For office use only

T1 _____

T2 _____

T3 _____

T4 _____

Team Control Number

2017019

For office use only

F1 _____

F2 _____

F3 _____

F4 _____

2017

IM^2C

Summary Sheet

"A meeting is an event at which the minutes are kept and the hours are lost" - Anonymous.

In order to make the most of the hours lost in meetings, it is essential to maximize the working ability of each participant by reducing the effects of jet lag and climate shock. In our attempt to maximize productivity at an international meeting, we assessed several different factors.

To start, we analyzed how traveling across time zones affects productivity. To do so we created a vector model to represent the time zones from which each attendee is traveling and used the resultant sum to determine the most efficient time zone for the meeting. Research revealed that eastward travel often causes an upturn in the severity of jet lag symptoms when compared to westward travel; thus we devised a method of weighting the vectors based on travel direction.

Next we created a zoning system to organize the effects of climate and travel time based on latitude and longitude. Climates were identified for each zone and weather data was used to mathematically represent how climatically different each possible location was from the home of the meeting's participants. Using the Haversine formula the team determined distances traveled by the participants to the possible destinations. These results were then translated into sleep loss as a result of flying and subsequently into a percentage of work lost. In sum, we identified the three factors causing jet lag to be changing of time zones, difference in climate, and flight distance, and in the process developed methods to quantify them.

After collecting data for these three factors, the team constructed an algorithm to weight the impacts according to the estimated loss in work productivity as a result of said factors. The algorithm would then output scores for each destination.

Furthermore, the team tested the sensitivity of its model by adjusting the constants determined from research to see how the location decision would be affected. Once the productivity algorithm was complete, the team considered how it could ensure an equal input from each individual at the international meeting. By taking the standard deviations of the impacts on the participants for each zone and then weighting these results against the productivity score, the team could see how the location decision changes depending upon how important equality is.

The team also evaluated the cost of the meeting. Based on distance traveled, sizes of airports, and average hotel costs in the destination cities, it determined a total cost for each zone depending on where attendees came from.

Finally, an easy-to-use Python program was developed to readily visualize the algorithm. The software simply takes in the latitude/longitude pairings of the attendees' hometowns, along with the month and the desired weighting of the productivity, equalization and cost factors. It then spits out a map of the regions where each region is overlaid with its corresponding rank.

After weighting, the team determined final recommendations. The small meeting should occur in Novosibirsk (Russia) and the large meeting in Beirut (Lebanon), Kampala (Uganda), or Perm (Russia), depending upon how important cost, equality of productivity, and total work productivity are according to the standards set by the International Meeting Management Corporation.

Lastly, the two required meetings given to test the algorithm did not include any attendees from Africa or South America. So, the team decided to additionally test the model on the G-20 summit, the Super Bowl of diplomatic conferences, in order to test its limits.

Jet Lag
$\left(IM^2C\right)$ team **2017019**

April 5, 2017

Contents

1 Restatement of the Problem

We have been enlisted by the International Meeting Management Corporation to develop an algorithm to calculate the best place to have meetings based on the home locations of the individuals who will be attending. These attendees will experience several effects from their travels that could negatively impact their performance. Jet lag, differing climates/times of year and overall distance traveled are all considerations that must be taken into account when determining the ideal location for a meeting. Choosing locations that impact the performance of each participant equally and minimizing the cost of the meeting are secondary criteria.

2 Assumptions and Justifications

2.1 Given Rules

- We assume that there are no political problems associated with the meetings. Thus any attendee can travel anywhere in the world without restriction.
- Attendees cannot be brought to the meeting place early to acclimatize to the new location; nor are they allowed to rest after their journeys.

2.2 Further Assumptions

- We assume that attendees will be able to fly into a given city at relatively equal times. Also, there will be no flight delays, crashes, or any other event of that sort. Attendees simply fly to their destination and arrive at around the same time. **Justification:** This is to ensure that all participants are on time to the meeting place. (And ensures their safety).
- The Earth is roughly a sphere. **Justification:** Although the Earth is truly an ellipsoid, if we assume it's a sphere, we will lose very little accuracy, and the calculations of distance will be much easier using properties of spheres.
- The people living in Western China follow a more realistic time zone than the set time for the entire country. **Justification:** All of China has just one time zone. Therefore this assumption allows our algorithm to more accurately model the time zones of the world based upon geographic location.
- We assume that the meeting will be held indoors in a climate-controlled facility. **Justification:** This is the typical venue for a business/science meeting. We don't want attendees getting frostbite or heatstroke, nor being whipped up by winds and storms.
- We assume that 10% of the attendees' time is spent outdoors. **Justification:** People spend part of their day outdoors. A source stated that the average American spends 7% of their day outside[14]. Since the participants are from around the world and they might be interested in sight-seeing during breaks, they will explore their area.
- We assume that all attendees will be affected at the same rate by the various factors causing jet lag. **Justification:** All participants will be working together and thus it is reasonable to declare that they all have similar physical and psychological abilities in combating jet lag.
- We assume that all flights cost the same per mile no matter what the time of year or flight path is. **Justification:** Although flight prices do vary based on both of these, in attempt to simplify the model, we did not account for them. There are too many variables that affect the cost of a flight, so we had to prioritize the most important ones.
- We assume that all attendees will by flying from home to their destination as long as their home is in a different zone than the destination. **Justification:** This problem involves the exploration of jet lag, which is caused by flying at high speeds. Only in extreme situations will an individual feel the effects of jet lag without flying.

3 Variables

a = cost of airfare from home to meeting location

d = distance from home to meeting location

h = average cost cost of a one-night stay for one individual in a hotel in a selected city from a given zone

m = average precipitation of a certain zone

s = current season in a certain time zone

t = average temperature of a certain zone given the season

z_e = number of time zone changes from home to meeting location when traveling eastward

z_w = number of time zone changes from home to meeting location when traveling westward

4 Model Introduction

Our model will assign a score to 75 different regions across the world that we created based off of latitude and longitude. Figure 2 displays these zones on a map. We decided that there are three main influencers that would effect the attendees of a global meeting: distance traveled (along with additional travel time), changing of time zones, and differences in climate. To find out which regions would be the best places for any particular meeting, we developed an algorithm that calculates a score for each region. This score is a weighted sum of three different sub-scores corresponding to the three problems we identified. Each sub-score is a percentage of productivity lost due to one of the three issues. Therefore, lower scores are better scores.

5 Modeling the Effects of Time Zones

5.1 Explanation of the Time Zone Model

There exists well-documented research confirming the detrimental affects of time zone shifts on a traveler's productivity. Changing time zones causes a disruption in the circadian rhythm. According to Source 9, this desynchronization impacts many anatomical functions including sleep cycle, digestion, menstruation, immune system operations, and more. These side affects have a significant influence on productivity, as seen in Source 8. Interestingly, the effects of time zone change are more pronounced when an individual travels east in comparison to traveling west. Source 7 explains that such a trend can be explained by human circadian rhythms, which naturally have a period of more than 24 hours. When a person travels west and the day seems longer, his or her circadian rhythm adjusts more easily. On the other hand, when the person travels east and the day seems shorter his or her circadian rhythm does not adapt as easily. Thus, it would be in the best interest of the International Meeting Management Corporation to minimize the total number of time zones attendees must cross while noting that eastward travel is more detrimental to attendees and distributing travel as equally as possible. In other words, a time zone must be found that distributes the time zone changes as equally as possible for all attendees while paying attention to the minimization of total travel and direction of travel. The team created a MATLAB program that gives equal weight to all time zones from which attendees come from. This program was founded upon the principles of the characteristics of resultant vectors. To start, each time zone was assigned a number: the time zone containing Greenwich, England was assigned 0, the time zone containing Madrid, Spain was assigned 1, and so the pattern continued from east to west for time zones 2-23. It is important to note that it does not matter which time zone is assigned a certain number so long as the numbering follows the previously described pattern. Then a circle was plotted and on its circumference were placed 24 equally-spaced dividers as can be seen in Figure 1. Each divider was assigned a number 0-23 to correspond with the numbered time zones. Next, a vector was drawn from the center of the circle in the direction of the time zone from which a certain attendee came from. This vector also had a magnitude that would correlate to the number of attendees from a certain time zone. For example, if there was one attendee from Greenwich, England and two attendees from Madrid, Spain, there would exist a vector of magnitude 1 in the direction of the 0 (0°) and a vector of magnitude 2 in the direction of the 1 (15°). Please note that units are not important in regard to the magnitude; it only matters that all magnitudes are proportional to each other. After all vectors have been accounted for, the resultant vector is calculated. This vector will point in the direction of the most efficient time zone based on the most equal distribution of total time zones crossed. To explain, in the example represented in Figure 1, the red resultant vector has a direction of 78.435°. Since 78.435° is closest to the interval of 75°, this means that the time zone with the most equal distribution is time zone 5 (Central Russia). However, the direction of travel can also impact a traveler. To account for this, the number of time zones crossed by a westerly attendee when traveling to the time zone designated by the resultant vector is divided by two. For example, someone traveling from Melbourne, Australia to Moscow, Russia crosses 7 time zones while going west, so when determining the best time zone for the meeting this attendee accounts for only 3.5 time zone crossings. This calculation is a suitable estimate of the effects of time zone change according to the descriptions provided by Source 10. No such treatment was applied to eastward travel. So, when the resultant vector points to a certain time zone, the previously described calculations regarding westward travelers are applied and the total time zones crossings

are added up for the time zone designated by the resultant vector and time zones near to it. Essentially, the program found the most equal time zone and from there found the time zone with the fewest overall changes while accounting for the differences in eastward and westward travel. Daylight savings time is a factor which cannot be ignored, as it varies from nation to nation and thus causes disruptions in the time zone numbering pattern. To counter this aspect, the team adjusted the time zone when necessary. For example, if an attendee from London, England was to attend a meeting in Beijing, China in June, the time zone for the individual from England would be entered as 1 and not 0 because in England daylight savings time starts in March and ends in October.

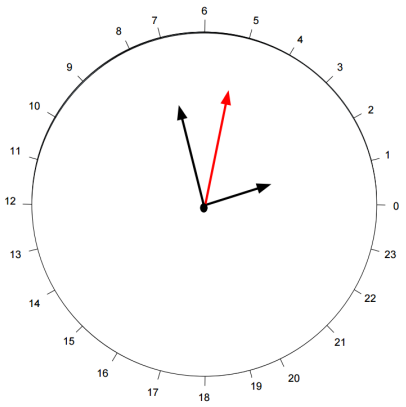

Figure 1: A model with vectors representing how the most effective distribution of time zone travel can be found if a meeting requires one attendee from Zutphen, Netherlands and two attendees from Tomsk, Russia. The red vector represents the resultant vector, which points to Time Zone 5 (Central Russia).

5.2 Results of the Time Zone Model

Using this vector model, it was determined that Time Zone 7 (Central Russia / Southeast Asia) was the most efficient time zone for the small meeting. This time zone presented the most effective combination regarding equality of time zone changes and total time zone changes. For the big meeting, Time Zone 5 (Central Russia / Western Asia) was the time zone that most efficiently balanced equality of time zone changes and total time zone changes.

6 Zone Based Model

After researching the effects of jetlag caused by switching time zones and climates, the team realized that it would need to be able to weight those factors along with distance traveled. For that reason, we developed a general model that could account for the factors of time zone shifting, distance traveled, and acclimatization, weight them, and then generate scores for different corners of the world. To start, the team split the continents into different zones that were 20 degrees of latitude wide and 20 degrees of longitude tall. This pared down to a reasonable number the areas that the team had to compare while still allowing for consideration of the differences amongst zones. Zones that encompassed two areas that were starkly different in climate or time zone were then split into subzones (Southern Spain, for example, has a Mediterranean climate that is quite different from the desert environment of west Algeria). In Figure 2, notice how the zones are numbered starting in the far northwest, and count down and towards the right. Please notice that there is some land, such as Northeast Russia, that is not contained in a zone. Those are areas we felt should not be considered because realistically they are either too remote, too unpopulated, or too cold (or a combination) to offer suitable locations for a meeting.

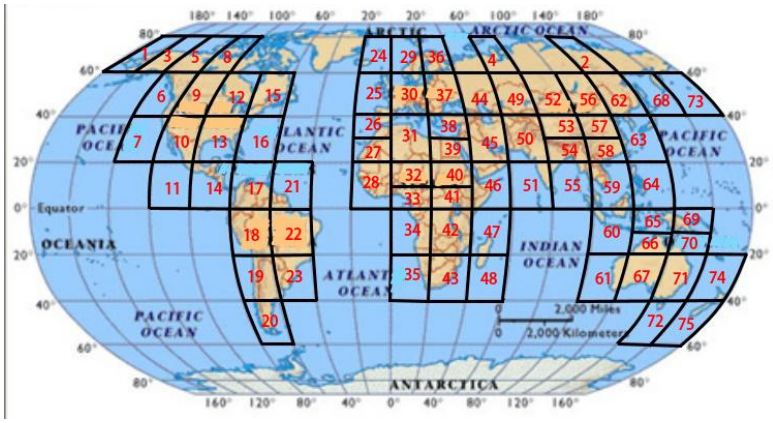

Figure 2: This map shows the layout of our regions and their corresponding numbers. Please note that regions 2 and 4 are located in Siberia.

In the end there were 75 zones, and each one was assigned a climate and time zone. Clearly, the locales within these zones will differ in both regards. However, because the team considered the average climate and time zone of each, this model provides a valuable representation of what the average inhabitant experiences in that location; such representation is the real idea behind this model. Additionally, all of the zones are accurate to within one hour of the time zone and the climates are accurate for the major population centers of each zone.

Using internet sources, we compiled a database of monthly average temperatures and monthly average precipitation using a characteristic city from each zone. This allows us to code the model to take in data for the cities where the members hail from, compare them to each zone, and quickly find the average absolute difference in temperature and precipitation. Also, with this model it is relatively simple to find how close the zones are to the ideal 22 degrees Celsius (as explained in the Climate Effects section).

For time zones, the team used the previous research which stated that traveling to the east was more detrimental to a passenger than traveling west. By assigning each zone a time from 1 to 24 starting at the International Date Line and continuing east, our code takes the difference between the destination and home to count how many time zones a person would cross. Of course if the result is greater than 12 or less than -12, the model has to account for the fact that the person would fly in the opposite direction. To determine how much sleep a person would lose due to jet leg, we divide negative values (westward travel) by two and take the absolute value. Positive values remain unchanged because we determined that traveling to the east causes the average person to lose as much sleep as the number of time zones he or she crosses. These constants are not exact, but they give us a starting point to work from, and can later be reevaluated during the sensitivity analysis portion. This method to model the affects of time zone changes replaced the time zone model from Section 5 to make the time zone calculations compatible with the code for the climate and distance factors.

6.1 Effects of Climate

Looking at the effects of climate, we realized that we are able to separate the world's climates into 16 general climate types. Then we labeled each zone with its dominant climate type, using data from source 20. Subsequently we found average temperature and rainfall data for each separate month at a sample location from each climate and used this data for any zone with the same climate. Using this data, we found the difference in temperature and rainfall levels between the home zone of any attendee and the meeting zone for any given month of the year. Differences between the climates of the home zone and meeting zone have proven to effect work output: as these differences increase, productivity decreases. Finally, when considering the meeting location, a location with a temperature close to 22 degrees Celsius is optimal for work according to source 4, an article titled "Effect of Temperature on Task Performance." All of this information will later be put into the algorithm for determining the best meeting location.

6.2 Distance

The distance the attendees must travel is an importan consideration because long plane rides are exhausting and therefore negatively affect productivity. Thus, we want to minimize the distance each attendee must travel. However, the minimum total distance often is unfair: some attendees will have to travel across the globe, while others might simply drive downtown to the meeting place. We want each attendee to contribute approximately equally to the meeting, so we also need to equalize the distances the people travel. Ideally, we will strike a balance between minimizing the total distance and equalizing the travel of each attendee.

To rank the 75 regions based on distance we developed a Python program based on the Law of Haversines. This trigonometric identity describes spherical triangles, but can be rearranged to calculate the distance between two points on a "great circle." A great circle is a circle around a sphere that has the same circumference as the sphere (a good example is a ball wrapped with rubber bands). Furthermore, the shortest distance between two points on a sphere is the length of the arc of the great circle connecting them. The Haversine Formula is the following (it will be derived in the appendix):

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\rho_2 - \rho_1}{2}\right) + \cos(\rho_1)\cos(\rho_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right).$$

where d is distance (miles), r is the radius of the great circle (circumference of Earth in this case), and the points (ρ_1, λ_1) and (ρ_2, λ_2) correspond to the latitude/longitude pairings of the two points converted to radians.

The program uses the Haversine formula to calculate the distances from each region to each attendee's home town. Then, the program can take those lists of distances and find the total distance for each region and the standard deviation of the distances.

6.3 Travel Time

In order to find the percentage of productivity lost due to travel, we need to convert each distance into time spent traveling. To do this, we need to establish a number of constants. We estimate that a person has to spend a total of 5 hours getting to and from the airport as well as getting through security, waiting at the airport, and getting his or her bags.

$$DriveTime = 5$$

We also estimate that each layover is approximately 5 hours long, giving us:

$$LayoverTime = 5 * Layovers$$

Furthermore, the time an attendee will spend in the air is approximately equal to the distance divided by the average speed of a plane. According to Source 13, the average speed of a plane is around 560 miles per hour. To this time value we can add .2 hours for each takeoff and .32 hours for each landing. These values also account for time lost accelerating to full speed as well as time spent decelerating.

$$FlightTime = \frac{Miles}{FlightSpeed} + (TakeoffTime + LandingTime) * (Layovers + 1)$$

Plugging in the values, we get:

$$FlightTime = \frac{Miles}{560} + (0.52) * (Layovers + 1)$$

We also multiply the distance by 1.1 if there are connections, since this causes the flight distance to increase.

Our final equation is:

$$TotalTravelTime = LayoverTime + FlightTime + DriveTime$$

We can graph this equation for 0 - 3 layovers, as seen in Figure 3, and find the hours of flight time for each attendee when traveling a certain distance.

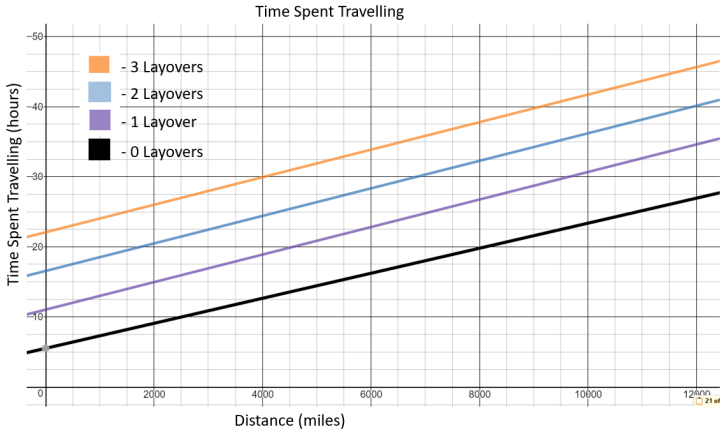

Figure 3: A Graph showing travel times with 0 - 3 layovers

This equation has two inputs which are distance and the number of layovers. We found an equation for distance earlier, but we still need a way to calculate the number of layovers. Looking at flight data from source 5 we see that large cites with a population larger than 1,000,000 require one connection to get to any other large or medium sized city. Medium sized cites with populations 200,000 $<$ population $<$ 1,000,000, can generally reach other medium size cities with 2 layovers. Finally, small cities can get to small and medium sized cities with 3 layovers. Putting this information into a chart, we obtain the following table:

Start City Size	End City Size	Number of Layovers
Small	Small	3
Small	Medium	3
Small	Large	2
Medium	Small	3
Medium	Medium	2
Medium	Large	1
Large	Small	2
Large	Medium	1
Large	Large	1

Figure 4: A Table showing the number of layovers based on the size of the start and end city

After adding up the distances to a certain zone for each attendee, we can put all the distances into the total travel time equation along with the size of the largest city in each zone and the size of the largest city in each attendee's zone. This results are displayed in the graph found in Figure 5.

