 'FontSize', 14);


 grid on;            % Show grid lines
 axis tight;         % Fit axes to data

 alpha_aplt = subplot(1,2,2);
 [ALAL, AA] = meshgrid(ALPHA, a);   % expand into 2D grids
 colors = parula(11);
 for a = 1:11
     Za = vertcat(Total_System_Cost_plot_a{a,:});
     alpha_a = surf(ALAL, AA, Za);
     hold on
     alpha_a.FaceColor = colors(a,:);
     alpha_a.DisplayName = ['a = '  num2str((a-1)/10)];
     Lab_figa = [Lab_figa alpha_a];
     alpha_a.DataTipTemplate.DataTipRows(1).Label = '\alpha';
     alpha_a.DataTipTemplate.DataTipRows(2).Label = 'b';
     alpha_a.DataTipTemplate.DataTipRows(3).Label = 'Total Cost';

 end

 legend(alpha_aplt,Lab_figa)
 hold off

 xlabel('\alpha', 'FontSize', 12, 'FontWeight', 'bold');
 ylabel('b', 'FontSize', 12, 'FontWeight', 'bold');
 zlabel('Total System Cost', 'FontSize', 12, 'FontWeight', 'bold');
 title('Total System Cost Plot against \alpha & a', 'FontSize', 14);
 grid on;            % Show grid lines
 axis tight;         % Fit axes to data


 end





 [xa_2_now, xl_2_now, F2, err2, it2] = Stackelberg_Game(2);
 [xa_10_now, xl_10_now, F10, err10, it10] = Stackelberg_Game(10);
 [xa_100_now, xl_100_now, F100, err100, it100] = Stackelberg_Game(100);


 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);

 Total_System_Cost_F2 = arrayfun(@(i,j) cell2mat(F2{i,j}), I_a, J_b, 'UniformOutput',
false);
 Total_System_Cost_F10 = arrayfun(@(i,j) cell2mat(F10{i,j}), I_a, J_b, 'UniformOutput',
false);
```

```
 Total_System_Cost_F100 = arrayfun(@(i,j) cell2mat(F100{i,j}), I_a, J_b, 'UniformOutput',
false);


 plot_F_(Total_System_Cost_F10,10);
 subplot(1,2,1)
 view([29.72 5.56])
 subplot(1,2,2)
 view([346.39 11.11])
 plot_F_(Total_System_Cost_F100,100);
 subplot(1,2,1)
 view([29.72 5.56])
 subplot(1,2,2)
 view([346.39 11.11])
```

Figure 3-1 Total System Cost when d = 10

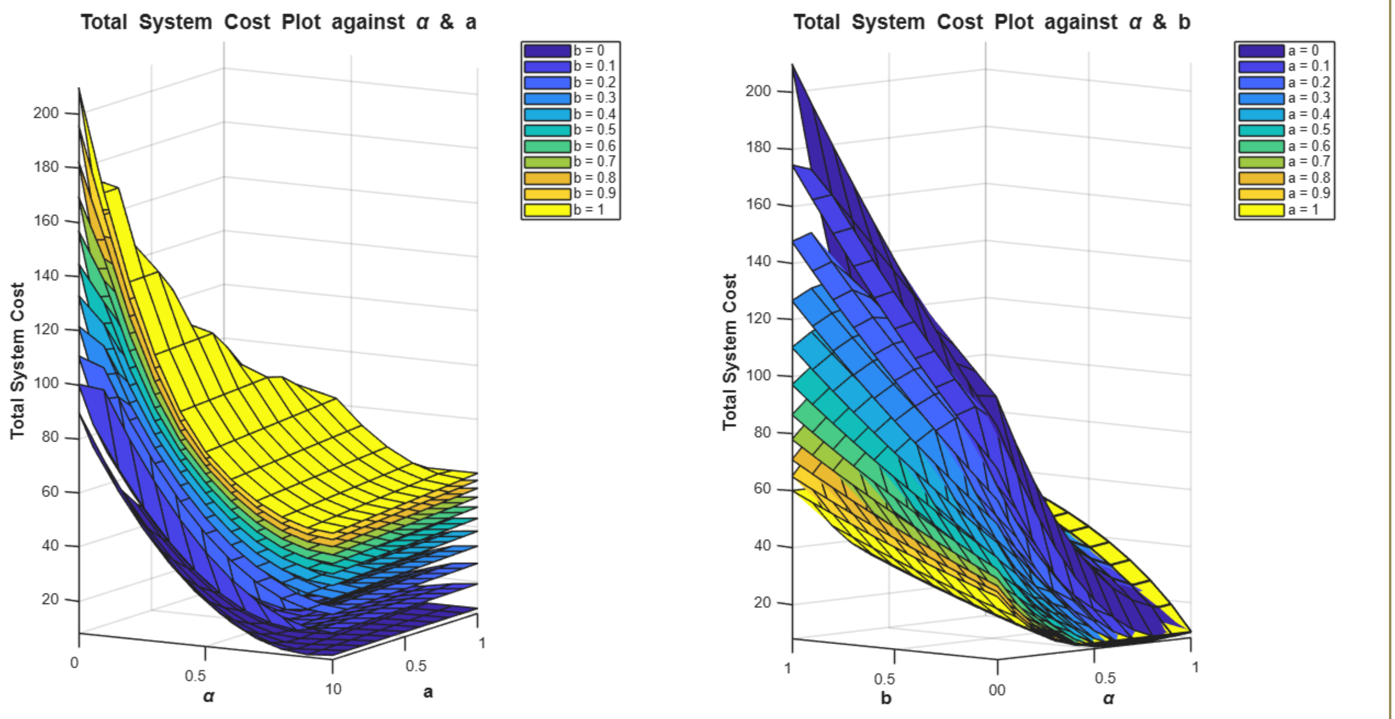
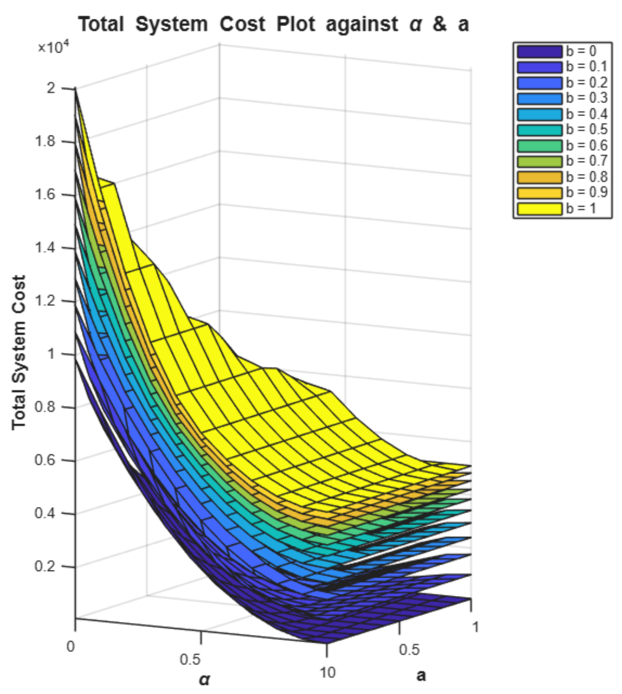


Figure 3-2: Total System Cost when d = 100



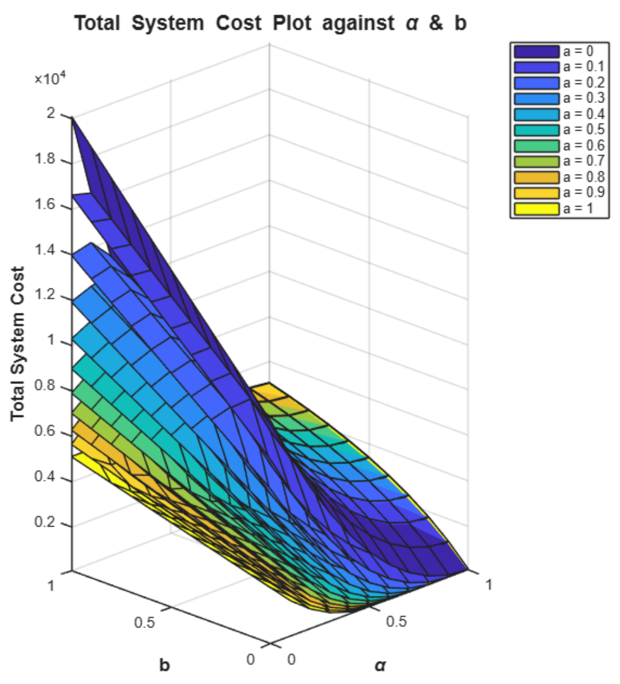


Figure 3-1 and Figure 3-2 show the plots of the Total System Cost when the plots of the two parallel link model for a demand of 100 and 10 users, respectively.

- **α – Proportion of legacy to autonomous vehicles on the system.**

  When $\alpha = 0$ (user equilibrium) As there are no autonomous vehicles, the Total System Cost is high regardless of the link used by the drivers. As α Increases the total system cost decreases due to autonomous vehicles making decisions that reduce the overall system cost. This can be seen on the graph labelled α – a graph.

- **a – free-flow cost**

  This is the cost of the link, regardless of the number of vehicles on it. The free-flow cost for link 1 is modelled to increase from 0 to 1, while the free-flow cost for link 2 is constant. As a increases, the system cost of the system seems constants for all values of b. The best scenario for this system is when there are only autonomous vehicles on both links, link 1 has a free-flow cost of 0, and there is no congestion on link 2. This can be seen on the graph labelled α –b graph.

- **b – congestiblility**
  The congestiblility for link 2 is modelled to increase from 0 to 1, while the congestiblility for link 1 is constant. This models how the link flows can substantially increase the cost of the link through traffic. The α – a graph shows the worst-case scenario on the system, where the value of b = 1. On the graph α – b graph, the total cost of the model drops as the $\alpha\ increases\ to\ 0$ for lower values of b. This shows how quickly the total cost drops as a result of congestiblility and autonomous users seeking a minimal total system cost

- **d – demand**

  The total cost increases as the demand increases as seen in both graphs for d = 10 and d = 100

  As the

B) SEPARATE LANES

I. **Demand** d

II. **Link Flow** – The distribution of the demand across the network or the number of vehicles on a link of the two parallel link system

a. x_1 – flows on a link 1 where $it\ can\ be\ x_1^{(A)}$ for autonomous vehicles is the $x_1^{(L)}$ for legacy vehicles. $x_1 \leq 0$

b. x_2 –flows on a link 2. where $it\ can\ be\ x_1^{(A)}$ for autonomous vehicles is the $x_1^{(L)}$ for legacy vehicles. $x_1 \leq 0\ x_1 \leq 0$

III. **Link Cost** – Price of using a link

a. $f_1(x_1) = a + \frac{1}{2}x_1,\ 0 \leq a \leq 1$

b. $f_2(x_2) = 1 + \frac{b}{2}x_2,\ 0 \leq b \leq 1$

IV. Constants

a. a – constants of the $f_1(x_1)$ function. Shifts the intercept of the link cost function $f_1(x_1)$ to value

b. $b - f_2(x_2)\ constant.$ Manipulates the slope of the link cost function $f_1(x_1)$

c. ∝ (alpha) - proportion of autonomous vehicles on the two parallel link model $0 \leq \alpha \leq 1.\ \alpha = 1 - is\ system\ optimal\ case\ , \alpha = 0 - user\ equilibrium$

Code Workflow : The congestiblility parameter is halved!

1. **Stackelberg_Game(d)** – starts the game by setting global parameters based on the demand d
2. **Step 0 functions**
   i) system_optimiser_step0 – broadcasts the a, b, and d constants to quad_prog_solver_step0 for $0 \leq a, b \leq 1$
   ii) quad_prog_solver_step0 – minimises the total system cost function using quadprog.

$$F = x_1\, f_1(x_1) + \ x_2\, f_2(x_2)$$
$$= \frac{x_1^2}{2} + \frac{b}{2}x_2^2 + \ ax_1 + x_2$$

3. **Step n functions**
   i) User Equilibrium
   (1) user_equilibrium_step_n:
   (2) quad_prog_solver_ue_stepn: Minimises the Beckmann transformation for legacy user
   (a) If $x_1 = \ x_L$

$$\hat{f} = \int_0^{x_1} f(x_1)\, dx_1 \ + \ \int_0^{x_2} f(x_2)\, dx_1$$
$$\hat{f} = \ ax_1 + \frac{1}{4}(x_1)^2 \ + \ (x_1 + b\frac{1}{2}\ (x_1x_2))$$

   (b) If $x_2 = \ x_L$

$$\hat{f} = \int_0^{x_1} f(x_1)\, dx_2 \ + \ \int_0^{x_2} f(x_2)\, dx_2$$
$$\hat{f} = \ ax_2 + \frac{1}{2}(x_1x_2) \ + \ (x_2 + \frac{b}{4}(x_2)^2)$$

   ii) System Optimal
   (1) system_optimiser_step_n
   (2) quad_prog_solver_so_stepn

$$F = x_1\, f_1(x_1) + \ x_2\, f_2(x_2)$$
$$= \frac{x_1^2}{2} + \frac{b}{2}x_2^2 + \ ax_1 + x_2$$

   (a) If $x_2 = \ x_L$
   Objective Function
   $F_0\ (x) = \frac{b}{2}x_2^2 + x_2$

Objective Constant

$$F_C = \frac{x_1^2}{2} + ax_1$$

(b) If $x_1 = x_L$

Objective Function

$$F_0\,(x) = \frac{x_1^2}{2} + ax_1$$

Objective Constant

$$F_C = \frac{b}{2}x_2^2 + x_2$$

4. plot_F_Cost- Plots the costs for a given demand and cost output

Table 3-2; Showing the list of objective terms, constraints and bounds using in minimising the system cost on the two link model.

Parameters for quadprog	**H**	***f***	***A***	***b***	***Aeq***	***beq***	***lb***	***ub***
System Optimal Step 0- quad_prog_solver_step0_b	$\begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$	$[a \quad 1]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$[d]$	$[0 \quad 0]$	$[(1-\alpha)d \quad \alpha d]$
User Equilibrium Step n – quad_prog_solver_ue_stepn_b **xl = x1**	$\begin{bmatrix} 1/2 & 0 \\ 0 & 0 \end{bmatrix}$	$\left[a+1+\frac{b}{2}x_2 \quad 0\right]$	$[\,]$	$[\,]$	$[1 \quad 0]$	$[(1-\alpha)d]$	$[0 \quad 0]$	$[(1-\alpha)d \quad 0]$
xl = x2	$\begin{bmatrix} 0 & 0 \\ 0 & b/2 \end{bmatrix}$	$\left[0 \quad 1+a+\frac{1}{2}x_1\right]$	$[\,]$	$[\,]$	$[0 \quad 1]$	$[(1-\alpha)d]$	$[0 \quad 0]$	$[0 \quad (1-\alpha)d]$
System Optimal Step n- quad_prog_solver_so_stepn **xl = x1**	$\begin{bmatrix} 0 & 0 \\ 0 & b/2 \end{bmatrix}$	$[0 \quad 1]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$(\alpha)d$	$[0 \quad 0]$	$[0 \quad \alpha d]$
xl = x2	$\begin{bmatrix} 1/2 & 0 \\ 0 & 0 \end{bmatrix}$	$[a \quad 0]$	$[\,]$	$[\,]$	$[1 \quad 1]$	$(\alpha)d$	$[0 \quad 0]$	$[\alpha d \quad 0]$

[] – implies empty

```
 [x_2_x1l_now,x_2_x2l_now,Fx1l_2,Fx2l_2,error, iteration] = Stackelberg_Game_B(2);
 [x_10_x1l_now,x_10_x2l_now,Fx1l_10,Fx2l_10,error, iteration] = Stackelberg_Game_B(10);

 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);

 Total_System_Cost_F2_x1l = arrayfun(@(i,j) cell2mat(Fx1l_2{i,j}), I_a, J_b,
'UniformOutput', false);
 Total_System_Cost_F2_x2l = arrayfun(@(i,j) cell2mat(Fx2l_2{i,j}), I_a, J_b,
'UniformOutput', false);


 Total_System_Cost_F10_x1l = arrayfun(@(i,j) cell2mat(Fx1l_10{i,j}), I_a, J_b,
'UniformOutput', false);
 Total_System_Cost_F10_x2l = arrayfun(@(i,j) cell2mat(Fx2l_10{i,j}), I_a, J_b,
'UniformOutput', false);

 %%

 function [x_x1l_now,x_x2l_now,Fx1l_d,Fx2l_d,error, iteration]= Stackelberg_Game_B(d)
     Max_Iteration = 120;
     Min_error = 2*11*1e-5;
     [x_d_ar_x1l,x_d_ar_x2l] = system_optimiser_step0_b(d);
     iteration = 1;
     error = Inf;
     while ((error > Min_error)  && (iteration < Max_Iteration))
     % Previous
     if iteration == 1
        [x1l_d_ar,x2l_d_ar] = user_equilibrium_step_b_n(x_d_ar_x1l,x_d_ar_x2l,d,1);
        [x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,Fx1l_d,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar,Fx2l_d]  =
system_optimiser_step_n_b(x1l_d_ar,x2l_d_ar,d,1);
        [x_x1l,x_x2l] =
add_cell_arrays(x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar);
     else
         x_x1l_prev = x_x1l;
         x_x2l_prev = x_x2l;

         [x1l_d_ar,x2l_d_ar] =
user_equilibrium_step_b_n(x_x1l_prev,x_x2l_prev,d,iteration);
         [x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,Fx1l_d,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar,Fx2l_d]
= system_optimiser_step_n_b(x1l_d_ar,x2l_d_ar,d,iteration);
         [x_x1l,x_x2l] =
add_cell_arrays(x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar);
     end
     % Present

     x_x1l_now = x_x1l;
     x_x2l_now = x_x2l;
     I_a = repelem([1:11].', 1, 11);
     J_b = [repelem([1:11].', 1, 11)].';

     if iteration == 1
```

```
         error_x1l = max(cell2mat(cellfun(@(i, j)
norm((cell2mat(x_x1l{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput',
false)));
         error_x2l = max(cell2mat(cellfun(@(i, j)
norm((cell2mat(x_x2l{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput',
false)));

     else
         error_x1l = max(cell2mat(cellfun(@(i, j) norm((cell2mat(x_x1l_prev{i,j})-
cell2mat(x_x1l_now{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput', false)));
         error_x2l = max(cell2mat(cellfun(@(i, j) norm((cell2mat(x_x2l_prev{i,j})-
cell2mat(x_x2l_now{i,j})),'fro'),num2cell(I_a), num2cell(J_b), 'UniformOutput', false)));

     end
     error = error_x1l + error_x2l;
     error = max(error);
     iteration = iteration+1;


     end

 end

 %%


 function [x_d_ar_x1l,x_d_ar_x2l] = system_optimiser_step0_b(d)
 a = linspace(0,1,11);

 b = linspace(0,1,11);
 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);

 [x_d_ar_x1l, x_d_ar_x2l] = cellfun(@(i, j) quad_prog_solver_step0_b(a(i),b(j),d), ...
                                     num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);

 end

 function [x_d_ar_x1l, x_d_ar_x2l] = quad_prog_solver_step0_b(a,b,d)
 H = [1,0
     0,b];
 f = [a; 1];
 alpha = 0.5;
 options = optimoptions('quadprog','Display','off');
 Aeq = [1, 1];
 beq = d;
 lb = [0 0];
 ub1l = [(1-alpha)*d (alpha)*d ];
 ub2l = [(alpha)*d  (1-alpha)*d ];

 x_d_ar_x1l = quadprog(H, f, [], [], Aeq, beq,lb,ub1l,[],options);
 x_d_ar_x2l = quadprog(H, f, [], [], Aeq, beq,lb,ub2l,[],options);
 end

 function [x1l_d_ar,x2l_d_ar] = user_equilibrium_step_b_n(x_d_ar_x1l,x_d_ar_x2l,d,n)
 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed
 a = linspace(0,1,11);
 b = linspace(0,1,11);
 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);
 options = optimoptions('quadprog','Display','off')
 if n == 1
     [x1l_d_ar,x2l_d_ar] = cellfun(@(i, j)
quad_prog_solver_ue_stepn_b(a(i),b(j),d,x_d_ar_x1l{i,j},x_d_ar_x2l{i,j},n), ...
                                   num2cell(I_a), num2cell(J_b), 'UniformOutput', false);

 else
     [x1l_d_ar,x2l_d_ar] = cellfun(@(i, j)
quad_prog_solver_ue_stepn_b(a(i),b(j),d,cell2mat(x_d_ar_x1l{i,j}),cell2mat(x_d_ar_x2l{i,j}
),n), ...
                                    num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);

 end
 end


 function [x1l_ar_aplha,x2l_ar_aplha] =
quad_prog_solver_ue_stepn_b(a,b,d,x12_x1l,x12_x2l,n)
 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed

 warning('off', 'all')
 Hx1l = [1/2,0
     0,0];
 Hx2l = [0,0
     0,b/2];

 lb = [0 0];
 Aeqx1l = [1, 0];
 Aeqx2l = [0, 1];
 alpha = linspace(0,1,11);
 I = linspace(1,11,11);
 options = optimoptions('quadprog','Display','off');
 if n == 1
     x1_x1l = x12_x1l(1);
     x2_x1l = x12_x1l(2);

     x1_x2l = x12_x2l(1);
     x2_x2l = x12_x2l(2);
     fx1l = [a+1+((b/2)*x2_x1l ); 0];
     fx2l = [0 ; a+1+(x1_x2l/2)];
     x1l_ar_aplha = cellfun(@(i) quadprog(Hx1l, fx1l, [], [], Aeqx1l, [(1-
alpha(i))*d],lb,[(1-alpha(i))*d 0],[],options), ...
```

```
                                     num2cell(I), 'UniformOutput', false);
     x2l_ar_aplha = cellfun(@(i) quadprog(Hx2l, fx2l, [], [], Aeqx2l, [(1-
alpha(i))*d],lb,[0 (1-alpha(i))*d],[],options), ...
                                     num2cell(I), 'UniformOutput', false);
 else

     x1_x1l = x12_x1l(1,:);
     x2_x1l = x12_x1l(2,:);

     x1_x2l = x12_x2l(1,:);
     x2_x2l = x12_x2l(2,:);

     %f = [a+x1_a; 1+(b*x2_a)];
     x1l_ar_aplha = cellfun(@(i) quadprog(Hx1l, [a+1+((b/2)*x2_x1l(i)); 0], [], [],
Aeqx1l, [(1-alpha(i))*d],lb,[(1-alpha(i))*d 0],[],options), ...
                                     num2cell(I), 'UniformOutput', false);
     x2l_ar_aplha = cellfun(@(i) quadprog(Hx2l, [0 ; a+1+((1/2)*x1_x2l(i))], [], [],
Aeqx2l, [(1-alpha(i))*d],lb,[0 (1-alpha(i))*d],[],options), ...
                                     num2cell(I), 'UniformOutput', false);



 end
 end


 function [x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,Fx1l_d,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar,Fx2l_d]
= system_optimiser_step_n_b(x1l_d_ar_aplha,x2l_d_ar_aplha,d,n)
 % x1_l & x2_l:- are being optimised
 % x1_a & x2_a:- are fixed
 a = linspace(0,1,11);
 b = linspace(0,1,11);
 I_a = repmat(1:11, 11, 1);
 J_b = repmat((1:11)', 1, 11);

 [x_a_ar_d_fin_x1l,Fx1l_d,x_a_ar_d_fin_x2l,Fx2l_d] = cellfun(@(i, j)
quad_prog_solver_so_stepn_b(a(i),b(j),d,cell2mat(x1l_d_ar_aplha{i,j}),cell2mat(x2l_d_ar_ap
lha{i,j}),n), ...
                                     num2cell(I_a), num2cell(J_b), 'UniformOutput',
false);

 xl_x1_fin_d_ar = x1l_d_ar_aplha;
 xl_x2_fin_d_ar = x2l_d_ar_aplha;
 end


 function [x_a_ar_aplhax1l,Fx1l,x_a_ar_aplhax2l,Fx2l] =
quad_prog_solver_so_stepn_b(a,b,d,x12_x1l,x12_x2l,n)
 % x1_l & x2_l:- are fixed
 % x1_a & x2_a:- are being optimised
 x1_x1l = x12_x1l(1,:);
 x2_x1l = x12_x1l(2,:);

 x1_x2l = x12_x2l(1,:);
 x2_x2l = x12_x2l(2,:);

 warning('off', 'all')
 alpha = linspace(0,1,11);
 I = linspace(1,11,11);

 Hx1l = [0,0
      0,b/2];

 Hx2l = [1/2,0
      0,0];

 Aeqx1l = [0, 1];
 Aeqx2l = [1, 0];

 %beq = [(alpha)*d];
 lb = [0 0];
 %ub = [(alpha)*d (alpha)*d];

  options = optimoptions('quadprog','Display','off');


 [x_a_ar_aplhax1l, Fvalx1l] = cellfun(@(i) quadprog(Hx1l,[0;1] , [], [], Aeqx1l,
[(alpha(i))*d],lb,[0 (alpha(i))*d],[],options), ...
                                    num2cell(I), 'UniformOutput', false);
 [x_a_ar_aplhax2l, Fvalx2l] = cellfun(@(i) quadprog(Hx2l, [a;0], [], [], Aeqx2l,
[(alpha(i))*d],lb,[(alpha(i))*d 0],[],options), ...
                                    num2cell(I), 'UniformOutput', false);



 Cx1l = 0.5.*x1_x1l.^2 + (a.*x1_x1l);
 Cx2l =  ((b/2).*x2_x2l.^2) + x2_x2l;

 Cx1l =  num2cell(Cx1l);
 Cx2l =  num2cell(Cx2l);
 Fx1l = cellfun(@(fval, c) fval + c, Fvalx1l, Cx1l, 'UniformOutput', false);
 Fx2l = cellfun(@(fval, c) fval + c, Fvalx2l, Cx2l, 'UniformOutput', false);

 end

 function [x_x1l,x_x2l] =
add_cell_arrays(x_a_ar_d_fin_x1l,xl_x1_fin_d_ar,x_a_ar_d_fin_x2l,xl_x2_fin_d_ar)
     x = repmat(1:11, 11, 1);
     y = repmat((1:11)', 1, 11);

   for k = 1:numel(x)
         i = x(k);
         j = y(k);
         x_x1l{i,j} = cellfun(@plus, x_a_ar_d_fin_x1l{i,j}, xl_x1_fin_d_ar{i,j},
'UniformOutput', false);
         x_x2l{i,j} = cellfun(@plus, x_a_ar_d_fin_x2l{i,j}, xl_x2_fin_d_ar{i,j},
'UniformOutput', false);
     end
```

```
end


%%
plot_F_(Total_System_Cost_F10_x1l,10)
subplot(1,2,1)
view([407.5 14.0])
subplot(1,2,2)
view([399.0 21.0])
plot_F_(Total_System_Cost_F10_x2l,10)

subplot(1,2,1)
view([407.5 14.0])
subplot(1,2,2)
view([399.0 21.0])

plot_F_(Total_System_Cost_F10_x1l,100)
subplot(1,2,1)
view([407.5 14.0])
subplot(1,2,2)
view([399.0 21.0])
plot_F_(Total_System_Cost_F10_x2l,100)
subplot(1,2,1)
view([407.5 14.0])
subplot(1,2,2)
view([399.0 21.0])

function plot_F_Cost = plot_F_(Total_System_Cost,d)
Za = [];
Zb = [];
alpha_b = [];
alpha_a = [];
Total_System_Cost_plot_a = [];
Total_System_Cost_plot_b = [];
A = linspace(0,1,11);
B = linspace(0,1,11);
ALPHA = linspace(0,1,11);
a = linspace(0,1,11);
b = linspace(0,1,11);

I_a = repmat(1:11, 11, 1);
J_b = repmat((1:11)', 1, 11);

Lab_fig = [];Lab_figa = [];
Total_System_Cost_plot_a = Total_System_Cost;
Total_System_Cost_plot_b = Total_System_Cost;
figure('Name', ['Total System Cost Plot FOR d = ' num2str(d)], 'NumberTitle',
'off','Position', [100, 100, 1000, 600]);
subplot(1,2,1);
[ALAL, BB] = meshgrid(ALPHA, b); % expand into 2D grids
colors = parula(11);
for b = 1:11
    Zb = vertcat(Total_System_Cost_plot_b{:,b});
    alpha_b = surf(ALAL, BB, Zb);
    hold on
    alpha_b.DisplayName = ['b = '  num2str((b-1)/10)];
    Lab_fig = [Lab_fig alpha_b];
    alpha_b.FaceColor = colors(b,:);
    alpha_b.DataTipTemplate.DataTipRows(1).Label = '\alpha';
    alpha_b.DataTipTemplate.DataTipRows(2).Label = 'a';
    alpha_b.DataTipTemplate.DataTipRows(3).Label = 'Total Cost';
end

legend(Lab_fig)
hold off

xlabel('\alpha', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('a', 'FontSize', 12, 'FontWeight', 'bold');
zlabel('Total System Cost', 'FontSize', 12, 'FontWeight', 'bold');
title('Total System Cost Plot against \alpha & b', 'FontSize', 14);


grid on;              % Show grid lines
axis tight;           % Fit axes to data

alpha_aplt = subplot(1,2,2);
[ALAL, AA] = meshgrid(ALPHA, a);   % expand into 2D grids
colors = parula(11);
for a = 1:11
    Za = vertcat(Total_System_Cost_plot_a{a,:});
    alpha_a = surf(ALAL, AA, Za);
    hold on
    alpha_a.FaceColor = colors(a,:);
    alpha_a.DisplayName = ['a = '  num2str((a-1)/10)];
    Lab_figa = [Lab_figa alpha_a];
    alpha_a.DataTipTemplate.DataTipRows(1).Label = '\alpha';
    alpha_a.DataTipTemplate.DataTipRows(2).Label = 'b';
    alpha_a.DataTipTemplate.DataTipRows(3).Label = 'Total Cost';

end

legend(alpha_aplt,Lab_figa)
hold off

xlabel('\alpha', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('b', 'FontSize', 12, 'FontWeight', 'bold');
zlabel('Total System Cost', 'FontSize', 12, 'FontWeight', 'bold');
title('Total System Cost Plot against \alpha & a', 'FontSize', 14);
grid on;              % Show grid lines
axis tight;           % Fit axes to data


end
```

Figure 3-3: Total System Cost of d = 10, while x1 = xl

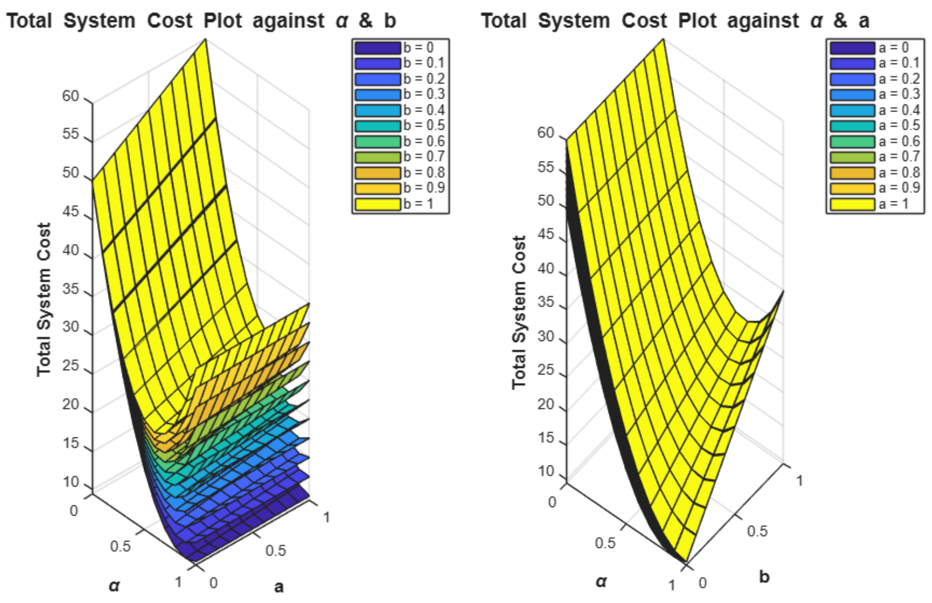


Figure 3-4 Total System Cost of d = 10, while x2 = xl

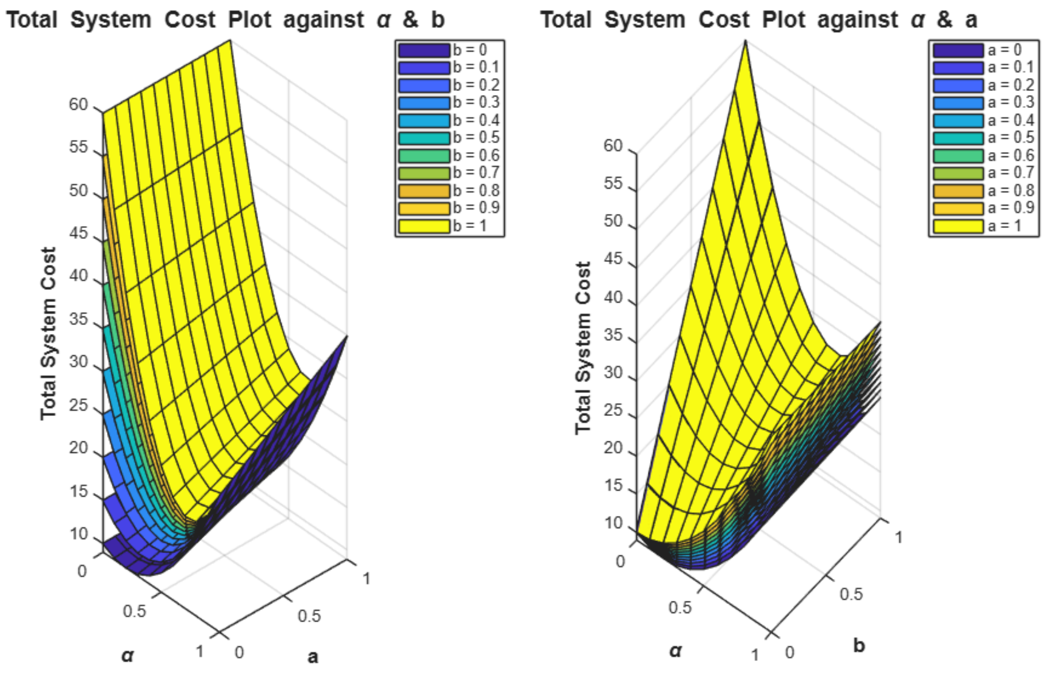
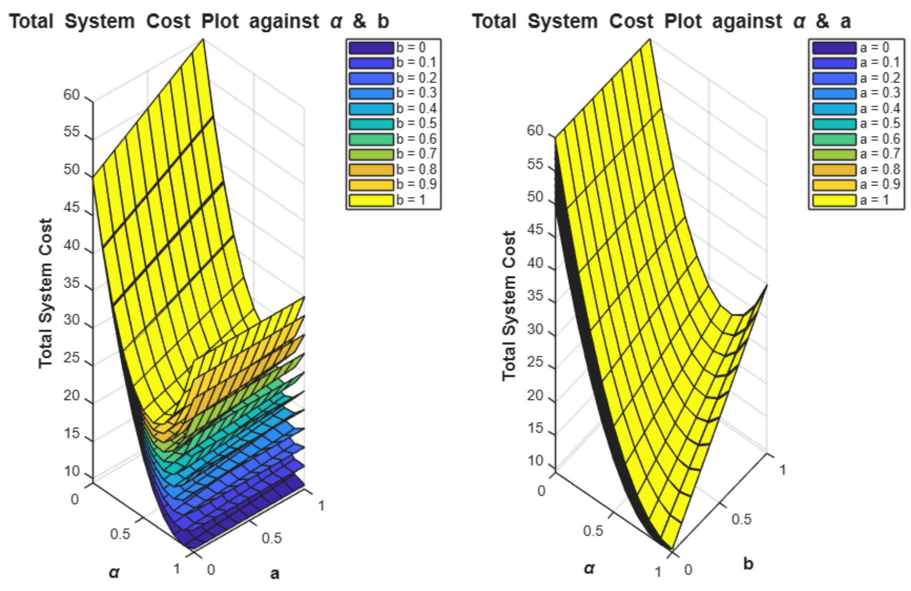
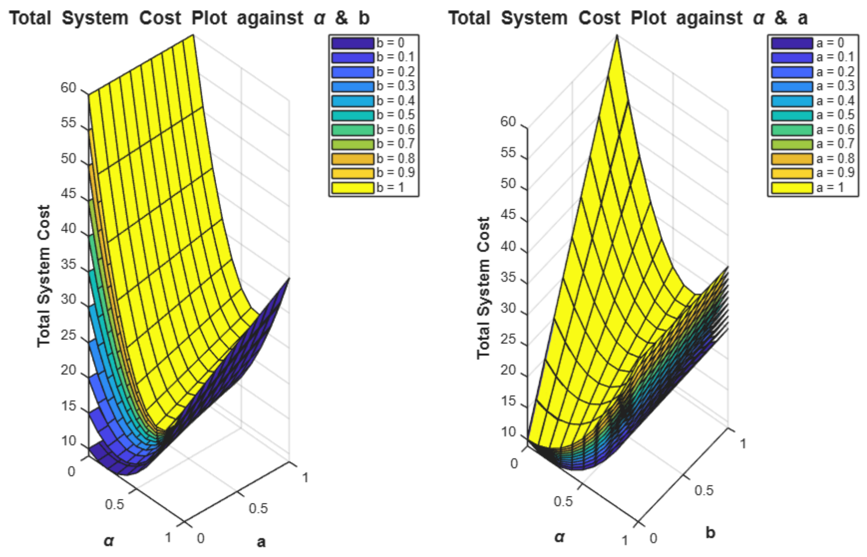


Figure 3-5 Total System Cost of d = 100, while x1 = xl



When autonomous vehicles are on link 2, they act to reduce the effect of the congestiblility value on the link. During UE scenarios i.e. no autonomous vehicles are on the link, the cost of the link is grows from 50 to 60 for d = 100 as a increases for all values of b. The link costs drop as the number of autonomous vehicles on the link increases. This indicate that the cost of this system depends on the congestiblility and system optimising vehicles on the network.

Figure 3-6 Total System Cost of d = 100, while x2 = xl



When autonomous vehicles are on link 1 the cost of the link depends on the congestiblility of link 2. The congestiblility of the road with legacy vehicles contributes to the increase in the cost of the System.

In both scenarios, strategies need to be in place to limit the effect of b. This could be breaking the traffic by adding traffic lights to link 2, discouraging passengers by encouraging car-pooling and incentivised scheme to reduce congestion on the roads and improve traffic flow.

QUESTION 4 : MICROSCOPIC MODELLING: WESTMINSTER BRIDGE DEMONSTRATION

The demonstration happening on Westminster Bridge (250m by 26m) has N participants.

The crowd flows ($F_c = \frac{Participants}{Time}$) such that $\frac{N}{3}$ Participants can fit on the bridge for a certain period of time. After 1hr (3600 s) all N participants pass through the bridge. For walking pedestrians, their speed: $v(p) = \frac{Distance}{Time} = 1.3\left(1 - \frac{p}{p_{max}}\right)^2$, where $p = \frac{Number\ of\ \ Participants\ that\ fit\ the\ area\ (N_{fit})}{Area}$ is the density and $p_{max} = 6m^{-2}$

A) ESTIMATED AVERAGE DENSITY OF THE CROWD

$$Crowd\ Flow\ Q_c = \frac{Number\ of\ \ Participants}{Time} = \frac{N_{fit}}{Area} \times \frac{Area}{Time}$$

$$\frac{N}{Time} = p\ v(p)\ width$$

$$\frac{N}{3600} = \frac{\frac{N}{3}}{250 \times 26} \times 1.3\left(1 - \frac{p}{6}\right)^2 \times 26$$

$$\overline{p} \approx 3.6m^{-2}\ , because\ \ 8.4 > 6\ (p_{max})$$

B) ESTIMATED TOTAL CROWD SIZE

$$p = \frac{N_{fit}}{Area}$$

$$3.6\ = \frac{\frac{N}{3}}{250 \times 26}$$

$$N\ = 70200\ participants$$

C) MAXIMUM CROWD FLOW UNDER TWO SCENARIOS

$$Q_c = \overline{p}\ v(\overline{p})\ width\ ,\ \ Proportion\ of\ wheelchair\ users\ are\ within\ the\ Area, \gamma = 5\%$$

To calculate the maximum crowd flow in any scenario, the maximum density for maximum crowd flow is needed.

$$\frac{dQ_c0}{dp} = 0$$

$$\frac{d\left(1.3p\left(1 - \frac{p}{6}\right)^2\right)}{dp} = 0$$

$$p =\ 2m^{-2}, p = 6m^{-2}$$

$p = 6m^{-2}\ is\ the\ maximum\ density, but\ it\ worsens\ crowd\ flow\ as\ the\ area\ is$

$completely\ filled\ , while\ p = 2m^{-2} improves\ crowd\ flow$

BLENDED CROWD: ALL PEDESTRIANS MOVE WITHIN THE SAME AREA AND AT THE SAME SPEED

Area of circular space, 0.75m radius = $\pi r^2 \approx 1.767m^2$

Recall the maximum densities under two scenarios:

- $p\ = 2m^{-2}, A_w = \frac{1}{2} = 0.5m^2$ (Maximum crowd flow density)
  $p\ = 6m^{-2}, A_w = \frac{1}{2} = 0.167m^2$ (Maximum crowd density)

With wheelchair participants, the number of participants that fit the total area will change to ensure that the area ($250 \times 26 = 6500m^2$)is only occupied.

$$6500 = N_{fit}(\gamma(A_{wc}) + (1-\gamma)(A_w)))$$

$6500 = N_{fit}\big(0.05(1.767) + 0.95(0.5)\big) \Rightarrow\ N_{fit}\ =\ 11538\text{ participants}, p = \frac{N_{fit}}{Area} = \frac{11538}{6500} = 1.775m^{-2}$. Using the velocity of the walking crowd $v(p) =\ 1.3\left(1 - \frac{1.775}{6}\right)^2 = 0.645m/s$

$$Q_c = (1.775)(0.645)(26) = 29.77\ people/s$$

For $p\ = 6m^{-2}$,

$6500 = N_{fit}\big(0.05(1.767) + 0.95(0.167)\big) \Rightarrow\ N_{fit}\ =\ 26316\text{ participants}, p = \frac{N_{fit}}{Area} = \frac{26316}{6500} = 4.05m^{-2}$. Using the velocity of the walking crowd $v(p) =\ 1.3\left(1 - \frac{4.05}{6}\right)^2 = 0.137m/s$

$$Q_c = (4.05)(0.137)(26) = 14.46\ people/s$$

Since $14.46\ people/s < 29.77\ people/s$, The maximum crowd flow $Q_c = 29.77\ people/s$

DESIGNATED AREA FOR WHEELCHAIR USERS & DIFFERENT SPEED.

For this design, we assume that no walking participants enter the wheelchair designated lane. For a single lane of wheelchair users, the density $\frac{1}{2(0.75)} = 0.67\text{ wheelchair users/m}$,

$$Q_{c_{wc}} = 0.67(0.7)2 = 0.47\ people/s$$

The maximum density of the region is $2m^{-2}\ for\ maximum\ crowd\ flow\ -$. As the walking population has decreased, so have the dimensions.

$$250(24) = N_{fit}(1-\gamma)(A_w) = N_{fit}\big(0.95(0.5)\big)$$
$$N_{fit}\ =\ 12632 \Rightarrow\ \ p = \frac{N_{fit}}{Area} = \frac{12632}{250(24)} = 2.105m^{-2}$$

$$Q_{c_w} = (2.105)\left(1.3\left(1 - \frac{2.105}{6}\right)^2\right)(24)$$
$$= 27.677 people/s$$

$$Q_c = Q_{c_w} + Q_{c_{wc}} =\ 28.14 people/s$$

Moreover, adding a wheelchair only single lane worsens he model by decreasing crowd flow from $29.77\ people/s\ to\ 28.14\ people/s$

D) VALUE OF γ WHERE IT IS NET POSITIVE TO HAVE A DESIGNATED AREA FOR WHEELCHAIR USERS

(No Attempt)

E) MODIFIED SOCIAL FORCE MODEL

On the bridge model, motion can be represented by a set of variables.

1. The positions of pedestrians $\overrightarrow{r_\alpha}$ on the bridge changes with time, t
2. Velocity at which the pedestrian is moving
3. Body orientation of the pedestrian

These parameters are needed to ensure motion is constrained to the type of motion the individual can make. Figure 4.1 describes the motion certain individuals can make on the Bridge:

Figure 4-1: Image showing Non-holonomic and Holonomic Motion Source: Rowan Hughes et al (5)

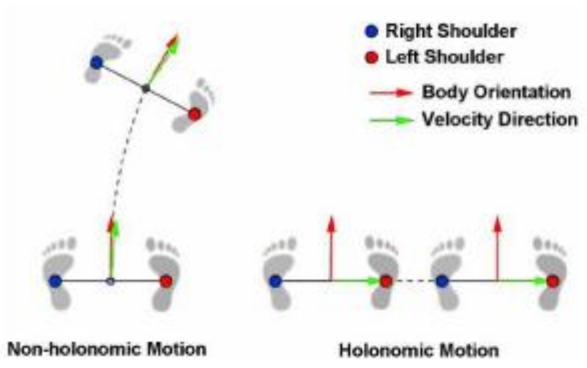


Figure 1: (Left) **Non-holonomic motion**: Direction of motion is supported by body orientation. (Right) **Holonomic motion**: Direction of motion and body orientation are decoupled.

1) The Wheelchair User: This user's motion can only be non-holonomic on the bridge i.e. their body orientation must always be aligned to their direction of velocity.
2) The Walking Pedestrian: This user's motion can be both non-holonomic and holonomic on the bridge.

The modified Headed Social Force Model (SFM) model used to exhibit flows on the bridge, would be taken from a paper titled "*When Helbing meets Laumond: The Headed Social Force Model*". [6]

The model represents variables with respect to their frame of reference; these are the global reference frame and the body frame. The steps involved include:

1. Evaluate the Social Forces on the Global Reference
2. Compute Human Locomotion Model (HLM) control inputs

DERIVATION OF SOCIAL FORCES ON GLOBAL REFERENCE

For a population P, P_w . are the proportion of pedestrians on a wheelchair given by γN and P_w, $walking\ pedestrains$. From the total population, the proportion of loud people are given by $P_L = \alpha_L P$ (inclusive of both walking pedestrians and wheelchair pedestrians)

The global forces could be defined as

Global Reference Variables [6]	**Wh**eelchair & Walking Pedestrian &
Position	$\mathbf{r_i} = [\boldsymbol{x_i},\ \boldsymbol{y_i}]'$
Velocity	$\mathbf{v_i} = \dot{\mathbf{r}}_\mathbf{i} = [\dot{\mathbf{x}}_\mathbf{i},\ \dot{\mathbf{y}}_\mathbf{i}]'$ $\mathbf{v^0_{wc_i}} =$ $\mathbf{v^0_{w_i}} =$
Acceleration	$\dot{\mathbf{v}}_\mathbf{i} = \dfrac{\mathbf{f_i}}{\mathrm{m_i}}$
Heading + Angular Velocity θ_i- the body orientation angle: $The\ angle\ between\ the\ projected$ direction and the head	$\mathbf{q_i} = [\theta_i,\ \boldsymbol{\omega}_\mathbf{i}]'$ θ_i

$\boldsymbol{\omega_i}$- angular acceleration	
Radius of Pedestrians	$r_{w_i} \approx 0.4m$ $r_{wc_i} = 0.75\, m$
Sum of Radii r_{ij}	$\boldsymbol{r_i + r_j}$
Relative Position Vector Between a loud person L and pedestrian i $\mathbf{r_{iL}}$ Between a pedestrian a i and pedestrian j $\mathbf{r_{ij}}$	$\mathbf{r_{iL}} = \mathbf{r_i} - \mathbf{r}_L$ $\mathbf{r_{ij}} = \mathbf{r_i} - \mathbf{r}_J$
Distance Center of Mass [7] Between a pedestrian a i and pedestrian j d_{ij} Between a pedestrian a i an obstacle b d_{ib} Between a loud person L and pedestrian i d_{iL}	$\boldsymbol{d_{ij}} = \lVert\mathbf{r_{ij}}\rVert = \lVert\mathbf{r_i} - \mathbf{r}_j\rVert$ $\boldsymbol{d_{ib}} = \lVert\mathbf{r_{ib}}\rVert = \lVert\mathbf{r_i} - \mathbf{r}_b\rVert$ $\boldsymbol{d_{iL}} = \lVert\mathbf{r_{iL}}\rVert = \lVert\mathbf{r_i} - \mathbf{r}_L\rVert$
Unit Normal Vector Between a pedestrian a i and pedestrian j $\mathbf{n}_{\mathrm{ij}}$ Between a pedestrian a i an obstacle b $\mathbf{n}_{\mathrm{ib}}$	$\mathbf{n_{ij}} = \dfrac{\mathbf{r_{ij}}}{\lVert\mathbf{r_{ij}}\rVert}$ $\mathbf{n_{ib}} = \dfrac{\mathbf{r_{ib}}}{\lVert\mathbf{r_{ib}}\rVert}$
Unit Tangential Vector $\mathbf{t}_{\mathrm{ij}}$ $\mathbf{t}_{\mathrm{ib}}$ $\mathrm{r'}_{\mathrm{ij}}$ & $\mathrm{r'}_{\mathrm{ib}}$ *are tan*gential vectors	$\mathbf{t_{ij}} = \dfrac{\mathbf{r'_{ij}}}{\lVert\mathbf{r_{ij}}'\rVert}$ $\mathbf{t_{ib}} = \dfrac{\mathbf{r'_{ib}}}{\lVert\mathbf{r_{ib}}'\rVert}$
Relative Tangential Velocity $\Delta\boldsymbol{v}_{ij}^{(t)}$	$\Delta v_{ij}^{(t)}$
Social forces [7] $\boldsymbol{f_{ij}}$ $\boldsymbol{-\, interaction\, force\, with\, other\, pedestrians}$ $\boldsymbol{f_{ib} - obstacle\, force\, with\, obstacles}$ $\tau_i - \boldsymbol{the\ response\ time} = 0.5s$. m_i = mass	$\boldsymbol{f_i} = m_i \dfrac{\boldsymbol{v_i^0 e_i - v_i}}{\tau_i} + \boldsymbol{f_{ij}} + \boldsymbol{f_{ib}}$ $\boldsymbol{v_i^0} - desired\ velocity$

Next Destination Vector $\mathbf{r}_i^{\mathbf{d}}$	$\mathbf{r}_i^{\mathbf{d}}$
Modelling Pedestrian Interaction Based on the $A_i = 2 \cdot 10^3 N,$ $B_i = 0.08\, m,$ $k1 = 1.2 \cdot 10^5\, kg\, s^{-2},$ $k2 = 2.4 \cdot 10^5\, kg\, m^{-1}\, s^{-1}$	$\boldsymbol{f_{ij}} = \left[A_i\, e^{\frac{(rij-dij)}{Bi}} + k_1 g\left(r_{ij} - d_{ij}\right)\right] \boldsymbol{n_{ij}} + k_2 g\left(r_{ij} - d_{ij}\right) \Delta v_{ij}^{(t)} \boldsymbol{t_{ij}}$ $A_i\, e^{\frac{(rij-dij)}{Bi}} \boldsymbol{n_{ij}}$ **-Repulsive Term** $k_1 g\left(r_{ij} - d_{ij}\right) \boldsymbol{n_{ij}}$ **– Compression Forces** $k_2 g\left(r_{ij} - d_{ij}\right) \Delta v_{ij}^{(t)} \boldsymbol{t_{ij}}$**– Friction Forces**
Modelling Obstacle Repulsion $A_b = 2 \cdot 10^3 N,$ $B_b = 0.08\, m$ $k1 = 1.2 \cdot 10^5\, kg\, s^{-2},$ $k2 = 2.4 \cdot 10^5\, kg\, m^{-1}\, s^{-1}$	$\boldsymbol{f_{ib}} = \left[\boldsymbol{A_b\, e^{\frac{(r_i-d_{ib})}{B_b}}} + \boldsymbol{k_1 g(r_i - d_{ib})}\right] \boldsymbol{n_{ib}} + \boldsymbol{k_2 g(r_i - d_{ib})} \Delta v_{ib}^{(t)} \boldsymbol{t_{ib}}$ $A_b\, e^{\frac{(r_i-d_{ib})}{B_b}} \boldsymbol{n_{ib}}$**-Repulsive Term** $k_1 g(r_i - d_{ib}) \boldsymbol{n_{ib}}$**– Compression Forces** $k_2 g(r_i - d_{ib}) \Delta v_{ib}^{(t)} \boldsymbol{t_{ib}}$**– Friction Forces**
Modelling Loud Pedestrian influence [7] $B_L = 2\, m, A_l = 200N$ $\boldsymbol{p_{iL}}$	**The effect of the noise or loudness is modelled to have an exponential effect that decreases with distance** **If** $d_{iL} < 5m$ $\boldsymbol{p_{iL}} = \boldsymbol{A_L}\, e^{\frac{(-d_{iL})}{B_L}} \boldsymbol{e_{il}}$ **If** $d_{iL} > 5m$ $\boldsymbol{p_{iL}} = \mathbf{0}$ **Where** $\boldsymbol{e_{il}} = \frac{\mathbf{r_i} - \mathbf{r}_L}{\lVert \mathbf{r_i} - \mathbf{r}_L \rVert}$
Desired Direction [7] e $\mathbf{r_i}(\boldsymbol{t})$ is the instantaneous position	$\boldsymbol{e_i} = \frac{\mathbf{r}_i^{\mathbf{d}} - \mathbf{r_i}(t)}{\lVert \mathbf{r}_i^{\mathbf{d}} - \mathbf{r_i}(t) \rVert}$ To add the influence of the on the velocity and the
Driving Force- It is the only force dependent on the agent's velocity It would need the following components. - **Typical Driving Force** - **Loud Pedestrian Driving Force**	$\boldsymbol{f_i^d} = \boldsymbol{m_i} \frac{\boldsymbol{v_i^0 e_i - v_i}}{\boldsymbol{\tau_i}}$

Computation of Body Orientation Reference

Body Reference Frame Variables [6]		
	Walking Pedestrian	**Wheelchair Pedestrian**
Body Frame velocity [6]	$\boldsymbol{v_i^B} = [\boldsymbol{v_i^f}, \boldsymbol{v_i^l}]'$	$\boldsymbol{v_i^B} = [\boldsymbol{v_i^f}, \mathbf{0}]'$

$\boldsymbol{v}_i^f$, -forward velocity $\boldsymbol{v}_i^l$ – lateral velocity		This is to ensure non-holonomic motion on the wheelchair users.
Body Frame acceleration [9] $\boldsymbol{I}_i = \frac{\boldsymbol{1}}{\boldsymbol{2}} m_i r_i^2$ $-is\ the\ inertia\ of\ the\ pedestrian\ i$	$\dot{\boldsymbol{v}}_i^B = \frac{\boldsymbol{1}}{m_i} \boldsymbol{u}_i^B$ $\boldsymbol{q}_i = A\, \boldsymbol{q}_i + \boldsymbol{b}_i u_i^{\theta_i}$ $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ $\boldsymbol{b}_i = \overset{0}{[\frac{1}{I_i}]}$	
Rotation Matrix (R) Converts Body Frame to Global Reference Frame	$\begin{matrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{matrix}$	
Conversion Principle	$\boldsymbol{v}_i = \boldsymbol{R}\boldsymbol{v}_i^B$ $\boldsymbol{v}_i = \dot{\boldsymbol{r}}_\iota = \boldsymbol{R}\dot{\boldsymbol{v}}_\iota$	
Human Locomotion Model (HLM) control inputs $\boldsymbol{u}_i^B - [\boldsymbol{u}_i^f, \boldsymbol{u}_i^l]'$ $force\ acting\ in\ the\ same\ direction$ $and\ orthogonal\ to\ heading$ **For Wheelchair users $\boldsymbol{u}_i^l = \boldsymbol{0}$** k^d – damping weight: instead of setting the lateral velocity for wheelchair users to zero the value of the damping weight can be increased significantly to reduce effect of sideways motion. k^f – body orientation direction weight $k^l - weight\ in\ the\ lateral\ direction$	$\boldsymbol{u}_i^B, u_i^{\theta_i}$ $\boldsymbol{u}_i^B$ **projects** $\boldsymbol{f}_i \boldsymbol{along\ the\ forward\ and\ sideward\ directions}$ u_i^θ is the torque about the vertical axis $\boldsymbol{u}_i^B, = \boldsymbol{K}^B\boldsymbol{R}(\boldsymbol{\theta}_i)'\boldsymbol{f}_i - \boldsymbol{k}^d v_i^o$ $\boldsymbol{K}^B = \begin{bmatrix} k^f & 0 \\ 0 & k^l \end{bmatrix}$ $\boldsymbol{k}^d = \begin{matrix} 0 \\ k^d \end{matrix}$ $u_i^{\theta_i} = -k^\theta(\theta_i - \theta_i) - k^\omega \omega_i$ **$\boldsymbol{k}^d, \boldsymbol{k}^f \boldsymbol{and}\ \boldsymbol{k}^l$ are positive constant parameters, if $\boldsymbol{k}^f = \boldsymbol{k}^l = \boldsymbol{1}, \boldsymbol{k}^d\boldsymbol{1}$, $\boldsymbol{k}^d = \boldsymbol{0}, \boldsymbol{\theta}_i$, the model is a traditional SFM model.**	
	$k^\theta = I_i k^\lambda f_i^l$ $k^\omega = I_i(1+\alpha)\sqrt{\frac{k^\lambda f_i^l}{\alpha}}\ such\ that\ k^\lambda > 0, \lambda_2 = \alpha\lambda_1 < 0\ and\ \alpha > 1$	
The control law parameters used are $\boldsymbol{k}^d = \boldsymbol{5}\ \boldsymbol{but\ for\ wheelchaire\ users}\ \boldsymbol{k}^d = \boldsymbol{600kg}, \boldsymbol{k}^f = \boldsymbol{1}, \boldsymbol{k}^0 = \boldsymbol{0.3}, \boldsymbol{\alpha} = \boldsymbol{3}\ \boldsymbol{amd}\ \boldsymbol{k}^\lambda = \boldsymbol{0.02}$		

The parameters have not been tested; these are just estimates.

QUESTION 5

A) TIME SERIES MODELLING

- Similarities
  a. Both models show a pattern of seasonality hence they were both designed or modelled with a SARIMA. This is shown by the periodicity of the plots,
  $\mathrm{SARIMA}\ (\mathrm{p}, \mathrm{d}, \mathrm{q})(\mathrm{P}, \mathrm{D}, \mathrm{Q}, \mathrm{S})\ for\ P\ \neq 0[10]$
  b. They are both smooth meaning that their diffusion component is very low, this is also likely because of the original plot being smooth as well with a low diffusion component.
- The Red Plot
  a. The red graph patterns a SARIMA (0,0,0) (2,0,0)
  b. The red graph follows that pattern of the black plot as it at peaks in, but it has bit a phase shift in time. This phase shift in time makes it appear like it predicts or fits the model based on data a time lag after hence it does not fit the data appropriately. That is the reason for the higher value of the seasonal autoregressive order P
- The Blue Plot
  a. The blue graph would have more of a SARIMA (1,0,0) (1,0,0).
  b. The blue does not have that shift in time that the red has but it does, but it seems to overshoot at peaks. This shows an added non-seasonal autoregressive order (p) to model equation. This is because the model is likely to predict the black curve based on past data further away in time.

B)

(No Attempt)

QUESTION 6

A) EGRESS FIGURES

The figures gotten are from the *Guide to safety at sports grounds* [11]

Figures [11, p. 84]	
Minimum Width of each part of the Exit OR Emergency Exit Route	New Constructions: 1.2m Existing Constructions 1.1m
Maximum Density of Reservoir Area (Standing)	40 persons per 10 square metres
Rate of Passage of People the Exit (Egress Flow Rates)	
Steeped Surface	66 spectators per metre width per minute, for width 1.2 m
Level Surface	82 spectators per metre width per minute, for width 1.2 m
Rates of Passage Seated	73 spectators per metre width per minute, on all staircases and routes within
Maximum Travel Distance	From seating < 30 m From standing <12m
Egress Time	$t_{max} = 8\, min$
Emergency Evacuation Time	$2 \text{ min } 30\ seconds \leq\ t_{max} \leq\ 8\ min$

- Steeped surfaces are usually at an angle θ to the level ground. An increase in θ, vertically lifts the surface from the ground.
- Based on Pythagoras theorem, the distance travelled for a person on a steeped surface is longer than a levelled surface.
- For the safe passage of traffic on s steeped surface whether going up or going down the velocity would be less than

EXIT LOCATION, DESIGN AND NUMBER CLOWNTOWN CITY

- **Egress Time**:
    - Currently, the Clowntown multi-purpose hall has a capacity of 2000 spectators, 4 exits, each with a 3 m width.
    - If as the current exits are levelled surfaces with an egress flow rate of 82 spectators per minute. The time required to exit would be 2.03 minutes, which is less than (and still approximately equal to) the stipulated exit time for the 4000-seater.
    - The present dimensions of the exits is likely not to handle the crowd flow out of the building. They could only allow $2 \times 82 \times 12 = 1968$ people, i.e 492 people per exit.
    - To ensure about 520 per exit, considering staff and other unseated personnel, the present ground floor exits would need to be increased by 1 – 4 m.
    - Some constraints with increasing the dimensions would be the tunnels as they would be tricky to expand given the current structure. Hence, the upper tier could have exits in which the lower tier can access
    - The additional minimum width required for the 4000-seater + 200 added staff would be $\frac{4200}{2\times 66} = 31.81\ m \sim 32m$.
    - Currently, the ground floor has 12m-wide exits; the upper floor would have a minimum of 20 m-wide exits.

- **Locations of Exit:**
    - Two possible types of exits would be the Main Events Area (Zone One) and the Outside the building perimeter.

    - Any member of the audience should be able to access these exits within a maximum distance of 30m.
    - Audience and Personnel on the topmost tiers would also be considered
    - **Main Events Area.**
    - Given the area dimensions $35\,m \times 35\,m$. Its current capacity of standing guests for the standing density for
        - a low physical condition figure of (0.5) (current exits do not support the required egress time)
        - a higher safety management of (0.6),
        - it can hold about 19 people per10 square meter ~ 2327 people based on the green guide
        - Zone 1 can only accommodate the lower tier, and so, more access to outer areas should be given to the upper tier
    - **Outside the building perimeter**
        - All the additional exits should have access to this, ensuring that crowd interaction from these exits is limited as much as possible
- **Number of Exit Paths:**
    - The aim is to increase accessibility to all the exit routes from every seating and standing.
    - Having too many would be cost-intensive, but since the safety of the premises is the city's priority, the egress systems need to be robust
    - I would add 6 exits to the current layout.
    - 4 on the first floor for upper tier audience, with allowance for lower tier audience to use the exit path. These would have the larger width and a stairs - 4m width
    - 2 on the topmost floor for only upper tier participants – 2m width

- **Stairways**: These would be needed more as there would be extra tiers added to the current layout. I would avoid adding steep ramps to the design, but the stairways should not be too steep. The design of these stairs would be in a zig-zag form to act

- **Seats:** I would recommend tippable seats to allow a clearer way for passengers to pass when an emergency occurs. As the recommended seat width in the Green Guide is 460 (+ 40mm) = $64\ people\ per\ row.$ A seating row depth of 610mm

- **Key Stakeholders:** These would be the

    - Emergency services, such as the police, fire and ambulance service. I would need to involve them on suitable entry points in the case of a fire or any other emergency situation
    - Architects and Engineers: They would probably have records of the layout of the building or the expertise to understand the limits of the current structure and the best ways to scale up the building. Conversations with them on modelling parameters could aid the egress rates

C) CHOSEN MICROSCOPIC MODEL

In the Clowntown model, egress time needs to be about 2 minutes. The model should be able to model emergency scenarios. It also needs to predict the reactions people make such as the need to move towards a particular direction a safely as possible. The Behavioural Heuristic Model [8] would be my go-to choice:

1. Reason for choice
    i) It is less about the social force hence easier to model. According to the research paper, it was designed to address limitations of Newtonian Force concepts like the social force model.

ii) It addresses key important factors that fit into the Clowntown model. It models human behaviour of adjusting a choice based on visual or given information.
iii) Adjustments on parameters such as number of ingress and egress paths, their widths and the head of the stairwell or the pattern of how queues would be organised.
iv) It can model delays of individuals who are afraid or indisposed and other randomness that my likely occur, which could guide manuals distributed to staff on the safety of the premises. Although this would need to be calibrated or estimated.

2. Steps
   i) Set N = 4000 for guest and Nw = 200 for workers on sits
   ii) Model the types of Surfaces, paths and exit widths around the building
   iii) Define the socio-demographic characteristics of the audience and workers within the building based on empirical data. This could be qualitative, based on staff and audience experiences, or quantitative, using observations or ticketing data.
   iv) Use these observations to define the following parameters
      (a) Speed – limited by several factors, such as age, mobility, e.t.c.
      (b) Area
      (c) Mass
      (d) Response time
      (e) Desired Path – This could be the exit or somewhere else to illustrate panic or staff behaviour.
      (f) Groups* - For families and friends
   v) Define a position for every individual (workers and audience)
   vi) Set a position for the point of disruption
3. Define success criteria based on the following assessed parameters:
      (a) Travel Distance
      (b) Rates of Passage Seated
      (c) Egress Parameter
      (d) Points of Injury or Danger Zones
4. Define the limitation of the design and try to reduce risks them based on changing geometry or the space, positions of certain individuals' demographics and the behaviours of staff members
5. Create a guide on how to use and design the space.

APPENDICES

CODE IMPLEMENTATION OF SOUTHVILLE

```
 % Notes: The code within this section is not structured for use without downloading the
map.xml using OSM bounding Box
 file = readstruct("/MATLAB Drive/TRANSPORT (2)/cOURSEWORK/map.xml");
 % The OSM MAP 'map.xml' was generated using Bounding Box option on the website[13] with
 %  Coordinates: Minimum Longitude:-2.61827 and Latitude:51.440019999999997  and Maximum
Longitude:-2.59421  and Latitude:51.446640000000002
 % Extract node attributes of the OSM map
 id = table2array(table(file.node.idAttribute))';
 latitude = table2array(table(file.node.latAttribute))';
 longitude = table2array(table(file.node.lonAttribute))';
 % Convert lon/lat (degrees) to (easting,northing)
 norteast = longlat2os(longitude,latitude,'degrees','northeast')
 Southville_nodes = table(id, longitude, latitude, norteast(:,2), norteast(:,1));
 Southville_nodes.Properties.VariableNames =
{'id','longitude','latitude','easting','northing'};

 % Extract way attributes of the OSM map
 way_id = table2array(table(file.way.idAttribute))';
 way_tag = file.way;
 node_way_struct = way_tag(1).nd;
 node_way_k_attribute = way_tag(1).tag;
 % Compile the attributes of each way tag
 for i = 1:length(way_tag)
     nd_cell{i} = way_tag(i).nd;
     nd_way_attribute{i} = way_tag(i).tag;
 end

