IEOR 174 Project Report

Natalie Andersson, Lucie Chen, Varsha Nekkanti

Prof. Zeyu Zheng

Industrial Engineering and Operations Research

University of California, Berkeley

December 17th, 2021

Contents

1 Overview

1.1 Problem Statement

The overarching goal of this project was to determine the optimal oversell percentage for airline tickets to maximize revenue. Within the airline industry, overselling of airline tickets is a very common practice given that on average, between 5%-15% of people do not end up showing up to their flights despite purchasing a ticket [6]. Airlines are able to oversell tickets because they don't need to assign a customer to a seat when the ticket is first sold. Moreover, since there exists a percentage of customers who do not show up to their flights, airlines leverage overselling as a way to ensure they avoid losing revenue from operating a flight with empty seats. Thus, our project explores this optimization problem further to determine the optimal oversell percentage for domestic Southwest Airlines flights departing from Oakland airport (OAK) in a given travel season (see appendix). We chose to focus on these flights because Southwest's domestic flights only include one class – economy, and tend to have a similar demographic of travelers.

Our objective incorporates the fact that we also want to minimize the likelihood of bumping passengers as this would lead to not only a loss of revenue through compensation packages, but also a loss of future revenue due to a decrease in brand perception.

1.2 Motivation

Overall, our motivation for conducting this simulation is to determine the ways in which airlines can minimize losses in revenues, while balancing factors like varying customer demand and seasonality. This problem is particularly relevant in an industry like the airline industry where airlines often operate with tight margins, and overselling will remain widespread as long as it remains legal. Overbooking and flight bumping are especially relevant now, as data from the U.S. Department of Transportation indicates that airlines bumped 13,104 paying passengers in the first six months of 2019. Southwest Airlines bumped 2,525 of these passengers for various 'operational challenges' [9].

1.3 High Level Simulation Approach

To model the dynamics of this system, we simulated bookings and the number of people who actually showed up to board the flight using historical distributions. With this, we determined the overbooked/bumped passengers and the expected revenue. Given that there exists two types of bumping: voluntary and involuntary,

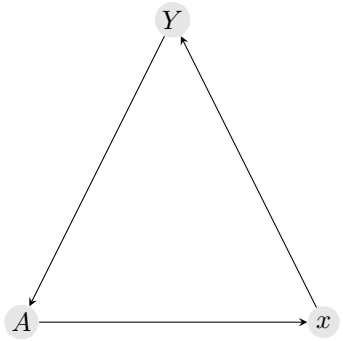
we also simulated the number of bumped people in each of these categories. Voluntary bumping refers to passengers that willingly give their seats in exchange for a seat on a later flight and a voucher. Involuntary bumping refers to passengers who are denied boarding due to a full flight. Involuntarily bumped customers receive a cash compensation package and a seat on a later flight. Involuntarily bumping typically results in a bigger decrease in brand perception than voluntary bumping, and certain cases of involuntary bumping have been featured in major news stories. In order to account for future loss of revenue to previous overselling actions, we incorporated a choice model. This will be discussed further in the methods section of the report.

2 Methods

2.1 Data

For our project, we utilized data from the U.S. Department of Transportation, the Bureau of Transportation Statistics, and U.S. Air CarrierTraffic Statistics on load factor, average fare price, and total number of operated flights [1] [2]. We specifically focused on the 2009 - 2019 timeframe, since we didn't want to include data collected during COVID-19 because it is an outlier and skews the distribution. Load factor is defined to be the "measure of the use of aircraft capacity that compares Revenue Passenger-Miles as a proportion of Available Seat-Miles" [3]. In other words, load factor represents the percentage of airplane seats that are actually occupied. We also found data on Southwest's airplane fleet to approximate the average number of physical seats available on a flight [4].

2.2 Model Setup



We modeled this system with 3 nodes: Y denotes the demand for Southwest airlines flights, A denotes Southwest's action, and x denotes Southwest customers' satisfaction with respect to bumping. The key concept in this model is that the action in the previous time period influences the demand for the current time period through customer satisfaction. In other words, this system acts like a Markov chain where the current state only depends on the previous state. Mathematically, this means that Y_{t+1} depends only on x_t and x_t depends only on A_t. By modeling revenue with this approach, we can see how changes in overselling each season will change revenue year over year (overtime).

2.2.1 Determining Y_t

We only have reliable data on the average load of Oakland Airport, so we used this and simulation techniques to find demand. We defined load as constrained demand. In other words, we assumed that given that load can never exceed 100%, demand will be greater than load (that's why the notion of overselling exists). We approximated demand as a normal distribution with mean $\mu >$ average load. To determine the second parameter for our normal distribution, σ^2, we simulated constrained demand as min(100, demand). We tried various different values for, μ, σ^2 where the expected value of these simulated random variables was equal to the average load factor 0.80 (for summer season) and 0.73 (for winter season). To learn our parameters, we used a very large size $n = 300000$. The final μ parameter (2,000,000 for summer season, 1,500,000 for winter season) is what we define as total demand, D, for Oakland Airport in our model. *See appendix for code.*

To model Y_t (demand for Southwest flights), we utilized D, calculated above, and a multinomial logit choice model (MNL). Namely:

$$Y_t = D * \phi(SW, OAK)$$

Using an MNL model, we aimed to represent the probability that a customer chooses to fly Southwest, $\phi(SW, OAK)$, when given an assortment of all other airlines flying domestic flights out of Oakland. The other airlines/competitors considered were: Spirit Airlines, Alaska Airlines, Allegiant Airlines, Hawaiian Airlines, Delta Airlines, and Frontier Airlines. This probability is defined as:

$$\phi(SW, OAK) = \frac{v_{SW}}{v_0 + \sum_{j \in OAK} v_j} = \frac{e^{\beta^T x_{SW}}}{1 + \sum_{j \in OAK} e^{\beta^T x_j}}$$

In this equation, the parameter v_{SW} represents propensity, a customer's overall motivation and inclination to pick Southwest airlines out of all the possible competitors. The customer can also choose to not fly at all, which we categorized as the outside option, v_0. To make the equation cleaner, v_0 has been normalized to 1. This parameter v_j depends on a variety of features, β, and feature weights, x_j, one of which depends on the number of bumped customers in the previous time period.

To further capture the dynamics of passengers, for the winter season, we also implemented an mixed multinomial logit (MMNL) choice model to see if that would change our findings. An MMNL choice model subsets the population of passengers into k types. For the purpose of this project, we set $k = 2$, for two categories of travelers: leisure (l) and business (b) travelers. For an MMNL model, $\phi(SW, OAK)$ is defined as:

$$\phi(SW, OAK) = \gamma_b \phi_b(SW, OAK) + \gamma_l \phi_l(SW, OAK) = \gamma_b \frac{v_{b,SW}}{v_{b,0} + \sum_{j \in OAK} v_{b,j}} + \gamma_l \frac{v_{l,SW}}{v_{l,0} + \sum_{j \in OAK} v_{l,j}}$$

With the above equations, similar to the scenario for an MNL choice model, one can incorporate features and feature weights for each category of traveler for the respective airline.

2.2.2 Defining an Action A_{t+1}

We define the no-show probability as α, which is exponentially distributed with mean .07 given [6]. We model α as an exponentially distributed random variable since the authors of [6] assert that $\alpha \in (0.05, 0.15)$. Since most passengers arrive for the flight they booked, there will be very little density as the value of α increases. The PDF of an exponential distribution captures this dynamic quite well as shown in the plot of the PDF below:

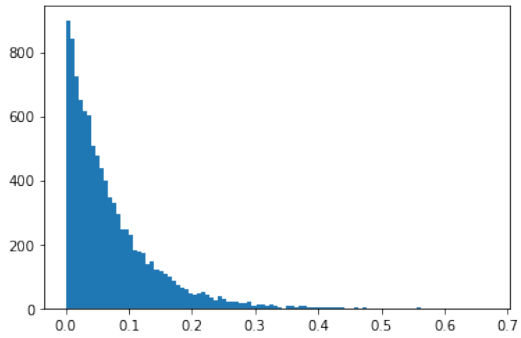


Figure 1: PDF of Exponential Random Variable with Expectation = 0.07

Since customers either show up or they don't, we model the number of customers that arrive for boarding as a binomial random variable with probability $p = 1-\alpha$ of showing up and $n = \max\{Y_t, \text{capacity} * (1+\theta)\}$ (for definition of θ see end of 2.2.2). Note that we simulate the boardings for an entire travel season as opposed to a single flight. When we implemented an MMNL choice model as opposed to an MNL choice model for the winter travel season, we simulated overall arrivals (not separated by category of passenger). Using the number of total arrivals, we used a binomial distribution to simulate the count of business versus leisure travelers. We assumed that there would be $p = 0.6$ probability of a passenger being leisure and $p = 0.4$ probability of a passenger being business.

From data on Southwest's (and it's competitors) airplane fleets, we determined that the number of seats available during the summer season is 1,536,878 and during the winter season it is 1,427,403 [4][7]. With this, and the number of simulated boardings, we can determine the total number of bumped customers. As shared earlier, we must consider two types of bumping: voluntary, which we identified to be 90 percent of cases, and involuntary, which we identified to be 10 percent of cases [10]. We simulated the probability that passengers would be involuntarily vs. voluntarily bumped using a binomial distribution. We needed to distinguish between involuntary and voluntary bumping because this affects the financial loss per customer and the expected revenue function. The financial losses are greater for passengers that are involuntarily bumped in

comparison to passengers that are voluntarily bumped off the plane. To understand compensation packages for these two groups, we investigated the federal regulations on the minimum mandatory compensation for involuntary bumping [8], and online information on Southwest's typical vouchers for voluntary bumping. For voluntary bumping, we modeled airlines' loss of revenue as $\min\{100 + f, 775\}$, and for involuntary bumping $\min\{2 * f, 775\}$, where f denotes the fare. The left side of the min function for voluntary bumping represents the \$100 voucher they must provide as well as f, the opportunity cost for putting the passenger on a new flight. The left side of the min function for involuntary bumping represents the mandatory cash compensation of the full ticket value, as well as the opportunity cost for putting the passenger on the next flight. The maximum mandatory compensation airlines must provide due to federal regulations is \$775.

Finally, with all of this information, we used a Greedy approach to determine expected revenue, and subsequently, determine the best action. Expected revenue can be broken down into two components: positive revenue from ticket sales, and losses due bumping. From here, we came up with our expected revenue function:

$$E[R] = \text{ticket-sales} - (\text{count-voluntary} * \min\{100 + f, 775\}) - (\text{count-involuntary} * \min\{2 * f, 775\})$$

This objective function is not closed form because the number of customers that are voluntarily and involuntarily bumped are random variables. We denoted θ as the oversell percentage and used simulation to find the E[R] for a variety of different θ values. We then greedily chose θ^*, which was the optimal oversell percentage value that maximized the expected revenue as the action for the next time period.

