LBA_Group_Assignment

April 12, 2020

1 LBA | Model Analysis Report

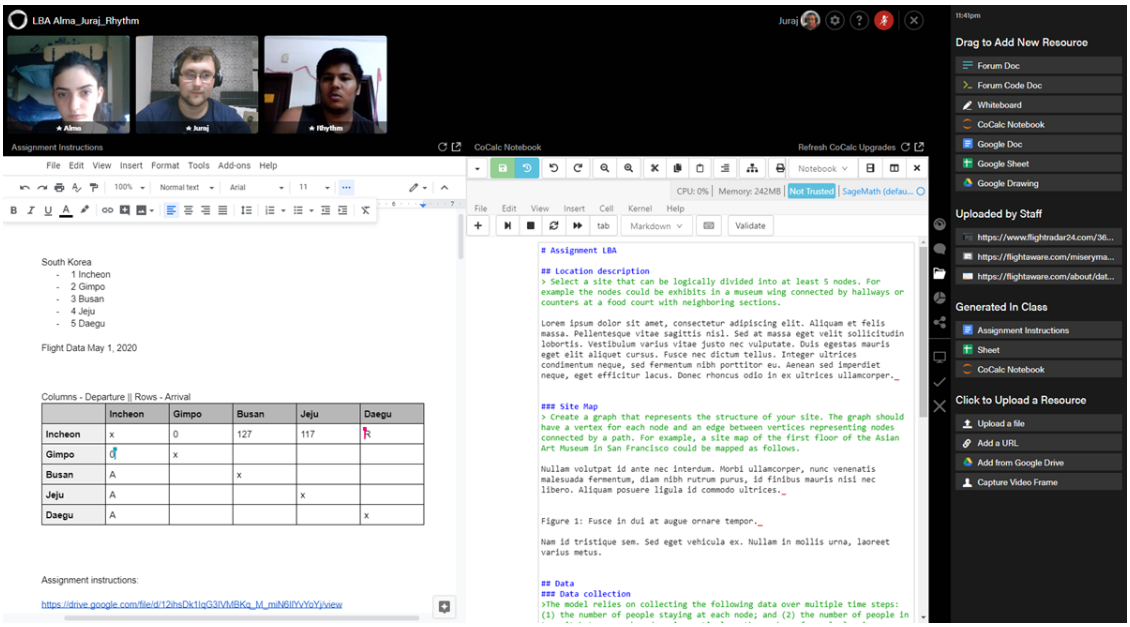
1.1 Location description

> A short description of the network you chose and a photo(s) of you and your group-mates at the location(s). (200 words)

We chose to see the movements of airplanes domestically in Korea for three alternate days, July 13 (Monday), 15 (Wednesday), 17 (Friday) 2020.

We considered the five biggest airports in South Korea; Incheon Int'l Airport, Gimpo Airport, Daegu Airport, Busan Airport and Jeju Airport. These represent the nodes in the system, while the edges between them represent the number of flights back and forth per day. For instance, the edge that connects Incheon to Jeju has 125 flights in 24h, and the edge that connects Jeju to Incheon has 117.

Network fetaures: Our network has 18 edges out of 20 possible. There is no connection between Incheon and Gimpo airport as they are close in physical distance. Thus, their network distance is the longest, as we need to pass through another node to go from Gimpo to Incheon or the other way around (assuming we are restricted to using airports). The robustness is 0.9 (see formula in the Appendix) which shows that the network is very interconnected. This means that there are many movements and connections in the network we chose.



1.1.1 Site Map

An appropriately labeled diagram illustrating the nodes and edges of your network

