EVOLUTIONARY DATA ASSIMILATION AT LONG VALLEY CALDERA, CA

BY

THERESE MONICAL

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Geology in the Graduate College of the University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Assistant Professor Patricia Gregg
ABSTRACT

Despite advancements in volcanic modeling, the time-dependent evolution of volcanoes is still poorly understood. Of particular need are methods for combining extensive monitoring data sets with dynamic models. Sequential data assimilation has been shown to be powerful approach for linking models and data to improve the use of both. One such approach, Evolutionary Data Assimilation (EDA), previously used in hydrological predictions [Dumedah, 2012], is adapted. EDA provides a “snapshot” of parameters such as location and volume change at each timestep, allowing users to update a dynamic model’s trajectory as new observations become available. To test the application of EDA to volcano monitoring, we first develop a series of synthetic numerical experiments to track the ability of the EDA to back out chosen model parameters. Specifically, synthetic GPS and interferometric synthetic aperture radar (InSAR, a satellite based measurement of deformation) data are created from an analytical model with prescribed values for geometry and volume change. We find that EDA performs well in synthetic tests using GPS and InSAR data. After establishing the EDA method with synthetic tests, the EDA is applied to investigate the recent unrest observed at Long Valley Caldera in California using GPS from 1995-2015 and InSAR data from 2012-2014. EDA performed reasonably well at finding the location of the chamber and estimating volume changes. However, due to the analytical Mogi model used and the inherent nonuniqueness of parameters such as depth vs pressure vs radius, EDA was not able to resolve the depth or radius of the chamber. With more robust models, EDA is a powerful method that could be used to track evolution of volcanoes.
ACKNOWLEDGMENTS

Thanks to Dr. Patricia Gregg for her mentorship and for funding from her startup funds.

Thanks also to Dr. Emily Montgomery-Brown for her help, especially with obtaining GPS and InSAR data.

Thank you to Dr. Steve Marshak for reading, and to the Department of Geology for a teaching assistantship that helped to fund my education.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>GEOLOGIC BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>METHODS</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>SYNTHETIC TEST RESULTS</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>APPLICATION TO LONG VALLEY</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>DISCUSSION</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>CONCLUSIONS</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>BIBLIOGRAPHY</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>APPENDIX A: PYTHON CODE FOR EDA</td>
<td>37</td>
</tr>
</tbody>
</table>
Chapter 1: INTRODUCTION

Active volcanic systems present hazards to surrounding populations, and large eruptions can cause regional and global environmental impacts. Supervolcanoes, by nature of their extreme size (ejecting upwards of 1000 km$^3$ material), present hazards that are unique in scale. Although rare, occurring every 10,000-100,000 years, supereruptions are so devastatingly large that it is difficult to predict their impact on Earth’s climate as a whole or how long that impact might persist [Self and Blake, 2008]. In terms of human impact, a supereruption is hypothesized to have caused a human population bottleneck 74 ka [Ambrose, 2003], possibly leading to the differentiation of human races 70 ka. A similar event today would be extremely destructive. However, critical gaps remain in our understanding of how large eruption systems evolve over time.

Long Valley Caldera, located in California, is a supervolcanic system which has previously hosted large eruptions, including the the Bishop Tuff eruption 760,000 years ago (750 km$^3$ erupted material). Unrest at Long Valley Caldera beginning in 1980 includes deformation and swarms of seismicity associated with magma influx [Bailey and Hill, 1990], as well as geochemical signals such as outgassing of CO$_2$ [Sorey et al., 1993], has led to a resurgence of interest in the current state of this system and its potential for producing another catastrophic caldera forming eruption. Geophysical measurements of the recent unrest, including tilt, surface deformation (by leveling since 1905 but using more modern methods after 1980 [Battaglia et al., 2003a, Rundle and Hill, 1988], by GPS from 1995, and by InSAR from 2011 [Montgomery-Brown et al., 2015, Battaglia et al., 2003a]), horizontal deformation using two-color Electronic Distance Measuring(EDM), seismic studies, and microgravity [Battaglia et al., 2003a], provide critical observations into the current state of activity and have prompted a variety of inverse models, forward models, and viscoelastic models [Montgomery-Brown et al., 2015, Battaglia et al., 2003a, Battaglia et al., 2003b, Newman et al., 2001, Hill et al., 2003].

Models of Long Valley from previous efforts have indicated that there is a persistant reservoir located under the resurgent dome [Montgomery-Brown et al., 2015] with episodic magma intrusion.
Efforts to model the system have sometimes concluded that there are two inflation sources [Bailey and Hill, 1990, Savage et al., 1987] but sometimes have been unable to resolve the second source in the south moat [Montgomery-Brown et al., 2015]. Because point sources do not represent the full geometry of the chamber, it is possible that this “second source” was an artifact of the model used [Savage et al., 1987]. Other studies which used more complex models have not used two sources; for example the shape of the chamber has been modeled as a prolate ellipsoid [Battaglia et al., 2003b] and has been modeled using finite element modeling software [Long and Grosfils, 2009]. However, despite spatial improvements in models, it is still unclear how the chamber is evolving in time. This is where sequential data assimilation methods become a useful tool.

Sequential data assimilation methods are methods which assimilate data into their algorithm sequentially, with new data added to the algorithm when it is available. This provides “snapshots” of the system at each timestep, which provides insight into how the system is evolving. Sequential data assimilation methods have been successfully utilized in hydrological, meteorological, and other nonlinear systems where a large number of time-dependent data need to be synthesized into a cohesive model. Unlike static inversions [Dzurisin et al., 2006, Battaglia et al., 2003a, Battaglia et al., 2003b], data assimilation methods provide a snapshot of the system at each timestep where there is new data [Dumedah, 2012]. Over the past two decades, several approaches based on such methods, such as the classic Kalman Filter [Kalman, 1960] and Extended Kalman Filter [Schmidt, 1966] have been adapted for volcanological problems [Segall, 2013, Anderson and Segall, 2013]. Unfortunately, these approaches are limited to the use of analytical solutions and can be computationally taxing to implement. More recently, Monte Carlo approaches have been used to calculate the covariance matrix in the Kalman Filter, thus improving forecasting ability in highly nonlinear systems [Gregg and Pettijohn, 2016, Anderson and Segall, 2013]. These approaches have the potential for greatly improving the use of monitoring data collected at active systems worldwide. However, there is still significant work to be done to further develop and test these methods.

In this study, we test a new data assimilation approach recently developed to forecast highly...
dynamic and nonlinear hydrologic systems: the Evolutionary Data Assimilation algorithm (EDA) [Dumedah, 2012]. EDA utilizes a genetic algorithm at each step. Genetic algorithms simulate evolution to find a population of best-fit models over the course of several generations. In the case of EDA, each timestep is a separate genetic algorithm, with the initial population coming from the last generation of the previous timestep. EDA and methods derived from it have previously been used to great success in hydrological forecasting studies [Dumedah, 2012, Dumedah and Coulibaly, 2013, Dumedah and Coulibaly, 2014]; however, its use in volcanic forecasting is unprecedented. In hydrological studies, EDA has been shown to outperform other data assimilation methods including the Particle Filter and the Ensemble Kalman Filter (EnKF) [Dumedah and Coulibaly, 2013], indicating that it may have great potential in for assessing unresting volcanoes. We have adapted EDA in Python for use in volcanic data assimilation in order to track the parameters governing the evolution of Long Valley Calderas magma chamber. In the following sections we briefly describe the technique, including synthetic tests of its implementation and then apply the EDA to GPS data from Long Valley to investigate the recent unrest events and the evolution of the Long Valley magmatic system over the past two decades. EDA shows great promise as a tool in assessing the evolution of volcanic systems.
Chapter 2: GEOLOGIC BACKGROUND

Figure 1: Map of Long Valley with GPS stations utilized in this study.

Long Valley Caldera, located in California east of the central Sierra Nevada (see map inset in Figure 1), has hosted eruptions both enormous and small. The caldera was formed by a supereruption 760,000 years ago, which blanketed much of the western United States in ash and formed the Bishop Tuff, hundreds of meters thick proximal to the caldera [Hildreth and Mahood, 1986]. Smaller eruptions have occurred both along the ring fault of the caldera until about 100,000 years ago and along the rim of the caldera most recently 16,000-17,000 years ago [Rundle and Hill, 1988].

There have been four episodes of inflation recorded at Long Valley over the past three decades.
These episodes are associated with hazards such as death of livestock due to outgassing of magmatic CO$_2$ [Lucic et al., 2015] as well as increased seismicity [Bailey and Hill, 1990]. During the most recent episode, over the past 6 years, Long Valley has experienced renewed inflation resulting in 80 mm of uplift at a rate of about 10 mm/yr [Montgomery-Brown et al., 2015], and the total uplift over all episodes is 80 cm [Hill, 2006, Montgomery-Brown et al., 2015]. The observed deformation in the geodetic signal as well as increased seismicity has motivated several investigations to constrain the source and evolution of unrest. Previous efforts have utilized inversions and joint inversions of gravity and deformation data to provide a 3-dimensional and 4-dimensional view of the evolution of the magma plumbing system [Battaglia et al., 2003a, Battaglia et al., 2003b, Newman et al., 2001]. Battaglia et al [2003a, 2003b] focused on microgravity, two-color EDM, and uplift measurements, showed that modeling the source with a spherical chamber as in a Mogi model biased the results, increasing the predicted density, volume change, and source depth significantly, and reduced the fit when the horizontal displacement was taken into consideration. Given this information, a Mogi model was deemed insufficient to reproduce the recent inflation of the Long Valley system. Other analytical solutions and static inversions improve the accuracy of the models, but are limited as they do not provide a way to systematically reproduce the temporal evolution of the system and what their eruption precursors may look like.

![Seismicity at Long Valley, 1983-2017](image)

Figure 2: Adapted from USGS Seismic Monitoring at Long Valley Caldera webpage [USGS, ]. The vertical black bars represent the number of earthquakes per week; the red line is the cumulative earthquake count since monitoring began in 1983. Note episodes during which there are large numbers of earthquakes, most of very small magnitude, indicating unrest.
Figure 3: From Hill (2006). Unrest at Long Valley shown via uplift (recorded by GPS, tiltmeter, and InSAR) and seismicity.
Chapter 3: METHODS

3.1 Evolutionary Data Assimilation (EDA)

EDA is an evolutionary-algorithm-based approach to data assimilation. In genetic algorithms, a set of solutions to a problem (the “population”) is generated randomly, then the process of natural selection is simulated to improve and refine solutions. Generally, the population is sorted by fitness (determining the fitness is often the most computationally expensive part of the algorithm), and the least-fit solutions removed. The remaining solutions are used to generate a new generation of solutions, and mutation may be applied to slightly alter the new generation. The process of selection and repopulation is repeated for a number of generations, either a set number or until a certain fitness criterion has been reached.

EDA uses a genetic algorithm at each timestep, using the last generation of the previous timestep as the initial population for the current timestep. The details of the process are outlined below.

3.2 Computational Approach

The following is a discussion of the implementation of EDA in Python. Full code is provided in Appendix A, which follows the methods below. This method takes GPS and InSAR data as inputs and at each timestep outputs a best-fit set of parameters (Radius, location, depth, and dV). Because a population of Mogi models (see Figure 4) is used, it is possible to output the range of the modeled values as well as the best-fit-values to provide a sense of how well the population as a whole agrees with the best-fit model.

Initially, each solution (a set of the parameters radius, overpressure, x-position, and y-position) was generated randomly, with each parameter within a range defined in Table 2, as is the number of solutions in the population of solutions. The population of these solutions was used as the initial population for the genetic algorithm during the first timestep. (During subsequent timesteps, the
final population from the last timestep is reused as the initial population for the new timestep).

Following the methods of Dumedah 2012, at each timestep, a number of generations were simulated. At each generation, the following steps were performed:

1. A fitness function was used to determine, for each solution, how well a model with the parameters in that solution fit the data. This fitness function used -RMSE (root mean squared error) as a fitness value, and the fitness values were used to sort the solutions from most to least fit.

2. The least-fit half of the population was discarded while the most-fit half was kept. The two most-fit solutions in the entire population were cloned and set aside for the moment.

3. For the last generation at each timestep only, the best, worst, and average solutions were recorded and saved, as well as the RMSE for each of these solutions.

4. The remaining members of the population were allowed to reproduce to produce an entirely new population. First, two surviving members of the population were selected at random. For each “gene” (parameter) in the child solution, a gene was randomly selected from one parent solution or the other. This was repeated until a new population of correct size was obtained.

5. The old population was deleted and the new population became the only population.

6. The population is mutated slightly according to the parameters in Table 1. As generations progressed to their maximum within a timestep, the mutation amount was decreased to zero. An alternative to this method with a similar effect on the method might be to use a smaller mutation amount but more generations, but this alternative would be slower.

7. The cloned members of the old population are placed into the population again. This ensures that best-fit solutions are never lost during the mutation/recombination process.

8. The final population for this generation becomes the initial population for the next generation.
3.3 Analytical model

![Figure 4: The setup of the Mogi model, to scale for the synthetic tests.](image)

There are many possible choices for analytical models of surface deformation based on different chamber geometry as well as different rheologies, and the choice between a viscoelastic time-dependent model and a static model was not trivial.

While previous studies [Newman et al., 2001] have found evidence of viscoelastic roll-off at Long Valley, this trend was not readily apparent in the somewhat-noisy GPS and InSAR data used in this study. A time-dependent viscoelastic model would also have required either an prescribed number and timing of intrusions, or would have required that the genetic algorithm find these timings and numbers as well as parameters such as radius, chamber location, chamber depth, and volume change. With every parameter solved for, the genetic algorithm has an exponentially larger search space and will require more time and computational resources to find a reasonable solution. Thus, a viscoelastic model is in this case not trivial. However, with additional resources it would be possible.

Because of these limitations in both data and computational resources, an elastic and analytical Mogi model [Mogi, 1958] was used as a proof-of-concept. It is likely that this model biases the solutions found to be deeper and to have a higher volume change [Battaglia et al., 2003b].
3.4 Synthetic Test of the EDA

Because this is the first-ever application of this method in a volcanological setting, synthetic tests were performed to test the concept.

To create the synthetic datasets, first a set of randomly placed GPS stations were generated. GPS stations produced data every day and there were a maximum of 25 stations online at any given timestep. InSAR was generated similarly, but with data produced once every 30 days and with many more sample points (250 per timestep). For datasets including GPS stations, the number of GPS stations at each timestep increased linearly with time to the maximum to simulate GPS stations being placed as time progressed.

The synthetic datasets were created with a Mogi model which used a given pressure, radius, location and depth of the magma chamber to calculate a displacement value at each synthetic GPS and/or InSAR location. For Figs 6–8 all of these values were held constant for ease of comparison.
Chapter 4: SYNTHETIC TEST RESULTS

EDA was used on the synthetic data to find the chamber location and depth, the volume change at each timestep, and the radius of the chamber. Results can be seen in figures 6–8. Note the good match between predicted and real latitude, longitude, and dV in all 3 figures. The InSAR-only case, while still performant, is not as precise as cases with GPS included. Only the case including both GPS and InSAR is able to capture a depth, though this depth does not match previous studies (green line). Cases where the predicted value of a parameter flatten against the lower or upper bound (for example, figure 6 (c)) are divergent and cannot predict that value.

<table>
<thead>
<tr>
<th>Mogi Parameter</th>
<th>Values</th>
<th>Mean Value EDA (GPS Only, last 3 timesteps)</th>
<th>Mean Value (EDA) (InSAR Only, last 3 timesteps)</th>
<th>Mean Value (EDA) (GPS and InSAR, last 3 timesteps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dV (m$^3$)</td>
<td>1.01 * 10$^6$</td>
<td>20.8 * 10$^6$</td>
<td>27.4 * 10$^6$</td>
<td>9.37 * 10$^6$</td>
</tr>
<tr>
<td>radius (m)</td>
<td>2000</td>
<td>8000.</td>
<td>696</td>
<td>1157</td>
</tr>
<tr>
<td>depth (m)</td>
<td>5000</td>
<td>21000.</td>
<td>11969</td>
<td>8838</td>
</tr>
<tr>
<td>x (longitude)</td>
<td>-118.53</td>
<td>-118.56</td>
<td>-118.67</td>
<td>-118.54</td>
</tr>
<tr>
<td>y (latitude)</td>
<td>37.57</td>
<td>37.59</td>
<td>37.68</td>
<td>37.56</td>
</tr>
</tbody>
</table>

Table 1: Values of synthetic parameters and the solutions found by EDA in GPS only, InSAR only, and GPS+InSAR cases

<table>
<thead>
<tr>
<th>E</th>
<th>Young’s Modulus</th>
<th>75 GPa</th>
</tr>
</thead>
<tbody>
<tr>
<td>eta</td>
<td>Poisson’s ratio</td>
<td>25</td>
</tr>
<tr>
<td>G</td>
<td>Shear Modulus</td>
<td>$E/(2*(1+\eta))$</td>
</tr>
<tr>
<td>K</td>
<td>Bulk Modulus</td>
<td>$E/(3*(1-2*\eta))$</td>
</tr>
<tr>
<td>z_range</td>
<td>Allowed range of depth</td>
<td>[1000, 21000] m</td>
</tr>
<tr>
<td>x_range</td>
<td>Allowed range of x</td>
<td>[310000, 380000] utm zone 11S</td>
</tr>
<tr>
<td>y_range</td>
<td>Allowed range of y</td>
<td>[4140000, 4200000] utm zone 11S</td>
</tr>
<tr>
<td>delP_range</td>
<td>Allowed range of pressure</td>
<td>[-10e8, 10e8] Pa</td>
</tr>
<tr>
<td>r_range</td>
<td>Allowed range of radii for the magma chamber</td>
<td>[100, 8000]</td>
</tr>
<tr>
<td>Population</td>
<td>Population in the genetic algorithm</td>
<td>250</td>
</tr>
<tr>
<td>Generations</td>
<td>Generations in the genetic algorithm</td>
<td>25</td>
</tr>
<tr>
<td>Mutation amount</td>
<td>Maximum amount that a gene can mutate in one generation (Expressed as a percent of the total range for that parameter)</td>
<td>20%</td>
</tr>
<tr>
<td>Mutation Probability</td>
<td>Probability that a given gene will mutate at all before being passed on to a child model</td>
<td>75%</td>
</tr>
</tbody>
</table>

Table 2: Values of parameters used in this study
Figure 5: An example plot of randomly generated GPS stations used in EDA synthetic tests
Figure 6: Synthetic test of EDA using GPS data only. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; Green horizontal line: The true value of each parameter from the synthetic model that generated the synthetic data.
Figure 6: Continued
Figure 7: Synthetic test of EDA using InSAR data only. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; Green horizontal line: The true value of each parameter from the synthetic model that generated the synthetic data.
Figure 7: Continued
Figure 8: Synthetic test of EDA using both GPS and InSAR data. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; Green horizontal line: The true value of each parameter from the synthetic model that generated the synthetic data.
Figure 8: Continued
Chapter 5: APPLICATION TO LONG VALLEY

GPS and InSAR data collected during the unrest period were obtained from the USGS. GPS data was pre-filtered and detrended when retrieved from the USGS website. There were 22 GPS stations in the dataset, but not all stations were active at all timepoints. In earlier timepoints (starting in 1998) there were at times as few as 3 active GPS timepoints. By the last timepoint (late 2015), however, most or all stations were generally active at one time. At timepoints where GPS sensors were inactive, those stations were not used for data, and RMSE during the fitness function was determined based only on active stations at that timestep.

InSAR data is available from mid 2011 through late 2014. The data is in quad tree format with 314 locations defined (see figure 9). Locations are the centers of rectangles, and the area of these rectangles were used to weight InSAR data.

EDA was used as with the synthetic data, and the results can be seen in figures 11–15.

For GPS-only data, X is found with excellent precision, and Y is found with good precision. Y varies slightly, with EDA’s best-fit model before 2006 being somewhat south of the location found by Montgomery-Brown [2015]. After 2006, the model is still near Montgomery-Brown’s location, but slightly north of it. RMSE is generally low, but does spike high at times. From figure 12 it is clear that the best-fit model at each timestep fits the available uplift data very well. As in synthetic data, there is no way for this model to determine the radius of the chamber uniquely, so dV is used as a factor that incorporates both radius and pressure.

For InSAR-only data, X is in the range of values found by EDA, but Y is not except at one timepoint. Depth and dV are not located at all. It is possible that there are data errors in the InSAR data, which may be why the InSAR case performs so poorly.

For the case with both GPS and InSAR, the plot looks very much like the GPS-only plot. Since InSAR is only available from 2012-2014, the only parts of this plot that should be significantly different than the GPS-only case are in this timespan. Indeed, it does look much the same, with the exception of dV in the timeframe of 2012-2014, which shows large negative and positive volume
changes during this time, where GPS alone showed none. The fit to uplift is still good, as shown in figure 15.
Figure 6. Initial data at long valleys (data were treated as points and weighted by the area of the bounding rectangle (not pictured).
Figure 10: Availability of data from GPS stations in the Long Valley region. Dotted line indicates when GPS station was active. Small gaps indicate that the station was temporarily down.
Figure 11: Application of EDA to Long Valley using GPS data only. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; Green horizontal line: The “true value” from previous publications [Montgomery-Brown et al., 2015, Battaglia et al., 2003b, Battaglia et al., 2003a]
Figure 11: Continued
Figure 12: GPS only. Observed uplift (green) and EDA modeled uplift (blue) at GPS stations at Long Valley. Station TILC has observed data, but it is outside the bounds of this figure.
Figure 13: Application of EDA to Long Valley using InSAR data only. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; The “true value” from previous publications [Montgomery-Brown et al., 2015, Battaglia et al., 2003b, Battaglia et al., 2003a]
Figure 14: Application of EDA to Long Valley both GPS and InSAR data. (a) Volume change; (b) Radius of the chamber; (c) Depth to the center of the chamber; (d) Longitude of the chamber center; (e) Latitude of the chamber center; (f) RMSE for solutions. Light blue: Range of the entire population of models at each timestep; Purple: best-fit model; Green horizontal line: The “true value” from previous publications [Montgomery-Brown et al., 2015, Battaglia et al., 2003b, Battaglia et al., 2003a]
Figure 14: Continued
Figure 15: GPS and InSAR. Observed uplift (green) and EDA modeled uplift (blue) at GPS stations at Long Valley. Station TILC has observed data, but it is outside the bounds of this figure.
Figure 16: Adapted from Montgomery-Brown et al (2015). Results from this study compared to previous studies. Red arrow indicates the direction of motion of the modeled magma chamber center over time while the black x indicates the two major locations this study found for the magma chamber.
Chapter 6: DISCUSSION

6.1 Synthetic tests

In synthetic tests, EDA performed well. In GPS-only and GPS+InSAR tests, EDA was able to find both X and Y (longitude and latitude) with good precision. EDA has difficulty resolving depth and radius, which is expected; the nonuniqueness of the model used makes it difficult to separate these two parameters. However, the volume change parameter, which incorporates both of these values in a way that cancels out the nonuniqueness, matches the true value very well, never varying from the true parameter more than approximately 2% of the searched space and generally much closer (< 0.5%) than that.

The main factor which limits the effectiveness of the EDA approach in this research is the model used. Because of the inherent nonuniqueness of the analytical Mogi model used, it is not possible to retrieve the radius and depth from the data. If more computational resources are available, it would be possible to run more complicated models such as an analytical viscoelastic model or a finite element model (as in Gregg 2016) with more complex geometry and rheology.

6.2 Long Valley

In the results of the EDA applied to Long Valley, the X and Y locations agreed relatively well with the location of the resurgent dome. X agreed better than Y, with Y moving from slightly south of the resurgent dome to alternating between slightly south and slightly north of it in late 2005-early 2006. There is no associated change in seismicity or deformation that is obvious in the literature or in the data, and it is possible that this location shift is a “second source” that is an artifact of the model used, as was a problem for previous studies. Because the model used is a point source Mogi model, it cannot capture the geometry of the chamber, which is almost certainly much more complex than a sphere. Other studies have showed that a variety of ellipsoidal models and
time-dependent rheologies might fit better, with a Mogi model being much too simple to capture the complexity of what is happening within Long Valley caldera [Montgomery-Brown et al., 2015, Battaglia et al., 2003b, Battaglia and Hill, 2009, Newman et al., 2001, Hildreth and Wilson, 2007]. It still not clear, however, what event might have caused the complexity to reach a “tipping point” where EDA could no longer find a single best-fit solution for the Y-position.

The depth that EDA found is uncertain. Because it switches so rapidly from the minimum allowed by the code (1000 m) to the maximum (21000 m) it is likely that EDA could not capture any particular depth. The GPS data used is very noisy, and it is possible that that random error overwhelmed enough of the signal that EDA could no longer find the depth. It is also possible that the depth is more complex than a Mogi model can capture, and that no Mogi model provided a good enough fit for EDA to find that fit as a solution.

Overall EDA performed well given the constraint of the Mogi model used, and with further research this approach may provide interesting time-dependent results.
Chapter 7: CONCLUSIONS

Overall, EDA is a promising and powerful method with the potential to be very useful in the volcanological field. It has the power to find chamber location and volume changes, even with a Mogi model as the model of choice. However, because this model is so simple, it misses some complexities of the chamber’s geometry. It is very important to use a more powerful model in future research in order to be able to resolve the various parameters that EDA searches for. A Mogi model, while a promising first step, is only the beginning.
Bibliography


APPENDIX A: PYTHON CODE FOR EDA

"""

main.py

Evolutionary Data Assimilation Algorithm Method for Volcanic Data Assimilation

Main file

by Therese Monical"

#standard modules
import random
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
from matplotlib.backends.backend_pdf import PdfPages
import scipy.io as sio
import utm
import string
import os #just for alarm at end to let you know code is done

#modules I've defined elsewhere
import fitness
from load_data import *

"""
from save_data import *
from mogiModel import mogiModel
from VE_model import VEModel
import mutate
import recombine
from get_gps_locs import *

"""These are used to plot but not in actual calculations (yet)"""
""" in mogiModel.py and VE_model.py for values used in calculations-
They SHOULD be the same"""

E   = 75.0 * (10**9) #Pa
eta = .25 #poisson's ratio
G   = E/(2*(1+eta)) # shear modulus
K   = E/(3*(1-2*eta)) # bulk modulus

"""Flags to change how data is used"""

synth_data = 0 #0 for real data, 1 for mogi, 2 for VE
method     = 0 #0 for mogi model, 1 for VE
save_to_matlab = False
truncate_t_for_testing = False

"""Volcano parameters"""

r1_range    = [100, 8000] #chamber radius (vertical) in meters
#r2_range    = [100, 2000] #chamber radius (horizontal) in meters
delP_range  = [-10e8, 10e8] #pressure/lithostatic pressure (Pa)
z_range     = [1000, 21000] #depth of the chamber center in meters
x_range = [310000, 380000] # center of the chamber - x (m utm)
y_range = [4140000, 4200000] # center of the chamber - y (m utm, all zone 11S)

parameters_ranges = [r1_range, delP_range, z_range, x_range, y_range] # put these all in an array for later

num_par = len(parameters_ranges)
num_stations_synth = 50
num_InSAR_synth = 250
num_days_synth = 365*2

### "Parameters for GPS Stations"
# coords to search for GPS stations
x_GPS = [-119.1, -118.6]
y_GPS = [37.5, 37.8]

### "Parameters for the genetic algorithm"
population = 250 # members of our population per generation
generations = 25 # number of generations to simulate at each timestep
selection_percent = .6 # percent of population that dies every generation
mutation_amount = .2 # max percent that a gene can mutate in one mutation
mutation_prob = .75 # percent chance that a gene will be mutated

### "Load data (real or synthetic)"
day = 60*60*24 # one day in seconds
year = 365.25 * 24 * 60 * 60

time = []
time_InSAR = []
if synth_data != 0: #if synthetic data

    #if synthetic WITH GPS:
    time = np.arange(0*day, num_days_synth*day, 2*day).tolist()

    #if synthetic with NO GPS:
    # for _ in range(round(num_days_synth/2)):
    #    time.append(-1.0)

    #if synthetic WITH InSAR
    time_InSAR = np.arange(0*day, num_days_synth*day, 30*day).tolist()

    #if synthetic with NO InSAR:
    # for _ in range(round(num_days_synth/30)):
    #    time_InSAR.append(-1.0)

#synthetic data parameters
u_data = []
n_data = []
e_data = []
test_delP = 10e6
test_r    = 2000
test_d    = 5000
test_chamber_x = 365000
test_chamber_y = 4160000

#create a bunch of GPS stations
x = np.arange(x_range[0], x_range[1], 200).tolist()
y = np.arange(y_range[0], y_range[1], 200).tolist()
#create grid of "stations"
xy_total = []
for i in range(len(x)):
    for j in range(len(y)):
        xy_total.append((x[i], y[j]))
#Select random stations from our bunch to be the actual stations- randomizes locations, basically
xy_InSAR_substep = random.sample(xy_total, num_InSAR_synth)
xy_total = random.sample(xy_total, num_stations_synth)

xy_InSAR = []
for i in range(len(time_InSAR)):
    xy_InSAR.append(xy_InSAR_substep)

#create a file of station names and locations
names_total= []
x_sta = {}
y_sta = {}
for i in range(num_stations_synth):
name = ''.join(random.choice(string.ascii_uppercase +
string.digits) for _ in range(4))

names_total.append(name)

LL = utm.to_latlon(xy_total[i][0], xy_total[i][1], 11, 'S')
x_LL = LL[1]
y_LL = LL[0]
x_sta[name] = x_LL
y_sta[name] = y_LL

stations = {}
stations['names'] = names_total
stations['x'] = x_sta
stations['y'] = y_sta

#save file
with open('stations.pickle', 'wb') as f:
    pickle.dump(stations, f)
print("Station data saved!!")

xy = []
names = []
for i in range(len(time)):
    stations_this_turn = []
    max_missing = max(num_stations_synth - 1, 0)
max_missing *= (len(time)-i)/len(time)  # more and more stations are likely to be online as time goes on

max_missing_int = round(max_missing)
missing_stations = random.randint(0,max_missing_int)

names_this_turn = []
for j in range(num_stations_synth-missing_stations):
    new_station = random.randint(0,num_stations_synth-1)
stations_this_turn.append(xy_total[new_station])
    names_this_turn.append(names_total[new_station])
names.append(names_this_turn)
xy.append(stations_this_turn)

errors = {}
for name in names_total:
    errors[name] = random.gauss(.004, .001)

# plot stations
if len(xy_total) != 0:
    f2 = plt.figure(figsize=(10,10))
    ax = f2.add_subplot(111)
    for point in xy[-1]:
        ax.plot(point[0], point[1], 'x')
    plt.savefig('stations_synth.png')  # save BEFORE showing- otherwise saved figure will be blank
    plt.show()
td_InSAR = []
look_InSAR = []
err_InSAR = []
for i in range(len(time)):
    if synth_data == 1:
        datapoints = mogiModel(xy[i], test_r, test_delP, test_d, test_chamber_x, test_chamber_y, time[i])
    else:
        datapoints = VEModel(xy[i], test_r, test_delP, test_d, test_chamber_x, test_chamber_y, time[i])
    u_data.append(datapoints[0]) #append "up" data to u_data
for GPS
for i in range(len(time_InSAR)):
    if synth_data == 1:
        datapoints = mogiModel(xy_InSAR[i], test_r, test_delP, test_d, test_chamber_x, test_chamber_y, time_InSAR[i])
    else:
        datapoints = VEModel(xy_InSAR[i], test_r, test_delP, test_d, test_chamber_x, test_chamber_y, time_InSAR[i])
    u_InSAR = datapoints[0]
    Un_data = datapoints[1]
    Ue_data = datapoints[2]
    n_timestep = []
e_timestep = []
td_InSAR.append([])
err_InSAR.append([])
look_InSAR.append([])
for j, Ur in enumerate(Un_data):
    td_timestep = (Un_data[j]**2 + u_InSAR[j]**2 + Ue_data[j]**2)**0.5
    td_InSAR[i].append(td_timestep)
root_2 = 1/(2**0.5)
    look_InSAR[i].append((0, root_2, root_2))
er_InSAR.append(.01)

areas = []
for i in range(num_InSAR_synth):
    areas.append(1)

# print(data)
# elif synth_data == 2:
#     datapoints = VEModel(xy, test_r, test_delP, test_d,
# test_chamber_x, test_chamber_y, time)
#     u_data = datapoints[0] # add "up" data to u_data
#     Ur_data = datapoints[1]
# #
# # #
# # for i, t in enumerate(time):
# #     n_timestep = []
# #     e_timestep = []
# #     for j, Ur in enumerate(Ur_data):
x_component = xy[i][j][0] - test_chamber_x
y_component = xy[i][j][1] - test_chamber_y
dist = (x_component**2 + y_component**2)**0.5
n_comp = Ur*x_component/dist
z_comp = Uz

# td_timestep.append(Ur*x_component/dist) #we're gonna say we're looking from 45 degrees up, looking due north

# u_data.append(datapoints[0]) #append "up" data to u_data

# e_data.append(n_timestep) #append radial data to e_data and n_data

e_data.append(n_timestep)
else:
    print("synth_data's value is invalid!!")
elif synth_data == 0: #if we're using real data

    u_data = []
    xy = []
    names = []
    time = []
e_data = []
n_data = []
errors = []

    R_2015_a = 3000
R_2015_b = 1700
DEPTH_THEORETICAL = 6000
RDOM_latlon = (37.67707, -118.89794)
RDOM_utm_temp = utm.from_latlon(RDOM_latlon[0], RDOM_latlon[1])
RDOM_utm = (RDOM_utm_temp[0], RDOM_utm_temp[1])

def get_gps_locs(): # saves the locations in a pickle file called stations.pickle for later use during plotting
    xy_err = get_data('data_dict.pickle', x_GPS[0], x_GPS[1], y_GPS[0], y_GPS[1]) # xy_err is a dict containing several processed variables
    xy = xy_err['xy']
    names = xy_err['names']
    names_total = xy_err['names_total']

    with open("errors_from_stdev.pickle", 'rb') as pickled_file:
        error_pickle = pickle.load(pickled_file)
    errors = error_pickle["errors"]

    u_data = xy_err["u"]
    time = xy_err["time"]

    with open("InSAR_pickled_data.pickle", 'rb') as pickled_file:
        InSAR_data = pickle.load(pickled_file)

        xy_InSAR = InSAR_data['xy']
        td_InSAR = InSAR_data['data']
look_InSAR = InSAR_data['look']
err_InSAR = InSAR_data['errs']
time_InSAR = InSAR_data['time']
#names_InSAR= InSAR_data['names']
areas = InSAR_data['areas']

#
# # #use this section if not using InSAR
#
# xy_InSAR = []
# td_InSAR = []
# look_InSAR = []
# err_InSAR = []
# time_InSAR = []
# #names_InSAR= InSAR_data['names']
# areas = []

print("timesteps:" +str(len(time)))
print("number of data timesteps: " + str(len(xy)))
if synth_data == 0:
    print("number of error timesteps: " + str(len(errors)))

if synth_data == 0 and truncate_t_for_testing:
    time = time[1000:1200]
    print("Timesteps truncated for testing!!")
```python
time_total = np.unique(time + time_InSAR)

for i in range(num_par):
    average_values.append([])
    min_values.append([])
    max_values.append([])
    st_dev.append([])
    best_member.append([])

for j in time_total:
    average_values[i].append(0.0)
    min_values[i].append(0.0)
    max_values[i].append(0.0)
    st_dev[i].append(0.0)
    best_member[i].append(0.0)
```
for i in time_total:
    RMSE_max.append(0.0)
    RMSE.append(0.0)
    RMSE_min.append(0.0)

"""Initial population""

# Generate initial population
P = [] # the members of our population will go here
for j in range(population*10): # each round of the loop makes one member of a population. The initial population is larger than it will be later
    member = [] # the member we're creating
    member.append(random.uniform(parameters_ranges[0][0], parameters_ranges[0][1])) # get the radius
    member.append(random.uniform(parameters_ranges[1][0], parameters_ranges[1][1])) # get the delP
    member.append(random.uniform(max(member[0], parameters_ranges[1][0]), parameters_ranges[2][1])) # we make sure that the depth is greater than the radius
    for r in range(num_par-3): # each loop adds a gene to the member we're making
        member.append(random.uniform(parameters_ranges[r+3][0], parameters_ranges[r+3][1]))
    P.append(member) # add the member to the population now that it's ready
    # print(member)
"""Main loop"""

for t_i, t in enumerate(time_total):

    if t<0:
        td_InSAR_step = []
        look_InSAR_step = []
        xy_InSAR_step = []
        areas_step = []

        u_GPS_step = []
        xy_step = []
        names_step = []

    else:

        print("TIME "+ str(t_i)+": " + str(t/year) + " years")
        #print("\tPopulation: " + str(len(P)))

        if t_i == 0:
            gens = generations *3
        else:
            gens = generations

        #a little rearranging to make timesteps work correctly
        t_check = time_InSAR[0]
        i = 0
try:
    while t_check <= (t-day):
        i += 1
        t_check = time_InSAR[i]
    t_i_InSAR = i
except:
    t_check = -1
    t_InSAR_step = t_check

i = 0
    
t_check = time[0]
try:
    while t_check <= (t-day):
        i += 1
        t_check = time[i]
    t_i_GPS = i
except:
    t_check = -1
    t_GPS_step = t_check

if abs(t- t_InSAR_step) > (day/2): #if this timestep was more than a day ago/ahead
    t_InSAR_step = -1
    td_InSAR_step = []
    look_InSAR_step = []
    xy_InSAR_step = []
    areas_step = []
else:
areas_step = areas

td_InSAR_step = td_InSAR[t_i_InSAR]

look_InSAR_step = look_InSAR[t_i_InSAR]

xy_InSAR_step = xy_InSAR[t_i_InSAR]

if abs(t - t_GPS_step) > (day/2): # if this timestep was
    # more than a day ago/ahead or there's no GPS step
    t_GPS_step = -1
    u_GPS_step = []
    xy_step = []
    names_step = []
else:
    u_GPS_step = u_data[t_i_GPS]
    xy_step = xy[t_i_GPS]
    names_step = names[t_i_GPS]

print("\tInSAR Data points: " + str(len(areas_step))")
print("\tGPS Data points: " + str(len(u_GPS_step))")

"""Genetic algorithm for each step begins"""
for g in range(gens):

    """Selection"""
    # find fitnesses for each member of the population
#fitness(P, names, xy_GPS, data_GPS, xy_InSAR, 
data_InSAR, look_angle, time, method, errors):

fitnesses = fitness.fitness(P, names_step, xy_step, 
u_GPS_step, xy_InSAR_step, td_InSAR_step, look_InSAR_step, 
areas_step, t, method, errors)

if len(fitnesses) > 0:
    # sort population by fitness
    P = [p for (f,p) in sorted(zip(fitnesses,P))]
    # kill off the losing solutions. Sorry, lil guys!
    survivor_index = round(len(P)*selection_percent)
    P = P[survivor_index-1:] # here is where we delete
    the victims

    # Find best solution to keep it in the population
    best = P[-1]
    sec_best = P[-2]

    """Bookkeeping""

    if g == gens-1: # on the last generation of each
timestep, do bookkeeping
        P_t = np.array(P).transpose().tolist()
        for i in range(num_par):
            average_values[i][t_i] =
            np.mean(P_t[i])
            max_values[i][t_i] = max(P_t[i])
            min_values[i][t_i] = min(P_t[i])
            st_dev[i][t_i] =
            np.std(P_t[i])

54
best_member[i][t_i] = P_t[i][-1]

RMSE[t_i] = -np.mean(fitnesses)

RMSE_min[t_i] = -max(fitnesses) #this is correct, max fitness is min RMSE

RMSE_max[t_i] = -min(fitnesses) #also correct

"""Save every 1000 steps""
if t_i%1000 == 0 or t_i == len(time_total)-1:
    save_data('results.pickle', time_total, names, xy, min_values, best_member, max_values, RMSE_min, RMSE, RMSE_max, u_data, n_data, e_data)
    print("SAVING")
"""Recombination""
P = recombine.recombine(P, population)

effective_mut = mutation_amount * (gens-g)**2/gens**2

"""Mutation""
P = mutate.mutate(P, parameters_ranges, mutation_prob, effective_mut) #mutation amount falls off every generation
    P.append(best) #the best member is cloned into the new population
    P.append(sec_best)

save_data('results.pickle', time_total, names, xy, min_values, best_member, max_values, RMSE_min, RMSE, RMSE_max, u_data, n_data, e_data)
print("SAVING")
print("EDA finished! Use plot_data.py to plot results")

os.system('say "your program has finished"')

"""Fitness.py"""

from mogiModel import mogiModel
from VE_model import VEModel
from math import sqrt
import numpy as np

def fitness(P, names, xy_GPS, data_GPS, xy_InSAR, data_InSAR, look_angle, areas, time, method, errors):
    ""
    Fitness function using a mogi model
    Therese Monical
    ""
    model_data_u_GPS = []
    model_data_u_InSAR = []
    model_data_n = []
    model_data_e = []
    # generate models based on parameters in each member of the population
    for member in P:
        if method == 0:  # if method is a mogi model
datapoints = mogiModel(xy_GPS, member[0], member[1], member[2], member[3], member[4], time)
model_data_u_GPS.append(datapoints[0])  # append "up" data to u_data

datapoints = mogiModel(xy_InSAR, member[0], member[1], member[2], member[3], member[4], time)
model_data_u_InSAR.append(datapoints[0])  # append "up" data to u_data
model_data_n.append(datapoints[1])  # append n data to n
model_data_e.append(datapoints[2])
elif method == 1:  # if Viscoelastic model
    datapoints = VEModel(xy_GPS, member[0], member[1], member[2], member[3], member[4], time)
    model_data_u_GPS.append(datapoints[0])  # add "up" data from time "time" to u_data

    datapoints = VEModel(xy_InSAR, member[0], member[1], member[2], member[3], member[4], time)
    model_data_u_InSAR.append(datapoints[0])  # add "up" data from time "time" to u_data
    model_data_n.append(datapoints[1])  # add radial data to e_data and n_data
    model_data_e.append(datapoints[2])

    # compare each model's output to the output in the real data
    model_fitnesses = []
    for _ in range(len(P)):
        model_fitnesses.append(0)
for j, member in enumerate(P):
    difference = 0.0
    #distance_summed = 0.0

    # GPS DATA
    for i in range(len(xy_GPS)):
        max_dif = abs(model_data_u_GPS[j][i] - data_GPS[i])
        max_dif_2 = errors[names[i]] * 1.5  # floor for error code seems pretty sensitive to this
        max_dif = max(max_dif, max_dif_2)
        difference += max_dif
        model_fitnesses[j] += -(difference)

    # InSAR DATA
    difference = 0.0
    for i in range(len(xy_InSAR)):
        model_td = ((model_data_u_InSAR[j][i]),
                    (model_data_e[j][i]),
                    (model_data_n[j][i]))
        model_td_angled = np.dot(model_td, look_angle[i])
        max_dif = abs(model_td_angled - data_InSAR[i])
        max_dif_2 = .002  # floor for error, could be more like .01 but we'll try this for now
        max_dif = max(max_dif, max_dif_2)
        difference += max_dif*areas[i]
        if len(xy_InSAR) > 0:
            model_fitnesses[j] += -(difference)/sum(areas))
length_total = len(xy_InSAR) + len(xy_GPS)

for fitness in model_fitnesses:
    if length_total != 0:
        fitness = fitness/length_total
    else:
        fitness = -99999

# print(model_fitnesses)
return model_fitnesses

def mogiModel(xy, a, delP, d, chamber_x, chamber_y, t):
    
    """This function uses the Mogi Model formula to calculate Uz (uplift in vertical direction) due to an inflating magma chamber

    Written 2016-02-12 by Therese Monical, modified as needed
    """

    E   = 75.0 * (10**9) # Pa
    eta = .25 # poisson's ratio
    G   = E/(2*(1+eta)) # shear modulus
K = E/(3*(1-2*eta)) # bulk modulus
l = (E*eta)/((1+eta)*(1-2*eta)) # Lame's constant
#nu = 2* (10**16) #viscosity
u0 = 0.5
u1 = 0.5
firstterm = 3/(4* (K - (2*G/3))) #first term of equations

x = []
for point in xy:
    x.append(sqrt((point[0]-chamber_x)**2 + (point[1]-chamber_y)**2))

Ur = []
Uz = []
Ue = []
Un = []

#calculate Mogi model without first term
for i in range(len(x)):
    Ur.append(a**3 * delP * x[i] / (x[i]**2 + d**2)**(3.0/2.0))
    Uz.append(( a**3 * delP * d) / ((x[i]**2 + d**2)**(3/2)))

for i in range(len(x)):
Uz[i] *= firstterm

magnitude = ((chamber_x - xy[i][0])**2 + (chamber_y-xy[i][1])**2)**0.5

direction = ((chamber_x - xy[i][0])/magnitude, (chamber_y-xy[i][1])/magnitude)

Un.append(Ur[i]*firstterm * direction[1])
Ue.append(Ur[i]*firstterm * direction[0])

return [Uz, Un, Ue]

"""
load_data.py
Therese Monical
Loads data from pickle file created in use_matlab_variables.py
"""

import pickle
import utm

def load_data(filename):
    with open(filename, 'rb') as f:
        data = pickle.load(f)
def get_data(filename, x0, x1, y0, y1):
    data = load_data(filename)

    zone = 11
    max_xy = 0
    xy = data['xy']

    x_stata = data['x_stata']
    y_stata = data['y_stata']
    new_x_stata = []
    new_y_stata = []
    for i in range(len(x_stata)):
        if x_stata[i] > x0 and y_stata[i] > y0 and x_stata[i] < x1 and y_stata[i] < y1:
            new_x_stata.append(x_stata[i])
            new_y_stata.append(y_stata[i])

    x_stata = new_x_stata
    y_stata = new_y_stata

    names = data['names']
    names_total = data['names_total']
    #errors = data['u_err'];
    u = data['u']
    e = data['e']
    n = data['n']
new_xy = []
new_names = []
new_errors = []
new_u = []
new_n = []
new_e = []
time = []
for i in range(len(xy)):
    new_time_xy = []
    new_time_names = []
    #new_time_errors= []
    new_time_u = []
    new_time_n = []
    new_time_e = []
    for j, point in enumerate(xy[i]):
        if point[0] > x0 and point[1] > y0 and point[0] < x1 and point[1] < y1:
            utm_coor = utm.from_latlon(point[1], point[0])
            new_point = (utm_coor[0], utm_coor[1])
            new_time_xy.append(new_point)
            new_time_names.append(names[i][j])
            new_time_u.append(u[i][j])
            new_time_n.append(n[i][j])
            new_time_e.append(e[i][j])
            #new_time_errors.append(errors[i][j])
        if utm_coor[2] != zone:
            zone = utm_coor[2]
print("ZONE NUMBER CHANGED TO: " + str(zone) + "CALCULATIONS ARE NOW WRONG")

if len(new_time_xy) > 0:
    if len(new_time_xy) > max_xy:
        max_xy = len(new_time_xy)
    new_xy.append(new_time_xy)
    new_names.append(new_time_names)
    new_u.append(new_time_u)
    new_n.append(new_time_n)
    new_e.append(new_time_e)
    time.append(data['t'][i])

    #data['t'] = time
    time = get_t(time)

    print("Number of stations: " + str(max_xy))

    return {'xy': new_xy, 'u': new_u, 'n': new_n, 'e': new_e,
            'names': new_names, 'names_total':names_total, 'x_sta':x_sta,
            'y_sta':y_sta, 'time':time}

def get_t(old_time):
    new_time = []
    year = 365.25 * 24 * 60 * 60
    for t in old_time:
        new_time.append(t*year)
    return new_time
get_gps_locs.py

Loads GPS locations from gps_loc.csv into stations.pickle
This makes it usable in other code very quickly
Tess Monical

08-17-2016  Code written (date approximate)

import pickle

def get_gps_locs():
    name = []
    x    = {}
    y    = {}

    with open('gps_loc.csv') as f:
        for line in f:
            thing = line.split(','
            name.append(thing[0].strip('" '))
            x[thing[0].strip('" ')] = (float(thing[1]))
            y[thing[0].strip('" ')] = (float(thing[2]))

    stations = {}
    stations['names'] = name
stations['x'] = x
stations['y'] = y

"""TIME TO PICKLE IT- save to python-specific data file""
with open('stations.pickle', 'wb') as f:
    pickle.dump(stations, f)
print("Station data saved!!")

"""recombine.py
By Therese Monical
A method of recombining genes in a population of solutions
"""
import random

def recombine(P, population):
    newP = []
    for i in range(population):
        parents = [random.choice(P), random.choice(P)]
        baby = []
        for gene_i in range(len(parents[0])):
            baby.append(random.choice(parents)[gene_i])
        newP.append(baby)
    return newP

"""
import random

def mutate(P, limit, mutation_chance, mutation_amount):
    for member in P:
        for i in range(len(member)):
            if random.random() < mutation_chance:
                #mutation = random.uniform(-mutation_amount/2, mutation_amount/2)
                member[i] = random.gauss(member[i],
                mutation_amount/2*(limit[i][1]-limit[i][0]))
                #if member is beyond what is allowed, move them back
                if member[i] < limit[i][0]:
                    member[i] = (limit[i][1]-limit[i][0])*random.uniform(0.00, 0.02) + limit[i][0]
                elif member[i] > limit[i][1]:
                    member[i] = (limit[i][1]-limit[i][0])*random.uniform(0.98, 1.00) + limit[i][0]
    return P

"""
Therese Monical

Loads data from pickle file created in use_matlab_variables.py

05-17-2016  started coding this
coded in unit changes andlatlong to utm
07-12-2016  loads error data now as well
07-26-2016  fixed major bug where data loading loaded ALL data
instead of just data in the correct region
        function names changed, get_u() folded into
        get_data()
09-01-2016  now we load e and n as well as u

"""

import pickle
import utm

def load_data(filename):
    with open(filename, 'rb') as f:
        data = pickle.load(f)
    return data

def get_data(filename, x0, x1, y0, y1):
    data = load_data(filename)

    zone = 11

68
max_xy = 0
xy = data['xy']

x_sta = data['x_sta']
y_sta = data['y_sta']
new_x_sta = []
new_y_sta = []
for i in range(len(x_sta)):
    if x_sta[i] > x0 and y_sta[i] > y0 and x_sta[i] < x1 and y_sta[i] < y1:
        new_x_sta.append(x_sta[i])
        new_y_sta.append(y_sta[i])

x_sta = new_x_sta
y_sta = new_y_sta

names = data['names']
names_total = data['names_total']
#errors = data['u_err'];
u = data['u']
e = data['e']
n = data['n']
new_xy = []
new_names = []
new_errors = []
new_u = []
new_n = []
new_e = []
time        = []
for i in range(len(xy)):
    new_time_xy     = []
    new_time_names  = []
    #new_time_errors= []
    new_time_u      = []
    new_time_n      = []
    new_time_e      = []

    for j, point in enumerate(xy[i]):
        if point[0] > x0 and point[1] > y0 and point[0] < x1
        and point[1] < y1:
            utm_coor = utm.from_latlon(point[1], point[0])
            new_point = (utm_coor[0], utm_coor[1])
            new_time_xy.append(new_point)
            new_time_names.append(names[i][j])
            new_time_u.append(u[i][j])
            new_time_n.append(n[i][j])
            new_time_e.append(e[i][j])
            #new_time_errors.append(errors[i][j])
            if utm_coor[2] != zone:
                zone = utm_coor[2]
                print("ZONE NUMBER CHANGED TO: " + str(zone)
+ "CALCULATIONS ARE NOW WRONG")

    if len(new_time_xy) > 0:
        if len(new_time_xy) > max_xy:
            max_xy = len(new_time_xy)
new_xy.append(new_time_xy)
new_names.append(new_time_names)
new_u.append(new_time_u)
new_n.append(new_time_n)
new_e.append(new_time_e)
time.append(data['t'][i])

# data['t'] = time

time = get_t(time)

print("Number of stations: " + str(max_xy))
return {'xy': new_xy, 'u': new_u, 'n': new_n, 'e': new_e,
'names': new_names, 'names_total':names_total, 'x_sta':x_sta,
'y_sta':y_sta, 'time':time}

def get_t(old_time):
    new_time = []
    year = 365.25 * 24 * 60 * 60
    for t in old_time:
        new_time.append(t*year)
    return new_time

""

use_matlab_variables.py

Therese Monical

this file imports my data from matlab into python for ease of use.
"""
# import standard modules
import scipy.io as sio
import numpy as np
import json
import pickle

"""Import data"""
# load file
filename = "ts_web_cleaned.mat"
mat_contents = sio.loadmat(filename)
# get them into weird format
oct_epochs   = mat_contents['EPOCHS']
oct_apcoords = mat_contents['apcoords']
oct_d        = mat_contents['D']
oct_epoch_i  = mat_contents['EPOCHINDEX']
oct_sites_i  = mat_contents['SITEINDEX']
oct_sites    = mat_contents['sites']
oct_errors   = mat_contents['errors']

# move them all into reasonable format
epochs = []
for i in oct_epochs:
    epochs.append(i[0])
apcoords = []
apcoords.append(oct_apcoords[0])
apcoords.append(oct_apcoords[1])
d = []
for i in oct_d:
    d.append(i[0])
epoch_i = []
for i in oct_epoch_i:
    epoch_i.append(i[0])
sites_i = []
for i in oct_sites_i:
    sites_i.append(i[0])

errors = []
for i in oct_errors:
    errors.append(i[0])


"""MAKE DATA USABLE""
#indexes are off in matlab by 1; this should fix it
for i in range(len(epoch_i)):
    epoch_i[i] -= 1
    sites_i[i] -= 1
sites = []
for i in oct_sites:
    sites.append(i[0][0])

print(sites)

print("Data imported!")
epoch_i_unique = sorted(list(set(epoch_i))) # eliminate duplicate epochs and re-sort
print("Time indexes sorted!")

data = np.array(d) # it's easier to do this with a np array
indexes = []
times = []
locations = []

# find the data that belongs to each timestep and sort into separate sub-lists
for j, time_i in enumerate(epoch_i_unique):
    print(time_i)
    ti = np.where(epoch_i == time_i)[0]
    indexes.append(ti)
    times.append(epochs[ti])  # add the timestamps to a list while we're at it

print("Saving data just in case this next part doesn't go well...")

""" PICKLE THIS SO WE DON'T LOSE IT IF THERE ARE ISSUES"""

save_dict = {}
save_dict['indexes'] = indexes
save_dict['times'] = times

with open('save_dict.pickle', 'wb') as f:
pickle.dump(save_dict, f)

print("Data saved!")
print("Formatting Data...")

"""FORMAT DATA FOR USE IN main.py"""

xy = []
u = []
n = []
e = []
td = []
u_err = []
n_err = []
e_err = []
new_error = []
names = []
for i in indexes: #for each timestep...
    xy_sub = []
    u_sub = []
    n_sub = []
    e_sub = []

    u_err_sub = []
    n_err_sub = []
    e_err_sub = []
for j in i:  # go through each data point and retrieve the
xy/u/e/n points
    data_point = d[j]
    err_point = errors[j]
    loc_i = sites_i[j]

    if j%3 == 2:  # if j is an "up" component
        u_sub.append(data_point)
        u_err_sub.append(err_point)
        x = apcoords[0][loc_i]  # only add the xy coords
        once per data triplet
        y = apcoords[1][loc_i]
        xy_sub.append((x, y))
        names_sub.append(sites[loc_i])

    if j%3 == 1:  # north component
        n_sub.append(data_point)
        n_err_sub.append(err_point)

    if j%3 == 0:
        e_sub.append(data_point)  # east component
        e_err_sub.append(err_point)

xy.append(xy_sub)
names.append(names_sub)
u.append(u_sub)
u_err.append(u_err_sub)
e.append(e_sub)
e_err.append(e_err_sub)
n.append(n_sub)
n_err.append(n_err_sub)

new_times = []
new_xy    = []
new_u     = []
new_n     = []
new_e     = []
new_names = []

for i in range(len(times)):
    if i%2 == 0 and i < len(times) -1:
        new_times.append(times[i])

new_xy.append([])
new_names.append([])
new_u.append([])
new_n.append([])
new_e.append([])

for j in range(len(xy[i])):
    new_xy[-1].append(xy[i][j])
    new_names[-1].append(names[i][j])
    new_u[-1].append(u[i][j])
new_n[-1].append(n[i][j])
new_e[-1].append(e[i][j])

for j in range(len(xy[i+1])):
    new_xy[-1].append(xy[i+1][j])
    new_names[-1].append(names[i+1][j])
    new_u[-1].append(u[i+1][j])
    new_n[-1].append(n[i+1][j])
    new_e[-1].append(e[i+1][j])

xy = new_xy
u = new_u
n = new_n
e = new_e
times = new_times

print("Data formatted!")
print("Saving data in better format...")

"""PUT ALL DATA IN ONE DICTIONARY FOR PICKLING"""
data_dict = {}
data_dict['t'] = times
data_dict['xy'] = xy
data_dict['x_sta'] = apcoords[0]
data_dict['y_sta'] = apcoords[1]
data_dict['u'] = u
#data_dict['u_err'] = u_err
data_dict['n'] = n
data_dict['e'] = e
data_dict['names'] = new_names
data_dict['names_total'] = sites

print(data_dict['names_total'])

#data_dict['e_err'] = e_err
#data_dict['errors'] = new_error

"""TIME TO PICKLE IT- save to python-specific data file"""
with open('data_dict.pickle', 'wb') as f:
    pickle.dump(data_dict, f)
print("Data saved!!")
print("Finished!")
end{python}