2.2.3 Updating Feature Parameter x_{t+1}

As shared previously, in the vector x, there exists elements for the following features: bumped customers, flight crew, in-flight amenities, price, and delays. For the purpose of this project we want to focus only on how overselling affects the demand for Southwest. Due this we kept the x_i value for all features other than the one dependent on the number of bumped customers constant. $x_i[1]$, the first element in the x vector, is what we defined as the "bumped feature", which is proportional to the number of bumped customers. The feature weight β corresponding to bumped feature is negative, so a higher value for bumped feature will result in an overall lower inclination to fly Southwest. We calculated the bumped feature as the following:

$$\text{bumped-feature} = \min\{(2 * \text{count-voluntary} + 20 * \text{count-forced})/\text{capacity}, 1\}$$

In the function above, we weight the impact of forced bumping as 10 times the weight for voluntary bumping due to the fact that involuntary bumping results in a much higher decrease in brand image. For normalization purposes, this is then divided by total capacity. We included the min function to ensure that there is an upper bound on how much bumping can affect overall propensity.

In this part, we use the oversell percentage calculated in the "determining action" step to simulate demand for a single season, $t+1$. From this, we get the number of bumped customers, and update the bumped-feature value and feature vector accordingly. This new feature vector is the input for propensity v_i used to calculate the demand Y_{t+1}.

In the case of an MMNL choice model, the bumped-feature has to be updated separately for business and leisure travelers. Likewise, the impact on brand perception when passengers are bumped, varies based on the type of passenger. We assumed that business travelers who are bumped would be much more unhappy than leisure travelers and thus their demand for traveling with Southwest would decrease more (than leisure travelers) in travel season $t+1$. Accordingly, the calculation for bumped-feature was the following:

$$\text{bumped-feature-leisure} = \min\{(2 * \text{count-voluntary-leisure} + 20 * \text{count-forced-leisure})/\text{capacity}, 1\}$$

$$\text{bumped-feature-business} = \min\{(10 * \text{count-voluntary-business} + 500 * \text{count-forced-business})/\text{capacity}, 1\}$$

To capture the fact that business travelers will be much more unhappy when bumped, particularly in the case in involuntary bumping as they often have a fixed schedule with important meetings/events, we heavily penalized the bumped feature for this instance by 500 (as opposed to just 20 for leisure travelers). In the case of voluntary bumping, we also penalized the bumped feature for business travelers more highly than leisure travelers, but not as extremely. Using a similar method above, we input the new feature vectors find demand for the next travel season, Y_{t+1}.

3 Findings and Implications

3.1 Summer Travel Season, MNL

After running our code for 20 summer travel seasons we observed that the average optimal oversell percentage, θ, is 0.1345. We see that there is a good amount of variance in θ^* from year to year however, revenue mostly seems to be very consistent. Likewise, we noticed that when revenue drops significantly this occurred either when the oversell percentage was very low (below 0.05) or very high (above 0.2) and our algorithm corrected for that. See below for figures that show the optimal theta, expected revenue, and demand for Southwest flights:

We note in the above figures that overselling does significantly impact revenue (improves revenue) and airlines should oversell. This is not surprising given that overselling is common practice in the airline industry, it does validate our findings. Similarly, we observe that expected revenue remains pretty consistent across the 20 years which is good for airlines because it means they can pretty easily predict revenue. This observation indicates that our model and algorithms can adjust overselling to compensate for changes in demand to

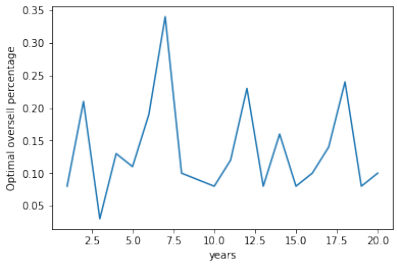


Figure 2: Optimal oversell percentage θ^* for each year

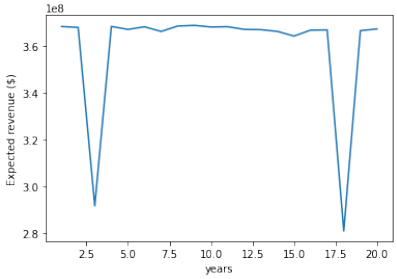


Figure 3: Expected revenue for each year

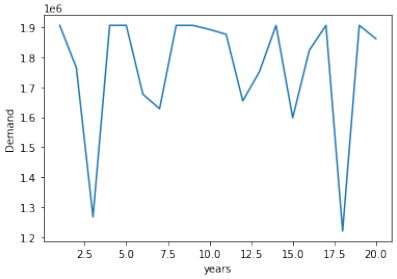


Figure 4: Demand for Southwest flights for each year

ensure revenue remains consistent.

3.2 Winter Travel Season, MNL

After running our code for 20 winter travel seasons we observed that the average optimal oversell percentage, θ, is 0.1050. We note that this is somewhat lower than during the summer travel season where optimal $theta = 0.1345$ which makes sense because demand is lower during the winter season. In fact, during some of the years, our model suggests the oversell percentage should be 0 because demand is below capacity and therefore there is no need to oversell. See below for figures that show the optimal theta, expected revenue, and demand for Southwest flights:

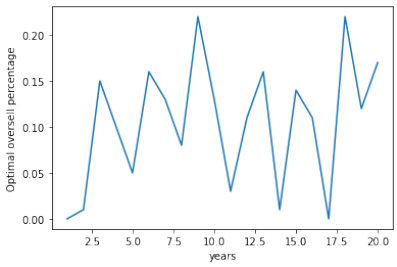


Figure 5: Optimal oversell percentage θ^* for each year

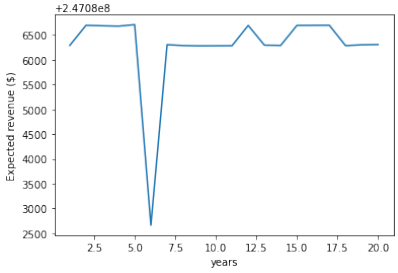


Figure 6: Expected revenue for each year

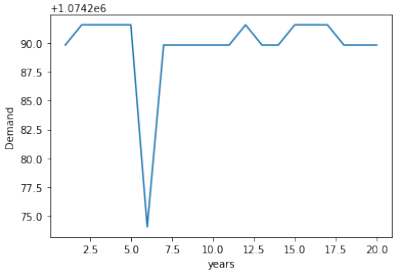


Figure 7: Demand for Southwest flights for each year

We note in the above figures, similar to the summer travel season, expected revenue is pretty consistent across the 20 years with the exception of one drop (year 6). Unlike the summer travel season, this drop cannot be connected to extreme oversell percentages in either direction. Thus, this is likely due to the innate randomness of α, the no-show percentage. While the θ value for year 6 is above average, it's by no means the highest oversell within our simulation. In year 6, the number of no show customers was lower then expected leading to more bumped customers and therefore a drop in revenue.

3.3 Winter Travel Season, MMNL

After running our code for 20 winter travel seasons where we subsetted the population of travelers into business and leisure passengers, we observed that the optimal oversell percentage θ is 0.0935. We note that this oversell percentage is lower than the winter MNL θ^* likely because the impact of bumping a business traveler is much higher with the MMNL model and therefore impacts demand more. We aren't sure if the features and values we chose for bumped-feature-business are realistic and likely require more tuning to make this choice model more accurate. See below for figures that show the optimal theta, expected revenue, and demand for Southwest flights:

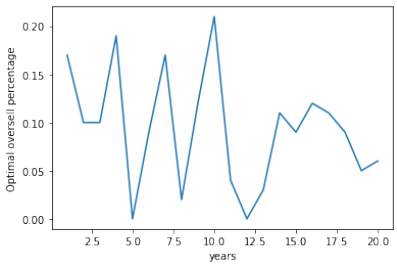


Figure 8: Optimal oversell percentage θ^* for each year

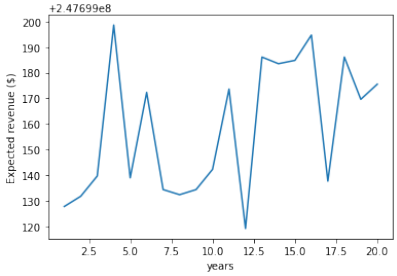


Figure 9: Expected revenue for each year

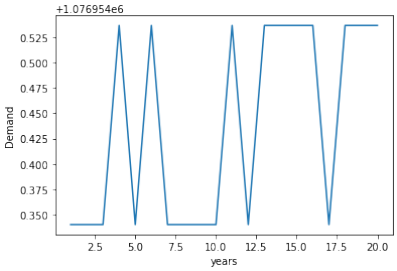


Figure 10: Demand for Southwest flights for each year

We note in the above figures, the expected revenue is much lower than in the corresponding MNL simulation. This is likely because of parameter calibration issues as well as the high impact of bumping business travelers. It's important to note that while the graphs for demand and expected revenue visually seem very inconsistent, when looking at the actual values and the y-axis, it's clear that revenue remains relatively consistent which is a good sign for our algorithm. We are the least confident about these graphs and results because the difference between the expected revenue from the MNL simulation and the MMNL simulation is so high. This could be do to the randomly generated features being quite different from the MNL context or the changes we made to β. We decreased β for the bumping feature by 1 for business travelers for MMNL choice model to further account for the fact that business travelers dislike being bumped. This change, along with the change in the bumped-feature-business calculation may have over corrected for the impact of bumping business passengers.

4 Further Discussion

This project addresses a classical problem within the operations research field. The airline industry has long been at the forefront for researching and developing models to maximize expected revenue while ensuring high brand loyalty among passengers. By using simulation to capture the actions of random travelers, one can optimize the overselling strategy. The factors considered in this report are likely also considered by operations research analysts for major airlines. In reality, airlines collects a myriad of data points for each

traveler allowing them to used more advanced modeling techniques that consider no-show probability for each individual traveler using Bayesian inference.

Given the limitations and assumptions of our project, our results seem reasonable. Overselling and the no-show probabilities simulated generally matches our intuition but we aren't sure how realistic our demand and revenue figures are. These areas were where we faced the greatest challenges gathering data because airlines treat them as proprietary information. Most of the data we collected relied on public government institutions such as the Bureau of Transportation Statistics and the U.S. Department of Transportation. While we believe the values provided from these sources are accurate, using averages based on these data points may be skewed since they don't eliminate outliers. Specifically, the Bureau of Transportation Statistics mentions that they include in their data flights that carry fewer than 60 passengers and cargo flights which could definitely be considered an outliers. Southwest specifically states in their corporate fact sheet that they carried over 158 million pounds of cargo in 2019, indicating that cargo flights may skew certain distributions to the right [11].

Overall, we aren't sure how well the physics of our model fits the actual system dynamics of airline booking. Customers often choose tickets based off of price, flights offered, and timing of the flight and don't pay close attention to the airline. The airline industry is an oligopoly due to high overheads and startup costs which influences the assortment of choices a customer has when booking a ticket. Often larger airlines fly more flights to more places and Southwest's high market share at OAK may be due to the fact that they are the only option for certain customers. This means that overselling may not have as large of an impact as our model depicts it to have.