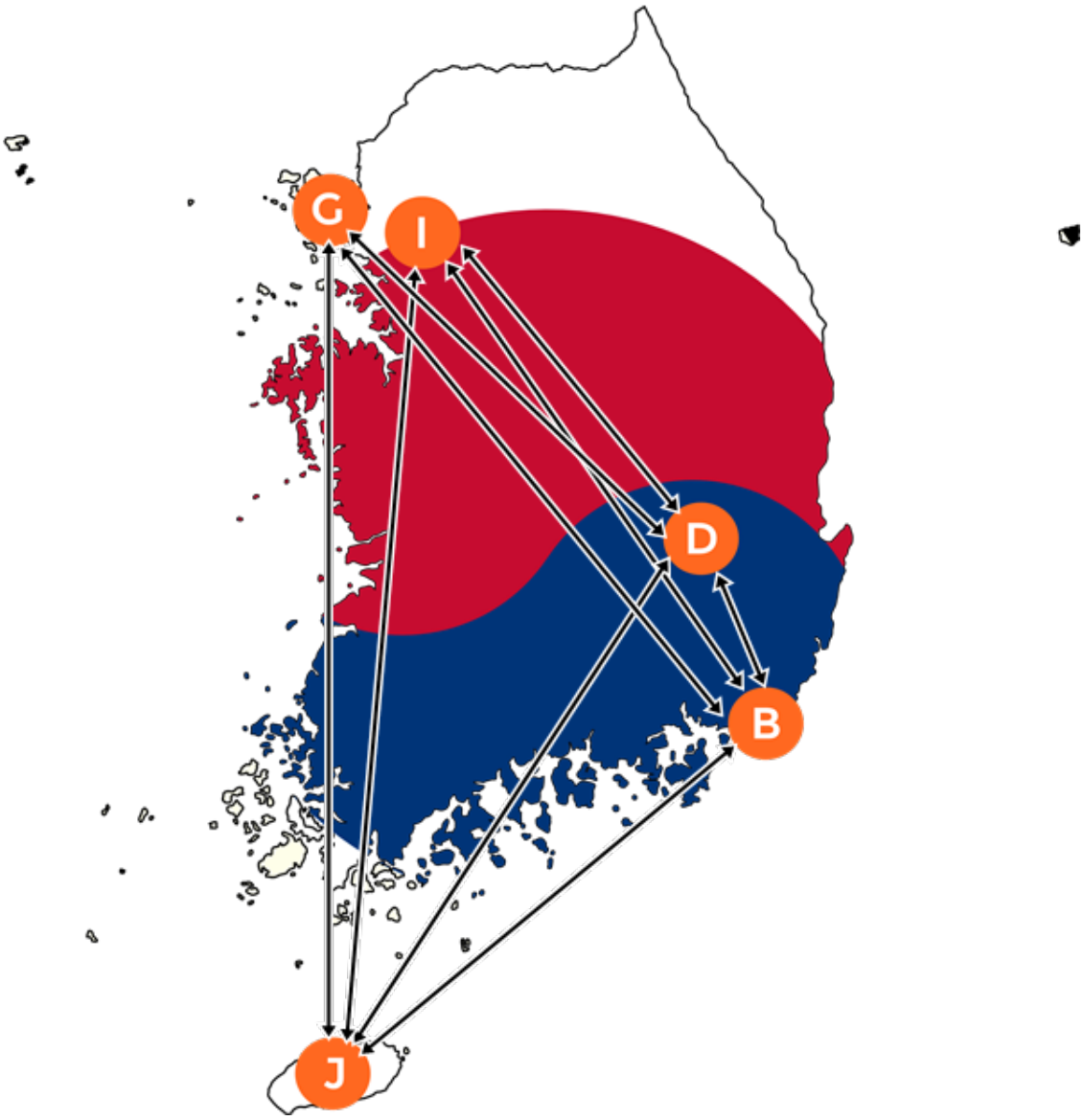


Figure 1. Free map of South Korea displaying real locations of the airports.

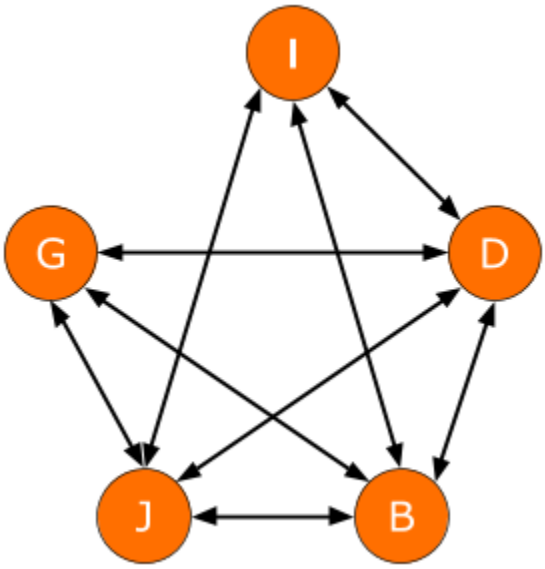


Figure 2. Network Graph

1.2 Data Collection

A short summary of your data collection procedure including your estimation process and any assumptions you made. (200 words)

Data is collected through Google Flights for the future dates, concretely for dates in July for the Korean Airports. Instead of the movement of people, we choose to examine the movement of aircrafts. We manually counted the number of flights between each permutation. We assume that these airport mainly function domestically, so we do not account for the number of international flights. We also assume that each flight is a direct flight. There is no available data to show how many airplanes visit the aiport (node) per day, so we calculate it from the given information. Specifically, we add the incoming flights and the departures, assuming that no same plane is returning to the airport in one day (e.g. if FLIGHT AX199 takes off from Busan today, it will not return on the same day back again). On the contrary, along the diagonal of the matrix, we show the difference between arrivals and departures as the number of aircrafts standing at the airport without flying/moving at all in a day. Consequently, we find the number of all airplanes that visit an aiport for a day and the number of airplanes that do not make any trips per day. We decided to examine the number of flights in 24h for 3 days (beggining-middle-end of the week), as it is practically impossible to find flights in a smaller unit of time (such as dividing into hours). Thus, our model works well for weekdays but may be limiting for weekends as we haven't integrated data from weekends.

1.3 Analysis of Model

An analysis of your model In particular, describe any relationships you found between multiplying powers of M by initial distributions and the eigenvectors (if any). Interpret your results in terms of how your system evolves over time. (Note that for ease of analysis, we make the unrealistic assumption that your network is a roughly closed system. In other words, not many people are entering or exiting the museum over your

given time interval. Feel free to try to propose ways to handle this otherwise.(300 words)

We see that from the initial data, the highest flow of traffic happens in Gimpo airport, represented by node G, or the second row in the matrix. After we multiply M in the power of 10 (that represented 10 time steps - days in our case) with starting distribution vectors (uniform spread, edge case with all planes on one airport and randomly choosen values) we get probability values that all get close towards same values; the highest probability for flights destination is still Gimpo Airport. Similarly, the ratio between flights and airports in the collected data is successfully represented with any set of initial conditions after long period of time. This may suggest that our model is insensitive to initial conditions. No matter if Gimpo airport starts the day with less aircrafts, it still ends up with the highest rate of traffic. We see a similar pattern when the eigenvalue is close to 1, but a slightly different pattern when the eigenvalue is more different from 1. Thus over time, the probability does not change.

Another important observation is that when the eigenvalue is 1, the corresponding normalized eigenvector is similar to what we get most of the time after multiplying M in the powers of 10 or 20 with the initial distributions. This is consistent with the class learnings that the eigenvector predicts the long-term behavior of the model, regardless of initial distribution.

Similar patterns are observed in the stochastic simulation model we created in the Appendix A. We see that when we generate random initial distributions and random airplane traffic based on our matrix for 50 days, the average still shows the highest traffic rate in Gimpo, the lowest in Incheon and the probabilies computed from the simulation are very close to both the eigenvector and values obtained by multipling many times by M. Simulation ilustrates how the probablities in our system work. Our obtained probabilies does not mean that every day the distribution of the airplanes on airports will be the same, but it will variate around the same average.

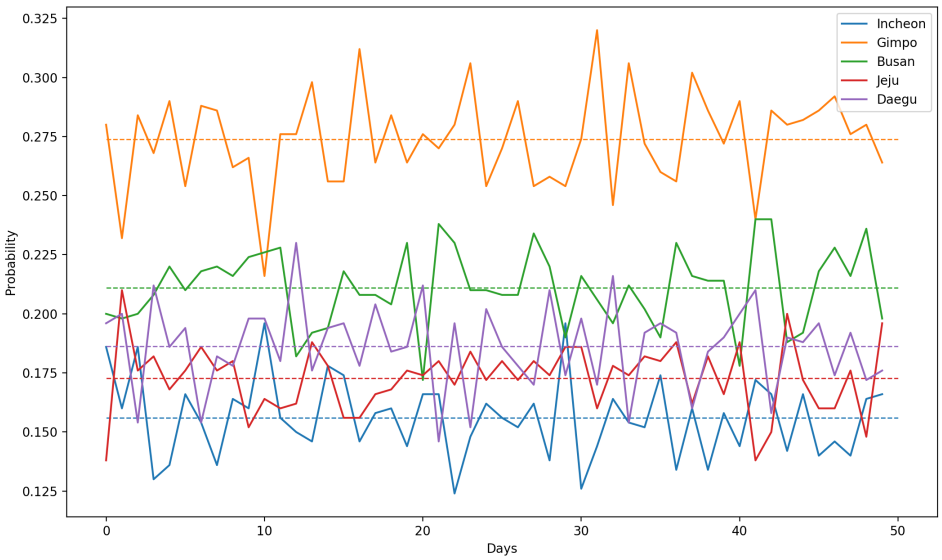


Figure 3: Results of simulation running with 500 airplanes for 50 days. Dashed lines are showing

the average value.

