SEMTM0010 Transport and Mobility Modelling

2024/2025 Coursework

Nathan Morris

xh21734

Student Number: 2102514

Question 1 - Demand Modelling

This question concerns the modelling of morning commuter flows aggregated at MSOA level in England and Wales, as described in the provided Matlab data file TMMcoursework2024Q1.mat. In this file you will find three data structures:

• MSOAdata which provides a code and full name for each of the relevant 7,264 MSOAs, together with centroid coordinates in the form of a northing and easting in the OS grid system.

• UTLAdata which provides a code and full name for each of the 174 upper tier local authorities (UTLAs) to which the MSOAs belong. Note the MSOAdata structure provides a lookup index into the UTLA structure, so that you can determine the parent UTLA for each MSOA.

• demandData which provides counts for pairs of origin-destination pairs of MSOAs, indexed according to the MSOAdata structure.

(The field names should make their purpose clear — but if you are uncertain, post on the discussion forum.)

```
% Importing TMMcoursework2024Q1.mat file with required data structures
TMMCourseworkQ1Data = load("TMMcoursework2024Q1.mat");

% Create new variables in the base workspace from those fields.
vars = fieldnames(TMMCourseworkQ1Data);
for i = 1:length(vars)
    assignin('base', vars{i}, TMMCourseworkQ1Data.(vars{i}));
end
```

a) Describe, develop, and demonstrate code to:

(i) Build a sparse origin-destination demand matrix and compute its row sums (total origin flows, called Oi in lectures) and column sums (total destination flows, called Dj in lectures).

Steps:

1. Understand the data given and how the structures correspond to each other
2. Using MATLAB sparse matrix, create a sparse matrix for all 7264 MSOAs
3. Compute the variable Oi by completing a sum of the sparse matrix rows sum([],2)
4. Compute the variable Dj by completing a sum of the sparse matrix columns sum([],1)
5. Complete a sanity check that the sums of Oi and Dj are equal
6. Plot Oi and Dj in histograms

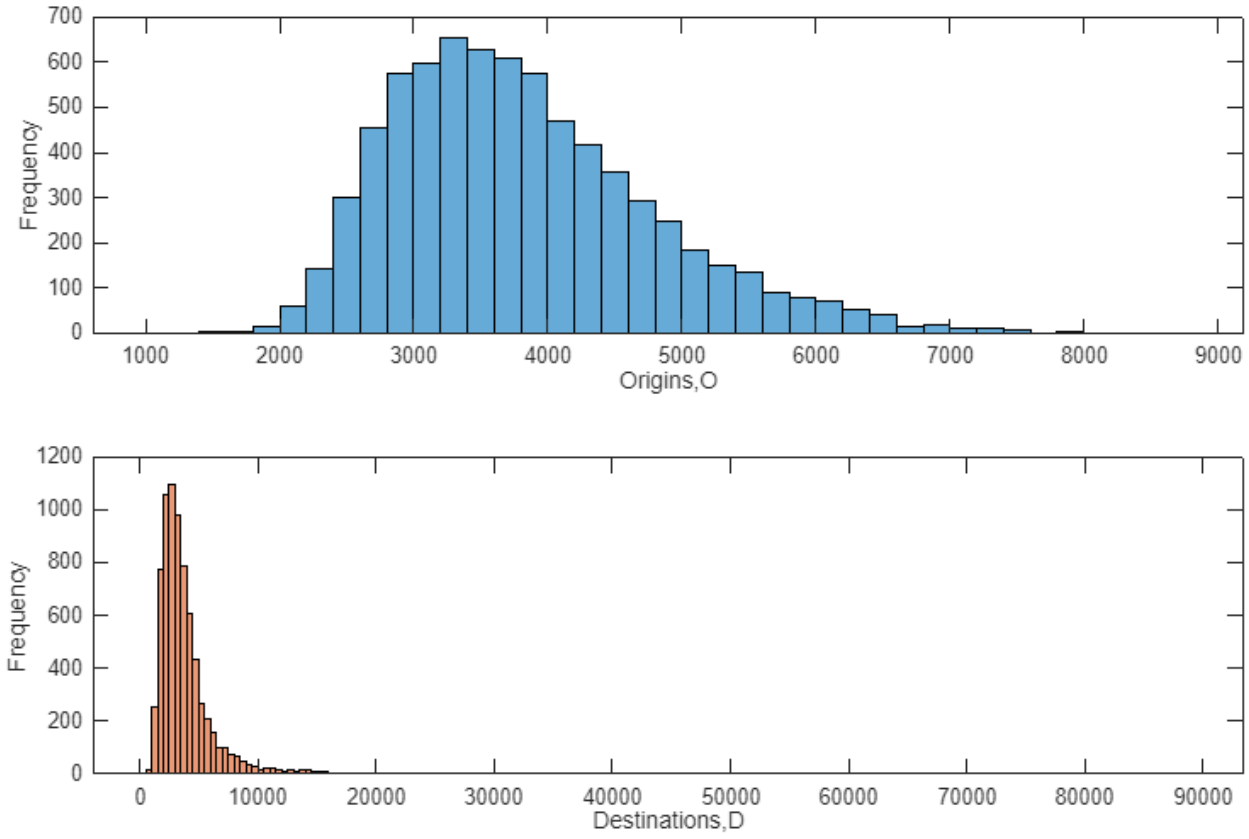
Plot the coordinates of MSOAs to understand locations (more useful for part b).

```
% Generating the sparse matrix - using demand data origin and destination
% indexes and counts
% Length of sparse matrix set as number of MSOAs
MSOASparseODMatrix =
sparse(demandData.MSOAoriginIndex,demandData.MSOAdestinationIndex ...
    ,demandData.count,length(MSOAdata.code),length(MSOAdata.code));
```

```
% Computing the row and column sums - Origin total trips from MSOAs (Oi) and
% destination total trip arrivals to MSOAs (Dj)
MSOAOi = sum(MSOASparseODMatrix,2);
MSOADj = sum(MSOASparseODMatrix,1);

% Sanity Check - checking that the number of Origin trips equals number of
% Destination arrivals
sum(MSOADj) == sum(MSOAOi) % 1==True
```

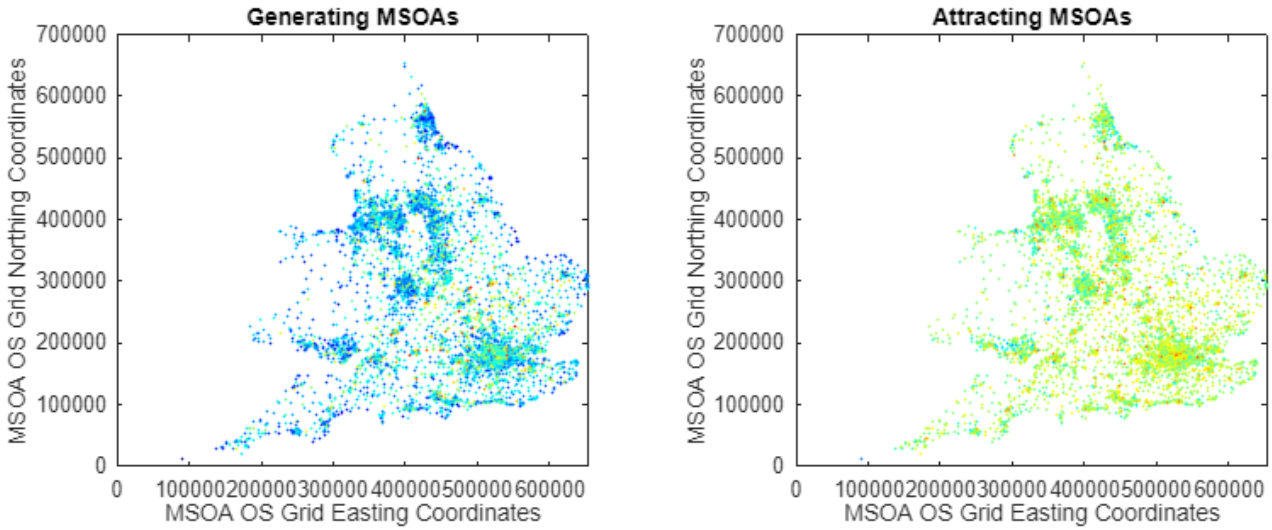
```
ans = sparse logical
   (1,1)      1
```

```
% Setting up figure
figure;

% Plotting the histograms of the Origin frequencies for all MSOA zones
subplot(2,1,1);
histogram(MSOAOi,'FaceColor',[0 0.4470 0.7410])
% Tidying up the graph to look pretty
xlabel("Origins,O")
ylabel("Frequency")

% Plotting the histograms of the Destination frequencies for all MSOA zones
subplot(2,1,2);
histogram(MSOADj,'FaceColor',[0.8500 0.3250 0.0980])
% Tidying up the graph to look pretty
ax = gca;
ax.XAxis.Exponent = 0;
xlabel("Destinations,D")
ylabel("Frequency")

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```




```
% Calculation of Colour code to see attraction vs generation
% To see which zones are mainly trip generators and which are mainly trip
% attractors
MSOAOiColourCodes = (MSOAOi - min(MSOAOi)) ./ (max(MSOAOi) - min(MSOAOi));
MSOADjColourCodes = log((MSOADj - min(MSOADj)) ./ (max(MSOADj) - min(MSOADj)));

% Setting up figure
figure;
sz = 2;
tiledlayout(1,2)

nexttile
% Plotting the scatter graph of the MSOA OS Grid coordinates colour coded
% by D/O for Origin Trips
scatter(MSOAdata.e, MSOAdata.n, sz,MSOAOiColourCodes,'filled','o');

% Tidying up the graph to look pretty
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;
xlabel("MSOA OS Grid Easting Coordinates")
ylabel("MSOA OS Grid Northing Coordinates")
```

```
title("Generating MSOAs")
box on

nexttile
% Plotting the scatter graph of the MSOA OS Grid coordinates colour coded
% by D/O for Destination Arrivals
scatter(MSOAdata.e, MSOAdata.n, sz,MSOADjColourCodes,'filled','o');

% Tidying up the graph to look pretty
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;
xlabel("MSOA OS Grid Easting Coordinates")
ylabel("MSOA OS Grid Northing Coordinates")
title("Attracting MSOAs")
box on

% Setting the colourmap and colourbar
colormap("jet");

% Resizing the figure
set(gcf,'units','inches','position',[0,0,16,6])
```



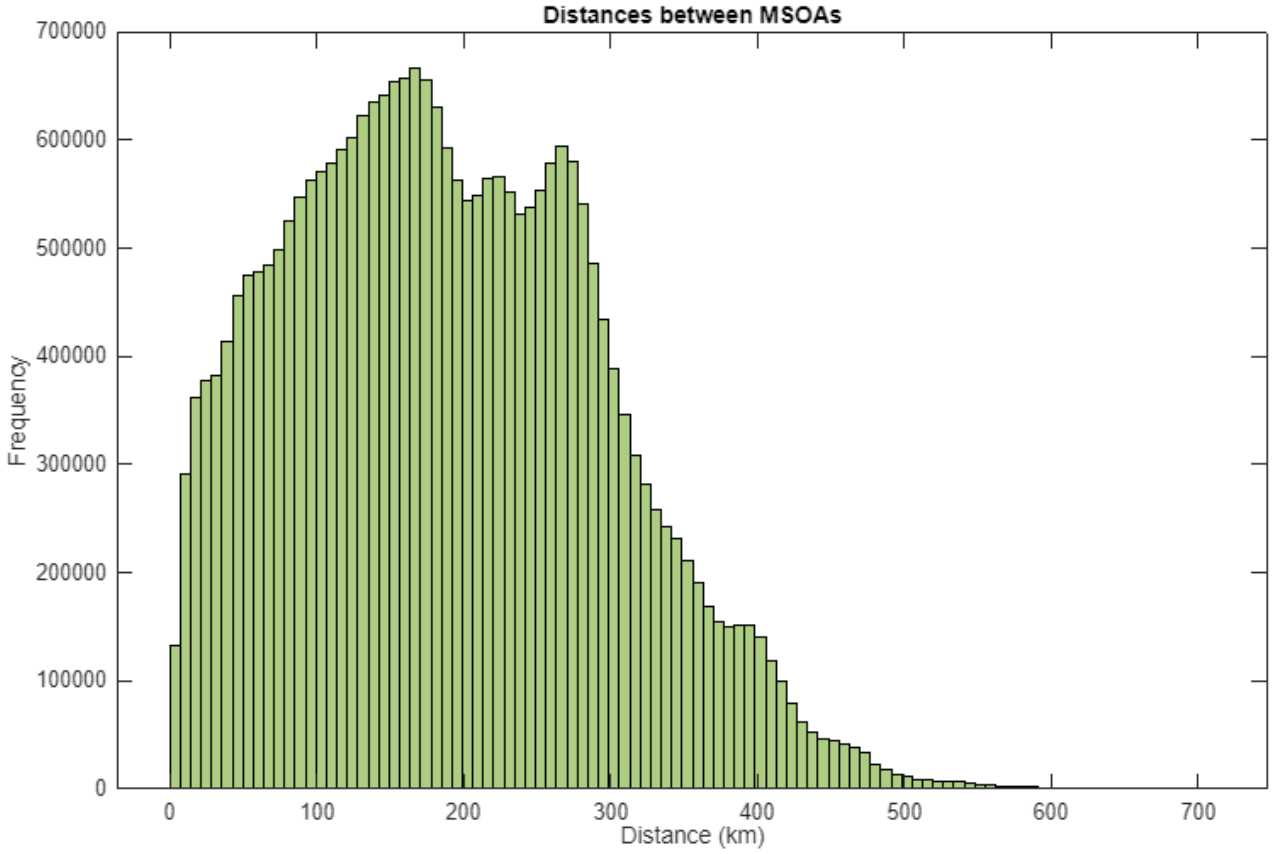
(ii) Compute a matrix of the distances between the centroids of origins and destinations, being careful to state its units

Steps:

1. Understand that MSOA locations have been given in 6 digit OS northing and easting coordinates
2. Put the coordinates into a 7264 by 2 matrix
3. Calculate the distances between each MSOA using the pdist MATLAB function and use sqaureform to convert from a 1D array to a 7264 by 7264 array

4. Divide by 1000 to convert from metres to kilometres
5. Plot the distances in a histogram as a sanity check

```
% Putting MSOA locations given in North and East 6 digit OS coodinates into
% a 7264 by 2 matrix
MSOAOSCoordinates = [MSOAdata.e MSOAdata.n];

%Calculating distances between MSOAs in km (hencce dividing by 1000) and then
putting into a square
% matrix of 7264 by 7264
MSOADistData = squareform(pdist(MSOAOSCoordinates))./1000;

% Setting up figure
figure;

% Plotting the histograms of the Origin frequencies for all MSOA zones
% Placed into 100 "bins" for ease of view
histogram(unique(MSOADistData),100,'FaceColor',[0.4660 0.6740 0.1880])

% Tidying up the graph to look pretty
xlabel("Distance (km)")
ylabel("Frequency")
title("Distances between MSOAs")
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```



(iii) Investigate best fits to the origin-destination demand data in the form cOiDj/(dij)^α , where dij is the distance data computed in (a)(ii), and c and α are constants to be fitted. You may use a combination of graphical and analytical techniques and full marks will be reserved for scripts which explain methods, discuss results and provide other insightful comment.

Steps:

1. Obtain all non-zero elements from the sparse OD matrix, making sure to record the row and column indices
2. Use these indices to obtain the corresponding distances making sure to set any 0 values to NaN
3. Setup various Tij matrices with six scenarios explored: c=1 α=1, c=2 α=1, c=3 α=1, c=1 α=2, c=1 α=3, c=1 α=4
4. Precalculate OiDj
5. Apply logs to the sparse OD matrix non-zero values to get easily visible empirical data
6. Calculate the relevant Tij values using the distances relevant distances Oi and Dj values specified by the non-zero sparse OD matrix row and column indices. Apply logs to the whole calulcation for easy visibility.
7. Calculate the correlation coefficient and Sorensen index for each Tij scenario using the empirical data calculated.
8. Plot and compare, making sure to include R and S values in the title of each plot and resize the figure appropriately

Discussion of results:

- When changing the value of c, there is very little difference due to the impact of computing the log. The same values of R and S are obtained while the plotting of the values changes very little. In short, c should have no real impact on the best fit of the gravity model compared to empirical data
- α on the other hand, has a big impact on the best fit of the gravity model to the empirical data. The values of R and S suggest that the best fit value of α sits between 2 and 3 with the plots obtained also showing this as when α=2, the values obtained by the gravity model are all positive while when α=3 some of the model values are negative. The optimal value of α is when the minimum gravity model value sits at zero. Using the plots below it is thought that the best fit value of α is approximately 2.4.

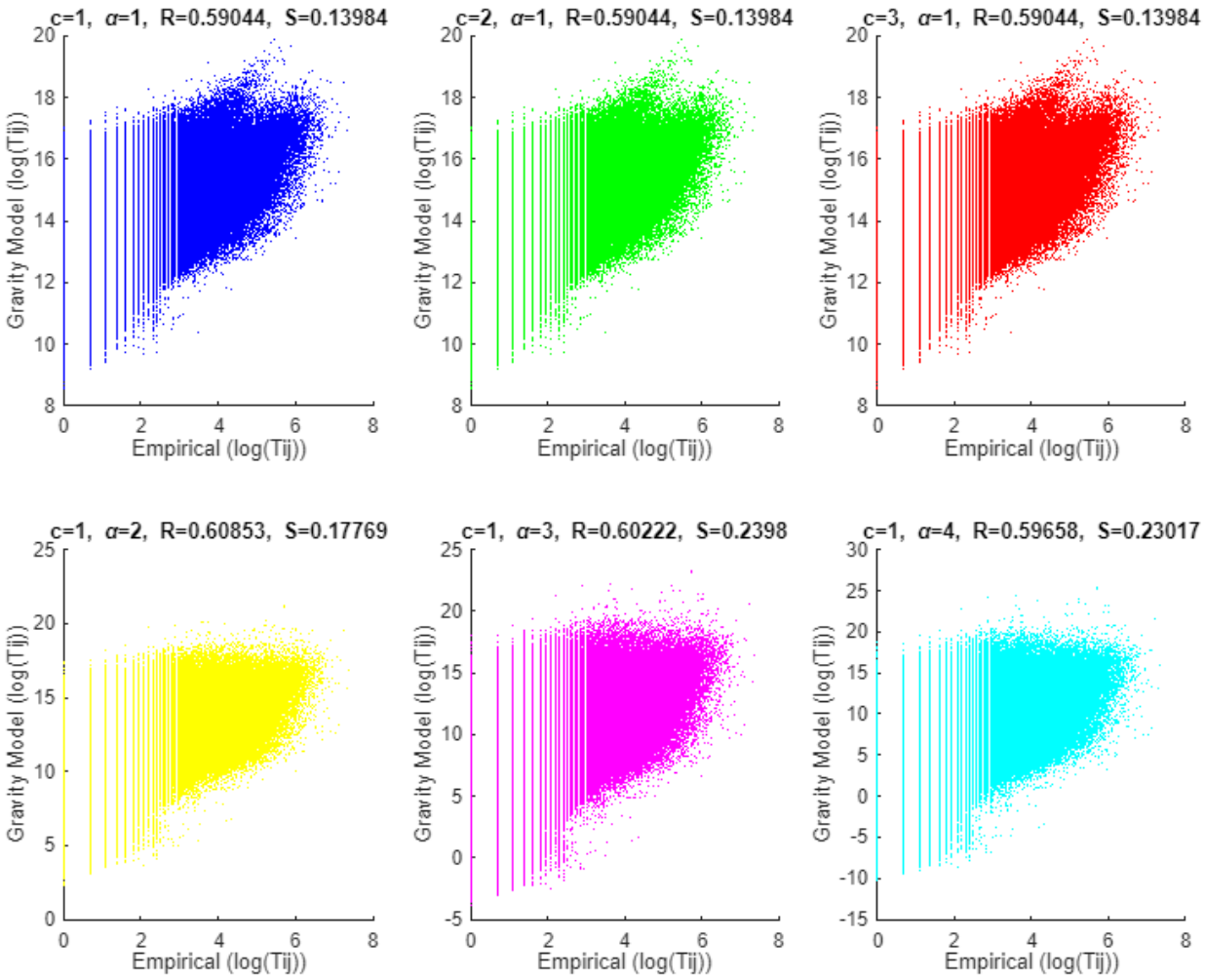
```
% Obtaining all the non-zero elements from the sparse OD matrix including
% the related row and column indices
[MSOASparseODNonZeroRows,MSOASparseODNonZeroCols,MSOASparseODNonZeroValues] =
find(MSOASparseODMatrix);

% Obtaining of the relevant distances between MSOAs form non-zero values in
% the initial sparse OD matrix
DistMatrix = MSOADistData(sub2ind(size(MSOADistData), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols));
% Making sure that no distance values are zero (important later to avoid
% infinite values)
DistMatrix(DistMatrix == 0) = NaN;

% Setting up the various Tij matrices
% c = 1, alpha = 1
OneAndOneMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));
% c = 2, alpha = 1
TwoAndOneMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));
% c = 3, alpha = 1
ThreeAndOneMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));

% c = 1, alpha = 2
OneAndTwoMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));
% c = 1, alpha = 3
OneAndThreeMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));
% c = 1, alpha = 4
OneAndFourMSOATij = zeros(length(MSOASparseODMatrix),length(MSOASparseODMatrix));

% Precalculation of OiDj for efficiency
OiTimesDj = MSOAOi .* MSOADj;

% Obtaining the empirical data
MSOAEmpirical = log(MSOASparseODNonZeroValues);

% Calculation of Tij for c = 1, alpha = 1 making sure to put value in
% corresponding position to sparse OD matrix and selecting the correct OiDj
% value - logs applied here
```

```
OneAndOneMSOATij(sub2ind(size(OneAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(OneAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (1 ./ DistMatrix));
% Calculation of Tij for c = 2, alpha = 1
TwoAndOneMSOATij(sub2ind(size(TwoAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(TwoAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (2 ./ DistMatrix));
% Calculation of Tij for c = 3, alpha = 1
ThreeAndOneMSOATij(sub2ind(size(ThreeAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(ThreeAndOneMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (3 ./ DistMatrix));

% Calculation of Tij for c = 1, alpha = 2
OneAndTwoMSOATij(sub2ind(size(OneAndTwoMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(OneAndTwoMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (1 ./ DistMatrix.^2));
% Calculation of Tij for c = 1, alpha = 3
OneAndThreeMSOATij(sub2ind(size(OneAndThreeMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(OneAndThreeMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (1 ./ DistMatrix.^3));
% Calculation of Tij for c = 1, alpha = 4
OneAndFourMSOATij(sub2ind(size(OneAndFourMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols)) = ...
    log(OiTimesDj(sub2ind(size(OneAndFourMSOATij), MSOASparseODNonZeroRows,
MSOASparseODNonZeroCols))...
    .* (1 ./ DistMatrix.^4));


% Extract non-zero values from the Tij matrices (unsure if strictly necessary but
helps to tidy up data)
[~, ~, OneAndOneMSOATijNonZeroValues] = find(OneAndOneMSOATij);
[~, ~, TwoAndOneMSOATijNonZeroValues] = find(OneAndOneMSOATij);
[~, ~, ThreeAndOneMSOATijNonZeroValues] = find(OneAndOneMSOATij);

[~, ~, OneAndTwoMSOATijNonZeroValues] = find(OneAndTwoMSOATij);
[~, ~, OneAndThreeMSOATijNonZeroValues] = find(OneAndThreeMSOATij);
[~, ~, OneAndFourMSOATijNonZeroValues] = find(OneAndFourMSOATij);

% Computing the value of correlation coefficient between the Tij values
% from the gravity model and the emipirical data
```

```
% Calculation of R for c = 1, alpha = 1
OneAndOneR =
corrcoef(OneAndOneMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');
% Calculation of R for c = 2, alpha = 1
TwoAndOneR =
corrcoef(TwoAndOneMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');
% Calculation of R for c = 3, alpha = 1
ThreeAndOneR =
corrcoef(ThreeAndOneMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');

% Calculation of R for c = 1, alpha = 2
OneAndTwoR =
corrcoef(OneAndTwoMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');
% Calculation of R for c = 1, alpha = 3
OneAndThreeR =
corrcoef(OneAndThreeMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');
% Calculation of R for c = 1, alpha = 4
OneAndFourR =
corrcoef(OneAndFourMSOATijNonZeroValues,MSOAEmpirical,'Rows','complete');

% Computing the value of Sorensen coefficient between the Tij values
% from the gravity model and the emipirical data

% Calculation of S for c = 1, alpha = 1
OneAndOneSorensen =
2*sum(min(OneAndOneMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(OneAndOneMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));
% Calculation of S for c = 2, alpha = 1
TwoAndOneSorensen =
2*sum(min(TwoAndOneMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(TwoAndOneMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));
% Calculation of S for c = 3, alpha = 1
ThreeAndOneSorensen =
2*sum(min(ThreeAndOneMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(ThreeAndOneMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));

% Calculation of S for c = 1, alpha = 2
OneAndTwoSorensen =
2*sum(min(OneAndTwoMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(OneAndTwoMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));
% Calculation of S for c = 1, alpha = 3
OneAndThreeSorensen =
2*sum(min(OneAndThreeMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(OneAndThreeMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));
% Calculation of S for c = 1, alpha = 4
OneAndFourSorensen =
2*sum(min(OneAndFourMSOATijNonZeroValues,MSOAEmpirical,'omitnan'),'omitnan')...
/(sum(OneAndFourMSOATijNonZeroValues,'omitnan') + sum(MSOAEmpirical,'omitnan'));
```

```
% Setting size of points
sz = 1;

% Setting up figure
figure;

% Plotting of Tij for c = 1, alpha = 1
subplot(2,3,1)
scatter(MSOAEmpirical,OneAndOneMSOATijNonZeroValues,sz,'filled','o','MarkerFaceColor
','blue')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=1, \alpha=1, R=" + OneAndOneR(1,2) + ", S=" + OneAndOneSorensen)

% Plotting of Tij for c = 2, alpha = 1
subplot(2,3,2)
scatter(MSOAEmpirical,TwoAndOneMSOATijNonZeroValues,sz,'filled','o','MarkerFaceColor
','green')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=2, \alpha=1, R=" + TwoAndOneR(1,2) + ", S=" + TwoAndOneSorensen)

% Plotting of Tij for c = 3, alpha = 1
subplot(2,3,3)
scatter(MSOAEmpirical,ThreeAndOneMSOATijNonZeroValues,sz,'filled','o','MarkerFaceCol
or','red')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=3, \alpha=1, R=" + ThreeAndOneR(1,2) + ", S=" + ThreeAndOneSorensen)

% Plotting of Tij for c = 1, alpha = 2
subplot(2,3,4)
scatter(MSOAEmpirical,OneAndTwoMSOATijNonZeroValues,sz,'filled','o','MarkerFaceColor
','yellow')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=1, \alpha=2, R=" + OneAndTwoR(1,2) + ", S=" + OneAndTwoSorensen)

% Plotting of Tij for c = 1, alpha = 3
subplot(2,3,5)
scatter(MSOAEmpirical,OneAndThreeMSOATijNonZeroValues,sz,'filled','o','MarkerFaceCol
or','magenta')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=1, \alpha=3, R=" + OneAndThreeR(1,2) + ", S=" + OneAndThreeSorensen)
```

```
% Plotting of Tij for c = 1, alpha = 4
subplot(2,3,6)
scatter(MSOAEmpirical,OneAndFourMSOATijNonZeroValues,sz,'filled','o','MarkerFaceColo
r','cyan')
xlabel('Empirical (log(Tij))');
ylabel('Gravity Model (log(Tij))');
% Relevant R and S value in title
title("c=1, \alpha=4, R=" + OneAndFourR(1,2) + ", S=" + OneAndFourSorensen)

% Resizing figure
set(gcf,'units','inches','position',[0,0,16,12])
```



b) Describe, develop, and demonstrate code to:

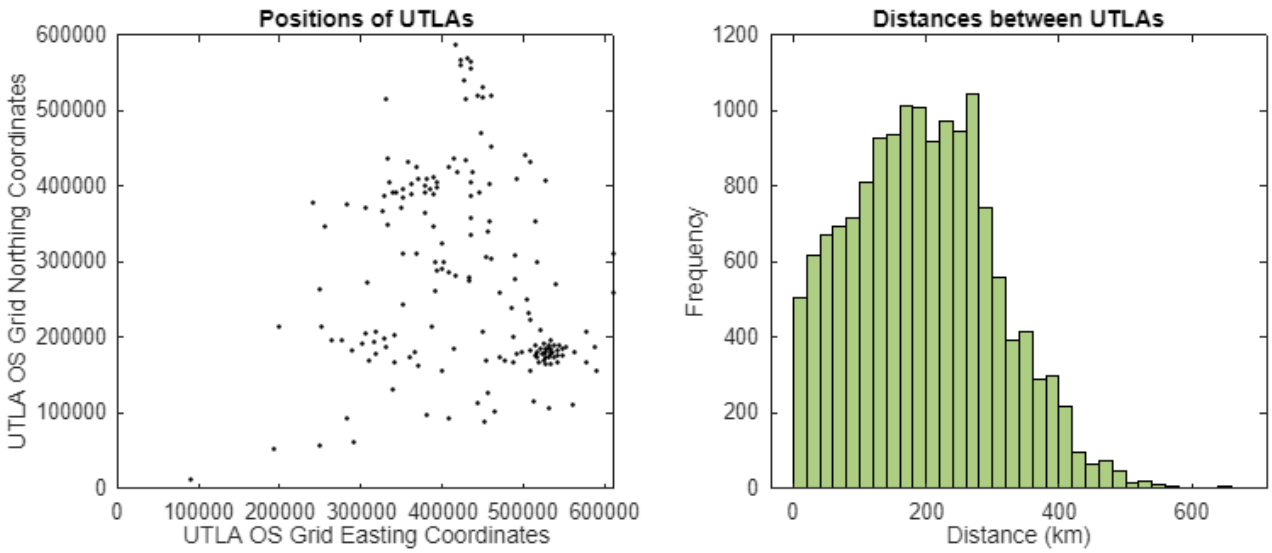
(i) Compute the centroid of each UTLA in OS grid coordinates. (You may use a simple average of the MSOA constituent members.) Compute a matrix of distances between UTLA centroids.

Steps:

1. Setup a matrix to store all "useful" UTLA values (effectively everything required to answer questions b and c).
2. Extract the UTLA codes and labels from the data structure given.
3. Use a loop to calculate the OS cooridnate centroids of UTLAs by taking the mean of the constituent MSOA members' OS coordinates. Place these in a 174 by 2 matrix for distance claculation.
4. Calculate the distances between the centroids of UTLAs using pdist and place in 174 by 174 matrix. Divide by 1000 to put in kilometres.
5. Plot the calculated UTLA centroid coordinates on a scatter plot and the distances in a histogram.
6. The UTLA centroid coordinates are also plotted in (ii) detailing the trip attracting and generating UTLAs.

```
% Extraction of UTLA codes and labels
UTLAUsefulInfo = [];
UTLAUsefulInfo(:,1) = UTLAdata.code;
UTLAUsefulInfo(:,2) = UTLAdata.label;

% Calculating centroids of UTLAs using the average of the MSOA constituent
% members
for i=1:length(UTLAUsefulInfo)
    indexes = find(MSOAdata.UTLAindex == i);
    UTLAUsefulInfo(i,3) = mean(MSOAdata.e(indexes));
    UTLAUsefulInfo(i,4) = mean(MSOAdata.n(indexes));
end

% Putting UTLA locations given in North and East 6 digit OS coodinates into
% a 174 by 2 matrix
UTLAOSCoordinates = [UTLAUsefulInfo(:,3) UTLAUsefulInfo(:,4)];

%Calculating distances between UTLAs in km and then putting into a square
% matrix of 174 by 174
UTLADistData = squareform(pdist(UTLAOSCoordinates))./1000;

% Setting up figure
figure;

subplot(1,2,1)
sz = 4;
% Plotting the scatter graph of the UTLA OS Grid coordinates
scatter(UTLAUsefulInfo(:,3), UTLAUsefulInfo(:,4),
sz,'filled','o','MarkerFaceColor','black');

% Tidying up the graph to look pretty
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;
xlabel("UTLA OS Grid Easting Coordinates")
ylabel("UTLA OS Grid Northing Coordinates")
title("Positions of UTLAs")
box on
```

```
subplot(1,2,2)

% Plotting the histograms of the distances between UTLA zones
histogram(unique(UTLADistData),'FaceColor',[0.4660 0.6740 0.1880])

% Tidying up the graph to look pretty
xlabel("Distance (km)")
ylabel("Frequency")
title("Distances between UTLAs")
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,6])
```



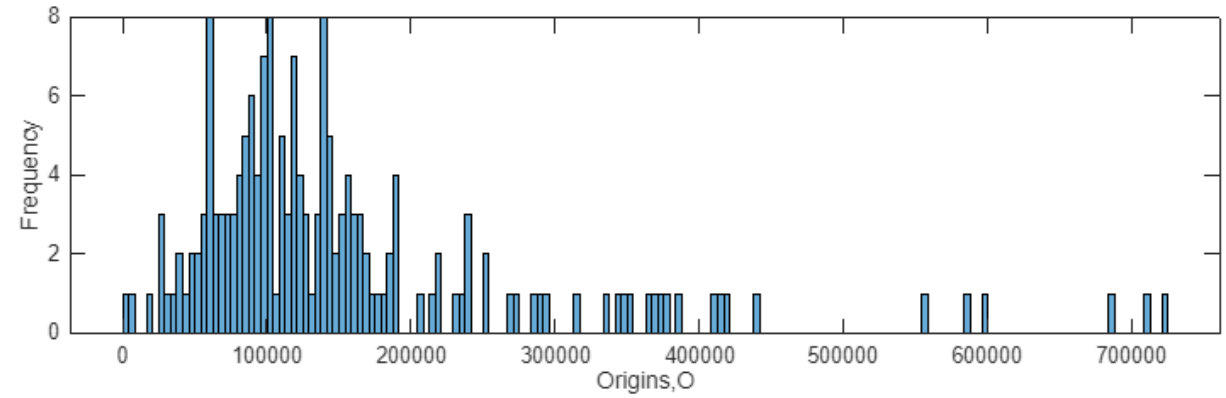
(ii) Compute the 174 × 174 origin-destination matrix that describes flows at UTLA-UTLA level, and the corresponding row sums (origin flows) and column sums (destination flows).

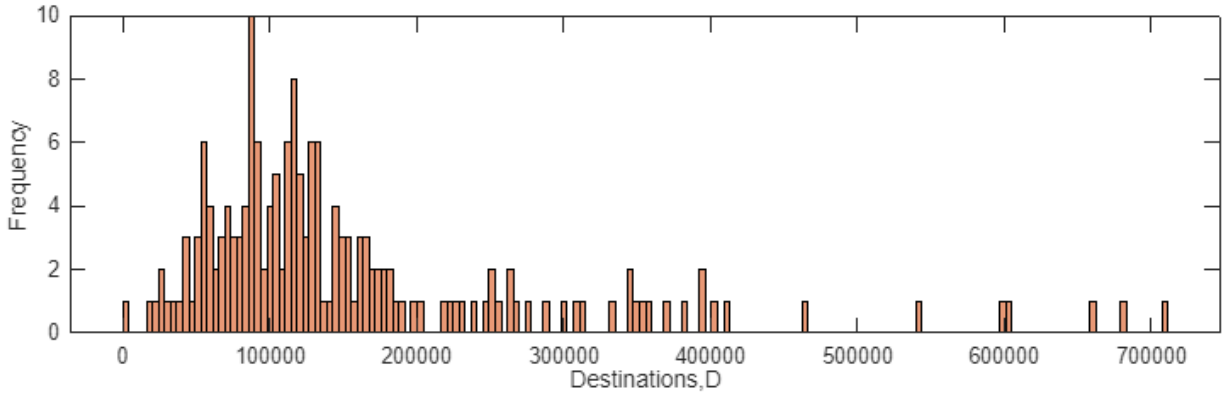
Steps:

1. Setup the UTLA OD matrix as a 174 by 174 matrix.
2. Extracting the origin and destination flows from the constituent MSOA OD matrix and summing these to obtain the total UTLA flows for each origin and destination UTLA.
3. Turn the UTLA OD matrix into a sparse matrix for efficiency.
4. Compute origin flows using row sum of UTLA OD sparse matrix sum([],2).
5. Compute destination flows using column sum of UTLA OD sparse matrix sum([],1).
6. Plot a histogram of all the non-zero origin and destination flows for a sanity check.
7. The UTLA centroid coordinates are also plotted in and colour coded detailing the trip attracting and generating UTLAs.

```
% Setting up UTLA OD Matrix with zeros based on number of individual UTLA
```

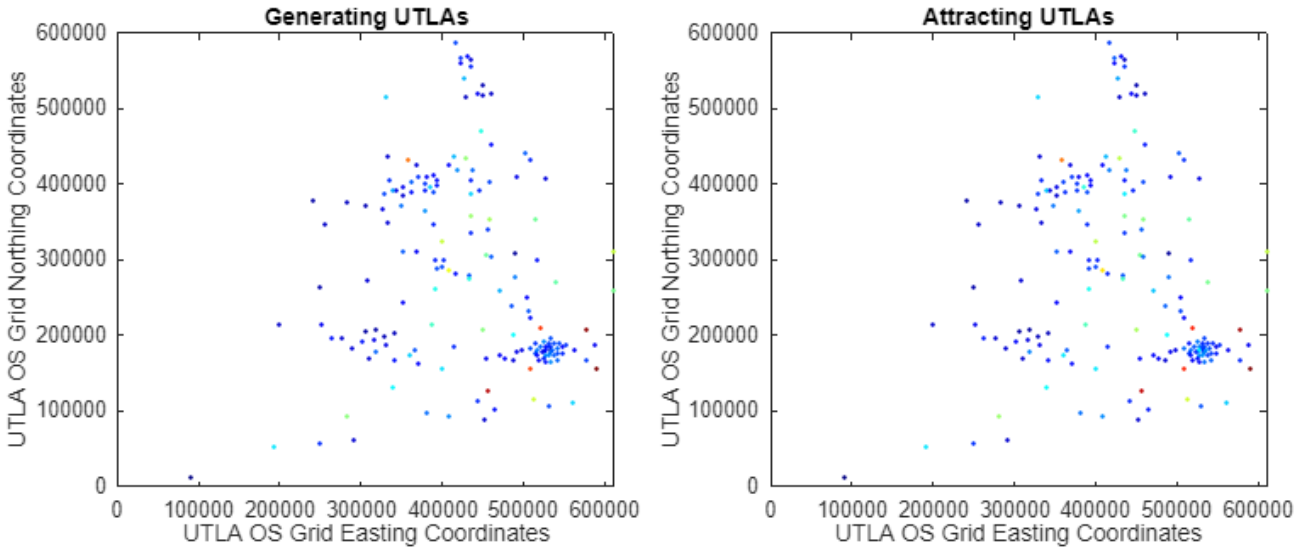
```
% codes
UTLAODMatrix = zeros(length(UTLAUsefulInfo),length(UTLAUsefulInfo));

% Extracting O and D totals using sums of constituent MSOA members
for i=1:length(UTLAUsefulInfo)
        indexesO = find(MSOAdata.UTLAindex == i);
    for j=1:length(UTLAUsefulInfo)
        indexesD = find(MSOAdata.UTLAindex == j);
        UTLAODMatrix(i,j) = sum(MSOASparseODMatrix(indexesO,indexesD),'all');
    end
end

% Turning OD UTLA matrix into a sparse matrix
UTLASparseODMatrix = sparse(UTLAODMatrix);

% Computing the row and column sums - Origin total trips from UTLAs (Oi) and
% destination total trip arrivals to UTLAs (Dj)
UTLAOi = sum(UTLASparseODMatrix,2);
UTLADj = sum(UTLASparseODMatrix,1);

% Setting up figure for hisograms of orign and destination frequencies
figure;
nBins = length(UTLAOi);
[~,~,OiVals] = find(UTLAOi);
[~,~,DjVals] = find(UTLADj);

% Plotting the histograms of the Origin frequencies for all UTLA zones
subplot(2,1,1);
histogram(OiVals,nBins,'FaceColor',[0 0.4470 0.7410])
% Tidying up the graph to look pretty
ax = gca;
ax.XAxis.Exponent = 0;
xlabel("Origins,O")
ylabel("Frequency")


% Plotting the histograms of the Destination frequencies for all UTLA zones
subplot(2,1,2);
histogram(DjVals,nBins,'FaceColor',[0.8500 0.3250 0.0980])
% Tidying up the graph to look pretty
ax = gca;
ax.XAxis.Exponent = 0;
xlabel("Destinations,D")
ylabel("Frequency")


% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```






```
% Calculation of Colour code to see attraction vs generation
% To see which zones are mainly trip generators and which are mainly trip
% attractors

UTLAOiColourCodes = (UTLAOi - min(UTLAOi)) ./ (max(UTLAOi) - min(UTLAOi));
UTLADjColourCodes = (UTLADj - min(UTLADj)) ./ (max(UTLADj) - min(UTLADj));

% Setting up figure
figure;

subplot(1,2,1)
sz = 4;
% Plotting the scatter graph of the UTLA OS Grid coordinates colour coded
% by trip generation
scatter(UTLAUsefulInfo(:,3), UTLAUsefulInfo(:,4),
sz,UTLAOiColourCodes,'filled','o');

% Setting the colourmap and colourbar
colormap("jet");

% Tidying up the graph to look pretty
ax = gca;
```

```
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;
xlabel("UTLA OS Grid Easting Coordinates")
ylabel("UTLA OS Grid Northing Coordinates")
title("Generating UTLAs")
box on

subplot(1,2,2)
sz = 4;
% Plotting the scatter graph of the UTLA OS Grid coordinates colour coded
% by trip attraction
scatter(UTLAUsefulInfo(:,3), UTLAUsefulInfo(:,4),
sz,UTLADjColourCodes,'filled','o');

% Setting the colourmap and colourbar
colormap("jet");

% Tidying up the graph to look pretty
ax = gca;
ax.YAxis.Exponent = 0;
ax.XAxis.Exponent = 0;
xlabel("UTLA OS Grid Easting Coordinates")
ylabel("UTLA OS Grid Northing Coordinates")
title("Attracting UTLAs")
box on

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,6])
```



Plots above are not quite identical - look around the london area to see some differing colours

(c) Using your results from part (b), compute, compare, and comment on the performance of the origin-constrained and destination-constrained gravity models on UTLA-UTLA demand data.

Steps:

1. First, setting all zero values of distance and flow to NaN in the distance and OD matrices.
2. Pre-calculate 1/dij^2 using the distance matrix. Non-zero values mean no inifinities obtained.
3. Calculate the initial Tij with no normalisation coefficients. Calculating the row and column sums sum([],2) and sum([],1) of this Tij matrix to obtain the origin and destination flows.
4. Calculation the normalisation values using the empirical Oi and Dj values and dividing them by those obtained by the initial Tij Oi and Dj values.
5. Computing the Origin-Constrained UTLA Tij using the Oi normalisation values.
6. Computing the Destination-Constrained UTLA Tij using the Dj normalisation values.
7. Sanity check using MSOA and UTLA empirical values making sure the total number of journeys is all equal.
8. Plotting the origin-constrained, empirical and destination-constrained flows as three heatmaps making sure to add the relevant title to each one.
9. Resizing the figure and showing the colourbar.

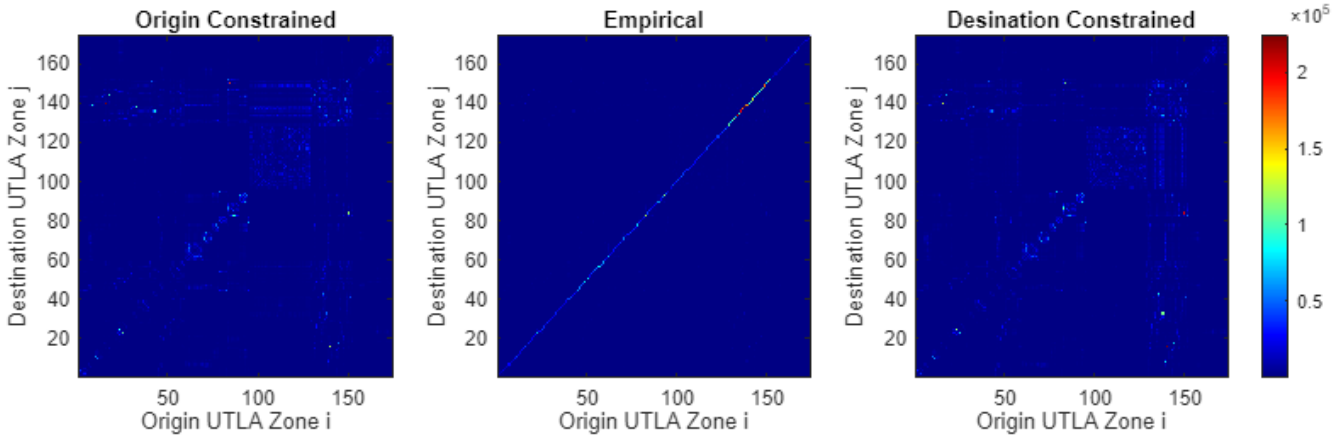
Comments on performance:

- Both the origin-constrained and destination-constrained gravity models perform similarly and poorly when compared to the empirical data.
- However, both constrained models show a similar trend compared to the empirical data of the most trips being within UTLAs and neighbouring UTLAs. This is shown by the diagonal lines seen in all the heat maps from the bottom left to top right of the plot.
- Both the orign and destination constrained models appear to show more trips in the top UTLAs with both more origins and destinations present for a few seemingly random UTLAs.
- Specifically there is a lot of journeys around UTLA zone 130. This makes sense as it is the main London UTLAs e.g. Westminster, Richmond etc. which, as the data given is about commutes, makes sense why it shows up so strongly as these are heavy commuter workplaces relative to other parts of the country. This is an indication of the (somewhat) validation of the models.

```
% Setting all zero values of distance to NaN
UTLADistData(UTLADistData == 0) = NaN;
% Pre-calulcation of 1/dij^2 for efficiency
fdij = 1./(UTLADistData.^2);

% Setting all non-zero values in the UTLA sparse OD matrix to NaN
UTLASparseODMatrix(UTLASparseODMatrix == 0) = NaN;

% Obtaining empirical data
UTLAEmpirical = UTLASparseODMatrix;

% Calculation of initial Tij
UTLATij = UTLAOi.*UTLADj.*fdij;
```

```
% Row and column sums for Oi and Dj respecitvely
UTLATijOi = sum(UTLATij,2,"omitnan");
UTLATijDj = sum(UTLATij,1,"omitnan");

% Normalisation constants for each row and column in the origin and
% destination constrained gravity model cases
OiNormalisation = UTLAOi./UTLATijOi;
DjNormalisation = UTLADj./UTLATijDj;

% Calculation using the normalisation constants for each row of the
% origin-constrained gravity model
OriginConstrainedUTLATij = (OiNormalisation.*UTLAOi).*UTLADj.*fdij;

% Calculation using the normalisation constants for each column of the
% destination-constrained gravity model
DestinationConstrainedUTLATij = UTLAOi.*(DjNormalisation.*UTLADj).*fdij;

% Sanity check to see if column and row sums match
OriginConstrainedUTLAOi =
sum(sum(OriginConstrainedUTLATij,2,"omitnan"),1,"omitnan");
DestinationConstrainedUTLADj =
sum(sum(DestinationConstrainedUTLATij,1,"omitnan"),2,"omitnan");
EmpiricalUTLAOi = sum(UTLAOi,1);
EmpiricalUTLADj = sum(UTLADj,2);
EmpiricalMSOAOi = sum(MSOAOi,1);
EmpiricalMSOADj = sum(MSOADj,2);

% Creating figure and layout
figure;
tiledlayout(1,3)

nexttile
% Heatmap of the origin-constrained gravity model
imagesc(OriginConstrainedUTLATij)
% Making y-axis go from low to high values
set(gca,'YDir','normal')
title("Origin Constrained")
xlabel("Origin UTLA Zone i")
ylabel("Destination UTLA Zone j")

nexttile
% Heatmap of the empirical data
imagesc(UTLAEmpirical)
% Making y-axis go from low to high values
set(gca,'YDir','normal')
title("Empirical")
xlabel("Origin UTLA Zone i")
ylabel("Destination UTLA Zone j")
```

```
nexttile
% Heatmap of the destination-constrained gravity model
imagesc(DestinationConstrainedUTLATij)
% Making y-axis go from low to high values
set(gca,'YDir','normal')
title("Desination Constrained")
xlabel("Origin UTLA Zone i")
ylabel("Destination UTLA Zone j")

% Setup of the colourmap and colourbar that shows the most distinct trends
% clearly
colormap("jet")
cb = colorbar;
cb.Layout.Tile = 'east';

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,5])
```



Question 2 - Network Modelling

In this question we are going to consider simple models of dense residential street networks and modifications to them that represent the implementation of low traffic neighbourhoods (LTNs).

The code generated in this section is does work for any value of n and it executes quickly producing (hopefully) what is required in relatively uncluttered plots.

(a) See lab sheet 02 Q1. Describe, develop, and demonstrate code which builds a Manhattan-style grid street network. The network's junctions should constitute an n×n grid, where n is a parameter. The 'horizontal'/'vertical' distance between neighbouring junctions is the same, and initially all streets should be two-way. Your implementation should use Matlab's graph data structure.

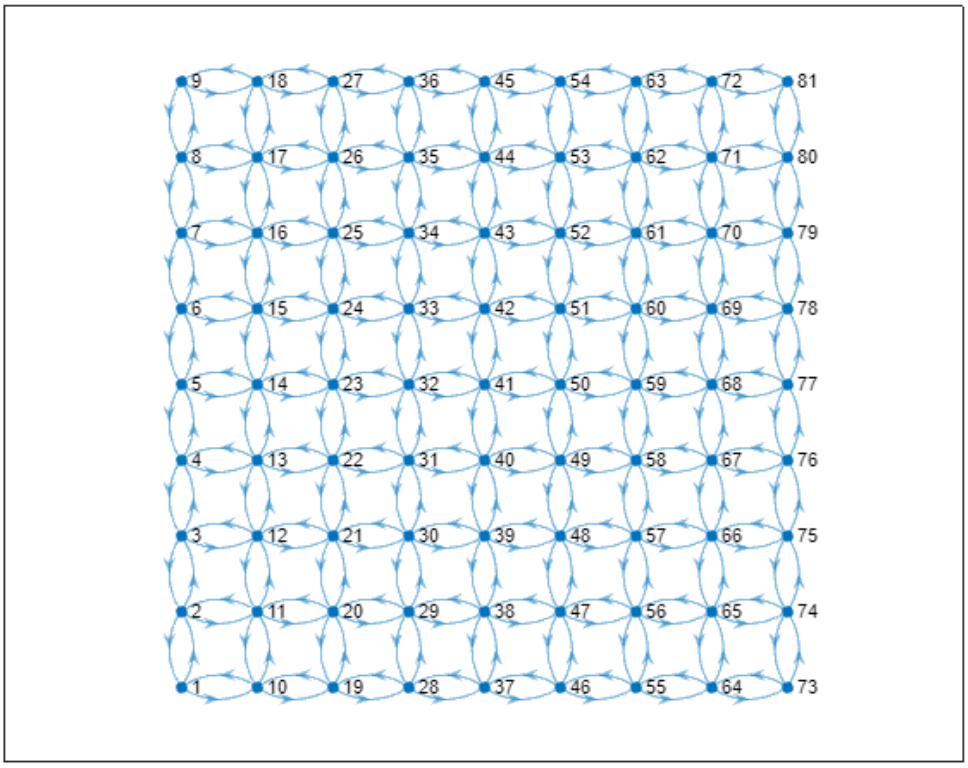
The following steps are required to build an n x n Manhattan-style grid street network where L is the vertical distance between neighbouring junctions and all streets are two way.

1. The number of nodes in each direction and the grid "pitch" in kilometres is set. These are the input parameters.
2. The figure is created.
3. The input parameters are passed to the "ManhattanSquareGrid" function which generates and plots the square grid, adjusting the coordinates of nodes before outputting the graph and plot variables.
4. The function is then called which initially uses numgrid generates a sparse grid using the finite differnce Laplacian where "s" specificies a square grid and the +2 allows for boundary padding. Then delsq turns this into a discrete Laplacian which is scaled by -L. This grid is then turned into a bi-directional graph with no self loops around each node and nodes names applied by ascending each column from left to right.
5. This graph is then plotted and finally the coordinates of each node are set for the grid.
6. Finally, the figure size is set to show clearly all nodes and node names.

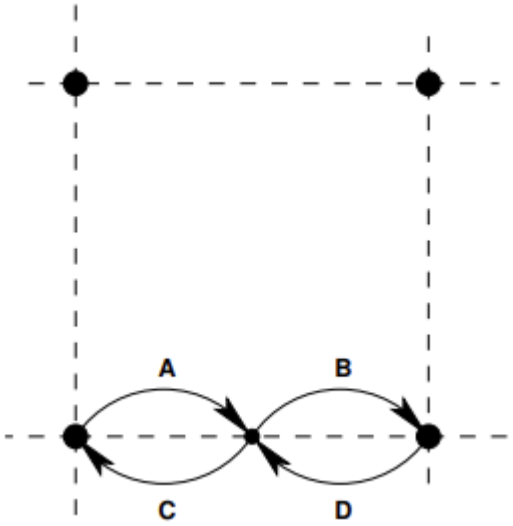
Here, the code used to create this street network is put into a function as the same code is required to be run for later questions.

```
function [OutputGraph, OutputPlot] = ManhattanSquareGrid(n,L)
    % numgrid generates the finite difference Laplacian where "s" specifies
    % a square grid and n + 2 enables a border of one boundary point or padding
around the grid
    % to create the desired number of visible nodes. It generates a sparse
    % grid.

    % delsq then computes the discrete Laplacian for the given sparse grid and the
    % multiplication of -L scales the grid by the grid "pitch"
    A = -L*delsq(numgrid('S',n+2));

    % Generating the directed graph using the matlab graphing function
    % Critically this does not create self loops around nodes and every
    % node is named from 1 to n^2 (square grid) going up the columns from
    % the bottom left and starting from the bottom row of each column
    OutputGraph = digraph(A, string(1:n^2), 'omitselfloops');
```

```
    % Plotting the directed graph
    OutputPlot = plot(OutputGraph);

    % Setting the node X and Y coordinates based upon the number of nodes
    % and grid pitch
    OutputPlot.XData = L*floor((0:n^2-1)/n);
    OutputPlot.YData = L*rem((0:n^2-1), n);
    axis equal
end

% Everything in this live script follows on from these values - these are
% the only values necessary to be changed
N = 9; % number of nodes in each direction
L = 0.1; % the grid "pitch" in kilometres

% Creating the figure
f2a = figure;

% Inputting values into grid generating function
[Q2aG, Q2ah] = ManhattanSquareGrid(N,L);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```



(b) We shall assume that the network's streets are residential, and each street is (in the morning) the origin for a mixture of trips which terminate at the four corners of the network (perhaps before continuing subsequently to further destinations). Describe, develop, and demonstrate code which builds a new network in which the midpoint of each edge is a new node which will model the origin of the demand on that street, as indicated roughly below, for just one street.



The following steps are required to build an n x n Manhattan-style grid street network where L is the vertical distance between neighbouring junctions and all streets are two way:

1. Call the previous question function to create the two-way directed grid with all required nodes
2. Create the plot and call the new function which creates midpoints and requires the input parameters and the directed graph just created
3. Get the node names and edges node names from the inital grid and generate the new node names
4. Delete all edges from the existing graph
5. Add in the new nodes to the graph making sure to name them correctly using the new node names
6. Set the coordinates for the "main" nodes
7. Calculate the unique midpoints between each "main" node next to one another
8. Setup some indexing parameters and run a for loop for all the midpoints (columns) including where new nodes are supposed to be
9. A decision of where the new node is is made where if it is in a column that has already got "main" nodes. If it does, edges are created vertically between "new" and "main" nodes and coordinates set with the same X coordinate. Otherwise, edges are created horizontally between "new" and "main" nodes and coordinates set with the same Y coordinate. All coordinates are saved in a 2D array during this process.
10. Finally plot the graph, set the X and Y coordinates of all nodes using the ones just calculated and resize the figure (making sure to plot node names too)

Here, the code used to create this street network is put into a function as the same code is required to be run for later questions.

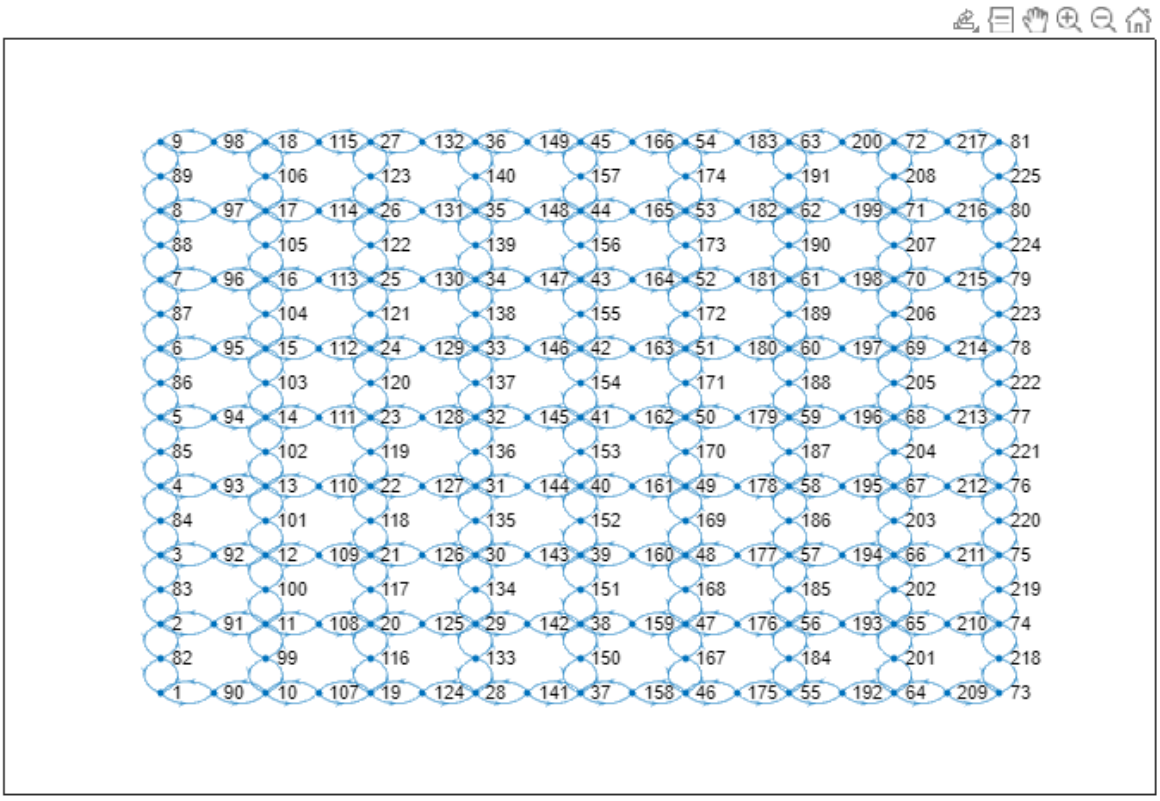
```
% Making previous figure not show again
f2b1 = figure('Visible','off');
```

```
% Obtaining Manhattan-style grid street network same as previous question
[Q2bG, ~] = ManhattanSquareGrid(N,L);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])

function [OutputGraph, OutputPlot, OutputMainNodeNames ,OutputNewNodeNames,
OutputMidpoints,...
    OutputXData, OutputYData] = ManhattanSquareGridWithMidpoints(n,L,InputGraph)

    % Initially setting the existing node names and new node names
    % Essential later on
    MainNodeNames = string(1:n^2);
    NewNodeNames = string(n^2+1:n^2+(n-1)*n*2);

    % Obtaining all the current edges the graph has
    EndNodes = InputGraph.Edges.EndNodes;

    % Deleting all edges currently in the graph
    for i=1:length(EndNodes)
        InputGraph = InputGraph.rmedge(EndNodes(i,1),EndNodes(i,2));
        InputGraph = InputGraph.rmedge(EndNodes(i,2),EndNodes(i,1));
    end

    % Adding in the new nodes to the graph
    for i=1:length(NewNodeNames)
        InputGraph = InputGraph.addnode(NewNodeNames(i));
    end

    % Setting the new coordinates for the "main" nodes in the graph taking into
    % account new nodes
    InitialXData = L*floor((0:n^2-1)/n);
    InitialYData = L*rem((0:n^2-1), n);

    % Obtaining the unique midpoints between the "main" nodes (a list of
    % 2n-1 size)
    Midpoints = unique((InitialXData(1:end-1) + InitialXData(2:end))/2);

    % Setting up array for storage of "new" node coordinates
    SubNodesCoords = zeros((n-1)*n*2,2);

    % Total index of all nodes
    index = 0;
    % Current column tracking
    Loops = 0;
    % New node index tracking
    NewNodeIndex = 0;

    % Looping over all columns of all nodes
```

```
    for i=1:length(Midpoints)

        % For the addition of edges to and from rows of "new" nodes
        % i.e. not "main" node columns
        if (rem(Midpoints(i),0.1) ~= 0)
            % Connecting "new" nodes to the "main" nodes to the left and
            % right by adding edges for each "new" node in the row
            for k = 1:n
                InputGraph =
InputGraph.addedge(MainNodeNames((Loops-1)*n+k),NewNodeNames(NewNodeIndex+k),0.1);
                InputGraph =
InputGraph.addedge(NewNodeNames(NewNodeIndex+k),MainNodeNames((Loops-1)*n+k),0.1);
                InputGraph =
InputGraph.addedge(MainNodeNames((Loops)*n+k),NewNodeNames(NewNodeIndex+k),0.1);
                InputGraph =
InputGraph.addedge(NewNodeNames(NewNodeIndex+k),MainNodeNames((Loops)*n+k),0.1);
            end
            % Adding the number of nodes that have been added by the
            % previous loop
            NewNodeIndex = NewNodeIndex + n;

            % Saving the coordinates of the new nodes for application later
            % on - loop cycles through all rows the "main" nodes are on
            for j=1:length(Midpoints)
                if (rem(Midpoints(j),0.1) == 0)
                    index = index + 1;
                    SubNodesCoords(index,1) = Midpoints(i);
                    SubNodesCoords(index,2) = Midpoints(j);
                end
            end
        % For the addition of edges to and from columns of "new" nodes -
        % i.e. "main" node columns
        else
            % Catching the "new" nodes at the top and bottom of each column
            InputGraph =
InputGraph.addedge(MainNodeNames(Loops*n+1),NewNodeNames(NewNodeIndex+1),0.1);
            InputGraph =
InputGraph.addedge(NewNodeNames(NewNodeIndex+1),MainNodeNames(Loops*n+1),0.1);
            InputGraph =
InputGraph.addedge(MainNodeNames(Loops*n+n),NewNodeNames(NewNodeIndex+(n-1)),0.1);
            InputGraph = InputGraph.addedge(NewNodeNames(NewNodeIndex+
(n-1)),MainNodeNames(Loops*n+n),0.1);
            % Connecting "new" nodes to the "main" nodes by the top and
            % bottom by adding edges for each "new" node in the column
            for k = 2:(n-1)
                InputGraph =
InputGraph.addedge(MainNodeNames(Loops*n+k),NewNodeNames(NewNodeIndex+k),0.1);
                InputGraph =
InputGraph.addedge(NewNodeNames(NewNodeIndex+k),MainNodeNames(Loops*n+k),0.1);
```

```
                InputGraph =
InputGraph.addedge(MainNodeNames(Loops*n+k),NewNodeNames(NewNodeIndex+k-1),0.1);
                InputGraph =
InputGraph.addedge(NewNodeNames(NewNodeIndex+k-1),MainNodeNames(Loops*n+k),0.1);
            end

            % Column index
            Loops = Loops + 1;
            % Adding the number of nodes that have been added by the
            % previous loop
            NewNodeIndex = NewNodeIndex + (n-1);
            % Saving the coordinates of the new nodes for application later
            % on - loop cycles through all rows the "main" nodes are NOT on
            for j=1:length(Midpoints)
                if (rem(Midpoints(j),0.1) ~= 0)
                    index = index + 1;
                    SubNodesCoords(index,1) = Midpoints(i);
                    SubNodesCoords(index,2) = Midpoints(j);
                end
            end
        end
    end

    % Saving the graph, node names and midpoints for output
    OutputGraph = InputGraph;
    OutputMainNodeNames = MainNodeNames;
    OutputNewNodeNames = NewNodeNames;
    OutputMidpoints = Midpoints;

    % Setting the output node x and y coordinates for output
    NewXData = [InitialXData SubNodesCoords(:,1)'];
    NewYData = [InitialYData SubNodesCoords(:,2)'];

    OutputXData = NewXData;
    OutputYData = NewYData;

    % Plotting the graph and labelling nodes
    OutputPlot = plot(InputGraph,'NodeLabel', InputGraph.Nodes.Name);

    % Setting the coordinates of the nodes on the plot
    OutputPlot.XData = NewXData;
    OutputPlot.YData = NewYData;
end

% creating the figure
f2b2 = figure;

% Calling the function making sure to specify the output variables if
% necessary
```

```
[Q2bGWithMidpoints, Q2bhWithMidpoints,
Q2bWithMidpointsMainNodeNames ,Q2bWithMidpointsNewNodeNames, ...
    Q2bWithMidpointsMidpoints, Q2bWithMidpointsXData, Q2bWithMidpointsYData] = ...
    ManhattanSquareGridWithMidpoints(N,L,Q2bG);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```



(c) Investigate designs for LTNs. You might want to try and make some streets oneway (for example, by closing links A and B in the above figure) or blocking off the ends of streets at some junctions (for example, by closing links B and D in the above figure). Good designs will try to maximise the number of blocked-off streets (which are thus free of through traffic) and make large numbers of the remaining through streets one-way, in order to simplify the traffic flow. However, each street's residents must retain a viable route to each corner of the network (and be able to return to their homes in the evening). Describe, develop, and demonstrate Matlab code which implements and displays your modified networks and which verifies the required connectivity. Note: highest marks will be reserved for solutions that work for any value of n, but if you find that too hard, setting n to a fixed value larger or equal to 10 is fine.

In this question, four designs of LTNs are considered using varying degrees of one-way and blocked streets at junctions. Once again, these are all scalable for any value of n. The four designs of LTN are:

LTN Design 1 - Making each row and column become one way, alternating in direction and maintaining the outer two-way streets.

Steps:

1. Iterating through all columns, using an if statement to select "new" and "main" node columns
2. In "new" node columns, all "inner" rows have one edge on the left and one edge on the right deleted horizontally before this is alternated for each row
3. In "main" node columns, the top and bottom nodes have one top and bottom edge deleted vertically and then the central nodes all have the same directed edge deleted to create a one-way street

Positives: Good implementation of alternating direction one-way streets in the vertical column direction. Relatively simple to implement with few "if" statements

Negatives: Doesn't implement one-way streets in the horizontal direction properly

LTN Design 2 - Similar to LTN Design 1 but maintaining a "cross" of two-way streets at the central nodes.

Steps:

1. Iterating through all columns, using an if statement to select "new" and "main" node columns but misses out the central column
2. In "new" node columns, all "inner" rows have one edge on the left and one edge on the right deleted horizontally before this is alternated for each row but misses out the central row
3. In "main" node columns, the top and bottom nodes have one top and bottom edge deleted vertically and then the central nodes all have the same directed edge deleted to create a one-way street

Positives: Once again good implementation of vertical one-way streets and helps traffic flow by keeping the two-way streets

Negatives: Doesn't implement one-way streets in the horizontal direction properly and probably doesn't add to much to ease traffic. Uses "if" statements to avoid central columns which are inefficient.

LTN Design 3 - Similar to LTN Design 1 but each row only contains one two way-street that is blocked on the left side

Steps:

1. Iterating through all columns, using an if statement to select "new" and "main" node columns
2. In "new" node columns, all "inner" rows have both edges on the left deleted horizontally before this is alternated for each row
3. In "main" node columns, the top and bottom nodes have one top and bottom edge deleted vertically and then the central nodes all have the same directed edge deleted to create a one-way street

Positives: Good implementation of one-way streets and useful road blocks that don't put too much pressure on most roads. A good implementation.

Negatives: Best implementation, only criticism is the most right "central" nodes are connected to the outer two way streets which could apply a lot of pressure there

LTN Design 4 - A combination of all i.e. similar to LTN Design 1 but each row only contains one two way-street that is blocked on the left but maintaining a "cross" at the central nodes of two-way streets.

Steps:

1. Combination of all of the above
2. Iterating through all columns, using an if statement to select "new" and "main" node columns but misses out the central column
3. In "new" node columns, all "inner" rows have both edges on the left deleted horizontally before this is alternated for each row but misses out the central row
4. In "main" node columns, the top and bottom nodes have one top and bottom edge deleted vertically and then the central nodes all have the same directed edge deleted to create a one-way street

Positives: Good implementation of one-way streets and useful road blocks that don't put too much pressure on most roads. Central two-way streets does aid traffic flow

Negatives: Once again, most right "central" nodes are connected to the outer two way streets which could apply a lot of pressure and uses if" statements to avoid central columns which are inefficient.

It is validated that each resident still has a viable route to each corner by the destination being randomly assigned and the connectivity calculated. A connectivity of 1 suggests all nodes can reach all other nodes and therefore their destination.

However, it is appreciated that although these designs may be good, they are not the best available with further road closures and one-way streets possible in all LTN designs.

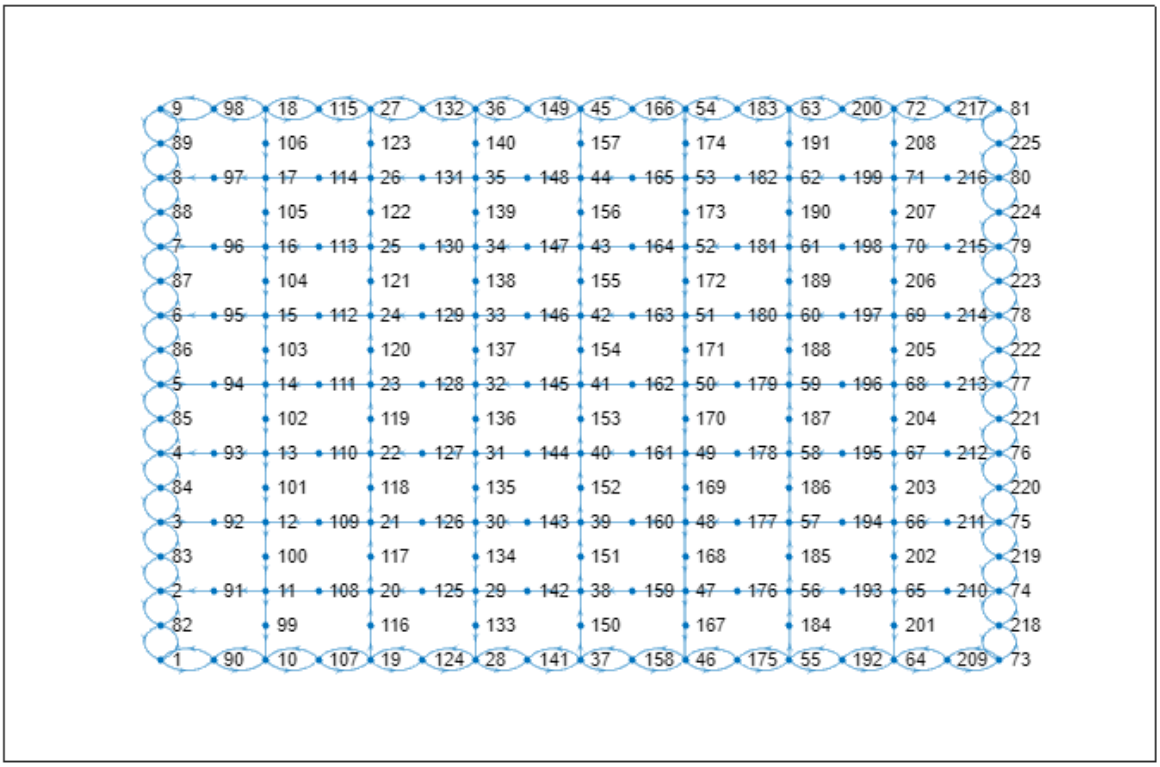
LTN design 1 - Maintaining outside loop + one way

```
% LTN design 1 - Maintaining outside loop + one way

% Setting up initial Manhattan-style square grid
f2c1a = figure('Visible','off');
[Q2cGLTN1, ~] = ManhattanSquareGrid(N,L);

% Setting up the "new" grid which includes the "new" midpoint nodes
f2c1b = figure('Visible','off');
[Q2cGLTN1, ~, Q2cGLTN1MainNodeNames, Q2cGLTN1NewNodeNames, Q2cGLTN1Midpoints,
Q2cGLTN1XData, Q2cGLTN1YData] = ...
    ManhattanSquareGridWithMidpoints(N,L,Q2cGLTN1);
```

```
Loops = 1; % this cycles through each column
NewNodeIndex = (N-1); % New nodes i.e. 101 to 280 fo n = 10
XAlt = 1; % Alternating rows index
YAlt = 1; % Alternating columns index


for i=1:length(Q2cGLTN1Midpoints)
    % "New" node columns
    if (rem(Q2cGLTN1Midpoints(i),0.1) ~= 0)
        % Iterating through th rows - missing out the "outer" rows
        for k = 2:(N-1)
            if XAlt == 1
                % Deleting left and right bits of columns horizontally
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames((Loops-1)*N+k),Q2cGLTN1NewNodeNames(NewNodeInd
ex+k));
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+k),Q2cGLTN1MainNodeNames((Loops)*N
+k));
                % Making sure to alternate
                XAlt = 2;
            elseif XAlt == 2
                % Deleting left and right bits of columns horizontally
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+k),Q2cGLTN1MainNodeNames((Loops-1
)*N+k));
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames((Loops)*N+k),Q2cGLTN1NewNodeNames(NewNodeIndex
+k));
                % Making sure to alternate
                XAlt = 1;
            end
        end
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + N;
    % "Old" node columns - missing out the "outer" columns
    elseif (Q2cGLTN1Midpoints(i)> Q2cGLTN1Midpoints(1) && Q2cGLTN1Midpoints(i) <
Q2cGLTN1Midpoints(end-1))
        if YAlt == 1
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(Loops*N+1),Q2cGLTN1NewNodeNames(NewNodeIndex+1
));
            Q2cGLTN1 = Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+
(N-1)),Q2cGLTN1MainNodeNames(Loops*N+N));
            % Iterating through the "inner" rows
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
```

```
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(Loops*N+k),Q2cGLTN1NewNodeNames(NewNodeIndex+k
));
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+k-1),Q2cGLTN1MainNodeNames(Loops*N
+k));
            end
            % Making sure to alternate
            YAlt = 2;
        elseif YAlt == 2
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+1),Q2cGLTN1MainNodeNames(Loops*N+1
));
            Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(Loops*N+N),Q2cGLTN1NewNodeNames(NewNodeIndex+
(N-1)));
            % Iterating through the "inner" rows
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(NewNodeIndex+k),Q2cGLTN1MainNodeNames(Loops*N+k
));
                Q2cGLTN1 =
Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(Loops*N+k),Q2cGLTN1NewNodeNames(NewNodeIndex+k
-1));
            end
            % Making sure to alternate
            YAlt = 1;
        end

        % Column index
        Loops = Loops + 1;
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + (N-1);
    end
end

% Plotting updated graph
f2c1c = figure;
Q2hLTN1 = plot(Q2cGLTN1,'NodeLabel',Q2cGLTN1.Nodes.Name);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])

% Specifying graph node coordinates
hold on;
Q2hLTN1.XData = Q2cGLTN1XData;
Q2hLTN1.YData = Q2cGLTN1YData;
```

```
hold off;
```




```
% Verifying required connectivity -> should be 1
sum(conncomp(Q2cGLTN1,'Type','strong'))/(length(Q2cGLTN1XData))
```

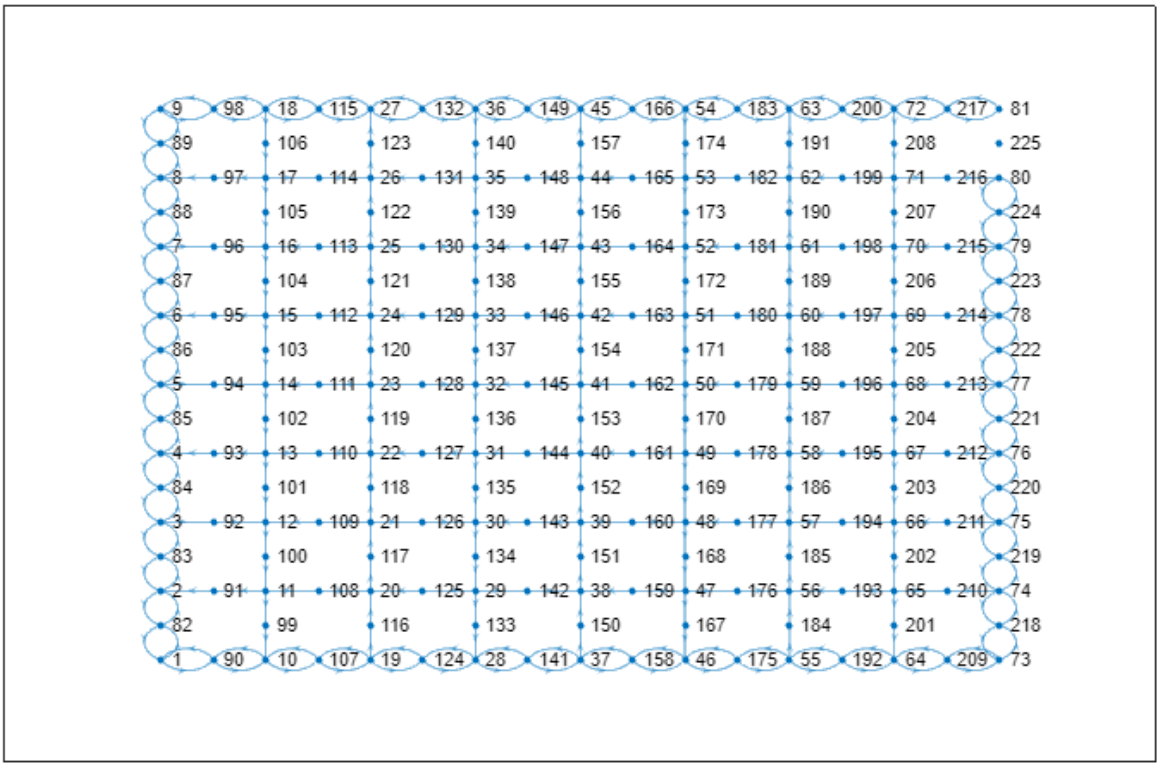
```
ans = 1
```

```
% Disconnecting the final "new" node in the top right corner of the plot as
% a sanity check
Q2cGLTN1 = Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(end),Q2cGLTN1MainNodeNames(end));
Q2cGLTN1 = Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(end),Q2cGLTN1NewNodeNames(end));
Q2cGLTN1 = Q2cGLTN1.rmedge(Q2cGLTN1NewNodeNames(end),Q2cGLTN1MainNodeNames(end-1));
Q2cGLTN1 = Q2cGLTN1.rmedge(Q2cGLTN1MainNodeNames(end-1),Q2cGLTN1NewNodeNames(end));

% Now all nodes will have varying values due to their independent
% connectivity
sum(conncomp(Q2cGLTN1,'Type','strong'))/(length(Q2cGLTN1XData))
```

```
ans = 1.9956
```

```
% Plot showing the disconnected node
f2c1d = figure;
Q2hLTN1 = plot(Q2cGLTN1,'NodeLabel',Q2cGLTN1.Nodes.Name);
set(gcf,'units','inches','position',[0,0,16,10])
```

```
hold on;
Q2hLTN1.XData = Q2cGLTN1XData;
Q2hLTN1.YData = Q2cGLTN1YData;
hold off;
```



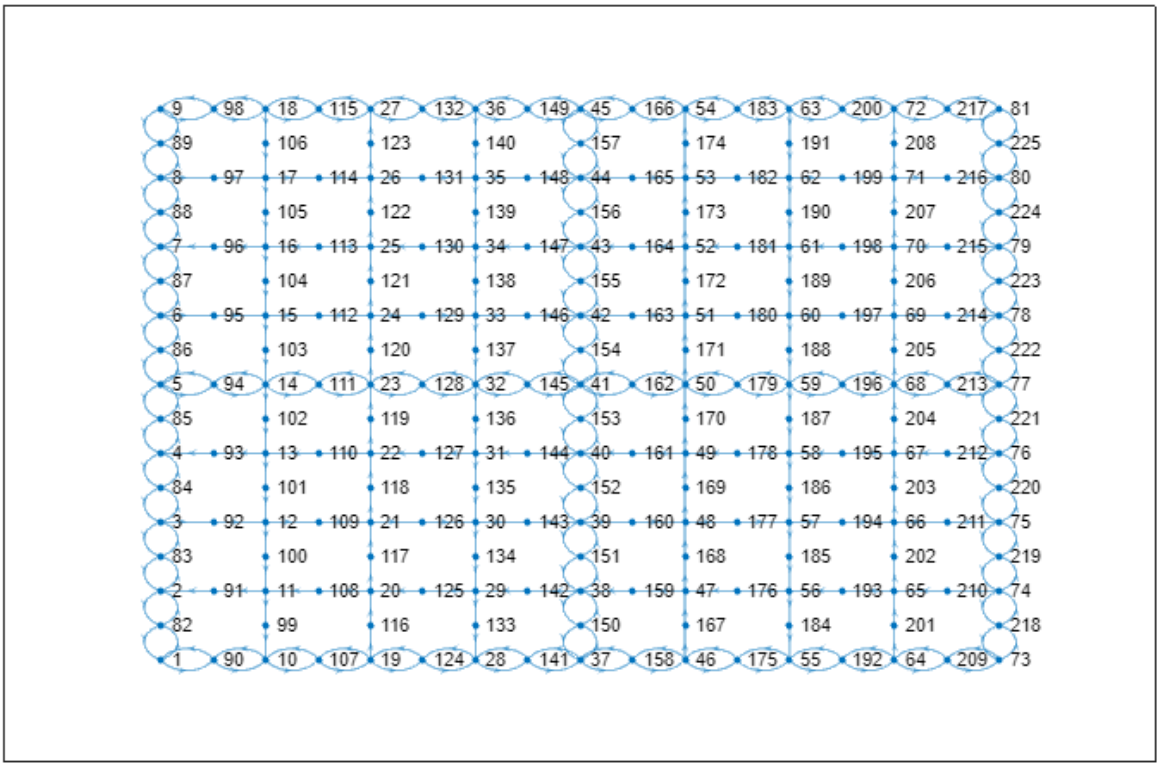
LTN design 2 Maintaining outside loop + one way + cross

```
% LTN design 2 Maintaining outside loop + one way + cross

% Setting up initial Manhattan-style square grid
f2c2a = figure('Visible','off');
[Q2cGLTN2, ~] = ManhattanSquareGrid(N,L);

% Setting up the "new" grid which includes the "new" midpoint nodes
f2c2b = figure('Visible','off');
[Q2cGLTN2, ~, Q2cGLTN2MainNodeNames, Q2cGLTN2NewNodeNames, Q2cGLTN2Midpoints,
Q2cGLTN2XData, Q2cGLTN2YData] = ...
    ManhattanSquareGridWithMidpoints(N,L,Q2cGLTN2);

Loops = 1; % this cycles through each column
NewNodeIndex = (N-1); % New nodes i.e. 101 to 280
XAlt = 1; % Alternating rows
YAlt = 1; % Alternating columns
```

```
% Skipping out the middle row
Rows = 2:(N-1);
Rows(round(end/2)) = [];

for i=1:length(Q2cGLTN2Midpoints)
    % Skipping out the middle column
    if (rem(N,2) == 0 && Q2cGLTN2Midpoints(i) == Q2cGLTN2Midpoints(round(end/2)))
        NewNodeIndex = NewNodeIndex + N;
    elseif (rem(N,2) ~= 0 && Q2cGLTN2Midpoints(i) == Q2cGLTN2Midpoints(round(end/
2)))
        NewNodeIndex = NewNodeIndex + N-1;
        Loops = Loops + 1;
    % "New" node columns
    elseif (rem(Q2cGLTN2Midpoints(i),0.1) ~= 0)
        % Iterating through th rows - missing out the "outer" and middle rows
        for k = Rows
            if XAlt == 1
                % Deleting left and right bits of columns horizontally
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames((Loops-1)*N+k),Q2cGLTN2NewNodeNames(NewNodeInd
ex+k));
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+k),Q2cGLTN2MainNodeNames((Loops)*N
+k));
                % Making sure to alternate
                XAlt = 2;
            elseif XAlt == 2
                % Deleting left and right bits of columns horizontally
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+k),Q2cGLTN2MainNodeNames((Loops-1
)*N+k));
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames((Loops)*N+k),Q2cGLTN2NewNodeNames(NewNodeIndex
+k));
                % Making sure to alternate
                XAlt = 1;
            end
        end
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + N;
    % "Old" node columns - missing out the "outer" columns
    elseif (Q2cGLTN2Midpoints(i)> Q2cGLTN2Midpoints(1) && Q2cGLTN2Midpoints(i) <
Q2cGLTN2Midpoints(end-1))
        if YAlt == 1
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(Loops*N+1),Q2cGLTN2NewNodeNames(NewNodeIndex+1
));
```

```
            Q2cGLTN2 = Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+
(N-1)),Q2cGLTN2MainNodeNames(Loops*N+N));
            % Iterating through the "inner" rows
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(Loops*N+k),Q2cGLTN2NewNodeNames(NewNodeIndex+k
));
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+k-1),Q2cGLTN2MainNodeNames(Loops*N
+k));
            end
            % Making sure to alternate
            YAlt = 2;
        elseif YAlt == 2
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+1),Q2cGLTN2MainNodeNames(Loops*N+1
));
            Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(Loops*N+N),Q2cGLTN2NewNodeNames(NewNodeIndex+
(N-1)));
            % Iterating through the "inner" rows
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(NewNodeIndex+k),Q2cGLTN2MainNodeNames(Loops*N+k
));
                Q2cGLTN2 =
Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(Loops*N+k),Q2cGLTN2NewNodeNames(NewNodeIndex+k
-1));
            end
            % Making sure to alternate
            YAlt = 1;
        end
        % Column index
        Loops = Loops + 1;
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + (N-1);
    end
end
% Plotting updated graph
f2c2c = figure;
Q2hLTN2 = plot(Q2cGLTN2,'NodeLabel',Q2cGLTN2.Nodes.Name);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])

% Specifying graph node coordinates
```

```
hold on;
Q2hLTN2.XData = Q2cGLTN2XData;
Q2hLTN2.YData = Q2cGLTN2YData;
hold off;
```




```
% Verifying required connectivity -> should be 1
sum(conncomp(Q2cGLTN2,'Type','strong'))/(length(Q2cGLTN2XData))
```

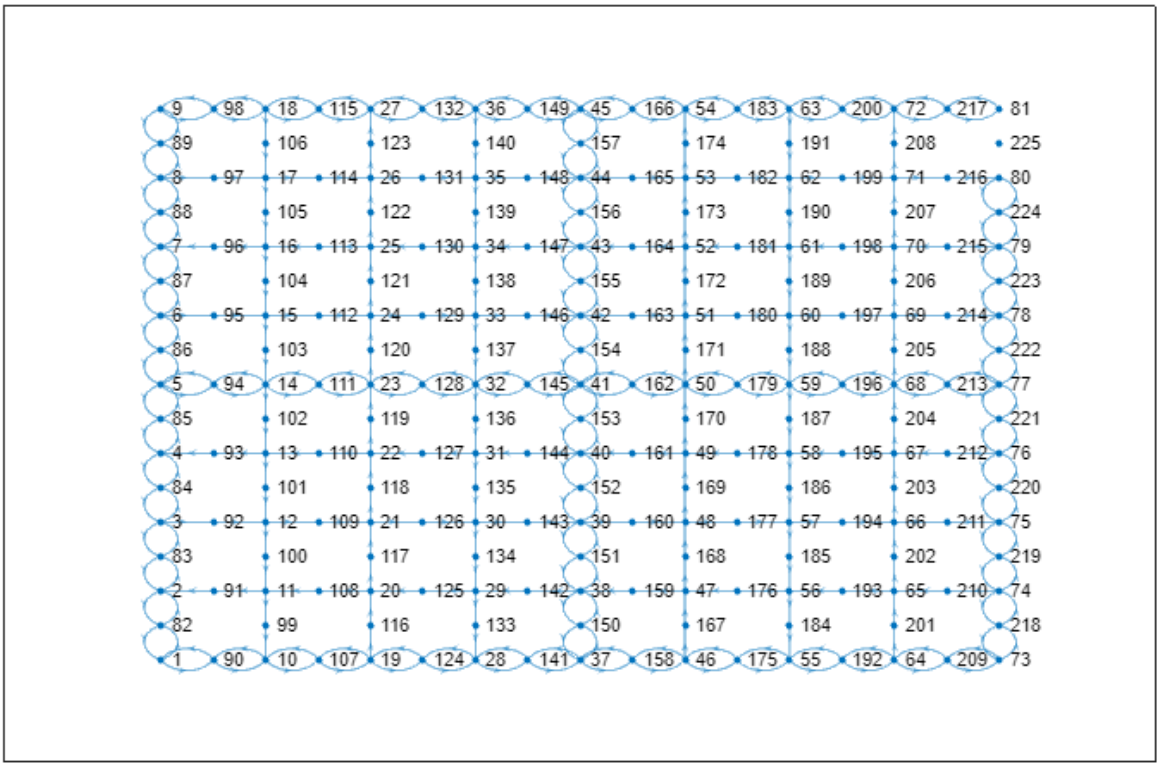
```
ans = 1
```

```
% Disconnecting the final "new" node in the top right corner of the plot as
% a sanity check
Q2cGLTN2 = Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(end),Q2cGLTN2MainNodeNames(end));
Q2cGLTN2 = Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(end),Q2cGLTN2NewNodeNames(end));
Q2cGLTN2 = Q2cGLTN2.rmedge(Q2cGLTN2NewNodeNames(end),Q2cGLTN2MainNodeNames(end-1));
Q2cGLTN2 = Q2cGLTN2.rmedge(Q2cGLTN2MainNodeNames(end-1),Q2cGLTN2NewNodeNames(end));

% Now all nodes will have varying values due to their independent
% connectivity
sum(conncomp(Q2cGLTN2,'Type','strong'))/(length(Q2cGLTN2XData))
```

```
ans = 1.9956
```

```
% Plot showing the disconnected node
```

```
f2c2d = figure;
Q2hLTN2 = plot(Q2cGLTN2,'NodeLabel',Q2cGLTN2.Nodes.Name);
set(gcf,'units','inches','position',[0,0,16,10])
hold on;
Q2hLTN2.XData = Q2cGLTN2XData;
Q2hLTN2.YData = Q2cGLTN2YData;
hold off;
```

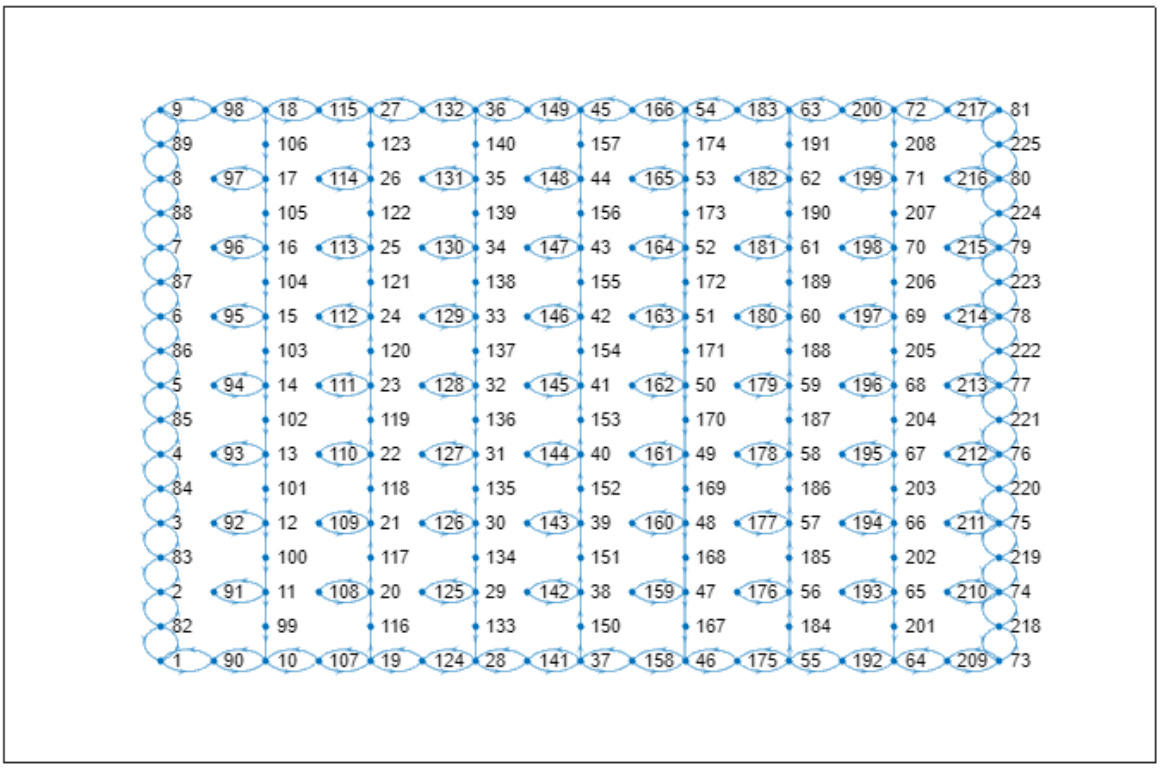


LTN design 3 - Maintaining outside loop + one way + blocking off

```
% LTN design 3 - Maintaining outside loop + one way + blocking off

% Setting up initial Manhattan-style square grid
f2c3a = figure('Visible','off');
[Q2cGLTN3, ~] = ManhattanSquareGrid(N,L);

% Setting up the "new" grid which includes the "new" midpoint nodes
f2c3b = figure('Visible','off');
[Q2cGLTN3, ~, Q2cGLTN3MainNodeNames, Q2cGLTN3NewNodeNames, Q2cGLTN3Midpoints,
Q2cGLTN3XData, Q2cGLTN3YData] = ...
    ManhattanSquareGridWithMidpoints(N,L,Q2cGLTN3);

Loops = 1; % this cycles through each column
```

```
NewNodeIndex = (N-1); % New nodes i.e. 101 to 280
XAlt = 1; % Alternating rows
YAlt = 1; % Alternating columns


for i=1:length(Q2cGLTN1Midpoints)
    % "New" node columns
    if (rem(Q2cGLTN1Midpoints(i),0.1) ~= 0)
        % Iterating through th rows - missing out the "outer" and middle rows
        for k = 2:(N-1)
            if XAlt == 1
                % Deleting left and right bits of columns horizontally
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames((Loops-1)*N+k),Q2cGLTN3NewNodeNames(NewNodeInd
ex+k));
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+k),Q2cGLTN3MainNodeNames((Loops-1
)*N+k));
                % Making sure to alternate
                XAlt = 2;
            elseif XAlt == 2
                % Deleting left and right bits of columns horizontally
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+k),Q2cGLTN3MainNodeNames((Loops-1
)*N+k));
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames((Loops-1)*N+k),Q2cGLTN3NewNodeNames(NewNodeInd
ex+k));
                % Making sure to alternate
                XAlt = 1;
            end
        end
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + N;
    % "Old" node columns - missing out the "outer" columns
    elseif (Q2cGLTN3Midpoints(i)> Q2cGLTN3Midpoints(1) && Q2cGLTN3Midpoints(i) <
Q2cGLTN3Midpoints(end-1))
        if YAlt == 1
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(Loops*N+1),Q2cGLTN3NewNodeNames(NewNodeIndex+1
));
            Q2cGLTN3 = Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+
(N-1)),Q2cGLTN3MainNodeNames(Loops*N+N));
            for k = 2:(N-1)
                % % Deleting central bits of columns vertically
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(Loops*N+k),Q2cGLTN3NewNodeNames(NewNodeIndex+k
));
```

```
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+k-1),Q2cGLTN3MainNodeNames(Loops*N
+k));
            end
            % Making sure to alternate
            YAlt = 2;
        elseif YAlt == 2
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+1),Q2cGLTN3MainNodeNames(Loops*N+1
));
            Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(Loops*N+N),Q2cGLTN3NewNodeNames(NewNodeIndex+
(N-1)));
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(NewNodeIndex+k),Q2cGLTN3MainNodeNames(Loops*N+k
));
                Q2cGLTN3 =
Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(Loops*N+k),Q2cGLTN3NewNodeNames(NewNodeIndex+k
-1));
            end
            %Making sure to alternate
            YAlt = 1;
        end
        % Column index
        Loops = Loops + 1;
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + (N-1);
    end
end
% Plotting updated graph
f2c3c = figure;
Q2hLTN3 = plot(Q2cGLTN3,'NodeLabel',Q2cGLTN3.Nodes.Name);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])

% Specifying node coordinates
hold on;
Q2hLTN3.XData = Q2cGLTN3XData;
Q2hLTN3.YData = Q2cGLTN3YData;
hold off;
```




```
% Verifying required connectivity -> should be 1
sum(conncomp(Q2cGLTN3,'Type','strong'))/(length(Q2cGLTN3XData))
```

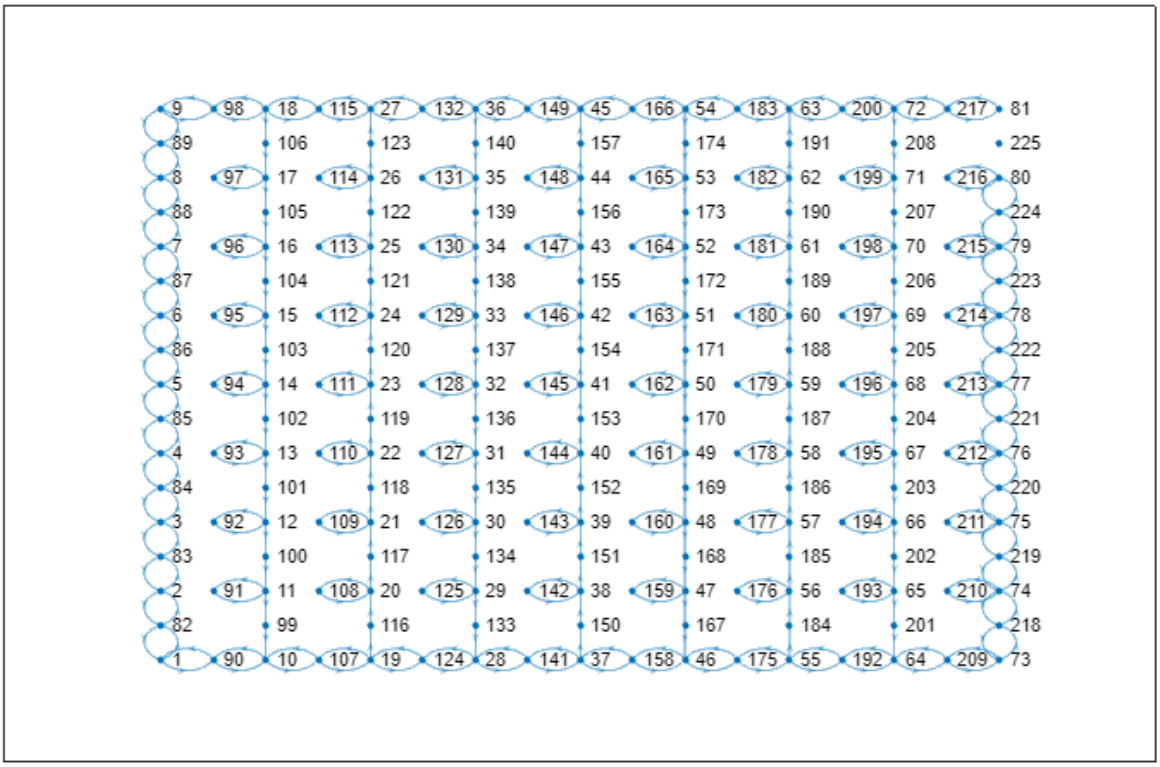
```
ans = 1
```

```
% Disconnecting the final "new" node in the top right corner of the plot as
% a sanity check
Q2cGLTN3 = Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(end),Q2cGLTN3MainNodeNames(end));
Q2cGLTN3 = Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(end),Q2cGLTN3NewNodeNames(end));
Q2cGLTN3 = Q2cGLTN3.rmedge(Q2cGLTN3NewNodeNames(end),Q2cGLTN3MainNodeNames(end-1));
Q2cGLTN3 = Q2cGLTN3.rmedge(Q2cGLTN3MainNodeNames(end-1),Q2cGLTN3NewNodeNames(end));

% Now all nodes will have varying values due to their independent
% connectivity
sum(conncomp(Q2cGLTN3,'Type','strong'))/(length(Q2cGLTN3XData))
```

```
ans = 1.9956
```

```
% Plot showing the disconnected node
f2c3d = figure;
Q2hLTN3 = plot(Q2cGLTN3,'NodeLabel',Q2cGLTN3.Nodes.Name);
set(gcf,'units','inches','position',[0,0,16,10])
hold on;
Q2hLTN3.XData = Q2cGLTN3XData;
```

```
Q2hLTN3.YData = Q2cGLTN3YData;
hold off;
```



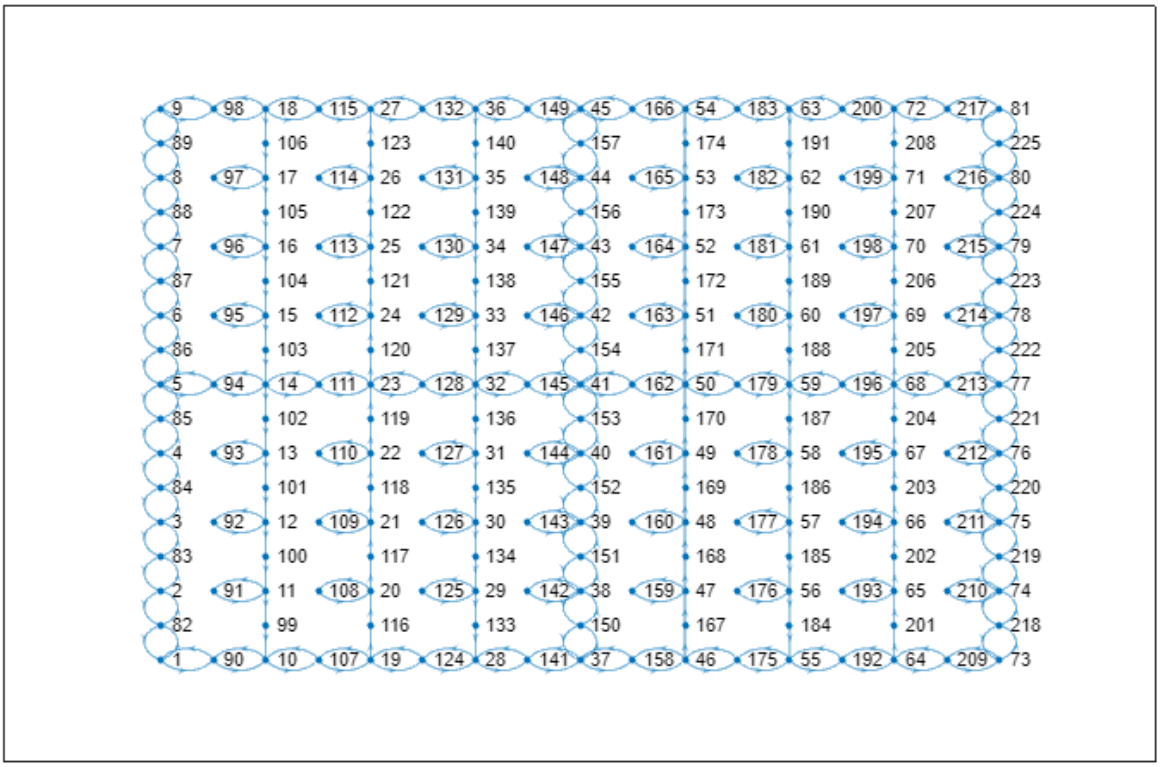
LTN design 4 - Maintaining outside loop + one way + cross + blocked streets

```
% LTN design 4 -  Maintaining outside loop + one way + cross +
% blocked streets

% Setting up initial Manhattan-style square grid
f2c4a = figure('Visible','off');
[Q2cGLTN4, ~] = ManhattanSquareGrid(N,L);

% Setting up the "new" grid which includes the "new" midpoint nodes
f2c4b = figure('Visible','off');
[Q2cGLTN4, ~, Q2cGLTN4MainNodeNames, Q2cGLTN4NewNodeNames, Q2cGLTN4Midpoints,
Q2cGLTN4XData, Q2cGLTN4YData] = ...
    ManhattanSquareGridWithMidpoints(N,L,Q2cGLTN4);

Loops = 1; % this cycles through each column
NewNodeIndex = (N-1); % New nodes i.e. 101 to 280
XAlt = 1; % Alternating rows
YAlt = 1; % Alternating columns
```

```
% Skipping out the middle row
Rows = 2:(N-1);
Rows(round(end/2)) = [];

for i=1:length(Q2cGLTN4Midpoints)
    % Skipping out the middle column
    if (rem(N,2) == 0 && Q2cGLTN4Midpoints(i) == Q2cGLTN4Midpoints(round(end/2)))
        NewNodeIndex = NewNodeIndex + (N);
    elseif (rem(N,2) ~= 0 && Q2cGLTN4Midpoints(i) == Q2cGLTN4Midpoints(round(end/
2)))
        NewNodeIndex = NewNodeIndex + (N-1);
        Loops = Loops + 1;
    % "New" node columns
    elseif (rem(Q2cGLTN4Midpoints(i),0.1) ~= 0)
        % Iterating through th rows - missing out the "outer" and middle rows
        for k = Rows
            if XAlt == 1
                % Deleting left and right bits of columns horizontally
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames((Loops-1)*N+k),Q2cGLTN4NewNodeNames(NewNodeInd
ex+k));
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+k),Q2cGLTN4MainNodeNames((Loops-1
)*N+k));
                % Making sure to alternate
                XAlt = 2;
            elseif XAlt == 2
                % Deleting left and right bits of columns horizontally
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+k),Q2cGLTN4MainNodeNames((Loops-1
)*N+k));
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames((Loops-1)*N+k),Q2cGLTN4NewNodeNames(NewNodeInd
ex+k));
                % Making sure to alternate
                XAlt = 1;
            end
        end
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + N;
    % "Old" node columns - missing out the "outer" columns
    elseif (Q2cGLTN4Midpoints(i)> Q2cGLTN4Midpoints(1) && Q2cGLTN4Midpoints(i) <
Q2cGLTN4Midpoints(end-1))
        if YAlt == 1
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(Loops*N+1),Q2cGLTN4NewNodeNames(NewNodeIndex+1
));
```

```
            Q2cGLTN4 = Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+
(N-1)),Q2cGLTN4MainNodeNames(Loops*N+N));
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(Loops*N+k),Q2cGLTN4NewNodeNames(NewNodeIndex+k
));
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+k-1),Q2cGLTN4MainNodeNames(Loops*N
+k));
            end
            % Making sure to alternate
            YAlt = 2;
        elseif YAlt == 2
            % Deleting top and bottom bits of columns vertically
            Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+1),Q2cGLTN4MainNodeNames(Loops*N+1
));
            Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(Loops*N+N),Q2cGLTN4NewNodeNames(NewNodeIndex+
(N-1)));
            for k = 2:(N-1)
                % Deleting central bits of columns vertically
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(NewNodeIndex+k),Q2cGLTN4MainNodeNames(Loops*N+k
));
                Q2cGLTN4 =
Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(Loops*N+k),Q2cGLTN4NewNodeNames(NewNodeIndex+k
-1));
            end
            % Making sure to alternate
            YAlt = 1;
        end
        % Column index
        Loops = Loops + 1;
        % Adding the number of nodes that have been added by the
        % previous loop
        NewNodeIndex = NewNodeIndex + (N-1);
    end
end
% Plotting updated graph
f2c4c = figure;
Q2hLTN4 = plot(Q2cGLTN4,'NodeLabel',Q2cGLTN4.Nodes.Name);

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])

% Specifying node coordinates
hold on;
Q2hLTN4.XData = Q2cGLTN4XData;
```

```
Q2hLTN4.YData = Q2cGLTN4YData;
hold off;
```




```
% Verifying required connectivity -> should be 1
sum(conncomp(Q2cGLTN4,'Type','strong'))/(length(Q2cGLTN4XData))
```

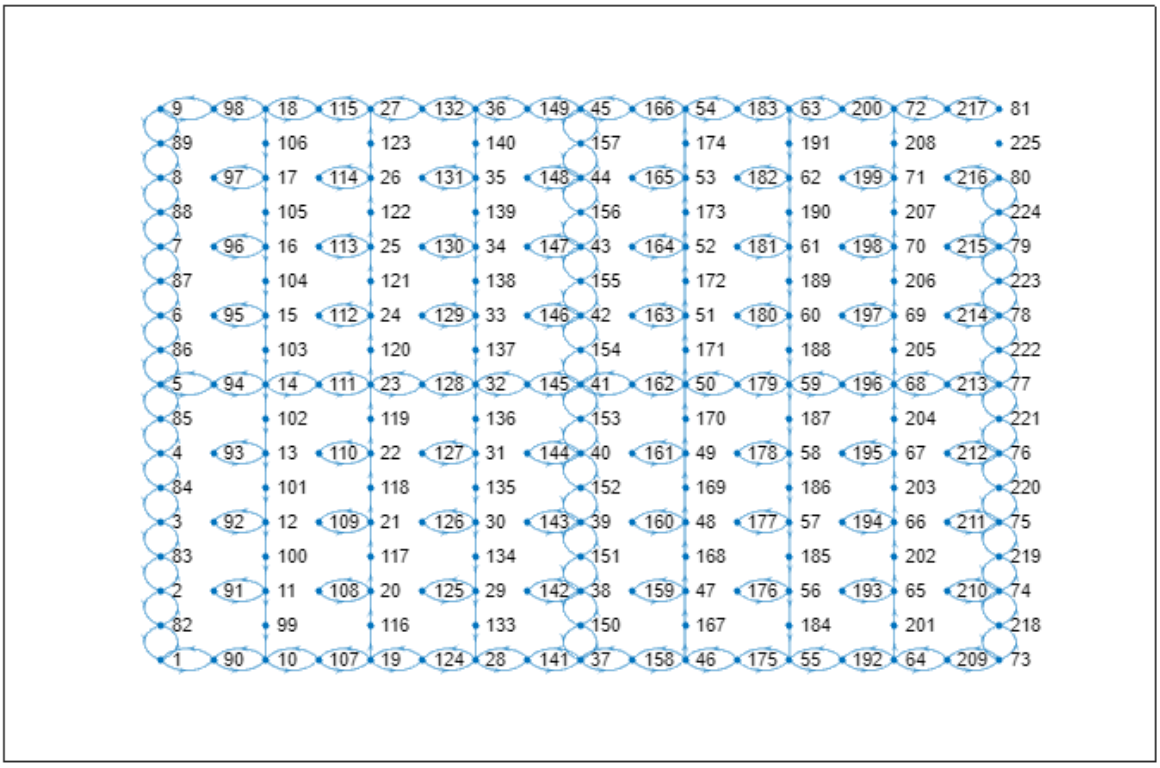
```
ans = 1
```

```
% Disconnecting the final "new" node in the top right corner of the plot as
% a sanity check
Q2cGLTN4 = Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(end),Q2cGLTN4MainNodeNames(end));
Q2cGLTN4 = Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(end),Q2cGLTN4NewNodeNames(end));
Q2cGLTN4 = Q2cGLTN4.rmedge(Q2cGLTN4NewNodeNames(end),Q2cGLTN4MainNodeNames(end-1));
Q2cGLTN4 = Q2cGLTN4.rmedge(Q2cGLTN4MainNodeNames(end-1),Q2cGLTN4NewNodeNames(end));

% Now all nodes will have varying values due to their independent
% connectivity
sum(conncomp(Q2cGLTN4,'Type','strong'))/(length(Q2cGLTN4XData))
```

```
ans = 1.9956
```

```
% Plot showing the disconnected node
f2c4d = figure;
Q2hLTN4 = plot(Q2cGLTN4,'NodeLabel',Q2cGLTN4.Nodes.Name);
set(gcf,'units','inches','position',[0,0,16,10])
```

```
hold on;
Q2hLTN4.XData = Q2cGLTN4XData;
Q2hLTN4.YData = Q2cGLTN4YData;
hold off;
```



(d) Let us suppose that all travellers take the shortest path from their residence to their respective destination (although noting that depending how many links you have deleted — that shortest path may not be unique). Also, for simplicity, you may assume that each street generates exactly the same demand to each of the four corners of the network. Describe, develop, and demonstrate Matlab code which computes the traffic flow on each street for your proposed LTN design. How do flows compare with the setting where no LTN interventions are made? Discuss briefly possible trade-offs in your design: e.g., close off fewer streets, and maybe improve the capacity of others?

A key point to note here is that due to the destination being randomised between the four corners, each time the code is run the traffic flow on the network changes. This is the main limitation of how this code has been implemented.

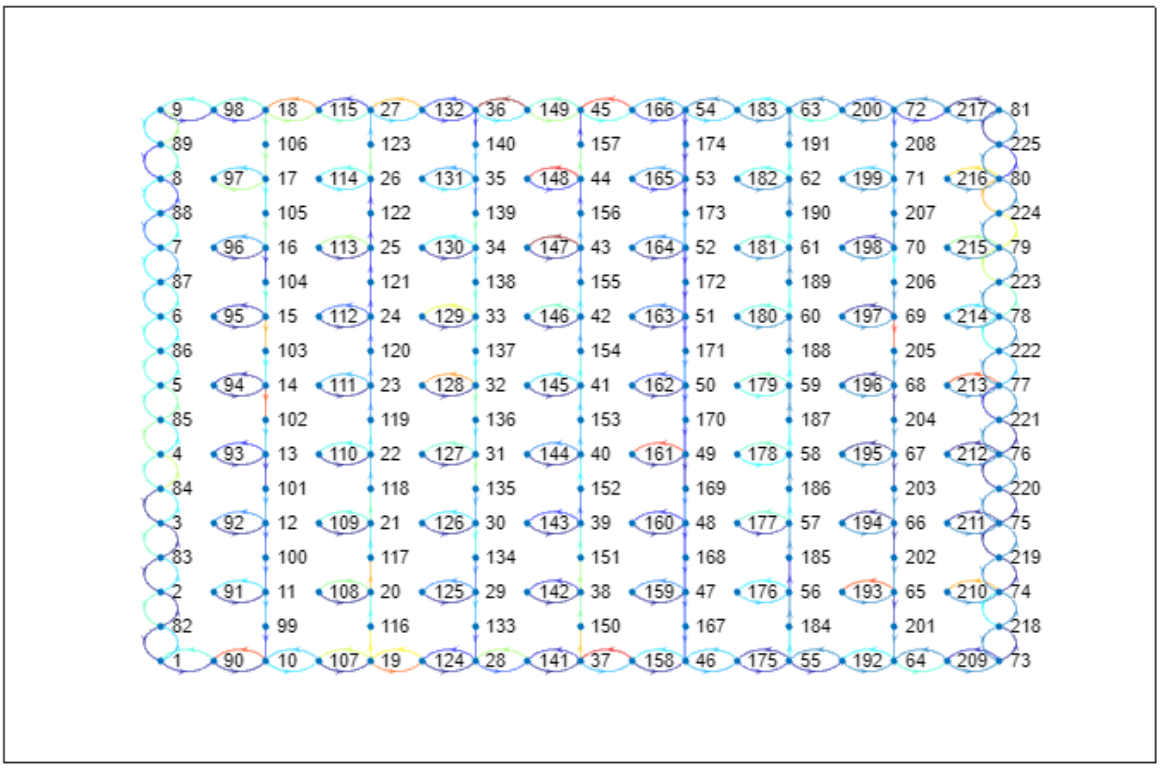
- Traffic flow using the desired LTN Design 3 shows a couple of interesting trends. The first, is that the outer two-way streets seem to carry most of the traffic, especially the left hand side outer nodes. This is what is desired from this network with the outer edges carrying most of the traffic flow much like a ring road would.

- However, something interesting that is commonly spotted when the LTN is implemented is the increase in frequencies on the two-way open sides of blocked-off streets. These streets have some of the highest traffic flows and the reason assumed is that the large traffic flow on the neighbouring one-way street could cause this. However, all nodes have the same amount of generation and so it is not fully clear why this occurs.
- Compared to not using an LTN, traffic appears to not utilise most links particularly frequently. There are however still a couple of links with extremely high flows espcially close to the destinations, in similar places to those seen when the LTN is implemented (especially the left hand side), with others spread out within the network randomly.
- Some trade-offs do appear with the use of blocked-off streets seen as they are only appropriate in some parts of the network, particularly not in the centre. However, overall they do have a traffic calming effect.
- The increase in capacity of the outer streets is also seen as an opportunity to reduce congestion throughout the whole network, especially close to the destinations. This would be much like a motorway ring road e.g. the M25.
- It could therefore be assumed that the implementation of an LTN, if done correctly, would have minimal impact on traffic flow and avoid extra congested for the most part. However this is very dependent on generation and destination nodes of journeys.

```
% Adding the nodes back into the graph

Q2cGLTN3 =
Q2cGLTN3.addedge(Q2cGLTN3NewNodeNames(end),Q2cGLTN3MainNodeNames(end),0.1);
Q2cGLTN3 =
Q2cGLTN3.addedge(Q2cGLTN3MainNodeNames(end),Q2cGLTN3NewNodeNames(end),0.1);
Q2cGLTN3 =
Q2cGLTN3.addedge(Q2cGLTN3NewNodeNames(end),Q2cGLTN3MainNodeNames(end-1),0.1);
Q2cGLTN3 =
Q2cGLTN3.addedge(Q2cGLTN3MainNodeNames(end-1),Q2cGLTN3NewNodeNames(end),0.1);

% Creating the figure of the "basic" design
f2d1 = figure('Visible','off');
[Q2dG, ~] = ManhattanSquareGrid(N,L);


% Creating the figure of the LTN design 3 - Maintaining outside loop + one way +
blocking off and
% specifying figure size
f2dLTN = figure;

Q2hLTN3 = plot(Q2cGLTN3,'NodeLabel',Q2cGLTN3.Nodes.Name);
set(gcf,'units','inches','position',[0,0,16,10]);

% Specifying node coordinates
hold on;
Q2hLTN3.XData = Q2cGLTN3XData;
Q2hLTN3.YData = Q2cGLTN3YData;
```

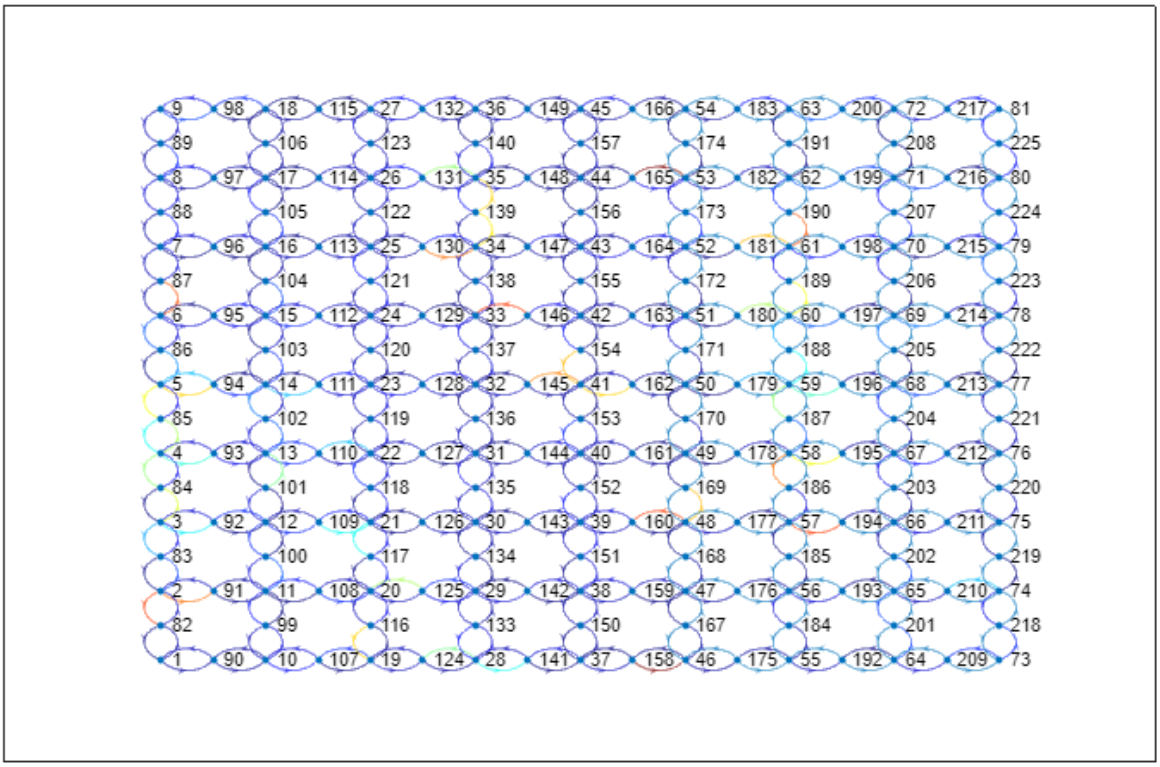
```
hold off;

f2dNoLTN = figure;

% Creating the graph for the basic design with midpoints and Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
[Q2dGWithMidpoints, Q2dhWithMidpoints,
Q2dWithMidpointsMainNodeNames ,Q2dWithMidpointsNewNodeNames,...
    Q2dWithMidpointsMidpoints,...
    Q2dWithMidpointsXData, Q2dWithMidpointsYData] =
ManhattanSquareGridWithMidpoints(N,L,Q2dG);

% Setting the names of the destination nodes
DestinationNodes = ["1" , string(N), string(N^2-N+1), string(N^2)];

% Setting the names of the ORIGIN of the trips nodes
OriginNodes = Q2cGLTN3NewNodeNames;

% Setting the number of nodes
TotalNodes = length(Q2cGLTN3MainNodeNames) + length(Q2cGLTN3NewNodeNames);

% Setting up the cell arrays to store the path node indices, length of shortest
path and
% indices of edges
LTNpaths = cell(TotalNodes,1);
LTNds = cell(TotalNodes,1);
LTNedgepaths = cell(TotalNodes,1);

NoLTNpaths = cell(TotalNodes,1);
NoLTNds = cell(TotalNodes,1);
NoLTNedgepaths = cell(TotalNodes,1);

% Iterating through all the origin nodes
for i=1:length(OriginNodes)
    % Randomly generating the destination
    Destination = randi([1 4]);
    %Saving the required paths and path lengths tp the relevant arrays
    [LTNpaths{i},LTNds{i},LTNedgepaths{i}] =
shortestpath(Q2cGLTN3,OriginNodes(i),DestinationNodes(Destination));
    [NoLTNpaths{i},NoLTNds{i},NoLTNedgepaths{i}] =
shortestpath(Q2dGWithMidpoints,OriginNodes(i),...
        DestinationNodes(Destination));
end

% Putting all edges used into a 1D array for frequency calculation
NoLTNtotaluniqueedges = [];
LTNtotaluniqueedges = [];
for i = 1:TotalNodes
    % Only saves unique paths
    if isempty(unique(LTNedgepaths{i})) == 0
```

```
        LTNtotaluniqueedges = [LTNtotaluniqueedges unique(LTNedgepaths{i})];
    end
    if isempty(unique(NoLTNedgepaths{i})) == 0
        NoLTNtotaluniqueedges = [NoLTNtotaluniqueedges unique(NoLTNedgepaths{i})];
    end
end

% Calculating frequency of usage for each edge
LTNedgefreq = histcounts(LTNtotaluniqueedges, unique(LTNtotaluniqueedges));
NoLTNedgefreq = histcounts(NoLTNtotaluniqueedges, unique(NoLTNtotaluniqueedges));

% Plotting the edge frequency on a colourmap for the LTN design

hold on;

% Setting up colourmap
cmap = jet(256);

colormap(cmap);
clim([min(LTNedgefreq), max(LTNedgefreq)]);

% Normalising edge frequencies
normedgefreq = normalize(LTNedgefreq,"range");

for i = 1:length(LTNedgefreq)
        %Specifying the colours of each edge
        colorIndex = round(normedgefreq(i)*255)+1;
        %Applying the colours to the relevant edge
        highlight(Q2hLTN3, 'Edges',i, 'EdgeColor', cmap(colorIndex,:))
end
```




```
hold off;


% Plotting the edge frequency on a colourmap for the no LTN (basic with midpoints)
design
hold on;

% Setting up colourmap
cmap = jet(256);

colormap(cmap);
clim([min(NoLTNedgefreq), max(NoLTNedgefreq)]);
% Normalising edge frequencies
normedgefreq = normalize(NoLTNedgefreq,"range");

for i = 1:length(NoLTNedgefreq)
        %Specifying the colours of each edge
        colorIndex = round(normedgefreq(i)*255)+1;
        %Applying the colours to the relevant edge
        highlight(Q2dhWithMidpoints, 'Edges',i, 'EdgeColor', cmap(colorIndex,:))
end
```

```
hold off;
```


9 98 18 115 27 132 36 149 45 166 54 183 63 200 72 217 81
89 106 123 140 157 174 191 208 225
8 97 17 114 26 131 35 148 44 165 53 182 62 199 71 216 80
88 105 122 139 156 173 190 207 224
7 96 16 113 25 130 34 147 43 164 52 181 61 198 70 215 79
87 104 121 138 155 172 189 206 223
6 95 15 112 24 129 33 146 42 163 51 180 60 197 69 214 78
86 103 120 137 154 171 188 205 222
5 94 14 111 23 128 32 145 41 162 50 179 59 196 68 213 77
85 102 119 136 153 170 187 204 221
4 93 13 110 22 127 31 144 40 161 49 178 58 195 67 212 76
84 101 118 135 152 169 186 203 220
3 92 12 109 21 126 30 143 39 160 48 177 57 194 66 211 75
83 100 117 134 151 168 185 202 219
2 91 11 108 20 125 29 142 38 159 47 176 56 193 65 210 74
82 99 116 133 150 167 184 201 218
1 90 10 107 19 124 28 141 37 158 46 175 55 192 64 209 73

Q3 - Assignment Modelling

(a) We analyse the parallel network with a single origin-destination pair and two links, shown above. Denote the link flows by x1, x2 ≥ 0 and prescribe total demand d, so that x1 + x2 = d. Per-user link costs c1(x1), c2(x2) are given by:

$$c_1(x_1) = 1 + \frac{1}{x_1},$$

$$c_2(x_2) = 3 + x_2.$$

Link 2 represents a standard congestible choice (e.g., car traffic) whose cost increases as it becomes more popular. However, link 1 represents a choice whose cost *decreases* as it becomes more popular — representing a public transport option where there is an economy of scale and more users enable a more frequent timetable and cheaper tickets.

(i) By considering link costs and a statement of Wardop's equilibrium, demonstrate that x1 = 0 and x2 = d is a user equilibrium (UE) assignment, and derive a condition on d for which x1 = d and x2 = 0 is also a UE assignment.

Wardrop's Principle (Equilibrium): [Also known as User Equilibrium]

↳ the costs of the used routes will be equal and less than or equal to the costs of the unused routes.

[Complementarity problem].

Note: routes ≡ links

① Demonstrating $x_1 = 0$ and $x_2 = d$ is a UE assignment.

when $x_1 = 0$ $\quad c_1(0) = 1 + \frac{1}{0} = \underline{\underline{\infty}}$ (infinity)

when $x_2 = d$ $\quad c_2(d) = 3 + d = \underline{\underline{3+d}}$

$\therefore c_1(0) \geq c_2(d)$ and so $x_1 = 0$ and $x_2 = d$ is a UE assignment

② Deriving a condition on d for which $x_1 = d$ and $x_2 = 0$ is also a UE assignment

when $x_1 = d$ $\quad c_1(d) = 1 + \frac{1}{d} = \frac{d+1}{d}$

when $x_2 = 0$ $\quad c_2(0) = 3 + 0 = 3$

$\therefore c_1(d) \leq c_2(0)$

From ① $\quad c_1(0) \geq c_2(d) \Rightarrow \infty \geq 3+d$ → can't use this

From ② $\quad c_1(d) \leq c_2(0) \Rightarrow \frac{d+1}{d} \geq 3$

$d + 1 \geq 3d$

$1 \geq 2d$

$\therefore d \leq 1/2$

(ii) Find an additional UE assignment where both links are used and their costs are equal, by solving an approriate quadratic for x1, and find the minimum demand d for which this assignment is valid (i.e., x1, x2 ≥ 0).

UE Assignment where both links are used and costs are equal.

Find minimum demand, d, for which this assignment is valid

$(x_1, x_2 \geq 0)$

$$C_1(x_1) = C_2(x_2) \Rightarrow 1 + \frac{1}{x_1} = 3 + x_2$$

$$\frac{1}{x_1} = 2 + x_2$$

$$1 = 2x_1 + x_2 x_1$$

reminder: $x_1 + x_2 = d$

$\therefore x_1 = d - x_2$

$x_2 = d - x_1$

Solving for x_1 $(x_2 = d - x_1)$

$1 = 2x_1 + x_1(d - x_1)$

$1 = 2x_1 + dx_1 - x_1^2$

$x_1^2 + -(2+d)x_1 + 1 = 0$

Subbing into:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where:

$a = 1$, $b = -(2+d)$, $c = 1$

$$x_1 = \frac{-(2+d) \pm \sqrt{(-(2+d))^2 - 4 \times 1 \times 1}}{2 \times 1}$$

$$= \frac{2+d \pm \sqrt{d^2 + 4d}}{2}$$

Solving for x_2 $(x_1 = d - x_2)$

$1 = 2(d - x_2) + x_2(d - x_2)$

$1 = 2d - 2x_2 + x_2 d - x_2^2$

$x_2^2 + 2x_2 - x_2 d - 2d + 1 = 0$

$x_2^2 + x_2(2 - d) + (1 - 2d) = 0$

Subbing into

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where:

$a = 1$, $b = (2 - d)$, $c = (1 - 2d)$

$$x_2 = \frac{-(2-d) \pm \sqrt{(2-d)^2 - 4 \times 1 \times (1-2d)}}{2 \times 1}$$

$$= \frac{-2 + d \pm \sqrt{d^2 - 4d + 4 - 4 + 8d}}{2}$$

Reminder : $x_1 \geq 0$ to be valid solution

$$\frac{2 + d \pm \sqrt{d^2+4d}}{2} \geq 0$$

$$\frac{d \pm \sqrt{d^2+4d}}{2} \geq -1$$

$$d \pm \sqrt{d^2+4d} \geq -2$$

For: $d + \sqrt{d^2+4d} \geq -2$

$$\sqrt{d^2+4d} \geq -2-d$$

$$d^2+4d \geq (-2-d)^2$$

$$d^2+4d \geq d^2+4d+4$$

$$0 \geq 4 \quad [\text{invalid}]$$

For: $d - \sqrt{d^2+4d} \geq -2$

$$2+d \geq \sqrt{d^2+4d}$$

$$(2+d)^2 \geq d^2+4d$$

$$4+4d+d^2 \geq d^2+4d$$

$$4 \geq 0 \quad [\text{Not useful!}]$$

Reminder $x_2 \geq 0$ to be a valid solution

$$\frac{-2 + d \pm \sqrt{d^2+4d}}{2} \geq 0$$

$$\frac{d \pm \sqrt{d^2+4d}}{2} \geq 1$$

$$d \pm \sqrt{d^2+4d} \geq 2$$

For: $d + \sqrt{d^2+4d} \geq 2$

$$\sqrt{d^2+4d} \geq 2-d$$

$$d^2+4d \geq (2-d)^2$$

$$d^2+4d \geq d^2-4d+4$$

$$8d \geq 4$$

[Valid solution] $d \geq 0.5$ (A)

For: $d - \sqrt{d^2+4d} \geq 2$

$$d-2 \geq \sqrt{d^2+4d}$$

$$(d-2)^2 \geq d^2+4d$$

$$d^2-4d+4 \geq d^2+4d$$

$$4 \geq 8d$$

[Valid solution] $d \leq 0.5$ (B)

Therefore using solution (A) $d \geq 0.5$ and (B) $d \leq 0.5$, d must be 0.5 to satisfy both solutions

$\boxed{d = 0.5}$

$C_1 = 1 + 1/x_1$

$C_2 = 3 + x_2$

sanity check using $d = 0.5$:

$x_1 = \frac{2+d \pm \sqrt{d^2+4d}}{2} = 2$ or 0.5 : invalid

$x_2 = \frac{-2+d \pm \sqrt{d^2+4d}}{2} = 0$ or -1.5 :

$C_1(2) = 1.5$

$C_1(0.5) = 3$

$C_2(0) = 3$

$C_2(-1.5) = 1.5$

→ Two pairs

∴ valid value of minimum d //

(iii) Find the total system cost for the assignments you found in parts (i) and (ii) and discuss and interpret your results in terms of the suggested application setting.

Discussion and interpretation of results

- Assignments 2 and 3 show the lowest total system cost of 1.5 while assignment 1 shows a higher total system cost of 1.75.
- Assignment 1 is the situation where all journeys are made with the standard congestible choice highlighting the inefficiency of individual traffic.
- While assignment 2 is the situation where all journeys are made using public transport and shows a lower total system cost of 0.25, highlighting how when there is high demand the cost of mass-transport enables more frequent timetables and cheaper tickets that benefits the majority.
- Assignment 3 on the other hand, where both the cost of the use of standard congestible and public transport are equal shows the same total system cost to all journeys being made by public transport. This suggests the optimal combination of system to minimise cost for the majority is a mixture of standard congestible and public transport but with a heavier emphasis on the use of public transport to make it economically efficient for most users.

Total System costs for assignments ①, ② and ③

$$f = x_1 c_1(x_1) + x_2 c_2(x_2)$$

① $x_1 = 0$ $\quad x_2 = d$

$c_1(0) = \infty$ $\quad c_2(d) = 3 + d$

$$f① = 0 \times \infty + d \times (3+d) = \underline{3d + d^2}$$

② $x_1 = d$ $\quad x_2 = 0$

$c_1(d) = 1 + \frac{1}{d} = \frac{d+1}{d}$ $\quad c_2(0) = 3$

$$f② = d \times \frac{d+1}{d} + 0 \times 3 = \underline{d+1}$$

③ $c_1(x_1) = c_2(x_2)$

using $x_1 = 0.5$ $\quad x_2 = 0$

$c_1(x_1) = 3$ $\quad c_2(x_2) = 3$

$$f③ = 0.5 \times 3 + 0 \times 3 = \underline{1.5}$$

invalid as $x_2 \geq 0$ condition

using $x_1 = 2$ $\quad x_2 = -1.5$

$c_1(x_1) = 1.5$ $\quad c_2(x_2) = 1.5$

$$f③ = 2 \times 1.5 - 1.5 \times 1.5 = 0.75$$

Summary:

$f① = 3d + d^2$ $\quad [x_1 = 0, x_2 = d]$

$f② = d + 1$ $\quad [x_1 = d, x_2 = 0]$

$f③ = 1.5$ $\quad [c_1(x_1) = c_2(x_2)]$ $\quad [x_1 = 0.5, x_2 = 0]$

Using Minimum demand $d = 0.5$:

$f① = \underline{1.75}$

$f② = \underline{1.5}$

$f③ = \underline{1.75}$

Reminder: c_1, x_1 in relation to public transport

c_2, x_2 in relation to congestible choice (e.g. car traffic)

i.e. $f①$ relates to car traffic only

$f②$ and $f③$ are for public transport only.

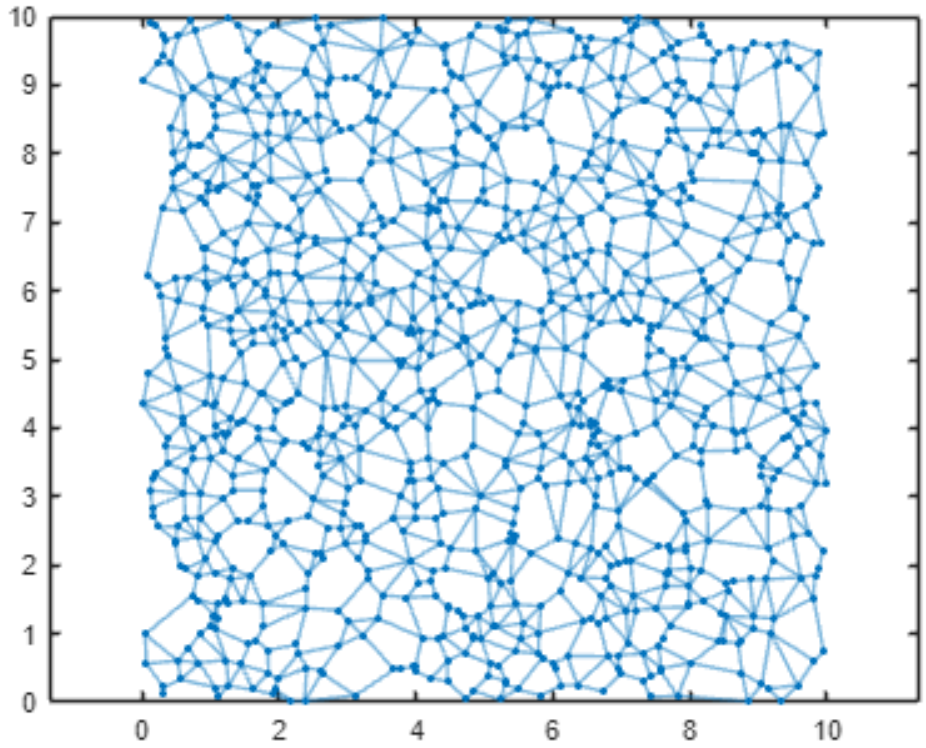
(b) We are now going to compute UE for large networks. Study and adapt the provided livescript TMMcourseworkQ3.mlx which loads in the network provided in BigNetworkExperimentsBetaSkeleton.mat and formulates it so that quadprog can be used to solve the UE problem.

Show graphically how the network 'fills up' as demand d is increased from 0 over an appropriate range that you should choose. You should certainly ensure that the peak Price of Anarchy (PoA) is captured — and this may require you to also compute the System Optimal (SO) assignment. Finally, for the UE assignment with a fixed value of d that you should choose, demonstrate that there are multiple routes with the same total cost.

I have done the following for this question:

1. Studied and adapted the provided livescript. Plotted the network provided.
2. Set values of demand I believe to be good values.
3. Solve the UE and SO problems for each value of demand, computing the system prices and PoA too.
4. Plotted the values of fUE and fSO alongside PoA.
5. Plotted the network and shown how it 'fills up' using colour coded links using normalised link flow values calculated using quadprog.
6. Demonstrated multiple routes have the same total cost for a fixed value of d=30 and plotted.

```
clear; close all
% Loading in required parameters
load BigNetworkExperimentsBetaSkeleton.mat

% Plotting the current graph (what is given in .mat file)
figure;
plot(G,'XData',nodeCoordinates(:,1),'YData',nodeCoordinates(:,2))
%,'NodeLabelMode','auto')
axis([0 L 0 L])
axis equal
```



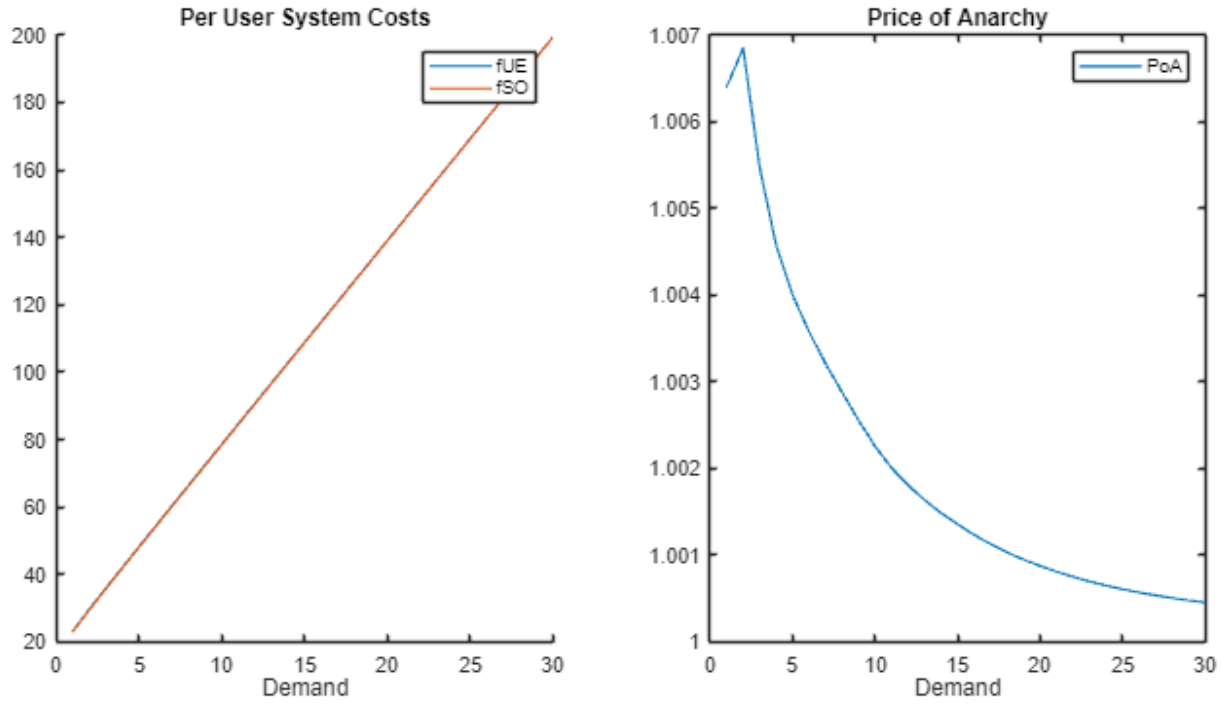

```
% Code provided
s = zeros(numNode,1);
s(oNode)=1;
s(dNode)=-1;

B=zeros(numNode,numEdge);
listEdge = 1:numEdge;
sNode = G.Edges.EndNodes(:,1);
tNode = G.Edges.EndNodes(:,2);

for i=1:numEdge
    B(sNode(i),listEdge(i))=1;
    B(tNode(i),listEdge(i))=-1;
end

Aeq=B;      % node conservation matrix
A=[];       % no inequality constraints
bvec=[];       % no inequality constraints - note avoiding name conflict with my
existing b
LB = zeros(numEdge,1); % set a lower bound of zero for all of the link flows.
UB=[]; % no upper bounds for the link flows - although you could use this facility
to set
% a capacity (ie max flow) for each link

H=diag(b); % the square matrix with the elements of b on its diagonal.
% Note this is what is needed for the Beckmann functional, because of the
```

```
% way matlab inserts a factor of 1/2 in front of the quadratic form.
% If we were doing SO, there would need to be a factor of 2 in here.

f=a; % the linear part of the quadratic form
```

```
% Setting the values of demand to iterate over and therefore investigate
dvals = 1:1:30;

% Setting up the required UE storage arrays
xUE = zeros(length(dvals),numEdge);
fUE = zeros(length(dvals),1);
fUEperuser = zeros(length(dvals),1);

% Setting up the required SO storage arrays
xSO = zeros(length(dvals),numEdge);
fSO = zeros(length(dvals),1);
fSOperuser = zeros(length(dvals),1);

% Setting up the required PoA storage array
PoA = zeros(length(dvals),1);

% Index for each array corresponding to value of d
index = 1;

Ops = optimoptions("quadprog","Display","off");
for d = dvals
    beq=d*s;   % node conservation vector

    % Computing UE link flows for network provided
    [xUE(index,:),~]=quadprog(H,f,A,bvec,Aeq,beq,LB,UB,[],Ops);


    % compute total (system) cost
    fUE(index,:) = a'*xUE(index,:)' + xUE(index,:)*H*xUE(index,:)';
    % compute total (system) cost per user:
    fUEperuser(index) = fUE(index)/d;

    % Computing SO link flows for network provided
    % as for UE solution in route variables - but double the Hessian.
    [xSO(index,:),~]=quadprog(2*H,f,A,bvec,Aeq,beq,LB,UB,[],Ops);

    % compute total (system) cost
    fSO(index,:) = a'*xSO(index,:)' + xSO(index,:)*H*xSO(index,:)';
    % compute total (system) cost per user
    fSOperuser(index) = fSO(index,:)/d;

    % Compute price of anarchy using total system costs per user
    PoA(index) = (fUEperuser(index))/(fSOperuser(index));
```

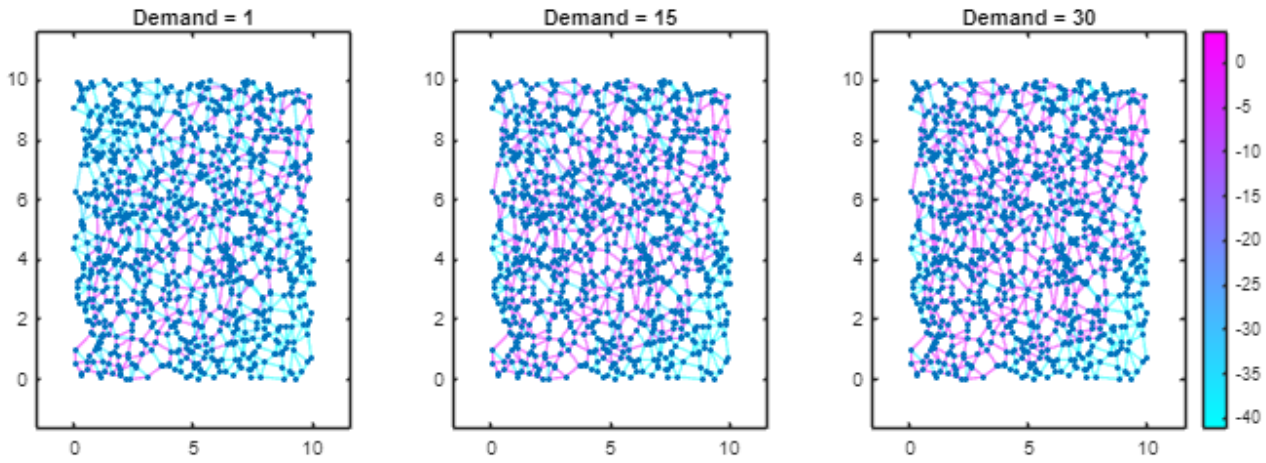
```
    % For arrays to save numbers for each value of d correctly
    index = index + 1;
end
```

```
% Figure of per user system costs and PoA
figure;

% Plotting per user system costs
subplot(1,2,1)
hold on
plot(dvals,fUEperuser,'LineWidth',1)
plot(dvals,fSOperuser,'LineWidth',1)
legend('fUE','fSO')
xlabel('Demand')
title('Per User System Costs')
hold off

% Plotting PoA
subplot(1,2,2)
plot(dvals,PoA,'LineWidth',1)
legend('PoA')
xlabel('Demand')
title('Price of Anarchy')

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,8])
```

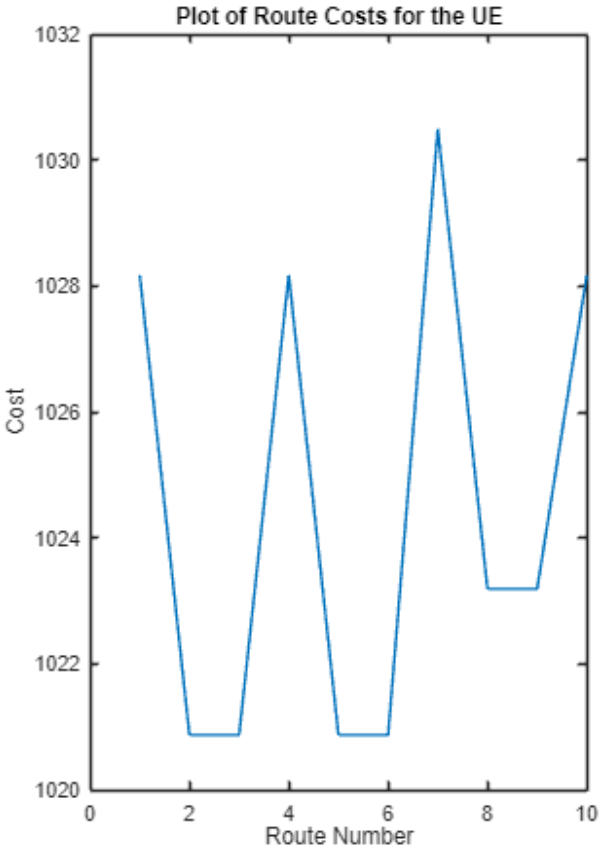


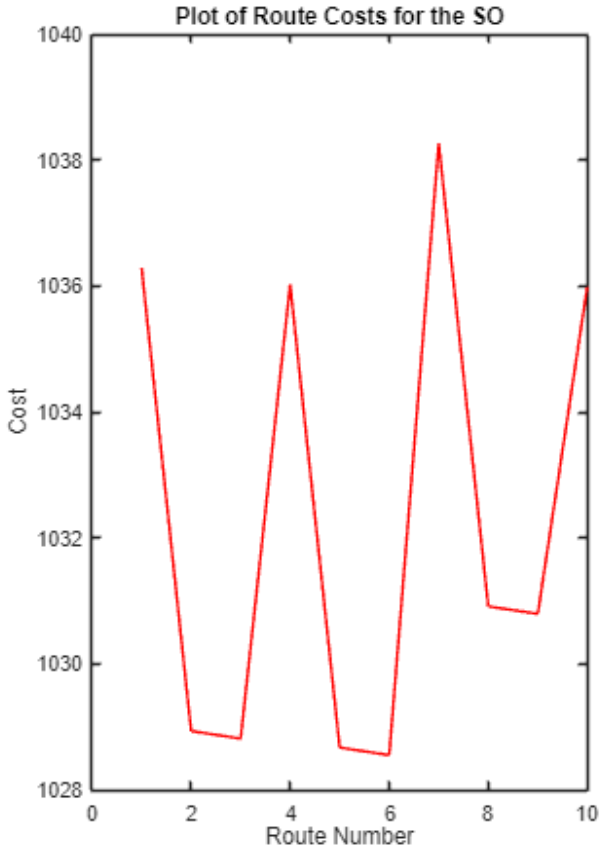

```
clf
```

```
% Detailing the network filling up as demand increases
figure;
tiledlayout(1,3)

% Plotting for first value of d
nexttile
p1 = plot(G,'XData',nodeCoordinates(:,1),'YData',nodeCoordinates(:,2));
% Setting title with correct d
title("Demand = "+string(dvals(1)))
% Colouring the edges using link flows
p1.EdgeCData = log(xUE(1,:)); % Using logs to show this more clearly

% Plotting for middle value of d
nexttile
p2 = plot(G,'XData',nodeCoordinates(:,1),'YData',nodeCoordinates(:,2));
% Setting title with correct d
title("Demand = "+string(dvals(15)))
% Colouring the edges using link flows
p2.EdgeCData = log(xUE(15,:)); % Using logs to show this more clearly

% Plotting for last value of d
nexttile
p3 = plot(G,'XData',nodeCoordinates(:,1),'YData',nodeCoordinates(:,2));
% Setting title with correct d
title("Demand = "+string(dvals(30)))
% Colouring the edges using link flows
p3.EdgeCData = log(xUE(30,:)); % Using logs to show this more clearly

% Colourmap and colourbar
colormap(cool);
colorbar;

% Specifying figure size
set(gcf,'units','inches','position',[0,0,15,5])
```



```
% Computing the route costs to show multiple routes have the same cost and
% plotting as a histogram for d = 30

% Obtaining 10 different paths
[nodepaths, edgepaths] = allpaths(G,oNode,dNode,'MaxNumPaths',10);

% compute link costs (could make an assumption routes=links???)
cUE = a + b.*xUE(30,:)';

% compute link costs
cSO = a + b.*xSO(30,:)';

% compute route costs
rUE = zeros(length(edgepaths),1);
rSO = zeros(length(edgepaths),1);

% Computing route costs
for i=1:length(edgepaths)
    edges = edgepaths(i);
    for j=1:length(edges{1})
        rUE(i) = rUE(i) + cUE(edges{1}(j));
        rSO(i) = rSO(i) + cSO(edges{1}(j));
    end
end

% Showing mutliple routes have the same cost for the UE by plotting a line graph
figure;
subplot(1,2,1);
plot(rUE)
% Making plot pretty
xlabel("Route Number")
ylabel("Cost")
title("Plot of Route Costs for the UE")

% Showing mutliple routes have the same cost for the SO by plotting a line graph
subplot(1,2,2);
plot(rSO,'Color','r')
% Making plot pretty
xlabel("Route Number")
ylabel("Cost")
title("Plot of Route Costs for the SO")

% Specifying figure size
set(gcf,'units','inches','position',[0,0,16,10])
```


Plot of Route Costs for the UE
Cost
Route Number
1032
1030
1028
1026
1024
1022
1020
0
2
4
6
8
10


Plot of Route Costs for the SO
Cost
Route Number
1040
1038
1036
1034
1032
1030
1028
0
2
4
6
8
10

Q4 - Microscopic Modelling

(a) The Optimal Velocity Model, is specified in eqs. 1 and 2 – referred to as Model specification 1 from now on:

$$\frac{\mathrm{d}v_\alpha}{\mathrm{d}t} = \frac{v'_e(d_\alpha) - v_\alpha}{\tau} \tag{1}$$

$$v'_\alpha(d_\alpha) = \frac{v_0}{2}\Big[\tanh(d_\alpha - d_c) + \tanh(d_c)\Big] \tag{2}$$

Consider an alternative specification – referred to as Model specification 2, specified by eqs. 3 – 5:

$$\frac{\mathrm{d}v_\alpha}{\mathrm{d}t} = \frac{v'_e(d_\alpha, v_\alpha) - v_\alpha}{\tau} \tag{3}$$

$$v'_\alpha(d_\alpha, v_\alpha) = \frac{v_0}{2} f(d_\alpha, v_\alpha) \tag{4}$$

$$f(d_\alpha, v_\alpha) = \begin{cases} \tanh(\frac{d_\alpha}{v_\alpha + \epsilon} - 1) + \tanh(1), & \text{if } d_\alpha > 10\text{m} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where ϵ is an infinitesimally small positive number.

(i) Analyse this new model specification and explain what the behavioural interpretation of the model is.

The basis of both of these models is to adjust the vehicles velocity based upon the vehicle in-front. The new model specification specifically only accelerates if the actual distance to the vehicle in front is greater than 10m compared to the Optimal Velocity Model (OVM) which accelerates and decelerates primarily based upon an expected distance. The OVM also is controlled by a speed limit while the alternative specification continues to accelerate with no indication of a speed limit.

(ii) Come up with two specific examples (either specific modelling scenarios or specific properties of the models), where one of the two model specifications is more realistic than the other one.

- The OVM is more realistic when placed in steady state traffic i.e. when travelling in a speed limited zone where everyone accelerates until the same speed is achieved.
- The alternate specification is more realistic at modelling traffic shock-waves i.e. moving off after a temporary stoppage and the impact this has on vehicles with backwards propagation.

(iii) Compare model specifications 1 and 2; explain what the differences and similarities in model dynamics is for the following three cases:

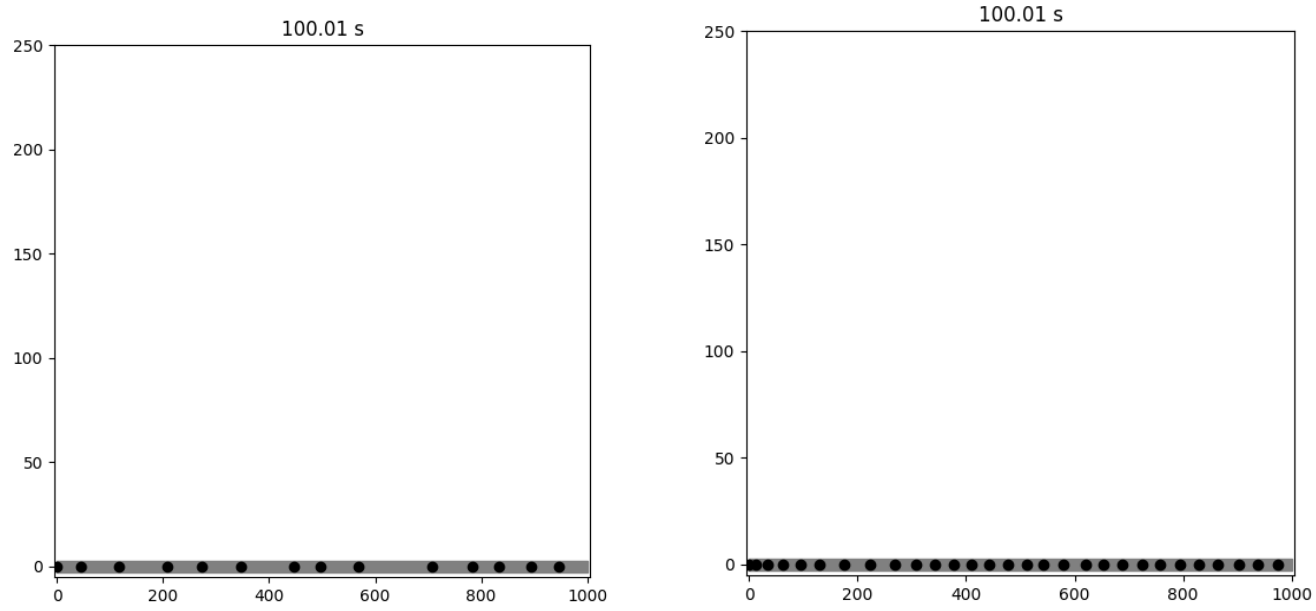
1. Steady state traffic (i.e., free flowing traffic with no particular perturbations or obstructions).

2. A queue of cars at a traffic light that has just turned green (assume that cars are spaced at 5 metres initially (net gap, bumper to bumper).

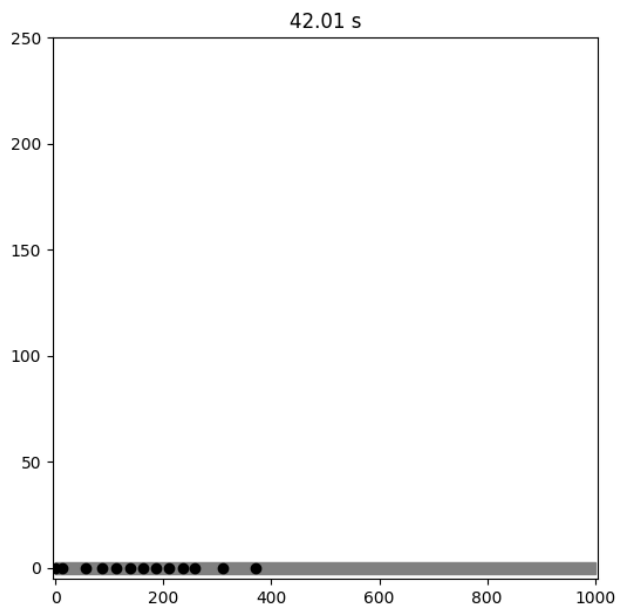
3. A sudden crash of one of the cars. The car has crashed into a stationary object and has reduced its speed to zero in approximately zero seconds.

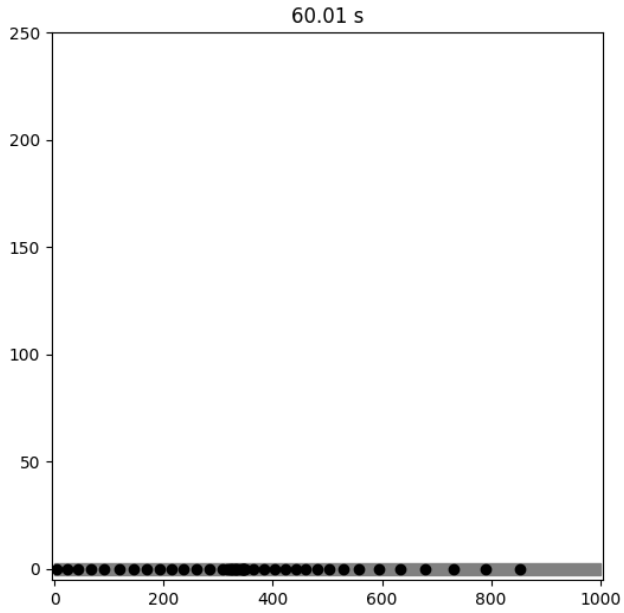
(OVM simulation is shown on the left while alternate model simulation is shown on the right)

1. The OVM shows linear, evenly-spaced traffic flow that travels often at the maximum speed and only accelerates when the space ahead is available. The alternate specification on the other hand shows congested traffic often with vehicles not able to accelerate often and not achieving their maximum speed.



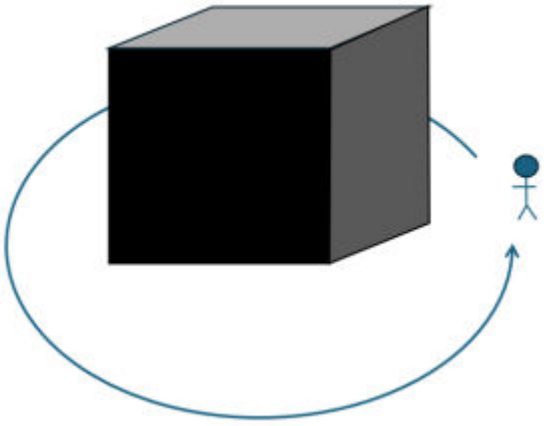
2. In the OVM, cars queue evenly while in the alternate model they do not. Therefore when the traffic light turns green in the OVM cars move off one-by-one with some gap between them and the car in front while in the alternate model specification all cars move off as soon as they can.





3. In this scenario it is understood that the cars in the both would slow down and stop as soon as they are close enough to the car in-front to be impacted. The vehicles in the OVM do this in a way that appears in terms of cautious, sensible driving i.e. slowing down well before the hazard ahead is met positionally. The alternate model on the other hand appears to do this in a less cautious way, only slowing down when absolutely essential to not cause a crash themselves.

(b) Consider the pilgrimage to Mecca where each pilgrim will walk 7 times around a central cubic building called Kaaba.



To simulate this scenario, write down a modified set of Social Force Model equations and add the following behaviours:

• The walking direction of each person should be circular, around a central point, in a counter-clockwise direction.

• If the local density around a pedestrian exceeds 6m^−2 (measured within a 5 metre radius around that person), an additional social force will act upon that pedestrian away from the central point, in order to seek out a lower density environment.

• When a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre.

General form of the social force model equations:

$$\frac{d\vec{v}_\alpha(t)}{dt} = \vec{f}_\alpha(t) + \vec{\xi}_\alpha(t)$$

Noise term

Where:

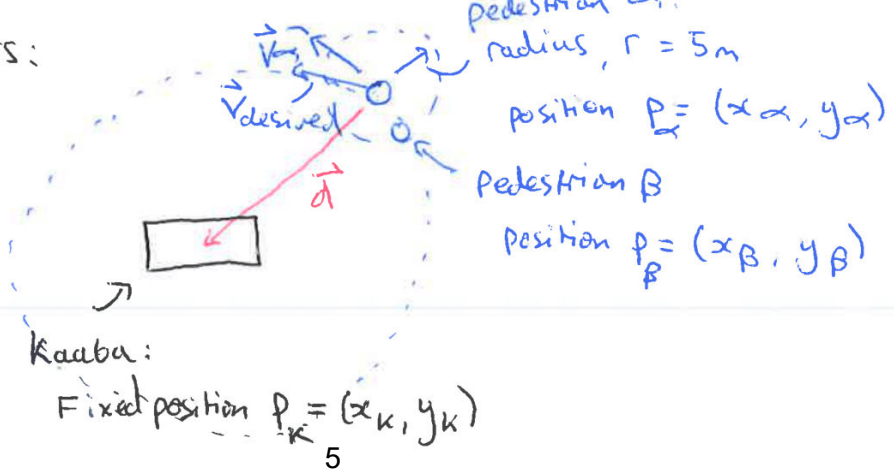
$$\vec{f}_\alpha(t) = \frac{1}{\tau_\alpha}\left(v_\alpha^0 \vec{e}_\alpha - \vec{v}_\alpha\right) + \sum_{\beta(\neq\alpha)} \vec{f}_{\alpha\beta}(t) + \sum_i \vec{f}_{\alpha i}(t)$$

- τ_α: Acceleration time
- $v_\alpha^0 \vec{e}_\alpha$: Desired velocity
- $\vec{v}_\alpha$: Actual velocity
- $\sum_{\beta(\neq\alpha)} \vec{f}_{\alpha\beta}(t)$: Forces from all other pedestrians β
- $\sum_i \vec{f}_{\alpha i}(t)$: Forces from all boundaries

Following behaviours need to be added:

1. The walking direction of each person should be circular around a central point, in a counter clockwise direction.
2. If the local density around a pedestrian exceeds $6 m^{-2}$ (measured with a 5-meter radius around that person), an additional social force will act upon that pedestrian away from the central point, in order to seek out a lower density environment
3. When a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre.

Diagram of behaviours:



For behaviours ① and ③ the desired velocity needs to be modified:

$$\vec{v}_{desired} = v^{\circ}_{\alpha}\, \vec{e}_{\alpha}$$

v°_{α} is a constant and so isn't changed. Therefore $\vec{e}_{\alpha}$ must be changed. This is known as the pedestrian α's desired direction and is defined as follows to simulate behaviours ① and ③:

$$\vec{e}_{\alpha} = \begin{cases} \dfrac{(d_j, -d_i)}{\|\vec{d}\|} & D_{\alpha} < D_{\text{seven Laps},\alpha} \\ \dfrac{\vec{v}_{\alpha}}{\|\vec{v}_{\alpha}\|} & \text{Otherwise} \end{cases}$$

Where:

$$\vec{d} = (d_i, d_j) = \frac{P_k - P_{\alpha}}{\|P_k - P_{\alpha}\|}$$

And:

D_{α} = Distance covered by pedestrian (Total over simulation) $= \sum_{i=1}^{t} v^{i}_{\alpha}(t)\, dt$

$D_{\text{seven Laps},\alpha}$ = Distance to be covered for seven laps to be complete (for each pedestrian α) $= 7(2\pi \|\vec{d}\|)$

For behaviour ②, the noise term should be edited to be the following:

$$\vec{\xi}_\alpha(t) = \begin{cases} \dfrac{(-di, -dj)}{\|\vec{d}\|} & \rho^{\alpha}_{5\text{-metre}} > 6 \\ 0 & \text{otherwise} \end{cases}$$

where $\rho^{\alpha}_{5\text{-metre}}$ is the local density in a 5-metre radius around pedestrian α. This is calculated using the following equation:

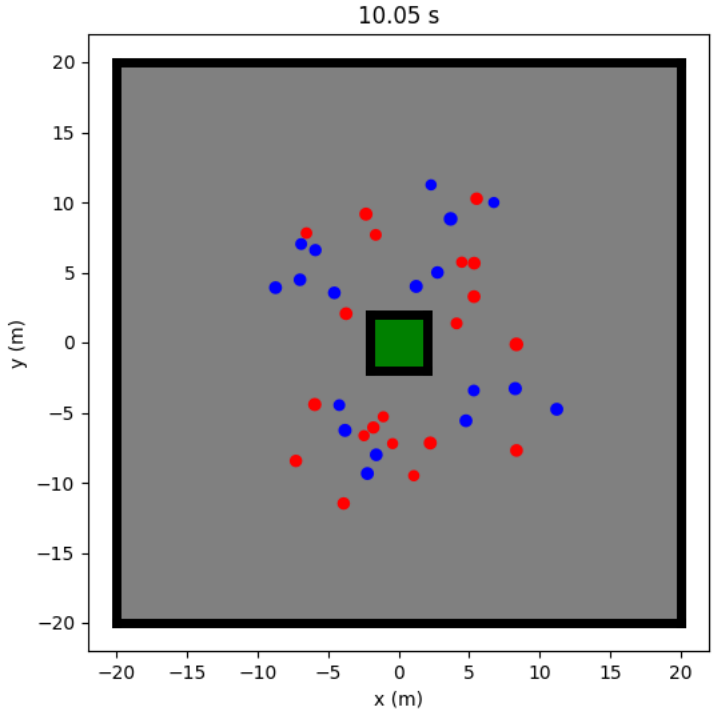
$$\rho^{\alpha}_{5\text{-metre}} = \sum_{(\beta \neq \alpha)} \rho_{\alpha\beta}$$

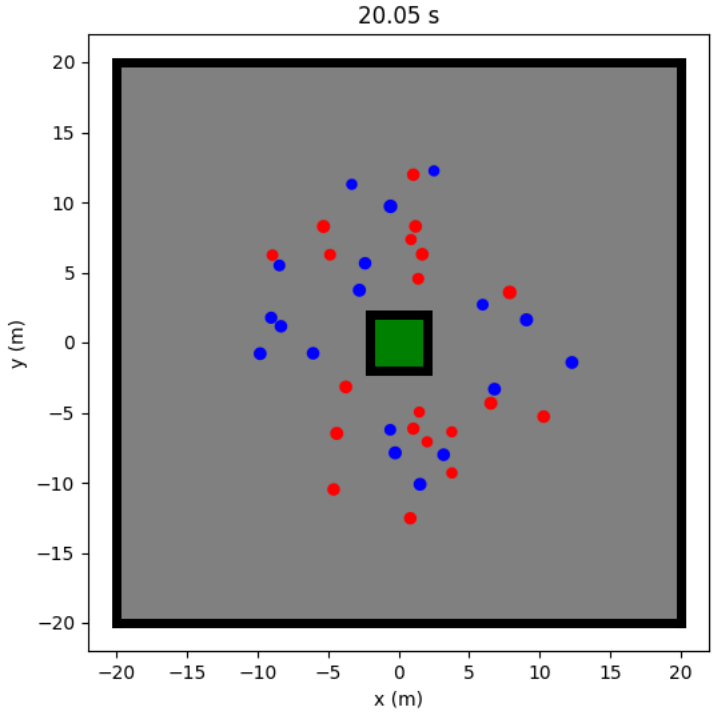
where:

$$\rho_{\alpha\beta} = \begin{cases} 1 & \sqrt{(x_\alpha - x_\beta)^2 + (y_\alpha - y_\beta)^2} < 5 \\ 0 & \text{otherwise} \end{cases}$$

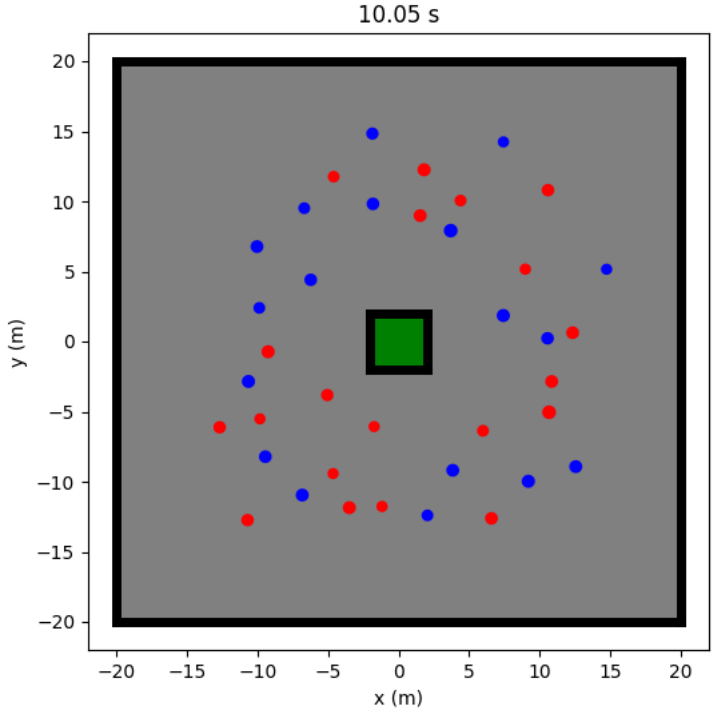
An example of an implementation of behaviors 1 and 3 using python code provided is shown below. Unfortunately condition 2 was not able to be applied due to time-constraints.

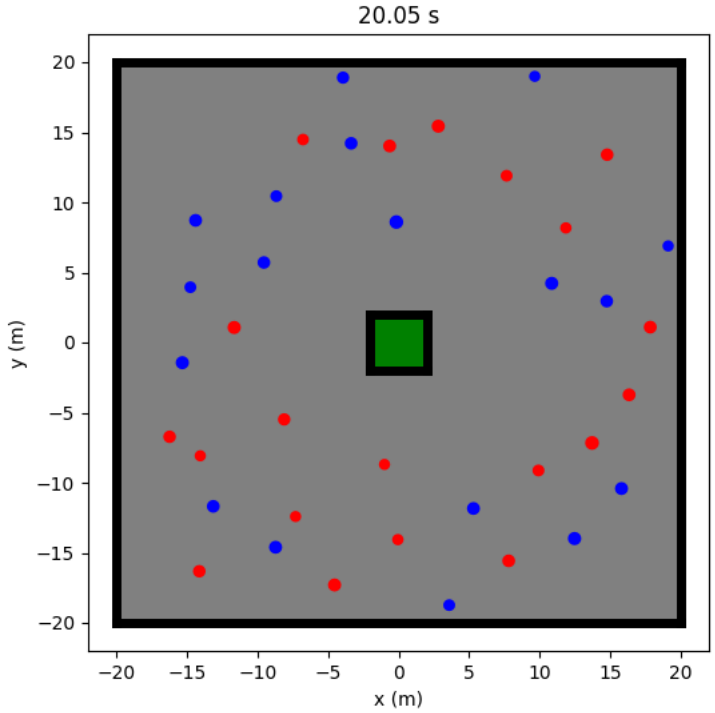
Pictures 1 & 2: The walking direction of each person should be circular, around a central point, in a counter-clockwise direction (i.e. condition 1):





Pictures 3 & 4: when a pedestrian has completed the seven laps, the person will change their walking direction to be tangentially away from the centre (i.e. condition 3):





```python
def update_directions(pedestrians, boundaries, polygons):
    for i, ped in pedestrians.items():
        # THIS IS WHAT NEEDS TO BE CHANGED!!!
        destination_polygon_centroid = polygons[ped.destination]["nodes"].mean(axis=0)
        if ped.distance_covered < ped.r:
            print(ped.r)
            v = normalise_vector(destination_polygon_centroid - ped.pos)
            tangv = [v[1],-v[0]]
            ped.desired_direction = normalise_vector(tangv)
        else:
            ped.desired_direction = normalise_vector(ped.vel)
```

```python
# Define modelling scenario
MosqueWidth = 40
MosqueHeight = 40

KaabaWidth = 4
KaabaHeight = 4
world_definition = {
    "space": {
        "InnerMosque": {"type": "rectangle", "coordinates": [-MosqueWidth/4, -MosqueHeight/4, MosqueWidth/4, MosqueHeight/4], "colour": "gray", "add_boundaries": False},
        "OuterMosque": {"type": "rectangle", "coordinates": [-MosqueWidth/2, -MosqueHeight/2, MosqueWidth/2, MosqueHeight/2], "colour": "gray", "add_boundaries": True},
        "Kaaba": {"type": "rectangle", "coordinates": [-(KaabaWidth/2), -(KaabaHeight/2), (KaabaWidth/2), (KaabaHeight/2)], "colour": "green", "add_boundaries": True},
    },
    "pedestrians": {
        "group1": {"source": "InnerMosque", "destination": "Kaaba", "colour": "red", "birth_rate": 20, "max_count": 20},
        "group2": {"source": "InnerMosque", "destination": "Kaaba", "colour": "blue", "birth_rate": 20, "max_count": 20},
    },
    "boundaries": [[-(KaabaWidth/2), -(KaabaHeight/2), +(KaabaWidth/2), -(KaabaHeight/2)],
    [-(KaabaWidth/2), -(KaabaHeight/2), -(KaabaWidth/2), +(KaabaHeight/2)],
    [-(KaabaWidth/2), +(KaabaHeight/2), +(KaabaWidth/2), +(KaabaHeight/2)],
    [+(KaabaWidth/2), -(KaabaHeight/2), +(KaabaWidth/2), +(KaabaHeight/2)]],
    #"periodic_boundaries": {"axis": "x", "pos1": 0, "pos2": 0},
    "functions": {
        "update_directions": update_directions,
        "process_interactions": process_interactions,
        "pedestrian_initialisation": pedestrian_initialisation
    }
}
```

```python
    def update_positions(self, dt=0.05):
        # Update velocities and positions
        self.time += dt
        for i, ped_i in self.pedestrians.items():
            if vector_length(ped_i.acc)>10:
                ped_i.acc = 10*normalise_vector(ped_i.acc)
            ped_i.vel += dt*ped_i.acc
            ped_i.pos += dt*ped_i.vel
            ped_i.distance_covered += vector_length(dt*ped_i.vel)
        # Apply periodic boundary conditions
        if self.periodic_boundaries != None:
            if self.periodic_boundaries["axis"] == "x":
```

Question 5 - Data-Driven Modelling

For all parts of the question, provide your discussion in bullet points.

(a) Consider the following time-series model for the count of vehicles, Y (t), on a road in Bristol. One unit of time is 12 hours, so at t = 0, the total count for 12 hours (7am-7pm) is given, and at t = 1 the count for the following 12 hours is given. Counts are given in 1000s of vehicles, so Y (t) = 5 means 5, 000 vehicles were counted.

$$
\begin{aligned}
Y(t) &= a(t) + b(t) + c(t) + d(t), \\
a(t) &= \alpha a(t-1), \\
b(t) &= \beta Y(t-1), \\
c(t) &= \sum_{j=1}^{s/2} cos\left(\frac{2\pi j}{s} c(t-1)\right), \\
d(t) &\sim Normal(0, \sigma).
\end{aligned}
$$

The parameters of the model have been estimated to take the following values, based on observational data: α = 0.99, β = 0.5, s = 4, and σ = 0.1

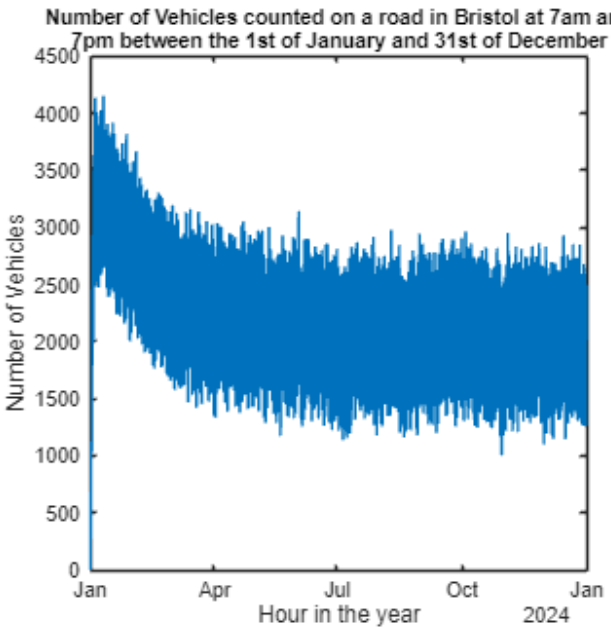
```
% Parameters estimted based on observational data
alpha = 0.99;
beta = 0.5;
s = 4;
sigma = 0.1;
```

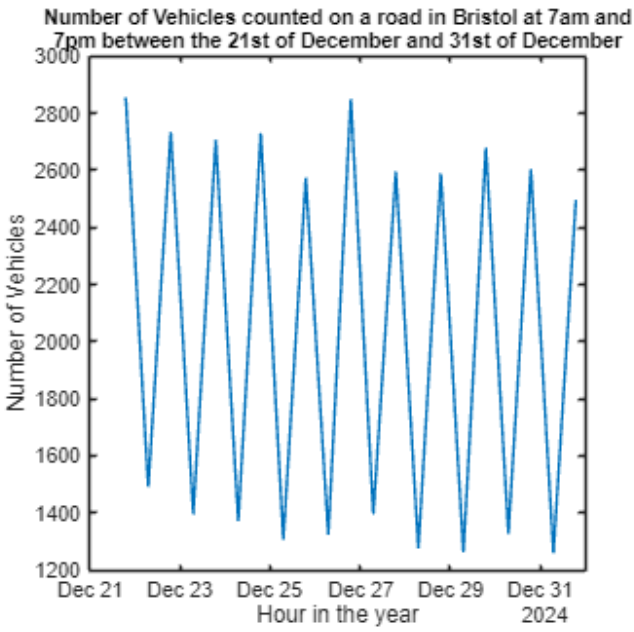
(i) Describe the dynamics produced by this model for the initial condition Y (0) = 0, a(0) = 0.75, c(0) = 0.5. Include a sketch/plot in your answer.

```
% The number of uses of the equation necessary to produce a full year's
% worth of data - inclusive
CyclesInYear = 365*2+2;

% Setting up the necessary arrays for time series model for count of
% vehicles
Y = zeros(CyclesInYear,1);
a = zeros(CyclesInYear,1);
```

```
b = zeros(CyclesInYear,1);
c = zeros(CyclesInYear,1);
Norm = makedist('Normal','mu',0,'sigma',sigma);
d = zeros(CyclesInYear,1);

% Setting the initial conditions
Y(1) = 0;
a(1) = 0.75;
c(1) = 0.5;
d(1) = random(Norm);
b(1) = Y(1) - a(1) - c(1) - d(1);

% Calculation of time series model predictions of vehicle count
for t=2:CyclesInYear
    % Equations used to calculate time series model
    a(t) = alpha*a(t-1);
    b(t) = beta*Y(t-1);

    for i=1:s/2
       c(t) = c(t) + cos((2*pi*i/s)*c(t-1));
    end
    d(t) = random(Norm);

    % Calculation of time series model prediction
    Y(t) = a(t) + b(t) + c(t) + d(t);
end

StartDate = datetime(2024,1,1, 7, 0, 0);
EndDate = datetime(2024,12,31, 19, 0, 0);
X = StartDate:hours(12):EndDate;

% Setting up figure
figure;

% Plotting all values obtained by the time-series model
ax1 = subplot(1,2,1);
plot(X,Y*1000)
xlabel("Hour in the year")
ylabel("Number of Vehicles")
title(sprintf("Number of Vehicles counted on a road in Bristol at 7am and" + ...
    "\n7pm between the 1st of January and 31st of December"),"fontsize",10)

% Plotting the final 20 values obtained by the time-series model
ax2 = subplot(1,2,2);
plot(X(CyclesInYear-20:CyclesInYear),Y(CyclesInYear-20:CyclesInYear)*1000)
xlabel("Hour in the year")
ylabel("Number of Vehicles")
title(sprintf("Number of Vehicles counted on a road in Bristol at 7am and" + ...
    "\n7pm between the 21st of December and 31st of December"),"fontsize",10)
```

```
set(gcf,'units','inches','position',[0,0,14,6])
```





The dynamics in this model are as follows:

- Average around 2100 vehicles in a 24 hour period, with typical deviation of around 500. This gives an approximate traffic flow of 175 vehicles per hour over a day, with approximately 233 vehicles per hour during the day (7am to 7pm) and 116 at night (7pm to 7am). This is approximately one vehicle every 15 seconds in the day and 31 seconds at night.
- a is a steadily decreasing positive value that follows an approximately exponential curve while b is much more fluctuating with b staying positive (apart from the initial values) between 0.65 and 1.45 for night and day respectively. c has the values of 0 and 2 after the initial spike, occupying the value of 2 in the day and 0 at night, suggesting the approximate 2000 vehicle difference between night and day. d is a random noise term that doesn't follow a trend.
- The model settles after an initial spike caused by the initial condition of Y(0) = 0 but there are still some large peaks and troughs at seemingly random intervals
- There is no distinction between weekdays and weekends or seasons (i.e. decreased counts during school/university holidays) and it assumes there is no traffic lights, queues or road works.

(ii) Discuss the usefulness of this model with the given parameter estimates for forecasting and explaining the dynamics of vehicle traffic counts on the road in Bristol, clearly justifying the arguments you make.

Positives:

- The model is useful for predicting traffic flow over a long time period i.e. a month or a year. It is most likely quite accurate for these time frames as any outliers are likely to be average out over this time frame.This is useful when looking at a flow rate over a whole year or such to see what traffic is like on the macro-scale.

- After the initial period, the model settles to approximate steady values over 12 hours, highlighting the much lower traffic count at night compared to day. This is also useful as it means a day-by-day situation of the model is possible to obtain and is accurate to a degree.

Drawbacks:

- This model uses very low "sampling" i.e. one sample every 12 hours so, for example, if a traffic model of this part of the network was desired to be on an hour-by-hour or minute-by-minute scale it is going to be inaccurate. This is simply because interpolating between the peak and trough over the 24-hour period is too crude a method to use. Fitting these two points to a curve could be a method around this but is heavily dependent on the curve selected.
- As previously stated the model doesn't take into account seasonal variations. This is significant as Bristol, a student and young person heavy city, is often busiest when term-time is around i.e. from the months September through to May. This isn't reflected in the model at all and just assumes that traffic flow remains the same all year round. There is a similar impact for if a road is placed near a school.
- There are also more variations to do with type of use as for example if the road is right next to a school, the traffic flow is likely to be heavily dependent on school timetable and demographic of students/parents too. In effect, if the school is located close to the road and is private or placed in a wealthy area the traffic flow is likely to be higher.
- The model also doesn't take into account traffic lights, queues or road works that could impact traffic flow on this road. This could once again have significant impact on hour-by-hour and minute-by-minute approximations from the data but is unlikely to affect the model at a larger scale. This once again limits the model to longer rather than shorter time frames to sample out of.

(b) This questions extends the worked direct demand modelling example on pedestrian traffic data from Melbourne covered in the lab sheet for data-driven modelling.

Guideline to allocation of marks for part (b):

• Simple answers build on the code and data provided in the lab sheet.

• Good answers consider additional data.

• Excellent answers start from scratch, obtaining all data from primary sources.

• Use a similar spatial and temporal coverage as shown in the lab sheet example, there are no extra marks for considering larger areas or longer time-frames.

(i) Develop and implement in code (Python or Matlab) a regression-based direct demand model for central Melbourne pedestrian traffic to forecast traffic on all links of part of the network 2 hours in advance. Provide a clear explanation and rationale for the model you choose, and for any additional data you obtain.

For this question, I obtained two sets of additional data: data from the Melbourne traffic data website for the sensors for June, July and August from 2019 to 2024 and a set of COVID coefficients i.e. how much in Melbourne the pedestrian traffic decreased in Melbourne during those times obtained from these two sources

(90% reduction in 2020 ATRF2021_Resubmission_40-1.pdf and 30% in 2022 Has CBD pedestrian activity in Melbourne recovered since Covid-19? - RMIT University, a mean of this was taken for 2021 i.e. 60% reduction). The rationale behind this was to give the model more data to train on and see what the impact of this extra data and approximation of COVID impact had compared to the lab sheet results obtained with a smaller sample size.

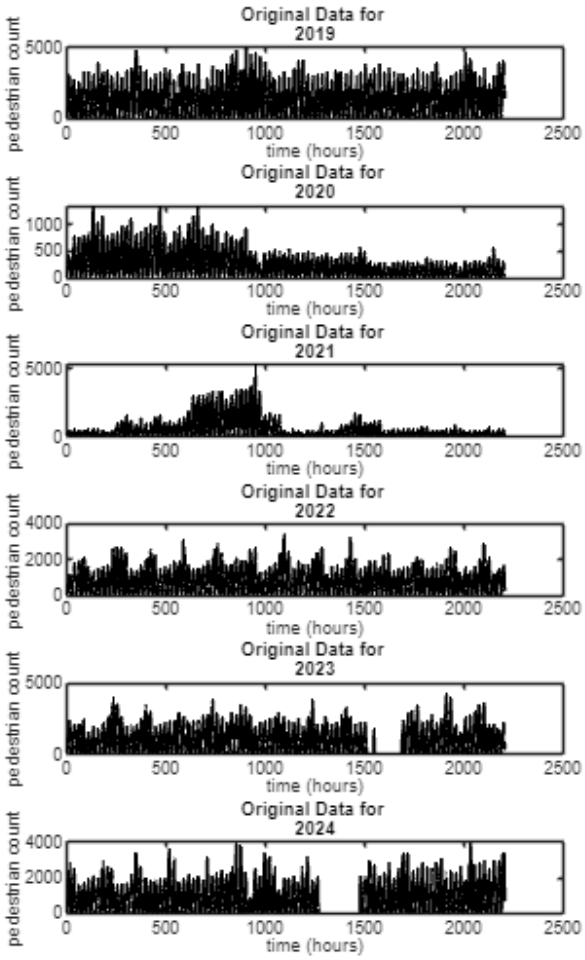
Following on from this, I chose the Seasonal ARIMA model (SARIMA) as the first part of my regression-based direct demand model. This model was chosen as it was deemed to be able to handle incomplete data sets appropriately and produce more accurate results than the ARIMA model due to the seasonality being set to a 24 hour period. The LSTM was inappropriate here as it was researched and appeared to not handle missing data points well.

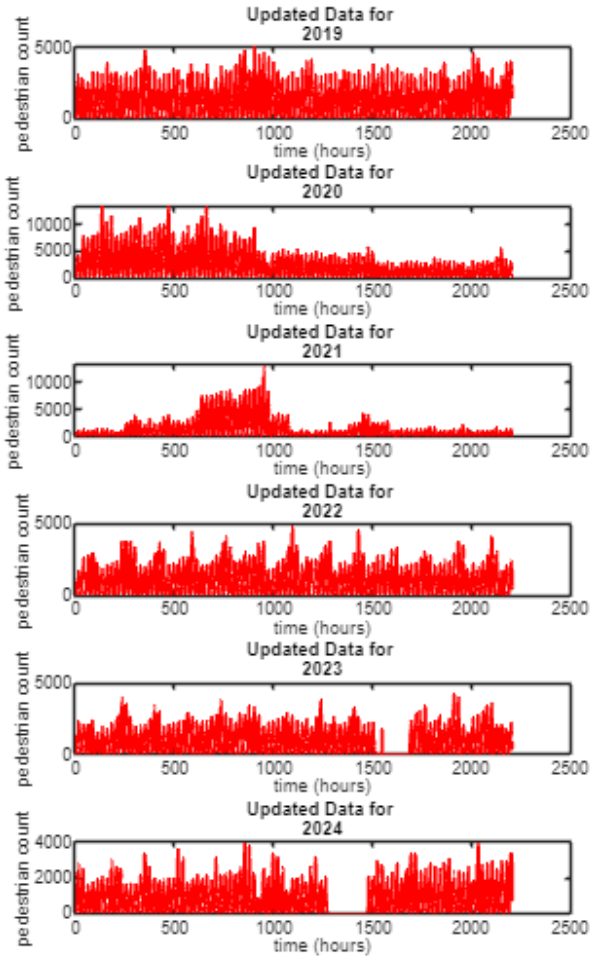
The second part of my regression based direct demand model was a gravity model where the impact of each sensor on the link is directly proportional to 1/(distance between sensor and link)^2. This is thought to be a relatively good estimate for the impact of links on other links due to the wide use of the gravity model in other parts of transport and mobility modelling, especially demand modelling. This was how the traffic flow on each link was predicted for the network 2 hours in advance.

```
% read in data from pedestrian counters
% hourly counts for 13 counters for 52 days in June and July 2019.

opts = delimitedTextImportOptions("NumVariables", 102);

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

SensorNames = ["MelbourneCentral", "LonsdaleSt-ElizabethSt(North)",...
"ElizabethSt-LonsdaleSt(South)","MelbourneCentral-
ElizabethSt(East)","StateLibrary",...
"LonsdaleSt(South)","Chinatown-SwanstonSt(North)","Chinatown-LtBourkeSt(South)",...
"BourkeSt-RussellSt(West)","BourkeStreetMall(North)","BourkeStreetMall(South)",...
"LittleCollinsSt-SwanstonSt(East)","TownHall(West)"];

% Specify column names and types
opts.VariableNames = ["Date", "Hour", SensorNames];
opts.SelectedVariableNames = ["Date", "Hour", SensorNames];
opts.VariableTypes = ["datetime", "double", "double", "double", "double", "double",
"double", "double",...
    "double", "double", "double", "double", "double", "double", "double"];

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Specify variable properties
opts = setvaropts(opts, "Date", "InputFormat", "dd/MM/yyyy", "DatetimeFormat",
"preserveinput");
opts.VariableNamingRule = 'preserve';  % This will preserve the original column
names
```

```
% Setting years of data to collect
StartYear = 2019;
EndYear = 2024;

% Setting up data obtaining arrays
AllData = cell(length(StartYear:EndYear),3);
dates = [];
OriginalData = [];
UpdatedData = [];

for i=StartYear:EndYear
    % File path names
    JunePath = strcat("..\Q5 - Data Driven
Modelling\Data\","June_",string(i),".csv");
    JulyPath =  strcat("..\Q5 - Data Driven
Modelling\Data\","July_",string(i),".csv");
    AugustPath =  strcat("..\Q5 - Data Driven
Modelling\Data\","August_",string(i),".csv");
    % Extracting data in tables
    JuneTab = readtable(JunePath, opts);
    JulyTab = readtable(JulyPath, opts);
    AugustTab = readtable(AugustPath, opts);
    % Putting data into appropriate parts of cell arrays
    AllData{(i-(StartYear-1)),1} = JuneTab(:,:);
    AllData{(i-(StartYear-1)),2} = JulyTab(:,:);
    AllData{(i-(StartYear-1)),3} = AugustTab(:,:);

    % Changing the data values based on COVID coefficients and saving
    % appropriately
    dates = [dates; table2array(JuneTab(:,1)); table2array(JulyTab(:,1));
table2array(AugustTab(:,1))];
    OriginalData = [OriginalData; table2array(JuneTab(:,3:15));
table2array(JulyTab(:,3:15)); table2array(AugustTab(:,3:15))];
    % 2020 - 90% reduction value  i.e. 10% of 2019 figures
    if i == 2020
        UpdatedData = [UpdatedData; table2array(JuneTab(:,3:15))*(100/10);
table2array(JulyTab(:,3:15))*(100/10); table2array(AugustTab(:,3:15))*(100/10)];
        %2021 - middle between the two values about and below: 60%
        %reduction i.e. 40% of 2019 figures
    elseif i == 2021
        UpdatedData = [UpdatedData; table2array(JuneTab(:,3:15))*(100/40);
table2array(JulyTab(:,3:15))*(100/40); table2array(AugustTab(:,3:15))*(100/40)];
        %2022 - 70% of 2019 figures i.e. 30% reduction
    elseif i == 2022
        UpdatedData = [UpdatedData; table2array(JuneTab(:,3:15))*(100/70);
table2array(JulyTab(:,3:15))*(100/70); table2array(AugustTab(:,3:15))*(100/70)];
    else
        UpdatedData = [UpdatedData; table2array(JuneTab(:,3:15));
table2array(JulyTab(:,3:15)); table2array(AugustTab(:,3:15))];
```

```
    end
end

clear opts
years = StartYear:EndYear;
index = 1;

% Plotting the original data against the updated data using COVID
% coefficients

figure;
for i = 1:length(StartYear:EndYear)
    subplot(length(StartYear:EndYear),2,2*i-1)
    plot(OriginalData(index:index+(720+744+743),5), 'k', 'LineWidth', 1.5);
    title(["Original Data for " years(i)'])
    xlabel('time (hours)');
    ylabel('pedestrian count');

    subplot(length(StartYear:EndYear),2,2*i)
    plot(UpdatedData(index:index+(720+744+743),5),'r', 'LineWidth', 1.5);
    title(["Updated Data for " years(i)'])
    xlabel('time (hours)');
    ylabel('pedestrian count');

    index = index + (720+744+744);

end

set(gcf,'units','inches','position',[0,0,16,12])
```

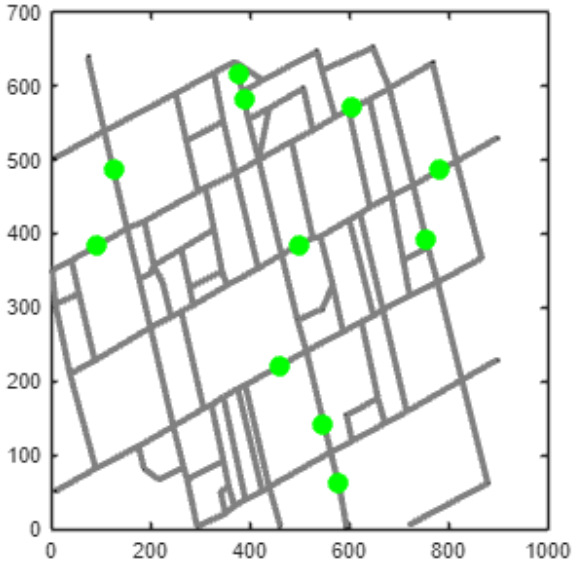


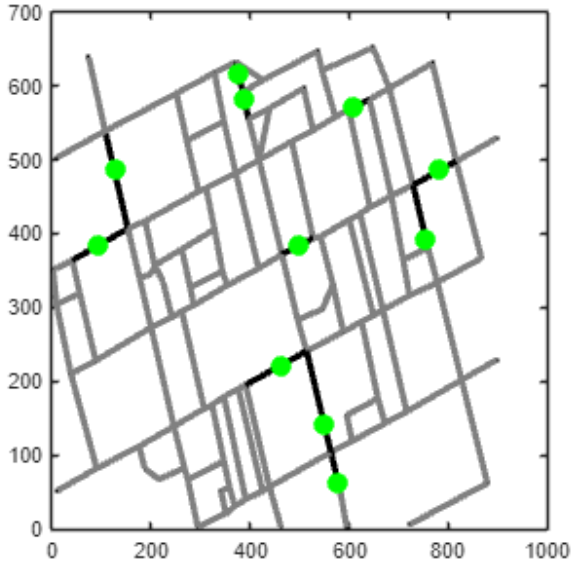



```
UpdatedData = UpdatedData';

% read in xy coordinates for line segments that define the road network
% for the central part of Melbourne investigated here
% (in arbitrary units that could be rescaled to metres)
xs = load('lines_xs.txt');
ys = load('lines_ys.txt');
ys = 655 - ys; % this resetting of y-values is just for plotting purposes
%... usually, you would have to obtain these line segments from mapping
%tools, e.g. Open Street Maps.

% read in sensor xy coordinates (in arbitrary units that could be rescaled to
metres)
sensx = load('sensors_xs.txt');
sensy = load('sensors_ys.txt');
%sensy = 655 - sensy; % this resetting of y-values is just for plotting purposes

% Convert street segments to network
```

```
% Initialize adjacency matrix
n = length(xs) / 2;
A = zeros(n, n);

% Calculate adjacency matrix
% the specifics of this are down to the format of the files
% edges indicate that two road segments are directly connected.
% the weight is 1/2 of both road segment lengths added together.
for i = 1:n
    for j = i:n
        if i ~= j
            % here we check if the two road segments share a start/end
            % coordinate
            xdiff = abs([xs(2*i-1)-xs(2*j-1), xs(2*i-1)-xs(2*j), xs(2*i)-xs(2*j-1),
xs(2*i)-xs(2*j)]);
            ydiff = abs([ys(2*i-1)-ys(2*j-1), ys(2*i-1)-ys(2*j), ys(2*i)-ys(2*j-1),
ys(2*i)-ys(2*j)]);
            diff = xdiff + ydiff;
            if any(diff == 0)
                % here compute the distance, as described above
                % this calculation could be changed, e.g. to include
                % factors that influence the distance perceived by
                % pedestrians, e.g. the pleasantness of links...
                A(i, j) = (sqrt((xs(2*i-1) - xs(2*i))^2 + (ys(2*i-1) -
ys(2*i))^2)/2 + sqrt((xs(2*j-1) - xs(2*j))^2 + (ys(2*j-1) - ys(2*j))^2)/2);
                A(j, i) = A(i, j);
            end
        end
    end
end

%% Find road segments containing sensors
n = length(xs) / 2;
midsx = (xs(1:2:end) + xs(2:2:end)) / 2;
midsy = (ys(1:2:end) + ys(2:2:end)) / 2;
nsens = length(sensx);
idsens = NaN(nsens, 1);

for i = 1:nsens
    dds = sqrt((midsx - sensx(i)).^2 + (midsy - sensy(i)).^2);
    if i == 9
        [~, oo] = sort(dds);
        idsens(i) = oo(2);
    else
        [~, idsens(i)] = min(dds);
    end
    %disp(idsens(i));
end
```

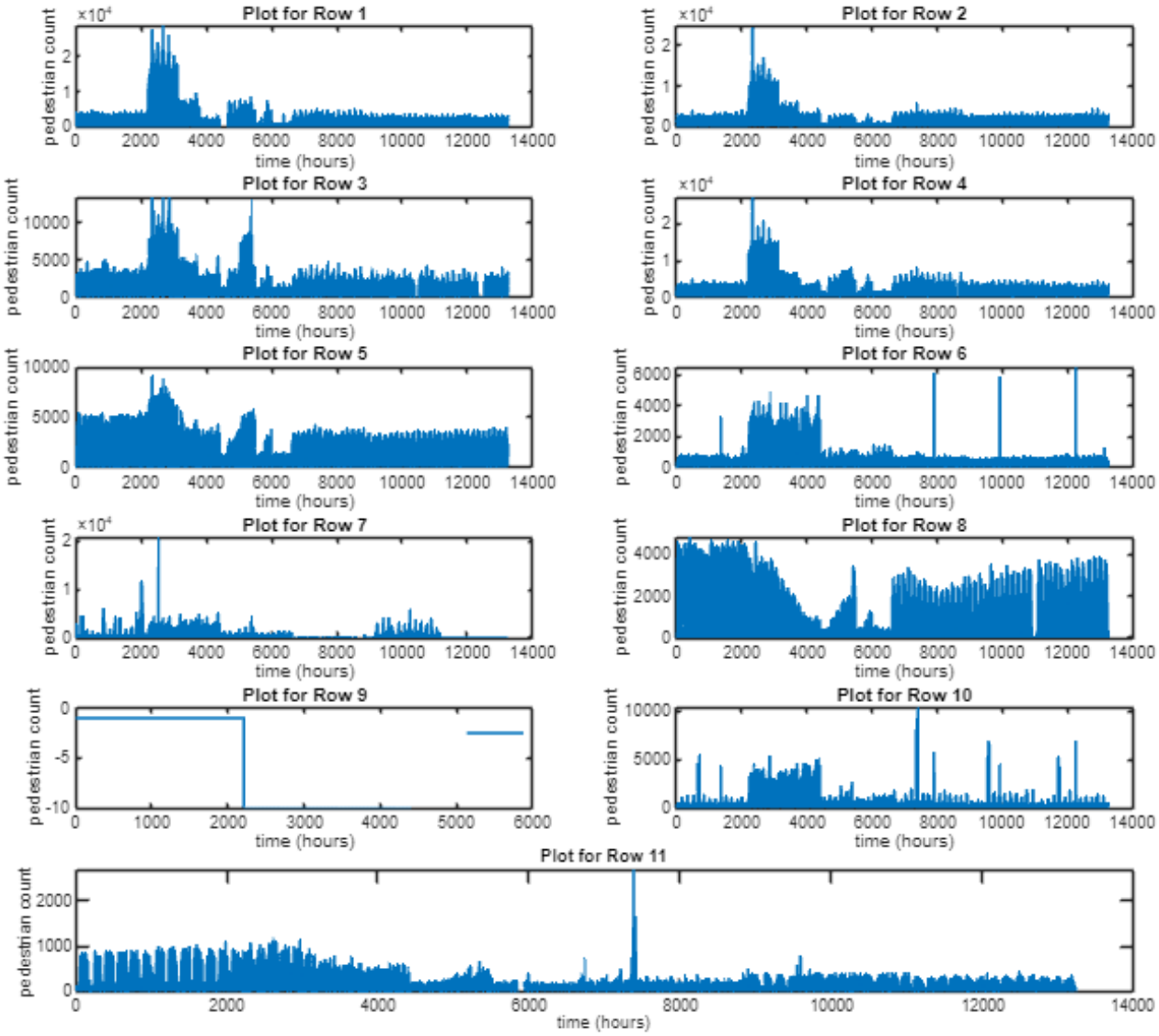
```
% Plot the x and y coordinates
figure;
subplot(1,2,1);
plot(xs, ys, 'k.');  % Plot as points (black dots)
hold on;

% Get number of line segments
n = length(xs) / 2;

% Links x1 x2 y1 y2
links = zeros(length(xs),4);
% Midpoints of links
linksMidpoints = zeros(length(xs),2);

% Loop through and plot each line segment
for i = 1:n
    % Color of the segment (grey)
    coll = [0.5, 0.5, 0.5]; % RGB for grey

    % Plot line segment with specified color and width
    line([xs(2*i-1), xs(2*i)], [ys(2*i-1), ys(2*i)], 'Color', coll, 'LineWidth', 3);
    % Getting coordinates of links
    links(i,:) = [xs(2*i-1), xs(2*i), ys(2*i-1), ys(2*i)];
    linksMidpoints = [(xs(2*i-1)+xs(2*i))/2, (ys(2*i-1)+ys(2*i))/2];
end

% Show sensor locations as green points
scatter(sensx, sensy, 100, 'g', 'filled'); % Size 100, color green, filled circles

hold off;


%% Plot the segments and sensor locations
% segments associated with sensors are highlighted in the plot
subplot(1,2,2);
plot(xs, ys, 'k.');
hold on;
for i = 1:n
    if ismember(i, idsens)
        line([xs(2*i-1), xs(2*i)], [ys(2*i-1), ys(2*i)], 'Color', 'k', 'LineWidth',
3);
    else
        line([xs(2*i-1), xs(2*i)], [ys(2*i-1), ys(2*i)], 'Color', [0.5, 0.5, 0.5],
'LineWidth', 3);
    end
end

scatter(sensx, sensy, 100, 'g', 'filled');
hold off;
set(gcf,'units','inches','position',[0,0,14,6])
```





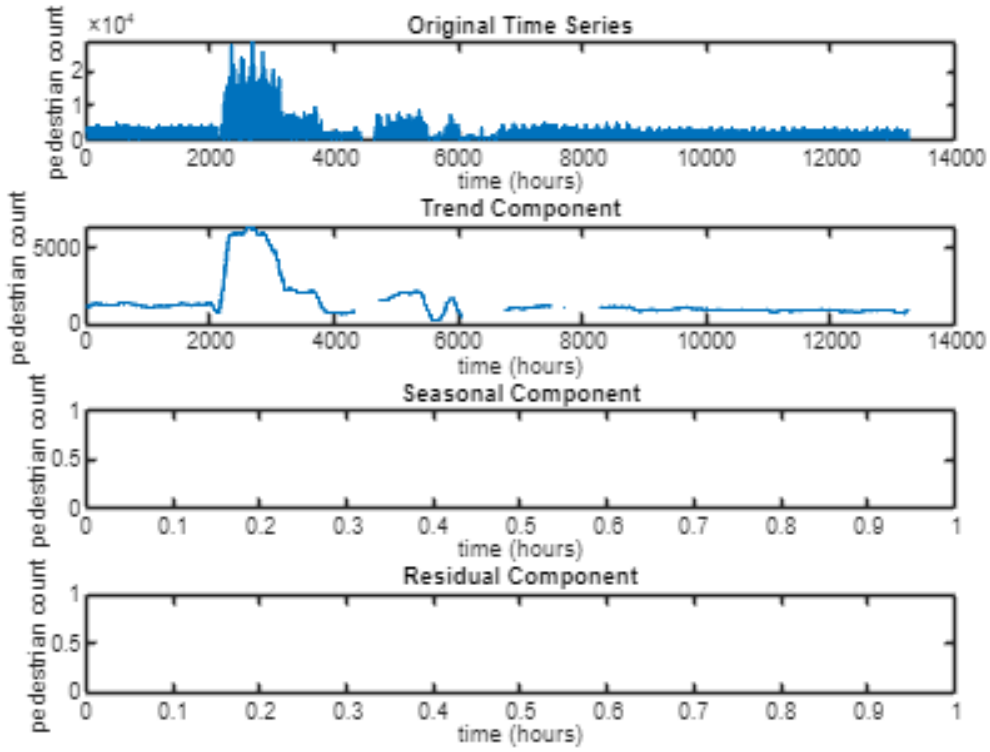

```
%% Merge sensors in the same location
signal = NaN(length(unique(idsens)), size(UpdatedData, 2));
cc = 1;
c = 1;
while c <= length(unique(idsens))
    if sum(idsens == idsens(cc)) > 1
        signal(c, :) = UpdatedData(cc, :) + UpdatedData(cc+1, :);
        % add counts of sensors on the same street segments, as these are
        % typically on opposite sides of the road.
        cc = cc + 2;
        c = c + 1;
    else
        signal(c, :) = UpdatedData(cc, :) * 2;
        % double the counts of sensors that do not share the street segment
        % with another sensor, as these usually cover only one side of the
        %
roaddatetime();datetime();datetime();datetime();datetime();datetime();datetime();dat
etime();uniqueindexesuniqueindexesuniqueindexesuniqueindexesuniqueindexesuniqueindex
esuniqueindexesuniqueindexes
        cc = cc + 1;
        c = c + 1;
    end
end
[sensnodes,uniqueindexes] = unique(idsens); % we're left with 11 sensors out of the
original 13.
UnqiueSensorNames = SensorNames(uniqueindexes);
UniqueSensx = sensx(uniqueindexes);
UniqueSensy = sensy(uniqueindexes);
```

```
UpdatedData = UpdatedData(uniqueindexes,:);

%% Compute shortest paths between nodes and sensors
net = graph(A);
alldists2 = distances(net, sensnodes);

% select the time series from the data that we will use throughout this section:
figure;

for i = 1:length(UnqiueSensorNames)-1
    subplot(round(length(UnqiueSensorNames)/2),2,i); % Create a 4x1 grid of
subplots and select the i-th one
    plot(UpdatedData(i, :), 'LineWidth', 1.5); % Plot the i-th row of data
    title(['Plot for Row ' num2str(i)]); % Title for each subplot
    xlabel('time (hours)');
    ylabel('pedestrian count');
end

subplot(round(length(UnqiueSensorNames)/2),2,11:12); % Create a 4x1 grid of
subplots and select the i-th one
plot(UpdatedData(11, :), 'LineWidth', 1.5); % Plot the i-th row of data
title(['Plot for Row ' num2str(11)]); % Title for each subplot
xlabel('time (hours)');
ylabel('pedestrian count');
set(gcf,'units','inches','position',[0,0,16,14])
```




```
% select the time series from the data that we will use throughout this section:
timeseries = UpdatedData(1,:);
n = length(timeseries);

% Plot the original time series
figure;
subplot(4,1,1);
plot(timeseries, 'LineWidth', 1.5);
title('Original Time Series');
xlabel('time (hours)');
ylabel('pedestrian count');
```

```
%% TREND COMPONENT
% Using a moving average filter to estimate the trend component
window_size = 24*7; % Adjust the window size for the moving average (e.g., weekly)
trend_component = movmean(timeseries, window_size);

% Subtract the trend component from the original series to get the detrended data
detrended = timeseries - trend_component;

% Plot the trend component
subplot(4,1,2);
plot(trend_component, 'LineWidth', 1.5);
title('Trend Component');
xlabel('time (hours)');
ylabel('pedestrian count');

%% SEASONAL COMPONENT
% Assuming the seasonality repeats every 'window_size2' (e.g., daily)
window_size2 = 24;
seasonal_component = detrended(1:window_size2); % First complete seasonal cycle
% compute average over all observed seasonal cycles:
for i = 2:ceil(n/window_size2)
    seasonal_component = seasonal_component + detrended(((i-1)*window_size2+1):
(i*window_size2));
end


seasonal_component = seasonal_component/ceil(n/window_size2);
seasonal_component = repmat(seasonal_component, 1, ceil(n/window_size2));
%seasonal_component = seasonal_component(1:n); % Trim to match time series length
% Plot the seasonal component
subplot(4,1,3);
plot(seasonal_component, 'LineWidth', 1.5);
title('Seasonal Component');
xlabel('time (hours)');
ylabel('pedestrian count');

%% RESIDUAL (RANDOM) COMPONENT
residual_component = timeseries - trend_component - seasonal_component;

% Plot the residual component
subplot(4,1,4);
plot(residual_component, 'LineWidth', 1.5);
title('Residual Component');
xlabel('time (hours)');
ylabel('pedestrian count');
```




```
% As we have assumed an additive decomposition, the time series can be expressed as:
% timeseries = trend + seasonal + residual

TestData = zeros(length(uniqueindexes),round(length(UpdatedData(1,:)) -
0.8*length(UpdatedData(1,:))));
forecast_sarima = zeros(length(uniqueindexes),round(length(UpdatedData(1,:)) -
0.8*length(UpdatedData(1,:))));
AppropriateRows = 1:length(uniqueindexes);
AppropriateRows(AppropriateRows == 9) = [];

for i=AppropriateRows
    timeseries = UpdatedData(i,:);

    % Split the data into training and test sets (80% training, 20% test)
    n = length(timeseries);
    train_size = round(0.8 * n);
    test_size = n - train_size;
    train_data = timeseries(1:train_size);
    test_data = timeseries(train_size+1:end);
    TestData(i,:) = test_data;
    %% recalculate the model forecasts

    %% SARIMA Model
    model_sarima = arima('D',1,'Seasonality',24);
```

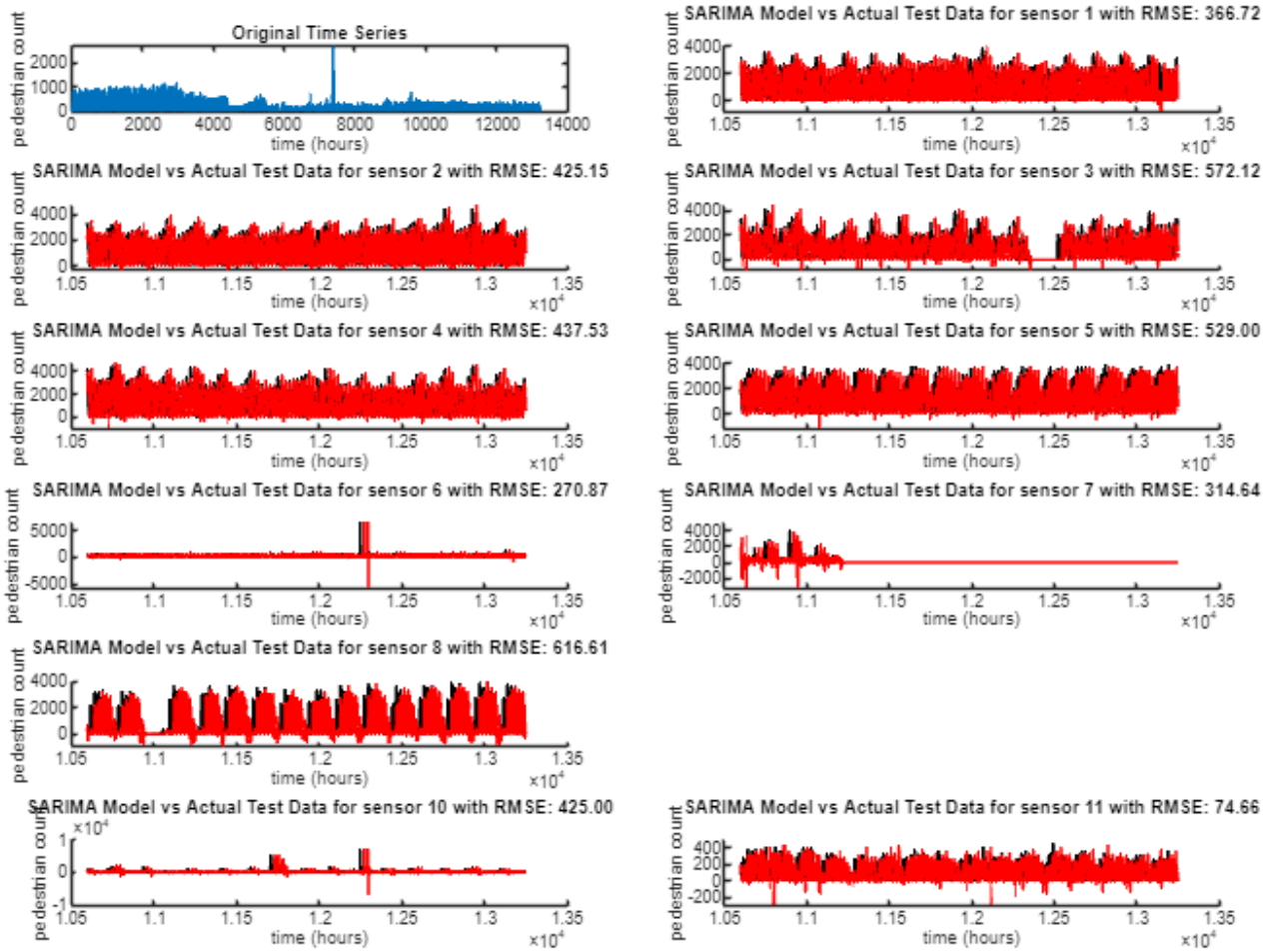
```
    % we have chosen the time period over which seasonality occures, here 24
    % hours.
    fit_sarima = estimate(model_sarima, train_data');

    % the SARIMA model needs more than one previous time step...
    data_for_forecasts2 = [train_data((length(train_data)-48):length(train_data)),
test_data];

    for j = 1:test_size+2
        forecast_sarima(i,j) = forecast(fit_sarima, 1, data_for_forecasts2(j:
(j+24))');
    end
end
```

```
ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

              Value       StandardError    TStatistic    PValue
            __________    _____________    __________    _______

Constant    -0.0053591       4.2507        -0.0012608    0.99899
Variance    1.8572e+05       676.35            274.59          0


ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

              Value       StandardError    TStatistic    PValue
            __________    _____________    __________    _______

Constant      0.10049         3.488          0.028812    0.97701
Variance    1.2854e+05       401.89            319.84          0


ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

              Value       StandardError    TStatistic    PValue
            __________    _____________    __________    _______

Constant     -0.025391       4.3202        -0.0058773    0.99531
Variance    1.9587e+05       1190.6            164.51          0


ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

              Value       StandardError    TStatistic    PValue
            __________    _____________    __________    _______

Constant      0.069161        3.8661         0.017889    0.98573
Variance    1.5601e+05       293.12            532.24          0


ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

              Value       StandardError    TStatistic    PValue
            __________    _____________    __________    _______

Constant      -0.01684        4.0458        -0.0041623    0.99668
Variance    1.7301e+05       996.88            173.55          0


ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):
```

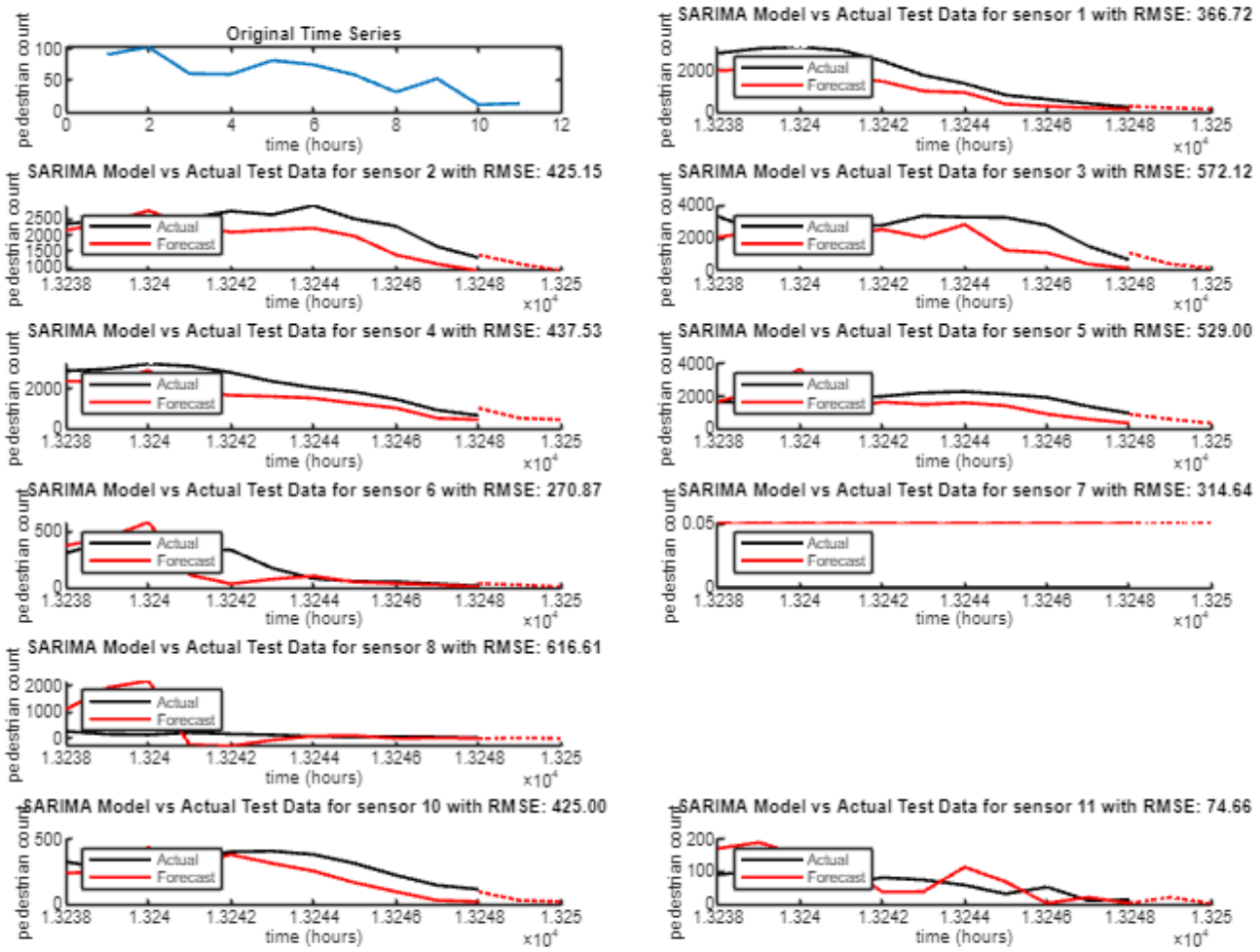
```
                Value       StandardError    TStatistic    PValue
              _________    _____________    __________    _______

Constant     -0.0030046        2.7253        -0.0011025    0.99912
Variance          76936         118.4             649.82          0

ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

                Value       StandardError    TStatistic    PValue
              _________    _____________    __________    _______

Constant       0.051256        6.1915        0.0082785     0.99339
Variance     3.4833e+05        488.87           712.51           0

ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

                Value       StandardError    TStatistic    PValue
              _________    _____________    __________    _______

Constant      -0.011532        3.5901        -0.0032122    0.99744
Variance     1.3261e+05        636.16            208.45          0

ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

                Value       StandardError    TStatistic    PValue
              _________    _____________    __________    _______

Constant      -0.025105        3.8798        -0.0064705    0.99484
Variance     1.5324e+05        292.81            523.35          0

ARIMA(0,1,0) Model Seasonally Integrated (Gaussian Distribution):

                Value       StandardError    TStatistic    PValue
              _________    _____________    __________    _______

Constant     0.0010022          1.0716        0.00093525    0.99925
Variance         11765           68.02            172.96          0
```

```
% Plot the original time series
figure;
subplot(round(length(UnqiueSensorNames)/2),2,1);
plot(timeseries, 'LineWidth', 1.5);
title('Original Time Series');
xlabel('time (hours)');
ylabel('pedestrian count');

for i = AppropriateRows
    %% Accuracy Comparison (RMSE)
    rmse_sarima = sqrt(mean((TestData(i,:)' -
forecast_sarima(i,1:length(train_size+1:n))').^2));
    RMSEForTitle = sprintf(' with RMSE: %.2f\n', rmse_sarima);
```

```
    subplot(round(length(UnqiueSensorNames)/2),2,i+1); % Create a 4x1 grid of
subplots and select the i-th one
    %Plotting historical data and forecasts against these values
    hold on
    plot(train_size+1:n, TestData(i,:), 'k', 'LineWidth', 1.5); hold on;
    plot(train_size+1:n, forecast_sarima(i,1:length(train_size+1:n)),'r',
'LineWidth', 1.5);
    %Plotting forecast data for future
    plot(n:n+2, forecast_sarima(i,length(train_size+1:n):length(train_size+1:n)
+2),'r', 'LineWidth', 1.5,'LineStyle',':');
    hold off
    % Making pretty
    title(['SARIMA Model vs Actual Test Data for sensor ' num2str(i) RMSEForTitle]);
    xlabel('time (hours)');
    ylabel('pedestrian count');
end

% Setting figure size
set(gcf,'units','inches','position',[0,0,16,12])
```

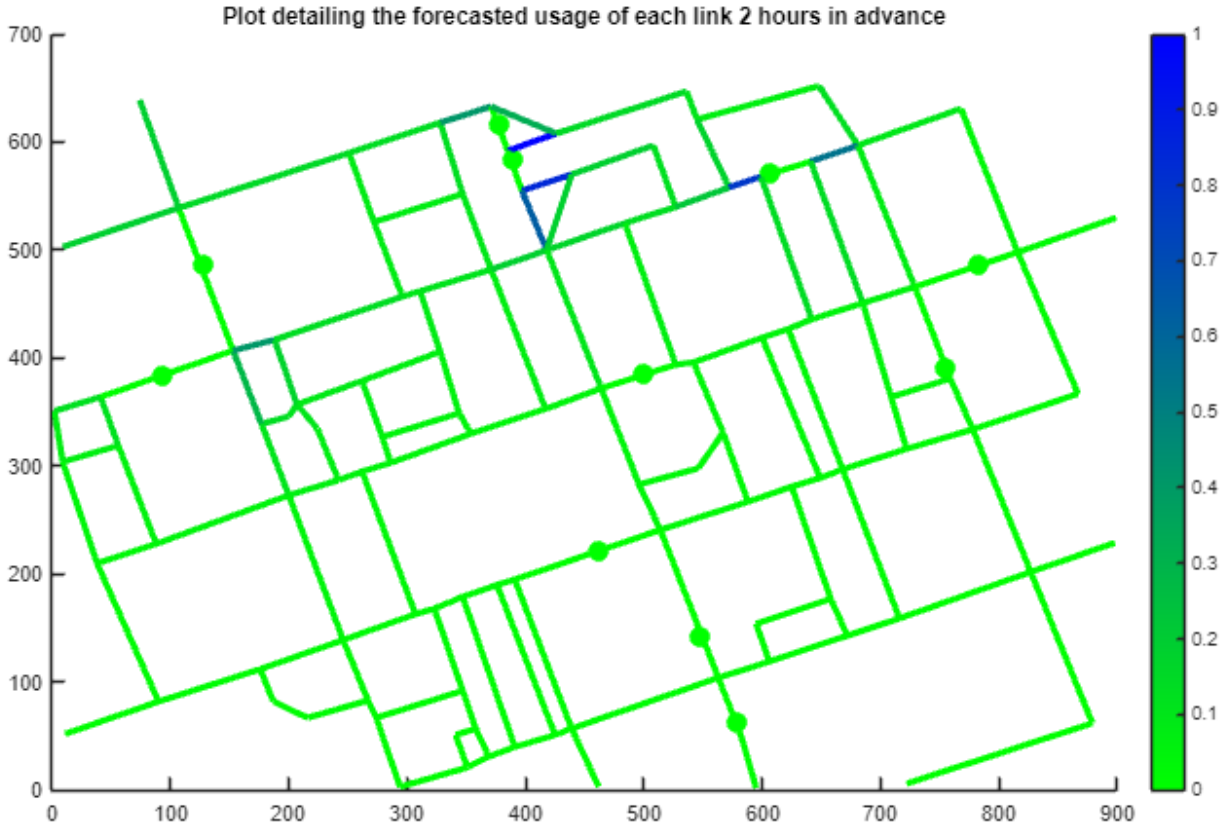


```
AmountOffEnd = 10;

% Plot the original time series - looking a bit closer
figure;
subplot(round(length(UnqiueSensorNames)/2),2,1);
plot(timeseries(end-AmountOffEnd:end), 'LineWidth', 1.5);
title('Original Time Series');
xlabel('time (hours)');
ylabel('pedestrian count');

for i = AppropriateRows
    %% Accuracy Comparison (RMSE)
    rmse_sarima = sqrt(mean((TestData(i,:)' -
forecast_sarima(i,1:length(train_size+1:n))').^2));
    RMSEForTitle = sprintf(' with RMSE: %.2f\n', rmse_sarima);
    subplot(round(length(UnqiueSensorNames)/2),2,i+1); % Create a 4x1 grid of
subplots and select the i-th one
    %Plotting historical data and forecasts against these values
    hold on
    plot(n-AmountOffEnd:n, TestData(i,end-AmountOffEnd:end), 'k', 'LineWidth',
1.5); hold on;
    plot(n-AmountOffEnd:n, forecast_sarima(i,end-AmountOffEnd:end),'r',
'LineWidth', 1.5);
    %Plotting forecast data for future
    plot(n:n+2, forecast_sarima(i,length(train_size+1:n):length(train_size+1:n)
+2),'r', 'LineWidth', 1.5,'LineStyle',':');
    hold off
    % Making pretty
    title(['SARIMA Model vs Actual Test Data for sensor ' num2str(i) RMSEForTitle]);
    xlabel('time (hours)');
    ylabel('pedestrian count');
    legend('Actual', 'Forecast',Location='northwest');
end

% Setting figure size
set(gcf,'units','inches','position',[0,0,16,12])
```




```
% Setting up required traffic array
Traffic = zeros(length(alldists2(1,:)),1);
% Making sure to not divide by zero
alldists2(alldists2 == 0) = NaN;

% Gravity model for all sensors and links
for i=1:length(Traffic)
    % All sensors
    for j=1:length(alldists2(:,1))
        Traffic(i) = Traffic(i) + forecast_sarima(j,end)*(1/(alldists2(j,i)^2));
    end
end

% Data preprocessing to colour code links
TrafficColourCodes = (Traffic - min(Traffic,[],'omitnan')) ./ (max(Traffic,
[],'omitnan') - min(Traffic,[],'omitnan'));
% Removing NaNs and infinities
TrafficColourCodes(isnan(TrafficColourCodes) | TrafficColourCodes == -inf) = 0;
```

```
% Removing negatives
TrafficColourCodes(TrafficColourCodes < 0) = -TrafficColourCodes(TrafficColourCodes
< 0);
% Normalising data to colour code links
TrafficColourCodes = TrafficColourCodes./max(TrafficColourCodes);

figure;
RGB1 = [0, 1, 0];  % Red [R, G, B]
RGB2 = [0, 0, 1];  % Blue [R, G, B]

% Show sensor locations as green points
scatter(sensx, sensy, 100, 'g', 'filled'); % Size 100, color green, filled circles

% Loop through and plot each line segment
for i = 1:length(Traffic)
    RGB = (max(TrafficColourCodes) - TrafficColourCodes(i)) * RGB1 +
TrafficColourCodes(i) * RGB2;

    % Plot line segment with specified color and width
    line([xs(2*i-1), xs(2*i)], [ys(2*i-1), ys(2*i)],'Color',RGB, 'LineWidth', 3);
end
% Apologies the colourbar to show the link flow usage was very difficult to
% plot
title('Plot detailing the forecasted usage of each link 2 hours in advance')
colormap([linspace(RGB1(1), RGB2(1), 255)', ...
                  linspace(RGB1(2), RGB2(2), 255)', ...
                  linspace(RGB1(3), RGB2(3), 255)']);  % Apply the custom colormap
colorbar;  % Display the colorbar

set(gcf,'units','inches','position',[0,0,16,10])
```



(ii) Evaluate the performance of your model using data (note that marking places a higher emphasis on the quality of the evaluation than on the prediction accuracy).

Evaluating performance of the model:

- When training the SARIMA model it tests poorly on most sensors when compared to when tested in the lab. It produces RMSE scores of up to 616 on sensor 8 with a low score of 270 on sensor 6. This is higher than the RMSE score of 220 obtained in the lab sheet although not by much and is heavily dependent on the sensor selected. This highlights however that a larger sample size doesn't necessarily correlate with more accurate scores for the SARIMA model and that the quality of data is much more important.
- Something else that can be noticed is the discontinuity in the forecasts for the network that can be compared to real data and the forecasts for 2 hours in advance. This is a criticism of the approach and not necessarily the data as discontinuities normally arise from how the training and testing data is split although the poor quality of data also has an impact here.
- In short, the model performs poorly with the colour coded links above only showing high traffic forecasted on small roads. This is caused by the gravity model approach where the longer the street, the less close it is likely to be to a sensor as the midpoint is taken of the link. This therefore means shorter roads which are very close to sensors, like the ones seen at the top, will always predict higher pedestrian traffic on

this part of the network while the other longer links predict much much lower pedestrian traffic than in reality.

Question 6 - Practitioner studies

For all parts of the question, please provide your answers as bullet points.

Suppose you are a transport consultant. You are being tasked to help design a pedestrian bridge from a public railway station to a large stadium that is used for sports matches, concerts, and political rallies. This bridge will have to deal with high volumes of pedestrian traffic. There are no level changes and the length of the bridge is fixed. Figure1 below gives an indication of passenger traffic at the railway station for a typical event starting at 19:00 finishing at 20:30. Your customer is concerned with safety, construction costs, and user experience in this order of importance. The authorities have provided a simple web tool based on a model for pedestrian traffic that you must use in your work.

You can find it here: https://seis.bristol.ac.uk/~nb14397/TMMcalculation.html

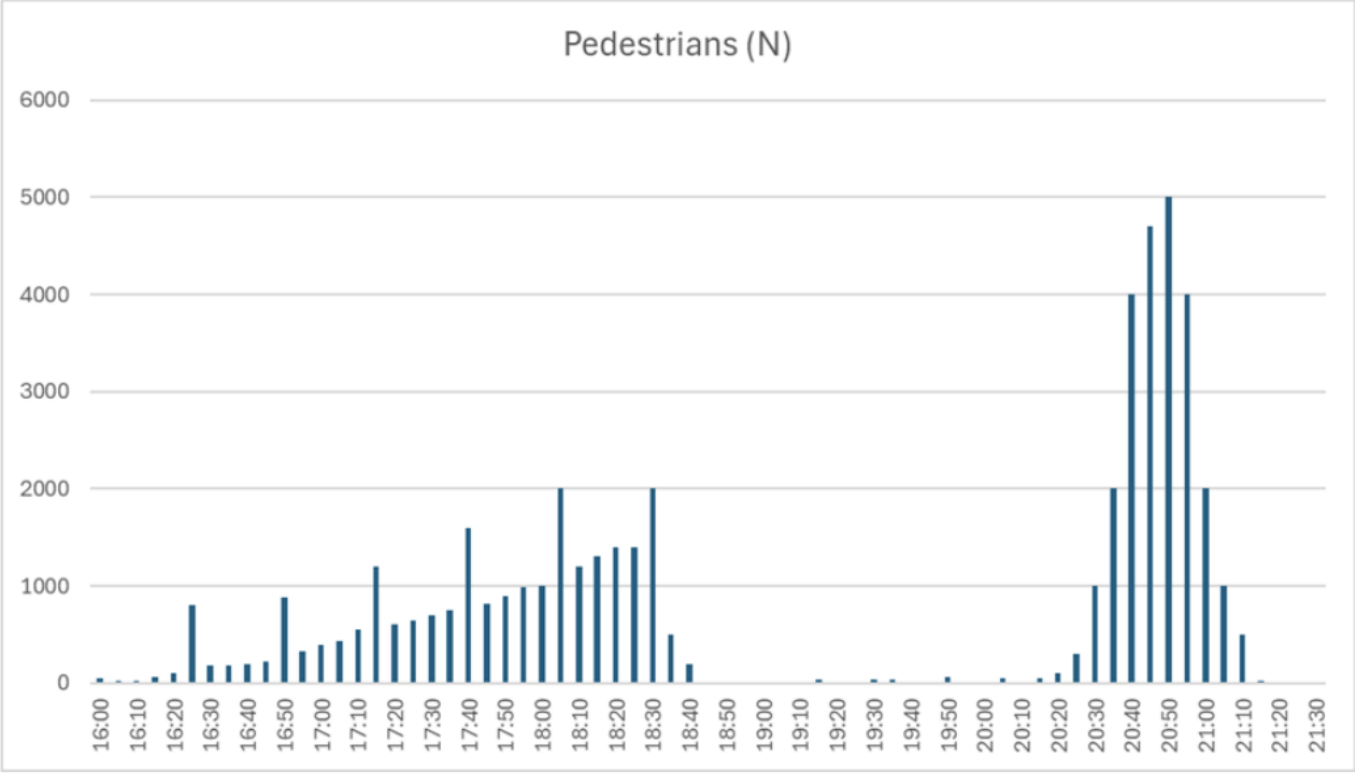


Figure 1: Passengers numbers at the railway station (sum of arriving and departing).

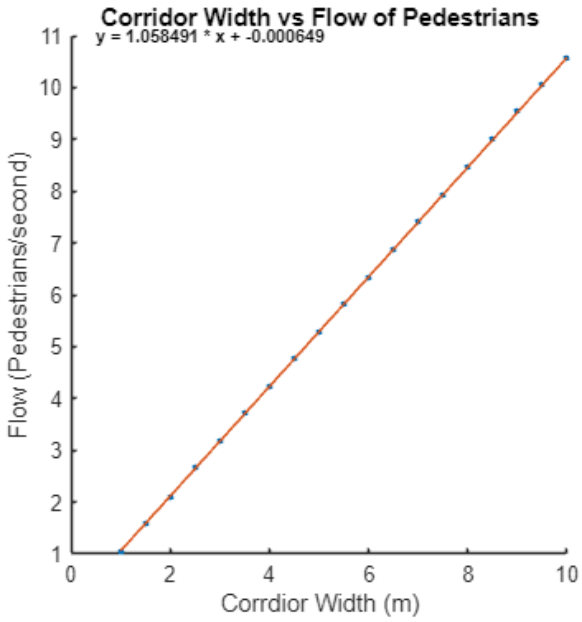
(a) For the benefit of your customer, and for writing your report, you should understand the model in the web tool. Investigate and then describe this model, its properties, and what it predicts about pedestrian traffic by using the output the website provides, calling the website no more than 100 times (to save energetic costs and because we can imagine there may be licensing fees). In your answer, you should focus on how you use the online model output and your justification for this rather than simply stating an inferred model formulation.

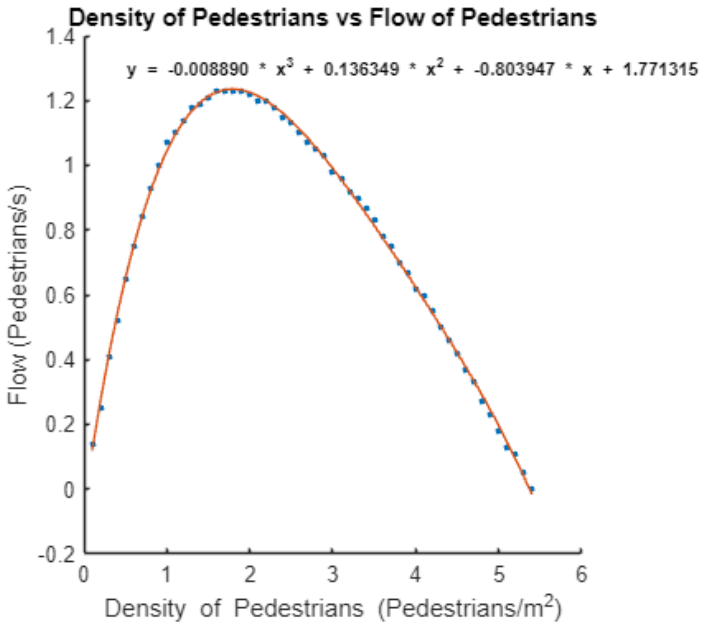
- From the model in the web tool I have obtained the values of flow when setting the number of pedestrians per square meter to 1 and varying the corridor width between the values of 1 and 10.

- I obtained values about the flow when changing the number of pedestrians per square meter between 0.1 and 5.4 when setting the corridor width to 1 to directly compare with the values of flow when changing the corridor width.
- I did this to explore the relationships between the variables of this model and their impacts when changed. The values obtained are plotted below and fitted as seemingly required.

```
% Values plugged into the web model when changing the corridor width
CorrdiorWidthsChangingWidth =
[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10];
CorrdiorPedsPerSqMChangingWidth = 1;
% Output pedestrian flow values
CorrdiorFlowsChangingWidth =
[1.05,1.59,2.11,2.66,3.17,3.71,4.23,4.78,5.29,5.82,6.33,6.88...
    ,7.41,7.92,8.48,9,9.54,10.05,10.58];
% Fitting a polynomial (here degree 1) to the data
CoeffsChangingWidth =
polyfit(CorrdiorWidthsChangingWidth,CorrdiorFlowsChangingWidth,1);
% Obtaining the y-values when fitting the polynomial to the data
PolyYChangingWidth = polyval(CoeffsChangingWidth,CorrdiorWidthsChangingWidth);

% Values plugged into the web model when changing the density of
% pedestrians
CorridorWidthsChangingPedsPerSqM = 1;
CorrdiorPedsPerSqMChangingPedsPerSqM =
[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1,1.2,1.3,1.4,1.5,1.6...

,1.7,1.8,1.9,2,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,3,3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8
,3.9,4,4.1,4.2...
    ,4.3,4.4,4.5,4.6,4.7,4.8,4.9,5,5.1,5.2,5.3,5.4];
% Output pedestrian flow values
CorridorFlowsChangingPedsPerSqM =
[0.14,0.25,0.41,0.52,0.65,0.75,0.84,0.93,1,1.07,1.1,1.14,1.18,1.19...

,1.21,1.23,1.23,1.23,1.23,1.22,1.2,1.2,1.18,1.15,1.13,1.1,1.07,1.05,1.03,0.98,0.96,0
.92,0.9,0.87...

,0.83,0.78,0.75,0.7,0.67,0.62,0.6,0.55,0.5,0.46,0.42,0.37,0.33,0.27,0.23,0.18,0.13,0
.11,0.05,0];
% Fitting a polynomial (here degree 3) to the data
CoeffsChangingPedsPerSqM =
polyfit(CorrdiorPedsPerSqMChangingPedsPerSqM,CorridorFlowsChangingPedsPerSqM,4);
% Obtaining the y-values when fitting the polynomial to the data
PolyYChangingPedsPerSqM =
polyval(CoeffsChangingPedsPerSqM,CorrdiorPedsPerSqMChangingPedsPerSqM);

% Setting up the figure
figure;
sz = 8;
```

```
% Plotting when the corridor width is changing
subplot(1,2,1)
hold on
% Corridor width on the x-axis, pedestrians flow on the y-axis
scatter(CorrdiorWidthsChangingWidth,CorrdiorFlowsChangingWidth,sz,'filled')
% Plotting the fitted data for testing
plot(CorrdiorWidthsChangingWidth,PolyYChangingWidth)
hold off
% Making the plot pretty
title("Corridor Width vs Flow of Pedestrians")
xlabel("Corrdior Width (m)")
ylabel("Flow (Pedestrians/second)")
caption = sprintf('y = %f * x + %f', CoeffsChangingWidth(1),
CoeffsChangingWidth(2));
text(0.5, 11, caption, 'FontSize', 8, 'Color', 'k', 'FontWeight', 'bold');

% Plotting when the pedestrian density is changing
subplot(1,2,2)
hold on
% Pedestrian density on the x-axis, pedestrians flow on the y-axis
scatter(CorrdiorPedsPerSqMChangingPedsPerSqM,CorridorFlowsChangingPedsPerSqM,sz,'fil
led')
% Plotting the fitted data for testing
plot(CorrdiorPedsPerSqMChangingPedsPerSqM,PolyYChangingPedsPerSqM);
hold off
% Making the plot pretty
title("Density of Pedestrians vs Flow of Pedestrians")
xlabel("Density of Pedestrians (Pedestrians/m^2)")
ylabel("Flow (Pedestrians/s)")
caption = sprintf('y = %f * x^3 + %f * x^2 + %f * x + %f',
CoeffsChangingPedsPerSqM(1)...
    , CoeffsChangingPedsPerSqM(2), CoeffsChangingPedsPerSqM(3),
CoeffsChangingPedsPerSqM(4));
text(0.5, 1.3, caption, 'FontSize', 8, 'Color', 'k', 'FontWeight', 'bold');

% Making the plot pretty
set(gcf,'units','inches','position',[0,0,14,6])
```





- The left-hand plot above therefore shows a linear relationship between the corridor width with a coefficient of approximately 1.058 and minimal y-intercept. As would be expected as if there is very minimal corridor space, there would be virtually no flow.
- The right-hand plot however shows an interesting polynomial relationship. It is an approximately quadratic relationship with roots at 0 and 5.4 and maximum value of flow of 1.22, obtained at a density of approximately 1.8.
- This suggests that pedestrian traffic flow will increase linearly with increasing corridor width but can only be maximised if pedestrians maintain a density of around 1.8 pedestrians per square meter. Any closer and the site becomes too tight to move and any further suggests a crowd that are not particularly close and so have no need to be somewhere with time pressure, unrealistic for the scenario detailed. The maintaining of a density of pedestrians between 1 and 2 pedestrians per square meter is essential in the case of this client and while, for this brief, it is understood that low volumes of pedestrian traffic are outside the scope of the brief.

(b) Use the model provided in the web tool and the information provided in the question to make recommendations about the design of the bridge, focusing on elements relevant to foot traffic and considering the relevant stakeholders. You may suggest adapting the model, if necessary. Explain and justify your approach.

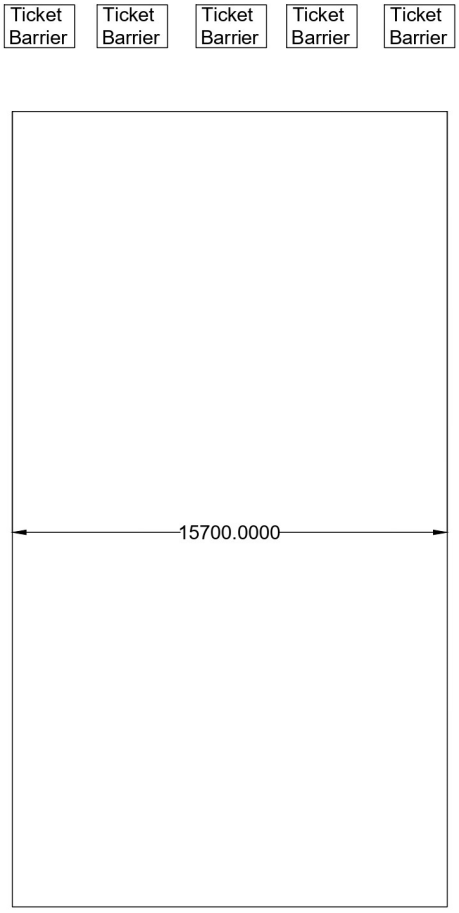
- The web tool details the relationships between corridor width and density of pedestrians, which is essential for this client as the bridge can be modelled as a corridor of infinite length.
- The density of pedestrians is a paramount safety consideration for high volumes of pedestrian traffic.
- The information shown in the question details the pedestrian flow at different times before and after a typical event starting and finishing, showing the typical pedestrian flow that would be apparent on the bridge between the railway station and stadium.

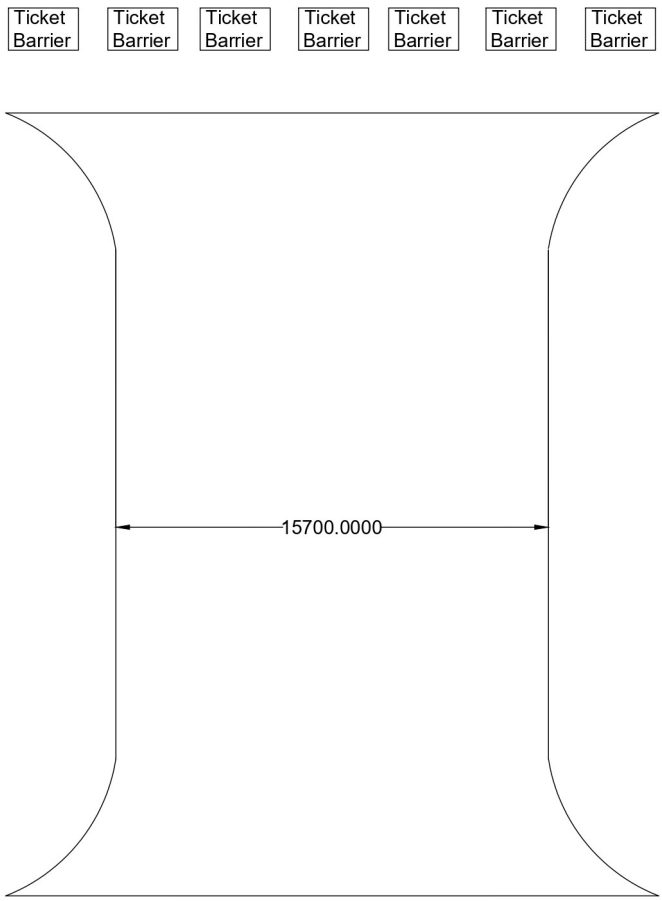
- Importantly it highlights the large spike after the event finishes, highlighting the flow of approximately 25,000 people in 30 minutes or a pedestrian flow of 13.9 pedestrians per second average over that time. This time period has a peak flow of 5,000 people in 5 minutes or a approximately 16.7 pedestrians per second.
- Another thing to understand is that the spikes before the event starts are approximately 25 minutes apart, suggesting that there is a train arriving and departing from the railway station every 25 minutes.

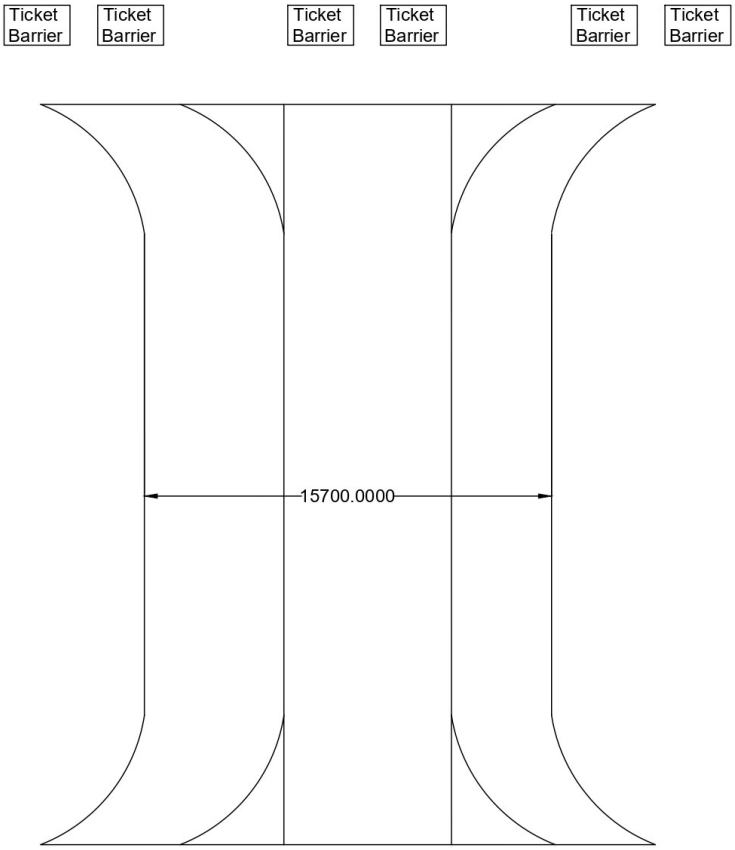
- These values are citicial to key stakeholders:
- **Pedestrians** - will provide input into accessibility, convenience, safety concerns and pedestrian flow. This includes local residents and commuters that use the station regularly as well as fans attending the events at the stadium.
- **Stadium operators** - will need to specify event schedules and capacity as well as be primarily responsible for pedestrian safety and flow. Importantly will also fund the project (probably) and so are the primary client.
- **Train operators** - will want to know the predicted passenger flow before & after events so they can plan their railway schedules, operations and staffing around events, therefore they will be able to serve the maximum number of pedestrians in the most efficient way. They could also possibly provide some funding and so are a secondary client.
- **Bridge/master-plan designers/engineers/architects** - should comprise of a group coordinating effectively with other stakeholders to understand pedestrian flows, safety and building code regulations as well as client desires including the surrounding area. It is critical that they use pedestrian flow modelling to understand safety, structural and accessibility requirements with the understanding that crowds normally step in-phase with each other creating larger loads on the bridge than what might be anticipated (i.e. the millennium bridge scenario).
- **Construction contractors/utility providers** - ensure correct use of materials to meet local government safety and building regulations as well minimise environmental impact. Critical for construction is the scheduling of work and deliveries to site as works during large events are likely to cause large disruptions both to construction and to pedestrian flow around the stadium, meaning it is essential to complete works around large events to minimise impacts.
- **Local government** - critical to make sure the bridge is designed and built safely that adheres to local governmental compliance measures including urban development plans, environmental impact assessments and enforce building codes and regulations. They are also likely to provide some funding and are therefore a secondary client.

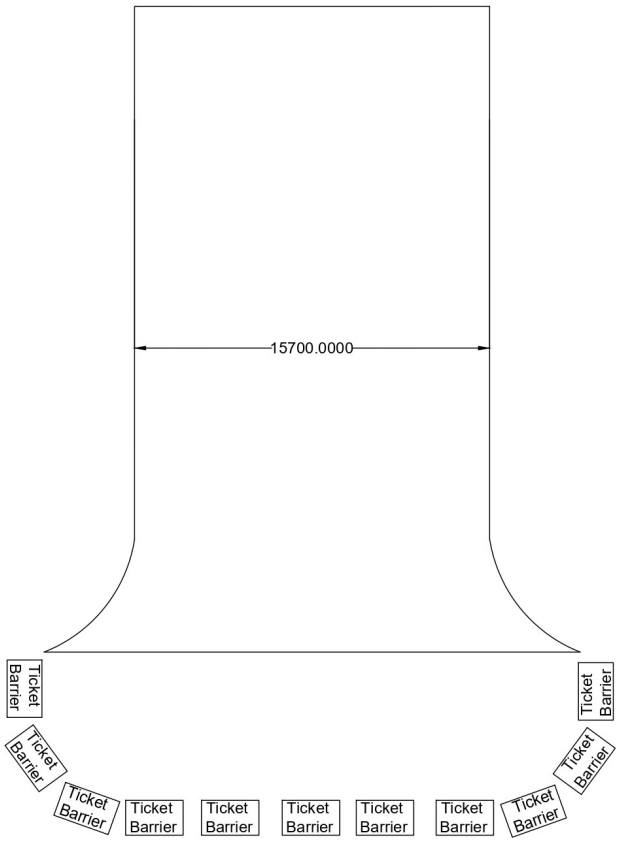
- Primarily pedestrian densities should be kept at approximately 2 pedestrians per m^2, and therefore at this density the corresponding value of flow is approximately 1.2 pedestrians per second.
- However, it is understood that pedestrian density varies significantly by event, especially during large events so a realistic density of 1 pedestrians per m^2 and therefore flow of 1.07 pedestrians per second is used to effectively handle the peak flow from the stadium.

- Using this value, to meet a peak flow of 16.7 pedestrians per second, a bridge width of 15.7 m is therefore necessary.
- This value was validated with the web tool.
- Although a gross overestimate for the large majority of the time the bridge is used, a width of 15.7m would ensure smooth and safe pedestrian flow after large events at the stadium. **A sketch of the initial bridge design is shown below in the 1st image.**






Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
15700.0000


15700.0000
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier
Ticket Barrier

- Three suggestions for the design of the bridge are therefore made and displayed using AutoCAD:

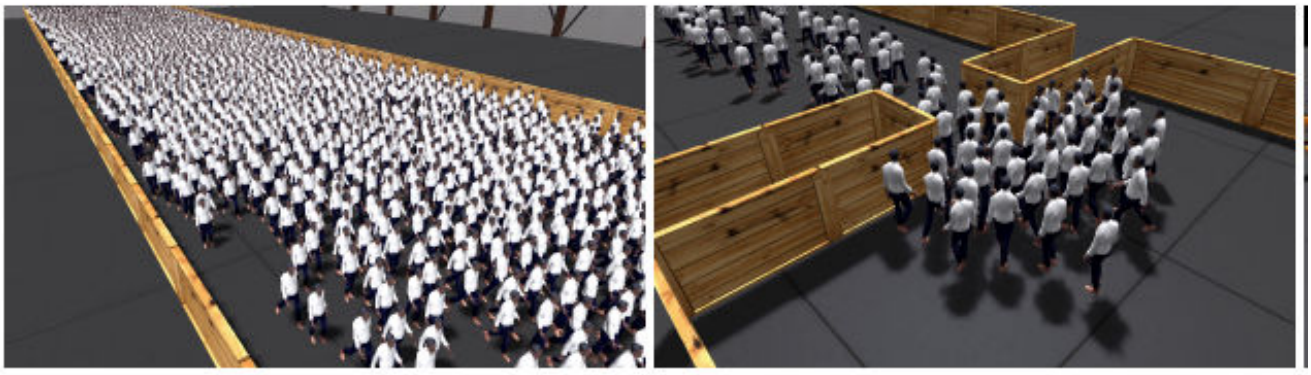
1. **Curved entrance and exits of the bridge (2nd image)** - it is expected that this will ease flow due to the removal of corners which, in all fluid models, are known to cause slight bottlenecks as it is much more difficult to turn at a right angle than to do it more gradually. By widening the end of the bridge it is also anticipated that more ticket barriers could be available for use by the crowd without the pedestrians being required to make a conscious decision to use them, aiding in preventing traffic build up. However, it is understood that this increase in passenger flow to the train station could cause issues with safety on the platform further along the path for pedestrians and the increase in people would cause extra loading on the bridge. Both issues would need to be investigated further to understand the safety implications for pedestrians but it is anticipated minimal extra materials/costs would be necessary for improved traffic flow.
2. **Split the pedestrian flow up within the bridge (3rd image)** - implementation of this could aid with the loading and pedestrian flow issues mentioned for just the curved model. Pedestrians generally for streamlines in crowds and this bridge model could use that idea to its advantage with the three sections thought to follow general streamline patterns maintaining smooth flow with no pedestrian flow collisions likely to take place between streamlines. It could also help manage this flow with one section able to be blocked if ticket barriers require maintenance, there are safety concerns further along the pedestrian path or accessibility needs to be increased for those who utilise wheelchairs. However, this implementation could also be temporary for when crowd numbers are at their highest, while returning to normal curved bridge design for most of the time. This design could also cause bottlenecks with people at the entrance to the bridge but should facilitate easier crowd management.
3. **Place ticket barriers at the start of the bridge in a semicircular pattern (4th image)** - This design is thought to be the best as it facilitates the easiest crowd management by the use of ticket barriers to maintain a steady flow. It also reduces the maximum loading on the bridge by meaning queueing due to high pedestrian traffic taking place before the bridge, minimising the number of people on the bridge. However, it does require extra investment in ticket barriers and therefore staff while also restricting the use of the bridge for pedestrians not using the bridge to get to the railway station as their destination. It is anticipated if this design is used, that a smaller bridge may also be available to be built, but this would come with very different planning permissions and building regulations for stakeholders to achieve.

- Adapting the web tool model to provide details of streamlines that the crowds fall into i.e. which part of the corridor will pedestrians enter into and how does their position change at the end of the corridor. This would give insights into the type of crowd flow in the corridor (i.e. if people form lines) and therefore the bridge design and ticket barrier configuration could be designed to better fit crowd flow.
- Another item that has been identified is that at the end of the event, spikes in passenger numbers being taken away by trains can't be seen as clearly as when the pedestrians arrive at the station in the information given in the question. It can therefore not be understood the distribution of trains being used to deliver passengers back to their origins and so further crowd management techniques could be explored further if this information is available (i.e. when to allow pedestrian flow and when to not after an event).

- More granular data given in figure 1 would also be useful to understand a more detailed pedestrian flow, especially after the event to give a more accurate peak pedestrian flow value and therefore greater insight into the maximum flow experienced at the bridge with pedestrians attempting to enter the railway station.

(c) If you were to be given access to one other model covered in the lecture slides on microscopic modelling to make your design recommendation, which model would it be, and why? Stick to the same brief as in part (b) and state clearly where in the lecture notes the model was covered. We expect a sufficiently detailed explanation as to why you choose this model including an explanation on how it would be used.

- In this scenario there is continuous space and time and therefore a social-force model (agent based) is appropriate to model the flow of pedestrians. The Cellular-automata model is not appropriate due to it requiring discrete space while macroscopic models are inappropriate due to the crowd scenario not allowing collisions between particles and obstacles (apart from if Milwall are playing).
- A path of least effort approach to crowd simulation is recommended (PLEdestrians - Week 4: Microscopic Modelling Introduction and Concepts - Pedestrian Crowd Modelling (video 2) - slide 2). PLEdestrians is chosen due to smooth collision free motion which utilises bio-mechanical energy expended (total metabolic energy) by an agent to quantify "least effort" rather than shortest path. This is especially critical in crowds as often the shortest path determined by Dijkstra's algorithm or similar is not the quickest nor easiest in a crowd due to this path often colliding with other pedestrian paths. Therefore a greater quantifying of self-organisation phenomena is used in PLEdestrians while still utilising the key shortest available route principle.
- The image below also details queueing of agents at a bottleneck, which the bridge is expected to be. It shows the PLEdestrian agents form a system around the doorway while still maintaining flow, typically an accurate representation of what happens at ticket barriers while the long corridor shows most agents travelling at the same speed in columns.



(a) *Long Corridor* **(b)** *Narrow Passage*

- An image of implementation of the PLEdestrains model and the comparison with a still from video footage from the Shibuya Station in Tokyo is shown below ((PDF) PLEdestrians: A Least-Effort Approach to Crowd Simulation.). The model appears to be fairly accurate and the implementation around a train station with large crowds is particularly relevant in this scenario.
- However, it should be recognised that a combination of visual, microscopic, macroscopic and phenomenological validation models should be created to accurately predict flow from the stadium to the railway station. An example of this implemented is the modelling of Wembley Stadium (Wembley Stadium - Pedestrian Modelling - PJA).

(a) *Simulated of Shibuya Station in Tokyo* (b) *A different view of the simulated crossing* (c) *A still from a video of the crossing*

Figure 1: *Our approach automatically generates many emergent crowd behaviors at interactive rates in this simulation of Shibuya Crossing (left, middle) that models a busy crossing at the Shibuya Station in Tokyo, Japan. (right). The trajectory for each agent is computed based on minimizing an biomechanically derived effort function.*