The Simplest Urban Models: The Basic Gravity Model

The basic model simulates flows of any kind of activity in a city – often called trips or spatial interactions. These are physical flows of transport between different locations which are referred to as origins and destinations. We subscript any variable at an origin by i and at a destination by j and any flow between an origin and destination ij by trips T_{ij}. The standard model of such a flow is based on gravitational principles meaning that as the deterrence measured by distance, travel cost or travel time between an origin and a destination, called henceforth d_{ij}, increases, the number of trips declines.

This model is usually written as

$$T_{ij} = K\frac{O_i D_j}{d_{ij}^{\beta}} = KO_i D_j d_{ij}^{-\beta} \quad . \tag{1}$$

Note that the model has two parameters – a scaling constant K which adjust the 'scale' of the model to ensure that we get it in the right units and β which is sometimes called the 'friction of distance' parameter and which adjust the deterrence function $d_{ij}^{-\beta}$ to the best possible simulation we can get of the real situation. We therefore use this parameter as a way to calibrate or fine tune the model.

The model also includes the size of the origin which we call O_i and the size of destination as D_j and as these get larger, then the number of trips also gets larger in proportion to these. So you can think of such a model as a balance between the size of two places – sometimes as in physics called the mass, that is $O_i D_j$ – and the deterrence of distance between these masses d_{ij}. The model was first proposed by Isaac Newton in the late 17th century where he wrote down the gravitational force or attraction between the planets as $F_{ij} = GP_i P_j d_{ij}^{-2}$ where the constant G is the gravitational constant, distance is assumed to be the inverse square law, and P_i and P_j are the size of the planets measured by their mass.

Now to estimate this model, we need to find β and K in the model above which ensure that the difference between the observed trips T_{ij}^{obs} and the model T_{ij} – predicted – trips are at a minimum of some sort. In fact we need to make sure the model generates no more or less than a total of T trips,

$$\sum_i \sum_j T_{ij} = \sum_i \sum_j T_{ij}^{obs} = T \quad . \tag{2}$$

Now it is easy to show that if we put the model in equation (1) above into equation (2), we get the following

$$\sum_i \sum_j T_{ij}^{obs} = \sum_i \sum_j KO_i D_j d_{ij}^{-\beta} = K\sum_i \sum_j O_i D_j d_{ij}^{-\beta} = T \tag{3}$$

and from this we can calculate K as

$$K = T / \sum_i \sum_j O_i D_j d_{ij}^{-\beta} \quad , \tag{4}$$

and the full model is thus

$$T_{ij} = T\frac{O_i D_j d_{ij}^{-\beta}}{\sum_i \sum_j O_i D_j d_{ij}^{-\beta}} \quad . \qquad (5)$$

To finally calibrate this model the best way we can do this at this point is to try different values of β and then choose one from the set of values that maximises the goodness of fit. A simple measure is the correlation coefficient between T_{ij} and T_{ij}^{obs}. I will not define the correlation other than say that this gives the percent of the model that explains the data and we want the correlation to be as large as possible.

Now the computer program we have to explore this model does five things in the following order. This is a set of workflows which are:

1. We set up an hypothetical set of zones arranged on a grid. The program asks the user to specify a number of zones on one side of the grid which might be N and it then creates a grid of N^2 zones. Try $N = 5$ or $N = 10$. More than this you can explore but as you go bigger, it takes exponentially longer to compute and if you put in 100, then this would give you 10,000 zones !

2. It then creates some hypothetical data for the grid you specify. In fact it produces an observed trip matrix

$$T_{ij}^{obs} = 1000\frac{d_{ij}^{-0.1-rnd(1)}}{\sum_i \sum_j d_{ij}^{-0.1-rnd(1)}}$$

   Now this is roughly similar to the model but it does generate hypothetical data that we can fit and is not the same as the model. We also get the observed origins and destinations activity from this equation, that is

$$\sum_j T_{ij}^{obs} = O_i^{obs}; \sum_i T_{ij}^{obs} = D_j^{obs}; \sum_i \sum_j T_{ij}^{obs} = \sum_i O_i^{obs} = \sum_j D_j^{obs} = T$$

3. The next stage is to compute the model in equation (1) or in fact in equation (5). You have to input a value for β and because we use a negative exponential function in the program, I advise putting in a value of 0.5. From the model, we then predict values of

$$T_{ij},\ \sum_j T_{ij} = O_i \text{ , and } \sum_i T_{ij} = D_j \quad .$$

   We can now compare these with the observed totals. First we compute three scatter graphs and their correlations, namely $O_i\ v\ O_i^{obs}$, $D_j\ v\ D_j^{obs}$ and $T_{ij}\ v\ T_{ij}^{obs}$

4. Then we can compare the grid maps of the same variables with each other. First $O_i\ v\ O_i^{obs}$ on the N^2 grid and then $D_j\ v\ D_j^{obs}$ on the N^2 grid.