What this says about our prediction is that from the small samples we have given the model, the highest traffic will always be predicted in Gimpo. In the South Korean context, this holds true as Gimpo Airport is a strictly domestic airport in Seoul, which is the capital of the country with a dense travelling population. Therefore, most of the domestic flights happen from Seoul. Incheon Int'l Airport is also located nearby Seoul, and has more flights in general, but it doesn't operate domestically as much as the other airports.

Limitations of the model:

1. Rounding during normalization: If we set the rounding number to more than 3 we get more precise and slightly different results. In a payoff between more concise matrices and more precise calculations, we choose a middle ground of rounding up to 3. If at some point in time, it is more important to know the numbers with the highest accuracy, we can change the function into "no rounding" and it will output all the given values with many decimal digits.
2. Our model works only domestically and for week days. We assume that our model is a closed system, although aircrafts are machines that fail sometimes, and new aircrafts come into the system consistently. However the trend remains unchanged, even with such minor changes.
3. The model is built on data from one season. We know that during particular times of the year, there are different traffic flows in the airport, for example, during Christmas. Thus this model may not predict as well the "festive season", or other seasons.
4. The simulation in Appendix A is a stochastic model, that gives different results on every run. It is not good for predicting the short-term patterns, but we can see the long term trend, that is stable.

2 APPENDIX A

An appendix showing your matrices, eigenvalues, eigenvectors, and results of your calculations. Although not required, it may be useful to include diagrams that portray how your system is forecasted to evolve over time.

Flight day: July 13		Incheon	Gimpo	Busan	Jeju	Daegu
	SUM	**362**	**410**	**400**	**298**	**345**
Incheon	**301**	61	0	104	100	97
Gimpo	**410**	0	0	143	151	116
Busan	**412**	124	136	12	35	117
Jeju	**330**	113	169	33	32	15
Daegu	**362**	125	105	120	12	17

Figure 4. The map of information for July 13.

Flight day: July 15		Incheon	Gimpo	Busan	Jeju	Daegu
	SUM	**369**	**630**	**555**	**354**	**498**
Incheon	**377**	8	0	127	117	133
Gimpo	**691**	0	61	269	184	238
Busan	**522**	116	252	33	41	113
Jeju	**322**	125	145	38	32	14
Daegu	**494**	128	233	121	12	4

Figure 5. The map of information for July 15.

Flight day: July 17		Incheon	Gimpo	Busan	Jeju	Daegu
	SUM	**344**	**743**	**508**	**357**	**466**
Incheon	**337**	33	0	135	79	123
Gimpo	**691**	0	52	243	225	223
Busan	**491**	106	245	17	43	97
Jeju	**434**	104	270	37	77	23
Daegu	**465**	134	228	93	10	1

Figure 6. The map of information for July 17.

This is the representation of the above information in Matrix Form. The columns represent the departure country and the rows represent the destination country. The numbers along the diagonal represent the number of airplanes not moving from the airport for the given day (this is computed by *number of departures-number of arrivals*).

[1]:
```
#        Columns: Incheon, Gimpo, Busan, Jeju, Daegu
A_july_13 = matrix(QQ, [[ 61,  0,104,100, 97],
                        [  0,  0,143,151,116],
                        [124,136, 12, 35,117],
                        [113,169, 33, 32, 15],
                        [125,105,120, 12, 17]])

A_july_15 = matrix(QQ, [[  8,  0,127,117,133],
                        [  0, 61,269,184,238],
                        [116,252, 33, 41,113],
                        [125,145, 38, 32, 14],
                        [128,233,121, 12,  4]])
```

```
A_july_17 = matrix(QQ, [[  33,  0,135,79,123],
                        [  0, 52,243,225,223],
                        [106,245, 17, 43, 97],
                        [104,270, 37, 77, 23],
                        [134,228, 93, 10,  1]])
```

Form a new matrix M by normalizing each of the columns - divide each entry by the sum of all the entries in the column. This has the effect of turning the entries of the column into probabilities - the probabilities that a visitor will stay at a node or travel to another specific node.

[2]:
```
# normalize each matrix by columns sum
def normalize_by_col(M):
    output_matrix = []
    for column in M.transpose():
        suma = 0
        output=[]
        for item in column:
            suma += item
        for item in column:
            output.append(round(item/suma,3))
        output_matrix.append(output)
    return matrix(output_matrix).transpose()

# take the average from the three data samples and find the probabilities

def average_and_normalize_by_col(M1, M2, M3):
    output_matrix = []
    for col in range(M1.transpose().ncols()):
        suma = 0
        output=[]
        for index in range(M1.ncols()):
            suma += M1.transpose()[col][index]  + M2.transpose()[col][index] +␣
 ↪M3.transpose()[col][index]

        for index in range(M1.ncols()):
            output.append(round((M1.transpose()[col][index] + M2.
 ↪transpose()[col][index] + M3.transpose()[col][index])/suma,3))

        output_matrix.append(output)

    return matrix(output_matrix).transpose()
```

This represents the probabilities of moving from one node to the other. For example the probability of moving from Incheon (node 1) to Jeju (node 2) is 29.1%. The values along the diagonal represent the probability of staying in the current node. For instance, if one airplane is at Daegu (node 5),

the probability of staying in Daegu is 1.7%. Of course that we know that airplanes have a fixed schedule, but if we have no information about the flights in one day, then those probabilities would be useful.

[3]:
```
# the normalized averaged matrix
A_norm = average_and_normalize_by_col(A_july_13, A_july_15, A_july_17)
show(A_norm)

# label matrix as M
M=A_norm
```

[3]:
```
[0.087   0.0  0.24 0.257 0.265]
[  0.0  0.06  0.43 0.487 0.434]
[0.294 0.334 0.041 0.103 0.246]
[0.291 0.308 0.071 0.123 0.039]
[0.329 0.299 0.219  0.03 0.017]
```

Use computational software to find the eigenvalues and corresponding eigenvectors of your matrix M.

[4]:
```
# find eigenvalues
A_norm.eigenvalues()
```

[4]:
```
[1.0008269399475842,
 -0.6371442270925228,
 -0.20572774269884983,
 0.07051200227461407,
 0.09953302756917433]
```

[5]:
```
#  find (eigenvalue, eigenvectors forming a basis for that eigenspace,␣
 ↪algebraic multiplicity of the eigenspace).
A_norm.eigenvectors_right()
```

[5]:
```
[(1.0008269399475842,
  [(-0.3466797595945687, -0.5961954002169405, -0.45973074601856395,
-0.37955460603223734, -0.41103543795218206)],
  1),
 (-0.6371442270925228,
  [(-0.3753864731175977, -0.6903910103089785, 0.3062779265027889,
0.37510214873941694, 0.384626388113327)],
  1),
 (-0.20572774269884983,
  [(-0.015386704399187188, 0.06783636759292598, -0.7463480224452429,
0.03282924256344241, 0.6610963218656685)],
  1),
 (0.07051200227461407,
  [(-0.6712208323293518, 0.6157924662256195, 0.2407852493185623,
-0.31084103795905066, 0.12515009236112912)],
```

```
  1),
 (0.09953302756917433,
  [(-0.48367471381403876, 0.5083876920226831, -0.19269101724115498,
0.5619290454113492, -0.39332765681352205)],
  1)]
```

Now that you have a stochastic model, choose three different starting distributions of visitors at your nodes. In particular, create three column vectors with the same number of entries as the number of your nodes and whose entries sum to one. For each of these vectors, repeatedly multiply by your matrix M until you notice a pattern emerging. More specifically, you are trying to observe long term behavior ($\lim_{n\to\infty} M^n v$), if it exists, for each of your initial distributions v. Note any relationship with the eigenvectors you found. (You may need to scale the eigenvectors to be length 1 in order to make a proper comparison.)

[6]:
```
# label eigenvectors

eigenvector1 = matrix([[-0.3466797595945687], [-0.5961954002169405], [-0.
↪45973074601856395], [-0.37955460603223734],[-0.41103543795218206]])

eigenvector2 = matrix ([[-0.3753864731175977],[-0.6903910103089785],[0.
↪3062779265027889],[0.37510214873941694],[0.384626388113327]])

eigenvector3 = matrix ([[-0.015386704399187188],[0.06783636759292598],[ -0.
↪7463480224452429,],[0.03282924256344241],[0.6610963218656685]])

eigenvector4 = matrix ([[-0.6712208323293518],[0.6157924662256195],[0.
↪2407852493185623],[ -0.31084103795905066],[0.12515009236112912]])

eigenvector5 = matrix ([[-0.48367471381403876],[ 0.5083876920226831],[-0.
↪19269101724115498],[0.5619290454113492,],[-0.39332765681352205]])
```