Going forward, the next steps for this project would be to primarily better calibrate the β values and the randomly generated features x. To do so, more data and investigation is required. This could be done by contacting airline companies or surveying Southwest passengers. However, in reality, getting access to proprietary data from airline companies would be difficult and conducting a survey may be time consuming and contain a lot of bias, as it is based on customer perceptions rather than past numerical values.

5 Thank Yous

We would like to thank Professor Zeyu Zheng for advising us on this project and providing us with expertise on simulation optimization and choosing distributions to model random processes. We couldn't have gone this far without him. Thank you to our GSI Yiduo Huang for suggesting we incorporate a time dimension to our model and consider the impact overbooking has on demand in future time periods. Lastly, thank you to Professor Rajan Udwani for introducing us to the topic of overselling within the airline industry as well

as teaching us about choice models and their relevance.

6 Video

Video link. Please note we meant to say at 6:34 that γ_l should be 80% or 0.8.

7 References

[1] R. Gan, N. Gans and G. Tsoukalas, "Overbooking with Endogenous Demand," The Wharton School Research Paper, 2019, pp. 13-14, doi: 10.2139/ssrn.3321409.

[2] Department of Transportation Office of the Assistant Secretary for Aviation and International Affairs, "Consumer Airfare Report: Table 1A - all U.S. Airport Pair Markets: Department of Transportation - Data Portal," data.transportation.gov, 12-Oct-2021. [Online]. Available: https://data.transportation.gov/Aviation/Consumer-Airfare-Report-Table-1a-All-U-S-Airport-P/tfrh-tu9e. [Accessed: 26-Oct-2021].

[3] "Load factor for U.S. air carrier domestic and international, scheduled passenger flights," FRED, 20-Oct-2021. [Online]. Available: https://fred.stlouisfed.org/series/LOADFACTOR0. [Accessed: 26-Oct-2021].

[4] Southwest Airlines Fleet Details and History. [Online]. Available: https://www.planespotters.net/airline/Southwest-Airlines. [Accessed: 26-Oct-2021].

[5] R. Gan, N. Gans and G. Tsoukalas, "Overbooking with Endogenous Demand," The Wharton School Research Paper, 2019, pp. 13-14, doi: 10.2139/ssrn.3321409.

[6] H. J. Kamps, "Why do airlines overbook their flights?," TechCrunch, 11-Apr-2017. [Online]. Available: https://techcrunch.com/2017/04/11/overbooking/. [Accessed: 26-Oct-2021].

[7] "Data elements - transtats.bts.gov." [Online]. Available: https://www.transtats.bts.gov/DataElements.aspx?Data=3. [Accessed: 26-Oct-2021].

[8] "Bumping Oversales," U.S. Department of Transportation. [Online]. Available: https://www.transportation.gov/individua consumer-protection/bumping-oversales. [Accessed: 26-Oct-2021].

[9] J. J. Chris Chmura, "Bumping is back: Federal Data Show more airline passengers denied boarding," NBC Bay Area, 20-Nov-2019. [Online]. Available: https://www.nbcbayarea.com/news/local/bumping-is-

back-federal-data-show-more-airline-passengers-denied-boarding/2151851/. [Accessed: 26-Oct-2021].

[10] Kelligrant, "Here's what you're entitled to if you're involuntarily bumped from a Flight," CNBC, 12-Apr-2017. [Online]. Available: https://www.cnbc.com/2017/04/12/heres-what-youre-entitled-to-if-youre-involuntarily-bumped-from-a-flight.html. [Accessed: 26-Oct-2021].

[11] "Southwest Corporate Fact sheet," Southwest Airlines Newsroom. [Online]. Available: https://www.swamedia.com/pages fact-sheet. [Accessed: 26-Oct-2021].

8 Appendix

8.1 Defining Travel Season

We defined the summer travel season as June, July, and August (and the winter season as November, December, January). Below is a graph from the U.S. Bureau of Transportation Statistics on load factor for each month from 2009 - 2019. The graph shows that there is a clear difference in load factor depending on which travel season one considers. It's also evident that the summer is the peak travel season (indicated by a higher load) and therefore has the greatest amount of overbooking. For that reason we decided to begin our analysis using summer data.

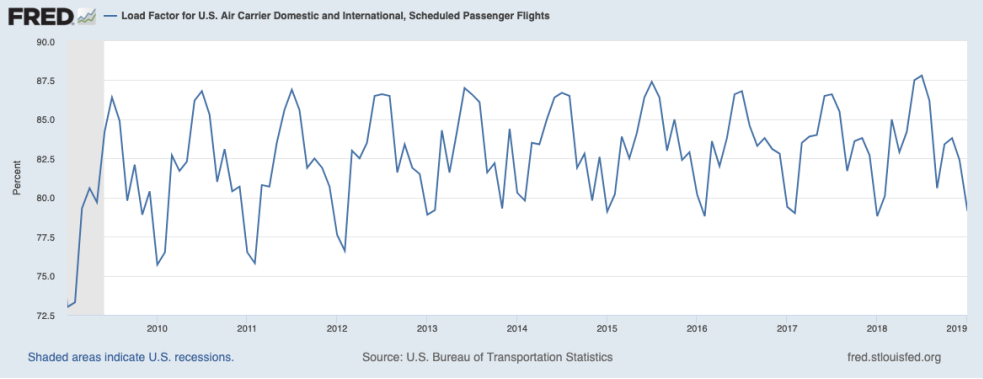


Figure 11: U.S. Domestic Load Factor, 2009 - 2019 [3]

8.2 Code

demand_with_system_dynamics

November 24, 2021

[95]:
```
clear all
```

[96]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import pickle
import pdb
```

Read in the number of passengers traveled from OAK airport on domestic flights (all airlines) during the summer months in 2009. We are assuming that the number of passengers traveling out of OAK makes up the demand for the types of flights that our airline of interest, Southwest, flies. This demand will be split among Southwest and it's competitors based on a variety of factors, to be analyzed later on.

[104]:
```
passengers_all_years_all = pd.read_csv('10_year_all_airline_demand.csv')
```

[107]:
```
passengers_2009_all = pd.read_csv('passengers_oak.csv') #starting year = 2009
passengers_2009_all.iloc[0]['DOMESTIC']
```

[107]: 1246775.0

Read in the number of passengers traveled from OAK on Southwest flights during the summer months in 2009

[101]:
```
demand_2009_sw = pd.read_csv('passengers_oak_southwest.csv')
initial_demand_sw = int(demand_2009_sw['DOMESTIC'][0])
initial_demand_sw
```

[101]: 918242

[109]:
```
sw_market_share = initial_demand_sw / passengers_2009_all.iloc[0]['DOMESTIC']␣
↪#initial_demand_all
sw_market_share
```

[109]: 0.7364937538850234

Read in the number of flights Southwest operated departing from OAK during summer months in 2009

[346]:
```
flights_2009_sw = pd.read_csv('flights_operated_summer_2009_oak.csv')
flights_operated_sw = int(flights_2009_sw['Flights Operated Southwest Summer␣
↪2009 '][0])
flights_operated_sw
```

[346]:
```
10178
```

Read in the number of flights operated by all airlines departing from OAK during summer months in 2009

[285]:
```
flights_2009_all = pd.read_csv('all_airlines_flights_operated.csv')
flights_operated_all = int(flights_2009_all['Flights Operated All Airlines'])
flights_operated_all
```

[285]:
```
13313
```

[286]:
```
#num_competitors = 6
#features = []
#for i in np.arange(num_competitors + 1):
    #rands = np.random.uniform(0, 1, size = 5)
    #features.append(rands.tolist())
```

[287]:
```
with open ('outfile', 'rb') as feature_list:
    features = pickle.load(feature_list)
```

[288]:
```
beta_southwest = [-5, 1, 1.5, 10, -3]
beta_alaska = [-2, 2, 3, 7, -1]
beta_allegiant = [-5, 0.75, 1, 9, -2]
beta_delta = [-2, 4, 3, 6, -1]
beta_frontier = [-3, 1.5, 1, 10, -2]
beta_hawaian = [-1.5, 4, 3, 2, -2]
beta_spirit = [-7, -1, 0.5, 20, -2]
feature_weights = [beta_southwest, beta_alaska, beta_allegiant, beta_delta,␣
↪beta_frontier, beta_hawaian, beta_spirit]
```

[289]:
```
params = []
for i in np.arange(len(features)):
    param_vi = np.exp(np.dot(features[i], feature_weights[i])) #for each airline
    params.append(param_vi)
phi_sw = params[0] / (1 + sum(params))
```

[290]:
```
phi_sw
```

[290]:
```
0.7946890061491149
```

Read in the average load factor for OAK aiport during summer months

[291]:
```
avg_oak_load_df = pd.read_csv('avg_load_oak.csv')
avg_oak_load = avg_oak_load_df['Average Load OAK']
avg_oak_load = avg_oak_load[0] / 100
avg_oak_load
```

[291]:
```
0.7979333332999999
```

Seat capacity for AirbusA320 and Boeing 737 on average is 151

[292]:
```
seat_capacity = 151 #(https://www.planespotters.net/airline/Southwest-Airlines)
```

We now want to figure out the distribution for demand given the mean load we found above. We consider that our load cannot exceed 100% or (150 passengers per plane), due to the space constraint and therefore when demand is greater than 100% we force it to be 100 to model the load accurately.

[337]:
```
demand_mu = 2000000
demand_var = 1000000
max_passengers = flights_operated_all * seat_capacity
target_passengers = flights_operated_all * seat_capacity * avg_oak_load
simulated_demands = np.random.normal(demand_mu, demand_var, size = 100000)
for i in np.arange(len(simulated_demands)):
    if simulated_demands[i] > max_passengers:
        simulated_demands[i] = max_passengers
    else:
        continue
constrained_avg_demand = np.mean(simulated_demands)
initial_demand_all = demand_mu
print('When we constrain demand we get: ', constrained_avg_demand)
print('The "target" demand we want to match from the load above is: ',␣
 ↪target_passengers)
print("Therefore, we will go with an initial demand for all airlines equal to:␣
 ↪", demand_mu)
```

```
When we constrain demand we get:  1606004.5074840933
The "target" demand we want to match from the load above is:  1604055.8563996577
Therefore, we will go with an initial demand for all airlines equal to:  2000000
```

Demand Function

[338]:
```
def simDemand(total_demand, airline_features, feature_weights):
    # beta = sw_weights
    # x = sw_features
    params = []
    for i in np.arange(len(airline_features)):
        param_vi = np.exp(np.dot(airline_features[i], feature_weights[i])) #for␣
 ↪each airline
```