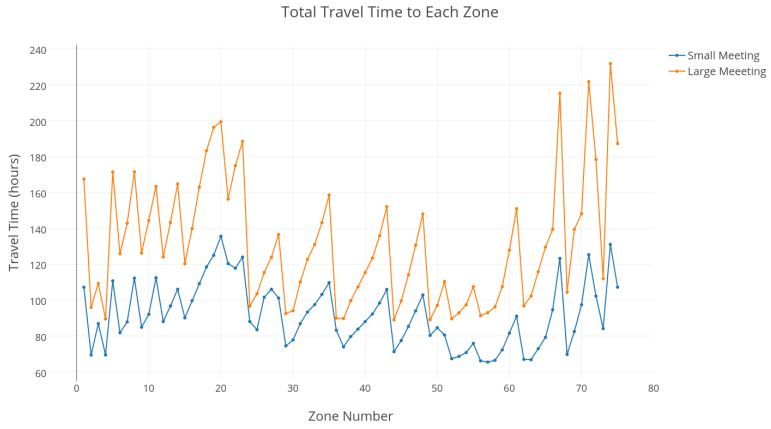

Figure 5: This graph displays the total travel times for both meetings.

7 Weighting the Factors to Determine Scores

The end goal of this model is to assess how the location of the meeting will affect productivity level. Therefore, we needed to determine a way of quantifying the effects of climate, time change, and distance flown on participants' work productivity.

Sleep deprivation directly leads to a decrease in work performance, and the relationship is linear as shown by Source 11. Therefore, once we determined how much sleep a person would lose due to time changes and transportation time, we simply converted that time into a percent of work productivity lost.

The effects of climate were not quite as easy to quantify. The team was able to find a research article about how ambient temperature affects office workers. It also showed a linear relationship, so once we knew how much the meeting location differed from 22 degrees Celsius, we could convert that into a percent work performance loss. On one side, the fact remained that this article was conducted on **indoor** temperatures, and the indoor facility for the meeting would most likely be climate-controlled. Still, we evaluated that people spend about a tenth (Source 17) of their day outside for commute and recreation, so the climate would still affect their overall work performance, but only by about one tenth of what the research article found.

How acclimatization would affect work performance was the most challenging aspect. Of course people are going to be distracted by an abrupt change in temperature or humidity. However, the effects of this are minimal as the participants will be working indoors. Still, sources show that even outdoor temperatures correlate to distracted office workers.[11], so we simply needed to find out how much change equals how much productivity lost. Easier said than done. Using personal data, we have noticed that students are noticeably distracted after a drop of about ten degrees Celsius. The Just Noticeable Percent Difference for humans is right around a 1% change. Using this information, we decided that every 10 degree change in temperature could equate to a 1% loss in productivity. Similarly, we determined that a precipitation change of 100 mm/month (an average 0.325 inches per day) would also correlate to a 1% loss in work efficiency. Granted, these conversion ratios are questionable, and for that reason it is essential to include a sensitivity analysis section, which will come later.

In the end, the algorithm ended up looking like this:

$$L = .8S + .11C + 0.1dT + 0.01dP$$

Where L is the Percent Work Loss, S is sleep deprivation, C is the deviation from 22 degrees Celsius, dT is the average absolute change in Temperature, and dP is the average absolute change in Precipitation.

8 Initial Results

Using this model (and a heck ton of coding, see Appendix 13.2), it was easy to see that for the small meeting Novosibirsk, with an average productivity loss of 6.0096%, should be selected as the meeting spot. Meanwhile, the big meeting should be held in Beirut, which resulted in an average productivity loss of 7.6820%.

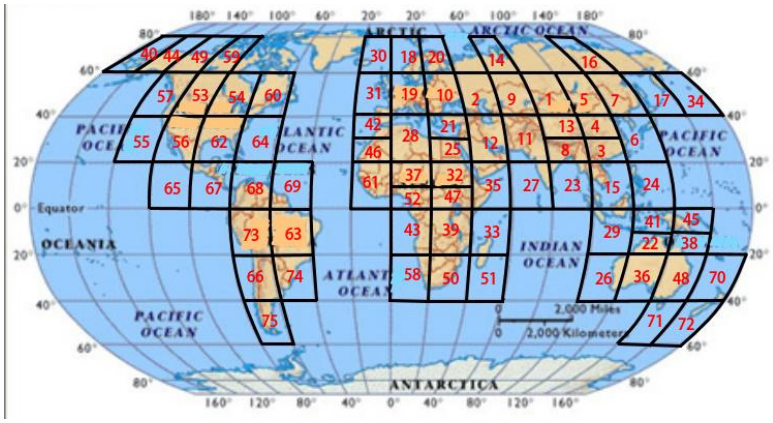

Figure 6: This map shows the rankings of the 75 regions for the Small Meeting.

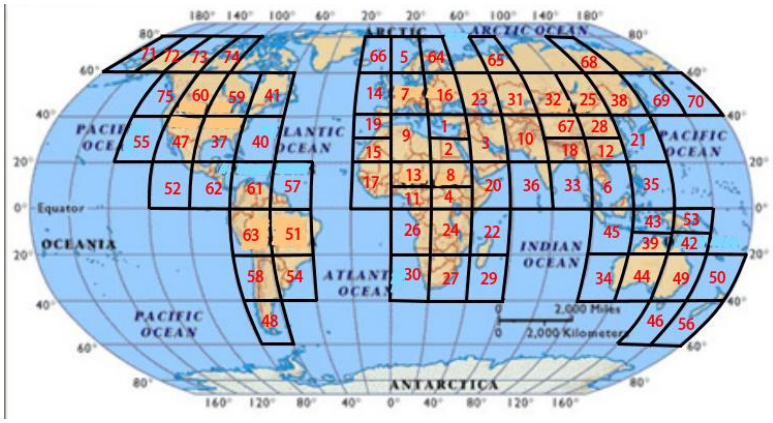

Figure 7: This map shows the rankings of the 75 regions for the Big Meeting.

9 Sensitivity Analysis

There are many variables in this model, and that means many constants. Nobody can say with 100% certainty what these specific constants should be (some of them depend upon personal preference), so we have to check that the constants we found are not skewing the results.

For starters, we will look at the climate effect constants (because those are the ones that we are least confident about). We take the constant affecting the average difference in precipitation and slide it from 0 (which would mena people don't mind temperature difference) to 0.1 (every millimeter difference causes a 0.1% drop in productivity). We graphed how this change in the constant would affect the scores of all 75 zones and then to make it more visually appealing, we narrowed it down to the two that have minimum scores

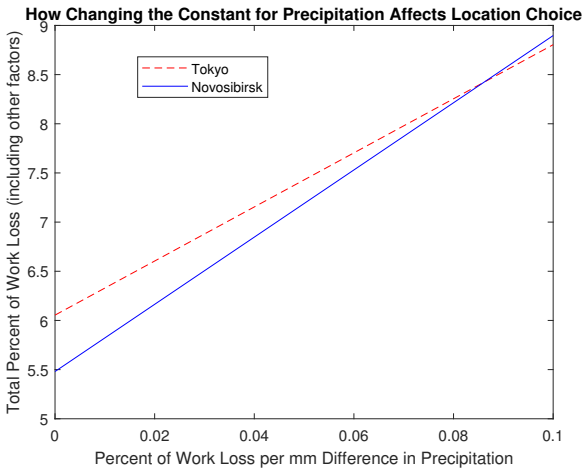

Figure 8: Adjusting the weight of the dP variable within reasonable bounds changes the location choice to Tokyo.

Evidently, if the participants are greatly perturbed by a change in rain/humidity, they should opt for Tokyo as the meeting location. Tokyo receives more rain than Novosibirsk (About 150mm to 50mm), which would be more familiar to the participants from humid climates. However, Tokyo only becomes preferable when the participants feel that a change in 1mm of precipitation would decrease their productivity by 0.83 percent. At this rate, they would be rendered incapable of any work if the precipitation were to differ by more than 120 mm/day, which is clearly unrealistic as people around the world manage to operate when traveling between countries that differ by much more than that. Therefore, our model's result of Novosibirsk remains accurate given what we feel is a reasonable leeway for the precipitation constant.

Using a similar method, we tested the sensitivity of our model to a change in the constants tied to the difference in Temperature, dT, and the closeness to the 22 degrees Celsius factor over a range of [0,0.4] and [0,0.25] respectively (Graphs in Appendix subsection 2). Both indicated that Novosibirsk was the optimal choice. Then we checked the ratios we determined for the hours of sleep lost when traveling across x time zones in an easterly or westerly direction.

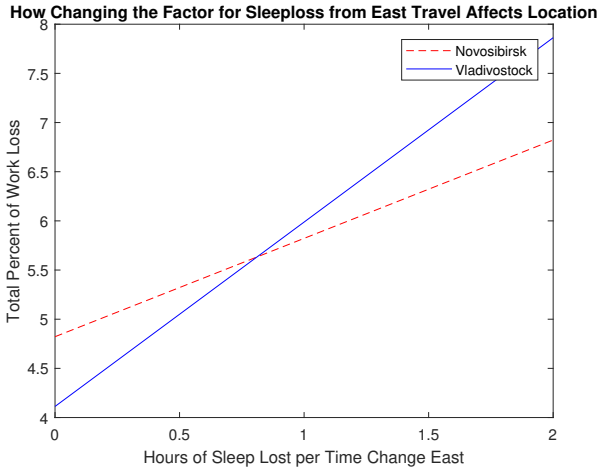

Figure 9: If people lose more than 0.79 hours of sleep from traveling east across a time zone, then Novosibirsk becomes the best choice over Vladivostok

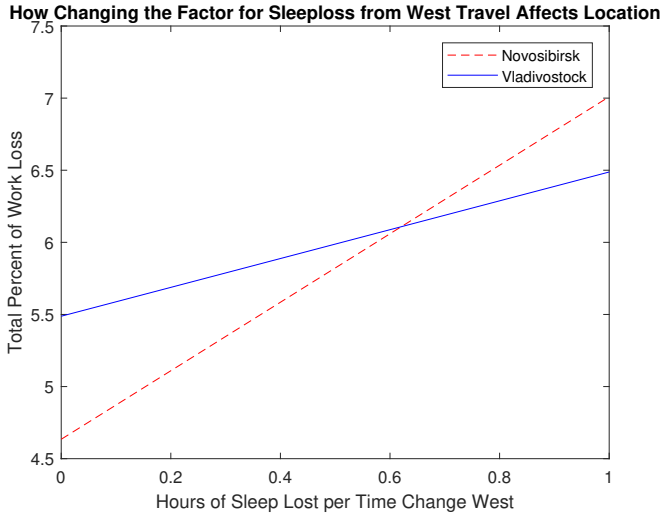

Figure 10: If people lose more than 0.63 hours of sleep from traveling west over a time zone, then Vladivostok becomes optimal.

From these two graphs we see that small, reasonable changes in the time zones to sleep loss ratios do have a significant impact upon the location choice. However, going back to the research on the effect of switching time zones, we remember that traveling east across time zones is more detrimental to a person than traveling west. Novosibirsk's results best fit this evidence as it performs best when the impact from eastward travel is high and westward travel is low. Also, this change in the constant only shifted the location from one Russian city to another one about 500 miles away. Therefore, our initial results must have been fairly accurate.

We also used this method of constant-shifting to check the accuracy of the big meeting results. The precipitation constant did not affect the location choice until it was ramped up to 0.124 (graph in appendix subsection 2), which as previously mentioned, is illogically high. Meanwhile the temperature constants showed an interesting pattern. First we will examine the effects of the constant tied to the average change in degrees Celsius, as shown in Figure 11.

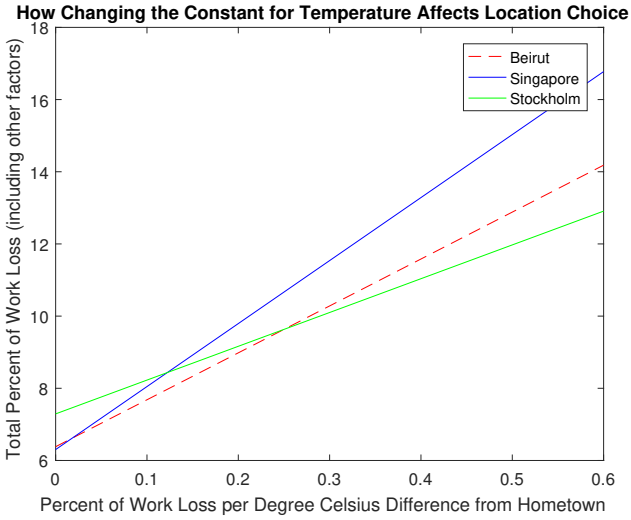

Figure 11: As the similarity in temperature becomes more important, the location choice goes northward.

To look at the temperature constant, we determined a reasonable range from 0 (people don't care about temperature similarity) to 0.5 (2 degrees of outside temperature difference cause 1% loss in concentration) When the constant is minimal, Singapore is the best choice. But if temperature has a moderate impact (0.012 - 0.387% per degree change) then Beirut would be optimal. With a Marine West Coast climate, Stockholm more closely resembles the temperatures of the majority of the participants' hometowns (Boston, Moscow, Warsaw, and Copenhagen), so when a lot of weight is placed upon that value, it reigns supreme. To see scores over the reasonable range, we integrated and divided by the range for an average score, yielding Stockholm with a score of 10.3728 and Beirut with a score of 10.0756. Even if the constant were to fluctuate depending upon the participants' preferences, Beirut remains a reasonable destination.

We also checked the Close To 22 degrees constant with the big meeting, as shown in Figure 12

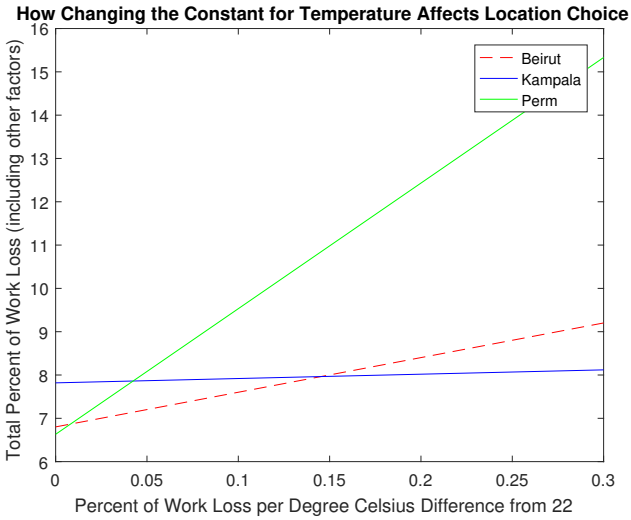

Figure 12: As closeness to 22 degrees Celsius becomes more important, the best location choice shifts Southward.

Again we plotted the scores of the location that were minimal at some point along the range of reasonable constants, and once more we saw that Beirut had the lowest average score.

10 Equalizing the Effects on the Participants

The problem states that each participant should contribute equally, so while our model has found a good spot to boost productivity, we need to check whether it is fairly evaluating the needs of all attendees. For that reason, we can take the impacts on each person and then take the standard deviation. We add up those standard

deviations for each factor using the same algorithm, and we can see how equal the different zones treat the participants.

For the small meeting, the detriments were most equal when the meeting was held in Novosibirsk (Russia), and for the big meeting the most equal treatment occurred in Kampala (Uganda).

While this result greatly implicates that Novosibirsk is a good choice, the Kampala decision is not very close to Istanbul. In fact, it is quite distant from all of the participants, which might be the reason why it had such similar impacts among all 11 participants. It's not a great idea to hurt everyone so that the production rates are more similar. Kampala has a work productivity loss of 9.8703 percent compared to Beirut's 7.6820. But how important is equal participation? Well, when we don't know exactly how to weight something, we turn to graphical analysis.

First we chose a range for how much we will weight the standard deviation results, such as (0,5). Then we selected the locations that are minimums in this range and plot just them. Figure 13 shows how the scores of Beirut and Kampala change as the equalizing score receives more importance.

Figure 13: Beirut was the best before we considered equalizing the effects. When the Equalizing Score was weighted by a mere 0.16 (Essentially saying that equality is 16% as important as productivity), Kampala became optimal.

It's really up to the meeting members how much weight they place on equalizing participant contributions. Still, from here it's not hard to see that Kampala remains the optimal choice for a wide range of reasonable values. Personally, we feel that they should be weighted equally, which would shift the location from Beirut to Kampala.

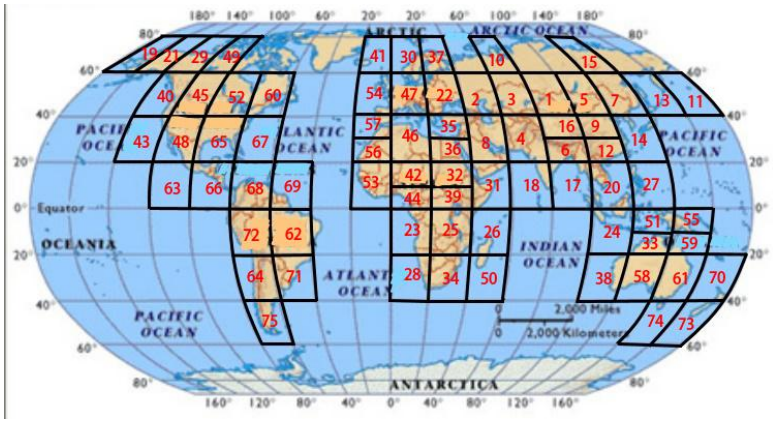

Figure 14: This map shows the rankings of the regions for the small meeting when work productivity and equalized work productivity are weighted equally.

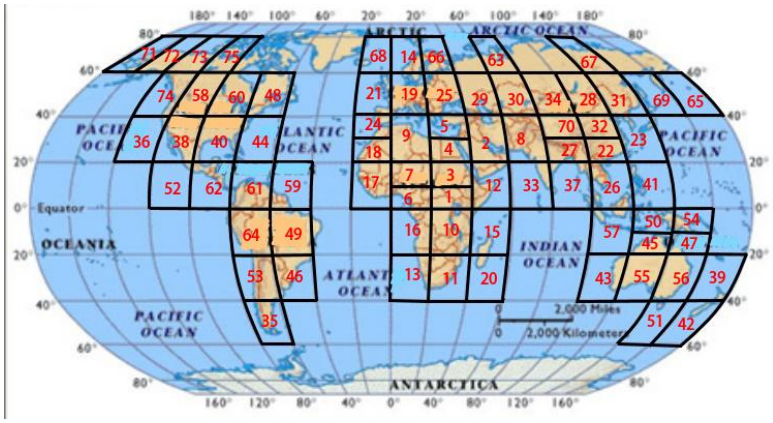

Figure 15: This map shows the rankings of the regions for the big meeting when work productivity and equalized work productivity are weighted equally.

11 Cost

A fundamental concern of any company, including the International Meeting Management Corporation, is cost. For this reason it is important that we factor cost into the decision regarding the ideal meeting location. In doing so, it was determined that there are two aspects that will influence cost: airfare and hotel prices.

11.1 Airfare

An algorithm for airfare was determined by flight data from source 5. The skeleton equation is:

$$Cost = StartingCost * Flights + Costpermile * (miles * inefficiency)$$

The starting cost is the base cost of a plane ticket, which is $50 according to source 18. Inefficiency is how much extra distance is added to the trip if there are connections. This is assumed to be 1.1, the same value used in the Flight Time section. Reworking the equation so that it is solved for Cost per Mile, we get:

$$CostperMile = \frac{Cost - StartingCost * Flights}{inefficiency * miles}$$

Now we can plug in real values from source 5 and find a reasonable estimate for Cost per Mile.

Start City	End City	Calculated Cost per Mile
Berlin	Paris	0.143
Berlin	Mumbai	0.13
Helena	Moscow	0.2
Auckland	New Dehli	0.17

Figure 16: Calculated light Cost per Mile for four different flights.

Averaging these, we get $0.168 per mile. By plugging in the distances and number of flights to each zone for the meetings (found in the explanation of travel time, Section6.3) and multiplying by 2 for the return trip, we get our travel costs. After adding the hotel costs to the travel costs, we get the following graph:

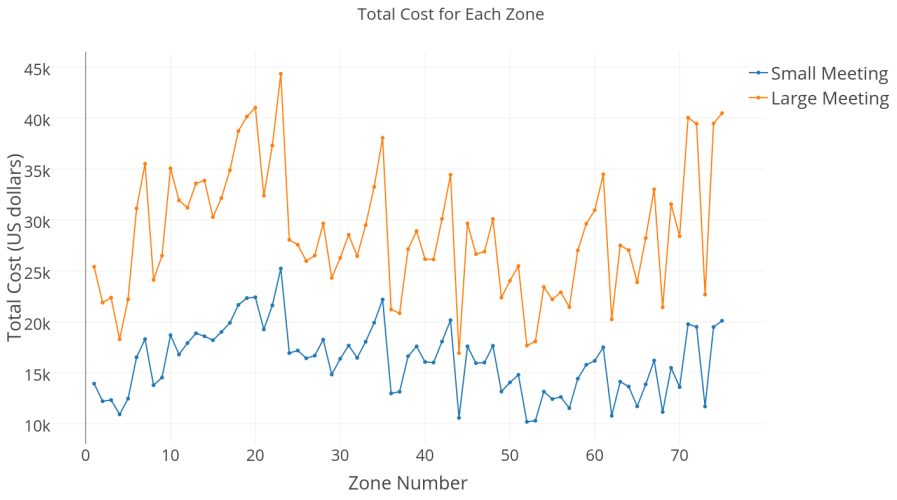

Figure 17: Total cost for flights and hotels for each zone for both meetings.

Novosibirsk is the cheapest option for the small meeting; thus there is no need to worry about cost optimization in regard to the small meeting.

For the larger meeting, we can weight the three factors in a simple formula:

$$Score = Productivity + c_1 * Equality + c_2 * Cost$$

To analyze these factors we plotted planes for each zone showing how the total score changed. Then we looked at the surface plots from beneath to see the minimums, as shown in Figure 18. The upper bound for the cost weight was determined by using the average hotel cost to find the number of dollars for one additional hour of sleep and consequently the percentage points of productivity per dollar as $\frac{1}{341}$ (Formula in Appendix 13.5).

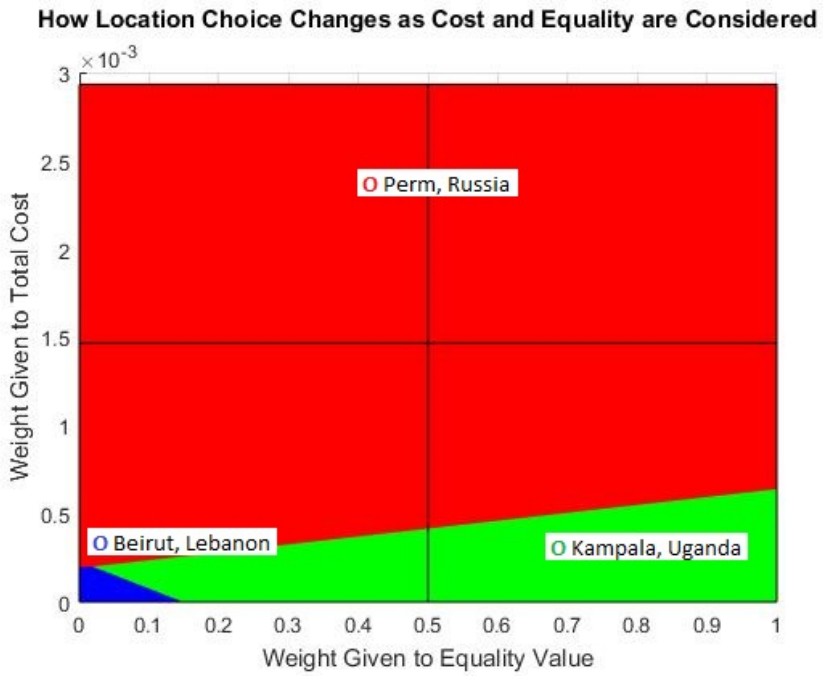

Figure 18: This graph shows which location is optimal for the big meeting depending upon how important cost and equality are over total productivity.

The problem mentioned that cost was not of utmost importance, but if funds are tight our model reveals that Perm (Russia) is the optimal solution. If funds are less important, and equality of contribution is prioritized, then Kampala (Uganda) should be selected. In the case that neither cost nor equality are considered, Beirut is the optimal location.

11.2 Hotel Costs

Hotel costs were calculated as an average of the prices provided by Source 16 for the selected cities from each zone. These calculated nightly hotel prices can be found in Table 16.3.2.

12 Combining the Three Facets of the Algorithm

As stated in the summary, we developed a computer program to readily run our algorithm and to easily visualize the scores our algorithm generates. Depending on how the three scores (productivity, equalization, and cost) are weighted in the code, there will be considerably different results. For example, in Figure 19 and Figure 20 below, the rankings are quite a bit different from our original model based on productivity only. So it really depends on the weights that are given. In the figures below, the categories are weighted equally, but the best scenario according to the IMMC is to weight productivity and equalization more. This really goes to show the extensive ability of our model. If the IMMC does in fact happen to be short on budget, they could simply weight cost more in our algorithm. The program is very intuitive and has the power to accommodate a plethora of needs that the IMMC might have.

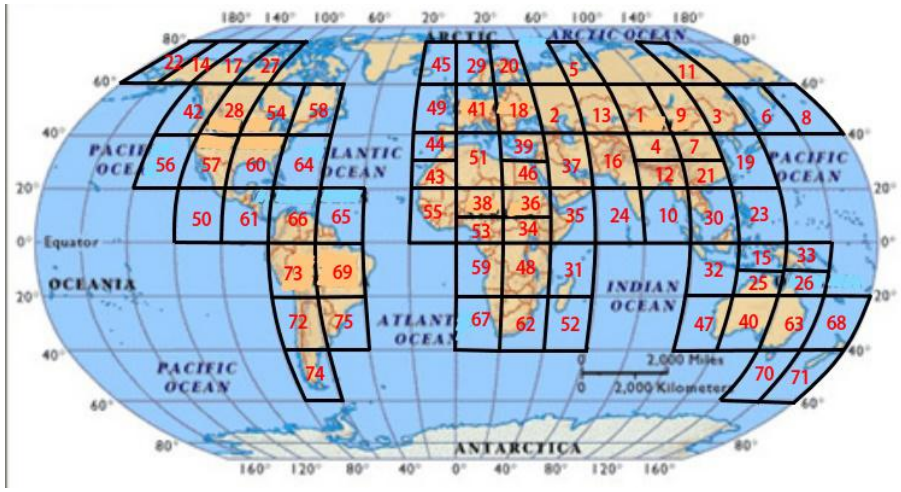

Figure 19: The final rankings for the small meeting using productivity, equalization, and cost, all weighted equally.

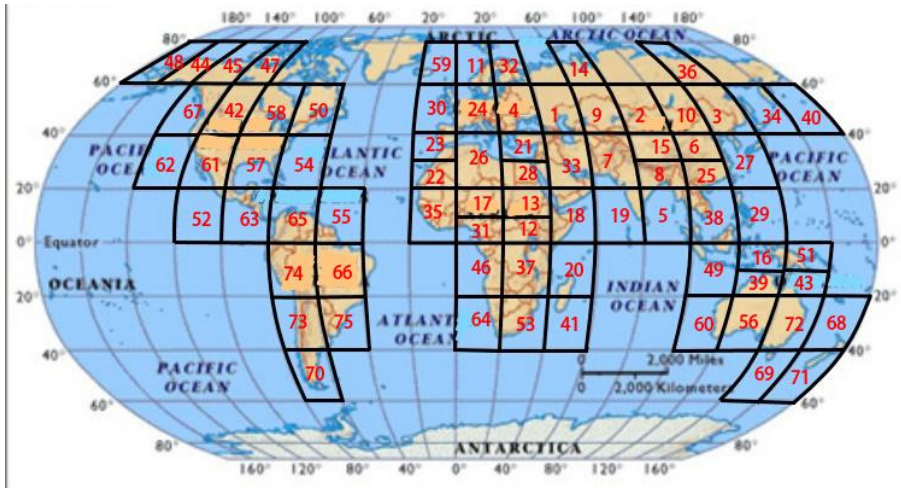

Figure 20: The final rankings for the big meeting using productivity, equalization, and cost, all weighted equally.

13 Final Test: G-20 Summit

The small and big meetings were good warm-ups for our software, but we really wanted to see what it could do. We decided to model the optimal meeting place for 20 foreign leaders at the 2017 G-20 summit, the Super Bowl of diplomatic conferences! The location for the actual summit is Hamburg, Germany; we intend to evaluate that choice.

Very simply, we put in the 20 Latitude/Longitude pairs of the 20 national capitals (one member is the European Union which is headquartered in Brussels, Belgium) and then its month: July. Below is Figure 21 which shows the Productivity rankings for the 75 regions for the Summit. We can see that Hamburg, Germany is a very good selection. (This is because there are a lot of European countries represented.) Then Figure 22 and Figure 23 show similar findings. The top spot is Region 37 for two out of the three measures. Istanbul, Turkey is the corresponding city.

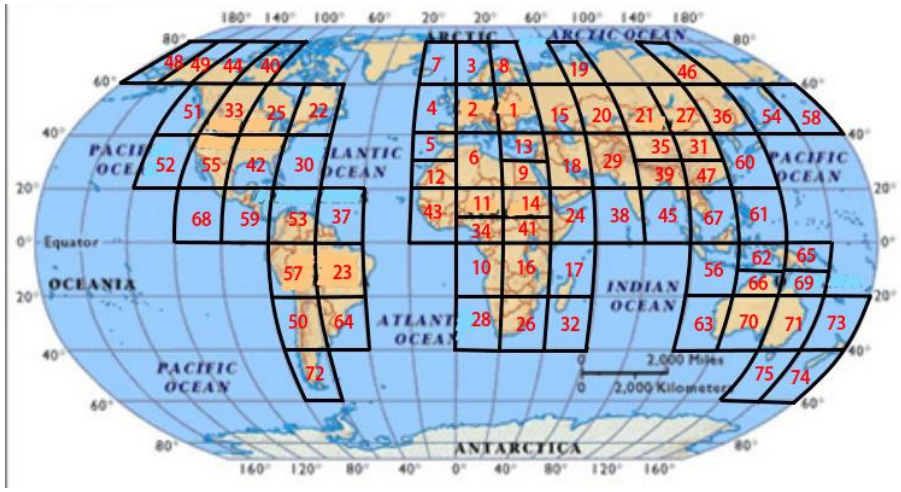

Figure 21: The G-20 rankings based on productivity.

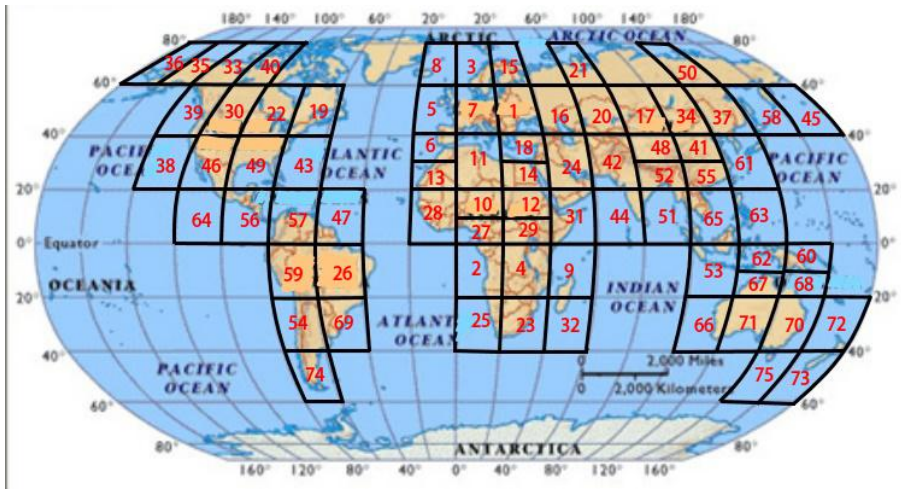

Figure 22: The G-20 rankings based on productivity and equalization weighted equally.

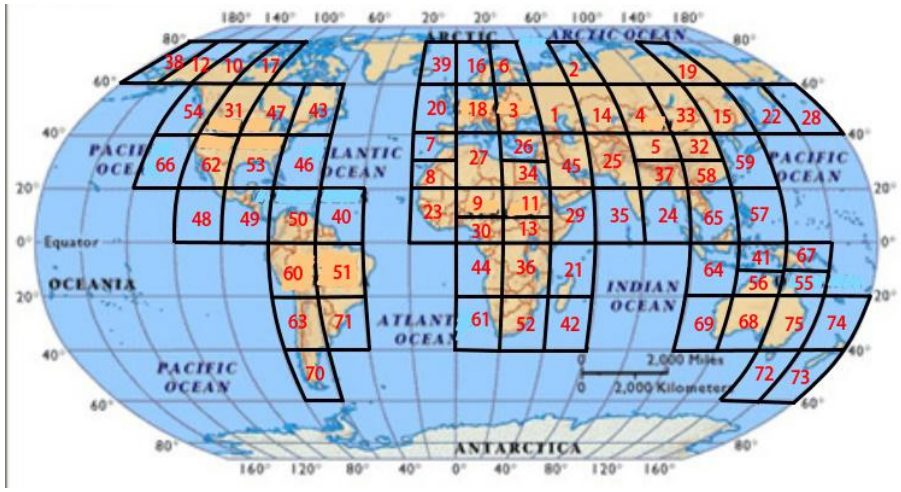

Figure 23: The G-20 rankings based on productivity, cost, and equalization weighted equally.

14 Strengths and Weaknesses

One strength of our model lies in the systematic approach that we took toward the problem. We started by analyzing various factors that impact the location choice, and even used a variety of means in order to check the accuracy of our methods. Then, once we had a grasp upon the individual factors, we created a way to assess all of those together and used a zoning system to organize the locations based on coordinates.

Another strength of our model is that we not only consider a multitude of factors that play into determining the meeting, but we also heavily evaluated the weighting of these factors. For several, we have scholarly articles that empirically support the constants we determined, and for those we were fuzzy on, we tested them within reasonable ranges to verify the sensitivity of our model to them. We did not arbitrarily assume constants.

A weakness of our model is that it generalizes the world just a tad. Some of the zones include different climates or time zones, and while we did our best to find a suitable average, making more zones would have yielded a more realistic model. Had we more time, we could have pursued this option.

15 Conclusion

In conclusion, our model found that using a weighted algorithm based on climate, distance, and time zone crossings we could take in any number of different participants from around the world, plug in their information, and find the location that optimizes the productivity of the meeting. It also determines which location is best depending upon the importance of equal contributions and costs.

For the small meeting, the model shows the Novosibirsk (Russia) not only optimizes work proficiency and equality of contributions, but it also has the lowest cost based upon flights and hotel rooms.

For the large meeting, the model revealed that the best location depends upon how the committee feels about equality and costs. If the budget cannot handle more than $20,000, then Perm (Russia) at $16,938 total should be chosen. If ensuring equal contribution is of utmost importance and a cost of $26,133 is not a concern, then Kampala (Uganda) should be selected. In fact, the initial location choice, Beirut (Lebanon) at $27,141, would not be a wise choice as it is more expensive and less equal than Kampala while only offering a 0.36% increase in overall productivity. Since the committee claimed that money was not a big issue, we recommend Kampala, Uganda as the site for the large meeting in January.

In the final hurrah, our algorithm found that Istanbul, Turkey was an optimal location for the G-20 summit. And the actual location of the 2017 Summit is in Hamburg, Germany, which is in Region 30. Region 30 scored 2nd for productivity in our algorithm, so our model truly does work!!!!

16 Appendices and Citations

16.1 Sources Consulted

1. http://www.thecityedition.com/2012/GPS.html
2. http://www.geomidpoint.com/
3. http://escholarship.org/uc/item/45g4n3rv#page-2
4. Seppanen, Olli; Fisk, William J.; & Lei, Q.H.(2006). Effect of temperature on task performance in office environment. Lawrence Berkeley National Laboratory. Lawrence Berkeley National Laboratory: Lawrence Berkeley National Laboratory. Retrieved from: http://escholarship.org/uc/item/45g4n3rv
5. https://www.google.com/flights
6. http://www.distancefromto.net/
7. http://www.sportsci.org/encyc/jetlag/jetlag.html
8. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656292/
9. Charles Pollak, Thorpy, Michael J., and Jan Yager. "Jet Lag." The Encyclopedia of Sleep and Sleep Disorders. N.p.: Infobase, 2010. 250. Online.
10. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3435929/
11. http://www.businessinsider.com/chart-on-sleep-deprivation-2013-5
12. https://crew.co/blog/how-climate-influences-productivity-why-the-future-of-productivity-doesnt-look-so-hot/
13. B. Kollmeier; T. Brand; B. Meyer (2008). "Perception of Speech and Sound". In Jacob Benesty; M. Mohan Sondhi; Yiteng Huang. Springer handbook of speech processing. Springer. p. 65. ISBN 978-3-540-49125-5.
14. https://www.quora.com/What-is-the-average-speed-of-an-aeroplane
15. http://snowbrains.com/brain-post-much-time-average-american-spend-outdoors/
16. https://www.tripadvisor.com/Hotels
17. http://www.nature.com/jes/journal/v11/n3/full/7500165a.html
18. https://www.rome2rio.com/blog/2013/01/02/170779446/ran
19. https://en.wikipedia.org/wiki/Haversine_formula
20. https://sites.google.com/site/climatetypes/
21. http://www.quotemaster.org/business+meetings
22. Newswise Staff. "Tips to Avoid Jet Lag, Drowsy Driving During Summer Travel." Smart News Connection. Newswise, 02 June 2006. Web. 04 Apr. 2017.

16.2 Graphs for Sensitivity Analysis

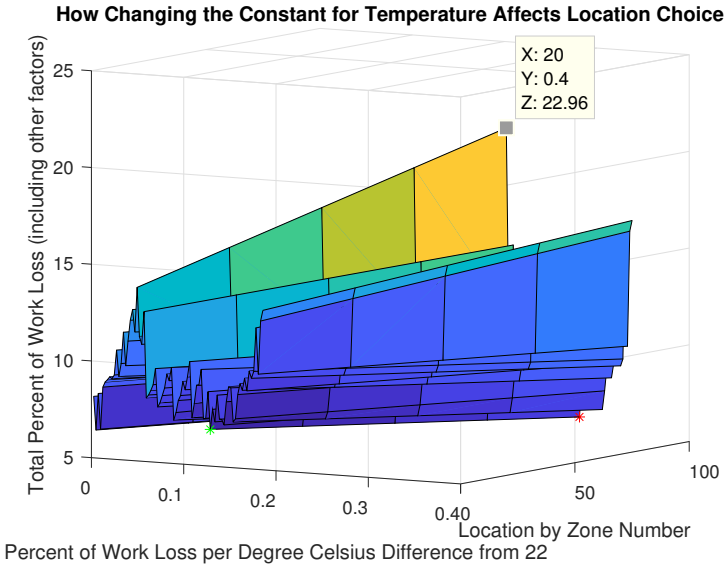

Figure 24: This plot shows how the scores of the 75 zones for the small meeting change as the difference in temperature score (dT) receives more weight. Notice how South Chile skyrockets and Novosibirsk remains at the minimum

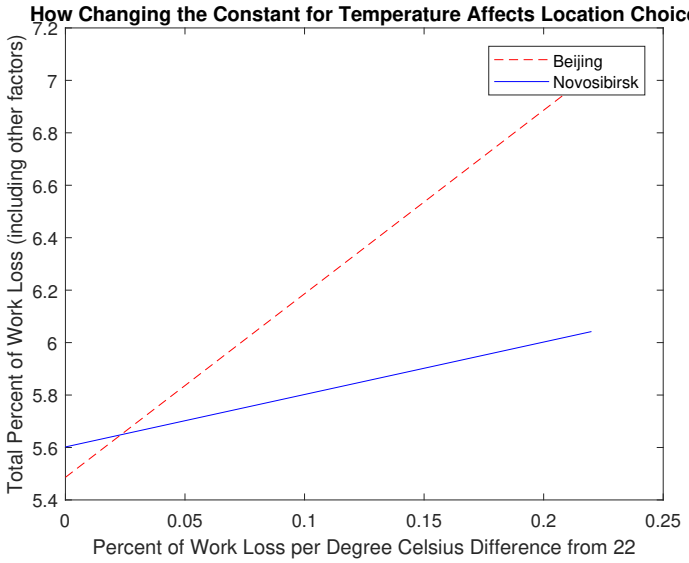

Figure 25: This plot shows how the Percent of Work Loss for Beijing and Novosibirsk for the small meeting change as the Closeness to 22 degrees Celsius score receives more weight.

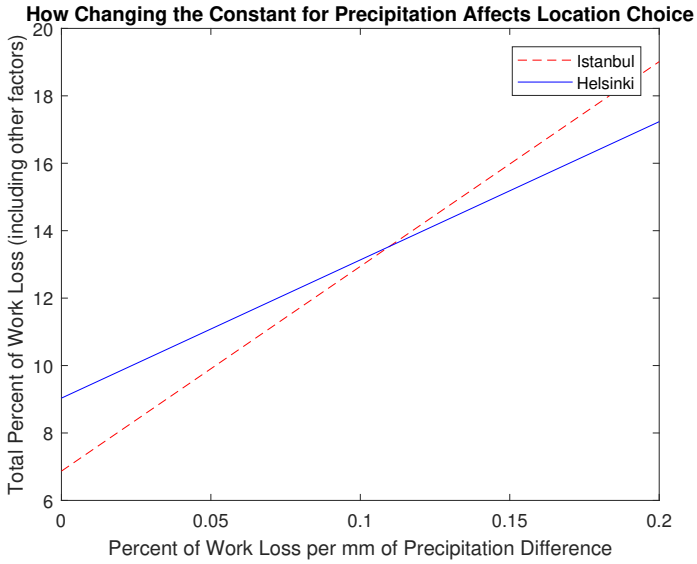

Figure 26: For the big meeting, this plot shows that when more weight is placed on precipitation similarity, Helsinki becomes a better option than Istanbul.

16.3 Tables

16.3.1 Zones

Table 16.3.1: Displays the relative size of the city selected from each zone and the climate of each zone.

Zone	City	Size	Climate
1	Bethel	Small	Subarctic
2	Yakutsk	Big	Subarctic
3	Anchorage	Medium	Subarctic
4	Surgut	Big	Subarctic
5	Whitehorse	Small	Subarctic
6	Seattle	Big	Subarctic
7	San Francisco	Big	Marine West Coast North
8	Yellowknife	Small	Subarctic
9	Calgary	Big	Cold Semi Arid North
10	Los Angeles	Big	Warm Semi Arid North
11	Acapulco	Medium	Tropical Wet/Dry North
12	Chicago	Big	Humid Continental
13	Atlanta	Big	Humid Subtropical North
14	Mexico City	Big	Tropical Wet
15	New York City	Big	Humid Continental
16	Newark	Big	Marine West Coast North
17	Bogotá	Big	Tropical Wet
18	Lima	Big	Tropical Wet
19	Santiago	Big	Warm Semiarid South
20	Comodoro Rivadavia	Medium	Cold Semiarid South
21	Paramaribo	Medium	Tropical Wet
22	Rio de Janeiro	Big	Tropical Wet/Dry South
23	São Paulo	Big	Humid Subtropical South
24	Reykjavik	Medium	Subarctic
25	London	Big	Marine West Coast North
26	Lisbon	Medium	Marine West Coast North
27	Las Palmas de Gran Canaria	Medium	Arid North
28	Conakry	Big	Tropical Wet/Dry North
29	Stockholm	Big	Marine West Coast North
30	Paris	Big	Marine West Coast North
31	Tripoli	Big	Arid North

Zone	City	Size	Climate
32	Niamey	Big	Arid North
33	Abuja	Big	Tropical Wet/Dry North
34	Luanda	Big	Tropical Wet/Dry South
35	Cape Town	Big	Warm Semiarid South
36	Helsinki	Medium	Subarctic
37	Istanbul	Big	Humid Continental
38	Beirut	Big	Warm Semiarid North
39	Cairo	Big	Arid North
40	Khartoum	Big	Arid North
41	Kampala	Big	Tropical Wet/Dry North
42	Lusaka	Big	Warm Semiarid South
43	Maputo	Big	Warm Semiarid South
44	Perm	Big	Humid Continental
45	Dubai	Big	Warm Semiarid North
46	Mogadishu	Big	Arid South
47	Antananarivo	Big	Tropical Wet/Dry South
48	Fianaranantsoa	Medium	Warm Semiarid North
49	Astana	Medium	Cold Semiarid North
50	Delhi	Big	Warm Semiarid North
51	Mumbai	Big	Tropical Wet
52	Novosibirsk	Big	Humid Continental
53	Urumqi	Big	Subarctic
54	Kathmandu	Big	Humid Subtropical North
55	Chennai	Big	Tropical Wet
56	Ulaanbaatar	Big	Cold Semiarid North
57	Beijing	Big	Cold Semiarid North
58	Hong Kong	Big	Humid Subtropical North
59	Singapore	Big	Tropical Wet/Dry North
60	Jakarta	Big	Tropical Wet
61	Perth	Big	Warm Semiarid South
62	Vladivostok	Big	Humid Continental
63	Tokyo	Big	Humid Subtropical North
64	Manila	Big	Tropical Wet
65	Taluk Ambon	Medium	Tropical Wet
66	Darwin	Small	Warm Semiarid North
67	Adelaide	Big	Arid South
68	Sapporo	Big	Subarctic
69	Port Moresby	Medium	Tropical Wet
70	Townsville	Small	Warm Semiarid South
71	Sydney	Big	Warm Semiarid South
72	Hobart	Medium	Marin West Coast South
73	Eastest Russia	Small	Subarctic
74	Aukland	Big	Marine West Coast South
75	Wellington	Big	Marine West Coast South

16.3.2 Costs

Table 16.3.2: Displays the average cost to stay in a hotel for one night in a selected city from each zone for 6 and 11 individuals.

Zone	City	Average Price for 6 Individuals	Average Price for 11 Individuals
1	Bethel	1290	2365
2	Yakutsk	1038	1903
3	Anchorage	672	1232
4	Surgut	606	1111
5	Whitehorse	576	1056
6	Seattle	1698	3113
7	San Francisco	1914	3509
8	Yellowknife	918	1683
9	Calgary	840	1540
10	Los Angeles	1776	3256
11	Acapulco	558	1023
12	Chicago	1770	3245
13	Atlanta	1548	2838
14	Mexico City	864	1584
15	New York City	1734	3179
16	Newark	1404	2574
17	Bogotá	1110	2035
18	Lima	1110	2035
19	Santiago	924	1694
20	Comodoro Rivadavia	978	1793
21	Paramaribo	882	1617
22	Rio de Janeiro	1140	2090
23	São Paulo	1956	3586
24	Reykjavik	2136	3916
25	London	1812	3322
26	Lisbon	1116	2046
27	Las Palmas de Gran Canaria	924	1694
28	Conakry	1062	1947
29	Stockholm	1596	2926
30	Paris	1902	3487
31	Tripoli	1764	3234
32	Niamey	954	1749
33	Abuja	1224	2244
34	Luanda	1488	2728
35	Cape Town	1836	3366
36	Helsinki	1122	2057
37	Istanbul	1062	1947
38	Beirut	1866	3421
39	Cairo	1926	3531
40	Khartoum	1152	2112
41	Kampala	870	1595
42	Lusaka	1170	2145
43	Maputo	1398	2563
44	Perm	378	693
45	Dubai	2328	4268
46	Mogadishu	1284	2354
47	Antananarivo	762	1397
48	Fianarananantsoa	750	1375
49	Astana	1362	2497
50	Delhi	1392	2552
51	Mumbai	1200	2200
52	Novosibirsk	492	902
53	Urumqi	450	825
54	Kathmandu	1266	2321
55	Chennai	702	1287

Zone	City	Average Price for 6 Individuals	Average Price for 11 Individuals
56	Ulaanbaatar	1380	2530
57	Beijing	1062	1947
58	Hong Kong	1962	3597
59	Singapore	2052	3762
60	Jakarta	1596	2926
61	Perth	1446	2651
62	Vladivostok	714	1309
63	Tokyo	1842	3377
64	Manila	1296	2376
65	Taluk Ambon	252	462
66	Darwin	702	1287
67	Adelaide	1038	1903
68	Sapporo	666	1221
69	Port Moresby	1308	2398
70	Townsville	438	803
71	Sydney	2094	3839
72	Hobart	1410	2585
73	Ossora	636	1166
74	Aukland	1644	3014
75	Wellington	1296	2376

16.4 Final Software

We wrote a ton of code in both MATLAB and Python to model different parts of the algorithm. These different pieces can be seen in their various stages in the next appendix subsection. However, all of these bits of code only were built to model the small and big meetings given to us. That is not a true algorithm (at least to us), because it is not automated. So we set out to make a Python program that only requires latitude/longitude input and the month of the meeting to produce our ranking.

Most of the program blends together many of our preliminary code bits. But the piece that makes it tick is the Region Converter function, which converts any lat/long pair into its respective region. Based off of this we can use all of the various lists we made to gather the different scores.