 % Extract Highway Data
 j = 0;
 for i = 1:length(way_tag)
   try
     if sum([nd_way_attribute{i}.kAttribute] == "highway") == 1
     j = j+1;
     nd_cell_not_build{j} = nd_cell{i};
     nd_way_attribute_not_build{j} = nd_way_attribute{i};
     end
   catch
   end
 end

 nd_highway =  cell2mat(nd_cell_not_build) ;

 tags =
["area","bicycle","bridge","bridge:movable","bus","colonnade:right","construction","constr
uction:footway","covered","crossing","crossing:markings","crossing_ref","cycleway","cyclew
ay:both","cycleway:both:lane","cycleway:left","cycleway:left:lane","cycleway:right","cycle
way:right:lane","cycleway:right:oneway","cycleway:surface","cycleway:width","designation",
"foot","footway","footway:surface","handrail","hgv","highway","horse","incline","informal"
,"lane_markings","lanes","lanes:backward","lanes:forward","lanes:psv:backward","lanes:psv:
forward","layer","lit","maxheight","maxheight:signed","maxspeed","maxweight","maxweightrat
ing","motor_vehicle","motor_vehicle:conditional","motorcycle","motorcycle:conditional","na
me","noname","not:name","note","oneway","oneway:bicycle","parking:left","parking:left:acce
ss","parking:left:bus","parking:left:coach","parking:left:orientation","parking:right","pa
rking:right:restriction","path","postal_code","psv","psv:lanes:backward","ramp","ramp:bicy
cle","ramp:wheelchair","ref","segregated","service","sidewalk","sidewalk:both","sidewalk:l
eft","sidewalk:right","smoothness","source","source:name","source:ref","source:track","ste
p_count","surface","tactile_paving","tunnel","turn:lanes","turn:lanes:backward","turn:lane
s:forward","width"];
 % Create table and screen out certain variables
 T = table('Size',[length(nd_way_attribute_not_build) numel(tags)], 'VariableTypes',
repmat("string",1,numel(tags)), 'VariableNames', tags);

 for i = 1:length(nd_way_attribute_not_build)
     v_attr = {nd_way_attribute_not_build{i}.vAttribute};
     k_attr = [nd_way_attribute_not_build{i}.kAttribute];

     T(i, find(ismember(tags,k_attr))) = num2cell([v_attr{find(ismember(k_attr,tags))}]);
 end

 Tags_array = table2array(T);
 Nodes_with_way_vars = [];

 for i = 1:length(nd_cell_not_build)
     node_list = (struct2array(nd_cell_not_build{i})');
     Tags_rep = repelem(Tags_array(i,:),length(node_list),1);
     N_T = [node_list Tags_rep];
     Nodes_with_way_vars = [Nodes_with_way_vars; N_T];
 end

 Nodes_with_way_vars_table = array2table(Nodes_with_way_vars, 'VariableNames', {'id',
tags{:}}); % Initialise Nodes with variables
 % Join nodes with way attribute dara
 Highways_Southville = outerjoin(Nodes_with_way_vars_table, Southville_nodes , 'Keys',
'id', 'Type', 'left', 'MergeKeys', true);
 % Count those nodes with Intersectiond
 intersection_per_id = pivot(Highways_Southville,Rows="id");
 intersection_per_id.Properties.VariableNames = {'id', 'intersection count'};

 Highways_Southville = outerjoin(Highways_Southville, intersection_per_id , 'Keys', 'id',
'Type', 'left', 'MergeKeys', true);
 % Clean out streets with no name
 Streets_Highways_Southville =rmmissing(Highways_Southville,'DataVariables','name') ;
 Streets_Highways_Southville = rmmissing(Streets_Highways_Southville,2,'MinNumMissing',
height(Streets_Highways_Southville));


 % Streets Data was gotten from a websearch on streets on Southville
 streets = [
     "Acramans Road","Alpha Road","Argyle Street","Ashton Gate Terrace","Ashton Road",...
     "Birch Road","British Road","Brook Road",...
     "Camden Road","Cannon Street","Catherine Mead Street","Clift Road","Coronation
Road",...
     "Dean Lane","Dean Street",...
     "Edgeware Road","Exeter Road","Fairfield Road","Frayne Road",...
     "Greville Road","Greenway Bush Lane","Greville Street","Greenbank Road","Gathorne
Road",...
     "Hamilton Road","Hebron Road","Howard Road",...
     "Leighton Road","Lime Road",...
     "Melville Terrace","Merrywood Road","Morley Road","Mount Pleasant Terrace","Myrtle
Street",...
     "Nelson Parade","North Street",...
     "Osborne Road",...
     "Park Road","Pembroke Road","Phipps Street",...
     "Raleigh Road",...
     "Southville Place","Southville Road","Stackpool Road","Summer Street",...
     "Truro Road",...
     "Upper Sydney Street","Upton Road",...
     "Vicarage Road","Victoria Place",...
     "Walter Street","Warden Road","West End"
 ];

 Streets_Highways_Southville =
Streets_Highways_Southville(ismember(Streets_Highways_Southville.name, streets), :);

 Streets_Highways_Southville = rmmissing(Streets_Highways_Southville,2,'MinNumMissing',
height(Streets_Highways_Southville));

 [highway_types,idhtype] = findgroups(Streets_Highways_Southville.highway);
 [street_names,idstname] = findgroups(Streets_Highways_Southville.name);

 variables_reets = Streets_Highways_Southville.Properties.VariableNames;

 % This code section is based off this source [14]
 Split_by_names = splitapply( @(varargin) varargin,
Streets_Highways_Southville,street_names);
 Streets_on_Southville = cell(size(Split_by_names , 1));

 for i = 1:size(Split_by_names , 1)
 Streets_on_Southville{i} = table(Split_by_names{i, :}, 'VariableNames', variables_reets);
 end
 % Plot geoplot
 figure
 for i = 1:size(Streets_on_Southville,1)

geoscatter(Streets_on_Southville{i}.latitude,Streets_on_Southville{i}.longitude,'blue',"fi
lled",SizeData=8)
    hold on
 end

 title('Southville SubZone with Highways');
 legend(["Highways)"])
 hold off
 geobasemap streets
```

```
geolimits([51.4402 51.446],[-2.619 -2.590])
```

```
% Initialise variables for graphing Southville


G_Southville = graph();
nodeIDs = [];
nodeE = [];
nodeN = [];
edgeLabels = {};

function index = Scatter_Plot_Points(E, N)
% Scatter_Plot_Points - This orders the set of Longitude and Latitude
% Points to resemble a scatter plot
%
% Notes: The geoscatter seemed to map out the point to their locations on
% the map but the geoplot function failed teribly. On assessing the scatter
% plot, I wanted a way to order the easting and nothing values E and N to
% resmble a scatter plot as much as possible. Some sources stated some
% information on scatter matrix which is why I impelemented the
% eigendecomoposition of the scatter or correlation matrix to get the both the direction
and magnitude"
% "1st Principal Component (PC1): The direction of maximum variance (most
% spread)" - I would need to perform the eigen-decomposition of the
% covarriance matrix and get the value of the 1st Principal Component
%
    coord = [E(:), N(:)];
    % Some nodes are duplicated in for some streeta
    [coord, ia] = unique(coord, 'rows', 'stable');
    [V,D] = eig(cov(coord)); % Eigen Decomposition of the Covariance Matrix
    [~, idx] = max(diag(D));% Find the max variance
    PC1 = V(:, idx);% Find its eigenvector
    projected  = (coord - mean(coord,1)) * PC1; %
    [~, order] = sort(projected, 'ascend');
    index = ia(order);
end
% Loop over each street table in Streets_on_Southville to build graph data
for i = 1:size(Streets_on_Southville,1)
    E = Streets_on_Southville{i}.easting; % Get Easting data to run the scatter plot
point sequence generator
    N = Streets_on_Southville{i}.northing; % Get Easting data to run the scatter plot
point sequence generator
    index = Scatter_Plot_Points(E, N); % Get sequence order of nodes

    Streets_on_focus = Streets_on_Southville{i}(index, :);
    [~, index] = unique(Streets_on_focus.id, 'stable'); % ensure that duplicate points
are removed
```

```
    Streets_on_focus = Streets_on_focus(index,:);
    deast = diff(Streets_on_focus.easting);
    dnort = diff(Streets_on_focus.northing);
    % Calculate the distance between points using the easting and northing
    % points
    DWeights = hypot(deast, dnort) / 1000;
    s = Streets_on_focus.id(1:end-1);
    t = Streets_on_focus.id(2:end);
    % Add edges between consecutive nodes and add distances as weights
    G_Southville = addedge(G_Southville, s, t, DWeights);
    edgeLabels = [edgeLabels; cellstr(Streets_on_focus.name(1:end-1))];
    % Accumulate node id and coordinates for plotting
    nodeIDs = [nodeIDs; Streets_on_focus.id];
    nodeE   = [nodeE; Streets_on_focus.easting];
    nodeN   = [nodeN; Streets_on_focus.northing];
end
% Plot the Graph
[uniqueIDs, ia] = unique(nodeIDs, 'stable');
E = nodeE(ia);
N = nodeN(ia);

Southville_Graph = plot(G_Southville, 'XData', E, 'YData', N,'NodeLabel','');
Southville_Graph.EdgeFontSize = 12;

axis equal tight
box on
grid on
set(gca,'FontName','Calibri','FontSize',12,'LineWidth',0.75)
title('Southville SubZone with Highways on Network
Graph','FontSize',12,'FontWeight','bold')
xlabel('Easting'); ylabel('Northing');
```

CODE IMPLEMENTATION FOR METHOD 2

```
% Function for add lengths
function [s, t, CL,B] = add_edge_circle_oat(N)
s = zeros(1,2*N,N/2);
t = zeros(size(s));
CL = zeros(1,N/2);
B = zeros(1,N/2);
a_rad = linspace(pi/N,pi,N); a_rad((N/2)+1:end) = [];
Chord_Lengths = (170/pi)*sin(a_rad);
Beta = (1/pi)*sin(a_rad);

for i = 1:N/2
    s(1,1:2*N,i) = repmat((1:N),1,2);
    t(1,1:N,i) = circshift(s(1,1:N,i),i);
```

```
     t(1,N+1:2*N,i) = circshift(s(1,N+1:2*N,i),-i);
     CL(1,i)= Chord_Lengths(i);
     B(1,i) = Beta(i);
 end
 end

 [as10, at10 ,CL10 ,B10]= add_edge_circle_oat(10)


 function G_add_edge = graph_add_edge_circle_oat(N)
 [s,t,d] = Graph_Circle_(N);
 G = graph(s,t,d);
 [s, t, CL,B] = add_edge_circle_oat(N);
 G_add_edge = cell(1,8,N/2);


 G_add_edge{1,1,1} = G;
 [G_add_edge{1,2,1},G_add_edge{1,3,1}, G_add_edge{1,4,1}] = Edge_Freq_Shortestpath(G, t,
s, N); % Calculate its link usage, distances and the sp array
 G_add_edge{1,5,1} = 0;
 G_add_edge{1,6,1} = sum(sum(G_add_edge{1,4,1}))/(N*(N-1)); % Sum of all shortest path
lengths or distance/ number of distinct pair of nodes.
 inverse_distance = 1./G_add_edge{1,4,1};

 n = size(inverse_distance,1);
 inverse_distance(1:n+1:end) = 0;
 G_add_edge{1,7,1} = sum(inverse_distance,'all')/(N*(N-1));
 G_add_edge{1,8,1} =
mean(centrality(G_add_edge{1,1,1},'closeness','Cost',G_add_edge{1,1,1}.Edges.Weight));




 for i = 2:(1+N/2)
     j = i - 1;
     if i == (1+N/2)
         G_add_edge{1,1,i} = addedge(G,s(:,1:(N/2),j),t(:,1:N/2,j),CL(j));
         [G_add_edge{1,2,i}, G_add_edge{1,3,i},G_add_edge{1,4,i}] =
Edge_Freq_Shortestpath(G_add_edge{1,1,i}, t, s, N);
         G_add_edge{1,5,i} = B(j);
         G_add_edge{1,6,i} = sum(sum(G_add_edge{1,4,i}))/(N*(N-1)); % Sum of all shortest
path lengths or distance/ number of distinct pair of nodes.
         inverse_distance = 1./G_add_edge{1,4,i};

         n = size(inverse_distance,1);
         inverse_distance(1:n+1:end) = 0;
         G_add_edge{1,7,i} = sum(inverse_distance,'all')/(N*(N-1));
         G_add_edge{1,8,i} =
mean(centrality(G_add_edge{1,1,i},'closeness','Cost',G_add_edge{1,1,i}.Edges.Weight));

     else
         G_add_edge{1,1,i} = addedge(G,s(:,1:N,j),t(:,1:N,j),CL(j));
         [G_add_edge{1,2,i},G_add_edge{1,3,i},G_add_edge{1,4,i}] =
Edge_Freq_Shortestpath(G_add_edge{1,1,i}, t, s, N);
         G_add_edge{1,5,i} = B(j);
                 G_add_edge{1,6,i} = sum(sum(G_add_edge{1,4,i}))/(N*(N-1)); % Sum of all
shortest path lengths or distance/ number of distinct pair of nodes.
         inverse_distance = 1./G_add_edge{1,4,i};

         n = size(inverse_distance,1);
         inverse_distance(1:n+1:end) = 0;
         G_add_edge{1,7,i} = sum(inverse_distance,'all')/(N*(N-1));
         G_add_edge{1,8,i} =
mean(centrality(G_add_edge{1,1,i},'closeness','Cost',G_add_edge{1,1,i}.Edges.Weight));
     end
 end

 end
 G_add_edge10_oat = graph_add_edge_circle_oat(N10)
 theta = linspace(0, 2*pi, N10+1); theta(end) = [];
 x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
 plot(G_add_edge10_oat{1,1,1}, 'XData', x, 'YData', y);
 plot(G_add_edge10_oat{1,1,2}, 'XData', x, 'YData', y);
 plot(G_add_edge10_oat{1,1,4}, 'XData', x, 'YData', y);
 plot(G_add_edge10_oat{1,1,6}, 'XData', x, 'YData', y);




 G_add_edge100_oat = graph_add_edge_circle_oat(N100)
 theta = linspace(0, 2*pi, N100+1); theta(end) = [];
 x = (170/(2*pi))*cos(theta); y = (170/(2*pi))*sin(theta);
 plot(G_add_edge100_oat{1,1,1}, 'XData', x, 'YData', y);
 plot(G_add_edge100_oat{1,1,11}, 'XData', x, 'YData', y);
 plot(G_add_edge100_oat{1,1,31}, 'XData', x, 'YData', y);
```

REFERENCES

[1] 'Iterative proportional fitting', *Wikipedia*. Nov. 11, 2025. Accessed: Nov. 30, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Iterative_proportional_fitting&oldid=1321668062

[2] 'IPF'. Accessed: Nov. 09, 2025. [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/24829-ipf

[3] V. Latora and M. Marchiori, 'Efficient Behavior of Small-World Networks', *Phys. Rev. Lett.*, vol. 87, no. 19, p. 198701, Oct. 2001, doi: 10.1103/PhysRevLett.87.198701.

[4] admin, 'Transit Costs Data – 2025 Update', Transit Costs Project. Accessed: Dec. 07, 2025. [Online]. Available: https://transitcosts.com/new-data/

[5] R. Hughes, J. Ondˇrej, and J. Dingliana, 'Holonomic Collision Avoidance for Virtual Crowds', 2014.

[6] F. Farina, D. Fontanelli, A. Garulli, A. Giannitrapani, and D. Prattichizzo, 'When Helbing meets Laumond: The Headed Social Force Model', in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 3548–3553. doi: 10.1109/CDC.2016.7798802.

[7] D. Helbing, I. Farkas, and T. Vicsek, 'Simulating Dynamical Features of Escape Panic', *Nature*, vol. 407, no. 6803, pp. 487–490, Sept. 2000, doi: 10.1038/35035023.

[8] D. Helbing and P. Molnar, 'Social Force Model for Pedestrian Dynamics', *Phys. Rev. E*, vol. 51, no. 5, pp. 4282–4286, May 1995, doi: 10.1103/PhysRevE.51.4282.

[9] F. Farina, D. Fontanelli, A. Garulli, A. Giannitrapani, and D. Prattichizzo, 'Walking Ahead: The Headed Social Force Model', *PLOS ONE*, vol. 12, no. 1, p. e0169734, Jan. 2017, doi: 10.1371/journal.pone.0169734.

[10] 'SARIMA (Seasonal Autoregressive Integrated Moving Average)', GeeksforGeeks. Accessed: Dec. 09, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/sarima-seasonal-autoregressive-integrated-moving-average/

[11] Stationery Office, Ed., *Guide to safety at sports grounds*, 5. ed., 1. publ. Norwich: TSO, 2008.

[12] M. Moussaïd, D. Helbing, and G. Theraulaz, 'How simple rules determine pedestrian behavior and crowd disasters', *Proc. Natl. Acad. Sci.*, vol. 108, no. 17, pp. 6884–6888, Apr. 2011, doi: 10.1073/pnas.1016507108.

[13] 'OpenStreetMap', OpenStreetMap. Accessed: Dec. 07, 2025. [Online]. Available: https://www.openstreetmap.org/

[14] 'How do you split a table into sub-tables based on entries in a spec...' Accessed: Dec. 07, 2025. [Online]. Available: https://uk.mathworks.com/matlabcentral/answers/535864-how-do-you-split-a-table-into-sub-tables-based-on-entries-in-a-specific-column