```
        params.append(param_vi)
    phi_sw = params[0] / (1 + sum(params))
    demand = total_demand * phi_sw
    return demand, phi_sw
```

Bumped Function (action / strategy)

[366]:
```
def simBumped(demand, theta, expectation, capacity):
    '''This function simulates the proportion oversold and therefore bumped␣
↪given demand, oversell percentage, capacity, and expected no-show␣
↪probability'''
    #alpha = np.random.exponential(expectation)
    alpha = np.random.exponential(expectation) #no show probability

    sold_tickets = min(capacity * (1 + theta), demand)
    arrivals = np.random.binomial(sold_tickets , 1 - alpha)
    #arrivals = int((1 - alpha) * (min(capacity * (1 + theta), demand)))
    bumped = max(arrivals - capacity, 0)
    if bumped == 0:
        bumped = 100
    return bumped
```

(Bumped) Satisfaction Level

[367]:
```
def simSatisfaction(number_bumped, capacity):
    if number_bumped == 0:
        bumped_feature = 0
    else:
        count_voluntary = np.random.binomial(number_bumped, 0.9) # 0.9 --> make␣
↪sure to state reasoning for this
        count_forced = number_bumped - count_voluntary
        bumped_feature_int = (2 * count_voluntary) + (20 * count_forced)
        bumped_feature = min(bumped_feature_int / capacity, 1) #WRITE ABOUT THIS
    return bumped_feature
```

Optimizing θ

[368]:
```
def argMaxTheta(demand, v_sw, expectation, capacity, iters, average_fare,␣
↪thetas):
    revenues = []
    for i in np.arange(len(thetas)):
        avg_r = findRevenue(demand, v_sw, thetas[i], expectation, capacity,␣
↪iters, average_fare)
        revenues.append(avg_r)
    highest_rev = max(revenues)
    opt_index = revenues.index(highest_rev)
    opt_theta = thetas[opt_index]
```

```
    return revenues, opt_theta, highest_rev
```

[369]:
```
def findRevenue(demand, v_sw, theta, expectation, capacity, iters,␣
↪average_fare):
    revs = []
    for i in np.arange(iters):
        bumped = simBumped(demand, theta, expectation, capacity)
        sold = (min(capacity * (1 + theta), demand))
        count_voluntary = np.random.binomial(bumped, 0.9) # 0.9 --> make sure␣
↪to state reasoning for this
        count_forced = bumped - count_voluntary
        ticket_sales = sold * average_fare
        loss_function = average_fare + (count_voluntary * (min(100 +␣
↪average_fare,775))) + (count_forced*(min(2 * average_fare,775)))
        revenue = ticket_sales - loss_function
        revs.append(revenue)
    average_rev = np.mean(revs)
    return average_rev
```

[370]:
```
def adjustFeatures(old_features, bumped_feature):
    sw_feature = old_features[0]
    sw_feature[0] = bumped_feature
    new_features = old_features
    return new_features
```

[371]:
```
def timeVarying(initial_demand_all, features, feature_weights, capacity,␣
↪expectation):
    thetas = np.arange(0, 0.35, 0.01)
    iters = 100
    average_fare = 230
    sw_d = []
    revs = []
    opt_thetas = []
    for i in np.arange(20):
        sw_demand, v_sw = simDemand(initial_demand_all, features,␣
↪feature_weights) #find SW demand
        r, theta, highest_rev = argMaxTheta(sw_demand, v_sw, expectation,␣
↪capacity, iters, average_fare, thetas)
        sw_bumped = simBumped(sw_demand, theta, expectation, capacity)
        bumped_feature = simSatisfaction(sw_bumped, capacity)
        new_features = adjustFeatures(features, bumped_feature)
        sw_d.append(sw_demand)
        revs.append(highest_rev)
        opt_thetas.append(theta)
    return sw_d, revs, opt_thetas
```

[379]:
```
sw_d, revs, opt_thetas = timeVarying(initial_demand_all, features,␣
↪feature_weights, (151 * flights_operated_sw), 0.07)
```

```
80303
198582
100
100
110700
124067
100
100
11347
23864
115930
85635
100
132086
54691
100
207619
100
33289
149292
```

[380]:
```
sw_d
```

[380]:
```
[1905730.4642464726,
 1764631.0433896428,
 1268245.003989278,
 1905719.9431986306,
 1905740.9841783042,
 1676206.1924060578,
 1628267.0661576686,
 1905719.9431986306,
 1905714.6822561736,
 1892076.2353652683,
 1875857.6062240591,
 1654206.6955278509,
 1751438.2403783377,
 1905714.6822561736,
 1598133.4118818832,
 1823200.386454906,
 1905698.8977544962,
 1221349.9688899645,
 1905719.9431986306,
```

```
 1861347.7639104829]
```

[388]:
```
opt_thetas
```

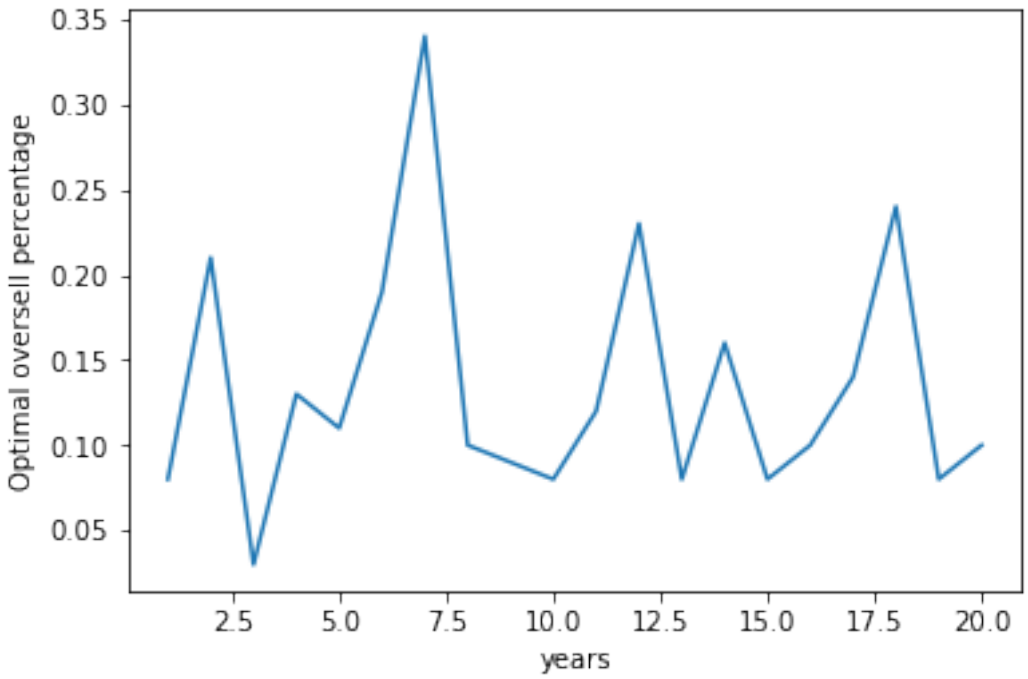
[388]:
```
[0.08,
 0.21,
 0.03,
 0.13,
 0.11,
 0.19,
 0.34,
 0.1,
 0.09,
 0.08,
 0.12,
 0.23,
 0.08,
 0.16,
 0.08,
 0.1,
 0.14,
 0.24,
 0.08,
 0.1]
```

[382]:
```
sw_d
```

[382]:
```
[1905730.4642464726,
 1764631.0433896428,
 1268245.003989278,
 1905719.9431986306,
 1905740.9841783042,
 1676206.1924060578,
 1628267.0661576686,
 1905719.9431986306,
 1905714.6822561736,
 1892076.2353652683,
 1875857.6062240591,
 1654206.6955278509,
 1751438.2403783377,
 1905714.6822561736,
 1598133.4118818832,
 1823200.386454906,
 1905698.8977544962,
 1221349.9688899645,
 1905719.9431986306,
 1861347.7639104829]
```

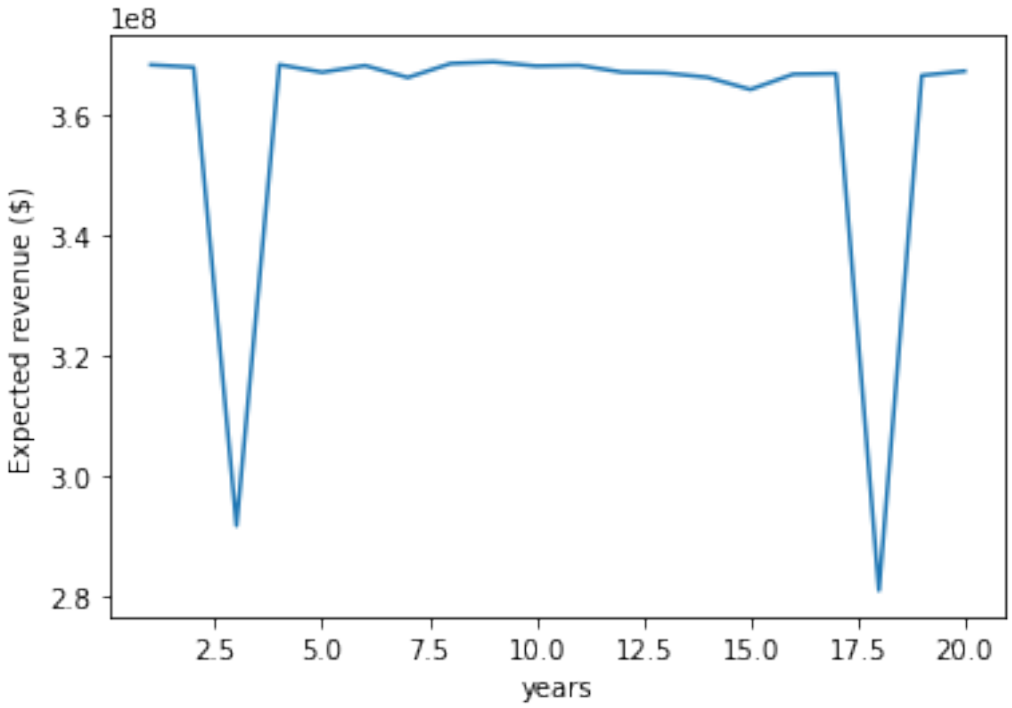
[409]:
```
years = np.arange(1, 21, 1)
plt.plot(years, opt_thetas)
#plt.title('Optimal oversell percentage for each year')
plt.xlabel('years')
plt.ylabel('Optimal oversell percentage')
```

[409]: Text(0, 0.5, 'Optimal oversell percentage')



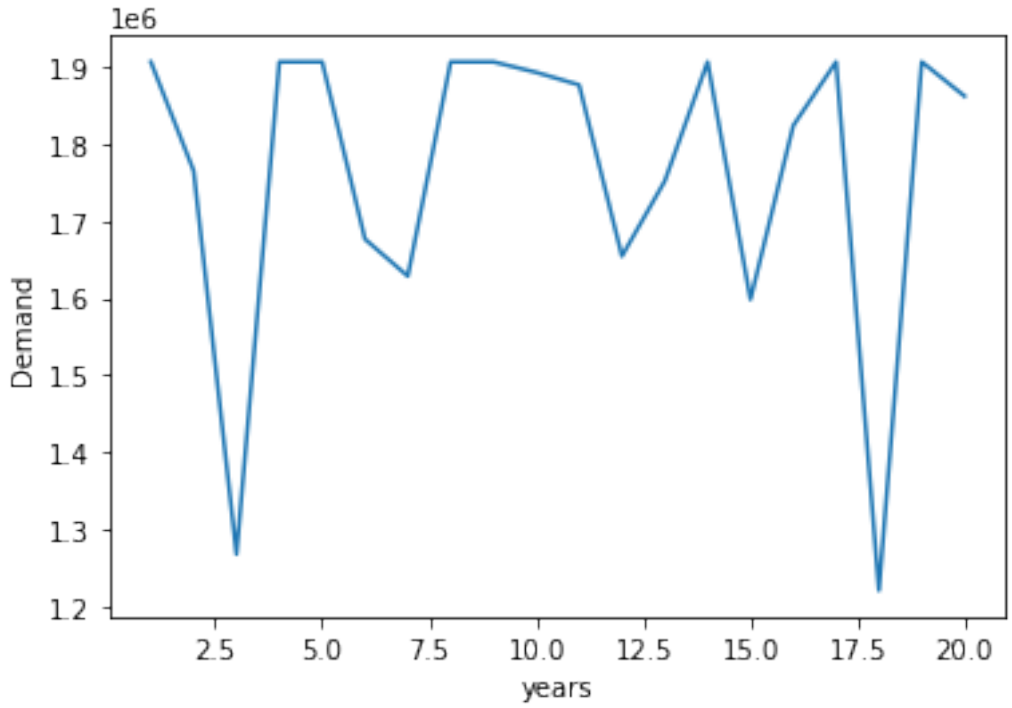
[410]:
```
plt.plot(years, revs)
#plt.title('Expected revenue for each year')
plt.xlabel('years')
plt.ylabel('Expected revenue ($)')
```

[410]: Text(0, 0.5, 'Expected revenue ($)')



[411]:
```
plt.plot(years, sw_d)
#plt.title('Demands for each year')
plt.xlabel('years')
plt.ylabel('Demand')
```

[411]: Text(0, 0.5, 'Demand')


1e6
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
Demand
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
years

[ ]:

demand_with_system_dynamics_winter

December 10, 2021

[1]:
```
clear all
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import pickle
import pdb
```

Read in the number of passengers traveled from OAK airport on domestic flights (all airlines) during the winter months in 2009. We are assuming that the number of passengers traveling out of OAK makes up the demand for the types of flights that our airline of interest, Southwest, flies. This demand will be split among Southwest and it's competitors based on a variety of factors, to be analyzed later on.

[104]:
```
passengers_all_years_all = pd.read_csv('10_year_all_airline_demand.csv')
```

[8]:
```
passengers_2009_all = 374231 + 386569 + 324164
passengers_2009_all
```

[8]:
```
1084964
```

Read in the number of passengers traveled from OAK on Southwest flights during the winter months in 2009

[5]:
```
initial_demand_sw = 281344 + 285764 + 243094
initial_demand_sw
```

[5]:
```
810202
```

[9]:
```
sw_market_share = initial_demand_sw / passengers_2009_all
sw_market_share
```

[9]:
```
0.7467547310325504
```

Read in the number of flights Southwest operated departing from OAK during winter months in 2009

[10]:
```
flights_operated_sw = 3140 + 3199 + 3114
flights_operated_sw
```

[10]:
```
9453
```

Read in the number of flights operated by all airlines departing from OAK during winter months in 2009

[11]:
```
flights_operated_all = 4055 + 4196 + 4114
flights_operated_all
```

[11]:
```
12365
```

[120]:
```
#num_competitors = 6
#features = []
#for i in np.arange(num_competitors + 1):
    #rands = np.random.uniform(0, 1, size = 5)
    #features.append(rands.tolist())
```

[132]:
```
#with open('outfile', 'wb') as feature_list:
    #pickle.dump(features, feature_list)
```

[122]:
```
#with open ('outfile', 'rb') as feature_list:
    #features = pickle.load(feature_list)
```

[133]:
```
beta_southwest = [-5, 1, 1.5, 10, -3]
beta_alaska = [-2, 2, 3, 7, -1]
beta_allegiant = [-5, 0.75, 1, 9, -2]
beta_delta = [-2, 4, 3, 6, -1]
beta_frontier = [-3, 1.5, 1, 10, -2]
beta_hawaian = [-1.5, 4, 3, 2, -2]
beta_spirit = [-7, -1, 0.5, 20, -2]
feature_weights = [beta_southwest, beta_alaska, beta_allegiant, beta_delta,␣
 ↪beta_frontier, beta_hawaian, beta_spirit]
```

[134]:
```
params = []
for i in np.arange(len(features)):
    param_vi = np.exp(np.dot(features[i], feature_weights[i])) #for each airline
    params.append(param_vi)
phi_sw = params[0] / (1 + sum(params))
```

[135]:
```
phi_sw
```

[135]:
```
0.7376104743221286
```

Here is the average load factor for OAK aiport during winter months

[302]:
```
avg_oak_load = (0.7495 + 0.7540 + 0.6743) / 3
avg_oak_load
```

[302]:
```
0.7259333333333333
```

Seat capacity for AirbusA320 and Boeing 737 on average is 151

[303]:
```
seat_capacity = 151 #(https://www.planespotters.net/airline/Southwest-Airlines)
```

We now want to figure out the distribution for demand given the mean load we found above. We consider that our load cannot exceed 100% or (150 passengers per plane), due to the space constraint and therefore when demand is greater than 100% we force it to be 100 to model the load accurately.

[423]:
```
demand_mu = 1500000
demand_var = 770000
max_passengers = flights_operated_all * seat_capacity
target_passengers = flights_operated_all * seat_capacity * avg_oak_load
simulated_demands = np.random.normal(demand_mu, demand_var, size = 100000)
for i in np.arange(len(simulated_demands)):
    if simulated_demands[i] > max_passengers:
        simulated_demands[i] = max_passengers
    else:
        continue
constrained_avg_demand = np.mean(simulated_demands)
initial_demand_all = demand_mu
print('When we constrain demand we get: ', constrained_avg_demand)
print('The "target" demand we want to match from the load above is: ',␣
 ↪target_passengers)
print("Therefore, we will go with an initial demand for all airlines equal to:␣
 ↪", demand_mu)
```