The most important part of the code for visualization is the printTotalScore function. It takes the rank of a region and overlays the text over a map on which we photoshopped the regions.

```
import numpy as np
import math as m
from PIL import ImageFont
from PIL import Image
from PIL import ImageDraw


regionList = [[70,-170],[70,130],[70,-150],[70,70],[70,-130],[50,-130],[30,-130],[70,-110]
                  ,[50,-110],[30,-110],[10,-110],[50,-90],[30,-90],
                  [10,-90],[50,-70],[30,-70],[10,-70],[-10,-70],[-30,-70],[-50,-70]
                  ,[10,-50],[-10,-50],[-30,-50],[70,-10],[50,-10],[35,-10],[25,-10],[10,-10],    #These
                  [70,10],[50,10],[30,10],[15,10],[5,10],[-10,10],[-30,10],                      #All 75
                  [70,30],[50,30],[35,30],[25,30],[15,30],[5,30],[-10,30],[-30,30],[50,50]
                  ,[30,50],[10,50],[-10,50],[-30,50],[50,70],[30,70],[10,70],[50,90],[35,90],[25,90]
                  ,[10,90],[50,110],[35,110],[25,110],[10,110],[-10,110],[-30,110],[50,130],
                  [30,130],[10,130],[-5,130],[-15,130],[-30,130],[50,150],[-5,150],[-15,150]
                  ,[-30,150],[-50,150],[50,170],[-30,170],[-50,170]]

mapCoordList = [(132,40),(558,48),(154,42),(468,46),(181,43),(148,81),(125,125),(212,43),(182,80),(163,
                (155,170),(217,82),(200,125),(195,170),(252,80),(240,125),(235,170),(231,215),(235,255)
                (271,168),(272,214),(273,255),(354,42),(350,78),(349,108),(349,135),(347,164),(383,43),
                (385,119),(388,157),(387,180),(387,210),(387,253),(410,43),(418,80),(422,110),(426,132)
                (426,178),(424,210),(424,257),(452,84),(462,126),(465,167),(464,212),(461,257),(487,84)
                (496,122),(502,167),(523,84),(536,110),(539,133),(542,167),(557,84),(568,110),(574,134)
                (580,212),(572,257),(587,85),(606,123),(618,166),(620,202),(617,223),(609,255),(628,85)
                (655,223),(647,257),(622,298),(661,86),(682,256),(652,301)]
```

```
TWT = [27, 27, 27, 27, 26, 26, 25, 27, 27, 27, 27, 27] #TropWetTemp
TWP = [250, 280, 260, 320, 300, 200, 170, 160, 190, 200, 240, 220]

TDNT = [21, 20, 21, 22, 23, 24, 26, 27, 27, 27, 26, 25] #TropWetDryNorthTemp
TDNP = [10, 10, 10, 30, 100, 200, 300, 550, 470, 200, 105, 15]

TDST = [23, 23, 23, 23, 23, 20, 20, 23, 25, 24, 23, 23] #TropWetDrySouthTemp
MT = [7, 8, 10, 13, 17, 21, 24, 24, 21, 16, 11, 8] #Mediteranean, Temp
MP = [104, 99, 69, 66, 38, 23, 13, 23, 59, 84, 120, 112]

HSNT = [7, 8, 11, 17, 21, 26, 28, 27, 24, 17, 11, 7] #HumiSubNorthTemp
HSNP = [150, 120, 130, 120, 100, 90, 80, 70, 60, 70, 100, 120]

HSST = [28, 27, 24, 17, 11, 7, 7, 8, 11, 17, 21, 26]
TDSP = [225, 220, 230, 110, 10, 0, 0, 5, 25, 90, 185, 230]

HSSP = [70, 60, 70, 70, 100, 120, 150, 120, 130, 120, 100, 80]

MNT = [2, 4, 6, 8, 12, 13, 17, 16, 14, 10, 6, 4] #MariWestCoasNorthTemp
MNP = [140, 120, 100, 60, 40, 50, 20, 30, 60, 120, 140, 150]

MST = [17, 16, 14, 10, 6, 4, 2, 4, 6, 8, 12, 13]
MSP = [20, 30, 60, 120, 140, 150, 140, 120, 100, 60, 40, 50]

HCT = [-7, -5, 3, 10, 15, 20, 23, 22, 18, 12, 5, -3] #HumiContTemp
HCP = [50, 40, 70, 90, 90, 100, 90, 90, 70, 60, 60, 50]

ST = [-43, -39, -25, -5, 5, 12, 15, 11, 5, -7, -30, -40] #SubarcticTemp
SP = [15, 10, 10, 15, 25, 30, 40, 35, 20, 15, 15, 10]

ANT = [13, 14, 17, 20, 23, 26, 27, 27, 25, 23, 21, 15] #Arid, North, Temp
ANP = [3, 2, 1, 1, 0, 0, 0, 0, 0, 0, 2, 3]

AST = [27, 27, 25, 23, 21, 15, 13, 14, 17, 20, 23, 26]
ASP = [0, 0, 0, 0, 2, 3, 3, 2, 1, 1, 0, 0]

WSNT = [14, 17, 20, 24, 26, 27, 28, 28, 25, 22, 18, 16] #WarmSemiNorthTemp
WSNP = [19, 18, 19, 32, 55, 71, 50, 85, 145, 69, 22, 15]

CSNT = [-7, -3, 1, 5, 10, 15, 19, 18, 13, 7, 0, -5] #Cold, Semi, North, Temp
CSNP = [15, 10, 18, 22, 44, 46, 31, 30, 29, 15, 14, 15]

WSST = [28, 28, 25, 22, 18, 16, 14, 17, 20, 24, 26, 27] #WarmSemiNorthTemp
WSSP = [50, 85, 145, 69, 22, 15, 19, 18, 19, 32, 55, 71]

CSST = [19, 18, 13, 7, 0, -5, -7, -3, 1, 5, 10, 15] #ColdSemiNorthTemp
CSSP = [31, 30, 29, 15, 14, 15, 15, 10, 18, 22, 44, 46]

Temp = [ST, ST, ST, ST, ST, ST, MNT, ST, CSNT, WSNT, TDNT, HCT, HSNT, TWT, HCT,
          MNT, TWT, TWT, WSST, CSST, TWT, TDST, HSST, ST, MNT, MNT, ANT, TDNT, MNT,
            MNT , ANT, ANT, TDNT, TDST, WSST , ST, HCT, WSNT, ANT, ANT, TDNT, TDST, WSST ,
              HCT, WSNT, AST, TDST, WSST, CSNT, WSNT, TWT, HCT, ST , HSNT, TWT, CSNT, CSNT,
                HSNT, TDNT, TWT , WSST, HCT, HSNT, TWT, TWT , WSST, AST, ST, TWT, WSST, WSST,

Prec = [SP, SP, SP, SP, SP, SP, MNP, SP, CSNP, WSNP, TDNP, HCP, HSNP, TWP, HCP, MNP,
          TWP, TWP, WSSP, CSSP, TWP, TDSP, HSSP, SP, MNP, MNP, ANP, TDNP, MNP, MNP , ANP,
            ANP, TDNP, TDSP, WSSP , SP, HCP, WSNP, ANP, ANP, TDNP, TDSP, WSSP , HCP, WSNP,
              ASP, TDSP, WSSP, CSNP, WSNP, TWP, HCP, SP , HSNP, TWP, CSNP, CSNP, HSNP, TDNP
                TWP , WSSP, HCP, HSNP, TWP, TWP , WSSP, ASP, SP, TWP, WSSP, WSSP, MSP, SP,
```

```
winterTime = [2, 19, 2, 14, 3, 3, 3, 4, 4, 3, 5, 5, 6, 5, 6, 6, 4, 5, 6, 7, 6, 7, 7, 10, 11, 11, 11,
              9, 12, 12, 11, 11, 11, 11, 12, 13, 12, 13, 12, 12, 12, 13, 12, 15, 14, 13, 13, 13, 16,
               15.5, 15.5, 17, 18, 16.5, 15.5, 18, 18, 18, 18, 17, 18, 20, 19, 18, 19, 19.5, 19.5, 19,
                20, 20, 20, 20, 24, 24, 24]

summerTime = [1, 19, 1, 14, 2, 2, 2, 3, 3, 2, 4, 4, 5, 4, 5, 5, 4, 5, 7, 8, 6, 8, 8, 10, 10, 10, 10,
              9, 11, 11, 11, 11, 11, 11, 12, 12, 12, 13, 12, 12, 12, 13, 12, 15, 14, 13, 13, 13, 16,
               15.5, 15.5, 17, 18, 16.5, 15.5, 18, 18, 18, 18, 17, 19, 20, 19, 19, 20, 19.5, 19.5, 19,
                20, 21, 21, 21, 24, 1, 1]

end_city_sizes = ['small','big','med','big','small','big','big','small','big','big','med','big','big',
                  'big','big','big','big','big','big','med','med','big','big','med','big','med','med',
                  'big','big','big','big','big','big','big','big','med','big','big','big','big','big',
                  'big','big','big','big','big','big','big','med','med','big','big','big','big','big',
                  'big','big','big','big','big','big','big','big','big','big','med','small','big','big'
                  'med','small','big','med','small','big','big']

hotelCosts = [215.0, 173.0, 112.0, 101.0, 96.0, 283.0, 319.0, 153.0, 140.0, 296.0, 93.0, 295.0, 258.0,
              289.0, 234.0, 185.0, 185.0, 154.0, 163.0, 147.0, 190.0, 326.0, 356.0, 302.0, 186.0, 154.
               266.0, 317.0, 294.0, 159.0, 204.0, 248.0, 306.0, 187.0, 177.0, 311.0, 321.0, 192.0, 145
                233.0, 63.0, 388.0, 214.0, 127.0, 125.0, 227.0, 232.0, 200.0, 82.0, 75.0, 211.0, 117.0
                 177.0, 327.0, 342.0, 266.0, 241.0, 119.0, 307.0, 216.0, 42.0, 117.0, 173.0, 111.0, 21
                  349.0, 235.0, 106.0, 274.0, 216.0]

def totalScore(meetingList, month, a , b, c): # last three inputs are the weights
    timeScore, STDTimescore = (TZCalculator(meetingList,month))
    totalFlightHours = (totalDistanceList(meetingList))/(560)/(len(meetingList))

    aveSleepLoss = timeScore + totalFlightHours/2

    STDFlightTimes = standardDevList(meetingList)

    averageTempDiff, averagePrecDiff, closeTo_22, STDTempDiff, STDPrecDiff = climateCalculator(meetingL

    totalCost1 = totalCost(meetingList)

    productivityScore = averageTempDiff/10 + averagePrecDiff/100 + (abs(closeTo_22))*(11/100) + aveSlee

    equalizedScore = STDTempDiff/10 + STDPrecDiff/100 + (abs(closeTo_22))*(11/100) + STDTimescore*(3/4)

    costScore = totalCost1 * 0.002

    totalScores = a*productivityScore + b*equalizedScore + c*costScore

    return totalScores

class TravelTime():
    """A Simple attempt to model the travel time of an airplane flight"""

    def __init__(self, miles,layovers,inefficiency = 1.1):
        """Initialize the trip's attributes"""
        self.miles = miles
        self.inefficiency = inefficiency
        self.layovers = layovers
        self.flight_speed = 560
        self.avg_layover_time = 5
        self.takeoff_time = 0.2
        self.landing_time = 0.32
        self.drive_time = 5
        if (layovers == 0):
            self.flight_time = self.miles / self.flight_speed
```

```
        else:
            self.flight_time = (self.miles*self.inefficiency) / self.flight_speed
        self.layover_time = self.avg_layover_time*self.layovers
        self.extra_time = (self.takeoff_time + self.landing_time)*(self.layovers + 1)
        self.flight_time = self.flight_time + self.extra_time
        self.travel_time = self.layover_time + self.flight_time + self.drive_time

    def get_time(self):
        print("The trip will take " + str(self.travel_time) + " hours.")

    def get_raw_time(self):
        print("The flight time is " + str(self.flight_time) + " hours.")

class Layovers():
    """A simple attempt to model how many layovers a flight path has"""

    def __init__(self, start_city_size, end_city_size):
        self.start_city = start_city_size
        self.end_city = end_city_size


    def get_layovers(self):
        if(self.start_city == "big" and self.end_city == "big"):
            return 1
        elif((self.start_city == "big" and self.end_city == "med") or (self.start_city == "med" and sel
            return 1
        elif((self.start_city == "big" and self.end_city == "small") or (self.start_city == "small" and
            return 2
        elif(self.start_city == "med" and self.end_city == "med"):
            return 2
        elif((self.start_city == "med" and self.end_city == "small") or (self.start_city == "small" and
            return 3
        elif(self.start_city == "small" and self.end_city == "small"):
            return 3


def totalTravelTime(meetingList):
    distances = totalDistanceList(meetingList)
    i=0
    total_times = []
    for distance in distances:
        total_layover_time = 0
        for j in range(0,len(meetingList)):
            start_city_size = 'big'
            my_layovers = Layovers(start_city_size,end_city_sizes[i])
            total_layover_time = (my_layovers.get_layovers() - 1) * 5.52 + total_layover_time
        my_trip = TravelTime(distance,1)
        total_time = my_trip.travel_time + total_layover_time
        total_times.append(total_time)
        i = i+1

    total_times2 = (np.array(total_times))
    return total_times2

def totalCost(meetingList):
    distances = totalDistanceList(meetingList)
    i = 0
    total_flight_costs = []
    for distance in distances:
        for i in range(0,len(meetingList)):
            start_city_size = 'big'
```

```
            my_layovers = Layovers(start_city_size,end_city_sizes[i])
            flight_start_cost= 50*(my_layovers.get_layovers()+1)
        total_flight_costs.append(flight_start_cost +(.168*distance*1.1))
        i = i + 1

    total_costs = []
    i=0
    for cost in hotelCosts:
        total_costs.append(cost * 3 * (len(meetingList)) + 2*total_flight_costs[i])
        i = i + 1

    totalCosts = np.array(total_costs)
    return totalCosts



def TZCalculator(meetingList,month):
    homeList = regionConverter(meetingList) #gets the lat/long of region for each location
    homeList = homeList.astype(int)
    homeList = homeList.tolist()

    for i in range(0,len(homeList)):
        if homeList[i][0] == -9:
            homeList[i][0] = -10

    if (4 <= month) and (month <= 9):
        daylightSave = summerTime

    elif ((10 <= month) and (month <= 12)) or ((1 <= month) and (month <= 3)):
        daylightSave = winterTime

    else:
        print("Error enter number between 1-12")

    regionNumList = []
    for i in range(0,len(homeList)): # obtains list with the regions of the locations
        regionNumList.append((regionList.index(homeList[i])) + 1)

    timeZones = np.zeros(shape=(75,1))
    tzSTD = np.zeros(shape=(75,1))

    for i in range(0,75): #calculate differences in climate
        counter = 0
        counter3 = np.zeros(shape=(len(regionNumList),1))
        for j in range(0,len(regionNumList)):
            num = daylightSave[i] - daylightSave[regionNumList[j] -1]

            if (num >= 12):
                num = abs((num-24)/2)
            elif (num < -12):
                num = num+24
            elif(num < 0):
                num = abs(num)/2
            counter = counter + num
            counter3[j] = num
        timeZones[i] = counter
        tzSTD[i] = np.std(counter3)
        timeZones1 = timeZones/len(regionNumList)

    return timeZones1, tzSTD
```

```
def climateCalculator(meetingList, month):
    homeList = regionConverter(meetingList) #gets the lat/long of region for each location
    homeList = homeList.astype(int)
    homeList = homeList.tolist()

    for i in range(0,len(homeList)):
        if homeList[i][0] == -9:
            homeList[i][0] = -10

    regionNumList = []
    for i in range(0,len(homeList)): # obtains list with the regions of the locations
        regionNumList.append((regionList.index(homeList[i])) + 1)

    tempDiff = np.zeros(shape = (75,1))
    precDiff = np.zeros(shape = (75,1))
    closeTo_22 = np.zeros(shape = (75,1))
    tempDiffSTD = np.zeros(shape = (75,1))
    precDiffSTD = np.zeros(shape = (75,1))

    for i in range(0,75): #calculate differences in climate
        counter1 = 0
        counter2 = 0
        counter3 = np.zeros(shape=(len(regionNumList),1))
        counter4 = np.zeros(shape=(len(regionNumList),1))
        for j in range(0,len(regionNumList)):
            counter1 = counter1 +  (abs((Temp[i][month-1]) - (Temp[(regionNumList[j])-1][month-1])))
            counter2 = counter2 +  (abs(Prec[i][month-1] - Prec[regionNumList[j]-1][month-1]))
            counter3[j] = abs((Temp[i][month-1]) - (Temp[(regionNumList[j])-1][month-1]))
            counter4[j] = abs(Prec[i][month-1] - Prec[regionNumList[j]-1][month-1])

        counter3 = np.std(counter3)
        counter4 = np.std(counter4)
        closeTo_22[i] = Temp[i][month-1] - 22
        tempDiff[i] = counter1
        precDiff[i] = counter2
        tempDiffSTD[i] = counter3
        precDiffSTD[i] = counter4

    averageTempDiff = tempDiff/(len(regionNumList))
    averagePrecDiff = precDiff/(len(regionNumList))

    return  averageTempDiff, averagePrecDiff, closeTo_22, tempDiffSTD, precDiffSTD



def regionConverter(meetingList): #converts the home lat/long coordinates to a region
    meetingList_np = np.array(meetingList) #turns meetingList into a numpy array

    for i in range(0,len(meetingList_np)): #converts the numpy array to within 20
        if (((0 < meetingList[i][0]) and (meetingList[i][0] < 20)) and ((0 < meetingList[i][1]) and (me
            meetingList_np[i][0] = meetingList_np[i][0] + 5 - (meetingList_np[i][0] % 10)

        elif (((20 < meetingList[i][0]) and (meetingList[i][0] < 40)) and ((20 < meetingList[i][1]) and
            meetingList_np[i][0] = meetingList_np[i][0] + 5 - (meetingList_np[i][0] % 10)

        elif (((20 < meetingList[i][0]) and (meetingList[i][0] < 40)) and ((80 < meetingList[i][1]) and
            meetingList_np[i][0] = meetingList_np[i][0] + 5 - (meetingList_np[i][0] % 10)

        elif (((-20 < meetingList[i][0]) and (meetingList[i][0] < 0)) and ((120 < meetingList[i][1]) an
            meetingList_np[i][0] = meetingList_np[i][0] + 5 - (meetingList_np[i][0] % 10)
```

```
        elif meetingList_np[i][0] < 0:
            if (abs(meetingList_np[i][0]) - (abs(meetingList_np[i][0])) % 20) % 20 == 0: # converts to
                meetingList_np[i][0] = -((abs(meetingList_np[i][0])) + 10 - ((abs(meetingList_np[i][0])

            else:
                meetingList_np[i][0] = -(abs(meetingList_np[i][0]) - ((abs(meetingList_np[i][0])) % 20)

        elif meetingList_np[i][0] > 0:
            if (meetingList_np[i][0] - meetingList_np[i][0] % 20) % 20 == 0: # converts to either 10 or
                meetingList_np[i][0] = meetingList_np[i][0] + 10 - (meetingList_np[i][0] % 20)

            else:
                meetingList_np[i][0] = meetingList_np[i][0] - (meetingList_np[i][0] % 20)
        else:
            meetingList[i][0] = meetingList[i][0] + 10

        # now do it for the second item

        if meetingList_np[i][1] < 0:
            if (abs(meetingList_np[i][1]) - (abs(meetingList_np[i][1])) % 20) % 20 == 0: # converts to
                meetingList_np[i][1] = -((abs(meetingList_np[i][1])) + 10 - ((abs(meetingList_np[i][1])

            else:
                meetingList_np[i][1] = -(abs(meetingList_np[i][1]) - ((abs(meetingList_np[i][1])) % 20)

        elif meetingList_np[i][1] > 0:
            if (meetingList_np[i][1] - meetingList_np[i][1] % 20) % 20 == 0: # converts to either 10 or
                meetingList_np[i][1] = meetingList_np[i][1] + 10 - (meetingList_np[i][1] % 20)

            else:
                meetingList_np[i][1] = meetingList_np[i][1] - (meetingList_np[i][1] % 20)
        else:
            meetingList[i][1] = meetingList[i][1] + 10

    return meetingList_np



def totalDistanceList(meetingList): #returns a list of total distances
    sumArray = np.zeros(shape=(75,1)) #this initializes an array that holds the total distances
    for i in range(0,len(regionList)): #loop through every region
        distArray = np.zeros(shape=(len(meetingList),1)) #sets up array for hold distances for each att
        for j in range(0,len(meetingList)): #loop through every attendee
            d = disty(regionList[i] , meetingList[j]) #calculate distance
            distArray[j] = d    #append the distance to distArray

        sumArray[i] = np.sum(distArray)
    return sumArray

def standardDevList(meetingList): #returns a list of standard deviations
    stdArray = np.zeros(shape=(75,1)) #this initializes an array that holds the standard deviations
    for i in range(0,len(regionList)): #loop through every region
        distArray = np.zeros(shape=(len(meetingList),1)) #sets up array to hold distances for each atte
        for j in range(0,len(meetingList)): #loop through every attendee
            d = disty(regionList[i] , meetingList[j]) #calculate distance
            distArray[j] = d    #append the distance to distArray

        stdArray[i] = np.std(distArray/560)

    return stdArray
```

```
def scoreRank(meetingList,month,a,b,c):
    distList_np = totalScore(meetingList,month,a,b,c) #creates np array
    distList = distList_np.tolist() #converts to list

    d = {} #creates dictionary

    for i in range(0,len(distList)): #fills the dictionary with corresponding distance/region
        num = str(distList[i])
        d[num] = i+1

    sortedDistList = sorted(distList) #sorts distList
    #print(sortedDistList)
    #print(d)

    rankedDict = {}

    for i in range(0,len(sortedDistList)): # the rank of each region is contained in the list compartme
        num = str(sortedDistList[i]) #puts a distance into a variable to be used in dict
        rankedDict[str(d[num])] = str(i+1) #each rank is matched up with the corresponding region from

    return rankedDict

def disty(region, home): #this function calculates the distance between 2 lat/long points
    radius = 3958.756
    regionLat = m.radians(region[0])
    homeLat = m.radians(home[0])
    changeLat = m.radians(home[0] - region[0])
    changeLon = m.radians(home[1] - region[1])
    haversine = (((m.sin(changeLat/2))** 2)) + m.cos(regionLat) * m.cos(homeLat) * (((m.sin(changeLon/2
    dist = 2 * radius * (m.asin((m.sqrt(haversine))))
    return dist

def printTotalScore(meetingList,month,a,b,c):
    rankedDict = scoreRank(meetingList,month,a,b,c)
    img = Image.open("map.jpg")
    font = ImageFont.truetype("ADOBEHEITISTD-REGULAR.OTF", 15)
    draw = ImageDraw.Draw(img)
    for i in range(0,75):
        draw.text(mapCoordList[i], str(rankedDict[str(i+1)]), (255,0,0), font=font)
    img.show()
    img.save("score.jpg")


smallMeetingList = [[36.600238,-121.894676],[52.142736,6.196058],[-37.813628,144.963058],
[31.230416,121.473701],[22.396428,114.109497],[55.755826,37.617300]]

bigMeetingList = [[42.360082,-71.058880],[42.360082,-71.058880],[1.352083,103.819836],
[39.904211,116.407395],[22.396428,114.109497],[22.396428,114.109497],[55.755826,37.617300],
[52.090737,5.121420],[52.229676,21.012229],[55.676097,12.568337],[-37.813628,144.963058]]

g20 = [[-34.603684,-58.381559],[-35.280937,149.130009],[-15.794157,-47.882529],[45.421530,-75.697193],
       [39.904211,116.407395],[48.856614,2.352222],[52.520007,13.404954],[28.613939,77.209021],[-6.1744
       [41.902783,12.496366],[35.689487,139.691706],[37.566535,126.977969],[19.432608,-99.133208],[55.7
       [24.713552,46.675296],[-33.924869,18.424055],[39.933363,32.859742],[51.507351,-0.127758],[38.907
       [50.850346,4.351721]]
print(printTotalScore(g20, 7,1,0,0))
#print(regionConverter(g20))
print(TZCalculator(g20, 7))
```

16.5 Computer Code

16.5.1 Distances

Our original Python code that automated distance calculations and utilized the law of haversines to find the total distance from a region to a certain location. A snapshot of the code is shown below.

```
def totalDistanceList(meetingList): #returns a list of total distances
    sumArray = np.zeros(shape=(75,1)) #this initializes an array that holds the total distances
    for i in range(0,len(regionList)): #loop through every region
        distArray = np.zeros(shape=(len(meetingList),1)) #sets up array for hold distances for each att
        for j in range(0,len(meetingList)): #loop through every attendee
            d = disty(regionList[i] , meetingList[j]) #calculate distance
            distArray[j] = d    #append the distance to distArray

        sumArray[i] = np.sum(distArray)
    return sumArray

def scoreRank(meetingList,month):
    distList_np = totalScore(meetingList,month) #creates np array
    distList = distList_np.tolist() #converts to list

    d = {} #creates dictionary

    for i in range(0,len(distList)): #fills the dictionary with corresponding distance/region
        num = str(distList[i])
        d[num] = i+1

    sortedDistList = sorted(distList) #sorts distList
    #print(sortedDistList)
    #print(d)

    rankedDict = {}

    for i in range(0,len(sortedDistList)): # the rank of each region is contained in the list compartme
        num = str(sortedDistList[i]) #puts a distance into a variable to be used in dict
        rankedDict[str(d[num])] = str(i+1) #each rank is matched up with the corresponding region from

    return rankedDict

def disty(region, home): #this function calculates the distance between 2 lat/long points
    radius = 3958.756
    regionLat = m.radians(region[0])
    homeLat = m.radians(home[0])
    changeLat = m.radians(home[0] - region[0])
    changeLon = m.radians(home[1] - region[1])
    haversine = (((m.sin(changeLat/2))** 2)) + m.cos(regionLat) * m.cos(homeLat) * (((m.sin(changeLon/2
    dist = 2 * radius * (m.asin((m.sqrt(haversine))))
    return dist

def printTotalScore(meetingList,month):
    rankedDict = scoreRank(meetingList,month)
    img = Image.open("map.jpg")
    font = ImageFont.truetype("ADOBEHEITISTD-REGULAR.OTF", 15)
    draw = ImageDraw.Draw(img)
    for i in range(0,75):
        draw.text(mapCoordList[i], str(rankedDict[str(i+1)]), (255,0,0), font=font)
    img.show()
    img.save("score.jpg")
    def showMap():
    #rankedDict = scoreRank(meetingList,month)
    img = Image.open("map.jpg")
```

```
    font = ImageFont.truetype("ADOBEHEITISTD-REGULAR.OTF", 15)
    draw = ImageDraw.Draw(img)
    for i in range(0,75):
        draw.text(mapCoordList[i], str(i+1), (255,0,0), font=font)
    img.show()
    img.save("region_map.jpg")


smallMeetingList = [[36.600238,-121.894676],[52.142736,6.196058],[-37.813628,144.963058],
[31.230416,121.473701],[22.396428,114.109497],[55.755826,37.617300]]

bigMeetingList = [[42.360082,-71.058880],[42.360082,-71.058880],[1.352083,103.819836],
[39.904211,116.407395],[22.396428,114.109497],[22.396428,114.109497],[55.755826,37.617300],
[52.090737,5.121420],[52.229676,21.012229],[55.676097,12.568337],[-37.813628,144.963058]]

#print(printDistRanks(smallMeetingList))
#print("middle")

#print(printDistRanks(bigMeetingList))
#print(climateCalculator(smallMeetingList, 6))
print(printTotalScore(smallMeetingList, 6))
#print(printTotalScore(bigMeetingList,1))
```

16.5.2 Time Zones

We created a MATLAB program that calculated the resultant vector given the time zone number from the numbering system described in Subsection 4.1. The program then converted the radian value of the direction of the resultant vector into the corresponding time zone. This time zone would be the most equal time zone in terms of time zones crossed. Next we devised a MATLAB program that calculated the total time zones crossed by all attendees for a given time zone. The program accounted for the lessened repercussions of westward travel when totaling the time zones. The provided codes have the inputs already in the correct places for the respective small and large meetings.

Code for determining the most efficient time zone (small meeting):

```
%Input Home Time Zone%
HoTiZo1 = 16;
%Input Home Time Zone%
HoTiZo2 = 1;
%Input Home Time Zone%
HoTiZo3 = 10;
%Input Home Time Zone%
HoTiZo4 = 8;
%Input Home Time Zone%
HoTiZo5 = 8;
%Input Home Time Zone%
HoTiZo6 = 3;
Htz1 = HoTiZo1 .*15;
Htz2 = HoTiZo2 .*15;
Htz3 = HoTiZo3 .*15;
Htz4 = HoTiZo4 .*15;
Htz5 = HoTiZo5 .*15;
Htz6 = HoTiZo6 .*15;
x = cos(Htz1.*pi./180) + cos(Htz2.*pi./180) + cos(Htz3.*pi./180) + cos(Htz4.*pi./180) + cos(Htz5.*pi./1
y = sin(Htz1.*pi./180) + sin(Htz2.*pi./180) + sin(Htz3.*pi./180) + sin(Htz4.*pi./180) + sin(Htz5.*pi./1
a = y./x;
if x < 0
    b = atan(a).*180./pi + 180
elseif  y > 0
    b = atan(a) .*180./pi
elseif y < 0
    b = 360 + atan(a).*180./pi
end
```

```
if b < 7.5
    TiZo = 0
elseif b > 352.5
    TiZo = 0
elseif  b < 22.5 & b > 7.5
    TiZo = 1
elseif  b < 37.5 & b > 22.5
    TiZo = 2
elseif  b < 52.5 & b > 37.5
    TiZo = 3
elseif  b < 67.5 & b > 52.5
    TiZo = 4
elseif  b < 82.5 & b > 67.5
    TiZo = 5
elseif  b < 97.5 & b > 82.5
    TiZo = 6
elseif  b < 112.5 & b > 97.5
    TiZo = 7
elseif  b < 127.5 & b > 112.5
    TiZo = 8
elseif  b < 142.5 & b > 127.5
    TiZo = 9
elseif  b < 157.5 & b > 142.5
    TiZo = 10
elseif  b < 172.5 & b > 157.5
    TiZo = 11
elseif  b < 197.5 & b > 172.5
    TiZo = 12
elseif  b < 212.5 & b > 197.5
    TiZo = 13
elseif  b < 227.5 & b > 212.5
    TiZo = 14
elseif  b < 242.5 & b > 227.5
    TiZo = 15
elseif  b < 257.5 & b > 242.5
    TiZo = 17
elseif  b < 272.5 & b > 257.5
    TiZo = 18
elseif  b < 297.5 & b > 272.5
    TiZo = 19
elseif  b < 312.5 & b > 297.5
    TiZo = 20
elseif  b < 327.5 & b > 312.5
    TiZo = 21
elseif  b < 342.5 & b > 327.5
    TiZo = 22
elseif  b < 357.5 & b > 342.5
    TiZo = 23
end

%% Total Number of Time Zone Changes for the Small Meeting
Time_Zone = 15;
w1 = HoTiZo1 - Time_Zone;
w2 = HoTiZo2 - Time_Zone;
w3 = HoTiZo3 - Time_Zone;
w4 = HoTiZo4 - Time_Zone;
w5 = HoTiZo5 - Time_Zone;
w6 = HoTiZo6 - Time_Zone;
if abs(w1) < 12
    Rtz1 = w1
elseif abs(w1) > 12 & HoTiZo1 > Time_Zone
```

```
    Rtz1 = Time_Zone+24 -HoTiZo1
elseif abs(w1) > 12 & Time_Zone > HoTiZo1
    Rtz1 = HoTiZo1 + 24 - Time_Zone
end
if abs(w2) < 12
    Rtz2 = w2
elseif abs(w2) > 12 & HoTiZo2 > Time_Zone
    Rtz2 = Time_Zone+24 -HoTiZo2
elseif abs(w2) > 12 & Time_Zone > HoTiZo2
    Rtz2 = HoTiZo2 + 24 - Time_Zone
end
if abs(w3) < 12
    Rtz3 = w3
elseif abs(w3) > 12 & HoTiZo3 > Time_Zone
    Rtz3 = Time_Zone+24 -HoTiZo3
elseif abs(w3) > 12 & Time_Zone > HoTiZo3
    Rtz3 = HoTiZo3 + 24 - Time_Zone
end
if abs(w4) < 12
    Rtz4 = w4
elseif abs(w4) > 12 & HoTiZo4 > Time_Zone
    Rtz4 = Time_Zone+24 -HoTiZo4
elseif abs(w4) > 12 & Time_Zone > HoTiZo4
    Rtz4 = HoTiZo4 + 24 - Time_Zone
end
if abs(w5) < 12
    Rtz5 = w5
elseif abs(w5) > 12 & HoTiZo5 > Time_Zone
    Rtz5 = Time_Zone+24 -HoTiZo5
elseif abs(w5) > 12 & Time_Zone > HoTiZo5
    Rtz5 = HoTiZo5 + 24 - Time_Zone
end
if abs(w6) < 12
    Rtz6 = w6
elseif abs(w6) > 12 & HoTiZo6 > Time_Zone
    Rtz6 = Time_Zone+24 -HoTiZo6
elseif abs(w6) > 12 & Time_Zone > HoTiZo6
    Rtz6 = HoTiZo6 + 24 - Time_Zone
end
if Rtz1 < 0
    Ftz1 = abs(Rtz1)./2
elseif Rtz1 >= 0
    Ftz1 = abs(Rtz1)
end
if Rtz2 < 0
    Ftz2 = abs(Rtz2)./2
elseif Rtz2 >= 0
    Ftz2 = abs(Rtz2)
end
if Rtz3 < 0
    Ftz3 = abs(Rtz3)./2
elseif Rtz3 >= 0
    Ftz3 = abs(Rtz3)
end
if Rtz4 < 0
    Ftz4 = abs(Rtz4)./2
elseif Rtz4 >= 0
    Ftz4 = abs(Rtz4)
end
if Rtz5 < 0
    Ftz5 = abs(Rtz5)./2
```

```
elseif Rtz5 >= 0
    Ftz5 = abs(Rtz5)
end
if Rtz6 < 0
    Ftz6 = abs(Rtz6)./2
elseif Rtz6 >= 0
    Ftz6 = abs(Rtz6)
end
TotalTZChanges = Ftz1 + Ftz2 + Ftz3 + Ftz4 + Ftz5 + Ftz6
```

Code for determining the most efficient time zone (large meeting):

```
%Input Home Time Zone's Degree%
HoTiZo1 = 20
%Input Home Time Zone's Degree%
HoTiZo2 = 20
%Input Home Time Zone's Degree%
HoTiZo3 = 8
%Input Home Time Zone's Degree%
HoTiZo4 = 8
%Input Home Time Zone's Degree%
HoTiZo5 = 8
%Input Home Time Zone's Degree%
HoTiZo6 = 8
%Input Home Time Zone's Degree%
HoTiZo7 = 3
%Input Home Time Zone's Degree%
HoTiZo8 = 2
%Input Home Time Zone's Degree%
HoTiZo9 = 2
%Input Home Time Zone's Degree%
HoTiZo10 = 2
%Input Home Time Zone's Degree%
HoTiZo11 = 10
Htz1 = HoTiZo1.*15
Htz2 = HoTiZo2.*15
Htz3 = HoTiZo3.*15
Htz4 = HoTiZo4.*15
Htz5 = HoTiZo5.*15
Htz6 = HoTiZo6.*15
Htz7 = HoTiZo7.*15
Htz8 = HoTiZo8.*15
Htz9 = HoTiZo9.*15
Htz10 = HoTiZo10.*15
Htz11 = HoTiZo11.*15
x = cos(Htz1.*pi./180) + cos(Htz2.*pi./180) + cos(Htz3.*pi./180) + cos(Htz4.*pi./180) + cos(Htz5.*pi./1
y = sin(Htz1.*pi./180) + sin(Htz2.*pi./180) + sin(Htz3.*pi./180) + sin(Htz4.*pi./180) + sin(Htz5.*pi./1
a = y./x;
if x < 0
    b = atan(a).*180./pi + 180
elseif  y > 0
    b = atan(a) .*180./pi
elseif y < 0
    b = 360 + atan(a).*180./pi
end
if b < 7.5
    TiZo = 0
elseif b > 352.5
    TiZo = 0
elseif  b < 22.5 & b > 7.5
    TiZo = 1
elseif  b < 37.5 & b > 22.5
```

```
    TiZo = 2
elseif  b < 52.5 & b > 37.5
    TiZo = 3
elseif  b < 67.5 & b > 52.5
    TiZo = 4
elseif  b < 82.5 & b > 67.5
    TiZo = 5
elseif  b < 97.5 & b > 82.5
    TiZo = 6
elseif  b < 112.5 & b > 97.5
    TiZo = 7
elseif  b < 127.5 & b > 112.5
    TiZo = 8
elseif  b < 142.5 & b > 127.5
    TiZo = 9
elseif  b < 157.5 & b > 142.5
    TiZo = 10
elseif  b < 172.5 & b > 157.5
    TiZo = 11
elseif  b < 197.5 & b > 172.5
    TiZo = 12
elseif  b < 212.5 & b > 197.5
    TiZo = 13
elseif  b < 227.5 & b > 212.5
    TiZo = 14
elseif  b < 242.5 & b > 227.5
    TiZo = 15
elseif  b < 257.5 & b > 242.5
    TiZo = 17
elseif  b < 272.5 & b > 257.5
    TiZo = 18
elseif  b < 297.5 & b > 272.5
    TiZo = 19
elseif  b < 312.5 & b > 297.5
    TiZo = 20
elseif  b < 327.5 & b > 312.5
    TiZo = 21
elseif  b < 342.5 & b > 327.5
    TiZo = 22
elseif  b < 357.5 & b > 342.5
    TiZo = 23
end
%% Total Time Zone Changes
Time_Zone = 2;
w1 = HoTiZo1 - Time_Zone;
w2 = HoTiZo2 - Time_Zone;
w3 = HoTiZo3 - Time_Zone;
w4 = HoTiZo4 - Time_Zone;
w5 = HoTiZo5 - Time_Zone;
w6 = HoTiZo6 - Time_Zone;
w7 = HoTiZo7 - Time_Zone;
w8 = HoTiZo8 - Time_Zone;
w9 = HoTiZo9 - Time_Zone;
w10 = HoTiZo10 - Time_Zone;
w11 = HoTiZo11 - Time_Zone;
if abs(w1) < 12
    Rtz1 = w1
elseif abs(w1) > 12 & HoTiZo1 > Time_Zone
    Rtz1 = Time_Zone+24 -HoTiZo1
elseif abs(w1) > 12 & Time_Zone > HoTiZo1
    Rtz1 = HoTiZo1 + 24 - Time_Zone
```

```
end
if abs(w2) < 12
    Rtz2 = w2
elseif abs(w2) > 12 & HoTiZo2 > Time_Zone
    Rtz2 = Time_Zone+24 -HoTiZo2
elseif abs(w2) > 12 & Time_Zone > HoTiZo2
    Rtz2 = HoTiZo2 + 24 - Time_Zone
end
if abs(w3) < 12
    Rtz3 = w3
elseif abs(w3) > 12 & HoTiZo3 > Time_Zone
    Rtz3 = Time_Zone+24 -HoTiZo3
elseif abs(w3) > 12 & Time_Zone > HoTiZo3
    Rtz3 = HoTiZo3 + 24 - Time_Zone
end
if abs(w4) < 12
    Rtz4 = w4
elseif abs(w4) > 12 & HoTiZo4 > Time_Zone
    Rtz4 = Time_Zone+24 -HoTiZo4
elseif abs(w4) > 12 & Time_Zone > HoTiZo4
    Rtz4 = HoTiZo4 + 24 - Time_Zone
end
if abs(w5) < 12
    Rtz5 = w5
elseif abs(w5) > 12 & HoTiZo5 > Time_Zone
    Rtz5 = Time_Zone+24 -HoTiZo5
elseif abs(w5) > 12 & Time_Zone > HoTiZo5
    Rtz5 = HoTiZo5 + 24 - Time_Zone
end
if abs(w6) < 12
    Rtz6 = w6
elseif abs(w6) > 12 & HoTiZo6 > Time_Zone
    Rtz6 = Time_Zone+24 -HoTiZo6
elseif abs(w6) > 12 & Time_Zone > HoTiZo6
    Rtz6 = HoTiZo6 + 24 - Time_Zone
end
if abs(w7) < 12
    Rtz7 = w7
elseif abs(w7) > 12 & HoTiZo7 > Time_Zone
    Rtz7 = Time_Zone+24 -HoTiZo7
elseif abs(w7) > 12 & Time_Zone > HoTiZo7
    Rtz7 = HoTiZo7 + 24 - Time_Zone
end
if abs(w8) < 12
    Rtz8 = w8
elseif abs(w8) > 12 & HoTiZo8 > Time_Zone
    Rtz8 = Time_Zone+24 -HoTiZo8
elseif abs(w8) > 12 & Time_Zone > HoTiZo8
    Rtz8 = HoTiZo8 + 24 - Time_Zone
end
if abs(w9) < 12
    Rtz9 = w9
elseif abs(w9) > 12 & HoTiZo9 > Time_Zone
    Rtz1 = Time_Zone+24 -HoTiZo9
elseif abs(w9) > 12 & Time_Zone > HoTiZo9
    Rtz9 = HoTiZo9 + 24 - Time_Zone
end
if abs(w10) < 12
    Rtz10 = w10
elseif abs(w10) > 12 & HoTiZo10 > Time_Zone
    Rtz10 = Time_Zone+24 -HoTiZo10
```

```
elseif abs(w10) > 12 & Time_Zone > HoTiZo10
    Rtz1 = HoTiZo10 + 24 - Time_Zone
end
if abs(w11) < 12
    Rtz11 = w11
elseif abs(w11) > 12 & HoTiZo11 > Time_Zone
    Rtz11 = Time_Zone+24 -HoTiZo11
elseif abs(w11) > 12 & Time_Zone > HoTiZo11
    Rtz1 = HoTiZo11 + 24 - Time_Zone
end
if Rtz1 < 0
    Ftz1 = abs(Rtz1)./2
elseif Rtz1 >= 0
    Ftz1 = abs(Rtz1)
end
if Rtz2 < 0
    Ftz2 = abs(Rtz2)./2
elseif Rtz2 >= 0
    Ftz2 = abs(Rtz2)
end
if Rtz3 < 0
    Ftz3 = abs(Rtz3)./2
elseif Rtz3 >= 0
    Ftz3 = abs(Rtz3)
end
if Rtz4 < 0
    Ftz4 = abs(Rtz4)./2
elseif Rtz4 >= 0
    Ftz4 = abs(Rtz4)
end
if Rtz5 < 0
    Ftz5 = abs(Rtz5)./2
elseif Rtz5 >= 0
    Ftz5 = abs(Rtz5)
end
if Rtz6 < 0
    Ftz6 = abs(Rtz6)./2
elseif Rtz6 >= 0
    Ftz6 = abs(Rtz6)
end
if Rtz7 < 0
    Ftz7 = abs(Rtz7)./2
elseif Rtz7 >= 0
    Ftz7 = abs(Rtz7)
end
if Rtz8 < 0
    Ftz8 = abs(Rtz8)./2
elseif Rtz8 >= 0
    Ftz8 = abs(Rtz8)
end
if Rtz9 < 0
    Ftz9 = abs(Rtz9)./2
elseif Rtz9 >= 0
    Ftz9 = abs(Rtz9)
end
if Rtz10 < 0
    Ftz10 = abs(Rtz10)./2
elseif Rtz10 >= 0
    Ftz10 = abs(Rtz10)
end
if Rtz11 < 0
```

```
    Ftz11 = abs(Rtz11)./2
elseif Rtz11 >= 0
    Ftz11 = abs(Rtz11)
end
TotalTZChanges = Ftz1 + Ftz2 + Ftz3 + Ftz4 + Ftz5 + Ftz6 + Ftz7 + Ftz8 + Ftz9 + Ftz10 + Ftz11
```

16.5.3 Travel Time Program

Using Python, we created a function in which we would plug the distance between two locations and the number of layovers into the equation described in the Travel Time section. The program then gave us the time of the trip in hours. Next, we created a function from figure 4 that would be given the size of the start and end cities and would spit out the number of layovers involved in the plane trip. A list with the sizes of the attendees' cities and a list of the sizes of all of the possible end cities was put into the function, and it spit out a list with the number of layovers it took to get between each pair of cities. This list and a list of all the distances from each starting city to each zone was plugged into the original function to give us a list containing the total hours spent in a flight trip to each region. We then plotted this list on a graph for easy viewing so we could analyze the results.

```
class TravelTime():
    """A Simple attempt to model the travel time of an airplane flight"""

    def __init__(self, miles,layovers,inefficiency = 1.1):
        """Initialize the trip's attributes"""
        self.miles = miles
        self.inefficiency = inefficiency
        self.layovers = layovers
        self.flight_speed = 560
        self.avg_layover_time = 5
        self.takeoff_time = 0.2
        self.landing_time = 0.32
        self.drive_time = 5
        if (layovers == 0):
            self.flight_time = self.miles / self.flight_speed
        else:
            self.flight_time = (self.miles*self.inefficiency) / self.flight_speed
        self.layover_time = self.avg_layover_time*self.layovers
        self.extra_time = (self.takeoff_time + self.landing_time)*(self.layovers + 1)
        self.flight_time = self.flight_time + self.extra_time
        self.travel_time = self.layover_time + self.flight_time + self.drive_time

    def get_time(self):
        print("The trip will take " + str(self.travel_time) + " hours.")

    def get_raw_time(self):
        print("The flight time is " + str(self.flight_time) + " hours.")

class Layovers():
    """A simple attempt to model how many layovers a flight path has"""

    def __init__(self, start_city_size, end_city_size):
        self.start_city = start_city_size
        self.end_city = end_city_size


    def get_layovers(self):
        if(self.start_city == "big" and self.end_city == "big"):
            return 1
        elif((self.start_city == "big" and self.end_city == "med") or (self.start_city == "med" and sel
            return 1
        elif((self.start_city == "big" and self.end_city == "small") or (self.start_city == "small" and
```

```
            return 2
        elif(self.start_city == "med" and self.end_city == "med"):
            return 2
        elif((self.start_city == "med" and self.end_city == "small") or (self.start_city == "small" and
            return 3
        elif(self.start_city == "small" and self.end_city == "small"):
            return 3


def totalTravelTime(meetingList):
    distances = totalDistanceList(meetingList)
    i=0
    total_times = []
    for distance in distances:
        total_layover_time = 0
        for j in range(0,len(meetingList)):
            start_city_size = 'big'
            my_layovers = Layovers(start_city_size,end_city_sizes[i])
            total_layover_time = (my_layovers.get_layovers() - 1) * 5.52 + total_layover_time
        my_trip = TravelTime(distance,1)
        total_time = my_trip.travel_time + total_layover_time
        total_times.append(total_time)
        i = i+1

    total_times2 = (np.array(total_times))
    return total_times2
```

16.5.4 Work Productivity Algorithm

Using MATLAB, we created horizontal matrices with monthly average temperatures and precipitation levels for each of the ten climates. Then we augmented these climate matrices in the order of the 75 zones' climates. We defined a variable for the month when the meeting is held, and then ran a for loop to evaluate the average difference between each zone's temperature and precipitation and those of the various zones where the participants hail from. We also took the absolute difference between each zones' temperature and 22 degrees Celsius.

Similarly, we created a matrix for the average time zone of each region (Using the International Date Line as 1). Then, we set up a for loop that evaluated the difference in time zones between each region and the zone where the participants are from. For values above 12 or below 12, we subtracted or added 24 to find the number of time zones crossed traveling the opposite (and shorter) way. Positive values indicated eastward travel and were multiplied by the sleep-lost-from-eastward-travel ratio (1). Negative values were absolute-valued and then multiplied by the sleep-lost-from-westward-travel ratio ($\frac{1}{2}$). The sum of these for all 6 or 11 or more participants' zones is that zones TimeScore, which are all augmented into a matrix.

Total flight hours are taken from the previous program and stored in a matrix which is multiplied by the sleep-lost-due-to-flying ratio ($\frac{1}{2}$).

Work percent loss for each zone was then determined as the sum of temperature difference times its ratio ($\frac{1}{10}$), precipitation difference in mm times a ratio ($\frac{1}{100}$), how close its temperature was to 22 degrees Celsius times a ratio ($\frac{11}{100}$), and how much sleep lost multiplied by its ratio ($\frac{3}{4}$). Then the program spit out a matrix that had all 75 zones and their respective Work Percent Loss values.

Code for determining optimal productive (set for the big meeting):

```
clc;clear;clf;

TWT = [27 27 27 27 26 26 25 27 27 27 27 27] %TropWetTemp
TWP = [250 280 260 320 300 200 170 160 190 200 240 220]

TDNT = [21 20 21 22 23 24 26 27 27 27 26 25] %TropWetDryNorthTemp
TDNP = [10 10 10 30 100 200 300 550 470 200 105 15]

TDST = [23 23 23 23 23 20 20 23 25 24 23 23] %TropWetDrySouthTemp
MT = [7 8 10 13 17 21 24 24 21 16 11 8] %Mediteranean Temp
MP = [104 99 69 66 38 23 13 23 59 84 120 112]
```

```
HSNT = [7 8 11 17 21 26 28 27 24 17 11 7] %HumiSubNorthTemp
HSNP = [150 120 130 120 100 90 80 70 60 70 100 120]

HSST = [28 27 24 17 11 7 7 8 11 17 21 26]
TDSP = [225 220 230 110 10 0 0 5 25 90 185 230]

HSSP = [70 60 70 70 100 120 150 120 130 120 100 80]

MNT = [2 4 6 8 12 13 17 16 14 10 6 4] %MariWestCoasNorthTemp
MNP = [140 120 100 60 40 50 20 30 60 120 140 150]

MST = [17 16 14 10 6 4 2 4 6 8 12 13]
MSP = [20 30 60 120 140 150 140 120 100 60 40 50]

HCT = [-7 -5 3 10 15 20 23 22 18 12 5 -3] %HumiContTemp
HCP = [50 40 70 90 90 100 90 90 70 60 60 50]

ST = [-43 -39 -25 -5 5 12 15 11 5 -7 -30 -40] %SubarcticTemp
SP = [15 10 10 15 25 30 40 35 20 15 15 10]

ANT = [13 14 17 20 23 26 27 27 25 23 21 15] %Arid North Temp
ANP = [3 2 1 1 0 0 0 0 0 0 2 3]

AST = [27 27 25 23 21 15 13 14 17 20 23 26]
ASP = [0 0 0 0 2 3 3 2 1 1 0 0]

WSNT = [14 17 20 24 26 27 28 28 25 22 18 16] %WarmSemiNorthTemp
WSNP = [19 18 19 32 55 71 50 85 145 69 22 15]

CSNT = [-7 -3 1 5 10 15 19 18 13 7 0 -5] %Cold Semi North Temp
CSNP = [15 10 18 22 44 46 31 30 29 15 14 15]

WSST = [28 28 25 22 18 16 14 17 20 24 26 27] %WarmSemiNorthTemp
WSSP = [50 85 145 69 22 15 19 18 19 32 55 71]

CSST = [19 18 13 7 0 -5 -7 -3 1 5 10 15] %ColdSemiNorthTemp
CSSP = [31 30 29 15 14 15 15 10 18 22 44 46]


Temp = [ST; ST; ST; ST; ST; ST; MNT; ST; CSNT; WSNT; TDNT; HCT; HSNT; TWT; HCT; MNT; TWT; TWT; WSST; CS
Temp(44,:)

Prec = [SP; SP; SP; SP; SP; SP; MNP; SP; CSNP; WSNP; TDNP; HCP; HSNP; TWP; HCP; MNP;
TWP; TWP; WSSP; CSSP; TWP; TDSP; HSSP; SP; MNP; MNP; ANP; TDNP; MNP; MNP ;ANP; ANP; TDNP; TDSP; WSSP ;H
Prec(30,:)


month = 1;
for i = 1:1:75
TempDiff(i) = 2*abs(Temp(i,month) - MNT(month))+abs(Temp(i,month) - WSST(month))+2*abs(Temp(i,month) -
PrecDiff(i) = 2*abs(Prec(i,month) - MNP(month))+abs(Prec(i,month) - WSSP(month))+2*abs(Prec(i,month) -
CloseTo22(i) = Temp(i,month)-22;
end

AverageTempDiff = TempDiff/11
AveragePrecDiff = PrecDiff/11

Time = [2 19 2 14 3 3 3 4 4 3 5 5 6 5 6 6 4 5 6 7 6 7 7 10 11 11 11 9 12 12 11 11 11 11 12 13 12 13 12


for i = 1:1:75
```

```
N = Time(i) - Time(15);
if (N >= 12)
N = abs(N-24)/2;
elseif (N < -12)
N = N+24;
elseif(N < 0)
N = abs(N)/2;
end

N1 = Time(i) - Time(59);
if (N1 >= 12)
N1 = abs(N1-24)/2;
elseif (N1 < -12)
N1 = N1+24;
elseif(N1 < 0)
N1 = abs(N1)/2;
end

N2 = Time(i) - Time(57);
if (N2 >= 12)
N2 = abs(N2-24)/2;
elseif (N2 < -12)
N2 = N2+24;
elseif(N2 < 0)
N2 = abs(N2)/2;
end


N3 = Time(i) - Time(58);
if (N3 >= 12)
N3 = (N3-24)/2;
elseif (N3 < -12)
N3 = N3+24;
elseif(N3 < 0)
N3 = abs(N3)/2;
end

N4 = Time(i) - Time(37);
if (N4 >= 12)
N4 = abs(N4-24)/2;
elseif (N4 < -12)
N4 = N4+24;
elseif(N4 < 0)
N4 = abs(N4)/2;
end


N6 = Time(i) - Time(71);
if (N6 >= 12)
N6 = abs(N6-24)/2;
elseif (N6 < -12)
N6 = N6+24;
elseif(N6 < 0)
N6 = abs(N6)/2;
end

N5 = Time(i) - Time(30);
if (N5 >= 12)
N5 = abs(N5-24)/2;
elseif (N5 < -12)
N5 = N5+24;
```

```
elseif(N5 < 0)
N5 = abs(N5)/2;
end
Timescore(i) = 2*N+N1+N2+2*N3+2*N4+2*N5+N6;
end

TotalFlightHours=  [87.03053512,       77.21702092,       89.3114396 ,       71.33513886,       90.5766
AverageFlightHours = TotalFlightHours/11;
AveSleepLoss = Timescore/11 + AverageFlightHours/2;

WorkPercentLoss = AverageTempDiff/10 + AveragePrecDiff/100 + abs(CloseTo22)*11/100 + AveSleepLoss*6/8
```

16.5.5 Sensitivity Analysis

To test the weight of the different variables, we treated the individual constants as variables in order to see how a small change would skew the location decision. First the program would define a constant as a matrix of 3 to 5 values. It would find the Work Percent Losses as a matrix of values for all 75 zones over the different constants. Then it plotted a surface plot of the 75 zones which showed how their scores fluctuated as the constant changed. This was completed for every constant. Once the locations resulting in minimum Work Percent Loss were determined, it plotted those as overlaying line plots for ease of viewing.

Code for analyzing the sensitivity of the difference in degrees Celsius constant for the big meeting:

```
clc;clear;clf;

t = linspace(1,75,75);
r = 0.5; %sleeploss due to flying standard is 2 hours flying, 1 hour sleeploss i.e. 1/2
k = 2; %;sleeploss due to time change west standard is 2
h = 1; %sleeploss due to time change east standard is 1
w = [0,1/10,3/5]; %workloss due to temp
p = 1/100; %workloss due to prec
c = 11/100; %workloss due to diff from 22 degrees


TWT = [27 27 27 27 26 26 25 27 27 27 27 27]; %TropWetTemp
TWP = [250 280 260 320 300 200 170 160 190 200 240 220];

TDNT = [21 20 21 22 23 24 26 27 27 27 26 25]; %TropWetDryNorthTemp
TDNP = [10 10 10 30 100 200 300 550 470 200 105 15];

TDST = [23 23 23 23 23 20 20 23 25 24 23 23]; %TropWetDrySouthTemp
MT = [7 8 10 13 17 21 24 24 21 16 11 8]; %Mediteranean Temp
MP = [104 99 69 66 38 23 13 23 59 84 120 112];

HSNT = [7 8 11 17 21 26 28 27 24 17 11 7]; %HumiSubNorthTemp
HSNP = [150 120 130 120 100 90 80 70 60 70 100 120];

HSST = [28 27 24 17 11 7 7 8 11 17 21 26];
TDSP = [225 220 230 110 10 0 0 5 25 90 185 230];

HSSP = [70 60 70 70 100 120 150 120 130 120 100 80];

MNT = [2 4 6 8 12 13 17 16 14 10 6 4]; %MariWestCoasNorthTemp
MNP = [140 120 100 60 40 50 20 30 60 120 140 150];

MST = [17 16 14 10 6 4 2 4 6 8 12 13];
MSP = [20 30 60 120 140 150 140 120 100 60 40 50];

HCT = [-7 -5 3 10 15 20 23 22 18 12 5 -3]; %HumiContTemp
HCP = [50 40 70 90 90 100 90 90 70 60 60 50];

ST = [-43 -39 -25 -5 5 12 15 11 5 -7 -30 -40]; %SubarcticTemp
SP = [15 10 10 15 25 30 40 35 20 15 15 10];
```

```
ANT = [13 14 17 20 23 26 27 27 25 23 21 15]; %Arid North Temp
ANP = [3 2 1 1 0 0 0 0 0 0 2 3];

AST = [27 27 25 23 21 15 13 14 17 20 23 26];
ASP = [0 0 0 0 2 3 3 2 1 1 0 0];

WSNT = [14 17 20 24 26 27 28 28 25 22 18 16]; %WarmSemiNorthTemp
WSNP = [19 18 19 32 55 71 50 85 145 69 22 15];

CSNT = [-7 -3 1 5 10 15 19 18 13 7 0 -5]; %Cold Semi North Temp
CSNP = [15 10 18 22 44 46 31 30 29 15 14 15];

WSST = [28 28 25 22 18 16 14 17 20 24 26 27]; %WarmSemiNorthTemp
WSSP = [50 85 145 69 22 15 19 18 19 32 55 71];

CSST = [19 18 13 7 0 -5 -7 -3 1 5 10 15] ;%ColdSemiNorthTemp
CSSP = [31 30 29 15 14 15 15 10 18 22 44 46];


Temp = [ST; ST; ST; ST; ST; ST; MNT; ST; CSNT; WSNT; TDNT; HCT; HSNT; TWT; HCT; MNT; TWT; TWT; WSST; CS
Temp(44,:);

Prec = [SP; SP; SP; SP; SP; SP; MNP; SP; CSNP; WSNP; TDNP; HCP; HSNP; TWP; HCP; MNP; TWP; TWP; WSSP; CS


month = 1;
for i = 1:1:75
TempDiff(i) = 2*abs(Temp(i,month) - MNT(month))+abs(Temp(i,month) - WSST(month))+2*abs(Temp(i,month) -
PrecDiff(i) = 2*abs(Prec(i,month) - MNP(month))+abs(Prec(i,month) - WSSP(month))+2*abs(Prec(i,month) -
CloseTo22(i) = Temp(i,month)-22;
end

for i=1:1:size(w,2)
AverageTempDiff(i,:) = TempDiff/11*w(i);
AveragePrecDiff(i,:) = PrecDiff/11*p;
CloseTo22Matrix(i,:) = c*abs(CloseTo22);
end

Time = [2 19 2 14 3 3 3 4 4 3 5 5 6 5 6 6 4 5 6 7 6 7 7 10 11 11 11 9 12 12 11 11 11 11 12 13 12 13 12

for i = 1:1:75
N = Time(i) - Time(15);
if (N >= 12)
N = abs(N-24)/k;
elseif (N < -12)
N = N+24;
elseif(N < 0)
N = abs(N)/k;
end

N1 = Time(i) - Time(59);
if (N1 >= 12)
N1 = abs(N1-24)/k;
elseif (N1 < -12)
N1 = N1+24;
elseif(N1 < 0)
N1 = abs(N1)/k;
end

N2 = Time(i) - Time(57);
```

```
if (N2 >= 12)
N2 = abs(N2-24)/k;
elseif (N2 < -12)
N2 = N2+24;
elseif(N2 < 0)
N2 = abs(N2)/k;
end


N3 = Time(i) - Time(58);
if (N3 >= 12)
N3 = (N3-24)/k;
elseif (N3 < -12)
N3 = N3+24;
elseif(N3 < 0)
N3 = abs(N3)/k;
end

N4 = Time(i) - Time(37)
if (N4 >= 12)
N4 = abs(N4-24)/k;
elseif (N4 < -12)
N4 = N4+24;
elseif(N4 < 0)
N4 = abs(N4)/k;
end


N6 = Time(i) - Time(71);
if (N6 >= 12)
N6 = abs(N6-24)/k;
elseif (N6 < -12)
N6 = N6+24;
elseif(N6 < 0)
N6 = abs(N6)/k;
end

N5 = Time(i) - Time(30);
if (N5 >= 12)
N5 = abs(N5-24)/k;
elseif (N5 < -12)
N5 = N5+24;
elseif(N5 < 0)
N5 = abs(N5)/k;
end

Timescore(i) = 2*N+N1+N2+2*N3+2*N4+2*N5+N6;

end

TotalFlightHours=  [87.03053512,       77.21702092,       89.3114396 ,       71.33513886,       90.5766

AverageFlightHours = TotalFlightHours/11;

for i = 1:1:size(w,2)
AveSleepLoss(i,:) = Timescore/11 + AverageFlightHours*r;
end

WorkPercentLoss = AverageTempDiff + AveragePrecDiff + CloseTo22Matrix + AveSleepLoss*6/8

clf;
```

```
figure(1)
surf(t,w,WorkPercentLoss)
hold on
plot3(59,0,6.1660,'*g')
plot3(37,1/10,7.4756,'*r')
plot3(29,3/5,13.1819,'*y')
xlabel('Location by Zone Number')
ylabel('Percent of Work Loss per Degree Celsius Difference from 22')
zlabel('Total Percent of Work Loss (including other factors)')
title('How Changing the Constant for Temperature Affects Location Choice')

figure(2)
AverageTempDiff(:,29)
AverageTempDiff(:,37)
plot(w,WorkPercentLoss(:,37),'r--')
hold on
plot(w,WorkPercentLoss(:,59),'b')
plot(w,WorkPercentLoss(:,29),'g')
legend('Istanbul','Singapore','Stockholm')
xlabel('Percent of Work Loss per Degree Celsius Difference from Hometown')
ylabel('Total Percent of Work Loss (including other factors)')
title('How Changing the Constant for Temperature Affects Location Choice')
```

16.5.6 Equality Algorithm

Essentially, this algorithm took the standard deviations for each factor and then weighted them using the original formula. First it created a matrix of the differences in temperature and precipitation with each participant as a column and each destination zone as a row. Then it simply took the standard deviation. Similar procedures were used to evaluate the sleep lost from time zones and flight times. Finally, it determined an Equaling Score for each zone by adding up the standard deviations from each factor using the previously mentioned weighting algorithm.

To determine how favoring equality impacts location decision, we summed the Equaling Score and the previously determined Work Percent Loss using a variable as the weight constant for Equaling score (from 0 to 1). Then we plotted how the 75 zones' total scores changed as the weighting constant increased and determined the optimal location for various weights.

Code used to find the standard deviations of each factor for each zone for the small meeting:

```
clc;clear;clf;

TWT = [27 27 27 27 26 26 25 27 27 27 27 27] %TropWetTemp
TWP = [250 280 260 320 300 200 170 160 190 200 240 220]

TDNT = [21 20 21 22 23 24 26 27 27 27 26 25] %TropWetDryNorthTemp
TDNP = [10 10 10 30 100 200 300 550 470 200 105 15]

TDST = [23 23 23 23 23 20 20 23 25 24 23 23] %TropWetDrySouthTemp
MT = [7 8 10 13 17 21 24 24 21 16 11 8] %Mediteranean Temp
MP = [104 99 69 66 38 23 13 23 59 84 120 112]

HSNT = [7 8 11 17 21 26 28 27 24 17 11 7] %HumiSubNorthTemp
HSNP = [150 120 130 120 100 90 80 70 60 70 100 120]

HSST = [28 27 24 17 11 7 7 8 11 17 21 26]
TDSP = [225 220 230 110 10 0 0 5 25 90 185 230]

HSSP = [70 60 70 70 100 120 150 120 130 120 100 80]

MNT = [2 4 6 8 12 13 17 16 14 10 6 4] %MariWestCoasNorthTemp
MNP = [140 120 100 60 40 50 20 30 60 120 140 150]

MST = [17 16 14 10 6 4 2 4 6 8 12 13]
MSP = [20 30 60 120 140 150 140 120 100 60 40 50]
```

```
HCT = [-7 -5 3 10 15 20 23 22 18 12 5 -3] %HumiContTemp
HCP = [50 40 70 90 90 100 90 90 70 60 60 50]

ST = [-43 -39 -25 -5 5 12 15 11 5 -7 -30 -40] %SubarcticTemp
SP = [15 10 10 15 25 30 40 35 20 15 15 10]

ANT = [13 14 17 20 23 26 27 27 25 23 21 15] %Arid North Temp
ANP = [3 2 1 1 0 0 0 0 0 0 2 3]

AST = [27 27 25 23 21 15 13 14 17 20 23 26]
ASP = [0 0 0 0 2 3 3 2 1 1 0 0]

WSNT = [14 17 20 24 26 27 28 28 25 22 18 16] %WarmSemiNorthTemp
WSNP = [19 18 19 32 55 71 50 85 145 69 22 15]

CSNT = [-7 -3 1 5 10 15 19 18 13 7 0 -5] %Cold Semi North Temp
CSNP = [15 10 18 22 44 46 31 30 29 15 14 15]

WSST = [28 28 25 22 18 16 14 17 20 24 26 27] %WarmSemiNorthTemp
WSSP = [50 85 145 69 22 15 19 18 19 32 55 71]

CSST = [19 18 13 7 0 -5 -7 -3 1 5 10 15] %ColdSemiNorthTemp
CSSP = [31 30 29 15 14 15 15 10 18 22 44 46]

Temp = [ST; ST; ST; ST; ST; ST; MNT; ST; CSNT; WSNT; TDNT; HCT; HSNT; TWT; HCT; MNT; TWT; TWT; WSST; CS

Prec = [SP; SP; SP; SP; SP; SP; MNP; SP; CSNP; WSNP; TDNP; HCP; HSNP; TWP; HCP; MNP; TWP; TWP; WSSP; CS

month = 6;
for i = 1:1:75
TempDiff(i) = 2*abs(Temp(i,month) - MNT(month))+abs(Temp(i,month) - WSST(month))+2*abs(Temp(i,month) -
PrecDiff(i) =  2*abs(Prec(i,month) - MNP(month))+abs(Prec(i,month) - WSSP(month))+2*abs(Prec(i,month) -
CloseTo22(i) = Temp(i,month)-22;
end

AverageTempDiff = TempDiff/6
AveragePrecDiff = PrecDiff/6
CloseTo22(30)

Time = [2 19 2 14 3 3 3 4 4 3 5 5 6 5 6 6 4 5 6 7 6 7 7 10 11 11 11 9 12 12 11 11 11 11 12 13 12 13 12

for i = 1:1:75
N = Time(i) - Time(7);
if (N >= 12)
N = abs(N-24)/2;
elseif (N < -12)
N = N+24;
elseif(N < 0)
N = abs(N)/2;
end

N1 = Time(i) - Time(30);
if (N1 >= 12)
N1 = abs(N1-24)/2;
elseif (N1 < -12)
N1 = N1+24;
elseif(N1 < 0)
N1 = abs(N1)/2;
end
```

```
N2 = Time(i) - Time(71);
if (N2 >= 12)
N2 = abs(N2-24)/2;
elseif (N2 < -12)
N2 = N2+24;
elseif(N2 < 0)
N2 = abs(N2)/2;
end


N3 = Time(i) - Time(63);
if (N3 >= 12)
N3 = (N3-24)/2;
elseif (N3 < -12)
N3 = N3+24;
elseif(N3 < 0)
N3 = abs(N3)/2;
end

N4 = Time(i) - Time(58)
if (N4 >= 12)
N4 = abs(N4-24)/2;
elseif (N4 < -12)
N4 = N4+24;
elseif(N4 < 0)
N4 = abs(N4)/2
end


N5 = Time(i) - Time(37);
if (N5 >= 12)
N5 = abs(N5-24)/2;
elseif (N5 < -12)
N5 = N5+24;
elseif(N5 < 0)
N5 = abs(N5)/2;
end


Timescore(i) = N+N1+N2+N3+N4+N5;
Timescore1(i,:) = [N,N1,N2,N3,N4,N5];

end

TotalFlightHours = [47.23842518, 43.019245, 48.89179119, 43.0731149, 50.46907321, 54.30926941, 59.77897

AveSleepLoss = Timescore/6+TotalFlightHours/2/6;

WorkPercentLoss = AverageTempDiff/10 + AveragePrecDiff/100 + abs(CloseTo22)*11/100 + AveSleepLoss*6/8;
AverageTempDiff(52);
AveragePrecDiff(52);
CloseTo22(52);
AveSleepLoss(52);

for i = 1:1:75
TempDiff1(i,:) = [abs(Temp(i,month) - MNT(month)), abs(Temp(i,month) - MNT(month)), abs(Temp(i,month) -
PrecDiff1(i,:) = [abs(Prec(i,month) - MNP(month)),abs(Prec(i,month) - MNP(month)) , abs(Prec(i,month) -
CloseTo221(i) = Temp(i,month)-22;
end

STDTempDiff = std(TempDiff1')
```

```
STDPrecDiff = std(PrecDiff1')
CloseTo22(30);

STDTimescore = std(Timescore1')

STDFlightTimes =  [1577.75292566,      1638.88284174,      1739.37722205,      2234.94880754,      1944

EqualingScore = STDTempDiff/10 + STDPrecDiff/100 + abs(CloseTo22)*11/100 + STDTimescore*6/8+STDFlightTi

WorkPercentLoss

clf
figure(1)
t = linspace(1,75,75);
k = [0,1/10,100];
size(k,2)
for i = 1:1:size(k,2)
TotalScore(i,:) = EqualingScore*k(i)+WorkPercentLoss
end
surf(t,k,TotalScore)
axis([1,72,0,1,0,40])
xlabel('Location Number')
ylabel('How much do we care about equalizing the meeting (STD sum)')
zlabel('Score of Location (Lower is better)')
title('How the Location changes depending upon the weight of Equalization')


figure(2)
plot(k,TotalScore(:,52),'r--')
hold on
plot(k,TotalScore(:,62),'b')
plot(k,TotalScore(:,68),'g')

plot(k,TotalScore(:,50),'c')
legend('Novosibirsk','Vladivostock', 'Sapporo')
xlabel('How much do we care about equalizing the meeting (STD sum)')
ylabel('Score of Location (Lower is better)')
title('How the Location changes depending upon the weight of Equalization')
```

16.5.7 Determining Cost for Each Zone

Using Python, we created a function that would plug the number of layovers and the distance between two cities into the equation:

$$Cost = StartingCost * Flights + Costpermile * (miles * inefficiency) \quad (1)$$

With inefficiency as 1.1, cost per mile as 0.168, and starting cost as 50. The list of flight costs was then added to a list containing the hotel costs of staying in each location, creating a list with the total cost of choosing each location. Also included in the cost calculations for each zone were average hotel prices in the major cities of each zone. The average hotel prices for each zone, found using Source 16, can be referenced in Table 16.3.2.

```
def totalCost(meetingList):
    distances = totalDistanceList(meetingList)
    i = 0
    total_flight_costs = []
    for distance in distances:
        for i in range(0,len(meetingList)):
            start_city_size = 'big'
            my_layovers = Layovers(start_city_size,end_city_sizes[i])
            flight_start_cost= 50*(my_layovers.get_layovers()+1)
        total_flight_costs.append(flight_start_cost +(.168*distance*1.1))
```

```
        i = i + 1

    total_costs = []
    i=0
    for cost in hotelCosts:
        total_costs.append(cost * 3 * (len(meetingList)) + 2*total_flight_costs[i])
        i = i + 1

    totalCosts = np.array(total_costs)
    return totalCosts
```

16.5.8 Balancing Productivity, Equality, and Cost

Using a very similar method to the Equality Balancing code, we took the costs and multiplied them by a second weighting constant and added that to the previous sum of the Equaling Score times a variable and Work Productivity. Using surface plots, the program graphed the scores for all 75 zones as the two weighting constants increased.

Code for evaluating Productivity, Equality, and Cost simultaneously for the big meeting:

```
clc;clear all;clf;

TWT = [27 27 27 27 26 26 25 27 27 27 27 27] %TropWetTemp
TWP = [250 280 260 320 300 200 170 160 190 200 240 220]
TDNT = [21 20 21 22 23 24 26 27 27 27 26 25] %TropWetDryNorthTemp
TDNP = [10 10 10 30 100 200 300 550 470 200 105 15]
TDST = [23 23 23 23 23 20 20 23 25 24 23 23] %TropWetDrySouthTemp
MT = [7 8 10 13 17 21 24 24 21 16 11 8] %Mediteranean Temp
MP = [104 99 69 66 38 23 13 23 59 84 120 112]
HSNT = [7 8 11 17 21 26 28 27 24 17 11 7] %HumiSubNorthTemp
HSNP = [150 120 130 120 100 90 80 70 60 70 100 120]
HSST = [28 27 24 17 11 7 7 8 11 17 21 26]
TDSP = [225 220 230 110 10 0 0 5 25 90 185 230]
HSSP = [70 60 70 70 100 120 150 120 130 120 100 80]
MNT = [2 4 6 8 12 13 17 16 14 10 6 4] %MariWestCoasNorthTemp
MNP = [140 120 100 60 40 50 20 30 60 120 140 150]
MST = [17 16 14 10 6 4 2 4 6 8 12 13]
MSP = [20 30 60 120 140 150 140 120 100 60 40 50]
HCT = [-7 -5 3 10 15 20 23 22 18 12 5 -3] %HumiContTemp
HCP = [50 40 70 90 90 100 90 90 70 60 60 50]
ST = [-43 -39 -25 -5 5 12 15 11 5 -7 -30 -40] %SubarcticTemp
SP = [15 10 10 15 25 30 40 35 20 15 15 10]
ANT = [13 14 17 20 23 26 27 27 25 23 21 15] %Arid North Temp
ANP = [3 2 1 1 0 0 0 0 0 0 2 3]
AST = [27 27 25 23 21 15 13 14 17 20 23 26]
ASP = [0 0 0 0 2 3 3 2 1 1 0 0]
WSNT = [14 17 20 24 26 27 28 28 25 22 18 16] %WarmSemiNorthTemp
WSNP = [19 18 19 32 55 71 50 85 145 69 22 15]
CSNT = [-7 -3 1 5 10 15 19 18 13 7 0 -5] %Cold Semi North Temp
CSNP = [15 10 18 22 44 46 31 30 29 15 14 15]
WSST = [28 28 25 22 18 16 14 17 20 24 26 27] %WarmSemiNorthTemp
WSSP = [50 85 145 69 22 15 19 18 19 32 55 71]
CSST = [19 18 13 7 0 -5 -7 -3 1 5 10 15] %ColdSemiNorthTemp
CSSP = [31 30 29 15 14 15 15 10 18 22 44 46]
Temp = [ST; ST; ST; ST; ST; ST; MNT; ST; CSNT; WSNT; TDNT; HCT; HSNT; TWT; HCT; MNT; TWT; TWT; WSST; CS
Prec = [SP; SP; SP; SP; SP; SP; MNP; SP; CSNP; WSNP; TDNP; HCP; HSNP; TWP; HCP; MNP; TWP; TWP; WSSP; CS
month = 1;
for i = 1:1:75
TempDiff(i) = 2*abs(Temp(i,month) - MNT(month))+abs(Temp(i,month) - WSST(month))+2*abs(Temp(i,month) -
PrecDiff(i) = 2*abs(Prec(i,month) - MNP(month))+abs(Prec(i,month) - WSSP(month))+2*abs(Prec(i,month) -
CloseTo22(i) = Temp(i,month)-22;
end
```

```
(2*MNT(month)+WSST(month) +2*(HSNT(month))+HCT(month))/6;
(2*MNP(month)+WSSP(month) +2*(HSNP(month))+HCP(month))/6;
CSNP(month);

AverageTempDiff = TempDiff/11
AveragePrecDiff = PrecDiff/11
CloseTo22(30)

Time = [2 19 2 14 3 3 3 4 4 3 5 5 6 5 6 6 4 5 6 7 6 7 7 10 11 11 11 9 12 12 11 11 11 11 12 13 12 13 12

for i = 1:1:75
N = Time(i) - Time(15);
if (N >= 12)
N = abs(N-24)/2;
elseif (N < -12)
N = N+24;
elseif(N < 0)
N = abs(N)/2;
end

N1 = Time(i) - Time(59);
if (N1 >= 12)
N1 = abs(N1-24)/2;
elseif (N1 < -12)
N1 = N1+24;
elseif(N1 < 0)
N1 = abs(N1)/2;
end

N2 = Time(i) - Time(57);
if (N2 >= 12)
N2 = abs(N2-24)/2;
elseif (N2 < -12)
N2 = N2+24;
elseif(N2 < 0)
N2 = abs(N2)/2;
end


N3 = Time(i) - Time(58);
if (N3 >= 12)
N3 = (N3-24)/2;
elseif (N3 < -12)
N3 = N3+24;
elseif(N3 < 0)
N3 = abs(N3)/2;
end

N4 = Time(i) - Time(37)
if (N4 >= 12)
N4 = abs(N4-24)/2;
elseif (N4 < -12)
N4 = N4+24;
elseif(N4 < 0)
N4 = abs(N4)/2
end


N6 = Time(i) - Time(71);
if (N6 >= 12)
N6 = abs(N6-24)/2;
```

```
elseif (N6 < -12)
N6 = N6+24;
elseif(N6 < 0)
N6 = abs(N6)/2;
end

N5 = Time(i) - Time(30);
if (N5 >= 12)
N5 = abs(N5-24)/2;
elseif (N5 < -12)
N5 = N5+24;
elseif(N5 < 0)
N5 = abs(N5)/2;
end

Timescore(i) = 2*N+N1+N2+2*N3+2*N4+2*N5+N6;
Timescore1(i,:) = [2*N+N1+N2+2*N3+2*N4+2*N5+N6];
end

TotalFlightHours =  [87.03053512,       77.21702092,       89.3114396 ,       71.33513886,       90.576


AverageFlightHours = TotalFlightHours/11;

AveSleepLoss = Timescore/11 + AverageFlightHours/2;

WorkPercentLoss = AverageTempDiff/10 + AveragePrecDiff/100 + abs(CloseTo22)*11/100 + AveSleepLoss*6/8


month = 1;
for i = 1:1:75
TempDiff1(i,:) = [abs(Temp(i,month) - MNT(month)), abs(Temp(i,month) - MNT(month)),abs(Temp(i,month) -
PrecDiff1(i,:) = [abs(Prec(i,month) - MNP(month)),abs(Prec(i,month) - MNP(month)), abs(Prec(i,month) -
CloseTo22(i) = Temp(i,month)-22;
end

STDTempDiff = std(TempDiff1')
STDPrecDiff = std(PrecDiff1')

STDTimescore = std(Timescore1')

STDFlightTimes =    [1269.09672493,      1375.50004624,      1445.12816594,      1867.78773924,      17

EqualingScore = STDTempDiff/10 + STDPrecDiff/100 + abs(CloseTo22)*11/100 + STDTimescore*6/8+STDFlightTi

Cost = [20678.23203711909, 18085.070122828656, 19917.324522819617, 16075.66170145613, 20103.19903130659
FindIt = WorkPercentLoss + Cost*0.002+EqualingScore


c = [0,1/121,2/121];
e = [0,0.5,1];


for k = 1:1:size(c,2)
for i = 1:1:size(e,2)
    Istanbul(i,k) = EqualingScore(37)*e(i)+Cost(37)*c(k)+WorkPercentLoss(37);
    Khartoum(i,k) = EqualingScore(40)*e(i)+Cost(40)*c(k)+WorkPercentLoss(40);
    Perm(i,k) = EqualingScore(44)*e(i)+Cost(44)*c(k)+WorkPercentLoss(44);
end
end
```

```
CO(:,:,1) = zeros(3); % red
CO(:,:,2) = zeros(3);
CO(:,:,3) = ones(3); % blue

CO2(:,:,1) = zeros(3); % red
CO2(:,:,2) = ones(3);
CO2(:,:,3) = zeros(3); % blue

CO3(:,:,1) = ones(3); % red
CO3(:,:,2) = zeros(3);
CO3(:,:,3) = zeros(3); % blue

surf(c,e,Istanbul,CO)
hold on
surf(c,e,Khartoum,CO2)
surf(c,e,Perm,CO3)

title('How Location Choice Changes as Cost and Equality are Considered')
xlabel('Weight Given to Total Cost')
ylabel('Weight Given to Equality Value')
zlabel('Total Score')


clf
figure(1)
t = linspace(1,75,75);
k = [0,1/10,5];
size(k,2)
for i = 1:1:size(k,2)
TotalScore(i,:) = EqualingScore*k(i)+WorkPercentLoss
end
surf(t,k,TotalScore)

WorkPercentLoss(37)

figure(2)
plot(k,TotalScore(:,37),'r--')
hold on
plot(k,TotalScore(:,40),'b')
plot(k,TotalScore(:,35),'g')
legend('Istanbul','Khartoum','Cape Town')
xlabel('How much do we care about equalizing the meeting over boosting total production')
ylabel('Score of Location (Lower is better)')
title('How the Location changes depending upon the weight of Equalization')
```

16.6 Formulas

To determine the Percent of Work Productivity Lost when choosing different locations, we used this formula:

$$L = .8S + .11C + 0.1dT + 0.01dP$$

Where L is the Percent Work Loss, S is sleep deprivation, C is the deviation from 22 degrees Celsius, dT is the average absolute change in Temperature, and dP is the average absolute change in Precipitation.

For the Total Score, including Work Loss, Equality, and Cost, we used this formula:

$$Score = Productivity + c_1 * Equality + c_2 * Cost$$

Where c_1 and c_2 represent how much weight is given to Equality and Cost respectively.

To determine the cost per percent productivity, we used the following steps:

$$\frac{AvgHotelCost}{1night} * \frac{1night}{8hourssleep} * \frac{4hoursleep}{3\%productivity} = \frac{\$34.4}{1\%productivity}$$

16.7 Derivations

16.7.1 Haversine Formula

Source 19 states that for any two points on a sphere, the haversine of the central angle between them is given by

$$\mathrm{hav}(\frac{d}{r}) = \mathrm{hav}(\rho_2 - \rho_1) + \cos(\rho_1)\cos(\rho_2)\mathrm{hav}(\lambda_2 - \lambda_1) \tag{2}$$

where

- hav is the haversine function:
$$\mathrm{hav}(\theta) = \sin^2(\frac{\theta}{2}) = \frac{1-\cos(\theta)}{2} \tag{3}$$
- d is the distance between the two points (along a great circle of the sphere),
- r is the radius of the sphere,
- p_1 and p_2 are, respectively, latitude of point 1 and latitude of point 2 in radians,
- λ_1 and λ_2 are, respectively,longitude of point 1 and longitude of point 2 in radians.

Solve for d by applying the inverse haversine (if available) or by using the arcsine function:

$$d = r\mathrm{hav}^{-1}(h) = 2r\arcsin(\sqrt{h}) \tag{4}$$

where h is $\mathrm{hav}(\frac{d}{r})$, or more explicitly:

$$d = 2r\arcsin\left(\sqrt{\mathrm{hav}(\rho_2 - \rho_1) + \cos(\rho_1)\cos(\rho_2)\mathrm{hav}(\lambda_2 - \lambda_1)}\right) \tag{5}$$

$$= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\rho_2 - \rho_1}{2}\right) + \cos(\rho_1)\cos(\rho_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{6}$$