5. Then we work out differences $(O_i - O_i^{obs})$ and $D(_j - D_j^{obs})$ and plot these.

There is a wealth of data here to compare the 'observed' (but hypothetical) data with the predicted and I will point out some issues when we discuss these in the class.

The Basic Gravity Model Program

```
%matplotlib inline
import numpy as np
import math
from matplotlib import pyplot as plt
from numpy import random
import numpy as np
plt.style.use('seaborn-whitegrid')
```

#Defined Functions: for Plotting observed and Predicted Model Results on the Grid, #the Friction of Distance Parameter, and the Location-Interactions Plots for Correlations

```
def friction(parameter,dis):
    frict=math.exp(parameter*dis)
    return frict

def locintplot(obs,pred,activities):
    maxo=np.max(obs); maxop=np.max(pred)
    if maxo>maxop:
        maxv=maxo
    else:
        maxv=maxop
    maxv=maxv*(1.1)
    plt.axis([0,maxv,0,maxv])
    plt.xlabel("Observed" + activities)
    plt.ylabel("Predicted" + activities)
    plt.title(activities + " Activity")
    plt.scatter(pred, obs, s=10,c='black')
    rO = np.corrcoef(pred, obs)
    print("Correlation ","\t","{:.3f}".format(rO[0,1]))
    return

def scattergraph(soutput,dataname,colours):
    plt.show()
    sizes=soutput**2
    plt.axis([0,n,0,n])
    plt.xlabel("x coordinate")
    plt.ylabel("y coordinate")
    plt.title(dataname + ' Activity')
    plt.scatter(xcoord, ycoord, c=sizes, s=sizes, alpha=0.75, cmap=colours)
    return
```

#Defining the Hypothetical Spatial System

```
xcoord=np.array([])
ycoord=np.array([])

n=input("Enter the Grid Size: The Number of Squares Along One Side of a Square Grid, 5 or 10, say --- ")
n=int(n); N=n*n
print("The Size of the Hypothetical Spatial System is",n, "Zones by", n,"Zones, Making", N,"in All")
print()

distance=np.full((N,N), 1.0)

n=n+1
for y in range(1,n):
    for x in range(1,n):
        xcoord = np.append(xcoord,[x])
        ycoord = np.append(ycoord,[y])

ij=0
for i in range (0,N):
    xi=xcoord[i]
    yi=ycoord[i]
    for j in range (0,N):
        ij=ij+1
        xj=xcoord[j]
        yj=ycoord[j]
        dis=math.sqrt((((xi-xj)**2)+((yi-yj)**2)))
        distance[i][j]=dis
        if distance[i][j]==0:
            distance[i][j]=0.5
```

```
        #print(i+1,j+1,distance[i][j])
```

#Defining the Hypothetical Trip, Origin and Destination Data

```
tobs=np.full((N,N), 1.0)
origins=np.full((N),1.0)
destinations=np.full((N),1.0)
differences=np.full((N),1.0)

for i in range (0,N):
    for j in range (0,N):
        tobs[i][j]=1.0/(distance[i][j]*(0.1+random.rand()))
Tobs = np.sum(tobs)
for i in range (0,N):
    for j in range(0,N):
        ij=ij+1
        tobs[i,j]=1000*(tobs[i][j]/Tobs)

origins = np.sum(tobs, axis = 1)
destinations = np.sum(tobs, axis = 0)

To=np.sum(origins)
Td=np.sum(destinations)
Tobs = np.sum(tobs)
```

#Defining and Running th Unconstrained Gravity Model

```
beta=input("Enter the Parameter Value on Distance - beta - that You Think Best Fits the Data: It should be greater than 0 and less than 1 ----
")
beta=float(beta)
print()
trips=np.full((N,N),1.0)
OPred=np.full((N),1.0)
DPred=np.full((N), 1.0)
trips1=np.full((N,N),1.0)
tobs1=np.full((N,N),1.0)

total=1000
for i in range(0,N):
    for j in range(0,N):
        trips[i][j]=origins[i]*destinations[j]/(friction(beta, distance[i][j]))
Ttrip=np.sum(trips)
for i in range(0,N):
    for j in range(0,N):
        trips[i][j]=total*((origins[i]*destinations[j])/(friction(beta, distance[i][j])))/Ttrip

#Printing the Predictions

OPred = np.sum(trips, axis = 1)
DPred = np.sum(trips, axis = 0)

print("Zone","\t","ObsO","\t","ObsD","\t","PredO","\t","PredD")
print()
for i in range(0,N):
    print(i+1, "\t","{:.2f}".format(origins[i]), "\t","{:.2f}".format(destinations[i]),"\t","{:.2f}".format(OPred[i]),"\t","{:.2f}".format(DPred[i]))
```

#Comparing Observed with Predicted Origin Activity Using Scattergraphs

```
input()
locintplot(origins, OPred, 'Origins')
plt.savefig('OriginOutputs.png', dpi=300, bbox_inches='tight')
plt.show()

input()
locintplot(destinations, DPred, 'Destinations')
plt.savefig('DestOutputs.png', dpi=300, bbox_inches='tight')
plt.show()

input()
locintplot(tobs, trips, 'Trips')
plt.savefig('TripOutputs.png', dpi=300, bbox_inches='tight')
plt.show()

rng = np.random.RandomState(0)
```

```
colors = rng.rand(N)
```

#Plotting Observed and Predicted Locations on the Hypothetical Grid

```
input()
scattergraph(origins, 'Observed Origin','winter')
plt.savefig('ObsOrigins.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()

input()
scattergraph(OPred, 'Predicted Origin','winter')
plt.savefig('PredOrigins.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()

input()
scattergraph(destinations, 'Observed Destination','winter')
plt.savefig('ObsDestinations.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()

input()
scattergraph(DPred, 'Predicted Destination','winter')
plt.savefig('PredDestinations.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()
```

#Measuring the Differences Between Predictions and Observations

```
for i in range(0,N):
   differences[i]=(origins[i]-OPred[i])*6
input()

scattergraph(differences, 'Differences in Origin ','autumn')
plt.savefig('Odifferences.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()

for i in range(0,N):
   differences[i]=(destinations[i]-DPred[i])*6
input()
scattergraph(differences, 'Differences in Destination ','autumn')
plt.savefig('Ddifferences.png', dpi=300, bbox_inches='tight')
plt.colorbar()
plt.show()

print("The model and its outputs are now complete")
print()
```

Model Inputs and Outputs

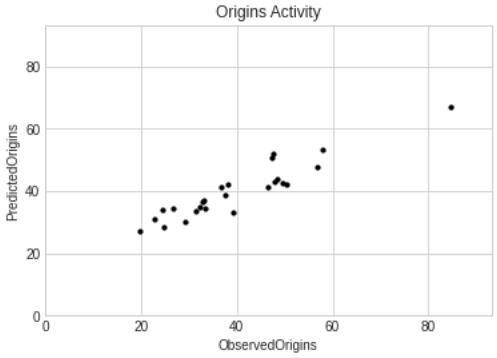
Enter the Grid Size: The Number of Squares Along One Side of a Square Grid, 5 or 10, say --- 5
The Size of the Hypothetical Spatial System is 5 Zones by 5 Zones, Making 25 in All

Enter the Parameter Value on Distance - beta - that You Think Best Fits the Data: It should be greater than 0 and less than 1 ---- 0.5

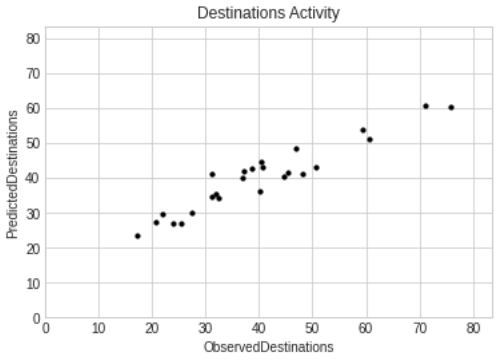
Zone	ObsO	ObsD	PredO	PredD
1	27.01	23.45	19.73	17.13
2	36.74	26.87	32.79	23.87
3	38.80	34.20	37.52	32.49
4	34.79	42.89	32.30	38.78
5	34.25	41.16	26.61	31.16
6	42.30	41.89	38.16	37.20
7	43.15	41.59	47.84	45.28
8	42.17	60.72	50.44	71.03
9	41.08	40.32	46.54	44.77

10	50.70	40.07	47.28	36.83
11	30.05	42.94	29.13	40.68
12	33.13	41.37	39.35	48.11
13	66.78	60.56	84.72	75.90
14	47.72	51.06	56.83	60.51
15	34.24	48.43	33.34	46.94
16	52.14	44.72	47.58	40.49
17	43.71	53.96	48.46	59.40
18	42.54	43.09	49.67	50.57
19	53.15	36.30	57.96	40.21
20	37.16	30.10	33.17	27.32
21	30.83	27.52	22.86	20.65
22	41.43	35.46	36.87	31.99
23	33.62	26.89	31.34	25.54
24	28.48	34.72	24.89	31.08
25	34.02	29.71	24.61	22.08

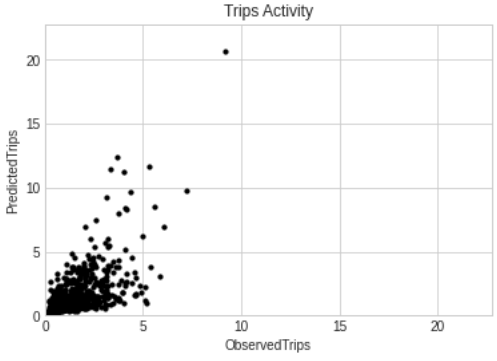
Correlation 0.933

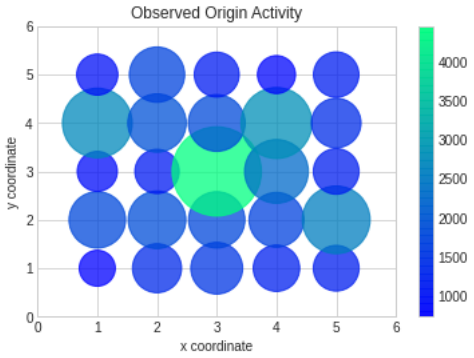


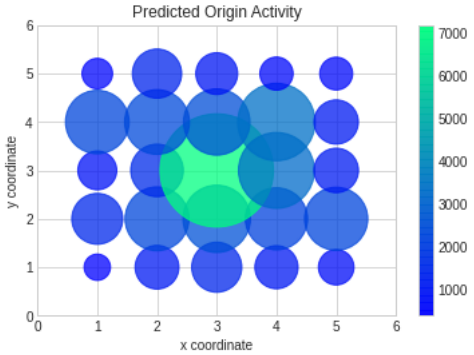
Correlation 0.953

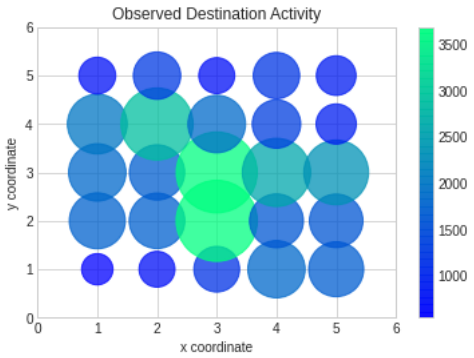


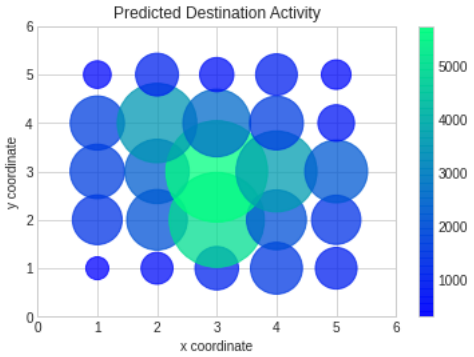
Correlation 0.890

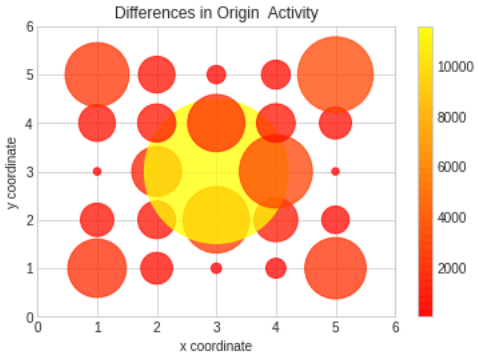


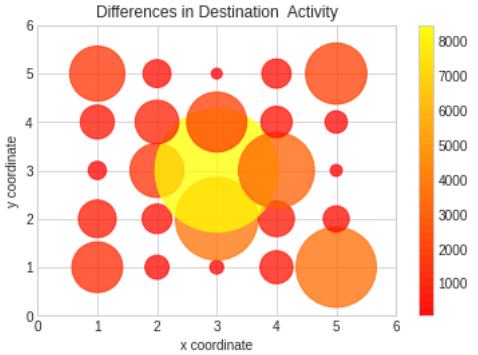

Observed Origin Activity
y coordinate
x coordinate
0
1
2
3
4
5
6
4000
3500
3000
2500
2000
1500
1000


Predicted Origin Activity
y coordinate
x coordinate
0
1
2
3
4
5
6
7000
6000
5000
4000
3000
2000
1000


Observed Destination Activity
y coordinate
x coordinate
0
1
2
3
4
5
6
3500
3000
2500
2000
1500
1000


Predicted Destination Activity
y coordinate
x coordinate
0
1
2
3
4
5
6
5000
4000
3000
2000
1000





The model and its outputs are now complete