CSYS5020 - Assignment 3

Kilian Liss

June 8, 2021

1 Transport Networks

Since American data was the easiest to obtain, we provide visualizations of various transport networks in the United States of America. All visualizations will include coastline data which was downloaded from [1].

1.1 Flights

The flight data downloaded from [5] contains information about various flight routes within the USA. Each entry has an IATA code of the airport of origin and destination, along with number of flights on that particular route for a given year. The dataset [4] was used to map the IATA codes of airports to their longitudinal and latitudinal coordinates.

1.1.1 Actual Flight Traffic

Since the number of flights on a given route is provided, a map of the actual air traffic is provided in figure 1a, which is useful for comparing our estimates with. Airports (nodes) are represented by purple dots, and their dot size corresponds to the number of arrivals and departures per year. Flight routes are represented by golden lines, where the line width and opacity corresponds to the number of flights per year. Coastlines are represented in blue.

1.1.2 Estimated Flight Traffic

In estimating the flight traffic, we completely ignored the number of flights on a given route data attribute, but based the trip distribution on network topology alone. We considered the degrees of nodes (airports) of origin O_i and destination D_i for each route. The trip distribution was then calculated as:

$$t_{ij} = \sqrt{\frac{O_i D_j}{d}} \tag{1}$$

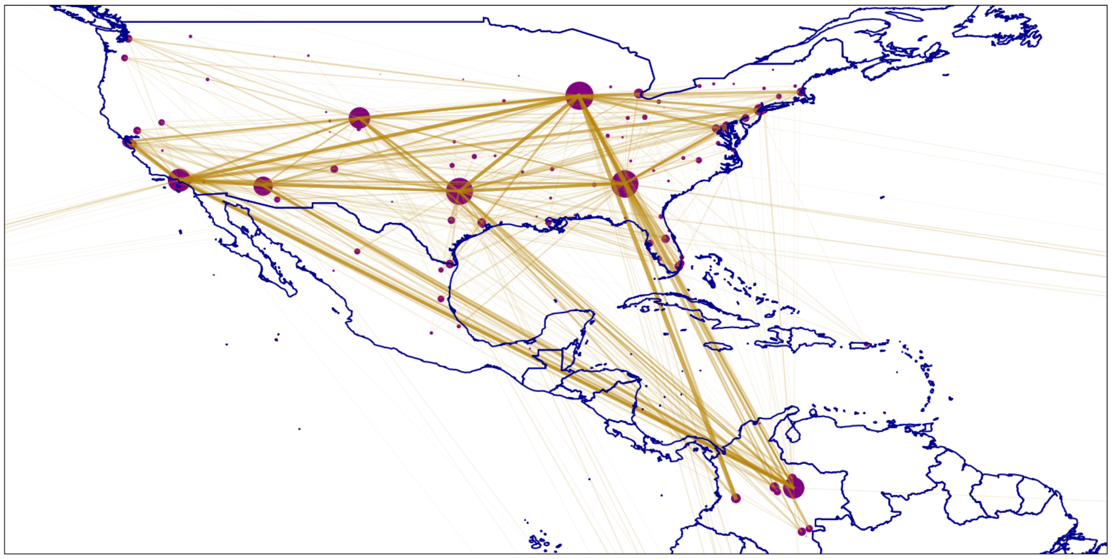
where d is the flight distance given by:

$$d^2 = (x_o - x_d)^2 + (y_o - y_d)^2 + (z_o - z_d)^2 \tag{2}$$

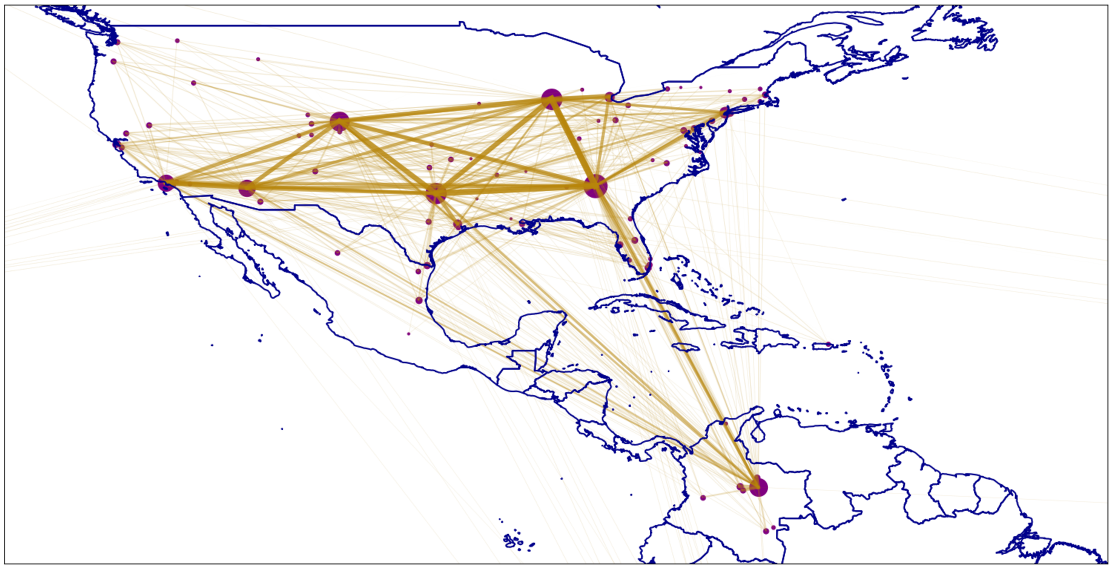
where subscripts o and d represent origin and destination. Cartesian coordinates are calculated by the transformation:

$$\begin{aligned} x &= r\cos\theta\sin\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos\phi \end{aligned} \tag{3}$$

where θ and ϕ are longitudinal and latitudinal coordinates, and $r = 6371$ is earths radius in kilometres.



(a) Map of actual flight traffic. Airports (nodes) are adjusted in size based on number of arrivals and departures. The opacity and line width of flight routes (edges) are based on the number of flights per year on that particular route.



(b) Map of estimated flight traffic

Figure 1: Actual flight traffic and estimated flight traffic in the USA

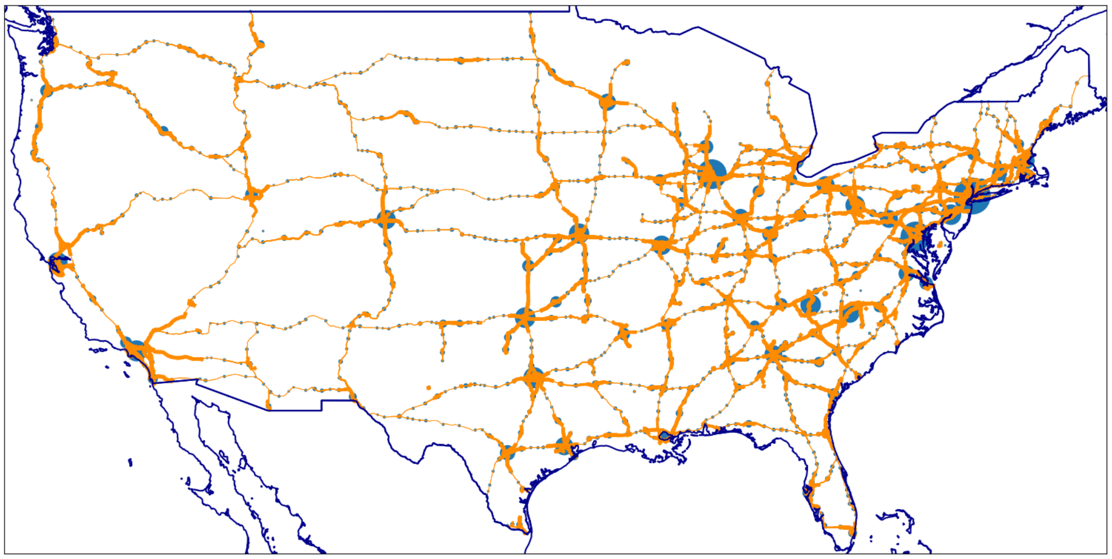


Figure 2: Map of estimated road traffic

Results are shown in figure 1b, where node size is modulated by its degree. Opacity and and line width is modulated by our traffic estimate t_{ij}.

1.1.3 Difficulties in Estimating Air Traffic

Our air traffic data contains primarily routes within the USA, and little information about international flights. This means the importance of some airports is misrepresented by degree count, since some international routes are not included in the count.

In comparing our actual and estimated air traffic maps, we can see that our method does a good job at estimating domestic flights, but fails at estimating traffic for routes that have international significance. This is clearly seen in flight routes between Colombia and mainland America, where many of the actual flight paths are significantly busier compared to our estimate.

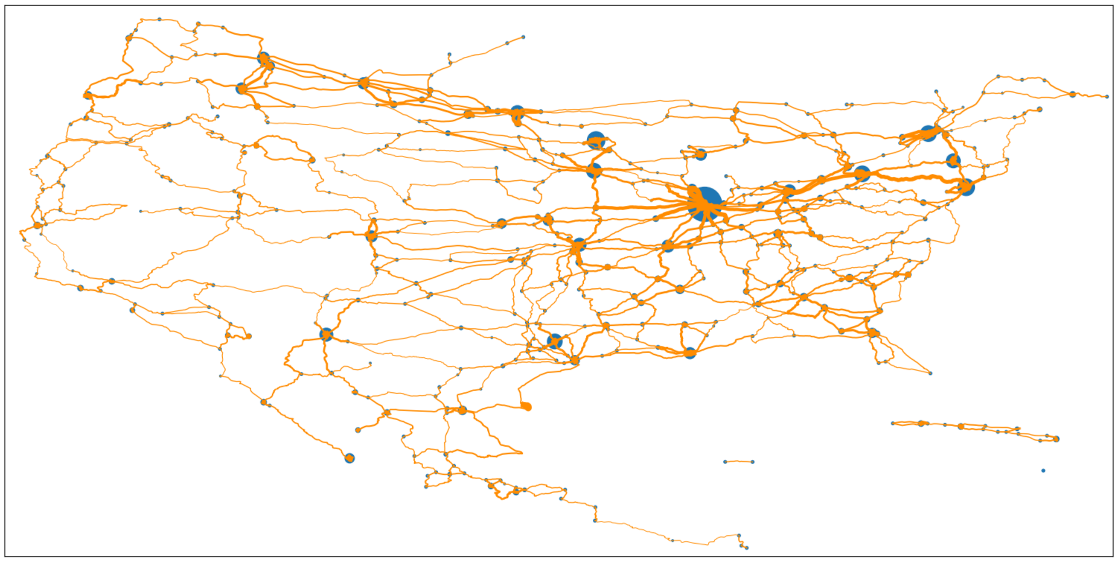
Additionally, there are too many other factors affecting flight traffic and we can not consider all factors in modelling such a complex system. These factors may include wealth in areas nearby airports, seasonal or economical factors, and others.

1.2 Road Traffic

The roads data set downloaded from [2] contains lists of longitudinal and latitudinal coordinates for each major road in the USA. Figure 2 shows a map of those roads, where line thickness indicates our traffic estimates for each particular road. Dots represent cities, and the size of each dot represents our estimate of the size of each city.

Since information about the cities was not included in this data set, we calculated an estimate of the location and size of cities based on road network topology. To do so, we created a list of beginning and end coordinates of each road, and used the *sklearn.cluster.DBSCAN()* clustering algorithm on these coordinate. The size of each cluster was used to estimate the size of each city. Here we made the assumption that at the end of each major road lies a city, and the more major roads end up in the same area, the larger a city is. After comparing results with google maps, this assumption seems to be valid.

To estimate the traffic on each road, we considered the cities as nodes of a network, and roads as edges. A similar approach was then used as for estimating air traffic, with the only difference being


(a) Map of estimated railway traffic

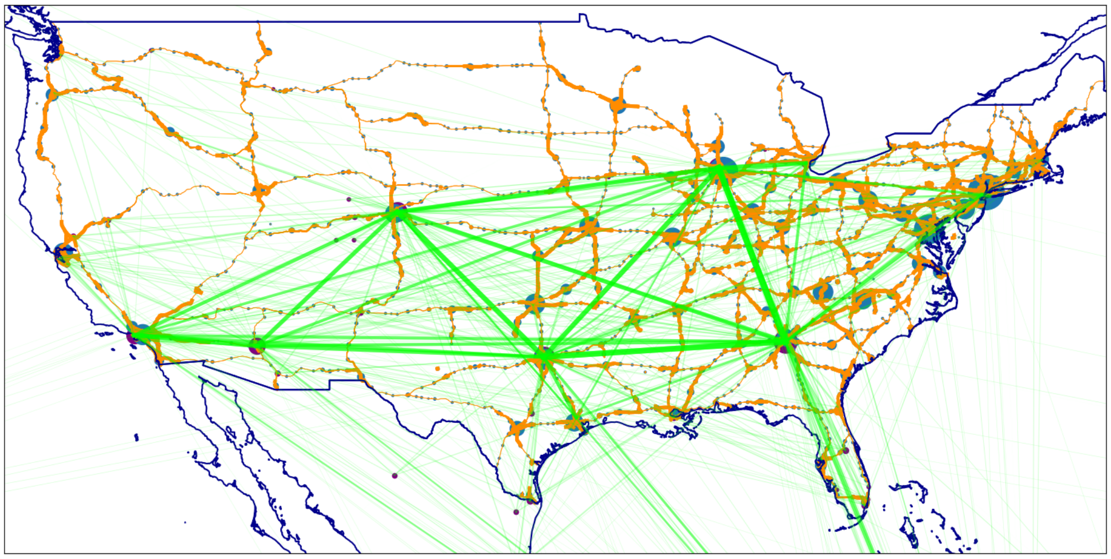

(b) Map of estimated flight traffic combined with estimated highway traffic

Figure 3

that the trip distribution was calculated by:

$$t_{ij} = \frac{O_i D_j}{\sqrt{d}} \tag{4}$$

1.2.1 Difficulties in Estimating Road Traffic

The main issues in estimating road traffic came from the fact that we only considered traffic from one end of a major road to the other end of a major road and real traffic does not behave in this way. Additionally, many major roads end in random small towns, so many of the major cities were not connected by direct links. To fix this issue, we require a method for constructing a road network model where edges include all possible reasonable routes a driver could take between nearby major cities, however, constructing such a network from our data is difficult, and I could not figure out how to do it.

1.3 Railway Traffic

To estimate the railway traffic, a similar approach to estimating road traffic was used and results are shown in figure 3a. The data used [3] did not use longitudinal or latitudinal coordinates, and I could not figure out which coordinate system they were using, as such I did not include coastline data for visualizing this dataset. I did try to use other data sets, but they were too large and the code took over an hour to run before I stopped it.

1.4 Combining Transport Networks

Figure 3b shows a combined map of flight and road networks. The railway traffic was not included due to the unknown coordinate system used in the data provided.

Although there are some technicalities to this assignment question which I did not complete, I hope that the extra effort that I put into these visualizations make up for it. The size of the networks are significantly larger than what was asked for, and the use of proper geospatial coordinates make the traffic networks much easier to interpret. I did not provide city names, but the inclusion of coastline data makes up for it. I did not use cytoscape, but the software is not suitable for the visualizations I was going for. I did not provide exact numbers for traffic estimates, but the resulting list would be too large to be included in a report.

2 Minimum Spanning Trees

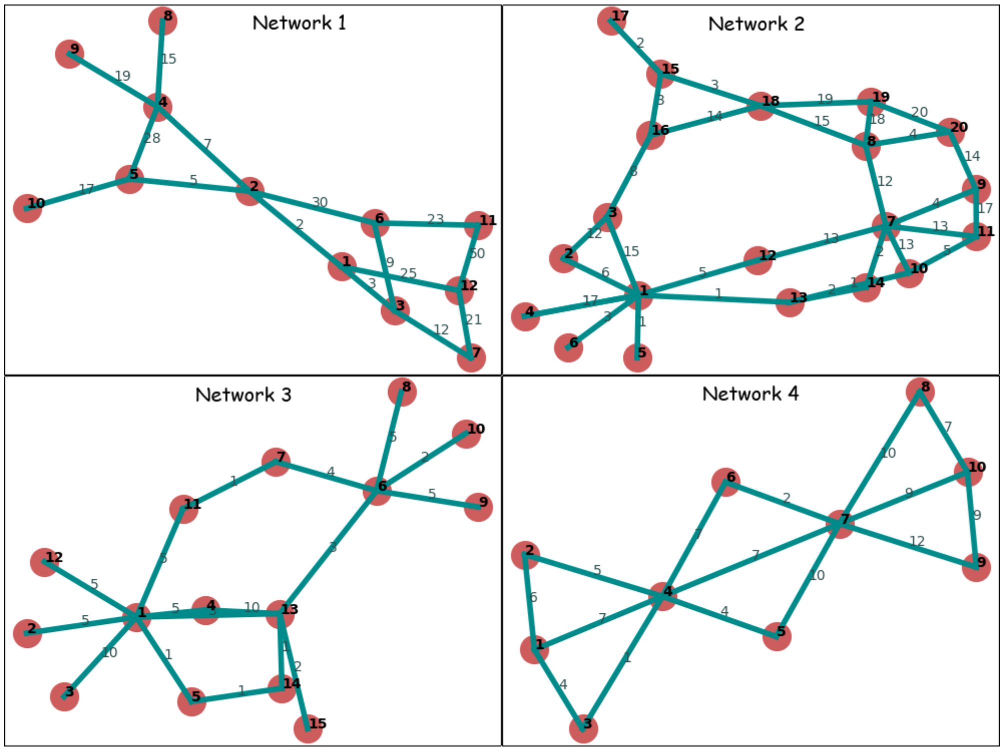
2.1 Visualizing Networks

Results are shown in figure 4a. The coloring scheme was chosen by my girlfriend.

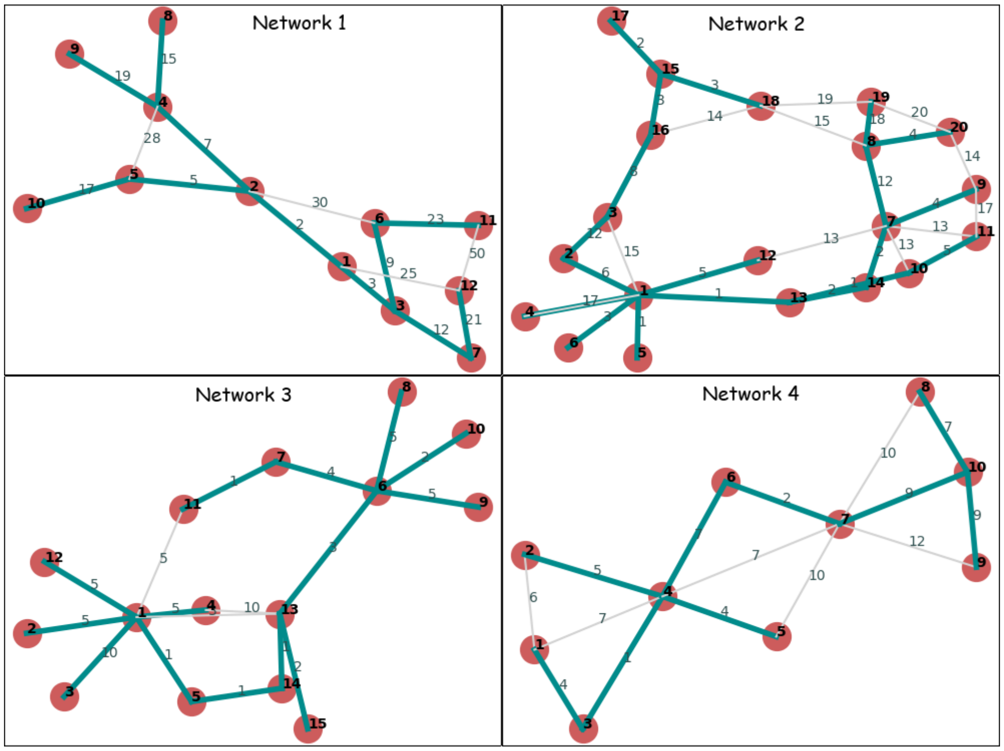
2.2 Computing Minimum Spanning Trees

The following code snipplet is the main algorithm used in identifying which edges of our networks belong to the minimum spanning network. It is based on *Prim's Algorithm* and the full code is provided as a python file in this assignment submission.

```
#List of nodes that are already in the MST
nodesInMST = [0]

#List of ones and zeros of same length and order as list of edges of main network
#A one indicates this edge is part of MST, zero otherwise
isMSTedge = np.zeros(len(edges))
```



(a) **Question 2.1** - Networks visualized using spring force directed layout



(b) **Question 2.3** - Minimum Spanning Trees of our networks

Figure 4

```
#Run loop while we still haven't traversed all nodes
while len(set(nodesInMST)) != numberOfNodes:

    #For each traversed node:
    # - Get weights of all edges that aren't allready part of the MST
    # - Get the node on other side of those edges
    idx = []
    otherNode = []
    weightsToChoseFrom = []
    for node in nodesInMST:
        for i, e in enumerate(edges):

            #Conditions to check for each edge in network
            edgeContainsThisNode = (node in e)
            isNotAlreadyPartOfMST = (isMSTedge[i] == 0)
            atLeastOneNodeOnEdgeIsNotPartOfMST = ((e[0] not in nodesInMST) or (e[1]
            ↪   not in nodesInMST))

            #If this edge satisfies all conditions, consider this edge as a
            ↪   possible MST edge
            if edgeContainsThisNode and isNotAlreadyPartOfMST and
            ↪   atLeastOneNodeOnEdgeIsNotPartOfMST:
                idx.append(i)
                other = e[e!=node]
                otherNode.append(other[0])
                weightsToChoseFrom.append(weights[i])

    #Add the lightest edge to the MST
    minWeightIndex = np.argmin(weightsToChoseFrom)
    isMSTedge[idx[minWeightIndex]] = 1
    nodesInMST.append(otherNode[minWeightIndex])
```

2.3 Visualizing the Minimum Spanning Trees

Results are shown in figure 4b

Question 3

June 8, 2021

1 Question 3.(a):

We calculate $f(x_1) + f(x_2) + f(x_3)$:

[1]:
```python
import numpy as np
x = np.array([1, 2, 7])
fx = np.exp(-x)
ans = np.sum(fx)
print('{:.4f} + {:.4f} + {:.4f} = {:.4f}'.format(fx[0], fx[1], fx[2], ans))
```

```
0.3679 + 0.1353 + 0.0009 = 0.5041
```

We calculate $f(y_1) + f(y_2) + f(y_3)$:

[2]:
```python
y = np.array([4, 3, 3])
fy = np.exp(-y)
ans = np.sum(fy)
print('{:.4f} + {:.4f} + {:.4f} = {:.4f}'.format(fy[0], fy[1], fy[2], ans))
```

```
0.0183 + 0.0498 + 0.0498 = 0.1179
```

We calculate $x_1 + x_2 + x_3$ and $y_1 + y_2 + y_3$:

[3]:
```python
print('{:.1f} + {:.1f} + {:.1f} = {:.1f}'.format(x[0], x[1], x[2], np.sum(x)))
```

```
1.0 + 2.0 + 7.0 = 10.0
```

[4]:
```python
print('{:.1f} + {:.1f} + {:.1f} = {:.1f}'.format(y[0], y[1], y[2], np.sum(y)))
```

```
4.0 + 3.0 + 3.0 = 10.0
```

2 Question 3.(b):

Since $\mathbf{x} = [1, 2, 7]$ is a more heterogeneous distribution compared to $\mathbf{y} = [4, 3, 3]$, and since:

$$f(x_1) + f(x_2) + f(x_3) > f(y_1) + f(y_2) + f(y_3)$$
$$0.5041 > 0.1179$$

we conclude that the negative exponential sum favours a more heterogenous distribution.

3 Question 3.(c, d, and e):

For a given passanger, the probability $P^k$of choosing a mode of transport $k \in [0, 1, 2]$ is calculated by the logit function:

$P^k = \frac{e^{-\beta c^k}}{\sum_{i=0}^{k} e^{-\beta c^i}}$

where c is the costs for a given transport and person type, and β is the sensitivity to cost based on person type. We start by defining the cost matrix and sensitivities:

[5]:
```
costs = np.array([
    [500, 400, 300],  #Working Professional
    [300, 400, 500],  #Student
    [600, 100, 400]]) #Senion Citizen
sensitivity = np.array([10, 5, 2])
```

We normalize our cost matrix along the rows axis, then apply the equation above:

[6]:
```
costNormalized = (costs.T / np.sum(costs, axis=1)).T
betaCost = (costNormalized.T * sensitivity).T
expBetaCost = np.exp(-betaCost)
sumExpBetaCost = np.sum(expBetaCost, axis=1)
probabilities = (expBetaCost.T / sumExpBetaCost).T
```

We print results to screen and round to 2 decimal places:

[7]:
```
personType = ['working professional', 'student', 'senior citizen']
text = 'For a {}, the probability of chosing bus, train or plane are:\n[{:.2f},␣
 ↪{:.2f}, {:.2f}]\n'
for i, p in enumerate(probabilities):
    print(text.format(personType[i], p[0], p[1], p[2]))
```

```
For a working professional, the probability of chosing bus, train or plane are:
[0.12, 0.27, 0.62]

For a student, the probability of chosing bus, train or plane are:
[0.48, 0.31, 0.21]

For a senior citizen, the probability of chosing bus, train or plane are:
[0.20, 0.50, 0.29]
```

4 Question 3.(f):

Assuming that by *generalised costs* we include the sensitivity parameter β, our new probabilities are given by:

$P^k = \frac{1}{\beta c^k}$

and we normalize P^k afterwards.

```
[8]: newProbs = 1 / betaCost
newProbsNormalized = (newProbs.T / np.sum(newProbs, axis=1)).T
for i, p in enumerate(newProbsNormalized):
    print(text.format(personType[i], p[0], p[1], p[2]))
```

```
For a working professional, the probability of chosing bus, train or plane are:
[0.26, 0.32, 0.43]

For a student, the probability of chosing bus, train or plane are:
[0.43, 0.32, 0.26]

For a senior citizen, the probability of chosing bus, train or plane are:
[0.12, 0.71, 0.18]
```

5 Question 3.(g):

We create the tables and display them:

```
[9]: import pandas as pd
personType = ['Working Professional', 'Student', 'Senior Citizen']
columnNames = ['P(Bus)', 'P(Train)', 'P(Plane)']

df = pd.DataFrame(probabilities,
             columns = columnNames,
             index=personType)

df.style.set_caption('Based on Logit Function').set_table_styles([{
    'selector': 'caption',
    'props': [('font-size', '16px')]}])
```

Based on Logit Function

	P(Bus)	P(Train)	P(Plane)
Working Professional	0.116340	0.267696	0.615963
Student	0.477592	0.314848	0.207560
Senior Citizen	0.203227	0.504422	0.292352

```
[10]: df = pd.DataFrame(newProbsNormalized,
             columns = columnNames,
             index=personType)

df.style.set_caption('Based on Linear Relationship With Costs').
 ↪set_table_styles([{
    'selector': 'caption',
    'props': [('font-size', '16px')]}])
```

Based on Linear Relationship With Costs

	P(Bus)	P(Train)	P(Plane)
Working Professional	0.255319	0.319149	0.425532
Student	0.425532	0.319149	0.255319
Senior Citizen	0.117647	0.705882	0.176471

[11]:
```
print(costs)
```

```
[[500 400 300]
 [300 400 500]
 [600 100 400]]
```

[12]:
```
print(sensitivity)
```

```
[10  5  2]
```

From the above observations, we can see that the logit function returns probabilities that better incorporates the sensitivity β values defined for each person type. This can be seen clearly when we compare the probabilities of *Working Professional* and *Senior Citizen*, who have a high (10) and low (2) sensitivity parameters. The probalities of chosing the prefered travel method (Plane) for the highly sensitive *Working Professional* is significantly higher when using logit compared to inverse. Meanwhile the probability of chosing the prefered travel method (Train) for the low sensitive *Senior Citizen* is lower using logit compared to inverse.

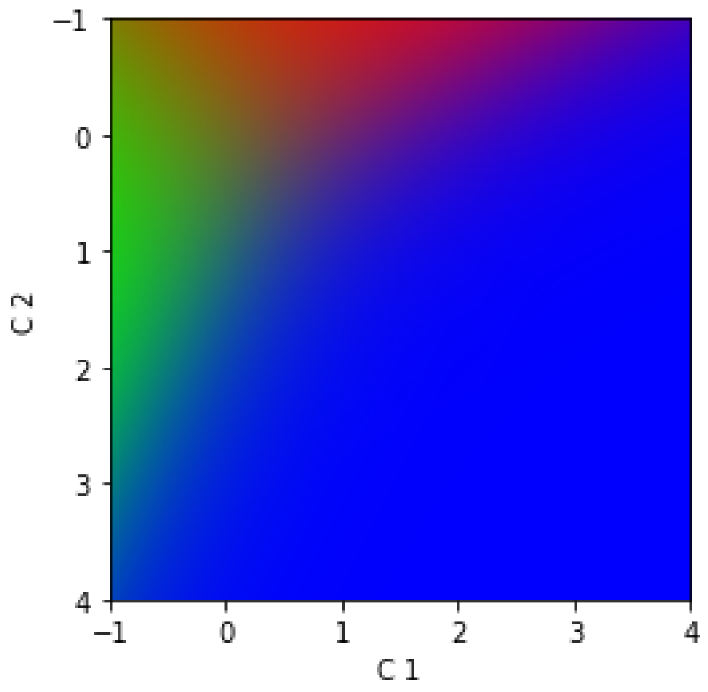
Additionally, as $\beta c \rightarrow 0$ we have $P = \frac{1}{\beta c} \rightarrow \infty$, so probabilities obtained by the inverse are extremely sensitive to low values of βc and can not handle zero values. Probabilities obtained by the inverse also had to be normalized after passing values through it.

Meanwhile the logit function is well behaved regardless of the input costs. This is demonstrated in the code bellow where the costs are c_1 & $c_2 \in [-1, 4]$, and $c_3 = 1 - c_1 - c_2$. A plot is provided where the x and y axii represent c_1 and c_2, and pixels represent probabilities mapped to colors by $[P(Bus), P(Train), P(Plane)] = [Red, Green, Blue]$. We can see that regardless of the input costs, the logit function is smooth and the resulting probabilities always add to 1.

[13]:
```
import matplotlib.pyplot as plt

#Generate arary of costs
betaC = np.array([[i, j, (1-i-j)] for i in np.arange(-1, 4, 0.01) for j in np.
 ↪arange(-1, 4, 0.01)])

#Do calculations
expBetaCost = np.exp(-betaC)
sumExpBetaCost = np.sum(expBetaCost, axis=1)
probabilities = (expBetaCost.T / sumExpBetaCost).T
```

```python
#Show that resulting probabilities always add to 1
sumProbs = np.sum(probabilities, axis=1)
print('Minimum Sum of Probabilities: {}'.format(min(sumProbs)))
print('Maximum Sum of Probabilities: {}'.format(max(sumProbs)))

#Show the image
imageData = probabilities.reshape(500, 500, 3)
plt.imshow(imageData, extent=[-1, 4, 4, -1])
plt.xlabel('C 1')
plt.ylabel('C 2')
plt.show()
```

```
Minimum Sum of Probabilities: 0.9999999999999997
Maximum Sum of Probabilities: 1.0000000000000002
```



References

[1] Datahub *Country Polygons as GeoJSON* https://datahub.io/core/geo-countries Visited on - 03/06/2021

[2] U.S. Census Bureau, *TIGER/Line Shapefile, 2016, nation, U.S., Primary Roads National Shapefile*, https://catalog.data.gov/dataset/tiger-line-shapefile-2016-nation-u-s-primary-roads-national-shapefile Visited on - 03/06/2021

[3] U.S. Geological Survey, *North America Railroads*, https://www.sciencebase.gov/catalog/item/4fb55b74e4b04cb937751de9, Visited on - 06/06/2021

[4] Datahub, *Airport Codes* https://datahub.io/core/airport-codes Visited on - 03/06/2021

[5] University of Sydney - COMP5048 (2018 Semester 2) - Assignment 2, https://github.com/QuangTrungNguyen/COMP5048-visualisations/tree/master/Assignment%202/2.%20Correlation%20Plot Visited on - 03/06/2021