[7]:
```
# normalized eigen vectors 1 to 5
show(normalize_by_col(eigenvector1))
show(normalize_by_col(eigenvector2))
show(normalize_by_col(eigenvector3))
show(normalize_by_col(eigenvector4))
show(normalize_by_col(eigenvector5))
```

[7]:
```
[0.158]
[0.272]
[ 0.21]
[0.173]
[0.187]
```

[7]:
```
[-1639.386]
[-3015.072]
```

```
[ 1337.575]
[ 1638.144]
[ 1679.738]
```

[7]:
```
[   -565.58]
[  2493.509]
[-27434.043]
[  1206.728]
[ 24300.386]
```

[7]:
```
[ 2009.268]
[-1843.346]
[ -720.779]
[  930.488]
[ -374.631]
```

[7]:
```
[-775.929]
[ 815.574]
[-309.122]
[ 901.467]
[-630.991]
```

[8]:
```
# multiply eigenvectors with M

show(M*eigenvector1)
show(M*eigenvector2)
show(M*eigenvector3)
show(M*eigenvector4)
show(M*eigenvector5)
```

[8]:
```
[-0.34696644293679607]
[ -0.5966884180099455]
[ -0.4601109157375797]
[ -0.3798684748982555]
[  -0.411375339575698]
```

[8]:
```
[ 0.23917532427550015]
[  0.4398786466549405]
[-0.19514321275711985]
[-0.23899416863932027]
[-0.24506248277385442]
```

[8]:
```
[0.0031654719636192707]
[-0.013955822777782345]
[   0.1535444939254103]
[ -0.00675388596709002]
[  -0.1360058540039363]
```

[8]:
```
[ -0.04732912485597545]
[ 0.043420759779191305]
[  0.01697825004764413]
[-0.021918023975611798]
[ 0.008824583597236316]
```

[8]:
```
[ -0.04814160862456515]
[  0.05060136616592281]
[-0.019179120331396043]
[  0.05593049916884771]
[ -0.03914909250933884]
```

[9]:
```
# first starting initial distribution and first 10 steps

dist_vector1 = vector([0.2,0.2,0.2,0.2,0.2])
for value in range(11):
    print "M^"+str(value),(M**value)*dist_vector1
```

```
M^0 (0.2, 0.2, 0.2, 0.2, 0.2)
M^1 (0.1698, 0.2822, 0.2036, 0.16640000000000002, 0.17880000000000001)
M^2 (0.1537834, 0.263116, 0.2136476, 0.1782254, 0.192862)
M^3 (0.16156693760000002, 0.2781533058000001, 0.20765388340000002, 0.1704030192,
0.184680663)
M^4 (0.1566272072166, 0.26911804630240005, 0.2119006470866, 0.175592739968,
0.1900512232162)
M^5 (0.1599736306526972, 0.2752602562656289, 0.2094604060451218,
0.1725217262258142, 0.1869015417243598)
M^6 (0.1580551955146035, 0.2717169397556838, 0.21130456671791434,
0.17471350673197739, 0.1891589480281306)
M^7 (0.15949239047964275, 0.2743444412967259, 0.21041376500344008,
0.17355246387760243, 0.18797633174091477)
M^8 (0.15879185270044083, 0.2730403633132322, 0.21126685194692052,
0.17452778085905263, 0.18898477050720047)
M^9 (0.15945353951738395, 0.27424158781445346, 0.21100884194362662,
0.17414213061997943, 0.18859760326987104)
M^10 (0.15924747244033333, 0.2738468747356807, 0.21145904332607318,
0.1746238054181903, 0.1890998088175823)
```

[10]:
```
# second starting initial distribution and first 10 steps

dist_vector2 = vector([1,0,0,0,0])
for value in range(11):
    print "M^"+str(value),(M**value)*dist_vector2
```

```
M^0 (1.0, 0.0, 0.0, 0.0, 0.0)
M^1 (0.087, 0.0, 0.294, 0.291, 0.329)
M^2 (0.240101, 0.41092300000000004, 0.148539, 0.09481499999999998, 0.107332)
M^3 (0.10934858199999999, 0.181284143, 0.250097692, 0.22282813699999998,
```

```
0.23905834099999998)
M^4 (0.19015406428800002, 0.33068767885300004, 0.184711042239, 0.142144025688,
0.15569987269)
M^5 (0.138665534595082, 0.23606489415146598, 0.226871135696977,
0.19385713198803498, 0.20879923998450997)
M^6 (0.17166605559386677, 0.2967457754302385, 0.20024695601207684,
0.16115510619422996, 0.17525444399008597)
M^7 (0.15085350622885474, 0.25845390301929466, 0.21750460369435845,
0.18222705626468974, 0.19687318124432662)
M^8 (0.16432910641832824, 0.2832217508306736, 0.2067924125725638,
0.16903698129392453, 0.18335568453022108)
M^9 (0.15595857186885706, 0.2678114194323, 0.21390361844761716,
0.1776767509120643, 0.1922232739392243)
M^10 (0.16150735675831365, 0.27800071968221207, 0.20965851330917631,
0.1724079665545803, 0.1868289751794686)
```

[11]:
```
# third starting initial distribution and first 10 steps

dist_vector3 = vector([0.1,0.6,0.1,0.1,0.1])
for value in range(11):
    print "M^"+str(value),(M**value)*dist_vector3
```

```
M^0 (0.1, 0.6, 0.1, 0.1, 0.1)
M^1 (0.0849, 0.1711, 0.2688, 0.2372, 0.2389)
M^2 (0.1961672, 0.345049, 0.1763298, 0.13498220000000002, 0.14913549999999998)
M^3 (0.1335970313, 0.22698589240000006, 0.2307395442, 0.19829825809999999,
0.2129096555)
M^4 (0.17438414337029998, 0.3018111997317, 0.19735163241329998, 0.1578650609165,
0.17192257719159998)
M^5 (0.1486666158637226, 0.2544645570891109, 0.2187183510537344,
0.18383798414266708, 0.19849257502687187)
M^6 (0.16527329413982664, 0.2849916402175937, 0.20543108534820578,
0.16751935420020062, 0.18178505136931356)
M^7 (0.15490774971605395, 0.26591150290256393, 0.21417384692853358,
0.17815205841145892, 0.1926927483876049)
M^8 (0.161727354822605, 0.27843814960002433, 0.20889052622977713,
0.17161296156501332, 0.18599659997673026)
M^9 (0.1575976362807551, 0.2708272519268681, 0.21254199449514685,
0.17601509936408882, 0.19051866275792032)
M^10 (0.16044439920268064, 0.27604714577577394, 0.21050137525736648,
0.1734462724296723, 0.18789293770474644)
```

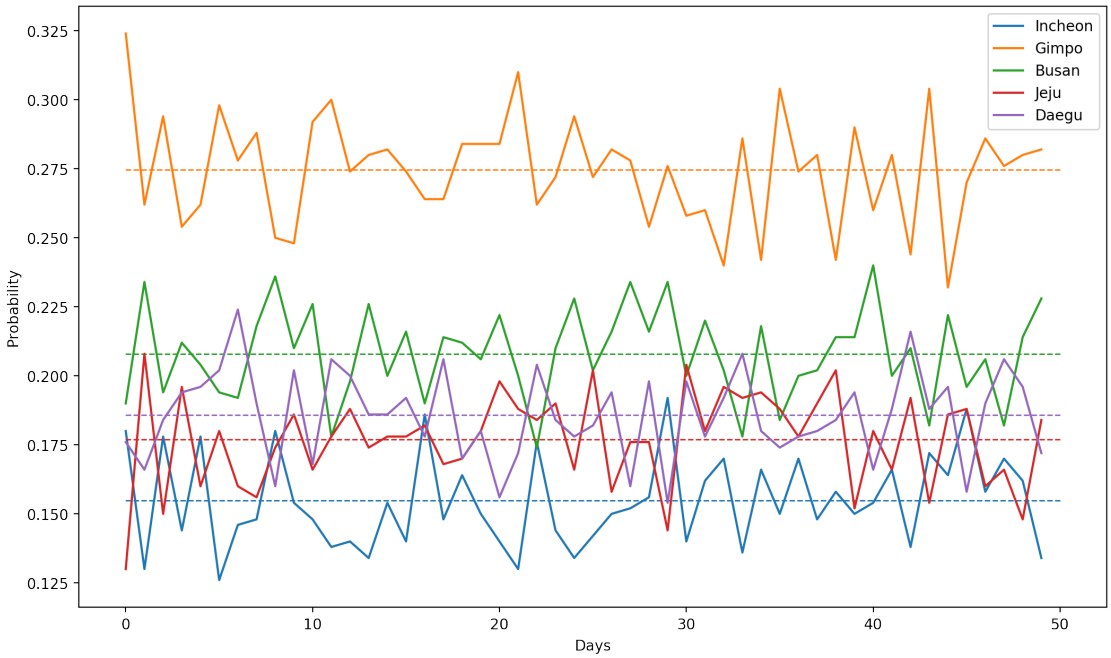
[46]:
```
# Third method to simulate the traffic and verify the probabilities.

import random
import matplotlib.pyplot as plt
```

```
def simulate_airtraffic(M, start_dist, days = 5, names=None):
    """
    Fuction simulates air traffic based on the given probabilities in matrix M

    Inputs:
    M - matrix with probabilities of going to airport on each row
    start_dist = if integer, then generates the random starting distribution␣
 ↪with givent start_dist as
                 maximum number of airplains in the system
                 if list or array-like object, expecting to be specific␣
 ↪starting distribution that will be used.
    days - integer, how many days should the simulation simulate. Default 5 days
    names = list of names of airports in order

    Prints the results and substeps.

    Returns:
        Boolean: True
    """
    if names == None:
        names = list(range(M.ncols()))

    if type(start_dist)==type(1):
        print "Generating the starting distribution"
        # genetate random the starting distribution
        m = M.ncols() # number of airports
        n = start_dist # number of airplanes in the system
        start_dist = [0] * m;
        for i in range(n) :
            start_dist[random.randint(0, n) % m] += 1;
        print "Using starting distribution:", start_dist

    # List of planes contains value from 0 to n, where n is number of airports
    # This value represents the current airport
    print("Generating the planes")
    list_of_planes = []

    # sprading the planes on initial position
    for airport in range(len(start_dist)):
        planes = [airport]*start_dist[airport]
        list_of_planes += planes

    # creating variables for storing data about the statistics
    # and for plots
    average = [0]*M.ncols()
    data = []
    for i in range(M.ncols()):
```

```
        data.append([])

    for day in range(1,days+1):
        # print "------->  Simulating day ",day , " <-------"
        updated_list = []
        for plane in list_of_planes:

            # generate transpose, becasue sage is reading matrix by rows
            probabilities = M.transpose()[plane]

            rand_value = random.random()
            for stage in range(len(probabilities)):
                if rand_value <= probabilities[stage]:
                    updated_list.append(stage)
                    break
                else:
                    rand_value -= probabilities[stage]
        list_of_planes = updated_list

        # print some numbers
        #print "STATS:",
        for airport in range(len(start_dist)):
            average[airport] += list_of_planes.count(airport)
            data[airport].append(list_of_planes.count(airport)/
↪(sum(start_dist)))
            # print names[airport], ":", list_of_planes.count(airport),
        #print ""
    print "------------TOTAL STATS------------"
    plt.figure(figsize=(2500/200, 1500/200), dpi=200)
    for airport in range(len(start_dist)):
            print names[airport], ":", round(average[airport]/
↪(sum(start_dist)*days),3),
            a = plt.plot(range(len(data[airport])),data[airport], label=␣
↪names[airport])
            plt.plot([0, len(data[airport])], [average[airport]/
↪(sum(start_dist)*days), average[airport]/(sum(start_dist)*days)], color=a[0].
↪get_color(), linestyle='--', linewidth=1)

    plt.xlabel("Days")
    plt.ylabel("Probability")
    plt.legend()
    plt.show()
    #plt.savefig('mychart.png')

a_names = ["Incheon", "Gimpo", "Busan", "Jeju", "Daegu"]
simulate_airtraffic(M,500, days=50, names=a_names)
```

```
Generating the starting distribution
Using starting distribution: [90, 87, 112, 100, 111]
Generating the planes
------------TOTAL STATS------------
Incheon : 0.155 Gimpo : 0.275 Busan : 0.208 Jeju : 0.177 Daegu : 0.186
```

[46]:



network robustness: `edges = 18`

`nodes = 20`

`network = edges / nodes(nodes-1)`

3 Appendix B

#networks: Graphed and analyzed a network, of airports as nodes and flights as connections, using real-life data to build a mathematical model. We computed the network properties and explained the interactions between the nodes. Applied probabilities and explained them contextually.

#probability: Normalized columns of matrix to calculate probabilities of visitors' next state. By using normalization we ensure to represent the given information of existing data in probability form that further helps our future predictions.

#modeling: Built a mathematical model to run simulations of a network of airports and flights between them, with real-life data and generated probabilities as predictions. Developed the model appropriate to the context supplemented by the explanation in Model Analysis.

#breakitdown: Decomposed the problem in tractable parts, and solved it step-by-step merging all the solutions at the end and finding the same results in a comparative model.

#context: Situated the mathematical concepts in the context of South Korea air traffic, making sure each numerical calculations is briefly explained in the model and interpreted in the given scenario with local context.

[0]: