RADIO-FREQUENCY TRANSMITTER GEOLOCATION USING NON-IDEAL RECEIVED SIGNAL STRENGTH INDICATORS

by

Samuel Whiting

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Todd Moon, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Reyhan Baktur, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

Radio-Frequency Transmitter Geolocation Using Non-Ideal Received Signal Strength Indicators

by

Samuel Whiting, Master of Science

Utah State University, 2018

Major Professor: Todd Moon, Ph.D.
Department: Electrical and Computer Engineering

Received signal strength (RSS) is a metric easily obtained with simple hardware that measures the amount of power at a frequency of interest. By taking RSS measurements, also known as indicators (RSSI), at different locations, the general location of a transmitter can be estimated in what is commonly known as geolocation.

Geolocation based on RSS measurements differs from other geolocation methods in a few critical ways. The first of these differences is the lack of time as a dimension in RSS measurements. This greatly simplifies the hardware requirements and processing, but at the cost of temporal information. Another key difference is that the RSS measurements have no phase, and therefore there is no need for phase coherency in any of the receivers. This again simplifies the measurements and calculations.

While the data may be easy to obtain, there are great challenges to overcome in order to make accurate transmitter location estimates with these measurements. Most significantly, the electromagnetic power measurements suffer from multi-path distortion, shadowing, additive thermal receiver noise, ambient radiation noise, hardware limitations, and quantization.

By appropriately modeling the problem, this thesis develops and proposes a number of algorithms that can overcome these issues in order to geolocate a transmitter based on spatially separated sequences of RSS measurements. In particular, a scheme for data collection is presented and used to collect real-world datasets. Algorithms are developed in simulation and tested on these real datasets. Comparisons are made as to which algorithms perform better and a decision is made that the subset method, as described in chapter 6, performs the best overall.

(129 pages)

PUBLIC ABSTRACT

Radio-Frequency Transmitter Geolocation Using Non-Ideal Received Signal Strength Indicators

Samuel Whiting

Locating a radio transmitter is important in a number of problems such as finding radio tags, people with radios, and devices that are collecting information in an unauthorized manner. Locating a radio transmitter is inherently difficult because the radio waves of concern are not in the visible spectrum, they reflect and distort easily, and they propagate at the speed of light.

A number of methods for locating transmitters are currently used, the majority of which require expensive hardware and extensive processing. This thesis presents a method of using simpler measurements to produce similar location estimates in order to augment or replace current systems. While other systems have significant advantages, the methods proposed in this thesis are advantageous because they only require easily-obtained measurements that are based on the observed power of the transmission.

The research uses simulations and experiments on real-world data collected locally to demonstrate the possibility of locating a transmitter using information of this type. The conclusion is that some methods are able to compensate for the difficulties in the problem more effectively, and produce useful location estimates.

Dedicated to my wife, Peitra.

ACKNOWLEDGMENTS

A number of people have helped me to continue my education, complete my research, and write this thesis.

Firstly I want to thank my wife, Peitra, and our children for allowing me to spend so much of my time studying and conducting research, and for wandering around with me to collect data.

I hold in the highest regard Dr. Todd Moon who has spent hours of his life instructing and advising me on countless topics. I consider him the best teacher and lecturer I've ever had and one of the most clever engineers I've met.

My parents, Peter and Natalie, have been endlessly supportive and full of helpful advice about algorithms, math, writing, and life.

Trevor Landeen has spent hours of his time working through math problems, talking about new ideas, and discussing research with me.

Finally, I would like to thank the team at the Laboratory for Telecommunication Sciences for funding our research and providing tools to help collect data and analyze the results.

Sam Whiting

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Overview

Determining the physical location of a radio-frequency (RF) transmitter is a well-known problem commonly referred to as geolocation. Geolocation information can be used to find RF jammers or find unauthorized transmitters, locate RF tags, find or track people and vehicles, as well address many other problems [1–5]. The problem is inherently difficult for a few key reasons. Depending on the method used, a geolocation system may require sophisticated hardware in order to achieve high degrees of synchronization and coherency. Additionally, processing of large amounts of data required to locate the source can quickly become intractable for modern computers. Other methods that may not require expensive hardware may still be affected by electromagnetic phenomena such as multipath distortion or fading [6, 7].

Among the most common methods for geolocation systems are time of arrival, time difference of arrival, angle of arrival, received signal strength indicators, Doppler, pseudo-Doppler, and the use of highly directional antennas. Of all of these methods, the one which generally requires the least amount of hardware to implement is using received signal strength (RSS) [8].

This thesis focuses on the use of RSS data in estimating transmitter location. Multiple algorithms are proposed, evaluated, and tested in simulation to develop a geolocation method best adapted to this problem. The algorithms are then tested on real-world datasets.

1.2 Literature Review

1.2.1 Geolocation

Geolocation, as stated above, is useful for a number of reasons and is associated with a considerable amount of research addressing its complexities as well as different methods for obtaining accurate location estimates in a variety of settings. The most common methods for geolocation include [1, 9]:

- Time of Arrival (TOA)
- Time Difference of Arrival (TDOA)
- Angle of Arrival (AOA or sometimes DOA)
- Received Signal Strength Indicators (RSSI)

It is also interesting to note that geolocation using WiFi fingerprinting is commonly used on smart phones when GPS location services are unavailable. This method involves making databases of known locations associated with known signal strengths when GPS services are available and saving these measurements for later use. This method generally has poor accuracy with comparison to other methods unless further processing is done [10]. This problem, where a receiver is trying to locate itself based on a known transmitter, is the inverse of the presented research problem. For this reason, the WiFi fingerprinting method is not considered here.

1.2.2 TOA/TDOA/AOA

Time-of-arrival methods are sometimes referred to as time of flight methods, and can be used when the time of transmission is known. Besides requiring prior knowledge about the signal, this method suffers greatly from synchronization problems, when the clocks between receivers or between transmitters and receivers are not perfectly aligned [9, 11].

Time-difference-of-arrival methods address the case where the exact transmission time of a signal is not known by the receivers. This is a commonly occurring case: searching

for a signal without knowing exactly when it will be sent. By determining when receivers capture a signal relative to each other, certain statements can be made about the location of the transmitter. Specifically, hyperbolas can be drawn between receivers that represent a constant difference of arrival time between the two receivers. When multiple receivers are able to compare measurements, these hyperbolas should intersect at the location of the transmitter. This method suffers from synchronization problems like the TOA method [9]. Both of these time methods also suffer from not having direct line-of-sight from transmitter to receiver, which is common in urban environments [7].

Angle of arrival methods rely on phase differences to determine the angle from which the signal of interest is propagating. Using an array of antennas (generally spaced at half-wavelength intervals) and assuming planar wave propagation, the phase difference between received signals at each antenna corresponds to the physical direction of the transmitter. Perhaps even more than time-based methods, this phase method suffers greatly from multipath distortion [3, 7].

In general, these synchronization and multipath problems can be overcome using extensive processing or improved hardware. The proposed research seeks to address this geolocation problem without the use of high-quality hardware or high degrees of time synchronization. Instead, another measurement will be used (RSSI) which will serve both to simplify the receiver design and to make data collection less rigorous [12].

1.2.3 RSSI

Received signal strength indicators give a raw power measurement and can be taken using relatively simple hardware [2, 4, 8, 13]. RSSI measurements are used to give measures of signal quality and roughly determine how far a receiver is from a transmitter [14]. In an ideal setting (where the transmitted power is known, there is no noise, and the signal propagates through free space,) the distance that the signal has traveled to reach the receiver can be found using the Friis transmission equation:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 R^2} \,. \tag{1.1}$$

Of course, it is rarely the case that all of these ideal settings exist in a real system. Specifically, this research project seeks to determine the location of a transmitter where the power transmitted is not known, the signal is noisy, and the region between the receiver and transmitter is not free space.

Not knowing the transmitted power can be dealt with in a way similar to how TDOA methods operate without knowing transmission time. Measurements from multiple receivers can be used to draw circles (in free space) of constant power ratios between those receivers.

The problems associated with RSSI measurements are well known, and "the mathematical and statistical methods to tackle RSSI-variance problem need further research." [6] This research aims to address the non-idealities associated with using RSSI measurements for geolocation.

1.2.4 Geolocation using Mobile Sensors

Distributed systems are well suited for taking RSSI measurements because the measurement is simple and measurements must be taken at different physical locations. In practice, one moving receiver can take multiple observations. For this reason, multiple studies have been done using drones or other autonomous robots to geolocate a transmitter [2, 3, 8, 9].

While the research does not dictate measurements be taken autonomously using robots, it does collect data using a mobile, distributed platform that allows a receiver to move physically and take multiple measurements at different locations. For this reason, the work done in this area is applicable to the research presented here, and provides useful insight into the possibilities available to a distributed network of portable receivers.

1.3 Chapter Outlines

Chapter 2 presents the objectives of the research in more specific terms and outlines the methods that will be used to develop the needed algorithms, including simulations and

real world data collection and analysis. Also presented are notes on mapping latitude and longitude positions to Cartesian coordinates, and the significance of loss coefficients, or path loss.

Beginning in chapter 3, proposed algorithms are presented. The algorithm in chapter 3 is referred to as the circles algorithm, because it uses a geometric-based approach to solve the geolocation problem. Derivations, simulations, and trials on real world datasets are presented.

Following the circles algorithm, the *Binary Cascading Probability* (BCP) algorithm is presented in chapter 4. The BCP algorithm relies on simple comparisons between power measurements to update a grid of probabilities that represents transmitter location likelihoods. Again, the algorithm is presented along with simulations and trials on real world datasets.

Chapter 5 presents the *3-parameter method*, which models the problem as a function of three parameters and finds the optimal solution to this function using Newton's method. A simplification to this algorithm is also presented and is referred to as the simplified algorithm, or *simplified 3-parameter method*. Both simulations and trials on real world data are presented.

Chapter 6 builds on the 3-parameter method to form a new method referred to as the *subset method*. It operates on subsets of measurements in order to compensate for noise and provides a more probabilistic interpretation than is available with just the 3-parameter method.

In an attempt to more accurately represent the system, Chapter 7 presents new models that consider loss coefficients and additional noise terms as parameters. These models and methods are generally referenced by the number of parameters they seek to estimate and other important information they assume (e.g., the 6-parameter method with an alternative cost function). These methods eventually prove to be unstable and do not produce useful location estimates, but are discussed in order to explain why these models do not represent the problem well.

In chapter 8, a comparison of results from each method is presented. Trade-offs are analyzed, and a determination is made as to which algorithm performs the best overall. Additional observations are made about the possibility of using these geolocation estimates in geographic analysis, alternative methods used for taking RSSI measurements, and non-stationary transmitters.

Chapter 9 concludes the thesis and declares the objectives satisfied. Appendices for complex equations and matrices are included at the end, along with code for performing the presented algorithms.

Throughout the chapters, a number of algorithms are presented which are not included in the final chapter's analysis, but are nonetheless important. Either the concepts they represent are used in other methods or the ideas behind them help to demonstrate that other methods were considered in addition to the final ideas presented.

CHAPTER 2

Problem Definition

2.1 Objective

The overall purpose of this research is to develop algorithms to locate radio frequency transmitters based on a set of received signal strength (RSS) measurements, which are also known as indicators (RSSI). These measurements are assumed to be spatially separated in the general local area of the transmitter. The center frequency of interest is known, and the transmitter is assumed to be stationary.

In direct comparison with other geolocation methods, such as time-difference of arrival (TDOA) and phase based algorithms such as direction of arrival (DOA) techniques, this method (RSSI) may prove not to be the best solution for producing a final, accurate location estimate quickly. To outperform all other methods is not the objective of this research. On the contrary, this RSS-based algorithm is to be designed in such a way as to complement other systems by providing additional information from available data.

The RSS data is substantially easier to obtain than other methods because, under the assumptions, time synchronization is not important. Along those same lines of reasoning, the individual power measurements made have no phase information, so phase coherency is not a concern either. This greatly simplifies the data collection process as well as the hardware. Simplicity in data collection is an important advantage of the methods presented in this research. The data collection hardware and software were developed prior to this research in order to create datasets to analyze the algorithms.

The ideal algorithm should be able to use noisy RSS measurements to produce a location estimate for a transmitter of interest while ignoring the adverse effects of shadowing, multi-path distortion, additive noise power, and thermal receiver noise. The desired algorithm would also produce some metric of certainty along with the estimate to aid in the

fusion of the estimate with estimates from other algorithms or methods.

2.2 Simulations

An accurate simulation can greatly aid the development of an algorithm by making available far more information than is accessible in real world data and by allowing for greater control over the system setup. Simulated data is also far easier to generate than to record, and can therefore be used to test edge cases or unusual circumstances in rapid succession. For these reasons and other, simulations were used initially to evaluate and develop algorithms.

The simulations and models used are described in the following sections as the algorithms are presented. In many cases, additional tests were done with the simulations, but the outcomes and reasons behind these changes are considered either irrelevant in the final conclusions or too lengthy to include in this thesis.

2.3 Data Collection Methods

An android phone was connected to a software defined radio (RTL-SDR) and ran a GNURadio script to record power measurements. The phone combined the power measurement with a set of GPS coordinates for the phone location. The data was offloaded to a computer for processing.

The entire processing chain can be visualized in figure 2.1.

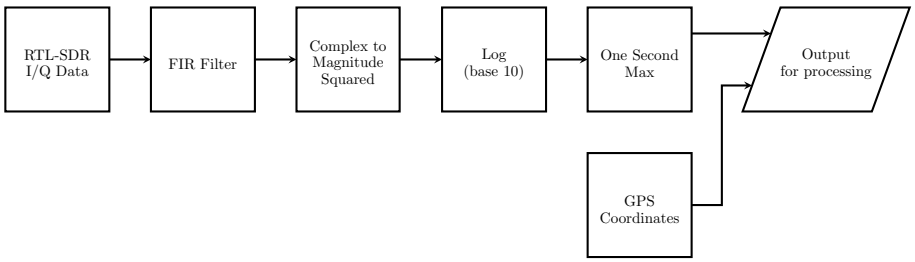


Fig. 2.1: Data collection scheme.

In practice, the method for taking measurements could easily be automated by attaching RSS measuring radios and phones to autonomous vehicles. The architecture of the system also allows for multiple measuring nodes to contribute data simultaneously, allowing for multiple spatially-separated observers. The datasets used in this study were obtained using a single receiver that moved around. Under our stationary transmitter assumption, there should be no difference in these collection methods as long as the radios, gains, and antennas are consistent.

2.4 Real Datasets

Real datasets were recorded in Logan, Utah, U.S, with a few different transmitters and at different scales. The most commonly referenced datasets are as follows:

- sant1 - Walkie-talkie transmitting near the Sant building on Utah State University (USU) campus.
- sant2 - Walkie-talkie near the Sant building on USU campus.
- quad3 - Walkie-talkie in the center of the USU Quad field.
- upr3 - Local FM radio station, measurements taken throughout USU campus.
- aggr1 - Local FM radio station, throughout USU campus and surrounding neighborhood.
- aggr4 - Local FM radio station, around the city of Logan, Utah.

Descriptions of the datasets are given below, with figures 2.2, 2.3, 2.4, 2.5, 2.6, and 2.7 depicting the individual power measurements as points on a satellite map. The colors of the points represent specific power measurements, which are not discussed in this thesis.

2.4.1 The sant1 dataset

This dataset will be referred to as the sant1 dataset throughout the paper. The sant1 dataset was taken Oct 21, 2017, in the small field outside the USU Sant building. The

transmitter was a 0.5-watt walkie-talkie in the family radio service band. The RTL-SDR recorded data with a 40 dB attenuator in line to prevent saturation. The sant1 dataset roughly covers a 65×65 square-meter region.

2.4.2 The sant2 dataset

This dataset will be referred to as the sant2 dataset throughout the paper. The sant2 dataset was taken Oct 21, 2017, in the small field outside the USU Sant building. The transmitter was a 0.5-watt walkie-talkie in the family radio service band. The RTL-SDR recorded data with a 40 dB attenuator in line to prevent saturation. The sant2 dataset roughly covers a 65×65 square meter region.

2.4.3 The quad3 dataset

This dataset will be referred to as the quad3 dataset throughout the paper. The quad3 dataset was taken Oct 21, 2017, on Utah State University campus around the field known as "The Quad." The transmitter was a 0.5-watt walkie-talkie in the family radio service band. The RTL-SDR recorded data with a 40 dB attenuator in line to prevent saturation. The Quad is an open field sized about 150×125 square-meters.

2.4.4 The upr3 dataset

This dataset will be referred to as the upr3 dataset throughout the paper. The upr3 dataset was taken Nov 9, 2017, around Utah State University campus. The transmitter was a local FM radio station broadcasting from a tower on campus. The upr3 dataset roughly covers a 600×850 square-meter region.

2.4.5 The aggr1 dataset

This dataset will be referred to as the aggr1 dataset throughout the paper. The aggr1 dataset was taken Nov 9, 2017, around Utah State University campus. The transmitter was a local FM radio station broadcasting from a tower on campus. The aggr1 dataset roughly covers a 1200×1000 square-meter region.

2.4.6 The aggr4 dataset

This dataset will be referred to as the aggr4 dataset throughout the paper. The aggr4 dataset was taken Feb 17, 2018, around the city of Logan, Utah. The transmitter was a local FM radio station broadcasting from a tower on USU campus. The aggr4 dataset roughly covers a 5000×5000 square-meter region.

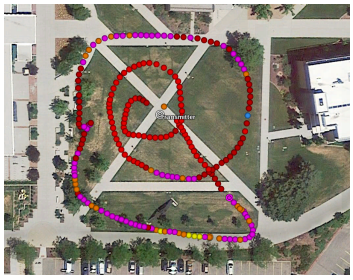


Fig. 2.2: The sant1 dataset.

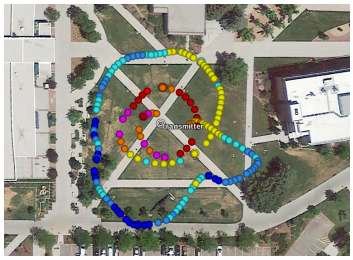


Fig. 2.3: The sant2 dataset.

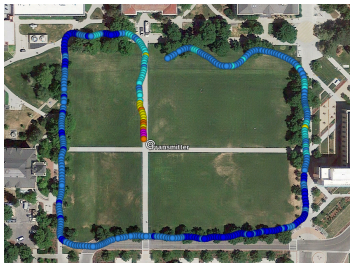


Fig. 2.4: The quad3 dataset.

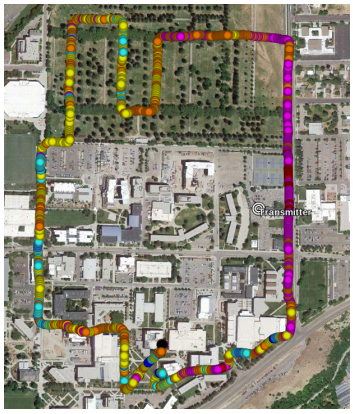


Fig. 2.5: The upr3 dataset.

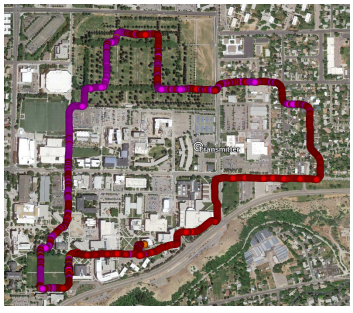


Fig. 2.6: The aggr1 dataset.

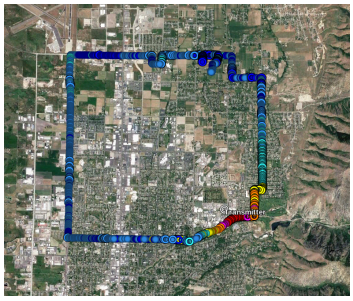


Fig. 2.7: The aggr4 dataset.

2.5 Note on Loss Coefficients

The Friis transmission equation models the propagation of electromagnetic waves in free space. Because of the free space assumption the exponent on the distance term in the denominator is 2 or, in other words, the power loss is quadratic. A significant difference and contribution of this research is to propose methods of estimating transmitter locations in the presence of non-quadratic power loss.

The exponent of the distance term will be referred to as the loss coefficient throughout this paper, though it is also commonly referred to as path loss. Typical loss coefficients are generally in the range of 2 to 4 depending on the presence of trees, buildings, and other obstructions. In some cases, loss coefficients can be less than 2 (for instance, inside some buildings that act as waveguides) and can be far greater than 4 due to shadowing and multipath effects. More detail about the circumstances and reasoning behind these values can be found in [15].

The ability to accurately determine the loss coefficients would be useful not only in improving location estimates, but also in determining features of the environment as well. Heavily forested regions, for example, would create different sets of loss coefficients when compared to measurements taken in a plains region with relatively open space, or compared to an urban region with significant shadowing and strong multipath effects from reflective surfaces. This possibility is discussed further in section 8.3.

Some of the proposed methods estimate loss coefficients as part of the algorithm. For those which do not, a method for estimating loss coefficients from a location estimate is provided in section 5.3.

2.6 Note on Latitudes and Longitudes

Most of the methods treat latitude and longitude coordinates as if they formed a uniform grid. While not an exact representation of the setup, empirically using these in a grid-like manner has no negative effect on the algorithms presented in this thesis. This is due to the size of the search grid and the location on earth where measurements were taken.

Whenever mentioned, the Cartesian mapping used is the one given in listing 2.1. The code presents a way to change a pair of latitude and longitude points to a pair of Cartesian points on a plane where the earth is represented as a flat surface with the x and y axes in meters and the origin at the intersection point of the Prime Meridian and Equator. The warping factor compensates for the distortion only at one given latitude, so the approximation is only good when close to that latitude. For the datasets used here, (all measurements within 8 km of each other) this approximation is sufficient.

Listing 2.1: Cartesian Mapping

```
% mapping
DEG2RAD = pi/180;
warp_factor = cos(lat0*DEG2RAD)

y0 = lat0 * 111111;
x0 = lon0 * 111111 * warp_factor;

y1 = lat1 * 111111;
x1 = lon1 * 111111 * warp_factor;

% now we can use the Cartesian distance
dist_met2 = sqrt( (x1-x0)^2 + (y1-y0)^2 )
```

2.6.1 Note on Elevation

The methods and hardware used to measure real-world data in this research do not produce elevation metrics for the observations. Because of this, and to simplify the algorithms and concepts, all observations are assumed to be on the same plane. While this assumption can clearly cause distance errors it is treated as part of the unknown variations of the GPS, power measurements, and loss coefficients.

Further work could be done to more accurately model the possibility of an elevated transmitter.

CHAPTER 3

Circles Method

The circles algorithm attempts to draw circles of constant power ratio between observations in order to determine the transmitter location. Locations with large numbers of intersections suggest a high likelihood of having a transmitter at that location.

3.1 Ideal Case

First examine the ideal case. Let there be measurements of power received, a known power transmitted, known gains and frequencies, and assume quadratic power loss due to distance. Friis transmission equation can be solved directly to obtain the distance, d.

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2}$$

$$d = \sqrt{\frac{P_t G_t G_r \lambda^2}{(4\pi)^2 P_r}}$$

Draw a circle of radius d that represents a locus of possible transmitter locations. This simple simulation can be seen in figure 3.1.

If there are three observations, the transmitter location can be exactly determined by finding where the circles intersect, as in figure 3.2.

3.2 Constant Power Ratio

Now assume that the transmitted power is not known. With two receivers, each can measure a received power.

$$P_{r1} = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d_1^2}$$

$$P_{r2} = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d_2^2}$$

Taking the ratio of these two powers, most of the terms in Friis equation cancel, leaving

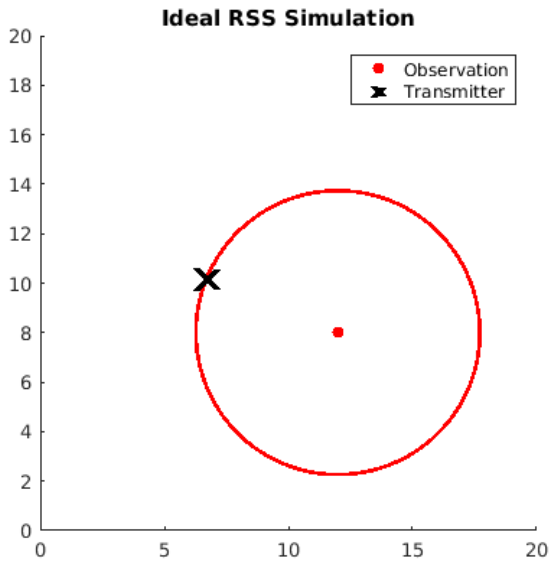


Fig. 3.1: Ideal case with one observation.

$$\frac{P_{r1}}{P_{r2}} = \frac{\frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d_1^2}}{\frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d_2^2}} = \frac{d_2^2}{d_1^2} .$$

Assuming everything is stationary, the distances d are constant, so this term can be reduced to a constant, k, leaving

$$d_2 = k d_1 .$$

Substitute this into the ratio to get

$$\frac{P_{r1}}{P_{r2}} = k^2 .$$

This constant ratio is easily understood with an example. If $k = 2$, d_2 is twice as long as d_1. This means the observation at d_2 is twice as far away from the transmitter as the d_1 observation. Circles of constant-ratio radius can be drawn using the power ratio between the two observations. This is depicted in figure 3.3.

The transmitter must be somewhere on the circle of constant radii ratio. Adding more observations, the location of the transmitter can be quickly determined based on the circle

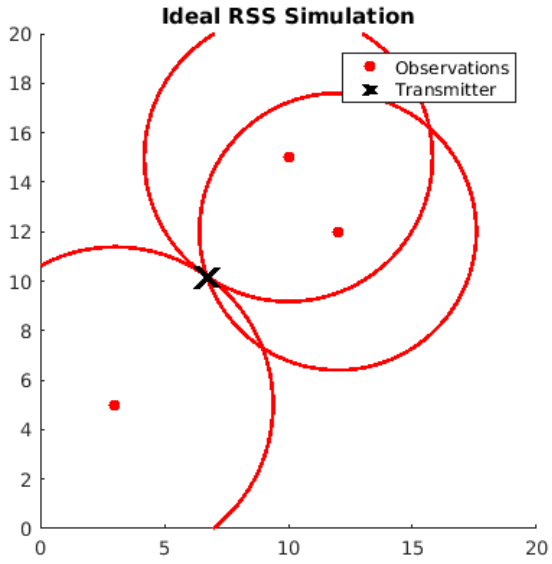


Fig. 3.2: Ideal case with three observations.

intersections (once we have 4 observations). This is depicted in figure 3.4.

How is it known where these circles lie, or in other words, how is it known which points are on the locus of valid transmitter locations? Consider two circles, centered at (a, b) and (c, d). These two circles have radii related by the constant k^2, as defined earlier.

$$(x-a)^2 \ + \ (y-b)^2 \ = \ r^2$$

$$(x-c)^2 \ + \ (y-d)^2 \ = \ k^2 r^2$$

Since this radii ratio is known, equate these two circles as

$$(x-a)^2 \ + \ (y-b)^2 \ = \frac{1}{k^2} \ \left(\ (x-c)^2 \ + \ (y-d)^2 \ \right) .$$

Solve to make an equation for a new circle as

$$(x-u)^2 \ + \ (y-v)^2 \ = \ w,$$

where

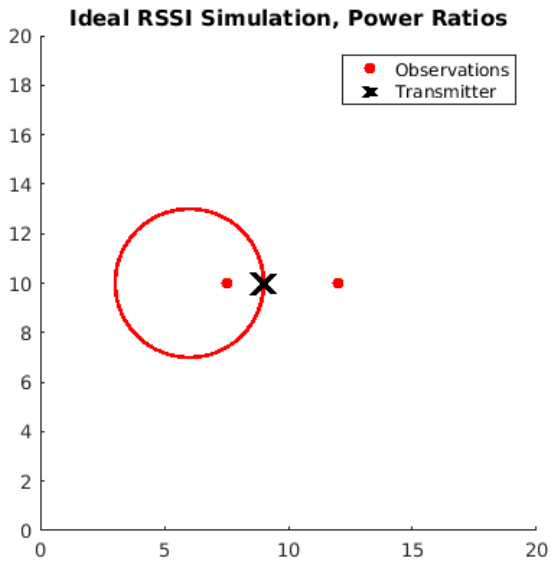


Fig. 3.3: Using power ratios with two observations.

$$u = \frac{\left(ak^2 - c\right)}{\left(k^2 - 1\right)}$$

$$v = \frac{\left(bk^2 - d\right)}{\left(k^2 - 1\right)}$$

$$w = u^2 + v^2 - \frac{\left(k^2\left(a^2 + b^2\right) - c^2 - d^2\right)}{k^2 - 1}.$$

This new circle represents the locus of valid transmitter locations for the given power ratio and observation locations.

3.3 Power Ratio with Additive Power Noise

In real-world observations, there will be noise added onto the measurements. In the context of the circles algorithm, the locus of points may no longer intersect the true transmitter location. Circle intersections may no longer provide a good estimate of location, as can be seen in figure 3.5.

There can even be cases where there are no intersections at all, as in figure 3.6. However, this only occurs if we allow for negative noise in our measurements and all the noise

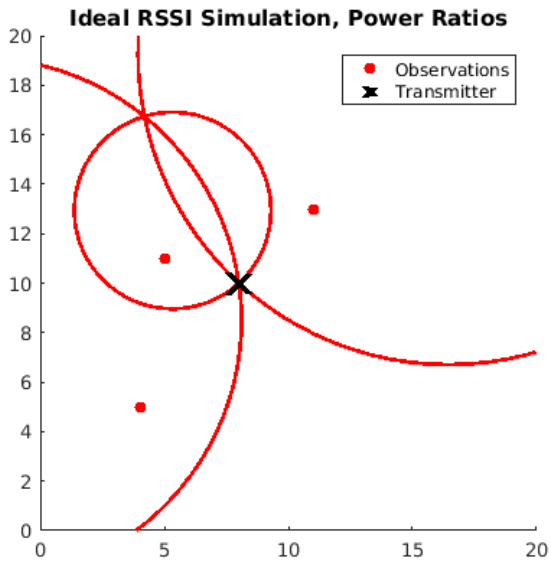


Fig. 3.4: Using power ratios with three observations.

experienced in our observations should be an additive power. For this simulation, the added noise was generated by taking the magnitude of a zero-mean Gaussian distribution with standard deviation 0.2.

In general, this problem of additive noise can be overcome using more than just three observations. Since it becomes tiresome to look at more than three or so circles, represent this data as a heat map for easier interpretation. By taking a two-dimensional histogram of the circle intersections, the data can be easily interpreted, as in figures 3.7 and 3.8.

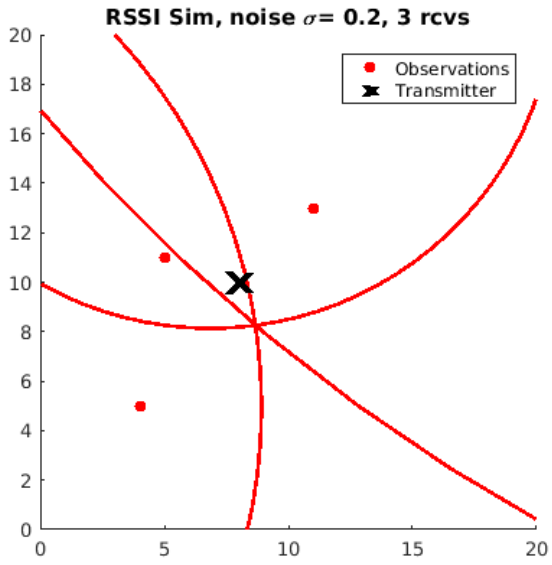


Fig. 3.5: Using power ratios with three observations and noise.

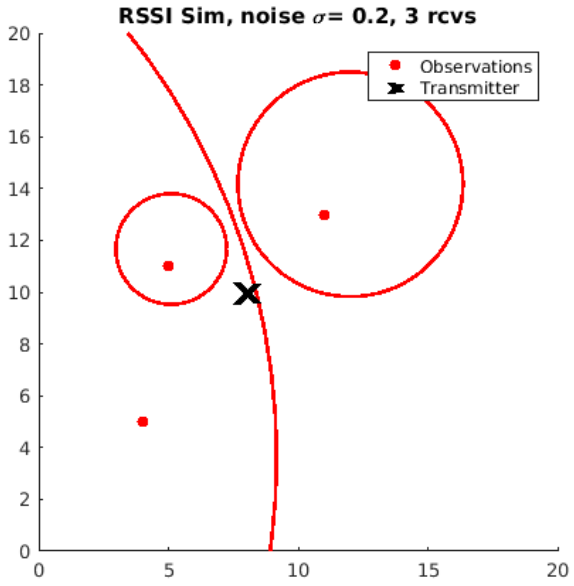


Fig. 3.6: Using power ratios with three observations and noise.

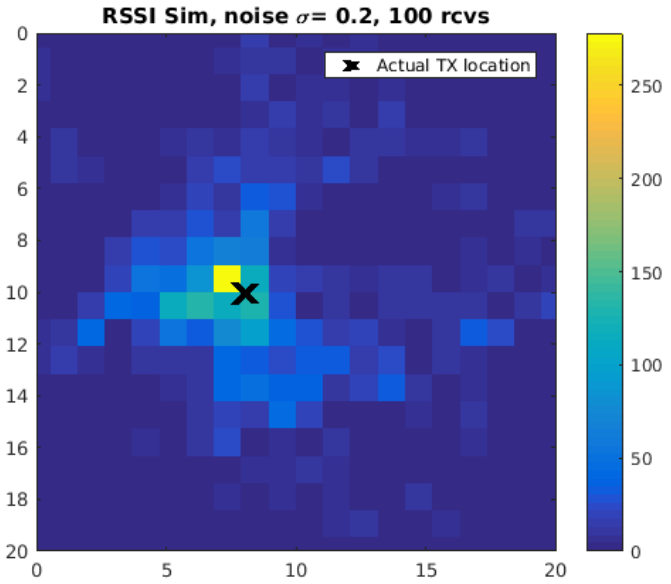


Fig. 3.7: Using power ratios with 100 observations and noise.

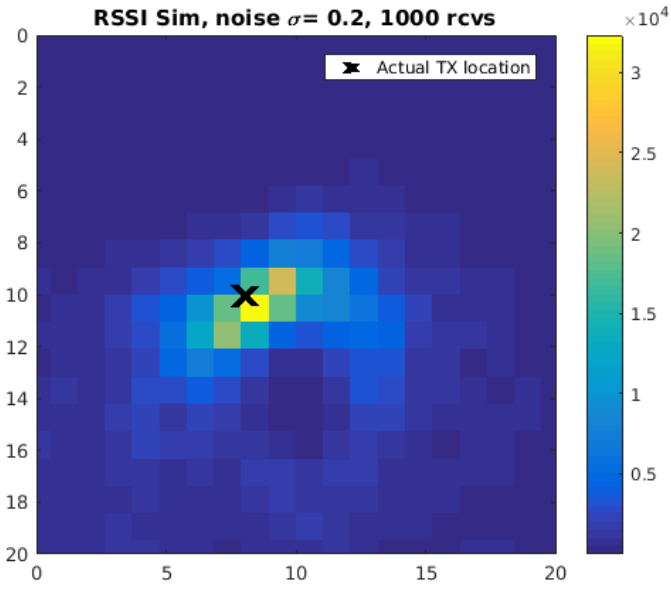


Fig. 3.8: Using power ratios with 1000 observations and noise.

3.4 Power Ratios with Loss Coefficient Noise

Another significant type of noise is differing loss coefficients. The Friis transmission equation is for free space and therefore has a loss coefficient of 2 as the exponent to the distance term, d. Previous work notes that differing loss coefficients occur frequently in the real world, and that the problem needs further research [6]. Allowing for differing loss coefficients, the equation for received power, P_r, then becomes

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^\alpha} .$$

The analysis is done the same as before, but with the introduction of noise in the model. Allowing for different loss coefficients in this way can negatively affect the location estimate.

Doing so in simulation degrades the quality of this estimate; however, it still performs well, as in figure 3.9. This simulation was run with loss coefficients having a Gaussian distribution with a mean of 3.0 and a standard deviation of 0.2.

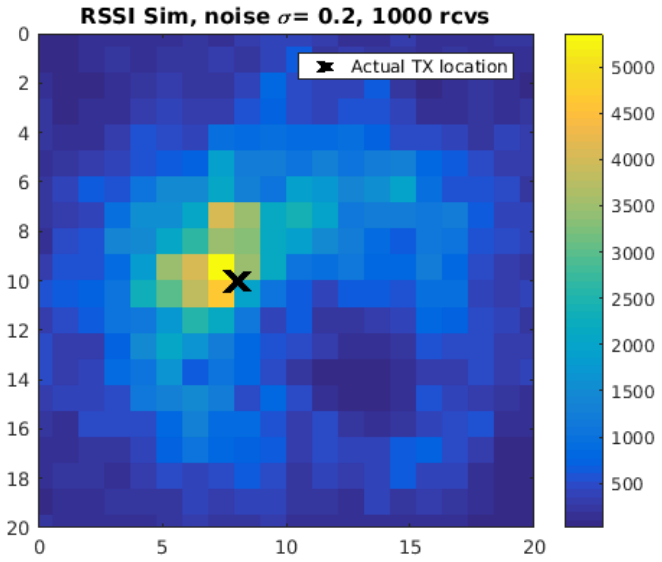


Fig. 3.9: Using power ratios with random loss coefficients and 1000 observations.

3.5 Trials on Real Data

The circles algorithm was applied to the real datasets. Below are shown heat maps representing two dimensional histogram data of circle intersections. Also provided are simplified diagrams with observations, the estimated location, and the true transmitter location marked.

Table 3.1: Circles algorithm errors for each dataset.

Dataset	Error in meters
sant1	15.76
sant2	7.92
quad3	108.94
upr3	116.26
aggr1	164.96
aggr4	1862.26

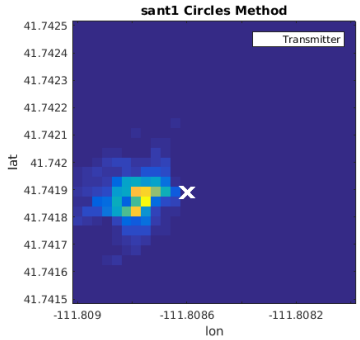


Fig. 3.10: sant1 circles heat map.

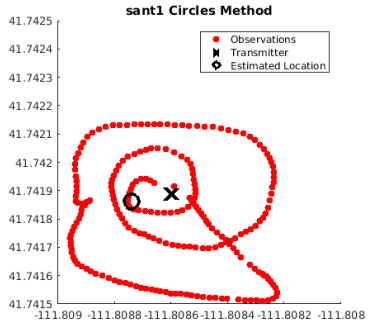


Fig. 3.11: sant1 circles diagram.

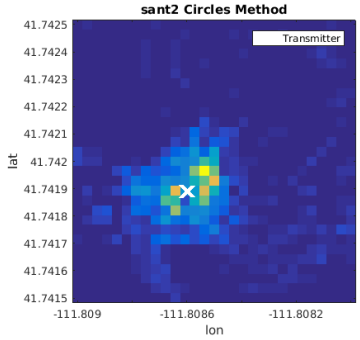


Fig. 3.12: sant2 circles heat map.

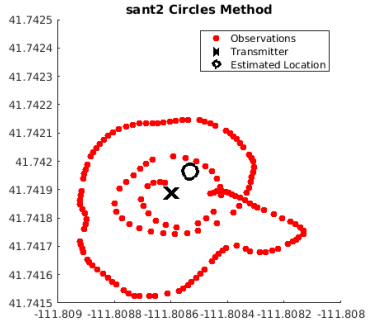


Fig. 3.13: sant2 circles diagram.

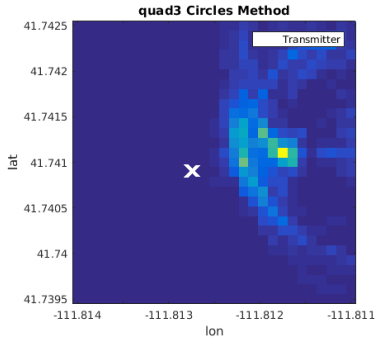


Fig. 3.14: quad3 circles heat map.

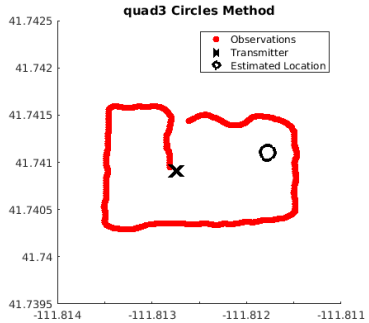


Fig. 3.15: quad3 circles diagram.

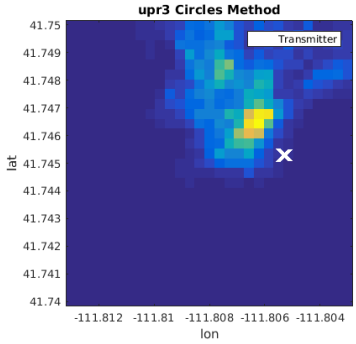


Fig. 3.16: upr3 circles heat map.

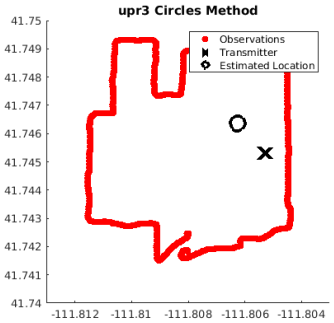


Fig. 3.17: upr3 circles diagram.

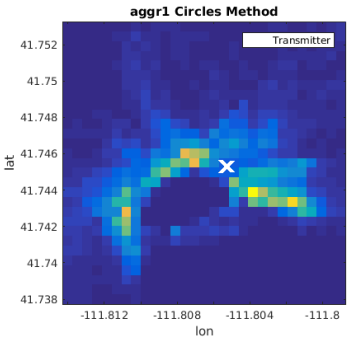


Fig. 3.18: aggr1 circles heat map.

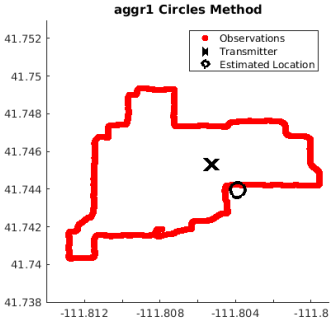


Fig. 3.19: aggr1 circles diagram.

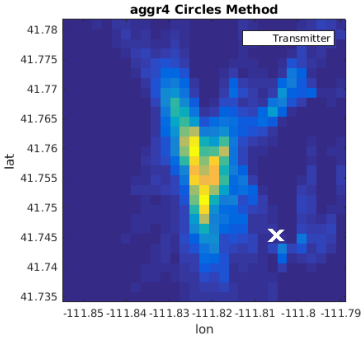


Fig. 3.20: aggr4 circles heat map.

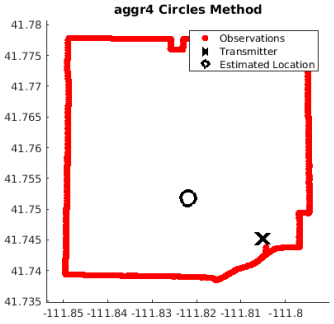


Fig. 3.21: aggr4 circles diagram.

3.5.1 Analysis of Real Data Results

The circles method is important because it is conceptually similar to triangulation, a commonly used technique for geolocation.

The trials on real data gave results that approximated the true locations but never in a completely convincing manner. The aggr1 and quad3 datasets were particularly poor, with results that were so scattered as to be not useful.

3.5.2 Possible Improvements

Instead of making a histogram of circle intersections, a Gaussian probability could be extruded along each locus of transmitter locations and then summed to the total probability grid. This may allow more information from pairs of observations to contribute to the overall estimate.

The algorithm, as described, first finds circles and then finds the intersections of those circles. Complexity could be reduced by solving for the circle intersections directly as a function of the three received powers and their positions.

CHAPTER 4

Binary-decision Cascading Probability (BCP)

In a simulation, let there be two observations, one transmitter, and no additive noise. Let the loss coefficient for the entire grid (call this term α) be 3. With a known loss coefficient, a circle of constant power ratio could be drawn between the two observations that would intersect the transmitter, as in figure 4.1. If the loss coefficient were guessed too low or too high, the locus circle would miss the transmitter, as in figures 4.2 and 4.3.

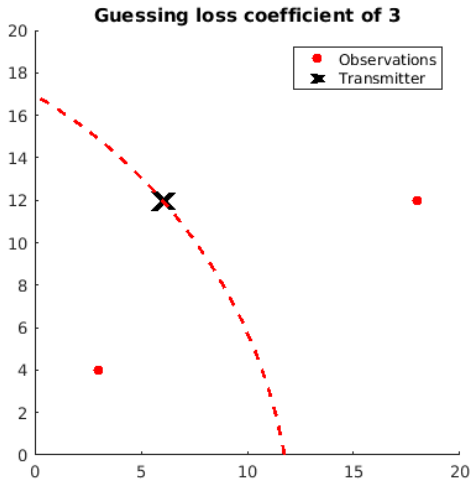


Fig. 4.1: Guessing a loss coefficient of 3.

The most significant assumption above is that the power loss coefficient is the same everywhere. The only real difference from the free-space model is that it is no longer limited to just being 2. In fact, this doesn't change the power-ratio circle equation presented before, besides needing to use a new value for constant k.

$$\frac{P_{r1}}{P_{r2}} = k^{\alpha}$$

It is important to notice that the k in the circle equation is still just k^2, and not k^{α}.

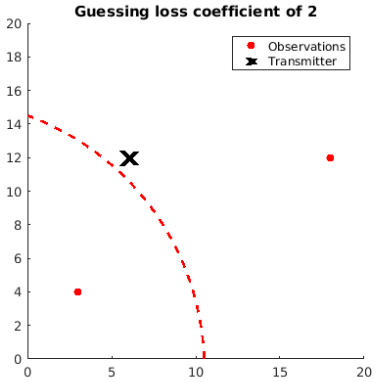


Fig. 4.2: Guessing a loss coefficient of 2.

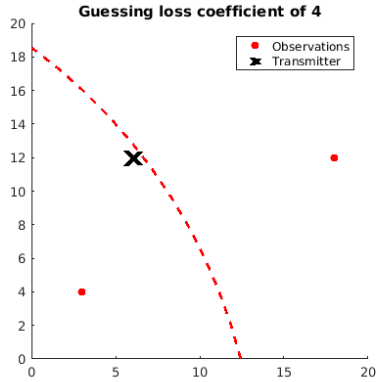


Fig. 4.3: Guessing a loss coefficient of 4.

The k^{α} relates the received powers. This square in the k^2 term then is not the loss coefficient α, it is just the method of representing a circle of possible transmitter locations. The same circle equation from before remains as

$$(x-a)^2 \; + \; (y-b)^2 \;\; = \frac{1}{k^2} \; \Big(\; (x-c)^2 \; + \; (y-d)^2 \; \Big) \; .$$

Making multiple guesses as to what the loss coefficient was in the grid many circles could be drawn, as in figure 4.4.

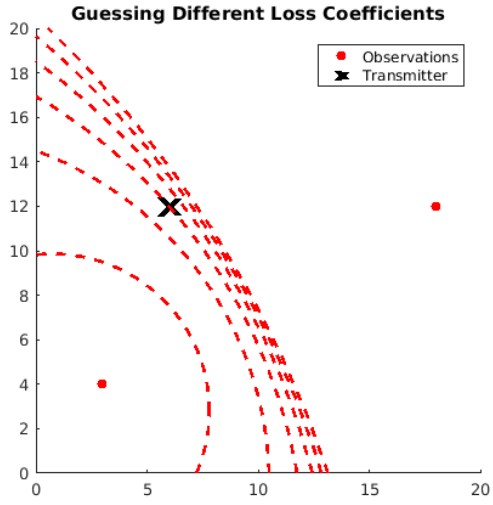


Fig. 4.4: Guessing loss coefficients as integers 1 through 6.

Increasing the guess for the loss coefficients across the grid, eventually approaches a line. This is depicted in figure 4.5 as the blue line.

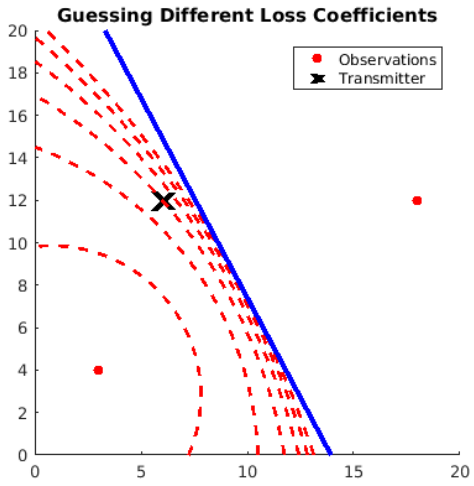


Fig. 4.5: Locus for infinite loss coefficient as the blue line.

In figure 4.5, it can be safely guessed that the receiver is on the left side of the blue line, where the blue line can be thought of as the loss coefficient being infinity. All finite loss coefficients would put the transmitter on the left side. The largest assumption here is that the loss coefficient is uniform across the entire grid. For now, however, employ this assumption to begin making statements about transmitter locations.

4.1 BCP Algorithm

Consider an algorithm for geolocation, referred to as the BCP algorithm or method, where BCP stands for binary-decision cascading probability, that works in the following manner.

1. Make a grid of probabilities, all equal to begin with, meaning each place is equally likely to have the transmitter located there.

2. Multiply all probabilities on the left side of the blue line in figure 4.5 by a factor (say 1.01) and divide all the probabilities on the other side of the line by that same factor.

3. Normalize the probability grid matrix.

4. Iterate through every combination of observations (of which there are $\binom{n}{2}$), updating the probability grid each time.

The general idea is that for any given pair of observations, the observation with the higher power is closer to the receiver most of the time. Step 2 then increases the overall estimate probability for grid locations closer to the stronger power measurement. If, over the entire set, this generalization holds true, the grid location containing the transmitter will have the highest probability of all grid locations.

In simulation, this method proves to be effective; however, improvements can readily be made.

Rather than using the if-loss-was-infinity line, consider using just the midpoint and drawing a line perpendicular to the line that connects observations. This concept is illustrated in figure 4.6.

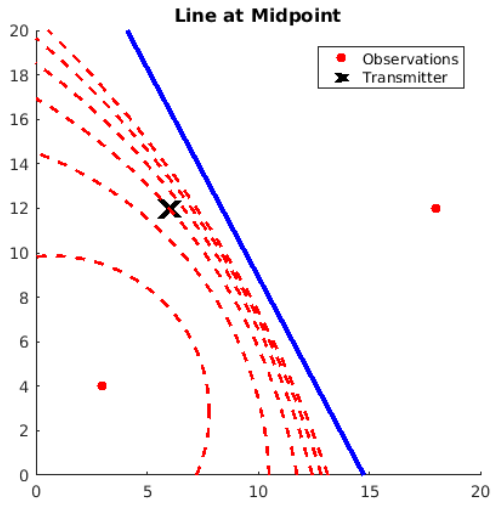


Fig. 4.6: The blue line intersects the midpoint between observations.

The results are similar to using the infinite-loss line, but prove to be a little more noise-resistant. Using the midpoint also has the benefits of being easier to compute, and of being conceptually simpler. This method is intuitively understood as asking "which point

is closer to the transmitter?" With the answer being (under these assumptions and in this algorithm) "the observation with the highest power."

With as few as 30 observations strong predictions can be made about transmitter location in the presence of additive noise. Increasing this to 100 observations can results in predictions that are exactly correct in simulation as seen in figures 4.7 and 4.9. In these figures, the brighter coloring suggests a higher likelihood of transmitter location.

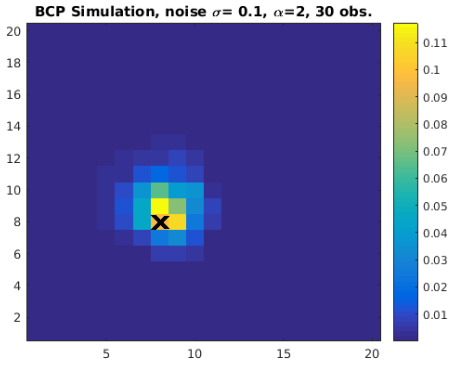


Fig. 4.7: BCP simulation results with 30 observations.

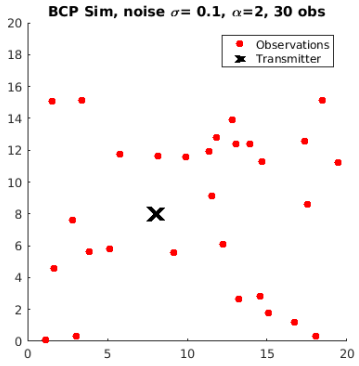


Fig. 4.8: BCP simulation setup with 30 observations.

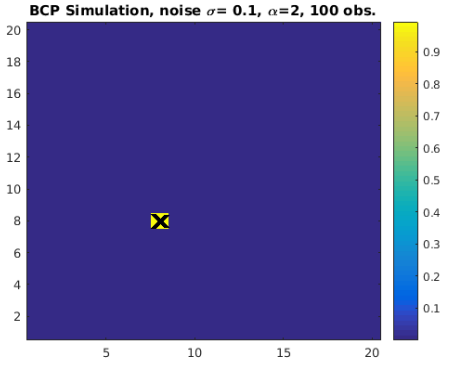


Fig. 4.9: BCP simulation results with 100 observations.

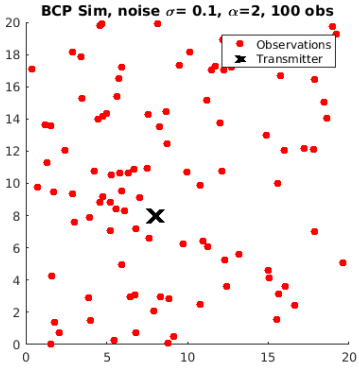


Fig. 4.10: BCP simulation setup with 100 observations.

However, when loss coefficients are not uniform, the estimates degrade in quality. Loss coefficients were assigned to observations from a Gaussian distribution with a mean of 3.0

and a variance of 0.04, and the BCP algorithm results are shown in figures 4.11 and 4.12. Sometimes the method provides accurate estimates despite the loss coefficient noise, but other times it estimates are biased.

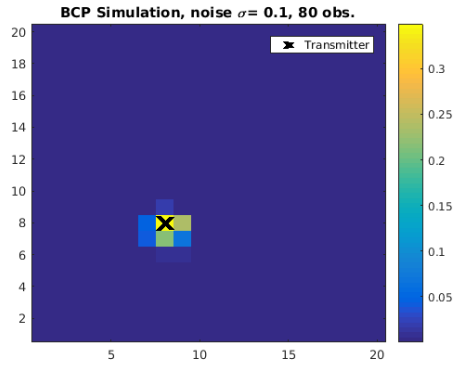


Fig. 4.11: BCP simulation results with random loss coefficients.

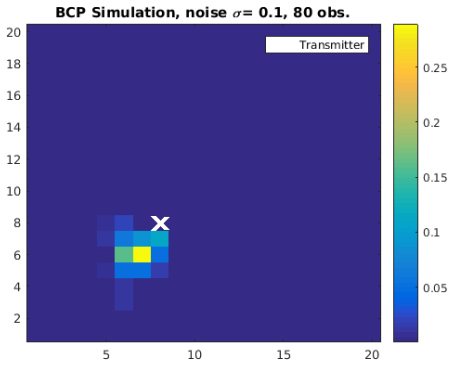


Fig. 4.12: BCP simulation results with random loss coefficients, showing bias.

4.2 Step-by-Step Visualization

A step-by-step example is provided to explain the BCP algorithm more completely. Included below are the first 9 iterations of a run of the BCP algorithm. The different symbols in the charts are significant. The black "X" marks the actual transmitter location. The two red circles represent the two observations being compared at that step of the algorithm. The larger red circle is the observation with the higher power measurement of the two. The dotted black line divides the area of the grid into two regions based on the midpoint of the two observations.

The color of the background grid represents the likelihood of the transmitter being located in that region. The more yellow an area is, the more likely that region is believed to contain the transmitter. The grid of likelihoods is updated from step to step, so that step 1 initializes the probability grid, step 2 updates that grid, and so on.

The images in figure 4.13 show the first 9 steps of the algorithm applied to the sant1 dataset in order to demonstrate the behavior of the BCP algorithm. The completed run (all steps executed) result can be seen in section 4.3.

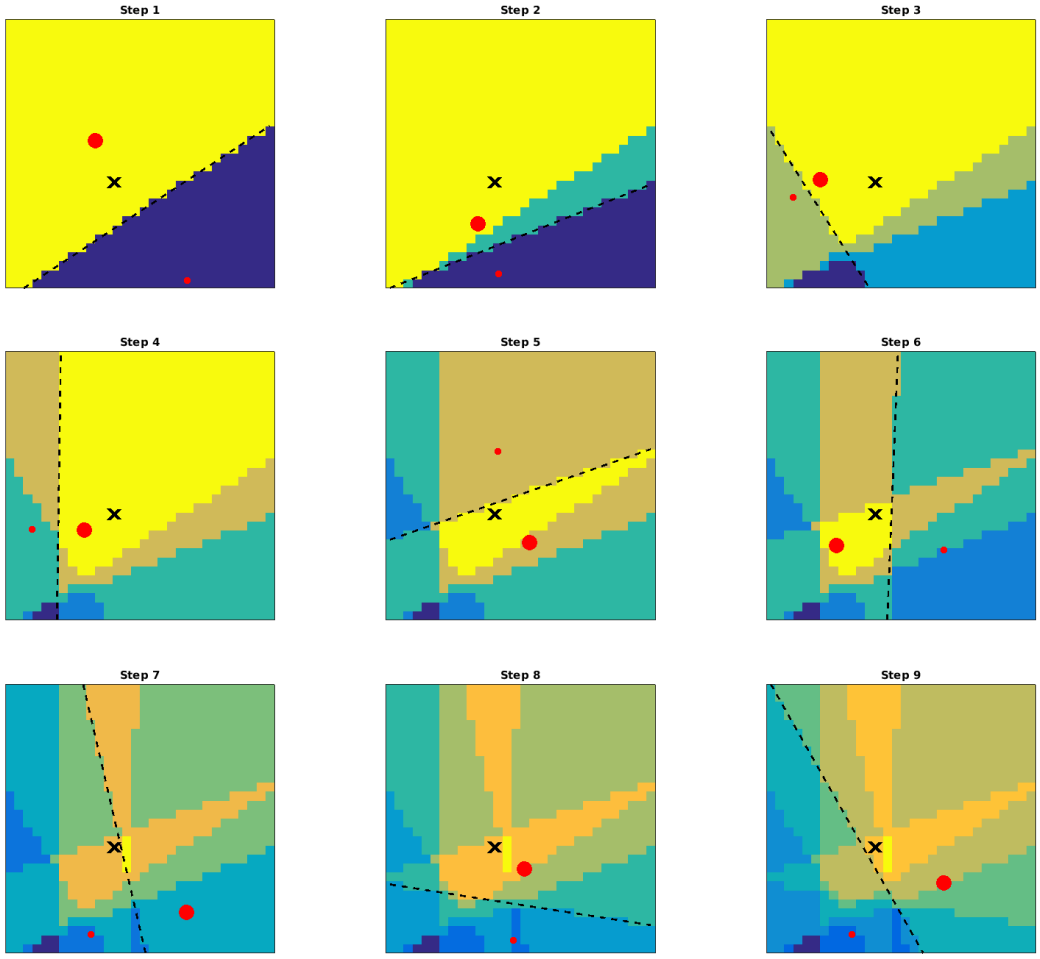


Fig. 4.13: The BCP Algorithm applied to the sant1 dataset, showing the first 9 steps of the algorithm updating the probability grid. Step 7 is the only step which incorrectly updates the grid of the 9 displayed here. The "X" marks the actual transmitter location, the red circles are the two observations being compared with the larger circle representing the observation with a higher power. The dotted line divides the grid along the midpoint between the two observations. The grid coloring represents the estimate of transmitter location, with yellow regions representing areas that are believed to be the most likely place to contain the transmitter.

4.3 Trials on Real Data

The BCP algorithm was applied to real datasets. Presented below are a table of errors in meters and heat maps that represent probability mass for transmitter locations.

Table 4.1: BCP algorithm errors for each dataset.

Dataset	Error in meters
sant1	2.50
sant2	5.46
quad3	41.46
upr3	181.36
aggr1	345.72
aggr4	841.77

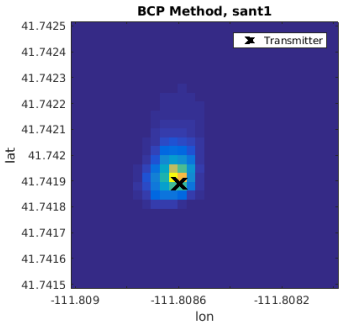


Fig. 4.14: sant1 BCP heat map.

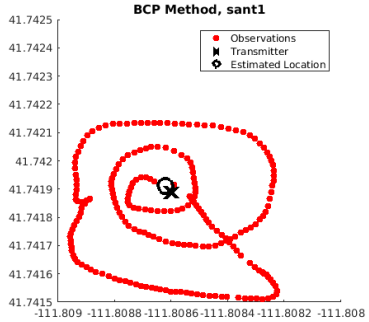


Fig. 4.15: sant1 BCP diagram.

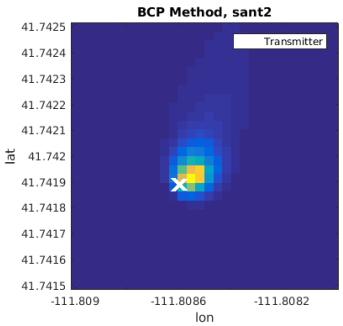


Fig. 4.16: sant2 BCP heat map.

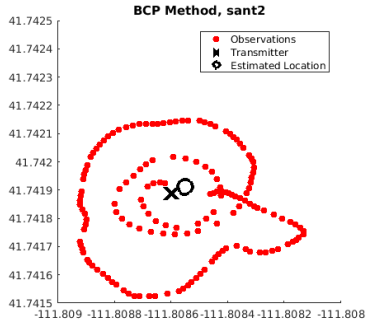


Fig. 4.17: sant2 BCP diagram.

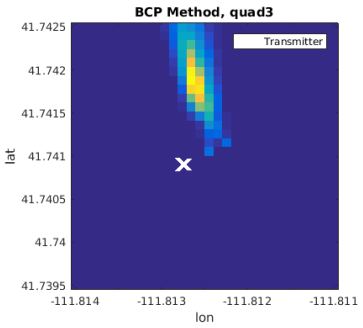


Fig. 4.18: quad3 BCP heat map.

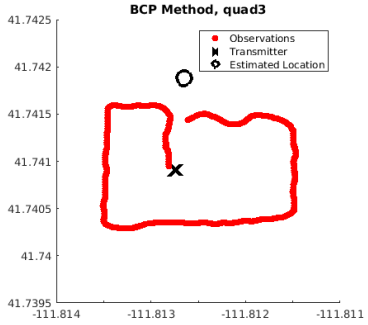


Fig. 4.19: quad3 BCP diagram.

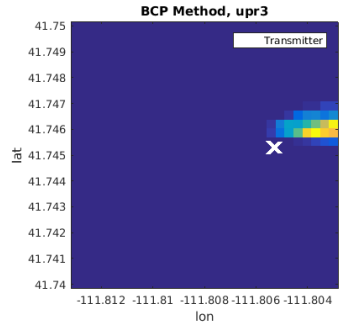


Fig. 4.20: upr3 BCP heat map.

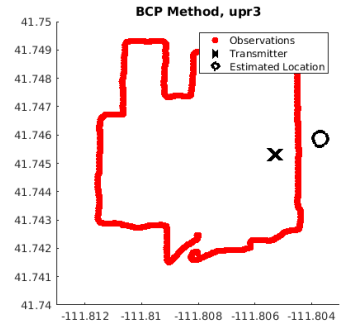


Fig. 4.21: upr3 BCP diagram.

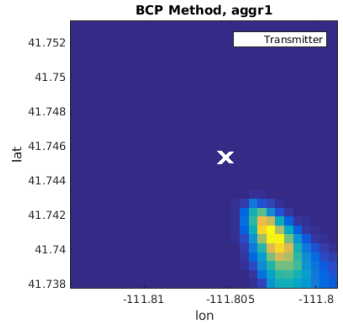


Fig. 4.22: aggr1 BCP heat map.

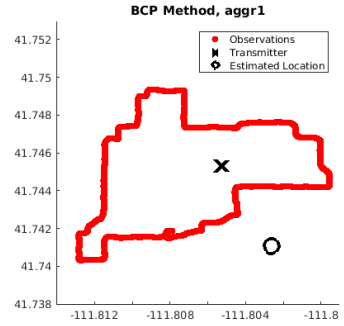


Fig. 4.23: aggr1 BCP diagram.

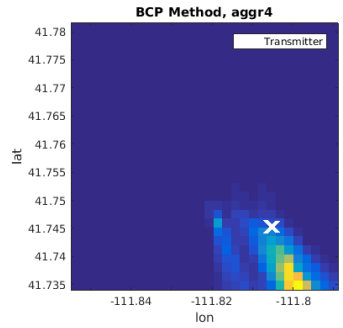


Fig. 4.24: aggr4 BCP heat map.

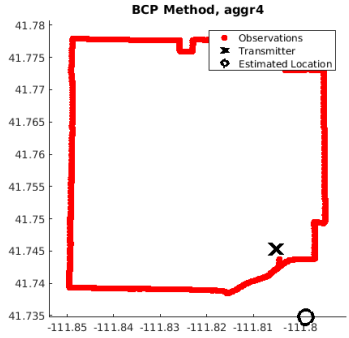


Fig. 4.25: aggr4 BCP diagram.

4.3.1 Analysis of Real Data Results

The BCP method is probably the most intuitive among the presented methods. It is similar to the concept of the "hotter/colder" child's game.

The BCP algorithm performs well on the datasets that were taken with direct line of sight (sant1, sant2, quad3) but poorly on the datasets without (upr3, aggr1, aggr4). Another difference between these two groups is that the direct line of sight group was transmitting from walkie talkies which transmit at different powers and frequencies than FM radio stations. While it is difficult to say what this method's weakness is, it is clear that it performs inconsistently.

4.3.2 Possible Improvements

A more algorithmic step size could be determined in order to give the desired "spread" in the final probability grid. Alternatively, the ratio of power received between observations could be used to determine a smarter step size to take. For instance, if the power ratio were quite large, the comparison would be more certain, and the probabilities could be increased by a larger scale.

It also may be beneficial to exclude comparisons for observations that are almost co-located since their received powers may be very similar. The comparison between two very similar power measurements is more susceptible to additive noise changing the binary result of the comparison.

CHAPTER 5

3-Parameter Method

Begin again with the Friis transmission equation, with α_i as the individual loss coefficients for each observation to form

$$P_i = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d_i^{\alpha_i/2}},$$

with d_i being defined as the squared distance to the i'th observation, or

$$d_i \triangleq (x_i - x_0)^2 + (y_i - y_0)^2 .$$

Here the notation is x_i and y_i for the location of the i'th observation and x_0 and y_0 for the transmitter location. Now eliminate terms that are not concerns in this model (such as gains and wavelength) to form

$$P_i = \frac{P_t}{d_i^{\alpha_i/2}} = \frac{P_t}{((x_i - x_0)^2 + (y_i - y_0)^2)^{\alpha_i/2}} .$$

Combine the terms on one side of the equation to form a new equation equal to zero.

$$P_i d_i^{\alpha_i/2} - P_0 = 0$$

Call this new function J_i, the cost function for the i'th observation.

$$J_i(x_0, y_0, P_0) = P_i d_i^{\alpha_i/2} - P_0$$

The next step is to minimize the magnitude of our cost function. It is easier to minimize the magnitude squared, and every J_i term should contribute to the overall cost, so sum J_i^2 $\forall i$. The new cost function that includes every individual observation cost is

$$J(x_0, y_0, P_0) = \sum_{\forall i} {J_i}^2 .$$

Seek to minimize our cost function in order to get it as close as possible to zero, the value it theoretically should be. This model allows for changes in x_0, y_0, and P_0 but leaves the loss coefficients α_i as known constants.

$$\underset{x_0,y_0,P_0}{\arg\min}\, J(x_0, y_0, P_0) = \sum_{\forall i} [P_i((x_i - x_0)^2 + (y_i - y_0)^2)^{\alpha_i/2} - P_0]^2$$

Determine the minimum using Newton's method. Minimizing or maximizing a function can be accomplished by looking for areas where the derivative of the function is zero. In other words, find the roots of the derivative.

For a function of multiple variables, the first derivative, $J'(\mathbf{x})$, turns into a gradient, and the second derivative, $J''(\mathbf{x})$, turns into a Hessian matrix. The update equation is then

$$\mathbf{x}^{[n+1]} = \mathbf{x}^{[n]} - (\nabla^2 J(\mathbf{x}^{[n]}))^{-1} \nabla J(\mathbf{x}^{[n]}) \tag{5.1}$$

where $\nabla J(\mathbf{x})$ is the gradient, and $\nabla^2 J(\mathbf{x})$ is the Hessian.

With the current model, three parameters are used to minimize the cost function. This results in a 3×1 gradient and a 3×3 Hessian.

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial x_0} \\ \frac{\partial J}{\partial y_0} \\ \frac{\partial J}{\partial P_0} \end{bmatrix}$$

$$\nabla^2 J = \begin{bmatrix} \frac{\partial^2 J}{\partial x_0 \partial x_0} & \frac{\partial^2 J}{\partial x_0 \partial y_0} & \frac{\partial^2 J}{\partial x_0 \partial P_0} \\ \frac{\partial^2 J}{\partial y_0 \partial x_0} & \frac{\partial^2 J}{\partial y_0 \partial y_0} & \frac{\partial^2 J}{\partial y_0 \partial P_0} \\ \frac{\partial^2 J}{\partial P_0 \partial x_0} & \frac{\partial^2 J}{\partial P_0 \partial y_0} & \frac{\partial^2 J}{\partial P_0 \partial P_0} \end{bmatrix}$$

5.1 Gradient and Hessian

The gradient for the cost function $J(x_0, y_0, P_0)$ consists of three partial derivatives, and

the Hessian for the cost function $J(x_0, y_0, P_0)$ consists of nine second-partial derivatives. Both the gradient and Hessian can be found in appendix A.1.

When the loss coefficients are restricted to 2 ($\alpha_i = 2, \forall i$), the gradient and Hessian simplify greatly. This cancels out many of the more complicated terms and leaves the gradient and Hessian that can be found in appendix A.2. Since initial guesses for the loss coefficients are not available in the datasets, this initialization, $\alpha_i = 2, \forall i$, will be commonly used in the described algorithms and simulations.

5.2 3-Parameter Method in Simulation

Use Newton's method to obtain a location and power estimate by choosing an initial guess somewhere nearby the observations and with an arbitrary power guess. Iterate on the estimate using (5.1) until the change from iteration to iteration is sufficiently small, or for some set number of steps.

Using this 3-parameter method in an ideal simulation works well, as can be seen in figure 5.1, where the blue diamonds represent the location estimate at different steps in the Newton's method optimization. For this simulation, ten steps were taken. The generated observations each had loss coefficients of exactly 2, and their measured powers had no additive noise. The simulation was, in this way, ideal.

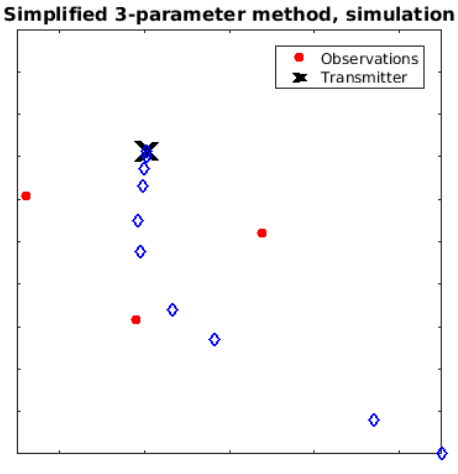


Fig. 5.1: 3-parameter method simulation with no noise.

Introducing noise into the simulation causes the final estimate using the Newton's method algorithm to not correctly identify the true transmitter location, as seen in figure 5.2. In this simulation, loss coefficients were assigned randomly to be 2.39, 2.61, and 2.47. Zero mean Gaussian noise was added to each received power with a standard deviation of 0.1.

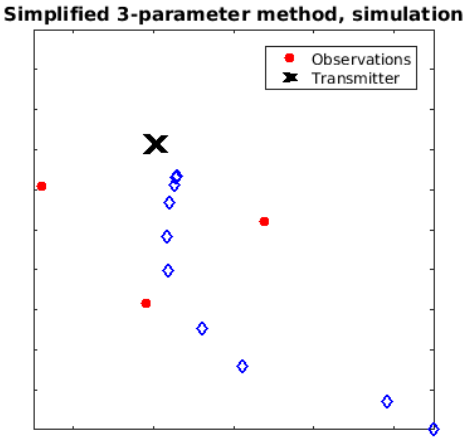


Fig. 5.2: 3-parameter method simulation with noise.

With enough observations, the effects of noise can be somewhat overcome, as seen in figure 5.3.

In these simulations, the original estimates for individual loss coefficients were taken to be 2. This allows for the use of the simplified gradient and Hessian equations and empirically has had little effect on the overall quality of the final estimate. This version of the 3-parameter method will be referred to as the simplified 3-parameter method.

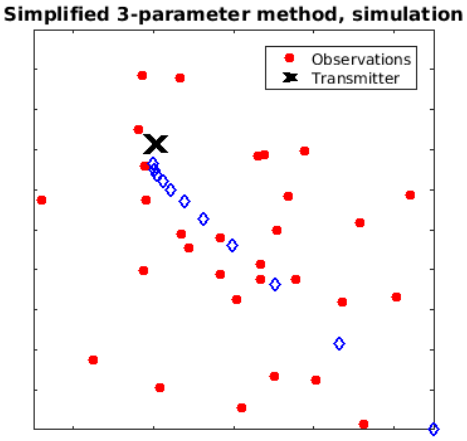


Fig. 5.3: 3-parameter method simulation with noise and additional observations.

5.3 Estimating Loss Coefficients

After using the 3-parameter method to make a location estimate it could be useful to determine which loss coefficients would fit the data to this estimate. The goal here is to estimate a loss coefficient for each individual observation.

Running the simplified version of the 3-parameter method results in a transmitted power estimate. If a power estimate is not available, it can be obtained by taking the mean of the received powers times the distances raised to the loss coefficients as

$$\hat{P}_0 = \frac{1}{n}\sum_{i}^{n}(P_i d_i^{\alpha_i})\,.$$

With the power estimate now available, solve for the individual loss coefficients as

$$\alpha_i = \frac{\log \frac{\hat{P}_0}{P_i}}{\log d_i}\,.$$

These loss coefficients can then be used in further analysis or in other algorithms.

5.4 Trials on Real Data

The simplified 3-parameter (3p) method algorithm was applied to real datasets. Below are diagrams with the final estimated location displayed alongside the observations and true transmitter locations.

Table 5.1: Simplified 3-parameter algorithm errors for each dataset.

Dataset	Error in meters
sant1	5.09
sant2	2.19
quad3	19.47
upr3	128.74
aggr1	107.05
aggr4	188.46

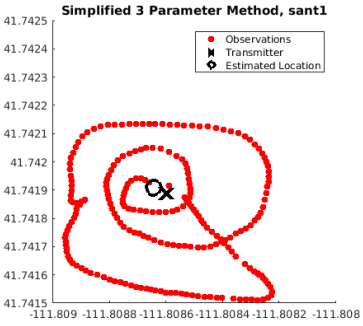


Fig. 5.4: sant1 3p diagram.

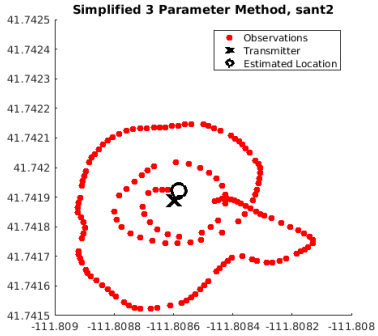


Fig. 5.5: sant2 3p diagram.

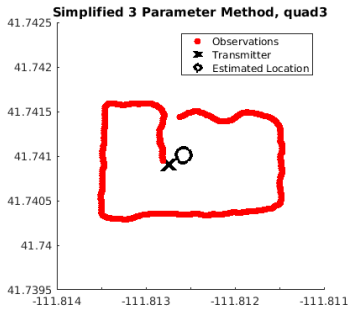


Fig. 5.6: quad3 3p diagram.

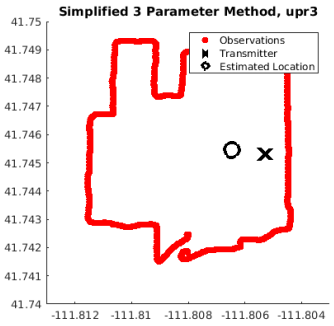


Fig. 5.7: upr3 3p diagram.

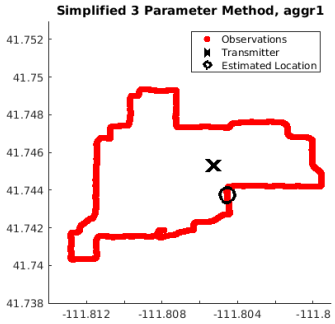


Fig. 5.8: aggr1 3p diagram.

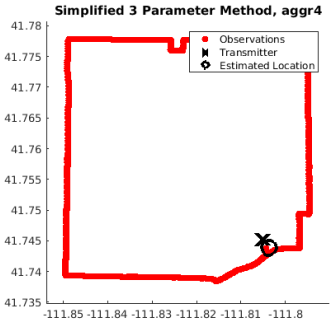


Fig. 5.9: aggr4 3p diagram.

5.4.1 Analysis of Real Data Results

The simplified 3-parameter method algorithm works well for the real datasets. The estimates are all reasonable, but the estimates for the datasets without direct line of sight (upr3, aggr1, and aggr4) are of lower quality than the others.

The simplified 3-parameter algorithm is significant because it uses the simplest model of the problem, ignoring many of the different complications inherent in the setup. It also runs very quickly and produces a plausible estimate with very few computations, regardless of starting location (anywhere on the grid of observations).

5.4.2 Possible Improvements

Taking into account more complex models is addressed later in this thesis. Improvements to this algorithm, without changing it fundamentally, can mainly only be made in regards to the implementation and efficiency.

CHAPTER 6

Subset Method

Returning to the simplified 3-parameter simulation, consider the case with three observations where loss coefficients are all identically two. The simplified 3-parameter method produces a location estimate under the assumption that all the loss coefficients are uniformly two. In simulation, as seen in the previous section, this produces a perfect location estimate, as in figure 6.1 when there is no additive power noise.

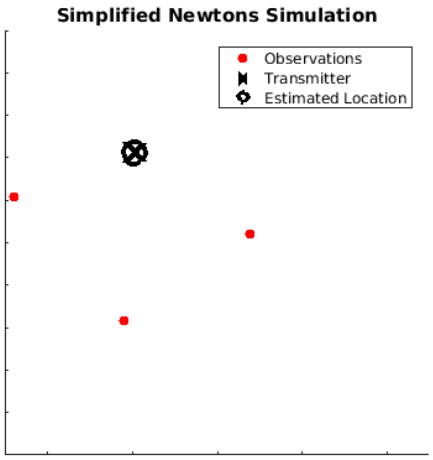


Fig. 6.1: Simplified 3-parameter method; loss coefficients are all 2.

Now consider the case where the loss coefficients are not strictly two. The estimates produced by the simplified 3-parameter method are likely to be close to the true transmitter location as long as the actual loss coefficients are distributed closely around some mean. Notice in figures 6.2, 6.3, and 6.4 that the estimated locations are close to the actual transmitter location, even though the loss coefficients in the simulation were not two, as assumed by the simplified 3-parameter method. However, if loss coefficients are distributed distant from each other (highly varied) the resulting estimate is poor, as can be seen in

figure 6.5.

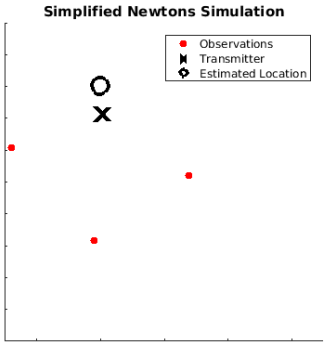


Fig. 6.2: Simplified 3-parameter method; true loss coefficients are all 3.

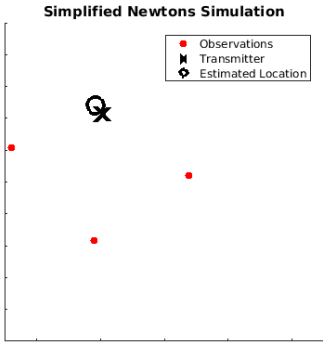


Fig. 6.3: Simplified 3-parameter method; true loss coefficients are all 4.

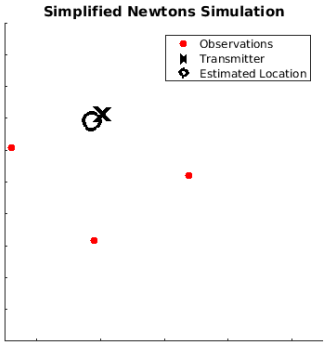


Fig. 6.4: Simplified 3-parameter method; true loss coefficients are all 5.

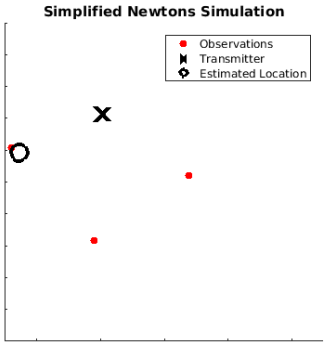


Fig. 6.5: Simplified 3-parameter method, true loss coefficients are 2, 3, and 4.

6.1 Subset Algorithm

The general idea of the subset algorithm is to repeat the 3-parameter method using different subsets of data and save each final estimate. As long as the loss coefficients of each observation are close to the same mean, the estimate should acceptable. By repeating the analysis on different subsets, the average quality of the estimates should be good.

A simulation of this method, shown as both individual estimates and a heat map histogram is shown in figures 6.6 and 6.7. In this simulation, the subset size was 3 samples,

the minimum amount needed for the cost function to be exactly determined. Also note that the number of observations was far fewer than what is present in the real datasets, in order to help visualize the behavior of the algorithm.

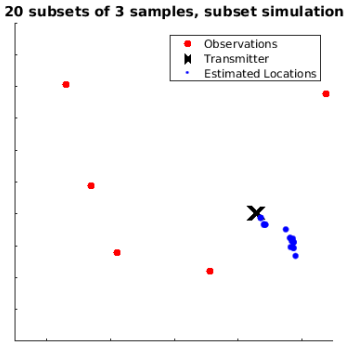


Fig. 6.6: Location estimates from different subsets, using the simplified 3-parameter method.

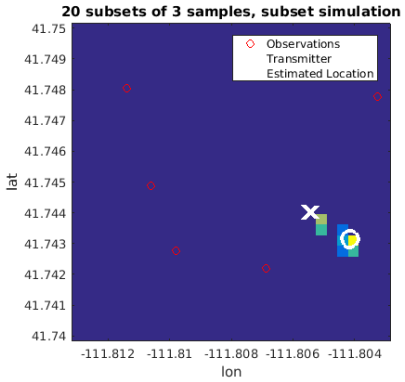


Fig. 6.7: Location estimates from different subsets, using the simplified 3-parameter method.

6.2 Trials on Real Data

The subset algorithm was applied to the real datasets. Below are heat maps that represent two-dimensional histograms of location estimates.

Subset sizes were chosen to be three samples each, the minimum number of samples needed to exactly determine the cost function, which has 3-parameters.

Table 6.1: Subset algorithm errors for each dataset.

Dataset	Error in meters
sant1	13.08
sant2	20.45
quad3	1.65
upr3	30.26
aggr1	64.92
aggr4	179.55

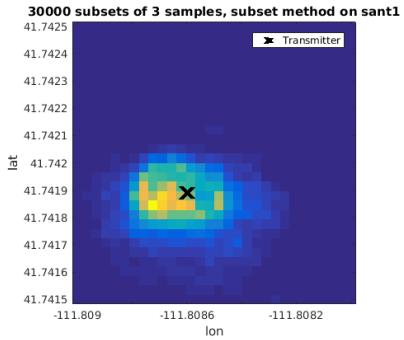


Fig. 6.8: sant1 subset heat map.

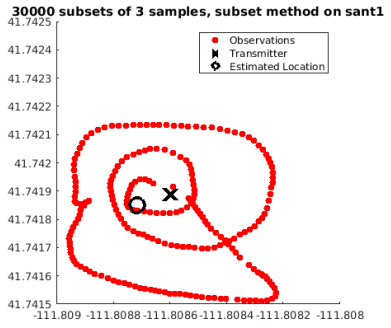


Fig. 6.9: sant1 subset diagram.

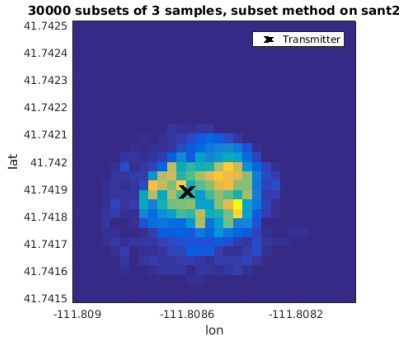


Fig. 6.10: sant2 subset heat map.

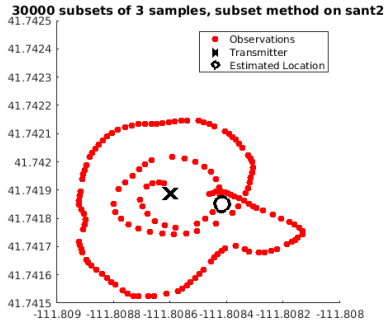


Fig. 6.11: sant2 subset diagram.

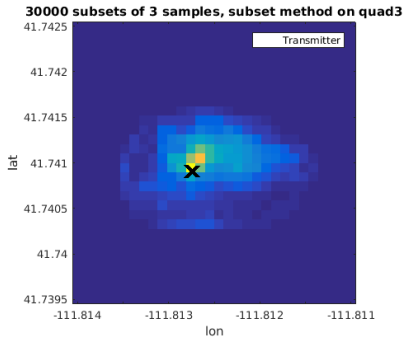


Fig. 6.12: quad3 subset heat map.

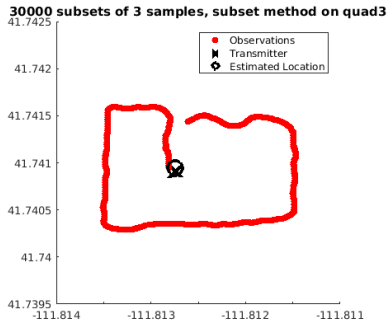


Fig. 6.13: quad3 subset diagram.

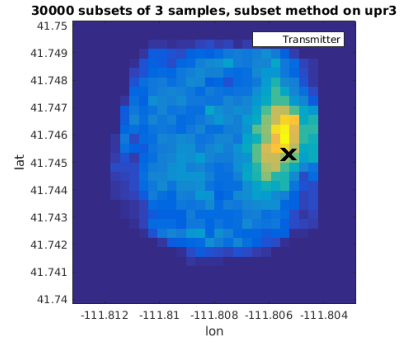


Fig. 6.14: upr3 subset heat map.

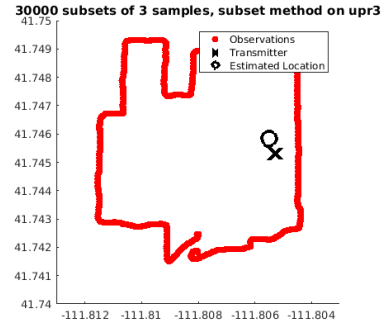


Fig. 6.15: upr3 subset diagram.

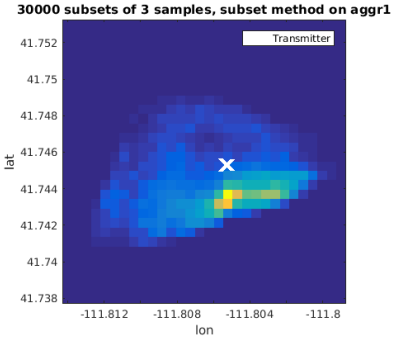


Fig. 6.16: aggr1 subset heat map.

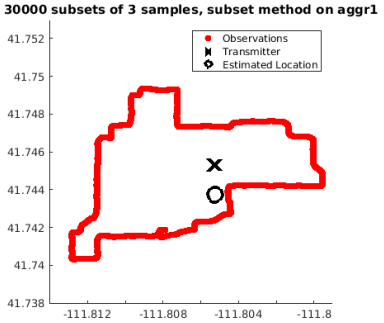


Fig. 6.17: aggr1 subset diagram.

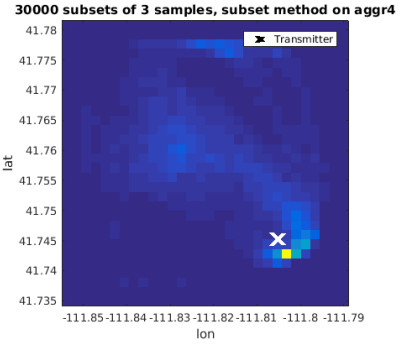


Fig. 6.18: aggr4 subset heat map.

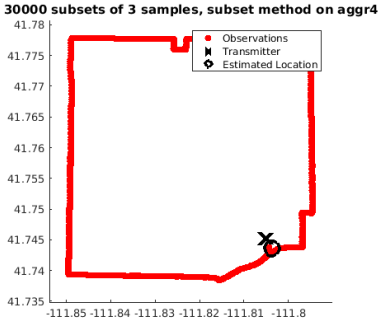


Fig. 6.19: aggr4 subset diagram.

6.2.1 Analysis of Real Data Results

The subset method produces good results on every dataset, with especially impressive results on the quad3 dataset where it is able to determine the true location as closely as the histogram quantization allows. It also does surprisingly well on the upr3 dataset, which did not have direct line of sight for a majority of the measurements in addition to having frequent shadowing.

The sant datasets still performed reasonably well, but suffered from the quantization effect of the histogram. While the heat maps appear fine, the error the subset method produces is somewhat large compared to the relatively small size of the observation grid.

To demonstrate this effect, re-run the algorithm on the sant2 dataset, with a histogram bin size equal to that of the upr3 dataset. As can be seen in figure 6.20, the true transmitter location is (correctly) in the highest bin of the histogram.

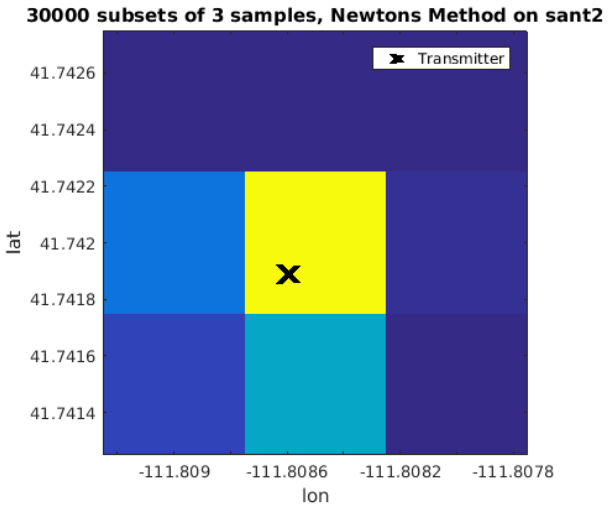


Fig. 6.20: Subset algorithm on the sant2 dataset with 37 meters per bin.

6.2.2 Possible Improvements

The subset method could be improved by developing a more algorithmic way to choose histogram bin spacing. For example, a scheme that quantized results based on real-world metrics such as 10×10 meter blocks may quantize results in a way that avoids the problems

inherent in the algorithm for small-scale observation grids.

Another possible improvement would be to avoid choosing as a subset three observations that are almost exactly co-located. This may eliminate some of the poorer estimates from the histogram and allow for more of the high-quality estimates to contribute.

CHAPTER 7

Clustering Method

Previous models did not allow for the discovery of the loss coefficients at the same time a location estimate was being made. By introducing the loss coefficients as parameters in the cost functions, Newton's method should be able to adjust the values until they reach some optimal point.

If each observation had its own individual loss coefficient parameter, the resulting system of equations would be underdetermined. Instead, assume that the observations can be clustered into k individual groups, where each group has its own loss coefficient. Clustering can be done spatially, or by estimating the loss coefficients using a previous method as discussed in section 5.3.

7.1 K-means clustering

The general approach to clustering in this chapter is to estimate the loss coefficients as in section 5.3 and then to group the observations using k-means clustering. K-means clustering iteratively classifies points into k different groups by selecting random centers and then re-evaluating the centers using the points that are nearest to each center. K-means is one of the most often used clustering algorithms [16].

7.2 6-parameter Method

For brevity, introduce again a distance squared term as

$$d_i \triangleq (x_i - x_0)^2 + (y_i - y_0)^2.$$

The new cost function associated with this model has six parameters,

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_0 - P_i d_i^{j(i)/2})^2$$

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} P_0^2 + P_i^2 d_i^{j(i)} - 2P_0 P_i d_i^{j(i)/2}\,, \tag{7.1}$$

where $j(i)$ selects which loss coefficient group is to be used,

$$j(i) = \begin{cases} \alpha_1 & i \in \text{Group 1} \\ \alpha_2 & i \in \text{Group 2} \\ \alpha_3 & i \in \text{Group 3}\,. \end{cases}$$

Newton's method is used to minimize the cost function in (7.1). The corresponding gradient and Hessian are found in appendix A.3. The entire chain of processing is to estimate location and power using the simplified 3-parameter method, estimate the loss coefficients, group the observations using k-means clustering, and finally minimize (7.1) using Newton's method.

Running on real data quickly suggests that this approach is not viable in every case. As seen in figure 7.1, the Newton iterations take the location estimate off the observation grid.

Rather than presenting the different results on each of the datasets, examine the cost function 7.1 and evaluate its structure.

As stated earlier, the distances used are pseudo distances obtained by approximating the latitude and longitudes as a uniformly spaced grid. This results in the squared distance term, d_i, being less than one. The two right terms in (7.1), $P_i^2 d_i^{j(i)}$ and $2P_0 P_i d_i^{j(i)/2}$, can be effectively driven to zero by allowing the loss coefficients to grow arbitrarily high. The leftmost term (P_0^2) can be driven to zero directly, as it is a free parameter of the system. This essentially creates a minimum that does not strictly lie near the true transmitter

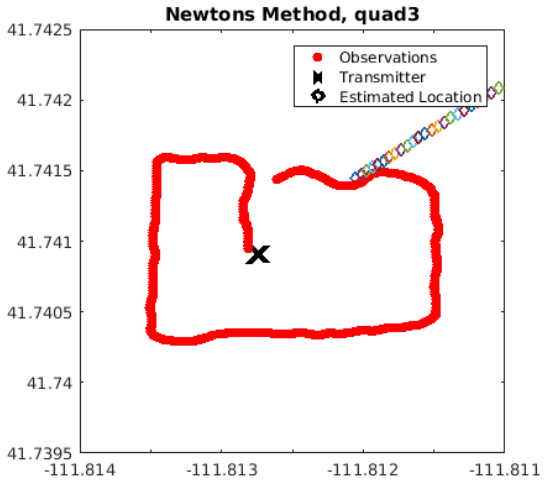


Fig. 7.1: 6-parameter method on the quad3 dataset.

location. In simulation and running on real datasets, this results in Newton steps that do not necessarily approach the true transmitter location.

By switching to an actual Cartesian grid (mapping to a meter grid as discussed in section 2.6) the distances are all greater than 1 ($d_i > 1, \forall i$). In this case, the two right terms, ($P_i^2 d_i^{j(i)}$ and $2P_0 P_i d_i^{j(i)/2}$) can be minimized by allowing the loss coefficients to grow towards negative infinity. The mapping does not address the problems inherent in (7.1).

7.3 6-parameter Method with an Alternative Cost Function

Address the problems of (7.1) by changing the cost function. Starting from the original model,

$$P_i = \frac{P_0}{d_i^{\alpha_i/2}},$$

leave the distance term in the denominator, resulting in

$$\frac{P_0}{d_i^{\alpha_i/2}} - P_i = 0.$$

Building a cost function in the same manner as before, sum the cost of each loss squared, and allow the loss coefficients to be clustered as

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_i - P_0 d_i^{-j(i)/2})^2$$

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} P_i^2 + P_0^2 d_i^{-j(i)} - 2P_0 P_i d_i^{-j(i)/2}. \tag{7.2}$$

In comparison with (7.1), (7.2) should not be able to drive the leftmost squared term, P_i^2, to zero directly, perhaps addressing the issue inherent in the previous model.

Again, optimization is done using Newton's method, and the relevant gradient and Hessian can be found in appendix A.4. Running on real data quickly demonstrates that this method diverges as well, as in figure 7.2. The Newton steps begin close to the true location (due to using the simplified 3-parameter estimate as a starting location) but quickly leave the observation grid.

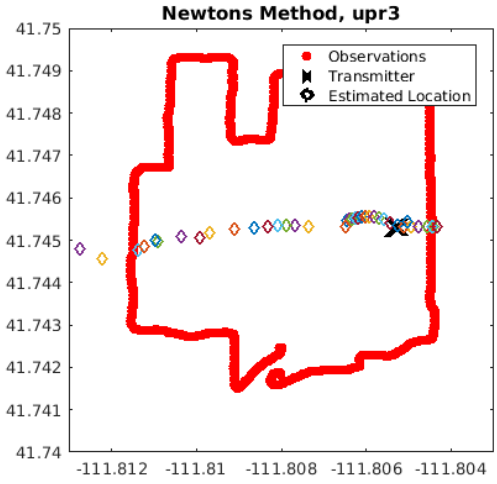


Fig. 7.2: 6-parameter method on the upr3 dataset, alternative cost function.

The largest problem immediately obvious with this method is that singularities in the cost function are created at every observation. Since the distance term is in the denominator, any location near the observation will have a cost function approaching infinity. Projecting the cost function onto a 2-dimensional plane of location, these singularities are

easily visualized for the datasets, as in figure 7.3. The logical conclusion to draw is that the singularities create regions that corrupt the cost function anywhere near an observation, making it difficult to find the transmitter location.

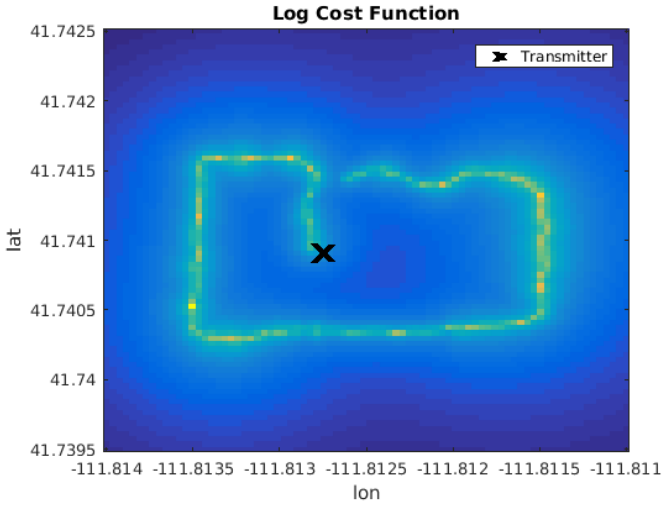


Fig. 7.3: 6-parameter log cost function on the quad3 dataset, alternative cost function.

7.4 7-parameter Newton's Alternative Cost Function

Consider another cost function, one that models the possibility of a constant noise floor, adding a 7th parameter to (7.2). The new cost function in (7.3) allows for an additive noise floor term to be added to each observation. This parameter models the electromagnetic interference present in every measurement from other transmitters and radiating bodies.

$$J(x_0, y_0, P_0, N_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_i - P_0 d_i^{-j(i)/2} - N_0)^2$$

$$J = \sum_{\forall i} P_i^2 + N_0^2 + P_0^2 d_i^{-j(i)} - 2P_i N_0 - 2P_0 P_i d_i^{-j(i)/2}(N_0 - P_i) \tag{7.3}$$

The performance of this method is similar to that of the 6-parameter method with the alternative cost function. Singularities caused by the distance term in the denominator again corrupt the cost function and result in diverging steps, as in figure 7.4.

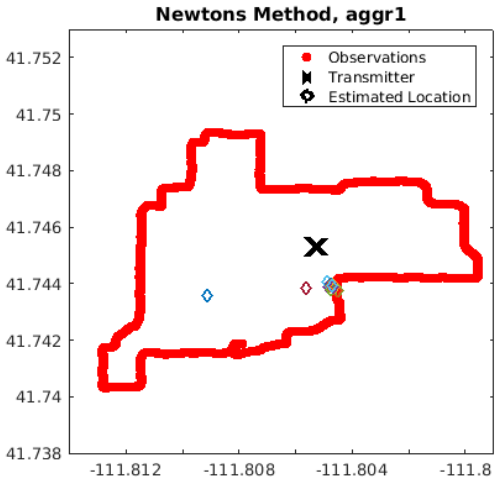


Fig. 7.4: 7-parameter method on the aggr1 dataset, alternative cost function.

The cost function, projected onto the location plane, is again plotted in order to visualize the singularities in figure 7.5.

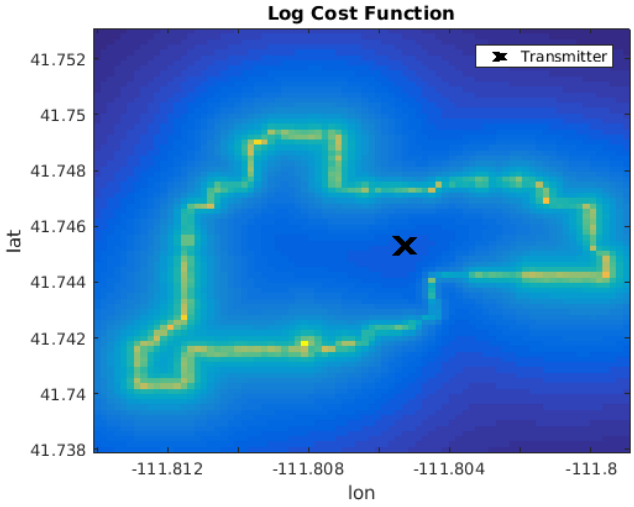


Fig. 7.5: 7-parameter log cost function on the aggr1 dataset, alternative cost function.

The constant noise term does little to correct any flaws in the previous cost functions and behaves similarly to the cost function in (7.2). Using a Cartesian mapping to produce distances greater than one does not affect the overall performance of this method. Rather, it changes the ending state of the parameters being estimated and changes the visualization

of the cost function. The resulting estimate is no better than when using the latitude and longitude as a uniform grid.

7.5 Backtracking Line Search and Log Barriers

The model in (7.2) imposes no restriction on the loss coefficients or the power transmitted. In the physical world, the transmitted power is always positive and the loss coefficients should also be positive. Incorporate these restrictions using a log barrier in the cost function. As practical values for loss coefficients, restrict them to the range $1.5 < \alpha_i < 4.5$ and restrict the power merely to be positive $P_0 > 0$.

The log-barrier method adds a finite amount to the cost function as the parameters approach the barrier, which then increases quickly to infinity as the barrier is approached. Practically, this is implemented by appending terms to the cost function in (7.2) to form (7.4) as

$$\begin{aligned} &J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \\ &\sum_{\forall i} P_i^2 + P_0^2 d_i^{-j(i)} - 2P_0 P_i d_i^{-j(i)/2} + \frac{-1}{t}[\log(P_0) \\ &+ \log(-\alpha_1 + 4.5) + \log(\alpha_1 - 1.5) \\ &+ \log(-\alpha_2 + 4.5) + \log(\alpha_2 - 1.5) \\ &+ \log(-\alpha_3 + 4.5) + \log(\alpha_3 - 1.5)]\,, \end{aligned} \tag{7.4}$$

where t is a value that determines how steep the approach towards infinity is as the barrier is approached. In practice, it is common to start t at a low value to allow for easy avoidance of the barrier, and to increase it in successive iterations in order to make it more closely match the ideal barrier, a step function.

Since Newton's method can take large steps, it is imperative to prevent a step being taken outside of the log barrier due to the step size being too large. The preventative method employed here is the backtracking line search, which finds the largest step size that

can be used in an iterative method which still minimizes the given cost function.

Using both the log barrier and the backtracking line search, the cost function in (7.4) is minimized using Newton's method. The resulting behavior is an iterative method that is too "timid" to take any steps. The initial estimate is generally the final estimate, and for this reason this approach was abandoned.

7.6 Possible Improvements

None of the clustering methods presented in this chapter proved to work well empirically. The conclusion drawn is that the more complex models allowed too much freedom to the loss coefficients to be useful. Other combinations of clustering with different methods may prove to be beneficial, or even loss coefficient estimation based on satellite imaging data. Such avenues are not explored further in this paper.

CHAPTER 8

Comparison of Methods and Results

The table below summarizes the results from the four methods evaluated.

Table 8.1: Errors in meters for each method and dataset.

Dataset/Method	circles	BCP	simplified 3p	subset
sant1	15.76	2.50	5.09	13.08
sant2	7.92	5.46	2.19	20.45
quad3	108.94	41.46	19.47	1.65
upr3	116.26	181.36	128.74	30.26
aggr1	164.96	345.72	107.05	64.92
aggr4	1862.26	841.77	188.46	179.55

Overall, the subset method algorithm performs the best. In cases where observation location is known to be very close to the transmitter (within 25 meters) the simplified 3-parameter method outperforms the subset method. The direct comparison between methods can be found in figure 8.1 as a direct visualization of the data in table 8.1. The errors in the aggr4 dataset cannot be represented accurately on the chosen scale for the circles method and the BCP method, so those values have been excluded from figure 8.1.

It may also be useful to view the average error for each method. Taking into account only this average error it seems that the subset method is the best overall, as in figure 8.2. As before, the aggr4 dataset was excluded from this average since it is on a much larger scale than the other datasets.

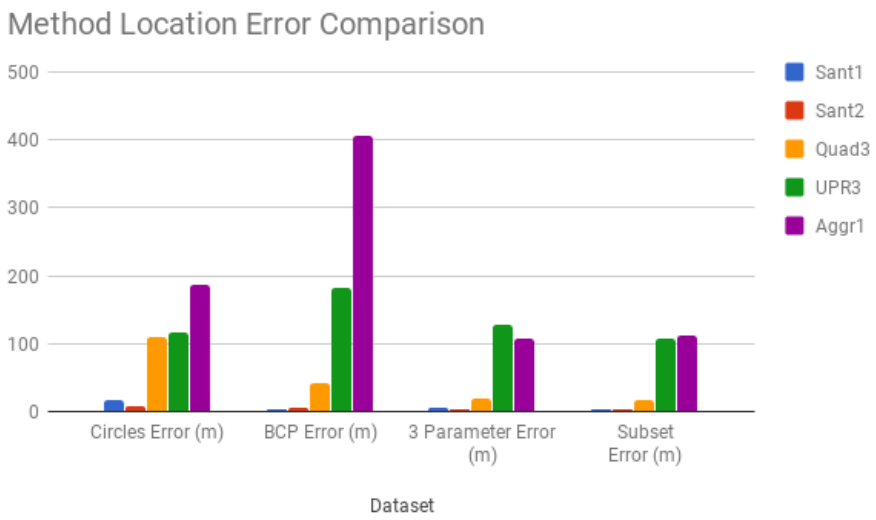


Fig. 8.1: Error comparison by method and dataset.

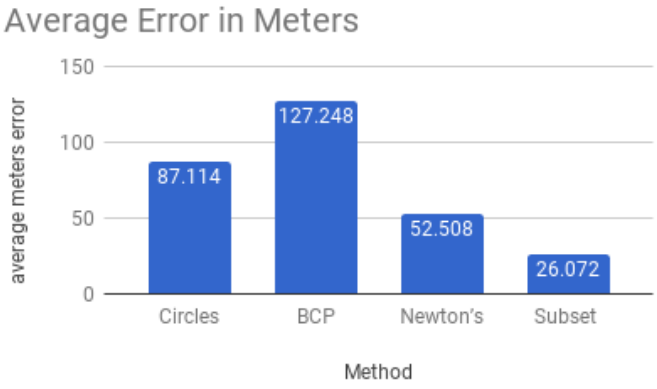


Fig. 8.2: Average error comparison by method and dataset in meters.

However, the approximate size of the observation grid should be accounted for. Measuring by the diagonal of the region with observations, the approximate dataset sizes are as listed in table 8.2.

Table 8.2: Dataset size by diagonals in meters.

Dataset	Diagonal length in meters
sant1	92
sant2	92
quad3	195
upr3	1040
aggr1	1562
aggr4	7071

Dividing each error by the diagonal dataset size, a feel for how significant each error is can be obtained. The average normalized errors for each method are shown in figure 8.3.

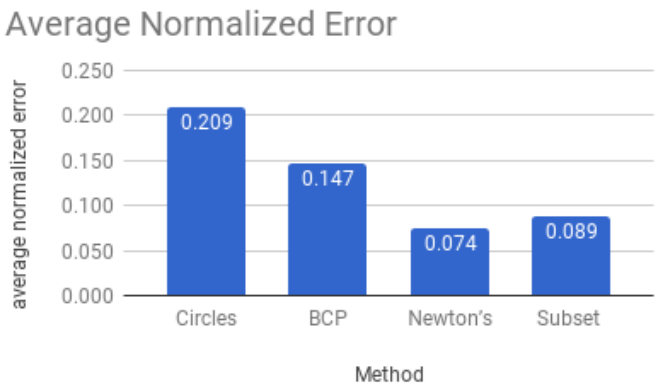


Fig. 8.3: Average error comparison by method and dataset, normalized.

The subset method now seems to perform slightly worse than the simplified 3-parameter method. However, both perform far better than the circles method or the BCP method in terms of both error in meters as well as normalized error.

Previously, a brief discussion of the problems with the subset method on the sant

datasets was presented. Removing these datasets from the average errors using the justification that they are too small, the normalized error is shown in figure 8.4.

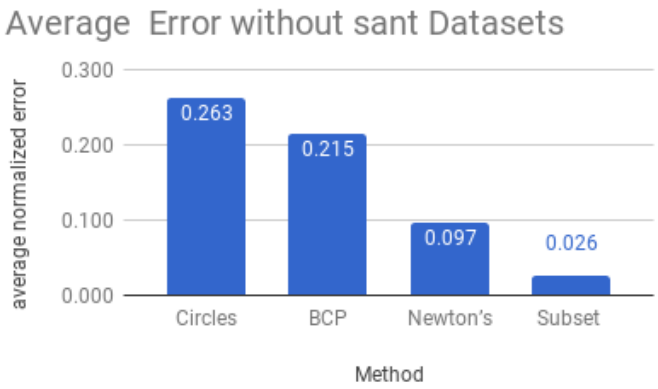


Fig. 8.4: Average error comparison by method and dataset, normalized, and excluding the sant datasets.

Using this interpretation of average errors, the subset method produced the best results. As another means of comparison, group the errors in meters by dataset. This representation can be seen in figure 8.5 and gives scale to the errors in each estimate. The subset method produces the best results for a majority of the datasets.

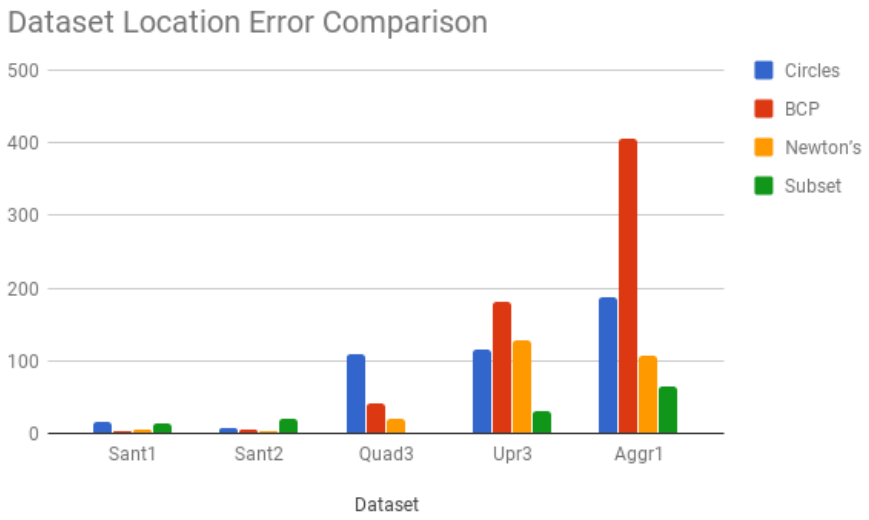


Fig. 8.5: Error comparison by dataset in meters.

8.1 Ending Location Estimate Representation

Another consideration in evaluating each method is the type of resulting final estimate for each method. If fusion of this data with other methods of geolocation is desired, a probabilistic representation of the location estimate would be necessary. Of the four methods compared above, the circles, BCP and subset methods all provide this. Since these three methods all produce heat maps, an easy visual comparison of the methods can be made as seen in figures 8.6 and 8.7.

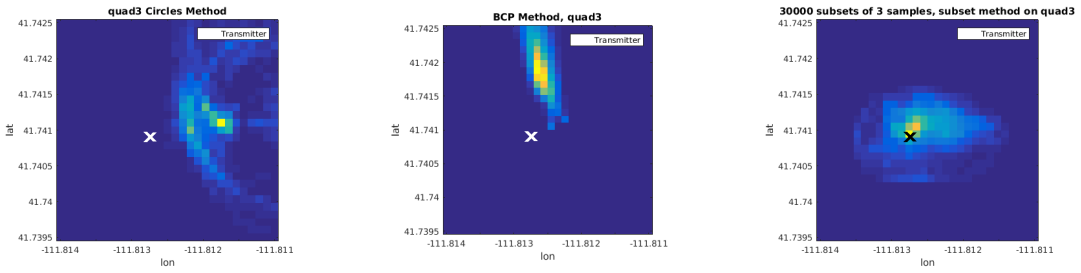


Fig. 8.6: The quad3 heat maps. From left to right: circles, BCP, subset.

The 3-parameter method estimate is, in this research, strictly a single best-fit estimate for all observations. Because of this, representing the estimate in a probabilistic manner is

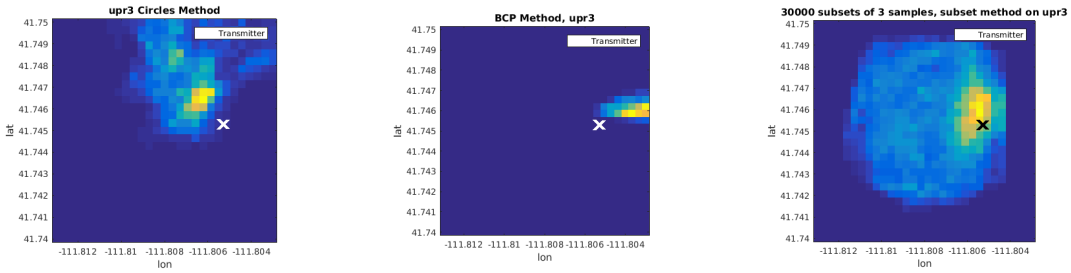


Fig. 8.7: The upr3 heat maps. From left to right: circles, BCP, subset.

difficult. Ellipses could be drawn using the final resulting Hessian matrix, but in practice the scale of these ellipses proved too inconsistent to be used in an algorithmic manner to represent the certainty of the estimates.

8.2 Error Analysis vs Number of Observations Used

A natural question to ask about the given analysis would be how many samples are needed in order to make an accurate location estimate. While many factors influence the answer to this question, insight can be gained by running these algorithms on subsets of the datasets and observing the quality of the estimates.

The simplified 3-parameter algorithm was applied to the datasets, taking different numbers of observations from the entire set each time. Using 25 trials of each observation amount, an average error term was made in order to analyze the effect that the number of observations has on the transmitter location estimate. The plots of these results is shown in figure 8.8. It is evident that the general behavior is as expected: the error of the estimate decreases as the number of observations used increases.

Similar analysis was done for the other algorithms and the results for the quad3 dataset are shown in figure 8.9. Trend lines have been fitted where appropriate. The circles algorithm does not necessarily produce better estimates with more observations available. The BCP method improves its estimate with more observations used, with diminishing returns around the 300 observation mark. Similarly, the 3-parameter method has diminishing returns, but it reaches this point far faster, around the 100 observation mark. The subset algorithm appears to plateau rapidly at the 100 observation mark but the error drops again

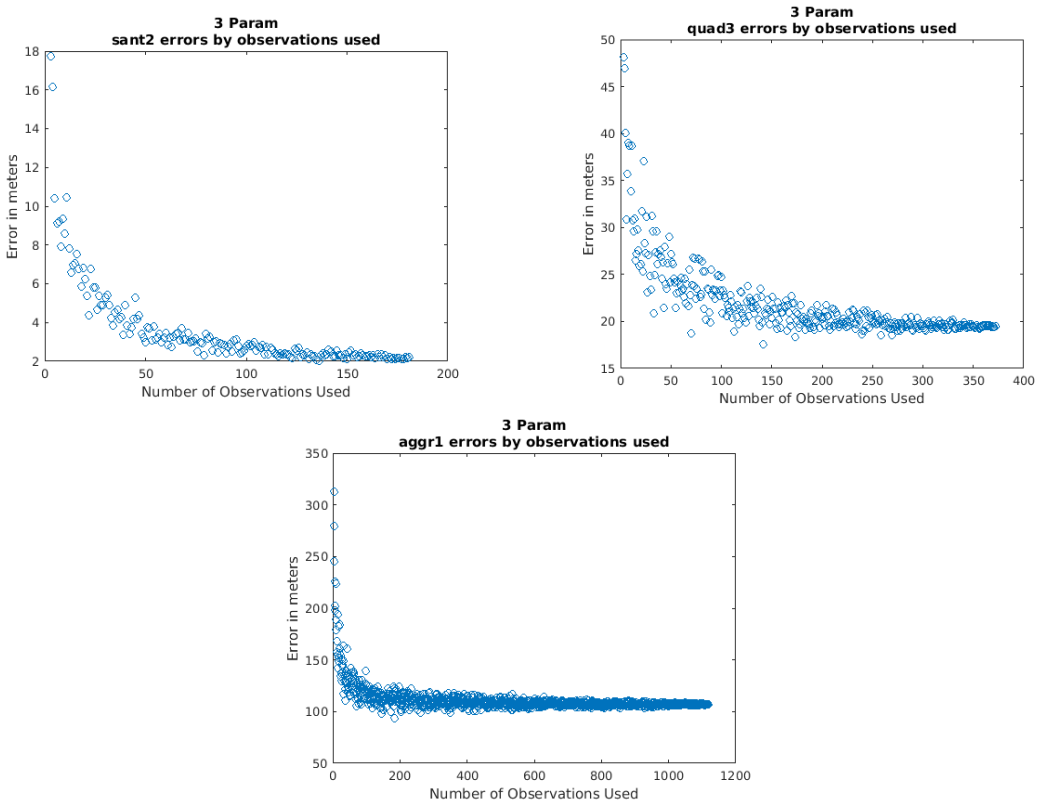


Fig. 8.8: Location estimate error as a function of number of observations used with the simplified 3-parameter method.

around the 300 sample mark until it reaches it's almost perfect location estimate.

Further analysis could be done on the number of observations needed in order to produce estimates of a sufficient quality.

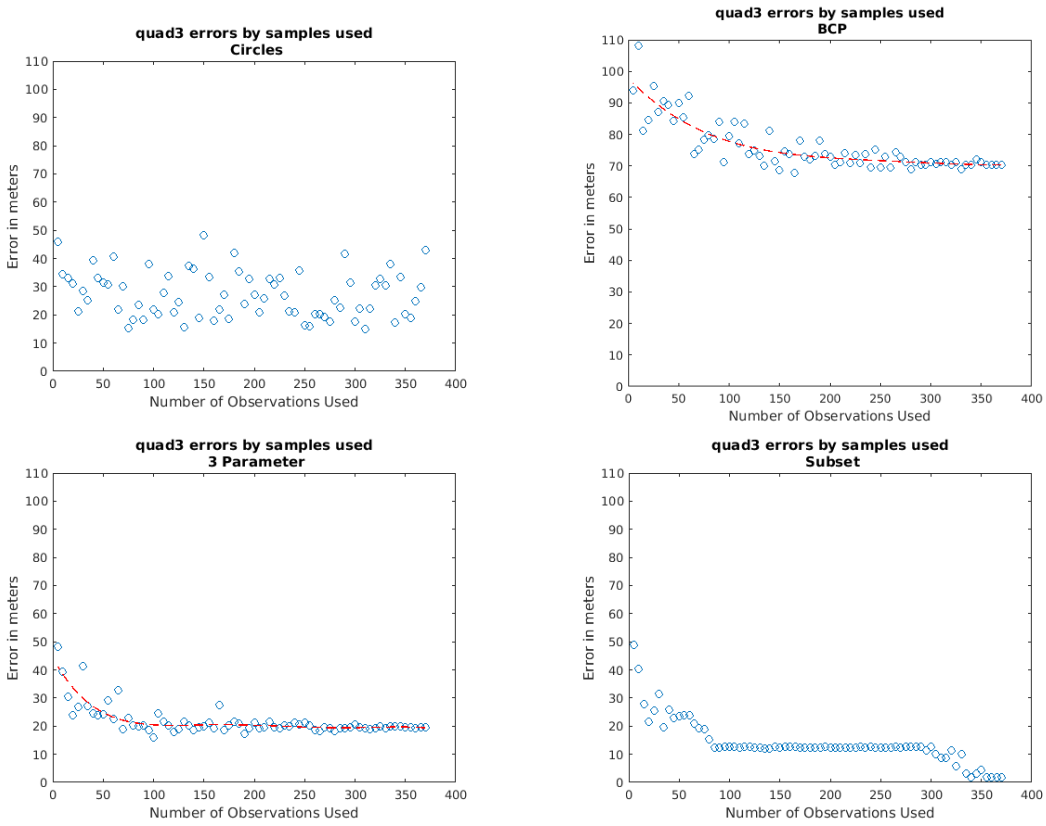


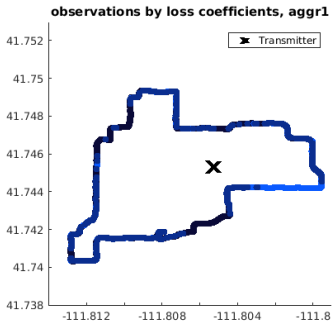
Fig. 8.9: Location estimate error as a function of number of number of observations used on the quad3 dataset. From left to right and top to bottom: circles, BCP, 3 Parameter, subset.

8.3 Estimating Environmental Features

Using the location estimates obtained by any of the methods above, it is possible to then estimate the loss coefficients for each observation, as described in section 5.3. The loss coefficient at a specific location may give insight into the geography or features in a region. For instance, a region with large loss coefficients may be shadowed by a building while a region with loss coefficients around 2 may have direct line of sight to the receiver.

Figure 8.10 depicts this type of analysis done for the aggr1 dataset. In this figure, the darker points mark areas where the loss coefficient is estimated to be very low. The brighter, more blue observations are locations with higher loss coefficients. For reference, figure 8.11 depicts the satellite image of the aggr1 dataset again. The brighter sections of observations suggest that some kind of shadowing or null in the antenna pattern might be

at that location.



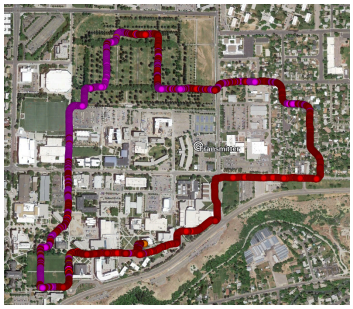


Fig. 8.10: The aggr1 loss coefficient groups. Fig. 8.11: The aggr1 dataset for comparison.

Further development of these types of estimation might prove useful for analyzing the radio propagation characteristics of regions, but is not discussed further here.

8.4 Alternative Power Measurements

In the processing chain shown in figure 2.1, a max operation is used over one second of data in order to output a final RSS measurement for the observation. An alternative approach would be to instead average the power over that second.

The GNURadio script responsible for power measurements was modified with a custom block that allowed for this averaging to be done in an efficient manner. The maximum powers were computed alongside the averages, resulting in a processing chain as seen in figure 8.12.

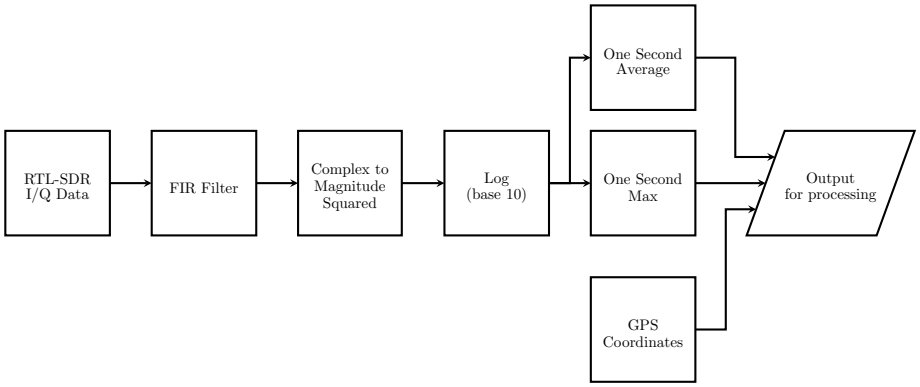


Fig. 8.12: Modified data collection scheme.

A new dataset was taken using this new collection scheme. The dataset is referred to as the aggr8 dataset and was taken around USU campus on a local FM radio channel. A satellite image with the observations marked can be seen in figure 8.13.

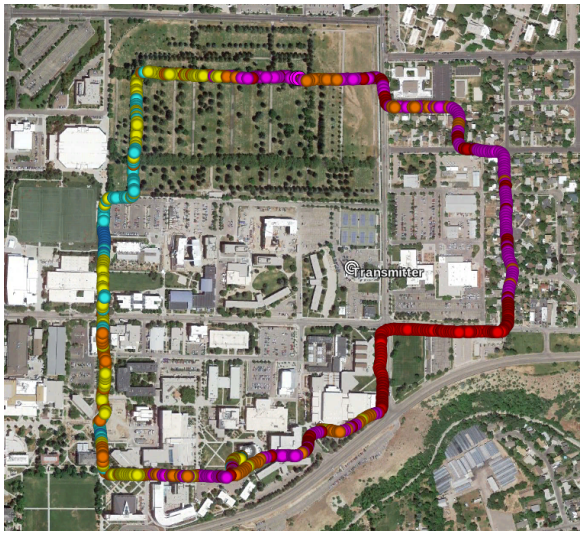


Fig. 8.13: The aggr8 dataset.

The algorithms were applied to the aggr8 dataset using both the average power and the maximum power data. The resulting errors in meters can be seen in table 8.3.

In all the cases except for using 3-parameter method, the ending location estimate ended up having the same error, and was also in the same location for the average power and maximum power measurements. The errors match in these cases exactly (to two decimal

Table 8.3: Errors in meters for each method on the aggr8 dataset using average power compared to maximum power.

Dataset/Method	circles	BCP	simplified 3p	subset
aggr8 power average	158.09	279.84	162.87	134.83
aggr8 power max	158.09	279.84	147.41	134.83

places) because the Circles, BCP, and Subset methods all quantize the estimates into a finite number of bins (for these results, a grid of 30×30 bins was used).

For the 3-parameter method, the average power measurements actually produced a worse location estimate than the maximum power measurements, but only by about 15 meters. The aggr8 dataset covers a 900×900 meter region, so normalizing this error difference by the diagonal length of 1273 meters means that the 15 meter difference is only about 1 percent of the span of the region. The two location estimates are effectively the same.

Taking the average power rather than the maximum power has no significant effect on the quality of the location estimate. Since the maximum operation is supported natively in GNURadio, the original data collection scheme is used as the default method in this research.

8.5 Notes on Non-Stationary Transmitters

The analysis presented assumes that the transmitter is stationary. If finding a moving transmitter is desired, a number of options are available for fitting these algorithms to this situation and are possible areas of further research.

Given a scenario where it is desired to find a unmanned aerial vehicle pilot, a number of searching drones could be deployed to take RSS measurements. These measurements would be reported into a central node to be processed using one of the discussed algorithms. The observations would be set to decay, or expire after a set amount of time, allowing for the possibility that the transmitter has moved. In this way, a constantly updating location estimate could be provided using multiple moving measurement nodes. A similar configuration, though not using the algorithms presented in this paper, is discussed in [9].

CHAPTER 9

Conclusion

9.1 Contributions

The research presented provides algorithms that locate a transmitter based on RSSI. Models for differing loss coefficients and noise were analyzed, and real world data was used to test the assumptions made. The work presented provides a stepping stone for future work in geolocation and modeling of RSS power loss as well as providing a viable method for performing geolocation.

9.2 Future Work

Areas of further research and improvement have been noted in sections above. In summary, each of the algorithms presented could be improved in a variety of ways. The BCP algorithm could be changed to deal with close comparisons more logically and the circles algorithm could extrude probabilities along the resulting loci in a way that represents the uncertainty of the measurements. The cost functions presented represent only a few of the possibilities that could be used in an optimization problem, and other models of the system could be developed to more closely model the complexities of the problem.

Further work could also be done to implement these algorithms in a way that could track a moving transmitter. Observations could be time-expiring, and a Kalman filter could be used to improve the estimations over time.

9.3 Conclusion

Geolocation using RSS measurements can help solve common geolocation problems when high degrees of synchronization are unavailable or impractical. In addition, geolocation based on RSS measurements can combine well with networks of distributed receivers

working together to locate a source since there is no time-dependent feature of the data.

Many different algorithms were proposed and presented in this text, four of which were presented in depth. The methods in section 7 were presented and developed in pursuit of a more complete model, but since none of those methods proved viable in the end, the complete results of their development and experiments were not included. It should suffice to say that the more complex models that allow more freedom in the estimation of loss coefficients demonstrated inconsistencies that made them impractical in regular use.

The best method for the datasets considered in this research was the subset method presented in section 6. It resulted in the smallest average error in meters and qualitatively produced the most useful and accurate histogram heat maps. As the purpose of this thesis was to develop and compare methods for geolocation based on RSS measurements, the objective was achieved.

REFERENCES

[1] G. A. Naik, M. P. Khedekar, M. Krishnamoorthy, S. D. Patil, and R. N. Deshmukh, "Comparison of RSSI techniques in wireless indoor geolocation," in *2012 National Conference On Computing And Communication Systems*, Nov 2012, pp. 1–5.

[2] S. O. D. S. Paul Scerri, Robin Glinton and K. Sycara, "Geolocation of RF emitters by many UAVs," school of Computer Science, Carnegie Mellon University. [Online]. Available: http://www.cs.cmu.edu/~pscerri/papers/infotech07.pdf

[3] D. J. Walter, K. Bryan, J. Stephens, C. Bullmaster, and V. Chakravarthy, "Localization of RF emitters using compressed sensing with multiple cooperative sensors," in *2012 IEEE National Aerospace and Electronics Conference (NAECON)*, July 2012, pp. 236–240.

[4] M. R. K. Aziz, S. N. Karimah, N. Yoshio, K. Anwar, and T. Matsumoto, "Achieving accurate geo-location detection using joint RSS-DOA factor graph technique," in *2016 10th International Conference on Telecommunication Systems Services and Applications (TSSA)*, Oct 2016, pp. 1–6.

[5] J. He, K. Pahlavan, S. Li, and Q. Wang, "A testbed for evaluation of the effects of multipath on performance of TOA-based indoor geolocation," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 8, pp. 2237–2247, Aug 2013.

[6] F. Brachmann, "About performance requirements set against consumer-grade geolocation technologies," in *2013 International Conference on System Science and Engineering (ICSSE)*, July 2013, pp. 309–312.

[7] M. Anisetti, C. A. Ardagna, V. Bellandi, E. Damiani, and S. Reale, "Map-based location and tracking in multipath outdoor mobile networks," *IEEE Transactions on Wireless Communications*, vol. 10, no. 3, pp. 814–824, March 2011.

[8] J. Adams, "Transmitter localization using autonomous robotic swarms," Master's thesis, Utah State University, Logan, UT, 2010.

[9] J. Liang and Q. Liang, "RF emitter location using a network of small unmanned aerial vehicles (SUAVs)," in *2011 IEEE International Conference on Communications (ICC)*, June 2011, pp. 1–6.

[10] L. Zhao, H. Wang, P. Li, and J. Liu, "An improved WiFi indoor localization method combining channel state information and received signal strength," in *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 8964–8969.

[11] F. Askarzadeh, K. Pahlavan, S. Makarov, Y. Ye, and U. Khan, "Analyzing the effect of human body and metallic objects for indoor geolocation," in *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*, March 2016, pp. 1–5.

[12] C. Medina, J. C. Segura, and S. Holm, "Feasibility of ultrasound positioning based on signal strength," in *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Nov 2012, pp. 1–9.

[13] A. Ren, D. Cao, F. Zhou, T. Zhu, F. Hu, Z. Zhang, X. Yang, M. U. Rehman, W. Zhao, and Q. H. Abbasi, "Characterization of the on-body received signal strength indication considering different propagation environment," in *2015 IEEE/CIC International Conference on Communications in China - Workshops (CIC/ICCC)*, Nov 2015, pp. 52–56.

[14] S. Bagirathi, S. Sankar, and Sandhya, "Tag detection in RFID system based on RSSI technique for LF and HF passive tags," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, April 2016, pp. 0452–0456.

[15] T. S. Rappaport, *Wireless Communications Principles and Practice, Second Edition.* Prentice Hall, 2001.

[16] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics.* Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992

APPENDICES

APPENDIX A

Gradients and Hessians

A.1 3-parameter Method

The 3-parameter Newton's iterations require a 3×1 gradient and a 3×3 Hessian, which are listed below as partial derivatives.

Cost Function

$$J(x_0, y_0, P_0) = \sum_i (P_i d_i^{\alpha_i} - P_0)^2$$

Gradient

$$\frac{\partial J}{\partial x_0} = 2 \sum_i P_i \alpha_i (x_0 - x_i) \left[P_i d_i^{\alpha_i - 1} - P_0 d_i^{\frac{\alpha_i}{2} - 1} \right]$$

$$\frac{\partial J}{\partial y_0} = 2 \sum_i P_i \alpha_i (y_0 - y_i) \left[P_i d_i^{\alpha_i - 1} - P_0 d_i^{\frac{\alpha_i}{2} - 1} \right]$$

$$\frac{\partial J}{\partial P_0} = 2 \sum_i P_0 - P_i d_i^{\frac{\alpha_i}{2}}$$

Hessian

$$\frac{\partial J^2}{\partial x_0 \partial x_0} = 4 \sum_i P_i \alpha_i \left[P_i d_i^{\alpha_i - 1} - P_0 d_i^{\frac{\alpha_i}{2} - 1} \right] +$$

$$2 P_i \alpha_i (x_0 - x_i)^2 \left[P_i (\alpha_i - 1) d_i^{\alpha_i - 2} - P_0 (\frac{\alpha_i}{2} - 1) d_i^{\frac{\alpha_i}{2} - 2} \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial y_0} = 4 \sum_i P_i \alpha_i \left[P_i d_i^{\alpha_i - 1} - P_0 d_i^{\frac{\alpha_i}{2} - 1} \right] +$$

$$2 P_i \alpha_i (y_0 - y_i)^2 \left[P_i (\alpha_i - 1) d_i^{\alpha_i - 2} - P_0 (\frac{\alpha_i}{2} - 1) d_i^{\frac{\alpha_i}{2} - 2} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = 4 \sum_i P_i \alpha_i (x_0 - x_i)(y_0 - y_i) \left[P_i (\alpha_i - 1) d_i^{\alpha_i - 2} - P_0 (\frac{\alpha_i}{2} - 1) d_i^{\frac{\alpha_i}{2} - 2} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = \frac{\partial J^2}{\partial y_0 \partial x_0}$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = -2 \sum_i P_i \alpha_i (x_0 - x_i) d_i^{\frac{\alpha_i}{2} - 1}$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = \frac{\partial J^2}{\partial P_0 \partial x_0}$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = -2 \sum_i P_i \alpha_i (y_0 - y_i) d_i^{\frac{\alpha_i}{2} - 1}$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = \frac{\partial J^2}{\partial P_0 \partial y_0}$$

$$\frac{\partial J^2}{\partial P_0 \partial P_0} = 2 \sum_i 1 = 2 * \text{number of observations}$$

A.2 Simplified 3-Parameter Method

Restricting all loss coefficients to be 2, the gradient and Hessian simplify as follows.

Simplified Gradient

$$\frac{\partial J}{\partial x_0} = 4\sum_i P_i(x_0 - x_i)(P_i d_i - P_0)$$

$$\frac{\partial J}{\partial y_0} = 4\sum_i P_i(y_0 - y_i)(P_i d_i - P_0)$$

$$\frac{\partial J}{\partial P_0} = 2\sum_i P_0 - P_i d_i$$

Simplified Hessian

$$\frac{\partial J^2}{\partial x_0 \partial x_0} = 4\sum_i P_i(P_i d_i - P_0) + 2P_i^2(x_0 - x_i)^2$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = 8\sum_i P_i^2(x_0 - x_i)(y_0 - y_i)$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = -4\sum_i P_i(x_0 - x_i)$$

$$\frac{\partial J^2}{\partial y_0 \partial y_0} = 4\sum_i P_i(P_i d_i - P_0) + 2P_i^2(y_0 - y_i)^2$$

$$\frac{\partial J^2}{\partial y_0 \partial x_0} = \frac{\partial J^2}{\partial x_0 \partial y_0}$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = -4\sum_i P_i^2(y_0 - y_i)$$

$$\frac{\partial J^2}{\partial P_0 \partial x_0} = \frac{\partial J^2}{\partial x_0 \partial P_0}$$

$$\frac{\partial J^2}{\partial P_0 \partial y_0} = \frac{\partial J^2}{\partial y_0 \partial P_0}$$

$$\frac{\partial J^2}{\partial P_0 \partial P_0} = 2 \sum_i 1 = 2 * \text{number of observations}$$

A.3 6-Parameter Method

The 6-parameter Newton's iterations require a 6×1 gradient and a 6×6 Hessian, which are listed below as partial derivatives. Note that the obviously symmetric partials have been assumed to be understood as their partial pair.

Cost Function

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_0 - P_i d_i^{j(i)/2})^2$$

$$j(i) = \begin{cases} \alpha_1 & i \in \text{Group 1} \\ \alpha_2 & i \in \text{Group 2} \\ \alpha_3 & i \in \text{Group 3} \end{cases}$$

Gradient

$$\frac{\partial J}{\partial x_0} = 2 \sum_i P_i j(i) (x_0 - x_i) \left[P_i d_i^{j(i)-1} - P_0 d_i^{\frac{j(i)}{2}-1} \right]$$

$$\frac{\partial J}{\partial y_0} = 2 \sum_i P_i j(i) (y_0 - y_i) \left[P_i d_i^{j(i)-1} - P_0 d_i^{\frac{j(i)}{2}-1} \right]$$

$$\frac{\partial J}{\partial P_0} = 2 \sum_i P_0 - P_i d_i^{\frac{j(i)}{2}}$$

$$\frac{\partial J}{\partial \alpha_k} = 2 \sum_{i,j(i)=\alpha_k} P_i^2 d_i^{j(i)} \ln(d_i) - 2P_0 P_i d_i^{\frac{j(i)}{2}} \ln(d_i^{\frac{1}{2}}), \text{ for } k = 1, 2, 3$$

Hessian

$$\frac{\partial J^2}{\partial x_0 \partial x_0} = 4 \sum_i P_i j(i) \left[P_i d_i^{j(i)-1} - P_0 d_i^{\frac{j(i)}{2}-1} \right] +$$

$$2P_i j(i)(x_0 - x_i)^2 \left[P_i (j(i)-1) d_i^{j(i)-2} - P_0 (\frac{j(i)}{2} - 1) d_i^{\frac{j(i)}{2}-2} \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial y_0} = 4 \sum_i P_i j(i) \left[P_i d_i^{j(i)-1} - P_0 d_i^{\frac{j(i)}{2}-1} \right] +$$

$$2P_i j(i)(y_0 - y_i)^2 \left[P_i (j(i)-1) d_i^{j(i)-2} - P_0 (\frac{j(i)}{2} - 1) d_i^{\frac{j(i)}{2}-2} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = 4 \sum_i P_i j(i)(x_0 - x_i)(y_0 - y_i) \left[P_i (j(i)-1) d_i^{j(i)-2} - P_0 (\frac{j(i)}{2} - 1) d_i^{\frac{j(i)}{2}-2} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = -2 \sum_i P_i j(i)(x_0 - x_i) d_i^{\frac{j(i)}{2}-1}$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = -2 \sum_i P_i j(i)(y_0 - y_i) d_i^{\frac{j(i)}{2}-1}$$

$$\frac{\partial J^2}{\partial P_0 \partial P_0} = 2 \sum_i 1 = 2 * \text{number of observations}$$

$$\frac{\partial J^2}{\partial x_0 \partial \alpha_k} = 2 \sum_{i,j(i)=\alpha_k} P_i (x_0 - x_i) d_i^{j(i)-1} \left[1 + j(i) \ln(d_i) \right] - P_0 P_i (x_0 - x_i) d_i^{\frac{j(i)}{2}-1} \left[1 + j(i) \ln(d_i^{\frac{1}{2}}) \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial \alpha_k} = 2 \sum_{i,j(i)=\alpha_k} P_i(y_0 - y_i) d_i^{j(i)-1} \left[1 + j(i)\ln(d_i)\right] - P_0 P_i (y_0 - y_i) d_i^{\frac{j(i)}{2}-1} \left[1 + j(i)\ln(d_i^{\frac{1}{2}})\right]$$

$$\frac{\partial J^2}{\partial P_0 \partial \alpha_k} = -2 \sum_{i,j(i)=\alpha_k} P_i d_i^{\frac{j(i)}{2}} \ln(d_i^{\frac{1}{2}})$$

$$\frac{\partial J^2}{\partial \alpha_k \partial \alpha_k} = \sum_{i,j(i)=\alpha_k} P_i^2 d_i^{j(i)} \ln(d_i)^2 - 2P_0 P_i d_i^{\frac{j(i)}{2}} \ln(d_i^{\frac{1}{2}})^2$$

$$\frac{\partial J^2}{\partial \alpha_l \partial \alpha_k} = 0, k \neq l$$

A.4 6-Parameter Method with Alternative Cost Function

The 6-parameter Newton's iterations with the alternative cost function also require a 6×1 gradient and a 6×6 Hessian, which are listed below as partial derivatives. Note that the obviously symmetric partials have been assumed to be understood as their partial pair.

Cost Function

$$J(x_0, y_0, P_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_i - P_0 d_i^{-j(i)/2})^2$$

Gradient

$$\frac{\partial J}{\partial x_0} = 2 \sum_i P_0 j(i)(x_0 - x_i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1}\right]$$

$$\frac{\partial J}{\partial y_0} = 2 \sum_i P_0 j(i)(y_0 - y_i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1}\right]$$

$$\frac{\partial J}{\partial P_0} = 2 \sum_i P_0 d_i^{-j(i)} - 2P_i d_i^{\frac{-j(i)}{2}}$$

$$\frac{\partial J}{\partial \alpha_k} = \sum_{i,j(i)=\alpha_k} P_0^2 d_i^{-j(i)} \ln(d_i^{-1}) - 2P_0 P_i d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})$$

Hessian

$$\frac{\partial J^2}{\partial x_0 \partial x_0} = 2\sum_i P_0 j(i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1} + 2(x_0 - x_i)^2 \right.$$
$$\left. \left[P_i (\frac{-j(i)}{2} - 1) d_i^{\frac{-j(i)}{2}-2} - P_0(-j(i)-1) d_i^{-j(i)-2} \right] \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial y_0} = 2\sum_i P_0 j(i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1} + 2(y_0 - y_i)^2 \right.$$
$$\left. \left[P_i (\frac{-j(i)}{2} - 1) d_i^{\frac{-j(i)}{2}-2} - P_0(-j(i)-1) d_i^{-j(i)-2} \right] \right]$$

$$\frac{\partial J^2}{\partial p_0 \partial p_0} = 2\sum_i d_i^{-j(i)}$$

$$\frac{\partial J^2}{\partial \alpha_k \partial \alpha_k} = P_0^2 d_i^{-j(i)} \ln(d_i^{-1})^2 - 2P_0 P_i d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})^2$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = 4\sum_i P_0 j(i)(x_i - x_0)(y_i - y_0) \left[P_i (\frac{-j(i)}{2} - 1) d_i^{\frac{-j(i)}{2}-2} - P_0(-j(i)-1) d_i^{-j(i)-2} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = 2\sum_i j(i)(x_0 - x_i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - 2P_0 d_i^{-j(i)-1} \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = 2\sum_i j(i)(y_0 - y_i) \left[P_i d_i^{\frac{-j(i)}{2}-1} - 2P_0 d_i^{-j(i)-1} \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial \alpha_k} = 2 \sum_{i,j(i)=\alpha_k} P_0(x_0 - x_i)$$

$$\left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1} + j(i) P_i d_i^{\frac{-j(i)}{2}-1} \ln(d_i^{\frac{-1}{2}}) - j(i) P_0 d_i^{-j(i)-1} \ln(d_i^{-1}) \right]$$

$$\frac{\partial J^2}{\partial y_0 \partial \alpha_k} = 2 \sum_{i,j(i)=\alpha_k} P_0(y_0 - y_i)$$

$$\left[P_i d_i^{\frac{-j(i)}{2}-1} - P_0 d_i^{-j(i)-1} + j(i) P_i d_i^{\frac{-j(i)}{2}-1} \ln(d_i^{\frac{-1}{2}}) - j(i) P_0 d_i^{-j(i)-1} \ln(d_i^{-1}) \right]$$

$$\frac{\partial J^2}{\partial P_0 \partial \alpha_k} = 2 \sum P_0 d_i^{-j(i)} \ln(d_i^{-1}) - P_i d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})$$

$$\frac{\partial J^2}{\partial \alpha_l \partial \alpha_k} = 0, k \neq l$$

A.5 7-Parameter Method with Alternative Cost Function

The 7-parameter Newton's iterations require a 7×1 gradient and a 7×7 Hessian, which are listed below as partial derivatives. Note that the obviously symmetric partials have been assumed to be understood as their partial pair.

Cost Function

$$J(x_0, y_0, P_0, N_0, \alpha_1, \alpha_2, \alpha_3) = \sum_{\forall i} (P_i - P_0 d_i^{-j(i)/2} - N_0)^2$$

Gradient

$$\frac{\partial J}{\partial x_0} = -2 \sum_i j(i)(x_0 - x_i) P_0 \left[P_0 d_i^{-j(i)-1} + (N_0 - P_i) d_i^{\frac{-j(i)}{2}-1} \right]$$

$$\frac{\partial J}{\partial y_0} = -2\sum_i j(i)(y_0 - y_i)P_0\left[P_0 d_i^{-j(i)-1} + (N_0 - P_i)d_i^{\frac{-j(i)}{2}-1}\right]$$

$$\frac{\partial J}{\partial P_0} = 2\sum_i P_0 d_i^{-2j(i)} + (N_0 - P_i)d_i^{-j(i)}$$

$$\frac{\partial J}{\partial N_0} = 2\sum_i N_0 - P_i + P_0 d_i^{\frac{-j(i)}{2}}$$

$$\frac{\partial J}{\partial \alpha_k} = \sum_{i,j(i)=\alpha_k} P_0^2 d_i^{-j(i)}\ln(d_i^{-1}) + 2P_0(N_0 - P_i)d_i^{\frac{-j(i)}{2}}\ln(d_i^{\frac{-1}{2}})$$

Hessian

$$\frac{\partial J^2}{\partial x_0 \partial x_0} = -2\sum_i j(i)P_0\left[P_0 d_i^{-j(i)-1} + (N_0 - P_i)d_i^{\frac{-j(i)}{2}-1}\right] +$$

$$2j(i)(x_0 - x_i)^2 P_0\left[P_0(-j(i)-1)d_i^{-j(i)-2} + (N_0 - P_i)(\frac{-j(i)}{2} - 1)d_i^{\frac{-j(i)}{2}-2}\right]$$

$$\frac{\partial J^2}{\partial y_0 \partial y_0} = -2\sum_i j(i)P_0\left[P_0 d_i^{-j(i)-1} + (N_0 - P_i)d_i^{\frac{-j(i)}{2}-1}\right] +$$

$$2j(i)(y_0 - y_i)^2 P_0\left[P_0(-j(i)-1)d_i^{-j(i)-2} + (N_0 - P_i)(\frac{-j(i)}{2} - 1)d_i^{\frac{-j(i)}{2}-2}\right]$$

$$\frac{\partial J^2}{\partial x_0 \partial y_0} = 4\sum_i j(i)(x_0 - x_i)(y_0 - y_i)P0\left[P_0(j(i)+1)d_i^{-j(i)-2} + (N_0 - P_i)(\frac{j(i)}{2} + 1)d_i^{\frac{-j(i)}{2}-2}\right]$$

$$\frac{\partial J^2}{\partial x_0 \partial P_0} = -2\sum_i j(i)(x_0 - x_i)\left[2P_0 d_i^{-j(i)-1} + (N_0 - P_i)d_i^{\frac{-j(i)}{2}-1}\right]$$

$$\frac{\partial J^2}{\partial y_0 \partial P_0} = -2\sum_i j(i)(y_0 - y_i)\left[2P_0 d_i^{-j(i)-1} + (N_0 - P_i)d_i^{\frac{-j(i)}{2}-1}\right]$$

$$\frac{\partial J^2}{\partial x_0 \partial N_0} = -2 \sum_i j(i)(x_0 - x_i) P_0 d_i^{\frac{-j(i)}{2}-1}$$

$$\frac{\partial J^2}{\partial y_0 \partial N_0} = -2 \sum_i j(i)(y_0 - y_i) P_0 d_i^{\frac{-j(i)}{2}-1}$$

$$\frac{\partial J^2}{\partial P_0 \partial P_0} = 2 \sum_i d_i^{-j(i)}$$

$$\frac{\partial J^2}{\partial N_0 \partial N_0} = 2 \sum_i 1 = 2 * \text{ number of observations}$$

$$\frac{\partial J^2}{\partial P_0 \partial N_0} = 2 \sum_i d_i^{\frac{-j(i)}{2}}$$

$$\frac{\partial J^2}{\partial \alpha_k \partial \alpha_k} = \sum_{i, j(i)=\alpha_k} P_0 d_i^{-j(i)} \ln(d_i^{-1})^2 + 2P_0(N_0 - P_i) d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})^2$$

$$\frac{\partial J^2}{\partial x_0 \partial \alpha_k} = -2 \sum_{i, j(i)=\alpha_k} (x_0 - x_i) P_0 \left[P_0 d_i^{-j(i)-1} + (N_0 - P_i) d_i^{\frac{-j(i)}{2}-1} \right]$$

$$+ j(i)(x_0 - x_i) P_0 \left[P_0 d_i^{-j(i)-1} \ln(d_i^{-1}]) + (N_0 - P_i) d_i^{\frac{-j(i)}{2}-1} \ln(d_i^{\frac{-1}{2}}) \right]$$

$$\frac{\partial J^2}{\partial x_0 \partial \alpha_k} = -2 \sum_{i, j(i)=\alpha_k} (y_0 - y_i) P_0 \left[P_0 d_i^{-j(i)-1} + (N_0 - P_i) d_i^{\frac{-j(i)}{2}-1} \right]$$

$$+ j(i)(y_0 - y_i) P_0 \left[P_0 d_i^{-j(i)-1} \ln(d_i^{-1}]) + (N_0 - P_i) d_i^{\frac{-j(i)}{2}-1} \ln(d_i^{\frac{-1}{2}}) \right]$$

$$\frac{\partial J^2}{\partial P_0 \partial \alpha_k} = 2 \sum_{i, j(i)=\alpha_k} P_0 d_i^{-j(i)} \ln(d_i^{-1}) + (N_0 - P_i) d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})$$

$$\frac{\partial J^2}{\partial N_0 \partial \alpha_k} = 2 \sum_{i, j(i)=\alpha_k} P_0 d_i^{\frac{-j(i)}{2}} \ln(d_i^{\frac{-1}{2}})$$

$$\frac{\partial J^2}{\partial \alpha_l \partial \alpha_k} = 0, k \neq l$$

APPENDIX B

Code

B.1 Circles

The circles method is included below as a Matlab script.

Listing B.1: The circles method.

```
% Sam Whiting Nov 2017
% Performs the circles algorithm on a dataset
clear;clc;close all;

%% pick which data set to run on
% original datasets
% dataset = 'sant1';
% dataset = 'sant2';
% dataset = 'quad3';
dataset = 'upr3';
% dataset = 'aggr1';

% new datasets
% dataset = 'upr4';
% dataset = 'aggr2';
% dataset = 'aggr2_trunc';
% dataset = 'aggr4'; % large dataset
% dataset = 'aggr6'; % average powers here
% dataset = 'aggr8'; % mixed power/averages (see read_rss_data.m)

% simulated dataset
dataset = 'sim';

%% some controls/parameters to change

% path loss (can be a range)
% n_range = 3.1 : .1 : 3.3;
n_range = 2;

% downsample amount
n_downsamp = 1;

% data truncation (what range of points to use)
truncate = 0; % flag to signal truncation or not
start = 1; % starting index
n_observations = 100; % how many points to use

% toggle plots
plot_heat_map = 1;
plot_heat_with_diagram = 0;
plot_diagram = 1;
diagram_draw_circles = 0;
plot_errors_vs_n = 0;

% bins in the heat map (along one axis)
nbins = 30;
```

```
47
48
49  %% open the file
50  [rx_location, rx_power, n_rx, tx_location] = ...
51      read_rss_data( dataset, n_downsamp, truncate, start, n_observations, 0,0);
52
53  %% determine dimensions/edges
54  [lon0, lon1, lat0, lat1] = get_dimensions(dataset);
55  x_edges = linspace(lon0,lon1,nbins);
56  y_edges = linspace(lat0,lat1,nbins);
57
58  %% run for each path loss coefficient guess
59  save_errors = []; save_n = [];
60  for z = 1:length(n_range)
61      n = n_range(z);
62      fprintf('Running with path loss n = %.2f...\n',n);
63
64      %% circles of constant radii ratio
65      combs = combnk(1:n_rx,2); % all the different pairs of receivers
66      n_combs = length(combs);
67      u = zeros(n_combs,1);v = zeros(n_combs,1);w = zeros(n_combs,1);
68      for q = 1:n_combs
69          k = combs(q,1);
70          m = combs(q,2);
71          [u(q),v(q),w2(q)] = power_ratio(rx_location(k,1), rx_location(k,2), ...
72                                          rx_location(m,1), rx_location(m,2), ...
73                                          nthroot( (rx_power(k)/rx_power(m)), n) );
74      end
75      if (w2 <= 0)
76          fprintf('ERROR: negative radius\n');
77          return;
78      end
79
80      %% find circle intersections
81      w = sqrt(w2);
82      ix = zeros(n_combs,2,2);
83      for q = 1:n_combs
84          k = combs(q,1);
85          m = combs(q,2);
86          [ix(q,1,:), ix(q,2,:)]  = circcirc(u(k),v(k),w(k), ...
87                                             u(m),v(m),w(m) );
88      end
89
90      %% histogram
91      [N,x_edges,y_edges] = histcounts2(ix(:,1,:),ix(:,2,:),x_edges,y_edges);
92
93      % find the middle of the max bin in the histogram
94      [~, ind1] = max(N(:)); % stack and find argmax
95      [x_guess_bin, y_guess_bin] = ind2sub(size(N),ind1); % turn a linear argmax into a
            2d one
96      bin_width = x_edges(2) - x_edges(1);
97      x_guess = x_edges(x_guess_bin) + .5*bin_width; % longitude guess (middle of max
            bin)
98      y_guess = y_edges(y_guess_bin) + .5*bin_width; % latitude guess (middle of max
            bin)
99
100     %% error term
101     error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
102     fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
103     fprintf('             Lon %.8f\n',tx_location(1));
104     fprintf('TX Estimate: Lat %.8f\n',y_guess);
105     fprintf('             Lon %.8f\n',x_guess);
106     fprintf('Error: %.2f meters\n',error_m);
107
108     %% heatmaps
```

```
    if plot_heat_map == 1
        heat_data = rot90(N);
        figure;
        imagesc([lon0,lon1],[lat0,lat1],flip(heat_data, 1));
%         title([dataset, ' with loss coeff n = ',num2str(n)]);
        title([dataset, ' Circles Method']);
        hold on;
        tx_plot = scatter(tx_location(1),tx_location(2),200,'w','LineWidth',5,'Marker','x');
        lgnd = legend(tx_plot,'Transmitter');
        xlabel('lon');ylabel('lat');
        axis('xy');pbaspect([1,1,1]);
%         set(lgnd,'color',[157,163,173]/255);
    end

    if plot_heat_with_diagram == 1
        heat_data = rot90(N);
        figure;
        imagesc([lon0,lon1],[lat0,lat1],flip(heat_data, 1));
%         title([dataset, ' with loss coeff n = ',num2str(n)]);
        title([dataset, ' Circles Method']);
        hold on;
        tx_plot = scatter(tx_location(1),tx_location(2),200,'w','LineWidth',5,'Marker','x');
        rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
        guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
        legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated Location');
        xlabel('lon');ylabel('lat');
        axis('xy');pbaspect([1,1,1]);
    end

    %% diagram
    if plot_diagram == 1
        figure;
        rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
        hold on; set(gca,'ydir','normal');

        % adding circles to the plot can slow things down a lot...
        if diagram_draw_circles == 1
            for k = 1:n_combs
                draw_circle(u(k),v(k),w(k),'r--',2);
            end
        end

        tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x');
        guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
%         title([dataset, ' with loss coeff n = ',num2str(n)]);
        title([dataset, ' Circles Method']);
        legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated Location');
        axis([lon0,lon1,lat0,lat1]);pbaspect([1,1,1]);
    end

    save_errors = [save_errors,error_m];
    fprintf('\n');
end

%% error vs loss coefficient plot
if plot_errors_vs_n == 1
    figure;
    plot(n_range,save_errors);
    title(['Error vs Loss Coefficient, ',dataset]);
    xlabel('loss coefficient n');
```

```
169     ylabel('error in meters');
170 end
```

B.2 BCP

The BCP method is included below as a Matlab script.

Listing B.2: The BCP method.

```
1  % Sam Whiting Nov 2017
2  % Performs the BCP algorithm on a dataset
3  clear;clc;close all;
4
5  %% pick which data set to run on
6  % original datasets
7  % dataset = 'sant1';
8  % dataset = 'sant2';
9  % dataset = 'quad3';
10 % dataset = 'upr3';
11 % dataset = 'aggr1';
12
13 % new datasets
14 % dataset = 'upr4';
15 % dataset = 'aggr2';
16 % dataset = 'aggr2_trunc';
17 % dataset = 'aggr4'; % large dataset
18 % dataset = 'aggr6'; % average powers here
19 % dataset = 'aggr8'; % mixed power/averages (see read_rss_data.m)
20
21 % simulated dataset
22 dataset = 'sim';
23
24 %% some controls/parameters to change
25
26 % downsample amount
27 n_downsamp = 1;
28
29 % data truncation (what range of points to use)
30 truncate = 0; % flag to signal truncation or not
31 start = 1; % starting index
32 n_observations = 100; % how many points to use
33
34 % toggle plots
35 plot_heat_map = 1;
36 plot_heat_with_diagram = 0;
37 plot_diagram = 1;
38
39 % toggle debugging steps (very verbose)
40 debug_steps = 0;
41 auto_step = 0; % how many seconds to pause, or 0 for key prompt
42
43 % seed random number generator if desired
44 rng(1234);
45
46 % how close is too close for observations? Will skip these pairs
47 too_close_distance = 0;
48
49 % bins in the heat map (along one axis)
50 n_bins = 30;
51
52 % bayesian step (multiplicative factor)
```

```
step_size = 1.001;
% step_size = 1.1;

% use the midpoint or the k-weighted midpoint
use_midpoint = 1; % 1 is true (use real midpoint)
n_estimate = 4; %doesn't matter if we're just using the midpoint...


%% open the file
[rx_location, rx_power, n_rx, tx_location] = ...
    read_rss_data( dataset, n_downsamp, truncate, start, n_observations, 0,0);

%% determine dimensions/edges
[lon0, lon1, lat0, lat1] = get_dimensions(dataset);
x_edges = linspace(lon0,lon1,n_bins);
y_edges = linspace(lat0,lat1,n_bins);

%% the BCP algorithm
prior = ones(n_bins,n_bins)/(n_bins^2); % grid of probabilities

combs = combnk(1:n_rx,2);
n_combs = length(combs);

% randomly mix up the combinations of observations order
combs = combs(randperm(length(combs)),:);

skip_total = 0;

for z = 1:n_combs
    % which combination of points do we use?
    index1 = combs(z,1);
    index2 = combs(z,2);

    point1 = rx_location(index1,:);
    power1 = rx_power(index1);
    point2 = rx_location(index2,:);
    power2 = rx_power(index2);

    % ignore the comparison if the observations are too close (testing)
    if lldistance(point1(1),point1(2),point2(1),point2(2)) < too_close_distance
        skip_total = skip_total +1;
        continue;
    end

    % k is the power ratio
    k = nthroot(power1/power2,n_estimate);

    % k_middle is the weighted midpoint
    k_middle = [(k*point1(1) + point2(1))/(k+1), (k*point1(2) + point2(2))/(k+1)];

    % true midpoint
    real_middle = [(point1(1) + point2(1))/2, (point1(2) + point2(2))/2];

    % k_slope is the orthogonal slope of the line between the two points
    k_slope = -( point1(1) - point2(1) ) / ( point1(2) - point2(2) );

    % k_intercept is the y intercept of the orthogonal line through the
    % k_middle point
    if use_midpoint == 1
        k_intercept = real_middle(2) - (real_middle(1)*k_slope); % try using actual
            midpoint instead
    else
        k_intercept = k_middle(2) - (k_middle(1)*k_slope);
    end

```

```
    % Which point is it closest to?
    if (k > 1) % k>1 means it was closer to point 1
        above = (point1(2) > point1(1)*k_slope + k_intercept);
    else % else it was closer to point 2
        above = (point2(2) > point2(1)*k_slope + k_intercept);
    end

%%%     % k_x axis
%%%     k_x = 0:.1:20;
%%%     % points on the k-line
%%%     k_locus = (k_x .* k_slope) + k_intercept;

    % update every grid point
    for a = 1:n_bins
       for b = 1:n_bins

           x0 = x_edges(b); % the x coordinate in degrees
           y0 = y_edges(a); % y coordinate

           if (above == 1) % use the space above the line
                if (y0 > (x0*k_slope + k_intercept) )
                    prior(a,b) = prior(a,b)*step_size;

                else
                    prior(a,b) = prior(a,b)/step_size;
                end
           else % else use below the line
                if (y0 < (x0*k_slope + k_intercept) )
                    prior(a,b) = prior(a,b)*step_size;
                else
                    prior(a,b) = prior(a,b)/step_size;
                end
           end

       end
    end

    % normalize the prior
    prior = prior/(sum(sum(prior)));

    % a debugging step (very verbose)
    if debug_steps == 1
        figure;
        imagesc([lon0,lon1],[lat0,lat1],prior);
        set(gca,'xtick',[])
        set(gca,'xticklabel',[])
        set(gca,'ytick',[])
        set(gca,'yticklabel',[])
        axis('xy'); hold on;
        pbaspect([1,1,1]);
        title(['Step ', num2str(z)]);
        tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker
            ','x');

        sz1 = 50; sz2 = 50;
        if (k > 1) sz1 = 250; % closer to p1
        else sz2 = 250; % closer to p2
        end
        scatter(point1(1), point1(2),sz1,'r','filled');
        scatter(point2(1), point2(2),sz2,'r','filled');

        % k_x axis
        k_x = linspace(lon0,lon1);
        % points on the k-line
        k_locus = (k_x .* k_slope) + k_intercept;
```

```
        % plot k line
        plot(k_x,k_locus,'k--','LineWidth',2);

        if auto_step == 0
            pause;
        else
            pause(auto_step);
        end
    end
end

%% find the middle of the max bin in the histogram
prior_flip = prior';
[~, ind1] = max(prior_flip(:)); % stack and find argmax
[x_guess_bin, y_guess_bin] = ind2sub(size(prior),ind1); % turn a linear argmax into a
    2d one
bin_width = x_edges(2) - x_edges(1);
% x_guess = x_edges(x_guess_bin) + .5*bin_width; % longitude guess (middle of max bin
    )
% y_guess = y_edges(y_guess_bin) + .5*bin_width; % latitude guess (middle of max bin)
x_guess = x_edges(x_guess_bin); % longitude guess
y_guess = y_edges(y_guess_bin); % latitude guess

%% generate an error term
error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
fprintf('             Lon %.8f\n',tx_location(1));
fprintf('TX Estimate: Lat %.8f\n',y_guess);
fprintf('             Lon %.8f\n',x_guess);
fprintf('Error: %.2f meters\n',error_m);

fprintf('\n');
fprintf('Number of pairs skipped:     %d\n',skip_total);
fprintf('Percentage of pairs skipped: %.2f percent\n',100*skip_total/nchoosek(n_rx,2)
    );

%% plot the heat map
if plot_heat_map == 1
    figure;
    imagesc([lon0,lon1],[lat0,lat1],prior);
    hold on;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'w','LineWidth',5,'Marker','x
        ');
%     guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
%     legend([tx_plot, guess_plot],'Transmitter', 'Estimated Location');
    legend([tx_plot],'Transmitter');
    xlabel('lon');ylabel('lat');
%     colorbar;
    axis('xy');
    title(['BCP Method, ', dataset ]);
    axis('xy');pbaspect([1,1,1]);

end

%% plot the heat map with diagram
if plot_heat_with_diagram == 1
    figure;
    imagesc([lon0,lon1],[lat0,lat1],prior);
    hold on;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r');
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
```

```
    xlabel('lon');ylabel('lat');
%     colorbar;
    axis('xy');
    title(['BCP, ',dataset]);
    xlabel('lon');ylabel('lat');
    axis('xy');pbaspect([1,1,1]);
end

%% plot the diagram
if plot_diagram == 1
    figure;
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    hold on;
    set(gca,'ydir','normal');

    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
    title(['BCP Method, ', dataset ]);
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end
```

B.3 3-parameter

The 3-parameter method is included below as a Matlab script.

Listing B.3: The 3-parameter method

```
% Sam Whiting Nov 2017
% Performs the 3-parameter method on a dataset
clear;clc;close all;

%% pick which data set to run on
% original datasets
% dataset = 'sant1';
% dataset = 'sant2';
% dataset = 'quad3';
% dataset = 'upr3';
% dataset = 'aggr1';

% new datasets
% dataset = 'upr4';
% dataset = 'aggr2';
% dataset = 'aggr2_trunc';
% dataset = 'aggr4'; % large dataset
% dataset = 'aggr6'; % average powers here
% dataset = 'aggr8'; % mixed power/averages (see read_rss_data.m)

% simulated dataset
dataset = 'sim';

%% some controls/parameters to change

% downsample amount
n_downsamp = 1;

% data truncation (what range of points to use)
```

```
truncate = 0; % flag to signal truncation or not
start = 1; % starting index
n_observations = 100; % how many points to use

% toggle plots
plot_diagram = 1;
plot_newton = 0;
plot_ellipses = 0;

% pausing
pause_each_step = 0;
auto_step = .5; % how many seconds to pause, or 0 for key prompt

% newtons method iterations
n_iter = 10;

%% open the file
[rx_location, rx_power, n_rx, tx_location] = ...
    read_rss_data( dataset, n_downsamp, truncate, start, n_observations, 0,0);

%% determine dimensions/edges
[lon0, lon1, lat0, lat1] = get_dimensions(dataset);

%% newton's method
x_data = rx_location(:,1);  y_data = rx_location(:,2);
newton_vector = [lon1; lat0; 0]; % step values. initial conditions go here

grad = zeros(3,1);
hess = zeros(3,3);

if plot_newton == 1
    figure;
    plot(newton_vector(1),newton_vector(2),'bd','LineWidth',2);
    hold on;
    title(['Newtons Method, ', dataset ]);
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
    ylabel('Lat');
    xlabel('Lon');
end

fprintf('init:   %.4f %.4f %.4f\n', newton_vector(1), newton_vector(2), newton_vector
    (3));
for z = 1:n_iter

    % squared distance metric to be used for grad/hess
    d2 = (newton_vector(1)-x_data(:)).^2 + (newton_vector(2) - y_data(:)).^2;

    % compute the gradient
    grad(1) = 4*sum( rx_power .* (newton_vector(1)-x_data(:)) .* (rx_power.*d2 -
        newton_vector(3)) );
    grad(2) = 4*sum( rx_power .* (newton_vector(2)-y_data(:)) .* (rx_power.*d2 -
        newton_vector(3)) );
    grad(3) = 2*sum( newton_vector(3) - rx_power.*d2 );

    % compute the Hessian
    hess(1,1) = 4*sum( rx_power .* (rx_power.*d2 - newton_vector(3)) + 2*rx_power.^2
        .*(newton_vector(1)-x_data(:)).^2 );
    hess(1,2) = 8*sum( rx_power.^2 .* (newton_vector(1)-x_data(:)) .* (newton_vector
        (2) - y_data(:)));
    hess(2,1) = hess(1,2);
    hess(1,3) = -4*sum(rx_power .* (newton_vector(1)-x_data(:)) );
```

```
    hess(3,1) = hess(1,3);

    hess(2,2) = 4*sum( rx_power .* (rx_power.*d2-newton_vector(3)) + 2*rx_power.^2
        .*(newton_vector(2)-y_data(:)).^2 );
    hess(2,3) = -4*sum(rx_power .* (newton_vector(2)-y_data(:)) );
    hess(3,2) = hess(2,3);

    hess(3,3) = 2*n_rx;

    % do one newton step
    newton_vector = newton_vector - hess\grad; % xstep - inv(hess) * grad
    fprintf('step %d: %.4f %.4f %.4f\n', z, newton_vector(1), newton_vector(2),
        newton_vector(3));

    if plot_newton == 1
        if pause_each_step == 1
            pause(auto_step);
        end
        plot(newton_vector(1),newton_vector(2),'bd','LineWidth',2);
    end

end
x_guess = newton_vector(1);
y_guess = newton_vector(2);
fprintf('\n');

if plot_newton == 1
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
end

%% Generate an error term
error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
fprintf('             Lon %.8f\n',tx_location(1));
fprintf('TX Estimate: Lat %.12f\n',y_guess);
fprintf('             Lon %.12f\n',x_guess);
fprintf('Error: %.2f meters\n',error_m);

%% diagram plot
if plot_diagram == 1
    figure;
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    hold on;
    set(gca,'ydir','normal');

    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
    title(['Simplified 3 Parameter Method, ', dataset ]);
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end

%% ellipses plot
if plot_ellipses == 1

    figure;
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    hold on;
    set(gca,'ydir','normal');
```

```
149
150     % plot the major and minor axis
151     mu = [x_guess;y_guess];
152     partials = hess(1:2,1:2);
153     [u,v] = eig(inv(partials));
154     v1 = u(:,1)/sqrt(v(1,1));
155     v2 = u(:,2)/sqrt(v(2,2));
156     plot([mu(1),mu(1)+v1(1)],[mu(2),mu(2)+v1(2)]);
157     plot([mu(1),mu(1)+v2(1)],[mu(2),mu(2)+v2(2)]);
158
159     % plot contours
160     for k = [.01,.1,1,10,100]
161         [y] = plotellipse(inv(partials),mu,k);
162         plot(y(1,:),y(2,:));
163     end
164
165     tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
166     rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
167     guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
168     legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
169     title(['Level Curves, ', dataset ]);
170     axis([lon0, lon1, lat0, lat1]);
171     pbaspect([1,1,1]);
172 end
173
174 %% functions
175 % plot an ellipse
176 function [x] = plotellipse(A,x0,c)
177 % determine the points to plot an ellispe in two dimensions,
178 % described by (x-x0)'*A*(x-x0) = c, where A is symmetric
179
180 dtheta = 0.1;
181 [u,d] = eig(A);
182 x = [];
183 d = inv(sqrt(d));
184 for theta = 0:dtheta:2*pi
185   w = sqrt(c)*[cos(theta); sin(theta)];
186   z = d*w;
187   x = [x u*z + x0];
188 end
189 x = [x x(:,1)];
190 end
```

B.4 Subset

The subset method is included below as a Matlab script.

Listing B.4: The subset method.

```
1  % Sam Whiting Nov 2017
2  % Performs the subset method on a dataset
3  clear;clc;close all;
4
5  %% pick which data set to run on
6  % original datasets
7  % dataset = 'sant1';
8  % dataset = 'sant2';
9  % dataset = 'quad3';
10 % dataset = 'upr3';
```

```
% dataset = 'aggr1';

% new datasets
% dataset = 'upr4';
% dataset = 'aggr2';
% dataset = 'aggr2_trunc';
% dataset = 'aggr4'; % large dataset
% dataset = 'aggr6'; % average powers here
% dataset = 'aggr8'; % mixed power/averages (see read_rss_data.m)

% simulated dataset
dataset = 'sim';

%% some controls/parameters to change

% downsample amount
n_downsamp = 1;

% data truncation (what range of points to use)
truncate = 0; % flag to signal truncation or not
start = 10; % starting index
n_observations = 250; % how many points to use

% how many times to do newton's method on a different subset
n_subsets = 3 * 10000;
% size of each subset can be set as a percentage or in samples
subset_size = .2; % set as a percentage
subset_samples = 3; % if this is not 0, this will override the percentage

different_size_subsets = 0; % do we want to go through a list of subset sizes?
subset_sizes_list = [3,6]; % the list of subset sizes to use

% toggle plots
plot_diagram = 1;
plot_newton = 0;
plot_heat_map = 1;
plot_heat_with_diagram = 0;

% estimate loss coefficients?
estimate_loss_cfs = 0;
n_classes = 3; % only 5 colors so far
plot_classes = 0; % plot the diagram?
plot_loss_cfs = 0; % observations vs loss cfs

%heatmap bins
n_bins = 30;

% psuedo inverse or normal
use_pinv = 0;

% verbose-ness
print_each_error = 0;

% newtons method iterations
n_iter = 10;

%% open the file
[rx_location, rx_power, n_rx, tx_location] = ...
    read_rss_data( dataset, n_downsamp, truncate, start, n_observations, 0,0);

%% determine dimensions/edges
[lon0, lon1, lat0, lat1] = get_dimensions(dataset);
x_edges = linspace(lon0,lon1,n_bins+1);
y_edges = linspace(lat0,lat1,n_bins+1);

```

```
%% newton's method loop

if plot_newton == 1
    figure;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    hold on;
    title([num2str(n_subsets),' subsets of ',num2str(subset_samples),' samples,
        Newtons Method on ',dataset]);
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
%     legend([rx_plot,tx_plot],'Observations','Transmitter');
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end

% arrays for holding results
X_GUESS = zeros(n_subsets,1);
Y_GUESS = zeros(n_subsets,1);

% find out how many samples are in each subset
if subset_samples == 0
    subset_samples = floor(subset_size*n_rx);
end

% if there is a list, find it's length
if different_size_subsets == 1
    divisor = n_subsets/length(subset_sizes_list) +1; % how many subsets for each
        size in the list
    idx = floor((1:n_subsets)/divisor) + 1; % [11111 2222222 33333] % indexes for
        size to use
end

for q = 1:n_subsets

    % combinations of different size subsets
    if different_size_subsets == 1
        subset_samples = subset_sizes_list(idx(q));
%         fprintf('%d ',subset_sizes_list(idx(q)));
    end

    % pick out our random subset of data
    subset_indexes = randperm( n_rx, subset_samples );
    x_data      = rx_location(subset_indexes,1);
    y_data      = rx_location(subset_indexes,2);
    power_data = rx_power(subset_indexes);

    x_step = [lon1; lat0; 0]; % step values. initial conditions go here

    grad = zeros(3,1);
    hess = zeros(3,3);

%     fprintf('init:   %.4f %.4f %.4f\n', x_step(1), x_step(2), x_step(3));
    for z = 1:n_iter

        % squared distance metric to be used for grad/hess
        d2 = (x_step(1)-x_data(:)).^2 + (x_step(2) - y_data(:)).^2;

        % compute the gradient
        grad(1) = 4*sum( power_data .* (x_step(1)-x_data(:)) .* (power_data.*d2 -
            x_step(3)) );
        grad(2) = 4*sum( power_data .* (x_step(2)-y_data(:)) .* (power_data.*d2 -
            x_step(3)) );
        grad(3) = 2*sum( x_step(3) - power_data.*d2 );

        % compute the Hessian
```

```
135         hess(1,1) = 4*sum( power_data .* (power_data.*d2 - x_step(3)) + 2*power_data
                .^2 .*(x_step(1)-x_data(:)).^2 );
136         hess(1,2) = 8*sum( power_data.^2 .* (x_step(1)-x_data(:)) .* (x_step(2) -
                y_data(:)));
137         hess(2,1) = hess(1,2);
138         hess(1,3) = -4*sum(power_data .* (x_step(1)-x_data(:)) );
139         hess(3,1) = hess(1,3);
140
141         hess(2,2) = 4*sum( power_data .* (power_data.*d2-x_step(3)) + 2*power_data.^2
                .*(x_step(2)-y_data(:)).^2 );
142         hess(2,3) = -4*sum(power_data .* (x_step(2)-y_data(:)) );
143         hess(3,2) = hess(2,3);
144
145         hess(3,3) = 2*n_rx;
146
147         % do one newton step
148         if use_pinv == 0
149             x_step = x_step - hess\grad; % xstep - inv(hess) * grad
150         else
151             x_step = x_step - pinv(hess)*grad; % xstep - inv(hess) * grad
152         end
153 %         fprintf('step %d: %.4f %.4f %.4f\n', z, x_step(1), x_step(2), x_step(3));
154
155     end
156 %     fprintf('\n');
157     x_guess = x_step(1);
158     y_guess = x_step(2);
159
160     if plot_newton == 1
161 %         guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
162 %         legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', '
    Estimated Locations');
163         scatter(x_guess,y_guess,50,'b','Marker','.');
164     end
165
166     % Generate an error term
167     if print_each_error == 1
168         error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
169         fprintf('For subset number %d, the error is %.2f meters\n',q,error_m);
170     end
171
172     % SAVE information for plotting later
173     X_GUESS(q) = x_guess;
174     Y_GUESS(q) = y_guess;
175 end
176
177 if plot_newton == 1
178     legend('Transmitter','Observations','Newtons Method Estimates');
179 end
180
181 %% histogram
182 % make the 2d histogram
183 [hist_data,x_edges,y_edges] = histcounts2(X_GUESS,Y_GUESS,x_edges,y_edges);
184
185 % find the middle of the max bin in the histogram
186 [~, ind1] = max(hist_data(:)); % stack and find argmax
187 [x_guess_bin, y_guess_bin] = ind2sub(size(hist_data),ind1); % turn a linear argmax
    into a 2d one
188 bin_width = x_edges(2) - x_edges(1);
189 x_guess = x_edges(x_guess_bin) + .5*bin_width; % longitude guess (middle of max bin)
190 y_guess = y_edges(y_guess_bin) + .5*bin_width; % latitude guess (middle of max bin)
191
192 %% heatmaps
193 if plot_heat_map == 1
194     heat_data = rot90(hist_data);
```

```
    figure;
    imagesc([lon0,lon1],[lat0,lat1],flip(heat_data, 1));
%     title([num2str(n_subsets),' subsets of size ',num2str(subset_size*100),'%, Newtons Method on ',dataset]);
    title([num2str(n_subsets),' subsets of ',num2str(subset_samples),' samples, subset method on ',dataset]);
    hold on;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x');
    legend(tx_plot,'Transmitter');
    xlabel('lon');ylabel('lat');
    axis('xy');pbaspect([1,1,1]);
end

if plot_heat_with_diagram == 1
    heat_data = rot90(hist_data);
    figure;
    imagesc([lon0,lon1],[lat0,lat1],flip(heat_data, 1));
    title([num2str(n_subsets),' subsets of ',num2str(subset_samples),' samples, subset method on ',dataset]);
    hold on;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x');
%     rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r');
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated Location');
%     legend([rx_plot,tx_plot],'Observations','Transmitter');
    xlabel('lon');ylabel('lat');
    axis('xy');pbaspect([1,1,1]);
end

%% diagram plot
if plot_diagram == 1
    figure;
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    hold on;
    set(gca,'ydir','normal');

    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
    legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated Location');
    title([num2str(n_subsets),' subsets of ',num2str(subset_samples),' samples, subset method on ',dataset]);
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end

%% generate a final error term
error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
fprintf('             Lon %.8f\n',tx_location(1));
fprintf('TX Estimate: Lat %.8f\n',y_guess);
fprintf('             Lon %.8f\n',x_guess);
fprintf('Error: %.2f meters\n',error_m);

%% estimate power and loss_cfs
if estimate_loss_cfs == 1
    loss_guess = 2;
    % distances = lldistance(repmat(x_step(1),n_rx,1),repmat(x_step(2),n_rx,1),
        rx_location(:,1),rx_location(:,2));
```

```
250     distances = sqrt( (x_step(1)-rx_location(:,1)).^2 + (x_step(2)-rx_location(:,2))
            .^2 );
251     p_estimate = sum(rx_power .* distances.^loss_guess)/n_rx; % power transmitted if
            loss coefficient=2
252     % we have our estimate of power, now let's find loss coefficients for each
253     % point
254
255     ratios = p_estimate./rx_power;
256     loss_cfs = log(ratios(:))./log(distances(:));
257
258     if plot_loss_cfs == 1
259         figure; plot(loss_cfs);title('Loss Coefficients');
260     end
261
262     % kmeans clustering of points
263 %     n_classes = 5;
264     [IDX, centers] = kmeans(loss_cfs,n_classes); % cluster [1,2,3,2,3,2,1,1,1]
265     grp = zeros(n_rx,n_classes); % for binary indexing
266     for z = 1:n_classes
267        grp(:,z) = (IDX == z); % make an indexing variable
268     end
269
270     % define some colors
271 %     color(1,:) = [232, 42, 13]/256; % red
272 %     color(2,:) = [237, 140, 37]/256; % orange
273 %     color(3,:) = [237, 233, 36]/256; % yellow
274 %     color(4,:) = [26, 219, 29]/256; % green
275 %     color(5,:) = [19, 19, 196]/256; % blue
276
277     % centers are now from 0 to 1
278     centers_norm  = (centers-min(centers))/max(centers-min(centers));
279
280 %     color = [centers_norm.^.5*86+170, centers_norm.^2*200, centers_norm*.1*13]/256;
        % red
281     color = [centers_norm.*0+10, centers_norm.*80+10, centers_norm.*200+56]/256; %
            red
282     % darker means lower loss cf
283
284
285     if plot_classes == 1
286         figure;
287         tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker
                ','x');
288         hold on;
289         title(['observations by loss coefficients, ', dataset ]);
290         for z = 1:n_classes
291 %           grp_tmp = grp(:,z);
292             grp = (IDX == z);
293             rx_plot(z) = scatter(rx_location(grp,1),rx_location(grp,2),[],color(z,:),
                    'filled');
294 %           rx_plot(z) = scatter(rx_location(grp(:,z),1),rx_location(grp(:,z),2)
            ,[],color(z,:),'filled');
295         end
296         legend('Transmitter');
297         axis([lon0, lon1, lat0, lat1]);
298         pbaspect([1,1,1]);
299     end
300
301 end
```

B.5 7-parameter

The 7-parameter method is included below as a Matlab script.

Listing B.5: The 7-parameter method.

```
% Sam Whiting Jan 2018
% Performs the 7-parameter method on a dataset
% Uses the model with J(x0,yo,po,no,a1,a2,a3) and kmeans clustering
clear;clc;close all;

%% pick which data set to run on
% original datasets
% dataset = 'sant1';
% dataset = 'sant2';
% dataset = 'quad3';
% dataset = 'upr3';
% dataset = 'aggr1';

% new datasets
% dataset = 'upr4';
% dataset = 'aggr2';
% dataset = 'aggr2_trunc';
% dataset = 'aggr4'; % large dataset
% dataset = 'aggr6'; % average powers here
% dataset = 'aggr8'; % mixed power/averages (see read_rss_data.m)

% simulated dataset
dataset = 'sim';

%% some controls/parameters to change

% downsample amount
n_downsamp = 1;

% data truncation (what range of points to use)
truncate = 0; % flag to signal truncation or not
start = 100; % starting index
n_observations = 10; % how many points to use

% toggle plots
plot_diagram = 0;
plot_newton = 1;
plot_classes = 1;
plot_cost = 1;

% pausing
pause_each_step = 0;
auto_step = .5; % how many seconds to pause, or 0 for key prompt

% cartesian or lat/lon for newton's
use_cartesian = 0;

% psuedo inverse or dangerous inverse
use_pinv = 1;

% newtons method iterations
n_iter = 100;

%% open the file
[rx_location, rx_power, n_rx, tx_location] = ...
    read_rss_data( dataset, n_downsamp, truncate, start, n_observations, 0,0);

%% determine dimensions/edges
```

```
[lon0, lon1, lat0, lat1] = get_dimensions(dataset);

%%%% convert from lat lon to a cartesian grid
%%%% (:,1) is to access the longitudes
%%%% (:,2) is the latitudes
%%% mean_latitude = mean(rx_location(:,2)); % what latitude are we working at
%%% warp_factor = cos(mean_latitude*pi/180); % away from the equator means scale us back
%%% meters_per_deg = 111111; % meters per degree at the equator
%%%
%%% rx_location_cartesian = zeros(size(rx_location));
%%% rx_location_cartesian(:,1) = rx_location(:,1) * meters_per_deg * warp_factor; % longitudes need warping
%%% rx_location_cartesian(:,2) = rx_location(:,2) * meters_per_deg ; % latitudes are ok

%% simplified 3 parameter newton's method
x_data = rx_location(:,1);  y_data = rx_location(:,2);
x_step = [lon1; lat0; 0]; % step values. initial conditions go here

%%% x_data = rx_location_cartesian(:,1);  y_data = rx_location_cartesian(:,2);
%%% mean_location = mean(rx_location_cartesian);
%%% x_step = [mean_location'; 0]; % step values. initial conditions go here

grad = zeros(3,1);
hess = zeros(3,3);

if plot_newton == 1
    figure;
    plot(x_step(1),x_step(2),'d','LineWidth',2);
    hold on;
    title(['Newtons Method, ', dataset ]);
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x');
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end

fprintf('init:   %.4f %.4f %.4f\n', x_step(1), x_step(2), x_step(3));
for z = 1:n_iter

    % squared distance metric to be used for grad/hess
    d2 = (x_step(1)-x_data(:)).^2 + (x_step(2) - y_data(:)).^2;
%     d2 = lldistance(x_step(1), x_step(2), x_data(:), y_data(:) );

    % compute the gradient
    grad(1) = 4*sum( rx_power .* (x_step(1)-x_data(:)) .* (rx_power.*d2 - x_step(3)) );
    grad(2) = 4*sum( rx_power .* (x_step(2)-y_data(:)) .* (rx_power.*d2 - x_step(3)) );
    grad(3) = 2*sum( x_step(3) - rx_power.*d2 );

    % compute the Hessian
    hess(1,1) = 4*sum( rx_power .* (rx_power.*d2 - x_step(3)) + 2*rx_power.^2 .*(x_step(1)-x_data(:)).^2 );
    hess(1,2) = 8*sum( rx_power.^2 .* (x_step(1)-x_data(:)) .* (x_step(2) - y_data(:)));
    hess(2,1) = hess(1,2);
    hess(1,3) = -4*sum(rx_power .* (x_step(1)-x_data(:)) );
    hess(3,1) = hess(1,3);

    hess(2,2) = 4*sum( rx_power .* (rx_power.*d2-x_step(3)) + 2*rx_power.^2 .*(x_step(2)-y_data(:)).^2 );
    hess(2,3) = -4*sum(rx_power .* (x_step(2)-y_data(:)) );
```

```
115     hess(3,2) = hess(2,3);
116
117     hess(3,3) = 2*n_rx;
118
119     % do one newton step
120     x_step = x_step - hess\grad; % xstep - inv(hess) * grad
121     fprintf('step %d: %.4f %.4f %.4f\n', z, x_step(1), x_step(2), x_step(3));
122
123     if plot_newton == 1
124         if pause_each_step == 1
125             pause(auto_step);
126         end
127         plot(x_step(1),x_step(2),'d','LineWidth',2);
128     end
129
130 end
131 fprintf('\n');
132 x_guess = x_step(1);
133 y_guess = x_step(2);
134
135 % % % % let's change back to lat lon here
136 % % % x_guess = x_step(1) / meters_per_deg / warp_factor;
137 % % % y_guess = x_step(2) / meters_per_deg;
138
139 if plot_newton == 1
140     guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
141     legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
142 end
143
144 %% generate an error term
145 error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
146 fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
147 fprintf('             Lon %.8f\n',tx_location(1));
148 fprintf('TX Estimate: Lat %.8f\n',y_guess);
149 fprintf('             Lon %.8f\n',x_guess);
150 fprintf('Error: %.2f meters\n',error_m);
151
152 %% diagram plot
153 if plot_diagram == 1
154     figure;
155     rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
156     hold on;
157     set(gca,'ydir','normal');
158
159     tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
160     rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
161     guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
162     legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
163     title(['Bayes Method, ', dataset ]);
164     axis([lon0, lon1, lat0, lat1]);
165     pbaspect([1,1,1]);
166 end
167
168 %% estimate power and loss_cfs
169 loss_guess = 2;
170 % distances = lldistance(repmat(x_step(1),n_rx,1),repmat(x_step(2),n_rx,1),
    rx_location(:,1),rx_location(:,2));
171 distances = sqrt( (x_step(1)-rx_location(:,1)).^2 + (x_step(2)-rx_location(:,2)).^2 )
    ;
172 p_estimate = sum(rx_power .* distances.^loss_guess)/n_rx; % power transmitted if loss
     coefficient=2
173 % we have our estimate of power, now let's find loss coefficients for each
```

```
% point

ratios = p_estimate./rx_power;
loss_cfs = log(ratios(:))./log(distances(:));
figure; plot(loss_cfs);title('Loss Coefficients');

% kmeans clustering of points
n_classes = 5;
[IDX, centers] = kmeans(loss_cfs,n_classes); % cluster [1,2,3,2,3,2,1,1,1]
grp = zeros(n_rx,n_classes); % for binary indexing
for z = 1:n_classes
   grp(:,z) = (IDX == z); % make an indexing variable
end

    % define some colors
    color(1,:) = [232, 42, 13]/256; % red
    color(2,:) = [237, 140, 37]/256; % orange
    color(3,:) = [237, 233, 36]/256; % yellow
    color(4,:) = [26, 219, 29]/256; % green
    color(5,:) = [19, 19, 196]/256; % blue

if plot_classes == 1
    figure;
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
    hold on;
    title(['observations by loss coefficients, ', dataset ]);
    for z = 1:n_classes
%         grp_tmp = grp(:,z);
        grp = (IDX == z);
        rx_plot(z) = scatter(rx_location(grp,1),rx_location(grp,2),[],color(z,:),'
            filled');
%         rx_plot(z) = scatter(rx_location(grp(:,z),1),rx_location(grp(:,z),2),[],
    color(z,:),'filled');
    end
    legend('Transmitter');
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end
return;

%% 7 parameter newton's method

x_data = rx_location(:,1);  y_data = rx_location(:,2);

% starting location / initial conditions
% from the simplified model newtons method run
newton_start = [x_guess; y_guess];

% newton_vector = [x0 y0 p0 n0 a1 a2 a3]
newton_vector = [newton_start; 5; 4; 2.1; 2.2; 2.3]; % step values. initial
    conditions go here
newton_vector = [newton_start; 5; 4; centers]; % step values. initial conditions go
    here

grad = zeros(7,1);
hess = zeros(7,7);

if plot_newton == 1
    figure;
    plot(newton_vector(1),newton_vector(2),'d','LineWidth',2);
    hold on;
    title(['Newtons Method, ', dataset ]);
    tx_plot = scatter(tx_location(1),tx_location(2),200,'k','LineWidth',5,'Marker','x
        ');
```

```
    rx_plot = scatter(rx_location(:,1),rx_location(:,2),'r','filled');
    axis([lon0, lon1, lat0, lat1]);
    pbaspect([1,1,1]);
end

%rename a few things
xi = x_data(:);
yi = y_data(:);
p_i = rx_power(:);
%%% cf = IDX(:);
%%% al = zeros(3,1);

% variable to store cost function values
SAVE_J = zeros(n_iter,1);

fprintf('init:   %.4f %.4f %.4f %.4f %.2f %.2f %.2f\n', newton_vector(1),
    newton_vector(2), ...
  newton_vector(3), newton_vector(4), newton_vector(5), newton_vector(6),
      newton_vector(7));

for z = 1:n_iter

%     newton_vector = [x0,y0,p0,n0,al(1),al(2),al(3)];
    x0 = newton_vector(1);
    y0 = newton_vector(2);
    p0 = newton_vector(3);
    n0 = newton_vector(4);
%%%     al(1) = newton_vector(5);
%%%     al(2) = newton_vector(6);
%%%     al(3) = newton_vector(7);
    al = newton_vector(IDX + 4); % [al1 al1 al2 al3 al3 al2 al2 al1...]

    % squared distance metric to be used for grad/hess
    d2 = (xi-x0).^2 + (yi-y0).^2;

%%%     % DEBUG output our cost function to see if it really is minimized
    J = sum( p_i.^2  + n0^2 + p0^2.*d2.^(-al) - 2*p_i*n0 + 2*p0.*d2.^(-al/2).*(n0-p_i
        ) );
    SAVE_J(z) = J;
%%%%     fprintf('J at step %d: %.8f\n',z,J);

    % compute the gradient
    grad(1) = -2*sum( al .* (x0-xi) .* p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1)
        ) );
    grad(2) = -2*sum( al .* (y0-yi) .* p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1)
        ) );
    grad(3) = 2*sum( p0*d2.^(-al) + (n0-p_i).*d2.^(-al/2) );
    grad(4) = 2*sum( n0 - p_i + p0.*d2.^(-al/2) );

    idx5 = (al == newton_vector(5));
    idx6 = (al == newton_vector(6));
    idx7 = (al == newton_vector(7));

    grad(5) = sum( idx5.*( p0^2*d2.^(-al).*log(d2.^(-1)) + 2*p0*(n0-p_i).*d2.^(-al/2)
        .*log(d2.^(-1/2)) ));
    grad(6) = sum( idx6.*( p0^2*d2.^(-al).*log(d2.^(-1)) + 2*p0*(n0-p_i).*d2.^(-al/2)
        .*log(d2.^(-1/2)) ));
    grad(7) = sum( idx7.*( p0^2*d2.^(-al).*log(d2.^(-1)) + 2*p0*(n0-p_i).*d2.^(-al/2)
        .*log(d2.^(-1/2)) ));

    % compute the Hessian
    %Jxx
    hess(1,1) = -2*sum( al*p0.*(p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1))  ...
            +2*al.*(x0-xi).^2.*p0.*( p0*(-al-1).*d2.^(-al-2) + (n0-p_i).*(-al/2-1).*
                d2.^(-al/2-2) ) );
```

```
289     %Jyy
290     hess(2,2) = -2*sum( al*p0.*(p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1))  ...
291             +2*al.*(y0-yi).^2.*p0.*( p0*(-al-1).*d2.^(-al-2) + (n0-p_i).*(-al/2-1).*
                    d2.^(-al/2-2) ) );
292
293     %Jxy
294     hess(1,2) = 4*sum( al.*(xi-x0).*(yi-y0).*p0.*( p0*(al+1).*d2.^(-al-2) + (n0-p_i)
            .*(al/2+1).*d2.^(-al/2-2) ) );
295     hess(2,1) = hess(1,2);
296
297     %Jxp
298     hess(1,3) = -2*sum( al.*(x0-xi).*( 2*p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1) ) )
            ;
299     hess(3,1) = hess(1,3);
300
301     %Jyp
302     hess(2,3) = -2*sum( al.*(y0-yi).*( 2*p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al/2-1) ) )
            ;
303     hess(3,2) = hess(2,3);
304
305     %Jpp
306     hess(3,3) = 2*sum( d2.^(-al) );
307
308     %Jnn
309     hess(4,4) = 2*n_rx;
310
311     %Jxn
312     hess(1,4) = -2*sum( al.*(x0-xi).*p0.*d2.^(-al/2-1) );
313     hess(4,1) = hess(1,4);
314
315     %Jyn
316     hess(2,4) = -2*sum( al.*(y0-yi).*p0.*d2.^(-al/2-1) );
317     hess(4,2) = hess(2,4);
318
319     %Jpn
320     hess(3,4) = 2*sum(d2.^(-al/2));
321     hess(4,3) = hess(3,4);
322
323     %Jaa
324     hess(5,5) = sum( idx5.*( p0^2*d2.^(-al).*(log(d2.^(-1))).^2 + 2*p0*(n0-p_i).*d2
            .^(-al/2).*(log(d2.^(-1/2))).^2 ));
325     hess(6,6) = sum( idx6.*( p0^2*d2.^(-al).*(log(d2.^(-1))).^2 + 2*p0*(n0-p_i).*d2
            .^(-al/2).*(log(d2.^(-1/2))).^2 ));
326     hess(7,7) = sum( idx7.*( p0^2*d2.^(-al).*(log(d2.^(-1))).^2 + 2*p0*(n0-p_i).*d2
            .^(-al/2).*(log(d2.^(-1/2))).^2 ));
327
328     %Jxa
329     hess(5,1) = -2*sum( idx5.*( (x0-xi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
330                     +al.*(x0-xi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                            .^(-al/2-1).*log(d2.^(-1/2)) )  ));
331     hess(6,1) = -2*sum( idx6.*( (x0-xi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
332                     +al.*(x0-xi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                            .^(-al/2-1).*log(d2.^(-1/2)) )  ));
333     hess(7,1) = -2*sum( idx7.*( (x0-xi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
334                     +al.*(x0-xi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                            .^(-al/2-1).*log(d2.^(-1/2)) )  ));
335     hess(1,5) = hess(5,1);
336     hess(1,6) = hess(6,1);
337     hess(1,7) = hess(7,1);
338
339     %Jya
```

```
340     hess(5,2) = -2*sum( idx5.*( (y0-yi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
341                         +al.*(y0-yi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                                .^(-al/2-1).*log(d2.^(-1/2)) )  ));
342     hess(6,2) = -2*sum( idx6.*( (y0-yi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
343                         +al.*(y0-yi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                                .^(-al/2-1).*log(d2.^(-1/2)) )  ));
344     hess(7,2) = -2*sum( idx7.*( (y0-yi).*p0.*( p0*d2.^(-al-1) + (n0-p_i).*d2.^(-al
            /2-1) ) ...
345                         +al.*(y0-yi).*p0.*( p0*d2.^(-al-1).*log(d2.^(-1)) + (n0-p_i).*d2
                                .^(-al/2-1).*log(d2.^(-1/2)) )  ));
346     hess(2,5) = hess(5,2);
347     hess(2,6) = hess(6,2);
348     hess(2,7) = hess(7,2);
349
350     %Jpa
351     hess(5,3) = 2*sum( idx5.*( p0*d2.^(-al).*log(d2.^(-1)) + (n0-p_i).*d2.^(-al/2).*
            log(d2.^(-1/2)) ));
352     hess(6,3) = 2*sum( idx6.*( p0*d2.^(-al).*log(d2.^(-1)) + (n0-p_i).*d2.^(-al/2).*
            log(d2.^(-1/2)) ));
353     hess(7,3) = 2*sum( idx7.*( p0*d2.^(-al).*log(d2.^(-1)) + (n0-p_i).*d2.^(-al/2).*
            log(d2.^(-1/2)) ));
354     hess(3,5) = hess(5,3);
355     hess(3,6) = hess(6,3);
356     hess(3,7) = hess(7,3);
357
358     %Jna
359     hess(5,4) = 2*sum( idx5.*( p0*d2.^(-al/2).*log(d2.^(-1/2)) ));
360     hess(6,4) = 2*sum( idx6.*( p0*d2.^(-al/2).*log(d2.^(-1/2)) ));
361     hess(7,4) = 2*sum( idx7.*( p0*d2.^(-al/2).*log(d2.^(-1/2)) ));
362     hess(4,5) = hess(5,4);
363     hess(4,6) = hess(6,4);
364     hess(4,7) = hess(7,4);
365
366     % ja1a2 (zeros)
367     hess(5,6) = 0;
368     hess(6,5) = 0;
369     hess(5,7) = 0;
370     hess(7,5) = 0;
371     hess(6,7) = 0;
372     hess(7,6) = 0;
373
374     % do one newton step
375     if use_pinv == 0
376         newton_vector = newton_vector - hess\grad; % xstep - inv(hess) * grad
377     else
378         newton_vector = newton_vector - pinv(hess)*grad; % xstep - inv(hess) * grad
379     end
380 %%%%     Try clipping
381 %%%     for s = 4:6
382 %%%         if newton_vector(s) < 1.5
383 %%%            newton_vector(s) = 1.5; % clip at 1.5
384 %%%         elseif newton_vector(s) > 4
385 %%%             newton_vector(s) = 4; %clip at 4
386 %%%         end
387 %%%     end
388
389     fprintf('step:   %.4f %.4f %.4f %.4f %.2f %.2f %.2f\n', newton_vector(1),
            newton_vector(2), ...
390   newton_vector(3), newton_vector(4), newton_vector(5), newton_vector(6),
          newton_vector(7));
391
392     if plot_newton == 1
393         if pause_each_step == 1
```

```
394             pause(auto_step);
395         end
396         plot(newton_vector(1),newton_vector(2),'d','LineWidth',2);
397     end
398
399 end
400 x_guess = newton_vector(1);
401 y_guess = newton_vector(2);
402 fprintf('\n');
403
404 if plot_newton == 1
405     guess_plot = scatter(x_guess,y_guess,200,'k','LineWidth',3,'Marker','o');
406     legend([rx_plot,tx_plot, guess_plot],'Observations','Transmitter', 'Estimated
        Location');
407 end
408
409 if plot_cost == 1
410    figure;
411    plot(SAVE_J);
412    title('Cost Function J');
413    %    axis([0,n_iter,-10,1e15]);
414 end
415
416 % shut down if the cost function is ever negative
417 if (sum(SAVE_J < 0) > 0)
418    fprintf('\n\nWARNING: J was once negative!!\n');
419    fprintf('ending execution...\n');
420     return;
421 end
422
423 %% generate an error term the second time
424 error_m = lldistance(x_guess,y_guess,tx_location(1),tx_location(2));
425 fprintf('TX Actual:   Lat %.8f\n',tx_location(2));
426 fprintf('             Lon %.8f\n',tx_location(1));
427 fprintf('TX Estimate: Lat %.8f\n',y_guess);
428 fprintf('             Lon %.8f\n',x_guess);
429 fprintf('Error: %.2f meters\n',error_m);
```

B.6 Generating simulated data

This script generates simulated RSSI data which is placed into a .csv file. The .csv file can then be processed using the algorithm scripts above.

Listing B.6: Script to generate simulated RSSI data.

```
1  % Sam Whiting 2018
2  % generate simulated data and save it in ../sim/merged.csv
3  clear;clc;close all;
4
5  %% some controls/parameters to change
6
7  % filename
8  filename = 'merged.csv';
9
10 % number of observations to simulate
11 n_rx = 30;
12
13 % random seed for consistincy if desired
14 rng(123);
15
```

```
% loss coefficient characteristics
mean_loss_cf = 2;
std_dev_loss_cf = 0;

% additive power noise characteristics (will be rectified to be >=0)
std_dev_noise = .1;

% define dimensions/edges
bigset = 1; % 1 for a bigger area lat/lon

if bigset == 1
    lon0 = -111.813; lon1 = -111.803; % upr3 size
    lat0 = 41.74; lat1 = 41.75;
elseif bigset == 0
    lon0 = -111.814; lon1 = -111.811; % quad3 size
    lat0 = 41.7395; lat1 = 41.7425;
end

%% generate data (lat/lon here)
% transmitter
tmp = rand(2);
tx_location = [lon0*tmp(1) + (1-tmp(1))*lon1, lat0*tmp(2) + (1-tmp(2))*lat1]; %
    random location in the region
tx_power = 10;
clear tmp;

% receivers / observastions
tmp(:,1) = rand(n_rx,1);
tmp(:,2) = 1-tmp(:,1);
tmp(:,3) = rand(n_rx,1);
tmp(:,4) = 1-tmp(:,3);
lon_region = [lon0;lon1;]; % random point between longitude limits
lat_region = [lat0;lat1;]; % random point between latitiude limits
rx_location = [tmp(:,1:2)*lon_region, tmp(:,3:4)*lat_region]; % random points in the
    region
rx_distance = sqrt((tx_location(1) - rx_location(:,1)).^2 + (tx_location(2) -
    rx_location(:,2)).^2);
loss_cfs_init = mean_loss_cf + std_dev_loss_cf*randn(size(rx_distance)); % loss
    coefficients
% loss_cfs_init = [2.39;2.61;2.47]; % hard code some values
rx_power = tx_power./(rx_distance.^loss_cfs_init);
rx_power = rx_power + abs(std_dev_noise*randn(size(rx_distance))); % don't allow
    negative power noise
rx_powerdb = 10*log10(rx_power);
dataset = 'Simulation'; % fake dataset for plot titles

%% generate data (cartesian)
% removed

%% write data out to file

% lon lat elev powerdb
full_filename = ['../sim/',filename];
data = [rx_location, zeros(size(rx_power)) ,rx_powerdb];
data = [[tx_location, 0, tx_power]; data]; % add tx info as the first row
% csvwrite(full_filename,data) % 5 digit, too low precision
dlmwrite(full_filename,data,'precision',16) % 16 digits should be plenty
```

B.7 Functions used in scripts

The read_rss_data function opens a .csv file of RSSI data and returns the relevant data in vectors.

Listing B.7: Read RSSI data from a .csv file.

```
function [ rx_location, rx_power, n_rx, tx_location ] = read_rss_data( dataset_string
    , n_downsamp, ...
                                                truncate, start, n_observations,
                                                    random_selection,
                                                    n_samples_less_than_total)
% returns the RSSI data
% rx_location contains lat/lon coordinates for each observation
% rx_power contains the power received (not in db) for each observation
% n_rx is number of observations
% tx_location is the true transmitter location for the dataset (for error
% checking)

if strcmp(dataset_string,'sim')
    filename = ['../',dataset_string,'/merged.csv']; % construct file name
    data = csvread(filename);
    tx_location = data(1,:); % first row is tx location for sim
    data = data(2:end,:); % throw away first row now
    rx_location = data(:,1:2); % first two columns are coordinates
    rx_powerdb = data(:,4); % fourth column is power (db)
    rx_power = 10.^(rx_powerdb/10);
    n_rx = length(rx_power);
    return
end

%% open the file
% assumes directory structure of ../dataset/merged.csv
filename = ['../',dataset_string,'/merged.csv']; % construct file name
data = csvread(filename);
data = downsample(data,n_downsamp);
total_n_rx = length(data(:,1));

if truncate == 1
    data = data(start:start+n_observations,:);
end

if random_selection == 1
    n_samples_desired = total_n_rx - n_samples_less_than_total;
   data = data(randperm(total_n_rx,n_samples_desired),:); % randomly select n rows
end

rx_location = data(:,1:2); % first two columns are coordinates
n_rx = length(rx_location);

rx_powerdb = data(:,4); % fourth column is power in db

% for the special dataset (aggr8 so far) averages and maxes
% are both included, with avg in 4th col and maxs in 5th col
% rx_powerdb = data(:,5); % fifth column (optional for avg/max)

rx_power = 10.^(rx_powerdb/10);

%% also return the true transmitter location for error checking
if strcmp(dataset_string,'sant1')
    tx_location = [-111.8086, 41.74189];
end
if strcmp(dataset_string,'sant2')
```

```
    tx_location = [-111.8086, 41.74189];
end
if strcmp(dataset_string,'quad3')
    tx_location = [-111.812753, 41.740911];
end
if strcmp(dataset_string,'upr3')
    tx_location = [-111.805308,41.745314];
end
if strcmp(dataset_string,'aggr1')
    tx_location = [-111.805308,41.745314];
end


% new datasets
if strcmp(dataset_string,'aggr2') || strcmp(dataset_string,'aggr2_trunc')
    tx_location = [-111.805308,41.745314];
end
if strcmp(dataset_string,'upr4')
        tx_location = [-111.805308,41.745314];
end
if strcmp(dataset_string,'aggr4')
    tx_location = [-111.805308,41.745314];
end
if strcmp(dataset_string,'aggr6')
    tx_location = [-111.805308,41.745314];
end
if strcmp(dataset_string,'aggr8')
    tx_location = [-111.805308,41.745314];
end

end
```

The get_dimensions() function returns latitude and longitude dimensions for each dataset.

Listing B.8: Get dimensions of a specific dataset.

```
function [ lon0, lon1, lat0, lat1 ] = get_dimensions( dataset_string )
% get_dimensions() returns the corners (bottom left and top right)
% that contain all observatios of the dataset of interest

lon0 = 0; lon1 = 0;
lat0 = 0; lat1 = 0;

%% determine dimensions/edges
if strcmp(dataset_string,'sim')
    lon0 = -111.813; lon1 = -111.803; % upr3 size simulation
    lat0 = 41.74; lat1 = 41.75;
end

if strcmp(dataset_string,'sant1')
    lon0 = -111.809; lon1 = -111.808;
    lat0 = 41.7415; lat1 = 41.7425;
end
if strcmp(dataset_string,'sant2')
    lon0 = -111.809; lon1 = -111.808;
    lat0 = 41.7415; lat1 = 41.7425;
%%%     lon0 = -111.809; lon1 = -111.806;
%%%     lat0 = 41.7415; lat1 = 41.7445;
end
if strcmp(dataset_string,'quad3')
    lon0 = -111.814; lon1 = -111.811;
    lat0 = 41.7395; lat1 = 41.7425;
end
if strcmp(dataset_string,'upr3')
```

```
    lon0 = -111.813; lon1 = -111.803;
    lat0 = 41.74; lat1 = 41.75;
end
if strcmp(dataset_string,'aggr1')
    lon0 = -111.814; lon1 = -111.799;
    lat0 = 41.738; lat1 = 41.753;
end


% new datasets
if strcmp(dataset_string,'aggr2') || strcmp(dataset_string,'aggr2_trunc')
    lon0 = -111.854790; lon1 = -111.790018;
    lat0 =  41.727532; lat1 = 41.780853;
end
if strcmp(dataset_string,'upr4')
    lon0 = -111.854790; lon1 = -111.790018;
    lat0 =  41.727532; lat1 = 41.780853;
end
if strcmp(dataset_string,'aggr4')
    lon0 = -111.853744; lon1 = -111.790018;
    lat0 =   41.734865; lat1 = 41.780853;
end

if strcmp(dataset_string,'aggr6')
    lon0 = -111.813; lon1 = -111.803;
    lat0 = 41.74; lat1 = 41.75;
end
if strcmp(dataset_string,'aggr8')
    lon0 = -111.812390; lon1 = -111.800378;
    lat0 =  41.741021; lat1 =  41.749668;
end


end
```

The lldistance() function returns the distance in meters between two latitude/longitude coordinate pairs.

Listing B.9: Return distance between latitude and longitude points.

```
function dist = lldistance(lat1, lon1, lat2, lon2)
% return distance in meters between lat and lon points
% based on the haversine formula
% https://www.movable-type.co.uk/scripts/latlong.html
    DEG_TO_RAD = pi / 180;
    RAD_TO_DEG = 180 / pi;
    theta = lon1 - lon2;
    dist = sin(DEG_TO_RAD*lat1) .* sin(DEG_TO_RAD*lat2) + cos(DEG_TO_RAD*lat1) .* cos
        (DEG_TO_RAD*lat2) .* cos(DEG_TO_RAD*theta);
    dist = acos(dist) .* RAD_TO_DEG;
    dist = dist * 60 * 1.1515 * 1.609344 * 1000;
end
```

The power_ratio() function is used in the circles algorithm to find the loci of possible transmitter locations.

Listing B.10: Find a circle given a power ratio and observation locations

```
function [u,v,w] = power_ratio(a,b,c,d,k)
% Sam Whiting 2017
% returns a circle centered at (u,v) with radius sqrt(w).
% circle 1 at (a,b); circle 2 at (c,d)
% constant ratio of radii, circle 1 radius / circle 2 radius = k
    u = (a*k - c) / (k-1);
    v = (b*k - d) / (k-1);
    w = u^2 + v^2 - (k*(a^2 + b^2) - c^2 - d^2 )/ (k-1);
end
```

The draw_circle() function is used in the circles algorithm to draw individual loci.

Listing B.11: Plot a circle given a center and radius.

```
function h = draw_circle(x,y,r,string,width)
% draw a circle at point (x,y) with radius r
% based on mathworks support team answer:
% https://www.mathworks.com/matlabcentral/answers/98665-how-do-i-plot-a-circle-with-a-given-radius-and-center
    hold on;
    th = 0:pi/50:2*pi;
    x0 = r * cos(th) + x;
    y0 = r * sin(th) + y;
    h = plot(x0, y0,string,'LineWidth',width);
end
```