```
When we constrain demand we get:  1342501.8086953273
The "target" demand we want to match from the load above is:  1355401.0156666667
Therefore, we will go with an initial demand for all airlines equal to:  1500000
```

Demand Function

[308]:
```
def simDemand(total_demand, airline_features, feature_weights):
    # beta = sw_weights
    # x = sw_features
    params = []
    for i in np.arange(len(airline_features)):
        param_vi = np.exp(np.dot(airline_features[i], feature_weights[i])) #for␣
 ↪each airline
        params.append(param_vi)
    phi_sw = params[0] / (1 + sum(params))
    demand = total_demand * phi_sw
```

```
    return demand, phi_sw
```

Bumped Function (action / strategy)

[364]:
```
def simBumped(demand, theta, expectation, capacity):
    '''This function simulates the proportion oversold and therefore bumped␣
↪given demand, oversell percentage, capacity, and expected no-show␣
↪probability'''
    #alpha = np.random.exponential(expectation)
    alpha = np.random.exponential(expectation) #no show probability

    sold_tickets = min(capacity * (1 + theta), demand)
    arrivals = np.random.binomial(sold_tickets , 1 - alpha)
    #arrivals = int((1 - alpha) * (min(capacity * (1 + theta), demand)))
    bumped = max(arrivals - capacity, 0)
    if bumped == 0:
        bumped = random.randint(0,1)
    return bumped
```

(Bumped) Satisfaction Level

[365]:
```
def simSatisfaction(number_bumped, capacity):
    if number_bumped == 0:
        bumped_feature = 0
    else:
        count_voluntary = np.random.binomial(number_bumped, 0.9) # 0.9 --> make␣
↪sure to state reasoning for this
        count_forced = number_bumped - count_voluntary
        bumped_feature_int = (2 * count_voluntary) + (20 * count_forced)
        bumped_feature = min(bumped_feature_int / capacity, 1) #WRITE ABOUT THIS
    return bumped_feature
```

Optimizing θ

[366]:
```
def argMaxTheta(demand, v_sw, expectation, capacity, iters, average_fare,␣
↪thetas):
    revenues = []
    for i in np.arange(len(thetas)):
        avg_r = findRevenue(demand, v_sw, thetas[i], expectation, capacity,␣
↪iters, average_fare)
        revenues.append(avg_r)
    highest_rev = max(revenues)
    opt_index = revenues.index(highest_rev)
    opt_theta = thetas[opt_index]
    return revenues, opt_theta, highest_rev
```

[367]:
```
def findRevenue(demand, v_sw, theta, expectation, capacity, iters,␣
↪average_fare):
    revs = []
    for i in np.arange(iters):
        bumped = simBumped(demand, theta, expectation, capacity)
        sold = (min(capacity * (1 + theta), demand))
        count_voluntary = np.random.binomial(bumped, 0.9) # 0.9 --> make sure␣
↪to state reasoning for this
        count_forced = bumped - count_voluntary
        ticket_sales = sold * average_fare
        loss_function = average_fare + (count_voluntary * (min(100 +␣
↪average_fare,775))) + (count_forced*(min(2 * average_fare,775)))
        revenue = ticket_sales - loss_function
        revs.append(revenue)
    average_rev = np.mean(revs)
    return average_rev
```

[368]:
```
def adjustFeatures(old_features, bumped_feature):
    sw_feature = old_features[0]
    sw_feature[0] = bumped_feature
    new_features = old_features
    return new_features
```

[397]:
```
def timeVarying(initial_demand_all, features, feature_weights, capacity,␣
↪expectation):
    thetas = np.arange(0, 0.25, 0.01)
    iters = 100
    average_fare = 230
    sw_d = []
    revs = []
    opt_thetas = []
    for i in np.arange(20):
        sw_demand, v_sw = simDemand(initial_demand_all, features,␣
↪feature_weights) #find SW demand
        r, theta, highest_rev = argMaxTheta(sw_demand, v_sw, expectation,␣
↪capacity, iters, average_fare, thetas)
        sw_bumped = simBumped(sw_demand, theta, expectation, capacity)
        bumped_feature = simSatisfaction(sw_bumped, capacity)
        new_features = adjustFeatures(features, bumped_feature)
        sw_d.append(sw_demand)
        revs.append(highest_rev)
        opt_thetas.append(theta)
    return sw_d, revs, opt_thetas
```

[401]:
```
sw_d, revs, opt_thetas = timeVarying(initial_demand_all, features,␣
↪feature_weights, (151 * flights_operated_sw), 0.07)
```

[402]:
```
sw_d
```

[402]:
```
[1074289.8144737384,
 1074291.5654382496,
 1074291.5654382496,
 1074291.5654382496,
 1074291.5654382496,
 1074274.0554979835,
 1074289.8144737384,
 1074289.8144737384,
 1074289.8144737384,
 1074289.8144737384,
 1074289.8144737384,
 1074291.5654382496,
 1074289.8144737384,
 1074289.8144737384,
 1074291.5654382496,
 1074291.5654382496,
 1074291.5654382496,
 1074289.8144737384,
 1074289.8144737384,
 1074289.8144737384]
```

[403]:
```
opt_thetas
```

[403]:
```
[0.0,
 0.01,
 0.15,
 0.1,
 0.05,
 0.16,
 0.13,
 0.08,
 0.22,
 0.13,
 0.03,
 0.11,
 0.16,
 0.01,
 0.14,
 0.11,
 0.0,
 0.22,
 0.12,
```

```
 0.17]
```

[409]:
```
np.mean(opt_thetas)
```

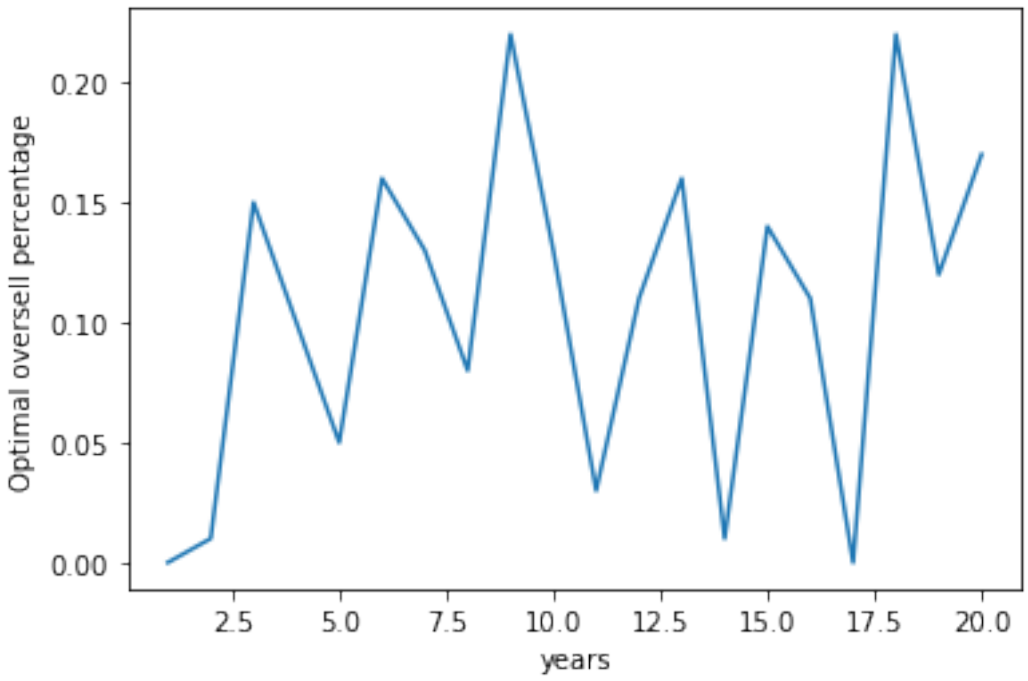
[409]:
```
0.10500000000000001
```

[410]:
```
revs
```

[410]:
```
[247086289.42895994,
 247086694.1507974,
 247086685.55079737,
 247086677.6507974,
 247086708.05079737,
 247082663.56453627,
 247086303.92895994,
 247086285.42895994,
 247086280.22895992,
 247086281.52895993,
 247086282.8289599,
 247086692.1507974,
 247086294.02895993,
 247086288.8289599,
 247086692.1507974,
 247086694.1507974,
 247086695.45079738,
 247086284.22895992,
 247086301.3289599,
 247086306.62895992]
```

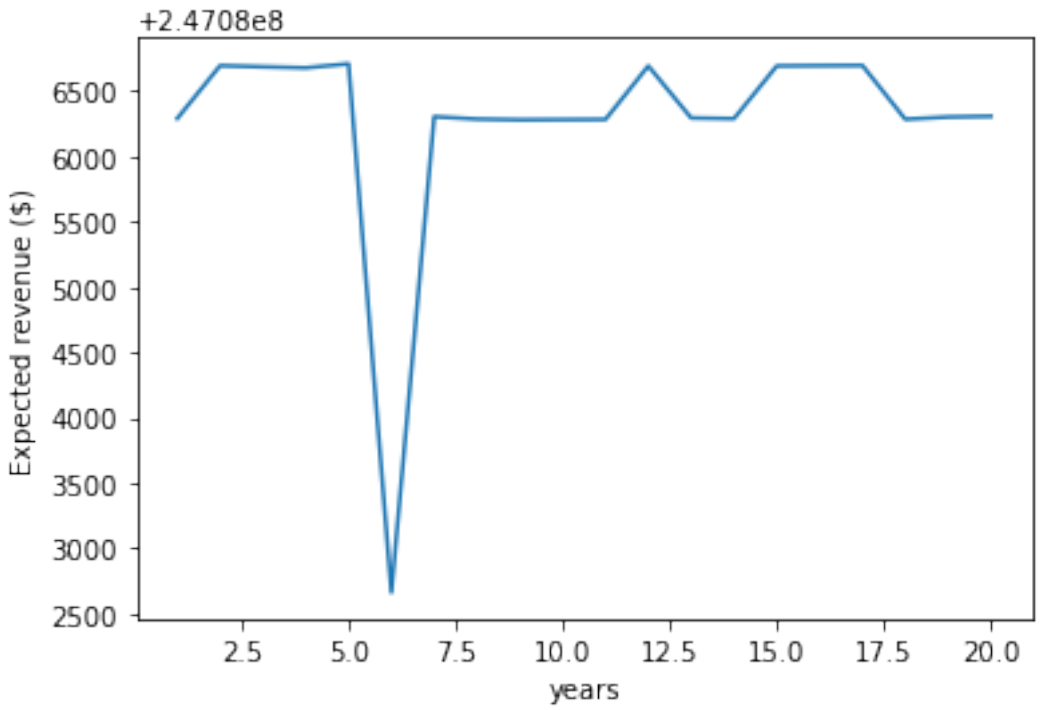
[411]:
```
years = np.arange(1, 21, 1)
plt.plot(years, opt_thetas)
#plt.title('Optimal oversell percentage for each year')
plt.xlabel('years')
plt.ylabel('Optimal oversell percentage')
```

[411]:
```
Text(0, 0.5, 'Optimal oversell percentage')
```



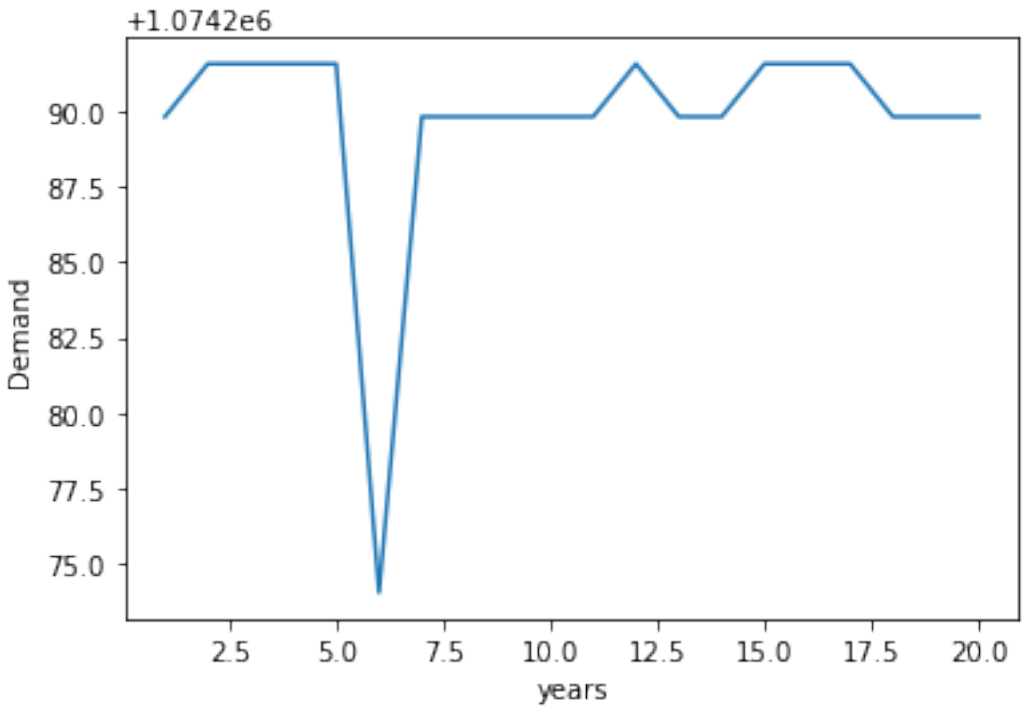
[424]:
```
plt.plot(years, revs)
#plt.title('Expected revenue for each year')
plt.xlabel('years')
plt.ylabel('Expected revenue ($)')
```

[424]: Text(0, 0.5, 'Expected revenue ($)')



[425]:
```
plt.plot(years, sw_d)
#plt.title('Demands for each year')
plt.xlabel('years')
plt.ylabel('Demand')
```

[425]: Text(0, 0.5, 'Demand')


+1.0742e6
90.0
87.5
85.0
82.5
80.0
77.5
75.0
Demand
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
years

[ ]:

demand_with_system_dynamics_winter_MMNL

December 10, 2021

[1]:
```
clear all
```

[55]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import pickle
import pdb
```

Read in the number of passengers traveled from OAK airport on domestic flights (all airlines) during the winter months in 2009. We are assuming that the number of passengers traveling out of OAK makes up the demand for the types of flights that our airline of interest, Southwest, flies. This demand will be split among Southwest and it's competitors based on a variety of factors, to be analyzed later on.

[56]:
```
passengers_all_years_all = pd.read_csv('10_year_all_airline_demand.csv')
```

[57]:
```
passengers_2009_all = 374231 + 386569 + 324164
passengers_2009_all
```

[57]: 1084964

Read in the number of passengers traveled from OAK on Southwest flights during the winter months in 2009

[58]:
```
initial_demand_sw = 281344 + 285764 + 243094
initial_demand_sw
```

[58]: 810202

[59]:
```
sw_market_share = initial_demand_sw / passengers_2009_all
sw_market_share
```

[59]: 0.7467547310325504

Read in the number of flights Southwest operated departing from OAK during winter months in 2009

[60]:
```
flights_operated_sw = 3140 + 3199 + 3114
flights_operated_sw
```

[60]: 9453

Read in the number of flights operated by all airlines departing from OAK during winter months in 2009

[61]:
```
flights_operated_all = 4055 + 4196 + 4114
flights_operated_all
```

[61]: 12365

while phi_sw < 0.7 or phi_sw > 0.77: num_competitors = 6 features_leisure = [] features_business = [] for i in np.arange(num_competitors + 1): rands = np.random.uniform(0, 1, size = 5) features_leisure.append(rands.tolist()) rands1 = np.random.uniform(0, 1, size = 5) features_business.append(rands1.tolist())

```
beta_southwest_l = [-5, 1, 1.5, 10, -3]
beta_southwest_b = [-6, 2, 1.5, 11, -2]
beta_alaska_l = [-1, 3, 4, 7, -1]
beta_alaska_b = [-2, 2, 3, 6, -2]
beta_allegiant_1 = [-5, 0.75, 1, 9, -2]
beta_allegiant_b = [-6, 0.75, 1, 9, -2]
beta_delta_l = [-2, 4, 3, 6, -1]
beta_delta_b = [-3, 4, 3, 6, -1]
beta_frontier_l = [-3, 1.5, 1, 10, -2]
beta_frontier_b = [-4, 1.5, 1, 10, -2]
beta_hawaian_l = [-1.5, 4, 3, 2, -2]
beta_hawaian_b = [-2.5, 4, 3, 2, -2]
beta_spirit_l = [-7, -1, 0.5, 20, -2]
beta_spirit_b = [-8, -1, 0.5, 20, -2]
feature_weights_leisure = [beta_southwest_l, beta_alaska_l, beta_allegiant_l, beta_delta_l, be
feature_weights_leisure = [beta_southwest_b, beta_alaska_b, beta_allegiant_b, beta_delta_b, be


params_leisure = []
for i in np.arange(len(features_leisure)):
    param_vi_leisure = np.exp(np.dot(features_leisure[i], feature_weights_leisure[i])) #for ea
    params_leisure.append(param_vi_leisure)
phi_sw_leisure = params_leisure[0] / (1 + sum(params_leisure))


params_business = []
for i in np.arange(len(features_business)):
    param_vi_business = np.exp(np.dot(features_business[i], feature_weights_business[i])) #for
    params_business.append(param_vi_business)
phi_sw_business = params_business[0] / (1 + sum(params_business))
```

```
phi_sw = (0.4 * phi_sw_business) + (0.6 * phi_sw_leisure)
print(phi_sw)
```

[222]:
```
#with open('outfile_business_winter', 'wb') as feature_list_business:
    #pickle.dump(features_business, feature_list_business)
```

[223]:
```
#with open('outfile_leisure_winter', 'wb') as feature_list_leisure:
    #pickle.dump(features_leisure, feature_list_leisure)
```

[224]:
```
with open ('outfile_business_winter', 'rb') as feature_list_business:
    features_business = pickle.load(feature_list_business)
```

[225]:
```
with open ('outfile_leisure_winter', 'rb') as feature_list_leisure:
    features_leisure = pickle.load(feature_list_leisure)
```

[226]:
```
beta_southwest_l = [-5, 1, 1.5, 10, -3]
beta_southwest_b = [-6, 2, 1.5, 11, -2]
beta_alaska_l = [-1, 3, 4, 7, -1]
beta_alaska_b = [-2, 2, 3, 6, -2]
beta_allegiant_1 = [-5, 0.75, 1, 9, -2]
beta_allegiant_b = [-6, 0.75, 1, 9, -2]
beta_delta_l = [-2, 4, 3, 6, -1]
beta_delta_b = [-3, 4, 3, 6, -1]
beta_frontier_l = [-3, 1.5, 1, 10, -2]
beta_frontier_b = [-4, 1.5, 1, 10, -2]
beta_hawaian_l = [-1.5, 4, 3, 2, -2]
beta_hawaian_b = [-2.5, 4, 3, 2, -2]
beta_spirit_l = [-7, -1, 0.5, 20, -2]
beta_spirit_b = [-8, -1, 0.5, 20, -2]
feature_weights_leisure = [beta_southwest_l, beta_alaska_l, beta_allegiant_l,␣
 ↪beta_delta_l, beta_frontier_l, beta_hawaian_l, beta_spirit_l]
feature_weights_leisure = [beta_southwest_b, beta_alaska_b, beta_allegiant_b,␣
 ↪beta_delta_b, beta_frontier_b, beta_hawaian_b, beta_spirit_b]
```

[227]:
```
params_leisure = []
for i in np.arange(len(features_leisure)):
    param_vi_leisure = np.exp(np.dot(features_leisure[i],␣
 ↪feature_weights_leisure[i])) #for each airline
    params_leisure.append(param_vi_leisure)
phi_sw_leisure = params_leisure[0] / (1 + sum(params_leisure))
```

[233]:
```
params_business = []
for i in np.arange(len(features_business)):
    param_vi_business = np.exp(np.dot(features_business[i],␣
 ↪feature_weights_business[i])) #for each airline
    params_business.append(param_vi_business)
```

```
phi_sw_business = params_business[0] / (1 + sum(params_business))
```

[234]:
```
phi_sw = (0.4 * phi_sw_business) + (0.6 * phi_sw_leisure)
```

[235]:
```
phi_sw
```

[235]: 0.7416076227741981

Here is the average load factor for OAK aiport during winter months

[236]:
```
avg_oak_load = (0.7495 + 0.7540 + 0.6743) / 3
avg_oak_load
```

[236]: 0.7259333333333333

Seat capacity for AirbusA320 and Boeing 737 on average is 151

[237]:
```
seat_capacity = 151 #(https://www.planespotters.net/airline/Southwest-Airlines)
```

We now want to figure out the distribution for demand given the mean load we found above. We consider that our load cannot exceed 100% or (150 passengers per plane), due to the space constraint and therefore when demand is greater than 100% we force it to be 100 to model the load accurately.

[356]:
```
demand_mu = 1450000
demand_var = 600000
max_passengers = flights_operated_all * seat_capacity
target_passengers = flights_operated_all * seat_capacity * avg_oak_load
simulated_demands = np.random.normal(demand_mu, demand_var, size = 100000)
for i in np.arange(len(simulated_demands)):
    if simulated_demands[i] > max_passengers:
        simulated_demands[i] = max_passengers
    else:
        continue
constrained_avg_demand = np.mean(simulated_demands)
initial_demand_all = demand_mu
print('When we constrain demand we get: ', constrained_avg_demand)
print('The "target" demand we want to match from the load above is: ',␣
 ↪target_passengers)
print("Therefore, we will go with an initial demand for all airlines equal to:␣
 ↪", demand_mu)
```

```
When we constrain demand we get:  1360708.3735288829
The "target" demand we want to match from the load above is:  1355401.0156666667
Therefore, we will go with an initial demand for all airlines equal to:  1450000
```

Demand Function

[357]:
```
def simDemand(total_demand, features_leisure, features_business,␣
↪feature_weights_leisure, feature_weights_business):
    # beta = sw_weights
    # x = sw_features
    params_leisure = []
    for i in np.arange(len(features_leisure)):
        param_vi_leisure = np.exp(np.dot(features_leisure[i],␣
↪feature_weights_leisure[i])) #for each airline
        params_leisure.append(param_vi_leisure)
    phi_sw_leisure = params_leisure[0] / (1 + sum(params_leisure))
    params_business = []
    for i in np.arange(len(features_business)):
        param_vi_business = np.exp(np.dot(features_business[i],␣
↪feature_weights_business[i])) #for each airline
        params_business.append(param_vi_business)
    phi_sw_business = params_business[0] / (1 + sum(params_business))
    phi_sw = (0.4 * phi_sw_business) + (0.6 * phi_sw_leisure)
    demand = total_demand * phi_sw
    return demand, phi_sw
```

Bumped Function (action / strategy)

[358]:
```
def simBumped(demand, theta, expectation, capacity):
    '''This function simulates the proportion oversold and therefore bumped␣
↪given demand, oversell percentage, capacity, and expected no-show␣
↪probability'''
    #alpha = np.random.exponential(expectation)
    alpha = np.random.exponential(expectation) #no show probability

    sold_tickets = min(capacity * (1 + theta), demand)
    arrivals = np.random.binomial(sold_tickets , 1 - alpha)
    #arrivals = int((1 - alpha) * (min(capacity * (1 + theta), demand)))
    bumped_all = max(arrivals - capacity, 0)
    if bumped_all == 0:
        bumped_all = random.randint(0,1)


    bumped_leisure = np.random.binomial(0.6, bumped_all)
    bumped_business = max(bumped_all - bumped_leisure, 0)

    return bumped_leisure, bumped_business
```

(Bumped) Satisfaction Level

[391]:
```
def simSatisfactionLeisure(number_bumped, capacity):
    capacity = capacity * 0.6
    if number_bumped == 0:
```

```
        bumped_feature = 0
    else:
        count_voluntary = np.random.binomial(number_bumped, 0.9) # 0.9 --> make␣
↪sure to state reasoning for this
        count_forced = number_bumped - count_voluntary
        bumped_feature_int = (2 * count_voluntary) + (2 * count_forced)
        bumped_feature = min(bumped_feature_int / capacity, 1) #WRITE ABOUT THIS
    return bumped_feature
```

[392]:
```
def simSatisfactionBusiness(number_bumped, capacity):
    capacity = capacity * 0.4
    if number_bumped == 0:
        bumped_feature = 0
    else:
        count_voluntary = np.random.binomial(number_bumped, 0.9) # 0.9 --> make␣
↪sure to state reasoning for this
        count_forced = number_bumped - count_voluntary
        bumped_feature_int = (10 * count_voluntary) + (500 * count_forced)
        bumped_feature = min(bumped_feature_int / capacity, 1) #WRITE ABOUT THIS
    return bumped_feature
```

Optimizing θ

[410]:
```
def argMaxTheta(demand, expectation, capacity, iters, average_fare, thetas):
    revenues = []
    for i in np.arange(len(thetas)):
        avg_r = findRevenue(demand, thetas[i], expectation, capacity, iters,␣
↪average_fare)
        revenues.append(avg_r)
    highest_rev = max(revenues)
    opt_index = revenues.index(highest_rev)
    opt_theta = thetas[opt_index]
    return revenues, opt_theta, highest_rev
```

[411]:
```
def findRevenue(demand, theta, expectation, capacity, iters, average_fare):
    revs = []
    for i in np.arange(iters):
        bumped_leisure, bumped_business = simBumped(demand, theta, expectation,␣
↪capacity)
        bumped = bumped_leisure + bumped_business
        sold = (min(capacity * (1 + theta), demand))
        count_voluntary = np.random.binomial(bumped, 0.9) # 0.9 --> make sure␣
↪to state reasoning for this
        count_forced = bumped - count_voluntary
        ticket_sales = sold * average_fare
```

```
        loss_function = average_fare + (count_voluntary * (min(100 +␣
↪average_fare,775))) + (count_forced*(min(2 * average_fare,775)))
        revenue = ticket_sales - loss_function
        revs.append(revenue)
    average_rev = np.mean(revs)
    return average_rev
```

[412]:
```
def adjustFeatures(old_features, bumped_feature):
    sw_feature = old_features[0]
    sw_feature[0] = bumped_feature
    new_features = old_features
    return new_features
```

[426]:
```
def timeVarying(initial_demand_all, features_leisure, features_business,␣
↪feature_weights_leisure, feature_weights_business, capacity, expectation):
    thetas = np.arange(0, 0.25, 0.01)
    iters = 100
    average_fare = 230
    sw_d = []
    revs = []
    opt_thetas = []
    for i in np.arange(20):
        sw_demand, v_sw = simDemand(initial_demand_all, features_leisure,␣
↪features_business, feature_weights_leisure, feature_weights_business) #find␣
↪SW demand
        #print(v_sw)
        r, theta, highest_rev = argMaxTheta(sw_demand, expectation, capacity,␣
↪iters, average_fare, thetas)
        bumped_leisure, bumped_business = simBumped(sw_demand, theta,␣
↪expectation, capacity)
        bumped_feature_leisure = simSatisfactionLeisure(bumped_leisure,␣
↪capacity)
        bumped_feature_business = simSatisfactionBusiness(bumped_business,␣
↪capacity)

        new_features_leisure = adjustFeatures(features_leisure,␣
↪bumped_feature_leisure)
        new_features_business = adjustFeatures(features_business,␣
↪bumped_feature_business)

        features_leisure = new_features_leisure
        features_business = new_features_business

        sw_d.append(sw_demand)
        revs.append(highest_rev)
        opt_thetas.append(theta)
```

```
    return sw_d, revs, opt_thetas
```

[434]:
```
sw_d, revs, opt_thetas = timeVarying(passengers_2009_all, features_leisure,␣
↪features_business, feature_weights_leisure, feature_weights_business, (151 *␣
↪flights_operated_sw), 0.07)
```

[442]:
```
np.mean(opt_thetas)
```

[442]:
```
0.09350000000000001
```

[443]:
```
sw_d
```

[443]:
```
[1076954.3403236328,
 1076954.3403236328,
 1076954.3403236328,
 1076954.5366038976,
 1076954.3403236328,
 1076954.5366038976,
 1076954.3403236328,
 1076954.3403236328,
 1076954.3403236328,
 1076954.3403236328,
 1076954.5366038976,
 1076954.3403236328,
 1076954.5366038976,
 1076954.5366038976,
 1076954.5366038976,
 1076954.5366038976,
 1076954.3403236328,
 1076954.5366038976,
 1076954.5366038976,
 1076954.5366038976]
```

[444]:
```
opt_thetas
```

[444]:
```
[0.17,
 0.1,
 0.1,
 0.19,
 0.0,
 0.09,
 0.17,
 0.02,
 0.12,
 0.21,
 0.04,
```

```
 0.0,
 0.03,
 0.11,
 0.09,
 0.12,
 0.11,
 0.09,
 0.05,
 0.06]
```

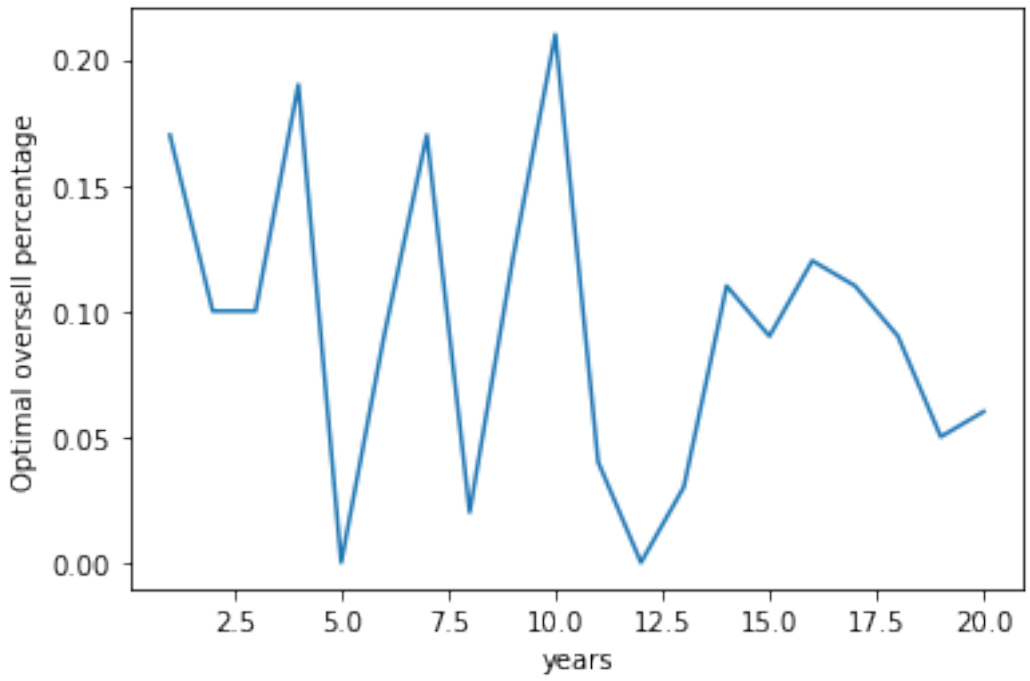
[ ]:

[445]:
```
revs
```

[445]:
```
[247699127.7744356,
 247699131.7744356,
 247699139.67443562,
 247699198.61889648,
 247699138.97443563,
 247699172.31889647,
 247699134.3744356,
 247699132.3744356,
 247699134.3744356,
 247699142.2744356,
 247699173.61889648,
 247699119.17443562,
 247699186.11889648,
 247699183.5188965,
 247699184.81889647,
 247699194.71889648,
 247699137.67443562,
 247699186.11889648,
 247699169.61889648,
 247699175.5188965]
```

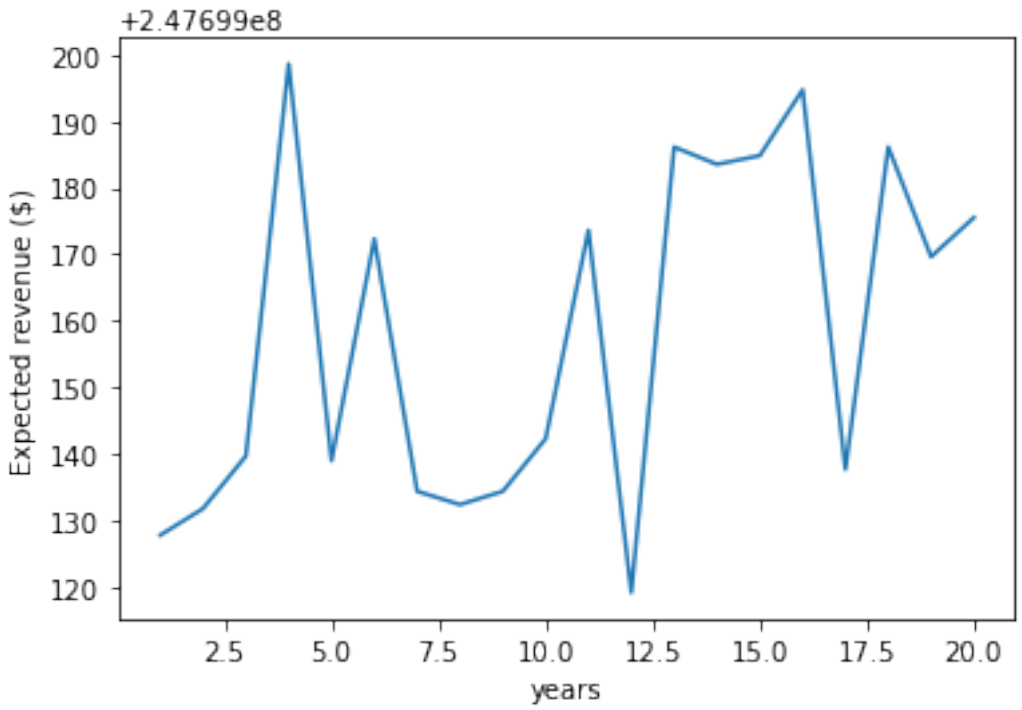
[446]:
```
years = np.arange(1, 21, 1)
plt.plot(years, opt_thetas)
#plt.title('Optimal oversell percentage for each year')
plt.xlabel('years')
plt.ylabel('Optimal oversell percentage')
```

[446]:
```
Text(0, 0.5, 'Optimal oversell percentage')
```



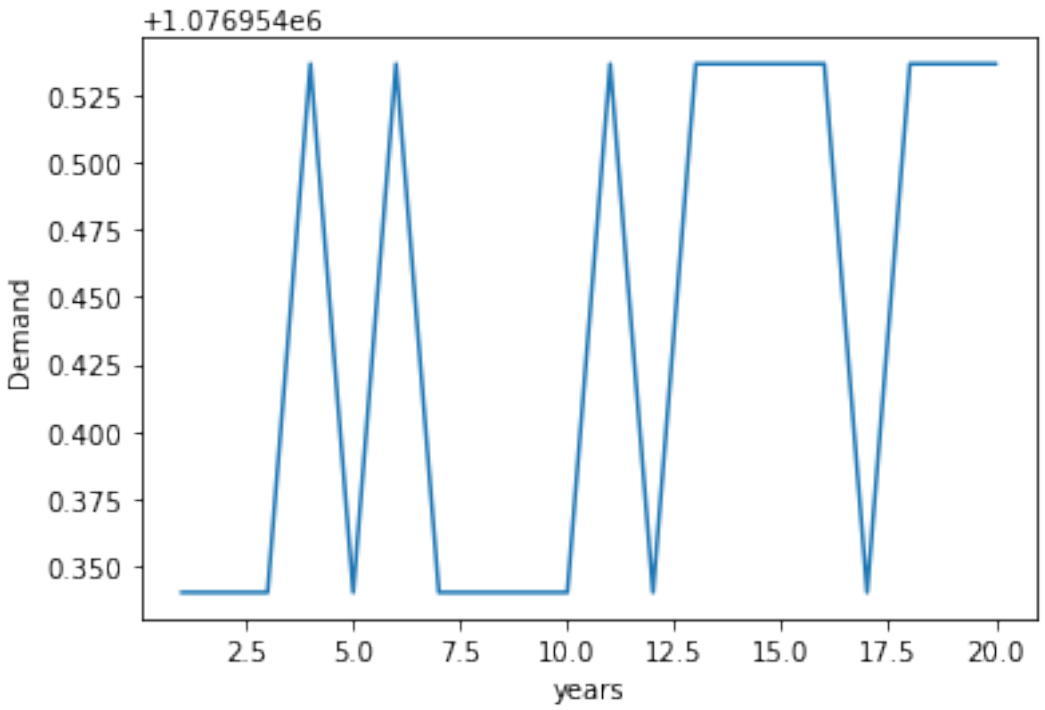
[447]:
```
plt.plot(years, revs)
#plt.title('Expected revenue for each year')
plt.xlabel('years')
plt.ylabel('Expected revenue ($)')
```

[447]: Text(0, 0.5, 'Expected revenue ($)')



[448]:
```
plt.plot(years, sw_d)
#plt.title('Demands for each year')
plt.xlabel('years')
plt.ylabel('Demand')
```

[448]: Text(0, 0.5, 'Demand')


+1.076954e6
0.525
0.500
0.475
0.450
0.425
0.400
0.375
0.350
Demand
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
years

[ ]: