Emergency Siren Sound Propagation and Coverage Optimization Analysis

Barbara Webster

University of Redlands

Follow this and additional works at: https://inspire.redlands.edu/gis_gradproj

Part of the Geographic Information Sciences Commons, Other Civil and Environmental Engineering Commons, and the Other Physics Commons

Recommended Citation

This work is licensed under a Creative Commons Attribution 4.0 License.
This material may be protected by copyright law (Title 17 U.S. Code).
This Thesis is brought to you for free and open access by the Theses, Dissertations, and Honors Projects at InSPIRe @ Redlands. It has been accepted for inclusion in MS GIS Program Major Individual Projects by an authorized administrator of InSPIRe @ Redlands. For more information, please contact inspire@redlands.edu.
Emergency Siren Sound Propagation and Coverage Optimization Analysis

A Major Individual Project submitted in partial satisfaction of the requirements for the degree of Master of Science in Geographic Information Systems

by
Barbara Webster

Fang Ren, Ph.D., Committee Chair
Mark Kumler, Ph.D.

December 2014
Emergency Siren Sound Propagation and Coverage Optimization Analysis

Copyright © 2014

by
Barbara Webster
The report of Barbara Webster is approved.

Fang Ren, Ph.D., Committee Chair

Mark Kumler, Ph.D.

December 2014
Acknowledgements

I would like to thank Fang Ren, my advisor, for providing positive and constructive guidance, support, and feedback throughout the development of the project. I would also like to thank Vito “Rocky” Milano for taking the time to patiently bring me up to speed on the basics of sound propagation. Third, I would like to thank Mark Kumler, my second reader for providing valuable feedback that helped to refine this project, and finally I would like to thank my friends, family, and the MS GIS community for unwavering support throughout the entire year.
Abstract

Emergency Siren Sound Propagation and Coverage Optimization Analysis

by
Barbara Webster

Emergency warning sirens are a mass notification method that is used worldwide to quickly notify the public of a variety of dangers including flooding and acts of war. Siren network planners are faced with the question of where to locate new sirens to effectively reach citizens in a time of crises. This process answers a request from a GIS Analyst at the Bahrain Ministry of the Interior to create an ArcGIS Desktop Tool that predicts how far a siren’s sound will reach while considering siren properties and environmental variables, and then recommending siren locations that will completely cover a study area. This was accomplished by implementing a sound propagation engineering model and optimization analysis into two Python script tools that accept user input for any study area and output recommended siren points and their associated coverage area polygons. The sound propagation engineering model used was the ISO-9613-2 model with alterations recommended in the Joule Report.
# Table of Contents

Chapter 1 – Introduction .................................................. 1  
  1.1 Client ................................................................. 1  
  1.2 Problem Statement ............................................... 1  
  1.3 Proposed Solution ............................................... 1  
    1.3.1 Goals and Objectives .................................. 2  
    1.3.2 Scope ....................................................... 2  
    1.3.3 Methods ................................................... 2  
  1.4 Audience ......................................................... 3  
  1.5 Overview of the Rest of this Report ...................... 3  

Chapter 2 – Background and Literature Review .................. 5  
  2.1 Outdoor Sound Propagation Engineering Models .......... 5  
  2.2 Existing GIS Implementations of Outdoor Sound Propagation Models .... 7  
  2.3 Location Set Coverage Problem ............................... 8  
  2.4 Summary .......................................................... 8  

Chapter 3 – Systems Analysis and Design ...................... 11  
  3.1 Problem Statement ............................................. 11  
  3.2 Requirements Analysis ...................................... 11  
    3.2.1 Functional Requirements ............................ 12  
    3.2.2 Non-Functional Requirements ................... 12  
  3.3 System Design .................................................. 13  
  3.4 Project Plan .................................................... 13  
  3.5 Summary ........................................................ 14  

Chapter 4 – Database Design ........................................ 15  
  4.1 Conceptual Data Model ..................................... 15  
  4.2 Logical Data Model ........................................... 16  
  4.3 Data Sources and Collection Methods ................... 18  
  4.4 Data Scrubbing and Loading ............................... 18  
  4.5 Summary ........................................................ 19  

Chapter 5 – Implementation ......................................... 21  
  5.1 Generating Receiver and Siren Point Grids ............... 21  
  5.2 Sound Propagation Algorithms ............................ 24  
    5.2.1 Attenuation Due to Geometric Divergence .......... 25  
    5.2.2 Atmospheric Absorption Attenuation ............. 26  
    5.2.3 Attenuation Due to Terrain ....................... 27  
    5.2.4 Attenuation Due to Wind ............................. 31  
    5.2.5 Finding Siren Coverage Area Polygons .......... 33  
  5.3 Optimal Siren Configurations ............................. 34  
  5.4 Existing Siren Coverage .................................... 36  
  5.5 Summary ........................................................ 38  

Chapter 6 – Results and Analysis ................................ 39  
  6.1 Sound Propagation Prediction Accuracy Assessment .... 39  
  6.2 Optimization Analysis ....................................... 40
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>System Design</td>
<td>13</td>
</tr>
<tr>
<td>4-1</td>
<td>Conceptual Model</td>
<td>15</td>
</tr>
<tr>
<td>4-2</td>
<td>Geodatabase Contents</td>
<td>18</td>
</tr>
<tr>
<td>5-1</td>
<td>Potential Siren Points</td>
<td>22</td>
</tr>
<tr>
<td>5-2</td>
<td>Receiver Points</td>
<td>23</td>
</tr>
<tr>
<td>5-3</td>
<td>Attenuation Factors Stored in Attribute Fields</td>
<td>24</td>
</tr>
<tr>
<td>5-4</td>
<td>Sight Lines</td>
<td>25</td>
</tr>
<tr>
<td>5-5</td>
<td>Geometric Spreading</td>
<td>26</td>
</tr>
<tr>
<td>5-6</td>
<td>Terrain Attenuation Case 1</td>
<td>28</td>
</tr>
<tr>
<td>5-7</td>
<td>Siren and Receiver Point Heights</td>
<td>29</td>
</tr>
<tr>
<td>5-8</td>
<td>Terrain Attenuation Case 2 and 3</td>
<td>30</td>
</tr>
<tr>
<td>5-9</td>
<td>Example Potential Siren Coverage Areas</td>
<td>33</td>
</tr>
<tr>
<td>5-10</td>
<td>Find Recommended Siren Locations Tool Structure</td>
<td>35</td>
</tr>
<tr>
<td>5-11</td>
<td>Update Study Area Tool Structure</td>
<td>37</td>
</tr>
<tr>
<td>6-1</td>
<td>Recommended Sirens Using S-ROPE Method</td>
<td>41</td>
</tr>
<tr>
<td>6-2</td>
<td>Find Recommended Siren Locations Tool Interface</td>
<td>43</td>
</tr>
<tr>
<td>6-3</td>
<td>Update Study Area Tool Interface</td>
<td>44</td>
</tr>
<tr>
<td>6-4</td>
<td>Siren Point in an Environment with Complex Terrain</td>
<td>45</td>
</tr>
<tr>
<td>6-5</td>
<td>Terrain Attenuation</td>
<td>46</td>
</tr>
<tr>
<td>6-6</td>
<td>Siren Coverage Area</td>
<td>47</td>
</tr>
<tr>
<td>6-7</td>
<td>Effect of Temperature on Siren Sound Propagation</td>
<td>48</td>
</tr>
<tr>
<td>6-8</td>
<td>Finding Siren Locations That Completely Cover a Coverage Area</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Tables

Table 3-1. Requirement Traceability Matrix ................................................................. 11
Table 4-1. Datasets by Data Type.................................................................................. 17
Table 5-1. Atmospheric-Absorption Attenuation Coefficients........................................ 27
Table 5-2. Pasquill Stability Category.............................................................................. 31
Table 5-3. Meteorological Category................................................................................ 32
Table 6-1. Tool Results vs. Manufacturer Specified Ranges......................................... 40
Table 6-2. Python Script Tool Runtimes ........................................................................ 45
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIS</td>
<td>Geographic information system</td>
</tr>
<tr>
<td>DEM</td>
<td>Digital elevation model</td>
</tr>
<tr>
<td>dB</td>
<td>Decibels</td>
</tr>
<tr>
<td>SPReAD</td>
<td>System for the Prediction of Acoustic Detectability</td>
</tr>
<tr>
<td>ROPE</td>
<td>Rank and Overlap Elimination</td>
</tr>
<tr>
<td>S-ROPE</td>
<td>Stochastical Rank and Overlap Elimination</td>
</tr>
</tbody>
</table>
Chapter 1 – Introduction

Emergency sirens are an internationally used method of quickly warning entire populations of natural or man-made emergencies. Anyone planning a new emergency siren network or updating an existing network is faced with the question of how far the sound from each siren will reach, and where to locate the sirens to achieve the most coverage with the smallest number of sirens. During an emergency, the safety of a large number of people depends on the successful operation of the siren network. Governments and private safety administrators invest large amounts of money in the planning and deployment of these siren networks in order to keep people safe, and the ability to eliminate unnecessary spending accelerates the completion of necessary siren networks.

An often used but simplistic method of finding the coverage area is to buffer siren points with the siren range provided by the manufacturer. This method does not consider the surrounding environment, and is not always a realistic representation. It is impractical to attempt to account for all the environmental variables that have an effect on how far sound travels over a landscape, but it is still possible to model major variables which often have significant effects on sound propagation. This project does that by modeling the effect of ambient temperature, ambient humidity, wind speed, terrain, and siren elevation (which is determined by siren height above surface, terrain elevation, and building height if the siren is located on top of a building), and also takes siren power and elevation into account. A geographic information system (GIS) lends itself well to this kind of analysis because it can quickly extract required information from spatial data and efficiently handle large datasets. GIS is also well suited to finding the best locations for sirens so that an entire area will be covered because the GIS is capable of iterating through many possible siren options and finding the best choice.

1.1 Client

The client for this project is Yasser Al-Burshaid, a GIS Specialist at the Bahrain Ministry of Interior. Mr. Al-Burshaid has worked on GIS projects in homeland security, public safety, and law enforcement fields. Mr. Al-Burshaid has requested a GIS tool that will allow him to make more informed recommendations regarding Bahrain’s existing siren network.

1.2 Problem Statement

Mr. Al-Burshaid requested an Esri ArcGIS Desktop Python script tool that recommends the best locations to put new sirens in an urban environment in order to cover the entire area. He also wanted the tool to consider siren power (in decibels), siren height, ambient temperature, ambient humidity, terrain, building height, wind direction, and wind speed.

1.3 Proposed Solution

The proposed solution is an Esri ArcGIS Desktop Python script tool that accepts user input data representing siren properties and environmental conditions for any study area.
The tool uses a sound propagation engineering model to generate coverage areas for each siren. These coverage areas are then used in an optimization analysis to determine which siren locations are recommended.

### 1.3.1 Goals and Objectives

The goal of this project was to make a GIS tool that allows the client to make more informed recommendations regarding Bahrain’s existing siren network. This broke down into two major objectives. The first objective was to determine the coverage area of any given siren location in a study area while considering the client’s list of environmental factors and siren properties. The second objective was to implement a coverage optimization analysis to select and recommend the best siren locations.

### 1.3.2 Scope

Mr. Al-Burshaid requested a system that takes environmental conditions and siren properties into account to help a user distribute sirens in a defined area so that the sirens can be heard anywhere in that area. The variables that may affect sound propagation were ambient air temperature, ambient humidity, wind speed, terrain, siren power, and siren elevation (including elevation added by being on top of buildings). The final deliverable should be an ArcGIS Desktop tool that recommends the best siren placements based on user-specified parameters. The best locations are those that minimize the number of sirens used to cover the entire area.

This research, development, and finalization of this project was limited to a time frame of less than a year. This limited the complexity of the sound propagation model that can be used. Further discussion of the models available and the models selected for this project is in Section 2.1 and 3.3 respectively.

### 1.3.3 Methods

The tool created in this project analyzes an area of desired coverage by creating a grid of potential siren location points and a grid of receiver points. It then calculates the receiver point attributes with the sound pressure level (in decibels) corresponding to each siren within range. The tool uses two sound propagation engineering models to generate these sound pressure level attributes while considering the siren properties and environmental variables. The model used is a model called ISO-9613-2 (International Standard for Organization, 1996) with attenuation factor calculation methods for wind and terrain from the CONCAWE model (Manning, 1981) and wind farm noise propagation model, commonly referred to as the Joule Report (Bass, Bullmore, and Sloth, 1998). After the receiver point sound pressure levels are calculated, the tool creates a coverage area polygon for each siren at a specified sound pressure level. This coverage area represents the sound coverage area for each siren. Once the coverage area for each siren is established, the recommended siren locations are determined by using Rana’s ROPE coverage optimization analysis on the coverage areas (Rana, 2006). The study area used for most of the development of this tool Portland, Oregon.
1.4 Audience

This project report is intended for GIS professionals who have a need for siren optimization. These individuals will most likely be familiar with the technical GIS vocabulary, the requirements of using Python script tools, and the required datasets such as DEMs and feature classes.

1.5 Overview of the Rest of this Report

The first chapter of this report briefly introduces the project, the client, the problem statement, the goals, scope, and methods included in the proposed solution, and the audience. The second chapter consists of a background and literature review of sound propagation engineering models and their implementation in GIS, and of coverage analyses. The third chapter discusses the systems analysis and design, and includes a problem statement, requirements analysis, system design, and project plan. The fourth chapter discusses database design, and includes the conceptual data model, logical data model, data sources, data collection methods, and data scrubbing and loading. The fifth chapter details each step of the project. The sixth chapter presents and discusses the results of the project, and the seventh chapter presents the conclusions and future work.
Chapter 2 – Background and Literature Review

This project involved two major components: modelling the sound propagation of a given siren while considering environmental factors and siren properties, and finding the best distribution of sirens to cover the entire study area. Section 2.1 discusses some of the outdoor sound propagation physics models and an overview of the existing GIS implementations of outdoor sound propagation models is provided in Section 2.2. Section 2.3 addresses the problem of finding the best siren locations. The chapter ends with a summary in Section 2.4.

2.1 Outdoor Sound Propagation Engineering Models

An acoustic engineering model provides the algorithm for calculating the noise propagation in a given environment (Kang, 2007). There are many engineering models to choose from, and there is no clear consensus on which model is the best. Most models use a variety of calculation methods and routinely borrow components from other models. The following is a brief discussion of some of the available engineering models, including ISO-9613-2, NORD2000, Harmonoise/Imagine, CONCAWE, and the Joule Report.

One of the most widely used engineering models is titled Acoustics – Attenuation of Sound During Propagation Outdoors – Part 2: General Method of Calculation, and is commonly known as ISO-9613-2. It is an international standard published by the International Organization for Standardization. Like most engineering models, and all models discussed in this paper, ISO-9613-2 is a method for calculating outdoor sound propagation from a source point to a receiver point given certain environmental conditions. It considers geometric divergence (how sound pressure levels decrease as distance from the source increases), atmospheric absorption, ground effect, reflection from surfaces, and screening by obstacles. Each of these factors is calculated as an attenuation factor. Attenuation factors are subtracted from the decibel level that is determined by the siren power and directivity correction. An important note for the application of ISO-9613-2 model to this project is that it only models favorable propagation conditions. This means that it does not model upwind attenuation, or how much the wind decreases the sound pressure level at a receiver point if it is blowing the sound back toward the siren. Also, its ground attenuation factor calculates attenuation from reflections off of a perfectly flat terrain, and does not consider complex terrain (International Standard for Organization, 1996).

In a small online survey, P.E. Mediterranean Acoustics Research & Development found that 74% of members of the Institute of Acoustics reported using ISO-9613-2 more than other models, and 63% of members from The Acoustical Society of America reported using the ISO-9613-2 model the most (Economou & Charalampous, 2012). One reason ISO-9613-2 is so widely used may be that its calculations are relatively simple, which means that they are easy to program, and that calculation results are consistent (Bass, Bullmore, and Sloth, 1998). ISO-9613-2 also commands a certain level of authority because it is the international standard, although proponents of newer, more complex models challenge the validity of this reasoning.
More complex models have been developed since the creation of ISO-9613-2. These models consider more complex environments, which is increasingly possible due to advances in computing technology (Nota & VanLeeuwen, 2004). The model Nord2000, for example, was published in 2000 and was funded primarily by the Nordic Council of Ministers. Nord2000 handles moving sources and both irregular and multipart screening objects (DELTA, 2002).

In 2002 the European Union adopted the Environmental Noise Directive, which aimed to monitor and reduce noise pollution in Europe. One of the requirements was that noise maps be created for major roads, railways, airports and agglomerations (European Parliament, Council of the European Union, 2002, p. 14). Noise maps are a way of representing sound levels across an area by creating contours at specific decibel levels (Kang, 2007). The directive sparked a demand for a standardized sound propagation model that was capable of handling complex acoustic environments. As mentioned above, the ISO-9613-2 model is capable of accurately predicting relatively simple acoustic situations, but the Environmental Noise Directive called for a model that was responsive to a greater range of sources and environmental complexities with even greater accuracy than previous models. This prompted the creation of the Harmonoise/Imagine project, which developed new prediction methods that met the Environmental Noise Directive’s requirements and provided a standard for the European states attempting to comply with the directive. The Harmonoise/Imagine Project used components of previously developed models (including both ISO-9613-2 and NORD2000), and it also developed new sound prediction sub-algorithms (Nota & VanLeeuwen, 2004).

Two other examples of engineering models are the CONCAWE model, which was prepared for the environmental group CONCAWE in 1981 to predict noise emissions from petroleum and petrochemical complexes (Manning, 1981), and the Joule Report model, which was written in 1998 by Bass, Bullmore and Sloth. Both of these models take advantage of empirical modeling results, which formulate calculations based on measured conditions and results, instead of modeling the processes that cause the results, as the advanced calculation models tend to do. Empirical prediction methods simplify what can otherwise be a very complicated calculation, although they can sacrifice accuracy when they are used in situations that differ too substantially from the original measurement collection conditions. Both the CONCAWE model and the Joule Report model provide simple methods for modeling complicated factors in noise propagation, such as upwind attenuation and complex terrain. Bass recommended a complex terrain attenuation factor be added to a modified ISO-9613-2 model. This modified ISO-9613-2 model is also known as the IEA (International Energy Agency) model, and simply omits all the ISO-9613-2 attenuation factors except for geometric spreading and atmospheric absorption. In his wind farm model, Bass defined three terrain types that prompt a specific attenuation in decibels. Similarly, the CONCAWE model defines eight meteorological categories with an associated equation that calculates the attenuation factor value based on the distance between source and receiver point. Each category is a function of wind speed, distance from the noise source, and the approximate amount of clouds or sun.

Economou et al. (2012) succinctly summarized the tradeoffs between simple models (like ISO-9613-2, CONCAWE and the Joule Report model) and the advanced calculation
methods (like NORD2000 and Harmonoise/Imagine). They stated that, “standards usually provide a simpler mathematical code, with fast and approximate results. On the other hand, state of the art science provides complicated mathematical computation, slower and yet more accurate results than standardised methods” (2012, p. 1).

The validation results of sound propagation engineering models depend greatly on the testing conditions and whether the model is being used in the type of environment for which it was designed. When field tested in a relatively simple environment, ISO-9613-2 performs very well. In one test, Evans and Cooper of AECOM reported that “ISO 9613-2 states a prediction accuracy of ± 3 dB for sources of heights up to 30 metres above ground and for distances up to 1000 metres from the source” (Evans & Cooper, 2012). Similarly, when tested at various wind farms, the Joule Report model was “shown to result in calculated sound pressure levels that lie within 2dB of the level not expected to be exceeded for at least 85% of the time” (Bass, Bullmore, and Sloth, 1998, p. 15). The advanced calculation models nearly always outperform the simple models in validation tests. In his comparison of measured results, Economou showed that NORD2000 matches measured sound levels much better than ISO-9613-2 in a test on flat ground with one barrier. However, it is also evident that the sound pressure level predictions from ISO-9613-2 do follow the trend of the measured sound levels even if they are not as precise as the NORD2000 predictions, and may be useful in general planning situations (Economou & Charalampous, 2012).

In summary, the models discussed here are just a few of the many outdoor sound propagation engineering models, which range from fast, simplified models, to complicated but more accurate advanced calculation models. The best model choice depends on the project requirements, and any project that incorporates a sound propagation engineering model should bear in mind the limitations of the model used.

### 2.2 Existing GIS Implementations of Outdoor Sound Propagation Models

Many software packages that use GIS to model outdoor sound propagation are already available. The majority of these are large commercial software packages such as SoundPlan and CadnaA, which implement advanced calculation methods and provide sophisticated cartographic visualizations of the results (SoundPlan-Acoustics, n.d., CadnaA, n.d.). Fewer open source GIS implementations exist, but Reed, Boggs, and Mann (2012) recently released an open source example called System for the Prediction of Acoustic Detectability (SPreAD-GIS). SPreAD-GIS is a GIS implementation of a US Forest Service engineering model called SPreAD (Reed, Boggs, & Mann, 2012).

Receiver grids are a common method of implementing propagation models to show sound distribution across an area (SoundPlan Grid Noise Map, n.d.). Receiver grids consist of generating a point grid that covers a desired area and then calculating the sound pressure generated by one or more sources at each point. The grid points can then be interpolated into noise contours representing specific sound pressure levels, or they can be used to generate some other form of visualization (Licitra, 2013).
2.3 Location Set Coverage Problem

Siren network optimization is an example of facility coverage optimization, which is a broader problem that applies to many disciplines. A question related to facility coverage optimization is the location set covering problem, which attempts to find the smallest number of facilities that will completely cover an area (Church & ReVelle, 1974). One way to address the location set covering problem is with an isovist analysis (De Smith, Goodchild, & Longley, 2007).

The term isovist literally means ‘same view.’ Benedikt (1979) defines the word as all the points that are visible from a given observation point. An isovist is similar to a viewshed, which uses digital elevation models (DEMs) to find lines of site between two points on a surface (Llobera, 2003). There are two major differences between a viewshed and an isovist. First, an isovist usually analyzes vector data while a viewshed analyzes raster data. Second, the term isovist is typically used within the domain of urban and architectural studies while the term viewshed is typically used in most other domains (Llobera, 2003).

In 2006, Rana proposed a type of isovist analysis that De Smith, Goodchild, and Longley effectively summarized as an analysis that is executed by performing a “…systematic evaluation of visibility from a fine grid of initial observer positions, and then reducing this set by a ranking rule” (De Smith, Goodchild, & Longley, 2007, p. 279). Rana’s method is called the Rank and Overlap Elimination (ROPE) method. The analysis starts with the point from which the most space is visible (the highest-ranking point), and then eliminates any observation points that are visible from that point. It repeats this with the next highest-ranking observation point until the set of observation points is reduced as much as possible. Rana later added a stochastic element to this method, resulting in the Stochastic Rank and Overlap Elimination (S-ROPE) method. This method repeats the ROPE method many times with a random, different initial point chosen every time. This results in many sets of recommended observer points, and the set with the smallest number of observer points is the set that is selected.

Isovist analysis typically operates in terms of observation points and visibility, however, it can also be applied to solve a sound coverage problem. The most important difference between sound propagation and visibility is how impeding objects affect the sound or site range. Sound can go around objects to a certain extent, while a line of site cannot unless the object is transparent. Rana’s isovist analysis allows for this difference because it only requires that each point has an associated coverage area so that the coverage area can be ranked and reduced. If the coverage area of each potential siren is determined, the highest-ranking location for siren will be identified through each iteration until the whole area is covered. The final output will contain the desired spatial configuration of sirens that provide complete sound coverage.

2.4 Summary

The numerous sound propagation engineering models have varying levels of complexity and accuracy. They can be modified to suit the needs of the user, as is seen in the Joule Report model and the Harmonoise/Imagine model. The most common way of
implementing engineering models in a GIS is by generating coverage areas based on a receiver point grid that reflects the sound pressure level throughout the study area.

Rana’s isovist analysis answers the location set coverage problem by iteratively selecting the siren with the largest coverage area, eliminating sirens within this area, and continuing to the next largest coverage area until the entire area it covered. He also suggested repeating this process multiple times, trying each siren as the first siren selected.
Chapter 3 – Systems Analysis and Design

This chapter defines the problem statement and both the functional and non-functional requirements of the project (Section 3.1 and 3.2). It then presents the system design solution to the problem statement and requirements (Section 3.3). Section 3.4 reviews the original project plan and offers a brief analysis of how closely the project team adhered to the plan throughout the span of the project.

3.1 Problem Statement

Mr. Al-Burshaid requested an Esri ArcGIS Desktop Python script tool that would recommend the best locations to put new sirens in order to cover the entirety of a study area. The tool would consider siren power (in decibels), siren height, ambient temperature, ambient humidity, terrain, and wind speed.

3.2 Requirements Analysis

A critical component of a successful project is to thoroughly understand the client’s requirements. The client’s requirements were determined by his need to help plan and improve Bahrain’s emergency siren system, and by his inability to provide site-specific data. The following two sections (3.2.1 and 3.2.2) discuss the functional and non-functional requirements of the project. These requirements are also summarized in Table 3-1.

Table 3-1. Requirement Traceability Matrix

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Notes and Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Tool will recommend the most efficient siren distribution.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tool will consider environmental factors.</td>
<td>Environmental factors to be considered are ambient temperature, ambient humidity, wind speed, wind direction, building height, and terrain</td>
</tr>
<tr>
<td></td>
<td>Tool will consider siren properties.</td>
<td>Siren properties to be considered: siren power (in dB) and siren height above terrain</td>
</tr>
<tr>
<td></td>
<td>Sirens that are not on buildings will be mounted on towers. The tool should be able to handle this.</td>
<td></td>
</tr>
</tbody>
</table>
3.2.1 Functional Requirements

The core functional requirement for this project was that the tool would be able to recommend emergency siren locations so that the siren sound would completely cover an area. This would take the form of an output point set representing the recommended siren locations. The client also specified two other functional requirements to be considered in determining the optimal siren locations: environmental factors and siren properties. The environmental factors were ambient temperature, ambient humidity, wind speed, wind direction, building height, and terrain. The siren properties are siren power (in dB) and siren height above terrain. According to the client, the sirens that are not located on top of buildings will be mounted on tall towers.

The first four functional requirements were explicitly requested by the client. The next two were not listed in the client’s original project request, but were necessary to meet the first three requirements. First, in order to find the best combination of siren locations, a set of possible siren locations had to be generated so that the most efficient subset could be selected. Second, the tool had to generate a sound coverage area for each siren that considered all of the requested factors. The project team decided that a simple sound propagation model would meet the requirements of this project better than one of the complex propagation models discussed in Chapter 2. The requirements call for an analysis that can be contained in a Python script tool, and be completed in less than a year. The intricacy of most of the complex calculation models like Harmonoise/Imagine makes their implementation outside the scope of the project. The client intends this tool only as a general planning tool, and the fast, approximate results of a simplified model would be appropriate for those purposes.

3.2.2 Non-Functional Requirements

The client included two non-functional requirements for this project. He could not supply any site-specific data for this project, so he requested a Python script tool for ArcGIS Desktop into which the environmental data and siren properties could be entered as user input. The second non-functional requirement was that the tool be compatible with ArcGIS Desktop 10.2.
3.3 System Design

The system design consisted of two ArcGIS Desktop Python script tools. The main tool was called Find Recommended Siren Locations, and was designed to execute both of the analytical components of this project, which are modelling siren sound coverage in an outdoor environment, and optimizing the spatial distribution of these sirens across a study area (Figure 3-1). The other tool, Update Study Area, finds the coverage area of existing sirens and subtracts them from the study area. Figure 3-1 shows the system design.

![Figure 3-1. System Design](image)

In the first component — modeling the sound propagation in an outdoor environment — the tool used the ISO-9613-2 outdoor sound propagation model algorithm modified with attenuation factor calculation methods from two other empirical models. The CONCAWE model was chosen for the attenuation factor of upwind sound loss and the Joule Report model was used to calculate the attenuation factor due to terrain. These algorithms generate the sound pressure level in decibels for a receiver point that is associated with a source point, while considering the required environmental and source factors specified by the client. The second component — siren optimization — was addressed by applying Rana’s ROPE coverage optimization (Rana, 2006).

The tool generates both a grid of receiver points and a grid of potential siren points. The decibel level associated with each siren is stored in the sight lines layer as a separate attribute, and can be joined to the receiver points. The tool creates a coverage area for each potential siren, and this allows the tool to apply Rana’s coverage optimization.

3.4 Project Plan

The plan for this project divided the work flow into four phases: initiation, development, implementation, and testing. The initiation phase consisted of confirming the requirements with the client and formulating a plan to test the accuracy of results. The first task of the development phase consisted of defining the format of the input datasets. The focus of this task was to find widely available datasets so that this tool would be easily used for any study area. The second task was to develop both components of the
analysis and a strategy for assuring that they were compatible with each other. The implementation phase consisted of writing the Python script for both analysis components. The testing phase involved testing for runtime errors, using the tool to calculate results for different study areas, and implementing the plan to test accuracy that was decided upon in the initiation phase.

The sequence of milestone and tasks generally went as planned, but the original schedule was pushed back. This was partially a result of scheduling conflicts. The project team could have prevented the delay by subdividing the development phase into smaller tasks. Another aspect of this project that was not foreseen in the project plan was the extent of research that had to be done to find a suitable sound propagation model.

### 3.5 Summary

The client required an ArcGIS Desktop Python script tool that utilized user entered environmental and siren data to recommend the siren locations that would completely cover an area. In response to these requirements, the project team developed a system design that references a modified outdoor noise propagation engineering model to calculate siren sound coverage areas, and then uses Rana’s ROPE coverage optimization process to select the best siren distribution. The project team used a project plan that necessitated some modifications during implementation, but that facilitated project organization and scheduling.
Chapter 4 – Database Design

The final deliverable of this project consisted of the ArcGIS Desktop Python script tool, which will be used for the study area that client is interested in. The data used for testing purpose will not be delivered to the client. This section will focus on the conceptual and logical data models of the tool input data and discuss data sources, data collection methods, and data scrubbing and loading of a sample dataset.

4.1 Conceptual Data Model

The conceptual model for this project describes how a sound’s environment affects its propagation. The field of acoustics studies how sound waves reflect, refract, and diffract in response to their environment. These factors could potentially be studied to a nearly infinitesimal level of detail, but this model attempts to convey only the most significant environmental factors in sound propagation.

Some of the most significant factors that affect sound propagation are geometric spreading, the directivity factor, the atmospheric condition (ambient temperature and ambient humidity), objects that act as barriers, and wind. Figure 4-1 illustrates this conceptual model.

![Conceptual Model](image)

Figure 4-1. Conceptual Model

The tendency of sound to diminish as distance from the source increases is called geometric spreading. This occurs because the siren’s sound power is distributed over an
increasingly large area, which causes the sound intensity at a given location to diminish (Crocker, 1998).

The directivity correction is based on whether the sound is propagating into free space, which is a complete sphere, or into some fraction of a sphere (Cutnell & Johnson, 2007). For example, if the sound source is sitting on a flat surface, its sound fills a half of a sphere, or a hemispherical sound propagation. But when sound does not enter into a free space, reflections of the sound waves can increase the sound pressure in the receiver point, which is called the directivity correction. In this project, it was assumed that the sound is propagating into free space, even though sound from each siren will almost certainly meet the ground at some point. This approach was taken for two reasons. First, the possibility of complex terrain, and the placement of sirens on top of buildings makes finding the directivity factor a very complicated calculation, since the source is not sitting above a simple flat plane. Second, assuming a free field directivity produces the most conservative prediction, since there are no reflections to add to the sound pressure level.

As sound waves travel through the air, the temperature and humidity of that air affect how sound waves propagate through it. In reality, there would be variations in the atmospheric condition across a study area. For example, the humidity will be greater over large bodies of water, and temperatures vary at different vertical elevations. For this project, the atmospheric condition was simplified to be consistent across the entire study area.

Objects that disrupt the sound’s path also influence the propagation of sound. Sound waves reflect (bounce off of), refract (change direction after partial absorption into a medium) and diffract (bend) around barriers (Cutnell & Johnson, 2007). The reflections in this process are affected by the porosity of the barrier surface. For example, a grassy hill will absorb more of a sound wave than a concrete building.

Wind influences a sound’s propagation by diffracting sound waves. In an upwind scenario, it bends sound waves upward, creating a shadow zone at ground level where the sound pressure is diminished or nonexistent. It can also attenuate sound that is downwind of the source by increasing the speed of the sound wave, and making it bend downward, sooner than it otherwise would have (Gabrielson, 2006). Like the atmospheric condition, wind also varies across a study area, but in the project it was simplified to one direction and velocity throughout the study area.

4.2 Logical Data Model

Once the entities in the analysis and the relationship between them were established in the conceptual data model, how these entities would be represented and organized in the geodatabase were determined. All logical data models must simplify real world phenomena to some extent. In this model the more transient conditions, like ambient temperature, ambient humidity, and wind were simplified to be represented by a non-spatial integer, text, or range value that was used across the entire study area, while the more permanent data like buildings and terrain, were represented by vector and raster data, respectively. See Table 4-1 for a complete list of the data by data type.
### Table 4-1. Datasets by Data Type

<table>
<thead>
<tr>
<th>Non-spatial Data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient Temperature</td>
<td></td>
</tr>
<tr>
<td>Ambient Humidity</td>
<td></td>
</tr>
<tr>
<td>Siren Frequency</td>
<td></td>
</tr>
<tr>
<td>Siren Height Above Terrain</td>
<td></td>
</tr>
<tr>
<td>Siren Height Above Building</td>
<td></td>
</tr>
<tr>
<td>Distance Between Receiver Points</td>
<td></td>
</tr>
<tr>
<td>Distance Between Potential Siren Points</td>
<td></td>
</tr>
<tr>
<td>Minimum Coverage Decibel Level</td>
<td></td>
</tr>
<tr>
<td>Wind Speed</td>
<td></td>
</tr>
<tr>
<td>General Weather Condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector Data</td>
<td></td>
</tr>
<tr>
<td>Study Area Polygon</td>
<td></td>
</tr>
<tr>
<td>Building Polygons (with height attributes)</td>
<td></td>
</tr>
<tr>
<td>Existing Sirens Points (Optional)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Raster Data</td>
<td></td>
</tr>
<tr>
<td>DEM</td>
<td></td>
</tr>
</tbody>
</table>

The spatial input data set was stored in an ArcGIS File Geodatabase format. Since much of the input data were non-spatial data, and did not need to be entered into a geodatabase, the File Geodatabase only contained four feature classes: the study area polygon, building polygons with height attributes, existing siren points, and the DEM (Figure 4-2).
4.3 Data Sources and Collection Methods

Since the client did not provide any data on the study area that he is interested in, a section of Portland, Oregon, was chosen as the study area for testing. The DEM for this example was obtained from Portland State University Center for Spatial Analysis and Research, and was created by rasterizing Lidar points that were classified as bare earth data. The building polygons with height attributes were obtained from a website called Civicapps (civicapps.org), which provides free datasets to encourage social change. The data were collected with a combination of techniques, including aerial photography, Lidar-derived, land use review or building permit information, photogrammetry, and 3D models.

The environmental variables represented by the non-spatial data are a hypothetical value that should be a realistic value for the study area.

4.4 Data Scrubbing and Loading

The sample dataset for this project required very little scrubbing. Both the building polygons and the DEM were selected because they consisted of the necessary data and metadata. An important issue to keep in mind when preparing data to use as input for this tool is to make sure that the extents of the DEM and polygon feature classes are at least as large as the extent of the receiver grid. This is one receiver grid width larger than the study area, as will be discussed in Chapter 5.
4.5 Summary

The conceptual data model is based on the study of sound propagation and consists of a list of factors that decrease the sound pressure at a specific receiver point. The logical data model was implemented in a File Geodatabase that represented the study area, existing sirens, and buildings as vector feature classes, and the terrain as a raster feature class. The study area and existing sirens were simple polygons created by the project team, and the buildings and DEMs were obtained from documented internet sources. This dataset required very little scrubbing or loading preparation.
Chapter 5 – Implementation

The deliverable of this project consists of two Esri ArcGIS Desktop Python script tools. The main tool is called Find Recommended Siren Locations. It creates a feature class of recommended siren points and a feature class of the coverage area polygons for the recommended siren points. A tool called Update Study Area is also provided in case there are previously existing sirens in the study area. This tool finds the coverage area of the existing sirens using the same methods as the main tool, and then subtracts the area covered by the existing sirens from the study area. Both the updated study area and the existing siren coverage areas are provided as output feature classes and the former can be loaded into the main tool.

This chapter explains each step in the analyses that these tools perform. Section 5.1 explains the process of generating the receiver and potential sirens point grids. Section 5.2 explains how the sound propagation algorithms are implemented in the tools, including how the tools calculate the attenuation from geometric divergence (Section 5.2.1), atmospheric absorption (Section 5.2.2), complex terrain (Section 5.2.3), and wind (Section 5.2.4), and how the siren coverage areas are generated by considering these attenuation factors (Section 5.2.5). Section 5.3 discusses how the recommended sirens are selected in the Find Recommended Siren Location tool, Section 5.4 describes how existing sirens are considered with the Update Study Area tool, and finally Section 5.5 summarizes the chapter.

5.1 Generating Receiver and Siren Point Grids

The first step in the analytical process is to generate the receiver and potential siren point grids. The Find Recommended Siren Locations tool then uses a sound propagation engineering model to calculate the decibel level associated with each siren in each receiver grid point. This allows the tool to generate a coverage area for each potential siren. These coverage areas are then used in an optimization analysis.

The intervals of both point grids are determined by user inputs. The grids are created by using the Create Fishnet function. The horizontal and vertical intervals are determined by the value specified by the user, and the extent is set to be the same as the study area polygon. The siren point grid is clipped with the study area polygon (Figure 5-1).
Unlike the siren points, the receiver grid is clipped with a polygon that is the result of the study area buffered by the receiver grid resolution (Figure 5-2). So if the receiver grid has a resolution of 100 meters, the receiver grid will be clipped by a shape that is the result of the study area buffered by 100 meters. Allowing the receiver points to extend past the study area boundary prevents edge effects from affecting the resulting siren coverage areas.
Generating the receiver and siren point grids is a simple step, but is important because it establishes the granularity, or resolution, of the analysis. Allowing the user to control this element of the analysis assumes that the user understands that the precision of the results depends on the resolution. For example, a receiver point grid with 100 meters between each point will provide a siren coverage area with a 100 meter resolution, as long as the DEM and building polygons are at this resolution or higher. The reason for allowing the user to control the grid resolution is that it allows the user to decide between higher precision results or faster processing time. It is recommended that the potential siren grid resolution be set to a larger value than the receiver grid resolution. This replicates the siren functionality in the real world, in which one siren will provide coverage to many locations.
5.2 Sound Propagation Algorithms

Sound propagation engineering models calculate the sound pressure level in decibels at a receiver point by considering the source sound power level and attenuation factors. The sound propagation engineering model used in this tool was the Wind Farm Model described in the Joule Report (Bass, Bullmore, and Sloth, 1998), which considers geometric divergence, atmospheric absorption, and terrain attenuation. Out of these three factors, the attenuation due to geometric divergence and the attenuation due to atmospheric absorption calculation methods are identical to those in the ISO-9613-2 model and the IEA model, while the attenuation due to terrain is unique to the Joule Report. The CONCAWE model was also applied for upwind attenuation. For more information on sound propagation engineering models, see Section 2.1.

Each attenuation factor was calculated and recorded as an attribute field. The highlighted columns in Figure 5-3 illustrate that the total sound pressure value is the result of the original sound power (100 dB in this example) minus the attenuation factor values in the five columns to the left of the total sound pressure. When an attenuation factor is calculated to be negative, this factor actually increases the total sound pressure instead of decreasing it.

![Figure 5-3. Attenuation Factors Stored in Attribute Fields.](image)

The sound propagation engineering model evaluates the relationship between one sound source and one receiver point. If multiple siren points are being evaluated, the sound propagation engineering model must evaluate the relationship between many receiver points and many sound sources. To simplify the data organization, the total sound pressure and associated attribute fields were stored in an intermediate feature class called sight lines. The sight lines feature class was created with the Construct Sight Lines function, available in the 3D Analyst extension of ArcGIS. Each sight line connects a receiver point to a siren point, and is used in generating the all of the attenuation factors (Figure 5-4). This feature class provides lines of sight that includes Z values. The geometry of this feature class is used to calculate the 3D distance between the receiver points and siren point, which is used several times in the analysis. The feature class geometry is also compared to the DEM raster to determine the visibility between points in Section 5.2.3.
The following subsections consist of a short description of each of the four attenuation factors (geometric divergence, atmospheric absorption, terrain, and upwind) and a summary of the geoprocessing approaches used in finding them, and also a subsection explaining how each potential siren coverage area is generated. See Appendix A for a spreadsheet of equations used in calculating all attenuation factors.

### 5.2.1 Attenuation Due to Geometric Divergence

As sound waves get further away from a sound source point, the total area that the sound wave covers increases. This means that the same sound power covers an increasing area, which diminishes the sound pressure as the area increases (Crocker, 1998). This is called the geometric divergence, which is the first attenuation factor (Figure 5-5).
The attenuation due to geometric divergence is a function of distance in meters. The equation to find the attenuation due to geometric divergence was obtained from ISO-9613-2 (International Standard for Organization, 1996, p. 8).

\[ A_{div} = \left[ 20 \log\left(\frac{d}{d_0}\right) + 11 \right] \]

where \( A_{div} \) is the attenuation due to geometric spreading measured in dB, \( d \) represents the distance in meters from the receiver point to the source point, and \( d_0 \) represents the reference distance of one meter. The distance \( d \) is obtained from the sight line feature class. The Python script tool adds and populates a three-dimensional distance in meters field to the sight lines feature class. The tool obtains the three dimensional distance by using the 3D length (length3D) property in the sight line polyline class, and is then converted to meters by using the meters per unit (metersPerUnit) property in the Spatial Reference class.

### 5.2.2 Atmospheric Absorption Attenuation

The amount that the sound pressure diminishes as a result of the atmospheric condition is determined by the distance between the source and receiver, the siren sound frequency, the ambient temperature, and the ambient humidity. The attenuation in decibels due to atmospheric absorption is equal to \( \alpha d/1000 \), where \( d \) is the distance in meters between the source and receiver point, and the atmospheric absorption coefficient (\( \alpha \)) is the value obtained from a series of tables depending on the sound frequency, ambient humidity, and ambient temperature. Table 5-1 illustrates the format of these tables, and includes some of the data from the full-length tables, which can be found in full in the ISO-9613-1 Report (International Standard for Organization, 1993).
Table 5-1. Atmospheric-Absorption Attenuation Coefficients.

<table>
<thead>
<tr>
<th>Air Temp.: -20 degrees C</th>
<th>% Humidity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (Hz)</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>1.59 2.27 3.08 4.92 6.75 8.22 9.07 9.31 9.09 8.6 7.98</td>
</tr>
<tr>
<td>1000</td>
<td>1.65 2.43 3.16 4.4 7.21 9.14 10.6 11.5 11.7 11.6 11.1</td>
</tr>
<tr>
<td>1250</td>
<td>1.74 2.43 3.27 5.28 7.57 9.88 11.9 13.5 14.4 14.8 14.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Air Temp.: -10 degrees C</th>
<th>% Humidity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (Hz)</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4.53 7.26 9.71 11.8 10.8 8.99 7.36 6.09 5.14 4.43 3.88</td>
</tr>
<tr>
<td>1000</td>
<td>4.65 7.6 10.6 14.4 14.6 12.9 10.9 9.19 7.82 6.75 5.91</td>
</tr>
<tr>
<td>1250</td>
<td>4.78 7.87 11.3 16.8 18.8 17.9 15.8 13.7 11.8 10.3 9.02</td>
</tr>
</tbody>
</table>

This set of tables was implemented as a triple-nested dictionary that was hardcoded into the Python script. Hard coding the data from the tables avoids having to include associated reference files with the tool, which simplifies the tool storage and download. The distance variable in the equation was obtained from the 3D distance in meters attribute field, as discussed in the previous section.

The atmospheric absorption coefficient reference tables consider the frequency, temperature, and humidity values incrementally. For example, the ambient humidity is considered in 10% increments, allowing input for percentage values of 0, 10, 20, etc., but not allowing for any value in between those increments. This restriction is placed on the user input values for frequency, temperature, and humidity, which are selected from a dropdown menu populated by a list of valid values.

5.2.3 Attenuation Due to Terrain

The attenuation due to terrain is 0 dB unless the terrain elevation between the siren point and receiver point is true for at least one of three cases. Each case has an associated, empirically-derived value in decibels that is assigned if that case is true (Bass, Bullmore, and Sloth, 1998).

Case 1 tests for a significant decrease in terrain elevation between the source and receiver point (Figure 5-6).
If both points are a certain amount higher than the terrain between them, then it will increase the final sound pressure level. The Joule Report provides the following formula to test for Case 1. Case 1 is true if:

\[ h_m \geq 1.5 \left( \frac{|h_s - h_r|}{2} \right) \]

where \( h_m \) equals the mean height above the ground of the direct line of sight between siren point and receiver point, \( h_s \) equals the height of the siren above ground, and \( h_r \) equals the height of the receiver point above ground level.

The variable \( h_m \) was derived by averaging the total elevation of the siren point and the elevation of the receiver points, and then subtracting the average terrain elevation below the sight line connecting the siren and receiver point. The elevation of the receiver point is the sum of the terrain elevation at the point and the receiver’s height above the terrain. The receiver point elevation was calculated by using the Add Surface Information function to obtain the terrain elevation at the point, and adding it to the receiver height above surface, which is 1.5 m for all receiver points, and the building height if the
receiver point happens to be located on a building. The height above surface value for all receiver points was selected because it is roughly human ear level (Figure 5-7).

![Receiver Point Height](image)

![Siren Point Height](image)

**Figure 5-7. Siren and Receiver Point Heights.**

The siren elevation was calculated using the same method as the receiver points, but the height above surface is entered by the user, and is dependent on whether the siren point is on a building or not. The height above surface is different for sirens on top of buildings than it is for sirens at ground level in order to satisfy the client’s requirement that the tool account for sirens mounted on towers (Figure 5-7). The variables $h_s$ and $h_r$ are both obtained by adding the height above surface to the building height if the point is on top of a building. If Case 1 is true, the attenuation due to terrain is -3 dB, indicating that sound waves will be strengthened due to terrain. The associated decibel levels for the terrain attenuation are the result of a true/false test, and remains the same regardless of the receiver point’s distance from the siren point.

Case 2 concerns whether the siren point is visible from the receiver point, and the distance of any terrain obstructions from the receiver point. If the line of sight is blocked by a large obstruction very close to (within 5 m of) the receiver point, this obstruction
will affect the total sound pressure more than a similar obstruction that is located halfway between the source and receiver point. Case 2 describes the latter scenario, and consequently decreases the total sound pressure, but only by 2 dB. Case 3 describes the former case, and is true if a large obstruction is very close to (within 5 m of) the receiver point, which would decrease the total sound pressure by 10 dB (Figure 5-8).

Figure 5-8. Terrain Attenuation Case 2 and 3.

To answer whether the line of sight is obstructed, and if it is, how far is the obstruction from the receiver point, the Line of Sight function is run on the sight lines feature class. It evaluates the visibility between each receiver point and siren point while considering obstructions presented by the input DEM. One of the outputs of the Line of Sight tool is an attribute field called TarIsVis, which is created in the sight lines feature class, and stores a binary result indicating if the target is visible or not. For example, a TarIsVis value of 0 would indicate that there is not a clear line of sight between a source point and receiver point because it is obstructed. Another output of the tool is a point feature class that identifies the location of the first obstruction from the observer point. Case 2 is true if there is an obstruction more than 5m from the receiver point, so it is necessary to enter the receiver point as the observer point when first creating the sight lines feature class. The resulting point feature class of the first obstruction locations enables the tool to analyze each obstruction’s proximity to its associated receiver point. To find the distance between receiver point and obstacle, the tool generates a line between each receiver point and its associated first obstruction. The three-dimensional length of this line is then used to populate an attribute field in the sight lines layer. Case 2
is determined to be true for any given sight line if the TarIsVis field is 0, meaning the
target is not visible, and the distance from the receiver point to the first obstacle is greater
than 5m. If this is the case, then the attenuation due to terrain is 2dB. Case 3 is
determined to be true if the TarIsVis field is 0 and the distance from the receiver point to
the first obstacle is 5 m or less. If Case 3 is true, the attenuation due to terrain is 10 dB.

Bass, Bullmore, and Sloth (1998) note that if any doubt exists regarding which
condition should be selected, then the default value of 2dB should be given. This is
implemented in the script with an “else” statement that assigns the attenuation to 2dB if
the target point is not visible, but the tests for both Case 2 and Case 3 fail.

An important note about the model interpretation is that the authors stated that Case
2 is true if the “direct line of sight between the receiver and the source is just interrupted,
or the interruption occurs due to a natural terrain feature that does not provide a distinct
and pronounced interruption to the direct path and does not lie within 5m of the receiver”
that an interruption of at least 0.5m (measured perpendicularly to the sight line) is
suggested as significant. This is a relatively small size compared to the scale of the
analysis, and it was concluded that any obstruction large enough to be represented in the
input DEM or buildings layer would most likely be of the scale that would warrant
classification as distinct and pronounced. If an obstruction is significant, but is more than
5 m from the receiver point, the obstruction would not test true for Case 3, so by default
be assigned to Case 2.

5.2.4 Attenuation Due to Wind

This tool finds the worst case scenario for the given wind speed and weather condition.
This means that the wind attenuation for each receiver point is calculated as if the wind
was blowing directly against the direction of the sound propagation. Calculating the wind
attenuation using the CONCAWE model involves three steps and utilizes two user inputs
values selected from dropdown menus: the wind speed and the general weather condition.

The first step uses these two input values to look up the Pasquill Stability Category
in Table 5-2. This table was implemented as a Python dictionary hardcoded into the tool,
with the two user inputs used as the keys, and the Pasquill Stability Category as the return
value.

Table 5-2. Pasquill Stability Category.

<table>
<thead>
<tr>
<th>Wind Speed m/s*</th>
<th>Day Time Incoming Solar Radiation mW/cm²</th>
<th>Night Time Cloud Cover (octas)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&gt; 60</td>
<td>30 - 60</td>
</tr>
<tr>
<td>≤ 1.5</td>
<td>A</td>
<td>A-B</td>
</tr>
<tr>
<td>2.0 - 2.5</td>
<td>A-B</td>
<td>B</td>
</tr>
<tr>
<td>3.0 - 4.5</td>
<td>B</td>
<td>B-C</td>
</tr>
</tbody>
</table>
The second step in the CONCAWE model uses the Pasquill Stability Category and the wind speed to look up the Meteorological Category from Table 5-3. This table was also implemented as a hardcoded Python dictionary, and uses the value obtained from the previous step and the user entered wind speed as keys.

### Table 5-3. Meteorological Category.

<table>
<thead>
<tr>
<th>Meteorological Category</th>
<th>Pasquill Stability Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A, B</td>
</tr>
<tr>
<td>1</td>
<td>v &lt; -3.0</td>
</tr>
<tr>
<td>2</td>
<td>-3.0 &lt; v &lt; -0.5</td>
</tr>
<tr>
<td>3</td>
<td>-0.5 &lt; v ≤ 0</td>
</tr>
<tr>
<td>4*</td>
<td>-0.5 ≤ 0</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

* Category with assumed zero meteorological influence

The original table includes both negative and positive vector wind speeds. Negative wind speeds indicate that the wind direction is the opposite of the sound propagation direction, blowing the sound back toward the siren. Positive wind speed values indicate that the wind is travelling in the same direction as the sound propagation. This tool examines the worst case scenario for a given wind speed, and since the negative wind values result in a much higher attenuation value, the negative wind speeds were the only values used in the analysis.

In the final step, the Meteorological Category determines which equation will be used to calculate the final attenuation due to wind. The CONCAWE model provides an option to use either equations that are dependent on frequency or equations that are frequency independent. The frequency levels listed in the frequency-dependent equations were not the same as the frequencies used in calculating the atmospheric absorption attenuation factor, so the more simplified, frequency-independent equations were implemented. All of the equations, which can be found in Appendix A, are a function of distance in meters between the receiver point and source point, except for Meteorological Category 4, which results in an attenuation of 0 dB. This distance between source and receiver point has already been calculated for previous steps in the analysis, and is contained in the distance3D_m attribute field in the sight lines feature class.
5.2.5 Finding Siren Coverage Area Polygons

All attenuation factors are subtracted from the siren sound power to find the total sound pressure for each sight line. In order to find the coverage area of each siren, the tool loops through the sight lines layer. For each siren number, the tool joins the total sound pressure field from the sight lines layer to the receiver point layer. The minimum coverage decibel level user input is then used to select the receiver points whose total sound pressure attribute values satisfy the user-input minimum coverage decibel level. Finally, the selected receiver points are buffered by $\sqrt{2}/2$ times the receiver grid resolution. This buffer distance will ensure that if two receiver points are considered covered, and are located diagonally from each other on the grid, the buffer for each of them will touch each other, creating a continuous coverage area polygon. The parameters for the buffer tool are set to dissolve the buffered points into one feature class. This process of joining, selecting, and buffering is repeated for each siren, resulting in a coverage area polygon for each potential siren.

Setting the buffer distance to three quarters of the grid resolution creates a continuous polygon when buffering adjacent receiver points, but will still show gaps in coverage if one point does not meet the decibel requirement and is surrounded by points that do. See Figure 5-9 for an example of potential siren coverage areas.

![Figure 5-9. Example Potential Siren Coverage Areas.](image-url)
5.3 Optimal Siren Configurations

Once the potential siren points and their coverage area polygons have been generated, the tool uses Rana’s Ranking and Overlap Elimination (ROPE) optimization analysis to reduce the potential sirens to the optimal set of sirens. This method selects the siren with the largest coverage area, eliminates any potential siren points that are within that coverage area, and then repeats this process with the siren with the next largest coverage area until all of the potential siren points have been either selected or eliminated.

This is implemented in the Python script by making the potential siren points and their associated coverage areas into feature layers. The tool loops through the coverage area polygons with a search cursor, selects the siren with the largest coverage area polygon, and writes its siren number to a variable. The selected siren number’s row is then appended to a table of selected sirens, and any siren points that intersect with its coverage area are deleted from the feature layer. This deletion is carried through to the coverage area polygon layer by joining the potential coverage polygon feature layer to the potential siren point feature layer; any rows that do not successfully join are deleted. This entire process is contained in a while loop that is set to continue while the number of available sirens is greater than zero. The tool then uses the table of selected sirens and the original potential siren feature classes to create output feature classes of the recommended siren points and their coverage areas.

In summary, the Find Recommended Siren Locations tool generates the potential siren points and their associated coverage areas using the methods described in Section 5.2, and then applies the ROPE method to select the recommended sirens. The output of the Find Recommended Siren Locations tool consists of two feature classes: the suggested siren points, and their associated coverage area polygons. See Figure 5-10 for a summary of the inputs, methods, and outputs of the Find Recommended Siren Locations Tool.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Find Recommended Siren Locations Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data</td>
<td><strong>Non-Spatial Data</strong></td>
</tr>
<tr>
<td></td>
<td>Ambient Temperature</td>
</tr>
<tr>
<td></td>
<td>Ambient Humidity</td>
</tr>
<tr>
<td></td>
<td>Siren Frequency</td>
</tr>
<tr>
<td></td>
<td>Siren Height Above Terrain</td>
</tr>
<tr>
<td></td>
<td>Siren Height Above Buildings</td>
</tr>
<tr>
<td></td>
<td>Distance Between Receiver Points</td>
</tr>
<tr>
<td></td>
<td>Distance Between Potential Siren Points</td>
</tr>
<tr>
<td></td>
<td>Minimum Coverage Decibel Level</td>
</tr>
<tr>
<td></td>
<td>Wind Speed</td>
</tr>
<tr>
<td></td>
<td>General Weather Condition</td>
</tr>
<tr>
<td></td>
<td><strong>Vector Data</strong></td>
</tr>
<tr>
<td></td>
<td>Study Area Polygon</td>
</tr>
<tr>
<td></td>
<td>Building Polygons (with height attributes)</td>
</tr>
<tr>
<td></td>
<td><strong>Raster Data</strong></td>
</tr>
<tr>
<td></td>
<td>DEM</td>
</tr>
<tr>
<td>Methods</td>
<td>Generate Receiver Grid</td>
</tr>
<tr>
<td></td>
<td>Generate Siren Point Grid</td>
</tr>
<tr>
<td></td>
<td>Generate Sight Lines</td>
</tr>
<tr>
<td></td>
<td>Calculate Attenuation for:</td>
</tr>
<tr>
<td></td>
<td>• Geometric Divergence</td>
</tr>
<tr>
<td></td>
<td>• Atmospheric Absorption</td>
</tr>
<tr>
<td></td>
<td>• Terrain</td>
</tr>
<tr>
<td></td>
<td>• Upwind</td>
</tr>
<tr>
<td></td>
<td>Calculate Total Sound Pressure</td>
</tr>
<tr>
<td></td>
<td>Generate Potential Siren Coverage Areas</td>
</tr>
<tr>
<td></td>
<td>Find Recommended Sirens with ROPE analysis</td>
</tr>
<tr>
<td>Output Data</td>
<td>Recommended Siren Location Points</td>
</tr>
<tr>
<td></td>
<td>Recommended Siren Location Coverage Areas</td>
</tr>
</tbody>
</table>

Figure 5-10. Find Recommended Siren Locations Tool Structure.
5.4 Existing Siren Coverage

If there are existing sirens in the study area, their coverage should be accounted for when analyzing the optimal siren locations. To do this, the Update Study Area tool was developed to calculate the coverage area of the existing sirens. It uses the methods described in Section 5.2 to generate the coverage areas of the existing sirens. The output of this tool consists of two feature classes: an updated study area polygon that has the existing coverage area removed and the existing siren coverage area. Both of these feature classes can be used as input for the Find Recommended Siren Locations tool. Figure 5-11 provides a summary of the inputs, methods, and outputs of the Update Study Area Tool.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Update Study Area Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data</strong></td>
<td>Non-spatial Data</td>
</tr>
<tr>
<td></td>
<td>Ambient Temperature</td>
</tr>
<tr>
<td></td>
<td>Ambient Humidity</td>
</tr>
<tr>
<td></td>
<td>Siren Frequency</td>
</tr>
<tr>
<td></td>
<td>Siren Height Above Terrain</td>
</tr>
<tr>
<td></td>
<td>Siren Height Above Buildings</td>
</tr>
<tr>
<td></td>
<td>Distance Between Receiver Points</td>
</tr>
<tr>
<td></td>
<td>Minimum Coverage Decibel Level</td>
</tr>
<tr>
<td></td>
<td>Wind Speed</td>
</tr>
<tr>
<td></td>
<td>General Weather Condition</td>
</tr>
<tr>
<td><strong>Vector Data</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vector Data</td>
</tr>
<tr>
<td></td>
<td>Study Area Polygon</td>
</tr>
<tr>
<td></td>
<td>Building Polygons (with height attributes)</td>
</tr>
<tr>
<td></td>
<td>Existing Sirens Points</td>
</tr>
<tr>
<td><strong>Raster Data</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEM</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Generate Receiver Grid</td>
</tr>
<tr>
<td></td>
<td>Generate Sight Lines</td>
</tr>
<tr>
<td></td>
<td>Calculate Attenuation for:</td>
</tr>
<tr>
<td></td>
<td>• Geometric Divergence</td>
</tr>
<tr>
<td></td>
<td>• Atmospheric Absorption</td>
</tr>
<tr>
<td></td>
<td>• Terrain</td>
</tr>
<tr>
<td></td>
<td>• Upwind</td>
</tr>
<tr>
<td></td>
<td>Calculate Total Sound Pressure</td>
</tr>
<tr>
<td></td>
<td>Generate Existing Siren Coverage Areas</td>
</tr>
<tr>
<td></td>
<td>Subtract Existing Coverage Area from Study Area</td>
</tr>
<tr>
<td><strong>Output Data</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Updated Study Area</td>
</tr>
<tr>
<td></td>
<td>Existing Siren Coverage Area</td>
</tr>
</tbody>
</table>

*Figure 5-11. Update Study Area Tool Structure.*
5.5 Summary

The sound propagation engineering models selected for this analysis were the Joule Report, and the CONCAWE model, and the optimization analysis selected was Rana’s ROPE method. These analyses were implemented in two Python script tools by generating a receiver grid, generating a siren point grid, creating a sight lines feature class that connects each receiver point with each siren point, populating the sight lines with all of the values necessary to calculate the attenuation factors, the attenuation factors, and the resulting total sound pressure. Then the tool generates a coverage area for each siren based on its total sound pressure attribute values. These coverage areas are used in Rana’s ROPE analysis, which provides the final result of recommended siren points and their associated coverage areas.
Chapter 6 – Results and Analysis

This chapter reviews and discusses the results of the project. All of the functional and non-functional requirements were met except for the wind direction input. Analyzing the worst case scenario makes the wind direction an unnecessary input. It is not a likely scenario that the wind will always blow from one direction, so finding the best siren locations for this scenario is not as useful as finding the best siren locations that suit the worst case scenario. Section 6.1 discusses the validation of the sound propagation prediction, Section 6.2 discusses the optimization analysis, Section 6.3 discusses various use cases for the tool, and then these sections are summarized in Section 6.4.

6.1 Sound Propagation Prediction Accuracy Assessment

The sound propagation model used in this analysis is the ISO-9613-2 model with the adjustments recommended in the Joule Report and the CONCAWE model. Bass et al. (1998) reported in the Joule Report that the prediction of ISO-9613-2 model is accurate within 2dB with an 85% confidence level. Manning (1981) reports that the CONCAWE model has an accuracy between 4.5 and 6.9 dB (depending on the meteorological category) with a 95% confidence level. Although only the upwind attenuation factor was considered in this project, these accuracy standards are still a useful gauge for the accuracy of the model developed in this project. There are several approaches that could be taken to further test the validity of this model. One approach is to compare the model’s predictions to a field measurements. Since field measurements are difficult to obtain, an alternative approach is to compare the model’s predictions to the predictions from the commonly used methods by sound propagation professionals. Such prediction methods include using commercial software like SoundPlan and CadnaA, and the manufacturer-specified coverage ranges that are usually field-validated by the manufacturer. Due to the software and data access constraints, the manufacturer-specified coverage ranges were used as a validation reference. Comparing the predicted coverage areas to manufacturer recommendations allowed the project team to determine if the predicted coverage area is a reasonable prediction.

Two siren manufacturers were selected for comparison: E2S Warning Signals and STAHL/Electromach. The manufacturers did not specify any environmental conditions in their recommendations, and both manufacturers indicated that their sirens produce multiple frequencies. A frequency, that was included in both of the manufacturer listings, 1250 Hz, was used to generate the results for 50 siren points in both high (100%) and low (10%) humidity conditions. These environmental conditions were chosen to provide a sense of how much the conditions affect the sound propagation. The manufacturers did not make any mention of terrain, and the terrain used in the Python tool was relatively flat with no complex topography. The minimum and maximum coverage area from these 50 points are compared to the manufacturer recommended areas in Table 6-1. The majority of the results are similar (36% difference or less) to the manufacturer specified ranges, although the high humidity scenario (Scenario 2) suggests a larger maximum area. The manufacturer’s specifications align best with the environmental settings that are least conducive to sound propagation. This is most likely because siren manufacturers
cannot know the environmental settings in which their equipment will be used, so they use the most conservative estimates. The results of this comparison suggest that the tool results are roughly consistent with what the manufacturers recommend but have the capacity to test different situations, instead being constrained to one range that must apply to all situations.

Table 6-1. Tool Results vs. Manufacturer Specified Ranges.

<table>
<thead>
<tr>
<th></th>
<th>Python Script Tool Results – Recommended Coverage Area Minimum (m²)</th>
<th>E2S Warning Signals Recommended Coverage Area Minimum (m²)</th>
<th>Python Script Tool Results - Recommended Coverage Area Maximum (m²)</th>
<th>E2S Warning Signals Recommended Coverage Area Maximum (m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1 (50 degrees C, 10% humidity, 140 dB siren power, 1250 Hz)</td>
<td>587,747</td>
<td>503,000</td>
<td>1,116,942</td>
<td>1,540,000</td>
</tr>
<tr>
<td>Scenario 2 (50 degrees C, 100% humidity, 140 dB siren power, 1250 Hz)</td>
<td>580,474</td>
<td>503,000</td>
<td>3,209,639</td>
<td>1,540,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Python Script Tool Results – Recommended Coverage Area Minimum (m²)</th>
<th>STAHL/Electromach Recommended Coverage Area Minimum (m²)</th>
<th>Python Script Tool Results - Recommended Coverage Area Maximum (m²)</th>
<th>STAHL/Electromach Recommended Coverage Area Maximum (m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1 (50 degrees C, 10% humidity, 140 dB siren power, 1250 Hz)</td>
<td>587,747</td>
<td>785,000</td>
<td>1,116,942</td>
<td>1,770,000</td>
</tr>
<tr>
<td>Scenario 2 (50 degrees C, 100% humidity, 140 dB siren power, 1250 Hz)</td>
<td>580,474</td>
<td>785,000</td>
<td>3,209,639</td>
<td>1,770,000</td>
</tr>
</tbody>
</table>

6.2 Optimization Analysis

The client requested a tool that would recommend siren locations in order to cover a study area. The Rank and Overlap Elimination (ROPE) method meets this requirement and also minimizes processing time. The ROPE method eliminates siren points from the
set of potential sirens by selecting the siren with the largest coverage area and eliminating any other potential siren points that are within that area and then repeating this process until all potential siren points are selected or eliminated. This produces a minimal set of siren points that cover the study area, but there is no guarantee that this set is the smallest set possible. It is possible to solve that question by performing an exhaustive S-ROPE analysis, which iteratively repeats the ROPE analysis using each potential siren point as the first selected point each time. However, the S-ROPE method exponentially increases processing time and provides only minimally better results. To demonstrate this, the project team repeated the ROPE method ten times on a set of 52 potential siren coverage areas with a random siren selected as the first siren each time. The histogram in Figure 6-1 illustrates the results of the test. The ROPE method recommended a set of six sirens, and only one of the ten S-ROPE iterations produced a recommended set that had fewer than six sirens (five). This shows that, according to the ten repetitions considered, the ROPE method has a good chance of producing a near optimal solution.

Figure 6-1. Recommended Sirens Using S-ROPE Method.

6.3 Use Cases

The two tools created in this project can be used in several ways. The original project goal was to create a tool that could recommend siren locations that will completely cover
an area, but some of the intermediate data could also be useful in itself. First, the tool interface will be illustrated, followed by the use case of the Update Project Area Tool where users can gain insight on how different environmental settings influence sound propagation. Finally, an example of how to use the main tool, Find Recommended Siren Locations, to find the recommended siren locations to cover an entire study area will be discussed.

6.3.1 User Interface

Both tools consist of Python scripts contained in an ArcGIS Toolbox. The interface is a standard ArcGIS Desktop tool interface, and is fully documented so the user is guided through the entry of each input. For the Find Recommended Siren Locations tool, the user will be prompted to enter the siren sound power in decibels, a study area polygon, a workspace where output will be saved, the ambient temperature, the ambient humidity, the siren frequency, the DEM path, the DEM unit, building polygons with height attributes, the building height attribute units, the distance between receiver points (in meters), the distance between potential siren points (in meters), the minimum coverage decibel level, the wind speed range, and the general weather conditions. The Find Recommended Siren Locations tool interface is shown in Figure 6-2.
The Update Study Area tool input will be the same as the input for the Find Recommended Siren Locations tool, except it requires a point feature class for existing sirens, and does not require a potential siren point grid resolution. The existing siren point feature class must have a height above surface attribute field (in meters). Update Study Area tool interface is shown in Figure 6-3.
The tools also provide feedback during tool runtime, so that the user knows when each step of the analysis is running and completed. The tool runtime varies depending on the size of the study area and the resolution of the receiver and siren grids. Table 6-2 includes the tool runtimes for various test scenarios, which illustrates that runtime increases significantly when the grid resolution of receivers increases in a larger study area.
### Table 6-2. Python Script Tool Runtimes

<table>
<thead>
<tr>
<th>Study Area Diameter (Circular Study Area)</th>
<th>Receiver Point Resolution (meters)</th>
<th>Siren Point Resolution (meters)</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5 mile</td>
<td>200</td>
<td>200</td>
<td>21.31 seconds</td>
</tr>
<tr>
<td>.5 mile</td>
<td>100</td>
<td>200</td>
<td>25.73 seconds</td>
</tr>
<tr>
<td>1 mile</td>
<td>200</td>
<td>200</td>
<td>3 min 38 seconds</td>
</tr>
<tr>
<td>1 mile</td>
<td>100</td>
<td>200</td>
<td>9 min 59 seconds</td>
</tr>
</tbody>
</table>

#### 6.3.2 Examine Effects of Environment on Siren Sound Coverage

The Update Study Area tool is not only used to prepare an updated study area for the Find Recommended Siren Points tool if there are existing sirens, but also allows the user to see how environmental variations affect the sound propagation of sirens. For example, a siren planner may want to know how complex terrain like the deep valleys and high ridgelines shown in Figure 6-4 might affect a siren’s sound if it is located on the side of a steep slope.

![Figure 6-4. Siren Point in an Environment with Complex Terrain](image)

The effect of the complex terrain is automatically calculated by the Update Study Area tool and shown in a study area in a mountainous area in southern Oregon. Figure 6-5 illustrates the process that tool follows to apply the affects due to terrain. Figure 6-5 shows the siren’s total sound pressures without the terrain attenuation. Geometric divergence and the atmospheric condition are the only factors being considered in Figure 6-5a, and the sound propagates in a regular, circular pattern. Figure 6-5d shows that once
the terrain attenuation is added, the siren sound propagates farther across the river valley to the East of the siren, and travels a shorter distance up the steep hill to the West.

Figure 6-5. Terrain Attenuation
Figure 6-5b and Figure 6-5c illustrate the terrain attenuation associated with each receiver point. As described in Chapter 5, terrain will affect sound travel under three different cases (only two are present in Figure 6-5). Case 1 occurs when the average elevation of the line of sight between the siren and receiver point is 1.5 times larger than the average terrain elevation under that line. In other words, Case 1 is true if the average height of the siren and receiver point are significantly higher than the average terrain. In this example, Case 1 is shown in pink in Figure 6-5b. Case 1 is true for most of the receiver points that are located across the river valley because there is a drop in terrain between the two points. This results in a 3 dB increase in the sound pressure level at those locations.

Case 2 occurs if the line of sight between the siren and receiver point is obstructed and the obstruction is at least five meters away from the receiver point. In rough terrain, line-of-sight obstructions are relatively common, and consequently a large proportion of the receiver points test true for Case 2 (shown in blue), resulting in a 2 dB decrease at those locations. Figure 6-5c shows that Case 2 is not as prevalent just to the east of the deepest part of the river valley. This is because there is an unobstructed line of sight between this area and the siren. Figure 6-5b and Figure 6-5c also show that it is possible for both Case 1 and Case 2 to be true, and in this event, both attenuation factors are applied, resulting in an overall 1 dB increase in sound pressure level at the receiver point.

The tool streamlines the computation process shown in Figure 6-5 by prompting the user for a minimum coverage decibel level, which allows the tool to convert the continuous sound pressure data to binary data that indicates which areas are covered and which areas are not. Figure 6-6 shows the area that the siren covers when the minimum coverage area decibel level is set at 70 dB.
Another example of testing the effect of environment on sound propagation is if a siren planner wants to know whether a siren’s coverage area will be larger on a cold day or a hot day, he or she could load a siren point into the Update Study Area Tool, and run the analysis once with an ambient temperature input of -20°C (-4°F) and once with an ambient temperature input of 50°C (122°F). As Figure 6-7 illustrates, the siren would cover a larger area on a cold day. The ability to test how different environmental conditions affect siren sound propagation will help the users better understand their current siren network, which will also help in planning future siren networks.

![Figure 6-7. Effect of Temperature on Siren Sound Propagation](image)

6.3.3 Finding Complete Coverage of a Study Area

If a siren network planner has an existing siren in a study area and wants to know where to place new sirens to completely cover the study area, he or she could leverage the core functionality of the project, which takes advantage of both the Update Coverage Area tool and the Find Recommended Siren Locations tool. First, the planner would input the study area, existing siren location points, DEM, building polygons, and environmental
parameters into the Update Coverage Area tool (Figure 6-8a). The resulting output from this step would be the existing siren coverage area and the updated study area, as shown in Figure 6-8b. The updated study area is then input into the Find Recommended Siren Locations tool along with all of the desired potential siren properties and environmental variables, and the final output is the recommended siren locations and their associated coverage areas (Figure 6-8c), which when combined with the existing siren coverage area, completely cover the study area.
Figure 6-8. Finding Siren Locations That Completely Cover a Coverage Area

In this example, four new sirens are recommended for a full coverage of the entire study area. However, Siren 27 may not be necessary in practice as there may not be residents living along the edge of the study area.
6.4 Summary

This project produced two python tools that can be used in conjunction with one another to assist siren network planners with understanding how environmental variables and siren properties affect sound propagation, and to recommend siren locations to ensure complete coverage of the study area. This tool provides a sound propagation engineering model implemented in a GIS, which eliminates the need for siren planners to complete sound propagation model calculations by hand or rely on manufacturer-provided sound ranges. This is provided in the form of a simple Python script tool.
Chapter 7 – Conclusions and Future Work

In this project, the client presented the challenge of creating an ArcGIS Python script tool that would consider environmental variables and siren properties in any study area and recommend siren locations to completely cover the area. This project fulfilled the client’s request by implementing the simplified sound propagation engineering model that was presented in the Joule Report to predict siren sound propagation. This model is empirically-based, which simplified processing, and allowed what is normally an extremely computationally-intensive process to be contained in a Python script tool. The tool also used Rana’s ROPE optimization analysis to generate a set of sirens that completely cover the area. This approach also solved the problem at hand without being as processing-heavy as other optimization analysis methods. These two analytical components are contained in two Python script tools that prompt users for the necessary datasets and parameters. This tool gives siren network planners the ability to quickly test different scenarios by allowing them to repeat the analysis with different parameters.

The tools created in this project can be augmented by further work. There are three opportunities for future work related to this project. The first opportunity was suggested by the client, and consists of creating a web application that allows a user to enter siren locations into a specific area and then enter atmospheric conditions and siren properties such as sound power. The application will then show the sound coverage area. Although this application may not be able to include all the environmental variables as the desktop tool because of processing limitations, this would greatly increase the convenience and user base of the analysis because it does not require a knowledge of GIS.

The second opportunity for future work is creating a visualization of the Python script tool results in Esri CityEngine. This was also suggested by the client, as it would provide an intuitive visual for decision makers who may not be familiar with GIS. This would increase the impact of the analysis results.

Third, there are a vast number of factors that influence sound propagation, and the level of precision can always be increased. Most of these additions and adjustments would not be suitable for the scope of this project. However, there is one addition that could be appropriate to add, and this is a method for considering the blocking and screening effect that buildings have on the sound propagation. This analysis has the potential to become very complicated, and would require consultation with a subject matter expert, but would be a valuable addition to the analysis.

In summary, this project provided the client with two ArcGIS Desktop Python Script tools that provide recommended siren locations in order to completely cover the entire study area. The tools consider environmental variables, existing sirens, and siren properties, and provide the user with increased siren network planning capabilities.
Works Cited


## Appendix A. Formulas

<table>
<thead>
<tr>
<th>Equations</th>
<th>Description/Supporting Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L = L_w + D_c - A$</td>
<td>Receiver point sound pressure = the sound power level in decibels + Directivity correction + attenuation</td>
</tr>
<tr>
<td>$L_w = 10 \log (W / 10^{-12} \ W)$ dB</td>
<td>the sound power level in decibels = $10 \log ([\text{sound power in watts}] / 10^{-12} \ W)$ dB</td>
</tr>
<tr>
<td>$D_c = 0$ dB</td>
<td>&quot;For an omni-directional point source radiating into free space, $D_c = 0&quot;$ - ISO-9613-2</td>
</tr>
<tr>
<td>$A = A_{\text{div}} + A_{\text{atm}} + A_{\text{ter}} + A_{\text{wind}}$</td>
<td>The attenuation coefficient = attenuation from geometrical divergence + attenuation from atmosphere + attenuation from ground + attenuation from barriers + miscellaneous attenuation</td>
</tr>
<tr>
<td>$A_{\text{div}} = [20 \ lg \ (d/1 \ m) + 11]$ dB</td>
<td>The attenuation from geometrical divergence = $[20 \ lg \ ([\text{distance from the source receiver}] / 1 \ m) + 11]$ dB</td>
</tr>
<tr>
<td>$A_{\text{atm}} = \alpha d/1000$</td>
<td>The attenuation from atmospheric absorption = [atmospheric attenuation coefficient (in dB per kilometer)] * the distance from source to receiver (in meters) / 1000</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>atmospheric attenuation coefficient is obtained from a table from ISO-9613-1 that references temperature and humidity</td>
</tr>
<tr>
<td>$A_{\text{ter}} = -3$ dB (if Case 1)</td>
<td>Case 1 is true if line of sight is direct and $h_m \geq 1.5 \ (\text{abs}(h_s-h_m)/2)$</td>
</tr>
<tr>
<td>$h_m = \text{the mean height above the ground} = (\text{total siren height} + \text{total receiver height})/2 - \text{average terrain elevation along sight line}$</td>
<td></td>
</tr>
<tr>
<td>$A_{\text{ter}} = 2$ dB (if Case 2)</td>
<td>Case 2 is true if the line of sight between source and receiver is just interrupted by less than .5 m and there is no significant interruption within 5m of the source or the receiver</td>
</tr>
<tr>
<td>$h_r = \text{the receiver height above local ground level} = \text{total receiver elevation} - \text{receiver terrain elevation}$</td>
<td></td>
</tr>
<tr>
<td>$A_{\text{ter}} = 10$ dB (if Case 3)</td>
<td>Case 3 is true if the line of sight between source and receiver is interrupted by more than .5 m and there is a significant interruption within 5m of the source or the receiver</td>
</tr>
<tr>
<td>$A_{\text{wind}} =$ for Meteorological Category 1 $= -38.9 + 26.4 \ (\log d) - 2.84 (\log d)^2 - 0.234 (\log d)^3$</td>
<td>The Meteorological Category is determined using the tables provided in the CONCAWE report</td>
</tr>
<tr>
<td>$A_{\text{wind}}$ for Meteorological Category 2 $= -114 + 119 \ (\log d) - 39.8 \ (\log d)^2 + 4.43 \ (\log d)^3$</td>
<td>d=distance in meters between receiver point and source point</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>$A_{\text{wind}}$ for Meteorological Category 3 $= -28 + 21.3 \ (\log d) - 3.85 \ (\log d)^2 + 0.0903 \ (\log d)^3$</td>
<td></td>
</tr>
<tr>
<td>$A_{\text{wind}}$ for Meteorological Category 4 $= 0$</td>
<td></td>
</tr>
<tr>
<td>$A_{\text{wind}}$ for Meteorological Category 5/6 $= 8.21 - 1.14 \ (\log d) - 2.87 \ (\log d)^2 + 0.671 \ (\log d)^3$</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B.  Python Code

```
# Find Recommended Siren Locations Tool

import math
import arcpy
from arcpy import env

# Get user input
sirenPower_dB = arcpy.GetParameter(0)
studyArea = arcpy.GetParameter(1)
Workspace = arcpy.GetParameter(2)
AmbientTemperature_Celsius = arcpy.GetParameter(3)
AmbientHumidity = arcpy.GetParameter(4)
frequency = arcpy.GetParameter(5)
DEM = arcpy.GetParameter(6)
DEM_VerticalUnits = arcpy.GetParameter(7)
Buildings = arcpy.GetParameter(8)
Building_height = arcpy.GetParameterAsText(9)
Building_height_units = arcpy.GetParameterAsText(10)
ReceiverGridSpacing_m = arcpy.GetParameterAsText(11)
PotentialSirenGridSpacing_m = arcpy.GetParameterAsText(12)
MinimumdB_forCoverage = arcpy.GetParameterAsText(13)
potentialSirenHeightAboveBuilding_m = arcpy.GetParameterAsText(14)
potentialSirenHeightAboveTerrain_m = arcpy.GetParameterAsText(15)
includeWind = arcpy.GetParameterAsText(16)
windSpeed = arcpy.GetParameterAsText(17)
weatherCondition = arcpy.GetParameterAsText(18)

# Set scratch workspace
arcpy.env.workspace = "in_memory"
arcpy.env.overwriteOutput = True
arcpy.AddMessage("Workspace: " + str(Workspace) + ".")

# Delete any fields or feature classes generated in previous calculations
arcpy.Delete_management("in_memory")
```
# Define functions

# Check out necessary licenses
def checkout_licenses():
    if arcpy.CheckExtension("3D") == "Available":
        arcpy.CheckOutExtension("3D")
    if arcpy.CheckInExtension("Spatial") == "Available":
        arcpy.CheckOutExtension("3D")

def checkIfDEMContainsStudyArea(DEM, studyArea):
    arcpy.AddMessage("Checking whether DEM contains Study Area...")
    DEM_desc = arcpy.Describe(DEM)
    DEM_extent = DEM_desc.extent
    studyArea_desc = arcpy.Describe(studyArea)
    studyArea_extent = studyArea_desc.extent
    if (studyArea_extent.within(DEM_extent)) == False:
        arcpy.AddError("ERROR: Study Area is larger than DEM")
        raise arcpy.ExecuteError

def generateRecPointGrid(AreaOfConcern, GridSpacing):
    arcpy.AddMessage("Generating receiver point grid...")
    arcpy.Buffer_analysis(AreaOfConcern, "AreaOfConcern_BufferforFishnet",
                          str(GridSpacing*4), "", "ROUND", "ALL")
    AreaOfConcern_BufferforFishnet = arcpy.mapping.Layer("AreaOfConcern_BufferforFishnet")
    desc = arcpy.Describe(AreaOfConcern_BufferforFishnet)

    arcpy.CreateFishnet_management("fishnet", str(desc.extent.lowerLeft), str(desc.extent.XMin) + " + str(desc.extent.YMax)
                                          , GridSpacing, GridSpacing, "0", "0", str(desc.extent.upperRight), "LABELS", ", ", ", "POLYLINE")

    arcpy.Buffer_analysis(AreaOfConcern, "AreaOfConcern_BufferforClip",
                          str(GridSpacing), "", "ROUND", "ALL")
    arcpy.Clip_analysis("fishnet_label", "AreaOfConcern_BufferforClip", "receiverPoints")
    arcpy.DefineProjection_management ( "receiverPoints", df_spatial_reference)
#Delete data that is no longer needed
arcpy.Delete_management("fishnet")
arcpy.Delete_management("AreaOfConcern_BufferforFishnet")
arcpy.Delete_management("fishnet_label")
arcpy.Delete_management("AreaOfConcern_BufferforClip")
return "receiverPoints"

def generateSirenPointGrid(AreaOfConcern, GridSpacing):
    arcpy.AddMessage("Generating siren point grid...")
desc = arcpy.Describe(AreaOfConcern)

    arcpy.CreateFishnet_management("fishnet",str(desc.extent.lowerLeft),str(desc.extent.XMin) + " " + str(desc.extent.YMax),GridSpacing,
GridSpacing,"0","0",str(desc.extent.upperRight),"LABELS","#","POLYLINE")

    arcpy.Clip_analysis("fishnet_label",AreaOfConcern,"potentialSirenPoints")
arcpy.DefineProjection_management ("potentialSirenPoints", df_spatial_reference)

    #Delete data that is no longer needed
arcpy.Delete_management("fishnet")
arcpy.Delete_management("fishnet_label")

    extent=desc.extent
    arcpy.env.extent = extent
    return "potentialSirenPoints"

def create_receiver_points_withZ():
    receiverPoints=generateRecPointGrid(studyArea, ReceiverGridSpacing_mapUnits)
arcpy.AddField_management(receiverPoints, "heightAboveSurface_m", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "heightAboveSurface_m", '1.5')
getZValues(receiverPoints, Buildings, Building_height, Building_height_units,
DEM, DEM_VerticalUnits, "heightAboveSurface_m")
giveReceiverZFieldsUniqueNames(receiverPoints)
return receiverPoints

def getZValues(points, Buildings, Building_height, Building_height_units, DEM,
DEM_VerticalUnits, heightAboveSurfaceField):
    arcpy.AddMessage("Calculating Z Values...")
#Add the terrain elevation to the receiver points and the siren points
arcpy.ddd.AddSurfaceInformation(points, DEM, "Z")
arcpy.AddField_management(points, "Point_terrainZ_m", "DOUBLE")

if DEM_VerticalUnits=="FEET":
arcpy.CalculateField_management(points, "Point_terrainZ_m", "float(!Z!)*.3048", "PYTHON")
else:
arcpy.CalculateField_management(points, "Point_terrainZ_m", "float(!Z!)", "PYTHON")

#Calculate siren points total height (in meters)
#Change null point height attributes to 0

codeblock= ""
"def changeNullheightsTo0(heightAboveSurface):
    if heightAboveSurface is None:
        return 0
    else:
        return heightAboveSurface"

arcpy.CalculateField_management(points, heightAboveSurfaceField, "changeNullheightsTo0(!"+heightAboveSurfaceField+"!)", "PYTHON", codeblock)

#Convert vertical units to meters
arcpy.AddField_management(Buildings, "buildingHeight_m", "DOUBLE")
if Building_height_units == "FEET":
arcpy.CalculateField_management(Buildings, "buildingHeight_m", "float(!"+Building_height+"!)*0.3048","PYTHON")
else:
arcpy.CalculateField_management(Buildings, "buildingHeight_m", "float(!"+Building_height+"!)", "PYTHON")

#If Buildings are entered, perform spatial join to add building height field to siren points
if arcpy.Describe(Buildings).file:
arcpy.SpatialJoin_analysis (points, Buildings, "Building_pt_spatial_join", "JOIN_ONE_TO_MANY", "KEEP_ALL", "#", "WITHIN")
arcpy.JoinField_management(points, "OID", "Building_pt_spatial_join", "TARGET_FID", "buildingHeight_m")
arcpy.Delete_management("Building_pt_spatial_join")

# Add field and calculate total siren point height (in meters).
# If buildings are entered, total height = building height + terrain elevation + siren height.
# If no buildings entered, total height = terrain elevation + siren height

arcpy.AddField_management(points, "Total_Point_Elevation_m", "DOUBLE")

codeblock="""def addTerrainZ_to_BuildingZ(buildingHeight_m, Z, heightAboveSurfaceField, sr_metersPerUnit):
    if buildingHeight_m is None:
        Total_Point_Elevation_m= (float(Z)*
        float(sr_metersPerUnit))+float(heightAboveSurfaceField)
        return Total_Point_Elevation_m
    else:
        Total_Point_Elevation_m= float(buildingHeight_m)+(float(Z)*
        float(sr_metersPerUnit))+float(heightAboveSurfaceField)
        return Total_Point_Elevation_m"
"
arcpy.CalculateField_management(points, "Total_Point_Elevation_m",
"addTerrainZ_to_BuildingZ(!buildingHeight_m!, !Z!, !"+heightAboveSurfaceField+"!,
"+sr_metersPerUnit_str+)", "PYTHON", codeblock)

arcpy.AddField_management(points, "totalPointElevation_mapUnit", "DOUBLE")
arcpy.CalculateField_management(points, "totalPointElevation_mapUnit",
"!Total_Point_Elevation_m! / float("+sr_metersPerUnit_str+)")", "PYTHON")


def giveReceiverZFieldsUniqueNames(receiverPoints):
    arcpy.AddField_management(receiverPoints, "Total_Rec_Point_Elevation_m", "DOUBLE")
    arcpy.CalculateField_management(receiverPoints,"Total_Rec_Point_Elevation_m",
"[Total_Point_Elevation_m]")
    arcpy.DeleteField_management(receiverPoints,"Total_Point_Elevation_m")

    arcpy.AddField_management(receiverPoints, "totalRecPointElevation_mapUnit", "DOUBLE")
    arcpy.CalculateField_management(receiverPoints,"totalRecPointElevation_mapUnit", 
"[totalPointElevation_mapUnit]")
    arcpy.DeleteField_management(receiverPoints,"totalPointElevation_mapUnit")

    arcpy.AddField_management(receiverPoints, "Rec_Point_terrainZ_m", "DOUBLE")
```python
arcpy.CalculateField_management(receiverPoints, "Rec_Point_terrainZ_m", "[Point_terrainZ_m]")
arcpy.DeleteField_management(receiverPoints, "Point_terrainZ_m")

def calculateDistanceBetweenPoints(sirenPoints, receiverPoints, sr_metersPerUnit_str):
arcpy.AddMessage("Calculating Distance Between Points...")

# Create lines of sight between each siren and each receiver
arcpy.ddd.ConstructSightLines (receiverPoints, sirenPoints, "sight_lines", "totalRecPointElevation_mapUnit", "totalPointElevation_mapUnit")

# Add field and calculate the 3D length of the sight lines (in meters). This length is the
distance between the associated siren and associated receiver
arcpy.AddField_management("sight_lines", "distance3D_m", "DOUBLE")

codeblock = "def convertDistanceToMeters(length, sr_metersPerUnit_str):
    distance_m=float(length)*float(sr_metersPerUnit_str)
    return distance_m"

arcpy.CalculateField_management("sight_lines", "distance3D_m", 
"convertDistanceToMeters(!shape.length3D!, 
"+sr_metersPerUnit_str+")", "PYTHON_9.3", codeblock)


def calculateGeometricDivergence():
arcpy.AddMessage("Calculating Geometric Divergence...")
arcpy.AddField_management("sight_lines", "a_div", "DOUBLE")

codeblock = "def calculateAdiv(distance):
    aDiv=20* (math.log10 ((float(distance)/1))) + 11
    return aDiv"

arcpy.CalculateField_management("sight_lines", "a_div", 
"calculateAdiv(!distance3D_m!)", "PYTHON_9.3", codeblock)


def calculateAtmosphericAbsorptionCoefficient(temperature, frequency, humidity):
    # Define atmospheric absorption attenuation coefficient dictionary with following format:
    # Atmospheric absorption attenuation coefficient={ambient temperature in Celsius:
    # (frequency in Hz : [humidity in %: atmospheric absorption attenuation coefficient] )
    A_atm_coef_Dict={-20:
        {800:
        1000:
```
1250:
1600:
2000:
{10:2.10, 15:2.80, 20:3.65, 30:5.73, 40:8.24, 50:11.0, 60:13.9, 70:16.6, 80:19.0, 90:21.0, 100:22.4},
2500:
3150:
4000:
-10:
{800:
1000:
1250:
1600:
2000:
2500:
{10:5.54, 15:8.85, 20:12.8, 30:21.8, 40:30.1, 50:35.6, 60:37.8, 70:3.74, 80:35.5, 90:32.9, 100:30.2},
3150:
{10:6.11, 15:9.44, 20:13.5, 30:23.1, 40:32.9, 50:40.9, 60:45.9, 70:47.9, 80:47.5, 90:45.7, 100:43.1},
4000:
{10:7.00, 15:10.3, 20:14.5, 30:24.4, 40:35.4, 50:45.5, 60:53.5, 70:58.5, 80:60.7, 90:60.6, 100:59.0}],
{800:
  {10:12.7, 15:14.7, 20:12.9, 30:8.49, 40:5.93, 50:4.52, 60:3.68, 70:3.16,
   80:2.82, 90:2.59, 100:2.43},
  1000:
     80:4.06, 90:3.66, 100:3.37},
  1250:
     80:6.01, 90:5.34, 100:4.85},
  1600:
    {10:15.9, 15:24.8, 20:29.1, 30:26.4, 40:20.3, 50:15.8, 60:12.7, 70:10.6,
     80:9.07, 90:7.98, 100:7.16},
  2000:
    {10:16.6, 15:27.2, 20:34.6, 30:36.0, 40:29.8, 50:23.8, 60:19.3, 70:16.1,
     80:13.8, 90:12.1, 100:10.8},
  2500:
     80:21.1, 90:18.5, 100:16.5},
  3150:
    {10:18.0, 15:30.9, 20:43.6, 30:58.2, 40:57.7, 50:50.9, 60:43.5, 70:37.3,
     80:32.3, 90:28.3, 100:25.2},
  4000:
    {10:19.0, 15:32.6, 20:47.0, 30:69.0, 40:75.2, 50:71.0, 60:63.3, 70:55.5,
     80:48.8, 90:43.2, 100:38.6}},

10:
{800:
  {10:15.7, 15:10.4, 20:7.27, 30:4.58, 40:3.56, 50:3.13, 60:2.94, 70:2.87,
   80:2.86, 90:2.89, 100:2.93},
  1000:
    {10:21.6, 15:15.4, 20:11.0, 30:6.77, 40:5.07, 50:4.26, 60:3.86, 70:3.66,
     80:3.57, 90:3.54, 100:3.55},
  1250:
     80:4.62, 90:4.48, 100:4.42},
  1600:
    {10:35.5, 15:32.1, 20:24.7, 30:15.5, 40:11.1, 50:8.83, 60:7.52, 70:6.73,
     80:6.23, 90:5.92, 100:5.72},
  2000:
     80:8.76, 90:8.14, 100:7.71},
  2500:
     80:12.7, 90:11.6, 100:10.8},
  3150:
4000:
{10:57.3, 15:84.0, 20:91.5, 30:76.6, 40:59.0, 50:46.7, 60:38.4, 70:32.8, 80:28.7, 90:25.7, 100:23.5}},

20:
{800:
1000:
1250:
1600:
2000:
2500:
3150:
4000:

30:
{800:
{10:6.10, 15:4.98, 20:4.85, 30:5.21, 40:5.60, 50:5.82, 60:5.88, 70:5.81, 80:5.66, 90:5.47, 100:5.25},
1000:
1250:
1600:
2000:
2500:
3150:
4000:
40:
{800:
1000:
1250:
1600:
{10:13.6, 15:11.6, 20:11.6, 30:12.8, 40:14.0, 50:14.9, 60:15.3, 70:15.4, 80:15.2, 90:14.8, 100:14.4},
2000:
2500:
3150:
4000:
50:
{800:
1000:
1250:
A_atm_coef = str(float(A_atm_coef_Dict[(float(temperature))][(float(frequency))][(float(humidity))]))
    return A_atm_coef

def populateSightLinesWith_a_atm(A_atm_coef):
    # Calculate attenuation from atmospheric absorption and add to receiver point attributes
    arcpy.AddField_management("sight_lines", "a_atm", "DOUBLE")
    codeblock = '''def calculate_a_atm( A_atm_coef, distance):
        a_atm = (float(A_atm_coef) * float(distance))/1000
        return a_atm"
    arcpy.CalculateField_management("sight_lines", "a_atm", "calculate_a_atm( "+A_atm_coef+", !distance3D_m! )", "PYTHON_9.3", codeblock)

def assignSiren_Numbers(sirenPoints):
    arcpy.AddMessage("Assigning Siren Numbers...")
    arcpy.AddField_management(sirenPoints, "Siren_Number", "DOUBLE")
    codeblock="""rec=0
def autoIncrement():
    global rec
    pStart = 1
pInterval = 1
if (rec == 0):
    rec = pStart
else:
    rec = rec + pInterval
return rec"

arcpy.CalculateField_management(sirenPoints, "Siren_Number", "autoIncrement()", "PYTHON_9.3", codeblock)

def calculateTerrainAttenuation(sirenPoints, receiverPoints, sirenPoints_heightAboveTerrain, DEM):
    arcpy.AddMessage("Calculating Terrain Attenuation...")
    #Calculate the mean elevation under the lines of sight
    arcpy.AddMessage("Calculating Terrain Attenuation - Adding Surface Information...")
    arcpy.ddd.AddSurfaceInformation("sight_lines", DEM, "Z_MEAN")

    #Add necessary calculation fields to sight lines layer

    #Add siren numbers, terrain elevation, and total elevations to lines of sight layer by
    #joining to siren points.
    arcpy.JoinField_management("sight_lines", "OID_TARGET", sirenPoints, "OID", [['SIREN_NUMBER', "Total_Point_Elevation_m", "Point_terrainZ_m"]])

    #Add receiver point total elevations and receiver point terrain elevations to lines of
    #sight layer by joining to receiver points.
    arcpy.JoinField_management("sight_lines", "OID_OBSERV", receiverPoints, "OID", [['Total_Rec_Point_Elevation_m", "Rec_Point_terrainZ_m",
    "totalRecPointElevation_mapUnit"]])

    #Test to see if sight lines test true for Case 1, then populate field with results.
    #Case 1 is true if the sight line mean height above the ground is greater than or equal to
    1.5((abs(siren height above surface-receiver height above surface))/2)
```python
codeblock=""""""""def
calculateAterr1(Siren_Point_terrainZ_m, Total_Siren_Point_Elevation_m, Total_Rec_Point_Elevation_m, Rec_Point_terrainZ_m, Z_Mean, sr_metersPerUnit):
    hm=((float(Total_Siren_Point_Elevation_m)+float(Total_Rec_Point_Elevation_m))/2)-
    (float(Z_Mean)*float(sr_metersPerUnit))
    hs=float(Total_Siren_Point_Elevation_m)-float(Siren_Point_terrainZ_m)
    hr=float(Total_Rec_Point_Elevation_m)-float(Rec_Point_terrainZ_m)
    if hm >= (1.5*(abs(float(hs)-float(hr))/2)):
        return -3
    else:
        return 0"
"
arcpy.AddField_management("sight_lines", "Aterr_Case1", "DOUBLE")
arcpy.AddMessage("Testing for Terrain Attenuation Case 1...")
arcpy.CalculateField_management("sight_lines", "Aterr_Case1", "calculateAterr1(!Point_terrainZ_m!,!Total_Point_Elevation_m!,!Total_Rec_Point_Elevation_m!,!Rec_Point_terrainZ_m!,!Z_Mean!,"+sr_metersPerUnit_str")", "PYTHON",
codeblock)
arcpy.AddMessage("Calculating Terrain Attenuation - Collecting Data for Case 2...")

# Test to see if sight lines test true for case 2 or 3.
# Case 2 is true if the line of sight is blocked and the obstruction is greater than 5 meters from the receiver
# Case 3 is true if the line of sight is blocked and the obstruction is less than or equal to 5 meters from the receiver
# Runs the line of sight tool. This provides the TarIsVis field, which indicates whether the siren is visible from the receiver points.
# It also generates the first obstruction on the sight line. Lines are created between the receiver point and obstruction so that the (3D) length can be measured.

# Run line of sight tool on sight lines
arcpy.LineOfSight_3d(DEM, "sight_lines", "sight_lines_vis", "sight_line_vis_obs")

# Add the 'target is visible' (TarIsVis) field to sight lines layer. A value of 0 means target is not visible, 1 means target is visible
arcpy.JoinField_management("sight_lines", "OID", "sight_lines_vis", "SourceOID", "TarIsVis")
arcpy.Delete_management("sight_lines_vis")

```
#Calculate the x, y and z values for obstacle points and x and y values for receiver points
arcpy.JoinField_management("sight_line_vis_obs", "SourceOID", "sight_lines", "OID", "OID_OBSERV")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_X", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_X", "!Shape!.centroid.X", "PYTHON_9.3")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_Y", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_Y", "!Shape!.centroid.Y", "PYTHON_9.3")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_Z", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_Z", "!Shape!.centroid.Z", "PYTHON_9.3")
arcpy.AddField_management(receiverPoints, "rec_point_x", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "rec_point_x", "!Shape!.centroid.X", "PYTHON_9.3")
arcpy.AddField_management(receiverPoints, "rec_point_y", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "rec_point_y", "!Shape!.centroid.Y", "PYTHON_9.3")

#Add the associated receiver point x, y and z field to the 'first obstacle on sight line' point feature class
arcpy.JoinField_management("sight_line_vis_obs", "OID_OBSERV", receiverPoints, "OID", ["rec_point_y", "rec_point_x", "Z"])

#Create lines connecting each receiver point to the first obstacle point on the sight line
arcpy.XYToLine_management("sight_line_vis_obs", "Rec_to_obs_lines", "rec_point_x", "rec_point_y", "obs_point_X", "obs_point_Y", "GEODESIC", "SourceOID")

#Add the associated receiver and siren z fields to the 'receiver to first obstacle on sight line' line feature class
arcpy.JoinField_management("Rec_to_obs_lines", "SourceOID", "sight_lines", "OID", ["totalRecPointElevation_mapUnit"])
arcpy.Delete_management("sight_line_vis_obs")

#Use z fields to convert the 'receiver to first obstacle on sight line' to 3d (this is so the line length with reflect height differences between obstacle and receiver)
arcpy.FeatureTo3DByAttribute_3d("Rec_to_obs_lines", "Rec_to_obs_lines_3d", "totalRecPointElevation_mapUnit", "obs_point_Z")
arcpy.Delete_management("Rec_to_obs_lines")

#Add 3d length to 'receiver to first obstacle on sight line'
arcpy.AddZInformation_3d("Rec_to_obs_lines_3d", "LENGTH_3D")

#rename 3d length field to "distance_to_obstacle_3D", and then join 3D length field to
sight lines layer.
arcpy.AddField_management("Rec_to_obs_lines_3d", "distance_to_obstacle_3D", "DOUBLE")
arcpy.CalculateField_management("Rec_to_obs_lines_3d", "distance_to_obstacle_3D", "[Length3D]"

arcpy.DeleteField_management("Rec_to_obs_lines_3d","Length3D")
arcpy.JoinField_management("sight_lines", "OID", "Rec_to_obs_lines_3d", "SourceOID", "distance_to_obstacle_3D")
arcpy.Delete_management("Rec_to_obs_lines_3d")
arcpy.AddField_management("sight_lines", "Aterr_Case2_or_3", "DOUBLE")
codeblock=""""def calculateAterr2_3(sr_metersPerUnit, TarIsVis, distance_to_obstacle_3D):
    if TarIsVis ==1:
        return 0
    elif distance_to_obstacle_3D is None:
        return 2
    elif 0<= (float(distance_to_obstacle_3D)*float(sr_metersPerUnit)) <=5:
        return 10
    else:
        return 2
"""
arcpy.CalculateField_management("sight_lines", "Aterr_Case2_or_3", "calculateAterr2_3( +sr_metersPerUnit_str+, !TarIsVis!, !distance_to_obstacle_3D!)", "PYTHON_9.3", codeblock)

def calculateWindAttenuation(weatherCondition, windSpeed):
arcpy.AddMessage("Calculating Wind Attenuation")

PasquillCategory=PasquillSuitabilityIndex[windSpeed][weatherCondition]

if windSpeed=='<.5 m/s':
    windSpeed_forMetCategoryLookup="0-5"
elif (windSpeed=="0.5 - 1.5 m/s") or (windSpeed=="2.0 - 2.5 m/s"):
    windSpeed_forMetCategoryLookup="0.5-3.0"
elif (windSpeed=="3.0 - 4.5 m/s") or (windSpeed=="5.0 - 6.0 m/s") or (windSpeed==">6.0 m/s"):
    windSpeed_forMetCategoryLookup=">3.0"

meteorologicalCategoryIndex={"A":{">3.0":1, "0.5-3.0":2, "0-5":3},
"B":{">3.0":1, "0.5-3.0":2, "0-5":3},
"C":{">3.0":2, "0.5-3.0":3, "0-5":4},
"D":{">3.0":2, "0.5-3.0":3, "0-5":4},
"E":{">3.0":2, "0.5-3.0":3, "0-5":4},
"F":{">3.0":3, "0.5-3.0":4, "0-5":5}}

meteorologicalCategory=str(meteorologicalCategoryIndex [PasquillCategory] [windSpeed_forMetCategoryLookup])

arcpy.AddField_management("sight_lines", "MeteorologicalCat", "TEXT")
arcpy.CalculateField_management("sight_lines", "MeteorologicalCat", "'"+meteorologicalCategory+ "'", "PYTHON")

arcpy.AddField_management("sight_lines", "Wind_Attenuation", "DOUBLE")

codeblock=""""def calculateWind_Attenuation(meteorologicalCategory, distance):
    if meteorologicalCategory==1:
wind_attenuation=-38.9+(26.4*(math.log10(distance)))-(2.84*((math.log10(distance))**2))-.234 *((math.log10(distance))**3))

elif meteorologicalCategory==2:
    wind_attenuation=-114+(119*(math.log10(distance)))-(39.8*((math.log10(distance))**2))+(4.43 *((math.log10(distance))**3))

elif meteorologicalCategory==3:
    wind_attenuation=-28+(21.3*(math.log10(distance)))-(3.85*((math.log10(distance))**2))+(.0903 *((math.log10(distance))**3))

elif meteorologicalCategory==4:
    wind_attenuation=0

elif meteorologicalCategory==5 or meteorologicalCategory==6:
    wind_attenuation=8.21-(1.14*(math.log10(distance)))-(2.87*((math.log10(distance))**2))+(.671 *((math.log10(distance))**3))

if wind_attenuation<0:
    wind_attenuation=0

return wind_attenuation

arcpy.CalculateField_management("sight_lines", "Wind_Attenuation", "calculateWind_Attenuation( "+meteorologicalCategory+" , !distance3D_m!)", "PYTHON_9.3", codeblock)

def calculateTotalSoundPressure(Lw, includeWind):
    arcpy.AddMessage("Calculating Total Sound Pressure...")
    arcpy.AddField_management("sight_lines", "totalSoundPressure", "DOUBLE")

    if includeWind=="true":
        codeblock= """"def calculateTotalLw( Lw, a_div, a_atm, Aterr_Case1, Aterr_Case2_or_3, Wind_Attenuation):
            total=float(Lw)-a_div-a_atm - Aterr_Case1 - Aterr_Case2_or_3-Wind_Attenuation
            return total"

        arcpy.CalculateField_management("sight_lines", "totalSoundPressure", "calculateTotalLw(" +Lw+ ", !a_div!, !a_atm!, !Aterr_Case1!, !Aterr_Case2_or_3!,
        !Wind_Attenuation!)", "PYTHON", codeblock)

    else:
        codeblock= """"def calculateTotalLw( Lw, a_div, a_atm, Aterr_Case1, Aterr_Case2_or_3):
            total=float(Lw)-a_div-a_atm - Aterr_Case1 - Aterr_Case2_or_3
            return total"

75
arcpy.CalculateField_management("sight_lines", "totalSoundPressure",
"calculateTotalLw(" +Lw+ ", !a_div!, !a_atm!, !Aterr_Case1!, !Aterr_Case2_or_3!)",
"PYTHON", codeblock)

def create_and_populate_height_above_surface_field(potentialSirenPoints, Buildings,
ReceiverGridSpacing_mapUnits, potentialSirenHeightAboveBuilding_m):
    arcpy.AddField_management(potentialSirenPoints, "heightAboveSurface_m")

    potentialSirenPoints_fl=arcpy.MakeFeatureLayer_management(potentialSirenPoints,
"potentialSirenPoints_fl")

    arcpy.SelectLayerByLocation_management (potentialSirenPoints_fl, "INTERSECT",
Buildings)

    arcpy.CalculateField_management(potentialSirenPoints,"heightAboveSurface_m",
"'"+potentialSirenHeightAboveBuilding_m+"'", "PYTHON")

    arcpy.SelectLayerByLocation_management (potentialSirenPoints_fl, ",", ",", ",",
"SWITCH_SELECTION")

    arcpy.CalculateField_management(potentialSirenPoints,"heightAboveSurface_m",
"'"+potentialSirenHeightAboveTerrain_m+"'", "PYTHON")

    arcpy.SelectLayerByAttribute_management(potentialSirenPoints_fl,"CLEAR_SELECTION")

    removelayer = arcpy.mapping.Layer(potentialSirenPoints)

    arcpy.mapping.RemoveLayer(df, removelayer)

def generatePotentialSirenCoverageAreas(potentialSirenPoints,
PotentialRecGridSpacing_mapUnits, df_spatial_reference, studyArea):
    arcpy.AddMessage("Calculating Potential Siren Coverage Areas...")

    potentialSirenCoverageAreas=arcpy.CreateFeatureclass_management("in_memory",
"potentialSirenCoverageAreas", "POLYGON", ",", ",", df_spatial_reference)
    arcpy.AddField_management("potentialSirenCoverageAreas", "Siren_Number",
"DOUBLE")

    #sirens = arcpy.SearchCursor(potentialSirenPoints)
    with arcpy.da.SearchCursor(potentialSirenPoints, ["Siren_Number"]) as cursor:
        for siren in cursor:
            whereClause=str("Siren_Number=" + str(int(siren[0])))

76
arcpy.MakeTableView_management("sight_lines", "sight_lines_forSirenPoint", whereClause)
arcpy.JoinField_management(receiverPoints, "OID", "sight_lines_forSirenPoint", "OID_OBSERV", ["totalSoundPressure", "Siren_Number"])

whereClause=str("""" "totalSoundPressure" >="""" + MinimumdB_forCoverage)
arcpy.Select_analysis(receiverPoints, "selected_receiverPoints", whereClause)
bufferDistance=ReceiverGridSpacing_mapUnits*(math.sqrt(2)/2)
potential_siren_coverage_area=arcpy.Buffer_analysis("selected_receiverPoints", "potential_siren_coverage_area", bufferDistance, "FULL", "ROUND", "LIST", "Siren_Number")
arcpy.Clip_analysis ("potential_siren_coverage_area", studyArea, "potential_area_clipped")

fm = arcpy.FieldMap()
fms = arcpy.FieldMappings()
fm.addInputField(receiverPoints, "Siren_Number")
name = fm.outputField
name.name ="Siren_Number"
fm.outputField = name

arcpy.Append_management("potential_area_clipped", "potentialSirenCoverageAreas", "NO_TEST", fms)
arcpy.DeleteField_management(receiverPoints, ["totalSoundPressure", "Siren_Number"])
arcpy.Delete_management("sight_lines_forSirenPoint")
arcpy.RefreshActiveView()
del cursor

return potentialSirenCoverageAreas

def sirenSelection(potentialSirenCoverageAreas,potentialSirenPoints ,studyArea, df):
arcpy.AddMessage("Selecting Sirens to Recommend...")
arcpy.CopyFeatures_management (potentialSirenCoverageAreas, "availCovAreas")

#Create Feature Layers
availCovAreas_fl=arcpy.MakeFeatureLayer_management("availCovAreas",
"availCovAreas_fl")
arcpy.CopyFeatures_management(potentialSirenPoints, "availableSirenPoints")
arcpy.AddField_management("availableSirenPoints", "JoinIndicator", "DOUBLE")
arcpy.CalculateField_management("availableSirenPoints", "JoinIndicator", "1")
availSPoints_fl=arcpy.MakeFeatureLayer_management("availableSirenPoints",
"availSPoints_fl")

table=arcpy.CreateTable_management("in_memory", "AllSelectedAreas",
availCovAreas_fl)

#Get Count of Available Sirens
arcpy.AddMessage("Getting Siren Count...")
availableSirens=str(arcpy.GetCount_management("availCovAreas"))
availableSirens_count=int(availableSirens)
count=1

#Loop
while (availableSirens_count>0):
    arcpy.AddMessage("Selecting Siren "+ str(count) + "...")
count=count+1
    maxArea=0.0
    highestRankingSiren=0

    with arcpy.da.SearchCursor(availCovAreas_fl, ["SHAPE@AREA",
"Siren_Number"])) as cursor:
        for siren in cursor:
            if float(siren[0])> maxArea:
                maxArea= float(siren[0])
                highestRankingSiren=str(siren[1])
            else:
                maxArea=maxArea
        del cursor

    #Create feature layer for 'selected coverage area' and copy to table
    whereClause=str(""""Siren_Number"="""" +str(highestRankingSiren))
    selectedArea=arcpy.MakeFeatureLayer_management (availCovAreas_fl,
"selectedArea", whereClause)

    arcpy.MakeTableView_management (selectedArea, "selectedAreaTableView")
    arcpy.Append_management("selectedAreaTableView", table, "NO_TEST")

    #Delete siren points that intersect with selected area
arcpy.SelectLayerByLocation_management (availSPoints_fl, "INTERSECT", selectedArea)
arcpy.DeleteRows_management (availSPoints_fl)
arcpy.SelectLayerByAttribute_management (availSPoints_fl, "CLEAR_SELECTION")

#Delete coverage areas associated with deleted siren points
arcpy.JoinField_management (availCovAreas_fl, "Siren_Number", availSPoints_fl, "Siren_Number", "JoinIndicator")
arcpy.SelectLayerByAttribute_management (availCovAreas_fl, "NEW_SELECTION", """"JoinIndicator"=1"""")
arcpy.SelectLayerByAttribute_management (availCovAreas_fl, "SWITCH_SELECTION")
arcpy.DeleteRows_management (availCovAreas_fl)
arcpy.SelectLayerByAttribute_management (availCovAreas_fl, "CLEAR_SELECTION")
arcpy.DeleteField_management (availCovAreas_fl, "JoinIndicator")

layer = arcpy.mapping.Layer("availSPoints_fl")
updateLayer = arcpy.mapping.Layer("availableSirenPoints")
arcpy.mapping.UpdateLayer (df, layer, updateLayer)
availableSirens = str (arcpy.GetCount_management (availSPoints_fl))
availableSirens_count = int (availableSirens)

#Add field to table of selected sirens to indicate successful join to feature layers
arcpy.AddField_management (table, "JoinIndicator", "DOUBLE")
arcpy.CalculateField_management (table, "JoinIndicator", "1")
arcpy.env.workspace = Workspace

#Join table to potential siren coverage areas feature layer, copy to feature class, and add to map for output
potentialSirenCoverageAreas_fl = arcpy.MakeFeatureLayer_management (potentialSirenCoverageAreas, "potentialSirenCoverageAreas_fl")
arcpy.JoinField_management (potentialSirenCoverageAreas_fl, "Siren_Number", table, "Siren_Number", "JoinIndicator")
arcpy.SelectLayerByAttribute_management (potentialSirenCoverageAreas_fl, "NEW_SELECTION", """"JoinIndicator" =1"""")
arcpy.CopyFeatures_management (potentialSirenCoverageAreas_fl, str (str (Workspace) + "/recommendedSirenCoverageAreas"))
arcpy.DeleteField_management (potentialSirenCoverageAreas_fl, "JoinIndicator")
layer = arcpy.mapping.Layer ("recommendedSirenCoverageAreas")
arcpy.mapping.AddLayer (df, layer)
Join table to potential siren points feature layer, copy to feature class, and add to map for output
try:
    arcpy.DeleteField_management(potentialSirenPoints,"JoinIndicator"")
except: # catch *all* exceptions
e = sys.exc_info()[0]
    potentialSirenPoints_fl=arcpy.mapping.Layer("in_memory\potentialSirenPoints")
arcpy.JoinField_management(potentialSirenPoints, "Siren_Number", table, "Siren_Number", "JoinIndicator")
arcpy.SelectLayerByAttribute_management(potentialSirenPoints_fl, "NEW_SELECTION", ""JoinIndicator" =1"")
arcpy.CopyFeatures_management (potentialSirenPoints_fl, str(str(Workspace)+"\recommendedSirenPoints"))
arcpy.Delete_management(potentialSirenPoints)
layer=arcpy.mapping.Layer("recommendedSirenPoints")
arcpy.mapping.AddLayer(df,layer)
arcpy.RefreshActiveView()

#Delete Feature Layers
arcpy.Delete_management(availCovAreas_fl)
arcpy.Delete_management(availSPoints_fl)

#Define global variables
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd)[0]
global df
df_spatial_reference = df.spatialReference
global df_spatial_reference

sr_metersPerUnit_str=str(df_spatial_reference.metersPerUnit)
global sr_metersPerUnit_str

ReceiverGridSpacing_mapUnits=float(ReceiverGridSpacing_m)/float(sr_metersPerUnit_str)
global ReceiverGridSpacing_mapUnits

PotentialSirenGridSpacing_mapUnits=float(PotentialSirenGridSpacing_m)/float(sr_metersPerUnit_str)
global PotentialSirenGridSpacing_mapUnits

#Call functions and assign returns as variables
checkout_licenses()
checkIfDEMContainsStudyArea(DEM, studyArea)
Lw=str(sirenPower_dB)
receiverPoints=create_receiver_points_withZ()

potentialSirenPoints=generateSirenPointGrid(studyArea, PotentialSirenGridSpacing_mapUnits)

create_and_populate_height_above_surface_field(potentialSirenPoints, Buildings, potentialSirenHeightAboveTerrain_m, potentialSirenHeightAboveBuilding_m)

assignSiren_Numbers(potentialSirenPoints)

getzValues(potentialSirenPoints, Buildings, Building_height, Building_height_units, DEM, DEM_VerticalUnits,"heightAboveSurface_m")

calculateDistanceBetweenPoints(potentialSirenPoints,receiverPoints, sr_metersPerUnit_str)

calculateGeometricDivergence()

A_atm_coef=calculateAtmosphericAbsorptionCoefficient(AmbientTemperature_Celsius, frequency, AmbientHumidity)

populateSightLinesWith_a_atm(A_atm_coef)

calculateTerrainAttenuation(potentialSirenPoints, receiverPoints, "heightAboveSurface_m", DEM)

if includeWind=="true":
    calculateWindAttenuation(weatherCondition, windSpeed)

calculateTotalSoundPressure(Lw, includeWind)

potentialSirenCoverageAreas=generatePotentialSirenCoverageAreas(potentialSirenPoints , ReceiverGridSpacing_mapUnits, df_spatial_reference, studyArea)
sirenSelection(potentialSirenCoverageAreas, potentialSirenPoints, studyArea, df)

arcpy.Delete_management("in_memory")

# UpdateStudy Areas Tool

import math
import arcpy
from arcpy import env

# Get user input
sirenPower_dB = arcpy.GetParameter(0)
sirenPoints = arcpy.GetParameter(1)
sirenPoints_heightAboveTerrain = arcpy.GetParameterAsText(2)
studyArea = arcpy.GetParameter(3)
Workspace = arcpy.GetParameter(4)
AmbientTemperature_Celsius = arcpy.GetParameter(5)
AmbientHumidity = arcpy.GetParameter(6)
frequency = arcpy.GetParameter(7)
DEM = arcpy.GetParameter(8)
DEM_VerticalUnits = arcpy.GetParameter(9)
Buildings = arcpy.GetParameter(10)
Building_height = arcpy.GetParameterAsText(11)  # height above terrain
Building_height_units = arcpy.GetParameterAsText(12)
ReceiverGridSpacing_m = arcpy.GetParameterAsText(13)
MinimumdB_forCoverage = arcpy.GetParameterAsText(14)
includeWind = arcpy.GetParameterAsText(15)
windSpeed = arcpy.GetParameterAsText(16)
weatherCondition = arcpy.GetParameterAsText(17)

# Set scratch workspace
arcpy.env.workspace = "in_memory"
arcpy.env.overwriteOutput = True
arcpy.AddMessage("Workspace: " + str(Workspace)+"...")

arcpy.DeleteField_management(sirenPoints, ["Siren_Number", "Z", "Point_terrainZ_m", "buildingHeight_m", "Total_Point_Elevation_m", "totalPointElevation_mapUnit"])
#Delete any fields or feature classes generated in previous calculations

arcpy.DeleteField_management(sirenPoints, ['Siren_Number', 'Z', 'Point_terrainZ_m', 'buildingHeight_m', 'Total_Point_Elevation_m', 'totalPointElevation_mapUnit'])
arcpy.Delete_management('sight_lines')
arcpy.Delete_management('point_receiver_distances')

#Define functions

#Check out necessary licenses
def checkout_licenses():
    if arcpy.CheckExtension('3D') == 'Available':
        arcpy.CheckOutExtension('3D')
    if arcpy.CheckInExtension('Spatial') == 'Available':
        arcpy.CheckOutExtension('3D')

def checkIfDEMContainsStudyArea(DEM, studyArea):
    arcpy.AddMessage('Checking whether DEM contains Study Area...')

    DEM_desc = arcpy.Describe(DEM)
    DEM_extent = DEM_desc.extent

    studyArea_desc = arcpy.Describe(studyArea)
    studyArea_extent = studyArea_desc.extent

    if (studyArea_extent.within(DEM_extent)) == False:
        arcpy.AddError('ERROR: Study Area is larger than DEM')
        raise arcpy.ExecuteError

def generatePointGrid(AreaOfConcern, GridSpacing):
    arcpy.Buffer_analysis(AreaOfConcern, 'AreaOfConcern_Buffer', str(GridSpacing*3), '', 'ROUND', 'ALL')
    AreaOfConcern_Buffer = arcpy.mapping.Layer('AreaOfConcern_Buffer')
    desc = arcpy.Describe(AreaOfConcern_Buffer)
arcpy.CreateFishnet_management("fishnet",str(desc.extent.lowerLeft),str(desc.extent.XMin) + " " + str(desc.extent.YMax),GridSpacing, GridSpacing,"0","0",str(desc.extent.upperRight),"LABELS","#","POLYLINE")

arcpy.Delete_management("fishnet")
arcpy.Clip_analysis("fishnet_label",AreaOfConcern,"pointGrid")
arcpy.DefineProjection_management ("pointGrid", df_spatial_reference)
## arcpy.CopyFeatures_management("pointGrid", str(str(Workspace)+"/recpoints"))

return "pointGrid"

def getZValues(points, Buildings, Building_height, Building_height_units, DEM, DEM_VerticalUnits, heightAboveSurfaceField):
    arcpy.AddMessage("Calculating Z Values...")

    #Add the terrain elevation to both the receiver points and the siren points
    arcpy.ddd.AddSurfaceInformation (points, DEM, "Z")
arcpy.AddField_management(points, "Point_terrainZ_m", "DOUBLE")

    if DEM_VerticalUnits=="FEET":
        arcpy.CalculateField_management(points, "Point_terrainZ_m", "float(!Z!)*.3048", "PYTHON")
    else:
        arcpy.CalculateField_management(points, "Point_terrainZ_m", "float(!Z!)", "PYTHON")

    #Calculate siren points total height (in meters)
    #Change null point height attributes to 0
    codeblock= """"def changeNullheightsTo0(heightAboveSurface):
        if heightAboveSurface is None:
            return 0
        else:
            return heightAboveSurface"
    arcpy.CalculateField_management(points, heightAboveSurfaceField, "changeNullheightsTo0(!"+heightAboveSurfaceField+"!)", "PYTHON", codeblock )

    #Convert vertical units to meters
    arcpy.AddField_management(Buildings, "buildingHeight_m", "DOUBLE")
if Building_height_units == "FEET":
    arcpy.CalculateField_management(Buildings, "buildingHeight_m",
    "float(!"+Building_height+"!)*0.3048","PYTHON")
else:
    arcpy.CalculateField_management(Buildings, "buildingHeight_m",
    "float(!"+Building_height+"!)", "PYTHON")

#If Buildings are entered, perform spatial join to add building height field to siren points
if arcpy.Describe(Buildings).file:
    arcpy.SpatialJoin_analysis (points, Buildings, "Building_pt_spatial_join",
    "JOIN_ONE_TO_MANY", "KEEP_ALL", "#", "WITHIN")

##fc = r"C:\temp\test.shp"
desc = arcpy.Describe(points)
flds = desc.fields
fldin = 'no'
for fld in flds:
    if fld.name == 'OID':
        fldin = 'yes'

if fldin == 'yes':
    arcpy.JoinField_management(points, "OID", "Building_pt_spatial_join",
    "TARGET_FID", "buildingHeight_m")
else:
    arcpy.JoinField_management(points, "OBJECTID", "Building_pt_spatial_join",
    "TARGET_FID", "buildingHeight_m")

arcpy.Delete_management ("Building_pt_spatial_join")

#Add field and calculate total siren point height (in meters).
#If buildings are entered, total height = building height + terrain elevation + siren height.
#If no buildings entered, total height = terrain elevation + siren height
arcpy.AddField_management(points, "Total_Point_Elevation_m", "DOUBLE")

codeblock= """"def addTerrainZ_to_BuildingZ(buildingHeight_m, Z,
    heightAboveSurfaceField, sr_metersPerUnit):

if Building_height_units == "FEET":
    arcpy.CalculateField_management(Buildings, "buildingHeight_m",
    "float(!"+Building_height+"!)*0.3048","PYTHON")
else:
    arcpy.CalculateField_management(Buildings, "buildingHeight_m",
    "float(!"+Building_height+"!)", "PYTHON")

#If Buildings are entered, perform spatial join to add building height field to siren points
if arcpy.Describe(Buildings).file:
    arcpy.SpatialJoin_analysis (points, Buildings, "Building_pt_spatial_join",
    "JOIN_ONE_TO_MANY", "KEEP_ALL", "#", "WITHIN")

##fc = r"C:\temp\test.shp"
desc = arcpy.Describe(points)
flds = desc.fields
fldin = 'no'
for fld in flds:
    if fld.name == 'OID':
        fldin = 'yes'

if fldin == 'yes':
    arcpy.JoinField_management(points, "OID", "Building_pt_spatial_join",
    "TARGET_FID", "buildingHeight_m")
else:
    arcpy.JoinField_management(points, "OBJECTID", "Building_pt_spatial_join",
    "TARGET_FID", "buildingHeight_m")

arcpy.Delete_management ("Building_pt_spatial_join")

#Add field and calculate total siren point height (in meters).
#If buildings are entered, total height = building height + terrain elevation + siren height.
#If no buildings entered, total height = terrain elevation + siren height
arcpy.AddField_management(points, "Total_Point_Elevation_m", "DOUBLE")

codeblock= """"def addTerrainZ_to_BuildingZ(buildingHeight_m, Z,
    heightAboveSurfaceField, sr_metersPerUnit):
if buildingHeight_m is None:
    Total_Point_Elevation_m = (float(Z) * float(sr_metersPerUnit)) + float(heightAboveSurfaceField)
    return Total_Point_Elevation_m

else:
    Total_Point_Elevation_m = float(buildingHeight_m) + (float(Z) * float(sr_metersPerUnit)) + float(heightAboveSurfaceField)
    return Total_Point_Elevation_m

arcpy.CalculateField_management(points, "Total_Point_Elevation_m", "addTerrainZ_to_BuildingZ(!buildingHeight_m!, !Z!, !"+heightAboveSurfaceField+"!, "+sr_metersPerUnit_str+)", "PYTHON", codeblock)

arcpy.AddField_management(points, "totalPointElevation_mapUnit", "DOUBLE")
arcpy.CalculateField_management(points, "totalPointElevation_mapUnit", "!Total_Point_Elevation_m! / float("+sr_metersPerUnit_str")", "PYTHON")

def giveReceiverZFieldsUniqueNames(receiverPoints):
    arcpy.AddField_management(receiverPoints, "Total_Rec_Point_Elevation_m", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "Total_Rec_Point_Elevation_m", 
                                "[Total_Point_Elevation_m]"
    arcpy.DeleteField_management(receiverPoints, "Total_Point_Elevation_m")

    arcpy.AddField_management(receiverPoints, "totalRecPointElevation_mapUnit", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "totalRecPointElevation_mapUnit", 
                                "[totalPointElevation_mapUnit]"
    arcpy.DeleteField_management(receiverPoints, "totalPointElevation_mapUnit")

    arcpy.AddField_management(receiverPoints, "Rec_Point_terrainZ_m", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "Rec_Point_terrainZ_m", 
                                "[Point_terrainZ_m]"
    arcpy.DeleteField_management(receiverPoints, "Point_terrainZ_m")

def calculateDistanceBetweenPoints(sirenPoints, receiverPoints, sr_metersPerUnit_str):
    arcpy.AddMessage("Calculating Distance Between Points...")

    # Create lines of sight between each siren and each receiver
    arcpy.ddd.ConstructSightLines(receiverPoints, sirenPoints, "sight_lines", 
                                "totalRecPointElevation_mapUnit", "totalPointElevation_mapUnit")

86
# Add field and calculate the 3D length of the sight lines (in meters). This length is the
distance between the associated siren and associated receiver

```python
arcpy.AddField_management("sight_lines", "distance3D_m", "DOUBLE")
```

codeblock="""
def convertDistanceToMeters(length, sr_metersPerUnit_str):
distance_m=float(length)*float(sr_metersPerUnit_str)
return distance_m"""

arcpy.CalculateField_management("sight_lines", "distance3D_m",
"convertDistanceToMeters(!shape.length3D!,
"+sr_metersPerUnit_str+")", "PYTHON_9.3", codeblock)

def calculateGeometricDivergence():
arcpy.AddMessage("Calculating Geometric Divergence...")
arcpy.AddField_management("sight_lines", "a_div", "DOUBLE")
```

codeblock = """
def calculateAdiv(distance):
aDiv=20* (math.log10 ((float(distance)/1))) + 11
return aDiv"""

arcpy.CalculateField_management("sight_lines", "a_div", "calculateAdiv(
!distance3D_m! )", "PYTHON_9.3", codeblock)
```

## def calculateMaxSoundCoverage(sirenPower_dB):
## # Calculates the maximum coverage area using the inverse square law to find the
## radius when the sound intensity is 0 watt/m^2
##
## # Convert sound power from decibels to watts
## # sirenPower_watts=10**((int(Lw)/10)-12)
##
## # Use inverse square law to solve for radius if sound intensity is 1x10^-9 W/m^2 (30
dB SPL, as loud as a quiet conversation)
##
## # MaxSoundCoverage_radius= math.sqrt(float(sirenPower_watts)/(4*math.pi*(10**
## -9)))
##
## # return MaxSoundCoverage_radius
```

def calculateAtmosphericAbsorptionCoefficient(temperature,frequency, humidity):
  # Define atmospheric absorption attenuation coefficient dictionary with following
  # format:
```

```
# Atmospheric absorption attenuation coefficient=
{ambient temperature in Celsius: [frequency in Hz: humidity in %: atmospheric absorption attenuation coefficient]}

```
A_atm_coef_dict={-20:
    800:
    1000:
    1250:
    1600:
    2000:
    {10:2.10, 15:2.80, 20:3.65, 30:5.73, 40:8.24, 50:11.0, 60:13.9, 70:16.6, 80:19.0, 90:21.0, 100:22.4},
    2500:
    3150:
    4000:
-10:
    800:
    1000:
    1250:
    1600:
    2000:
    2500:
    {10:5.54, 15:8.85, 20:12.8, 30:21.8, 40:30.1, 50:35.6, 60:37.8, 70:3.74, 80:35.5, 90:32.9, 100:30.2},
```
3150:
{10:6.11, 15:9.44, 20:13.5, 30:23.1, 40:32.9, 50:40.9, 60:45.9, 70:47.9, 80:47.5, 90:45.7, 100:43.1},
4000:
{10:7.00, 15:10.3, 20:14.5, 30:24.4, 40:35.4, 50:45.5, 60:53.5, 70:58.5, 80:60.7, 90:60.6, 100:59.0},
0:
{800:
{10:12.7, 15:14.7, 20:12.9, 30:8.49, 40:5.93, 50:4.52, 60:3.68, 70:3.16, 80:2.82, 90:2.59, 100:2.43},
1000:
1250:
1600:
2000:
{10:16.6, 15:27.2, 20:34.6, 30:36.0, 50:29.8, 60:23.8, 70:19.3, 80:13.8, 90:12.1, 100:10.8},
2500:
3150:
4000:
{10:19.0, 15:32.6, 20:47.0, 30:69.0, 40:75.2, 50:71.0, 60:63.3, 70:55.5, 80:48.8, 90:43.2, 100:38.6},
5:
{800:
{10:15.7, 15:10.4, 20:7.27, 30:4.58, 40:3.56, 50:3.13, 60:2.94, 70:2.87, 80:2.86, 90:2.89, 100:2.93},
1000:
1250:
1600:
2000:
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4000:</td>
<td>{10:57.3, 15:84.0, 20:91.5, 30:76.6, 40:59.0, 50:46.7, 60:38.4, 70:32.8, 80:28.7, 90:25.7, 100:23.5}</td>
</tr>
<tr>
<td></td>
<td>10:</td>
<td>{800:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10:15.7, 15:10.4, 20:7.27, 30:4.58, 40:3.56, 50:3.13, 60:2.94, 70:2.87, 80:2.86, 90:2.89, 100:2.93}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4000:</td>
</tr>
<tr>
<td></td>
<td>15:</td>
<td>{800:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{10:12.5, 15:7.63, 20:5.50, 30:3.89, 40:3.41, 50:3.29, 60:3.31, 70:3.38, 80:3.48, 90:3.57, 100:3.65}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1250:</td>
</tr>
</tbody>
</table>
1600:
2000:
2500:
{10:62.5, 15:54.8, 20:42.2, 30:26.9, 40:19.7, 50:15.9, 60:13.6, 70:12.2, 80:11.4, 90:10.8, 100:10.4},
3150:
4000:
20:
{800:
1000:
1250:
1600:
2000:
2500:
3150:
4000:
30:
{800:
{10:6.10, 15:4.98, 20:4.85, 30:5.21, 40:5.60, 50:5.82, 60:5.88, 70:5.81, 80:5.66, 90:5.47, 100:5.25},
1000:

1250:

1600:

2000:

2500:

3150:

4000:

40:
{800:

1000:

1250:

1600:
{10:13.6, 15:11.6, 20:11.6, 30:12.8, 40:14.0, 50:14.9, 60:15.3, 70:15.4, 80:15.2, 90:14.8, 100:14.4},

2000:

2500:

3150:

4000:

50:
A_atm_coef = str(float(A_atm_coef_Dict[(float(temperature))][(float(frequency))][(float(humidity))]))
return A_atm_coef

def populateSightLinesWith_a_atm(A_atm_coef):
    #Calculate attenuation from atmospheric absorption and add to receiver point attributes
    arcpy.AddField_management("sight_lines", "a_atm", "DOUBLE")

codeblock = """"def calculate_a_atm( A_atm_coef, distance):
    a_atm = (float(A_atm_coef) * float(distance))/1000
    return a_atm"
    arcpy.CalculateField_management("sight_lines", "a_atm", codeblock)"

def assignSiren_Numbers(sirenPoints):
    arcpy.AddField_management(sirenPoints, "Siren_Number", "DOUBLE")
def autoIncrement():
global rec
pStart = 1
pInterval = 1
if (rec == 0):
    rec = pStart
else:
    rec = rec + pInterval
return rec

arcpy.CalculateField_management(sirenPoints, "Siren_Number", "autoIncrement()", "PYTHON_9.3", codeblock)

def calculateTerrainAttenuation(sirenPoints, receiverPoints, sirenPoints_heightAboveTerrain, DEM):
    arcpy.AddMessage("Calculating Terrain Attenuation...")

    #Calculate the mean elevation under the lines of sight
    arcpy.ddd.AddSurfaceInformation("sight_lines", DEM, "Z_MEAN")

    #Add necessary calculation fields to sight lines layer

    #Add siren numbers, terrain elevation, and total elevations to lines of sight layer by joining to siren points.
    arcpy.JoinField_management("sight_lines", "OID_TARGET", sirenPoints, "OBJECTID", 
                             ["SIREN_NUMBER", "Total_Point_Elevation_m", "Point_terrainZ_m"])

    #Add receiver point total elevations and receiver point terrain elevations to lines of sight layer by joining to receiver points.
    arcpy.JoinField_management("sight_lines", "OID_OBSERV", receiverPoints, "OID", 
                             ["Total_Rec_Point_Elevation_m", "Rec_Point_terrainZ_m", 
                              "totalRecPointElevation_mapUnit"])

    #Test to see if sight lines test true for Case 1, then populate field with results.
Case 1 is true if the sight line mean height above the ground is greater than or equal to 1.5(abs(siren height above surface-receiver height above surface))/2)

codeblock="""def calculateAterr1(Siren_Point_terrainZ_m,Total_Siren_Point_Elevation_m,Total_Rec_Point_Elevation_m,Rec_Point_terrainZ_m, Z_Mean, sr_metersPerUnit):
    hm=((float(Total_Siren_Point_Elevation_m)+float(Total_Rec_Point_Elevation_m))/2)-(float(Z_Mean)*float(sr_metersPerUnit))
    hs=float(Total_Siren_Point_Elevation_m)-float(Siren_Point_terrainZ_m)
    hr=float(Total_Rec_Point_Elevation_m)-float(Rec_Point_terrainZ_m)
    if hm >= (1.5*(abs(float(hs)-float(hr))/2)):
        return -3
    else:
        return 0"""

arcpy.AddField_management("sight_lines", "Aterr_Case1", "DOUBLE")
arcpy.CalculateField_management("sight_lines","Aterr_Case1", "calculateAterr1(!Point_terrainZ_m!,!Total_Point_Elevation_m!,!Total_Rec_Point_Elevation_m!,!Rec_Point_terrainZ_m!,!Z_Mean!, "+sr_metersPerUnit_str")", "PYTHON", codeblock)

#Test to see if sight lines test true for case 2 or 3.
#Case 2 is true if the line of sight is blocked and the obstruction is greater than 5 meters from the receiver
#Case 3 is true if the line of sight is blocked and the obstruction is less than or equal to 5 meters from the receiver
#Runs the line of sight tool. This provides the TarIsVis field, which indicates whether the siren is visible from the receiver points.
#It also generates the first obstruction on the sight line. Lines are created between the receiver point and obstruction so that the (3D) length can be measured.

#Run line of sight tool on sight lines
arcpy.LineOfSight_3d(DEM, "sight_lines", "sight_lines_vis", "sight_line_vis_obs")

#Add the 'target is visible' (TarIsVis) field to sight lines layer. A value of 0 means target is not visible, 1 means target is visible
arcpy.JoinField_management("sight_lines", "OID", "sight_lines_vis", "SourceOID", "TarIsVis")
# Calculate the x, y and z values for obstacle points and x and y values for receiver points
arcpy.JoinField_management("sight_line_vis_obs", "SourceOID", "sight_lines", "OID", "OID_OBSERV")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_X", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_X", "!Shape!.centroid.X", "PYTHON_9.3")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_Y", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_Y", "!Shape!.centroid.Y", "PYTHON_9.3")
arcpy.AddField_management("sight_line_vis_obs", "obs_point_Z", "DOUBLE")
arcpy.CalculateField_management("sight_line_vis_obs", "obs_point_Z", "!Shape!.centroid.Z", "PYTHON_9.3")
arcpy.AddField_management(receiverPoints, "rec_point_x", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "rec_point_x", "!Shape!.centroid.X", "PYTHON_9.3")
arcpy.AddField_management(receiverPoints, "rec_point_y", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "rec_point_y", "!Shape!.centroid.Y", "PYTHON_9.3")

# Add the associated receiver point x, y and z field to the 'first obstacle on sight line' point feature class
arcpy.JoinField_management("sight_line_vis_obs", "OID_OBSERV", receiverPoints, "OID", ["rec_point_y", "rec_point_x", "Z"])

# Create lines connecting each receiver point to the first obstacle point on the sight line
arcpy.XYToLine_management("sight_line_vis_obs", "Rec_to_obs_lines", "rec_point_x", "rec_point_y", "obs_point_X", "obs_point_Y", "GEODESIC", "SourceOID")

# Add the associated receiver and siren z fields to the 'receiver to first obstacle on sight line' line feature class
arcpy.JoinField_management("Rec_to_obs_lines", "SourceOID", "sight_lines", "OID", ["totalRecPointElevation_mapUnit"])

# Use z fields to convert the 'receiver to first obstacle on sight line' to 3d (this is so the line length with reflect height differences between obstacle and receiver)
arcpy.FeatureTo3DByAttribute_3d("Rec_to_obs_lines", "Rec_to_obs_lines_3d", "totalRecPointElevation_mapUnit", "obs_point_Z")
#Add 3d length to 'receiver to first obstacle on sight line'
arcpy.AddZInformation_3d ("Rec_to_obs_lines_3d", "LENGTH_3D")

#rename 3d length field to "distance_to_obstacle_3D", and then join 3D length field to
sight lines layer.
arcpy.AddField_management ("Rec_to_obs_lines_3d", "distance_to_obstacle_3D", "DOUBLE")
arcpy.CalculateField_management("Rec_to_obs_lines_3d", "distance_to_obstacle_3D", "[Length3D]"

#Determine whether Case 2 or 3 is true and populate attributes with results
arcpy.AddField_management("sight_lines", "Aterr_Case2_or_3", "DOUBLE")
codeblock=""""def calculateAterr2_3(sr_metersPerUnit, TarIsVis, distance_to_obstacle_3D):
  if TarIsVis ==1:
    return 0
  elif distance_to_obstacle_3D is None:
    return 2
  elif 0<= (float(distance_to_obstacle_3D)*float(sr_metersPerUnit)) <=5:
    return 10
  else:
    return 2
"""
arcpy.CalculateField_management("sight_lines", "Aterr_Case2_or_3", "calculateAterr2_3( '+sr_metersPerUnit_str+', !TarIsVis!, !distance_to_obstacle_3D!)", "PYTHON_9.3", codeblock)
def calculateWindAttenuation(weatherCondition, windSpeed):
arcpy.AddMessage("Calculating Wind Attenuation")

  "0.5 - 1.5 m/s":{"Very Sunny Day":"A", "Mostly Sunny Day":"A", "Partly Cloudy Day":"B", "Overcast Day":"C", "Twilight":"D", "Clear Night":"F", "Partly Cloudy Night":"F", "Cloudy Night":"D"},
}
PasquillCategory=PasquillSuitabilityIndex[windSpeed][weatherCondition]

if windSpeed=="<.5 m/s":
    windSpeed_forMetCategoryLookup="0-0.5"
elif (windSpeed=="0.5 - 1.5 m/s") or (windSpeed=="2.0 - 2.5 m/s"):
    windSpeed_forMetCategoryLookup="0.5-3.0"
elif (windSpeed=="3.0 - 4.5 m/s") or (windSpeed=="5.0 - 6.0 m/s") or (windSpeed==">6.0 m/s"):
    windSpeed_forMetCategoryLookup=">3.0"

meteorologicalCategoryIndex={"A":{">3.0":1, "0.5-3.0":2, "0-0.5":3},
"B":{">3.0":1, "0.5-3.0":2, "0-0.5":3},
"C":{">3.0":2, "0.5-3.0":3, "0-0.5":4},
"D":{">3.0":2, "0.5-3.0":3, "0-0.5":4},
"E":{">3.0":2, "0.5-3.0":3, "0-0.5":4},
"F":{">3.0":3, "0.5-3.0":4, "0-0.5":5}}

meteorologicalCategory=str(meteorologicalCategoryIndex [PasquillCategory] [windSpeed_forMetCategoryLookup])

arcpy.AddField_management("sight_lines", "MeteorologicalCat", "TEXT")
arcpy.CalculateField_management("sight_lines", "MeteorologicalCat", "'
+meteorologicalCategory+ "'", "PYTHON")
arcpy.AddField_management("sight_lines", "A_Wind", "DOUBLE")

codeblock=""""def calculateWind_Attenuation(meteorologicalCategory, distance):
    if meteorologicalCategory==1:
        wind_attenuation=-38.9+(26.4*(math.log10(distance))-(2.84*((math.log10(distance))**2))-(.234 *(math.log10(distance))**3))
    elif meteorologicalCategory==2:
        wind_attenuation=-114+(119*(math.log10(distance))-(39.8*((math.log10(distance))**2))+(4.43 *(math.log10(distance))**3))""""
elif meteorologicalCategory==3:
    wind_attenuation=-28+(21.3*(math.log10(distance))-(3.85*((math.log10(distance))**2))+(.0903 *((math.log10(distance))**3))

elif meteorologicalCategory==4:
    wind_attenuation=0

elif meteorologicalCategory==5 or meteorologicalCategory==6:
    wind_attenuation=8.21-(1.14*(math.log10(distance))-(2.87*((math.log10(distance))**2))+(.671 *((math.log10(distance))**3))

    if wind_attenuation<0:
        wind_attenuation=0
    return wind_attenuation

arcpy.CalculateField_management("sight_lines", "A_Wind", 
"calculateWind_Attenuation( "+meteorologicalCategory+" , !distance3D_m!)", 
"PYTHON_9.3", codeblock)

def calculateTotalSoundPressure(Lw, includeWind):
    arcpy.AddMessage("Calculating Total Sound Pressure...")
    arcpy.AddField_management("sight_lines", "totalSoundPressure", "DOUBLE")

    if includeWind=="true":
        codeblock= """"def calculateTotalLw( Lw, a_div, a_atm, Aterr_Case1, Aterr_Case2_or_3, A_Wind):
            total=float(Lw)-a_div-a_atm-Aterr_Case1-Aterr_Case2_or_3-A_Wind
            return total"
        arcpy.CalculateField_management("sight_lines", "totalSoundPressure", 
"calculateTotalLw(" +Lw+ ", !a_div!, !a_atm!, !Aterr_Case1!, !Aterr_Case2_or_3!, !A_Wind!)", 
"PYTHON", codeblock)
    else:
        codeblock= """"def calculateTotalLw( Lw, a_div, a_atm, Aterr_Case1, Aterr_Case2_or_3):
            total=float(Lw)-a_div-a_atm-Aterr_Case1-Aterr_Case2_or_3
            return total"
        arcpy.CalculateField_management("sight_lines", "totalSoundPressure", 
"calculateTotalLw(" +Lw+ ", !a_div!, !a_atm!, !Aterr_Case1!, !Aterr_Case2_or_3!)", 
"PYTHON", codeblock)"
def GenerateExistingSirenCoverageArea(MinimumdB_forCoverage, receiverPoints, sirenPoints):
    whereClause=str(""""totalSoundPressure" \>="""" + MinimumdB_forCoverage)
    arcpy.MakeTableView_management("sight_lines", "sight_lines_for_coverage_area", whereClause)
    arcpy.JoinField_management(receiverPoints, "OID", "sight_lines_for_coverage_area", "OID_OBSERV", "totalSoundPressure")
    arcpy.Select_analysis(receiverPoints, "selected_receiverPoints", whereClause)
    bufferDistance=ReceiverGridSpacing_mapUnits*(math.sqrt(2)/2)

    desc = arcpy.Describe(DEM)

    ## expression=str(str(desc.extent.XMin-1000) + " "+ str(desc.extent.YMin-1000) + " "
    ## +str(desc.extent.XMax+1000) + " " +str(desc.extent.YMax+1000))
    ## arcpy.env.XYDomain = expression
    arcpy.Buffer_analysis("selected_receiverPoints", str(str(Workspace)+"existing_siren_coverage_area"), bufferDistance, "FULL", "ROUND", "ALL")

    def SubtractExistingSirenCoverageArea(receiverPoints):
        arcpy.Erase_analysis(studyArea, str(str(Workspace)+"existing_siren_coverage_area"), str(str(Workspace)+"studyArea_minusExistingCoverage"))
        arcpy.DeleteField_management(receiverPoints, "totalSoundPressure")
        ##arcpy.Erase_analysis(receiverPoints, str(str(Workspace)+"existing_siren_coverage_area"), str(str(Workspace)+"receiverPoints_forUpdatedStudyArea"))

    existingCoverage=arcpy.mapping.Layer(str(str(Workspace)+"existing_siren_coverage_area"))
    arcpy.mapping.AddLayer(df,existingCoverage)

    updatedStudyArea=arcpy.mapping.Layer(str(str(Workspace)+"studyArea_minusExistingCoverage"))
    arcpy.mapping.AddLayer(df,updatedStudyArea)
# Define global variables
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd)[0]
global df

df_spatial_reference = df.spatialReference
sr_metersPerUnit_str=str(df_spatial_reference.metersPerUnit)
global sr_metersPerUnit_str

ReceiverGridSpacing_mapUnits=float(ReceiverGridSpacing_m)/
float(sr_metersPerUnit_str)
global ReceiverGridSpacing_mapUnits

# Call functions and assign returns as variables
checkout_licenses()
checkIfDEMContainsStudyArea(DEM, studyArea)
receiverPoints=generatePointGrid(studyArea, ReceiverGridSpacing_mapUnits)
assignSiren_Numbers(sirenPoints)
##maxSoundCoverage=calculateMaxSoundCoverage
# Calculate sound pressure in already existing sirens

arcpy.AddField_management(receiverPoints, "heightAboveSurface_m", "DOUBLE")
arcpy.CalculateField_management(receiverPoints, "heightAboveSurface_m", '1.5')

getzValues(receiverPoints, Buildings, Building_height, Building_height_units, DEM,
DEM_VerticalUnits, "heightAboveSurface_m")
giveReceiverZFieldsUniqueNames(receiverPoints)
getzValues(sirenPoints, Buildings, Building_height, Building_height_units, DEM,
DEM_VerticalUnits,sirenPoints_heightAboveTerrain )

calculateDistanceBetweenPoints(sirenPoints,receiverPoints, sr_metersPerUnit_str)
Lw=str(sirenPower_dB)
# Calculate attenuation factors for existing sirens
calculateGeometricDivergence()
A_atm_coef=calculateAtmosphericAbsorptionCoefficient(AmbientTemperature_Celsius,
frequency, AmbientHumidity)
populateSightLinesWith_a_atm(A_atm_coef)
calculateTerrainAttenuation(sirenPoints, receiverPoints,
sirenPoints_heightAboveTerrain, DEM)
if includeWind=="true":
    calculateWindAttenuation(weatherCondition, windSpeed)

#Calculate total sound pressure for existing sirens
calculateTotalSoundPressure(Lw, includeWind)

#Generate coverage area of existing sirens
GenerateExistingSirenCoverageArea(MinimumdB_forCoverage, receiverPoints, sirenPoints)
SubtractExistingSirenCoverageArea(receiverPoints)

arcpy.DeleteField_management (sirenPoints, ["Siren_Number", "Z", "Point_terrainZ_m", "buildingHeight_m", "Total_Point_Elevation_m", "totalPointElevation_mapUnit"])
#Define directivity correction
D_c=0