# Origin-destination flow matrix and its eigenvectors.

## What is the flow matrix?

We want to build a representation of the origin-destination data that tells us about the flow of people from one area to another based on journeys captured within the time period contributing to the matrix. Note that we do this by summing over parts of the day we have included in our dataset (which is broken down to average journeys by start hour).

We have a vector, *v,* (which is 9370 elements long, for the 9370 MSOAs, IZs and SOAs in the data). This vector tells us how many people are in each MSOA before the matrix is applied.

So *v7am* is the number of people in each MSOA at 7 am. We can find *v8am* as:

*v8am = M(7am to 8am)v7am*

We find the entries of the matrix *M(7am to 8am)* using the journey counts where the hour\_part is 7 (these are journeys beginning between 7 and 8am).

*M* is a 9370 x 9370 matrix. Each matrix element corresponds to the flow between MSOA\_i and MSOA\_j, and the diagonal elements reflect people who remain in the same MSOA.

If we look at the product, we can see what the matrix elements need to be:

The *ith*element is: *M\_i,0 \* v\_0 + M\_i,1 \* v\_1 + … + M\_i,i \* v\_i + … + M\_i,9370 v\_9370*

This is then the number of people is MSOA\_i after the journeys in *M.*

These pieces correspond to:  
(people moving from MSOA\_0 to MSOA\_i) +   
(people moving from MSOA\_1 to MSOA\_i) +   
… +   
(people remaining in MSOA\_i) +   
… +   
(people moving from MSOA\_9730 to MSOA\_i)

To make this simple interpretation work, we need to build the flow matrix elements to be proportions of the population in each MSOA.

## Eigenvalues and eigenvectors of the flow matrix

Quick refresh of linear algebra:

Key properties of square matrices (*M* is square) are encoded in eigenvectors. An eigenvector of a matrix is a vector which returns a scalar multiple of itself when multiplied by the matrix (conventionally labelled λ).

*Mv = λ v*

The number of eigenvectors will be the dimension of the matrix (9370 for us).

### What do eigenvectors tell us?

The eigenvectors are then subsets of the MSOA data which behave simply under the transformation given by the origin-destination matrix. The result of multiplying of the matrix is a straightforward multiple of the starting vector:

As an example

Means that an effect of the matrix *M* is that people go from MSOA\_3 to MSOA\_1 and MSOA\_2.

Note that because we can associate each vector component with an MSOA, we can plot the eigenvectors as a map.

### What do we expect to find?

We have an expectation that the matrix eigenvectors will tell us about functional economic areas. This is because we know that ‘most’ travel is localised.

If we imagine re-arranging the rows and columns such that MSOAs in each area have adjacent indices, we can further infer what the eigenvectors will look like.

Imagine we have London MSOAs with indices 0 to *i,* and then the Birmingham area for *i* to *j*, etc.

Then the matrix will be approximately block-diagonal. This means in the top *i* rows and columns there will be a submatrix containing movement within London. Outside of this will be travel between areas (which we assume here is smaller).

The eigenvectors of this matrix will contain localised eigenvectors, which have zeroes in most of the components. In this example, we’d see:

Where *v1* vector contain 9370 entries, *vLondon* is a subvector with *i* components.

So if we extract the dominant travel patterns from the flow matrix and map out the eigenvectors, we should see:

* Eigenvectors clustered geographically around city centres and the travel area surrounding them. Note that we would see multiple vectors highlighting the same cluster (the number is approximately the number of MSOAs in the cluster).
* Cross-regional travel – MSOAs containing airports are likely to be prominent.

### Processing [still rough draft for this, as this needs further consideration]

If our aim is to group MSOAs, then we need to ignore eigenvectors which tell us the same thing as we’re already getting from other eigenvectors. This means we want to find those which are approximately (up to a threshold we need to define) orthogonal to the ones we already have.

We also need to assign MSOAs to be appreciably non-zero within an eigenvector. The plotting code is using Fisher-Jenks clustering to find breaks for the plotting, but this is just a built-in option in matplotlib. All we need is a threshold to call most entries basically zero, and mark which MSOAs are sufficiently far to be classed as interesting.