The Rocky Road

Paxton Poole, Noah Fields, Sarah Mewhinney, Ryker Fish
Stratigraphic Complexity - Dr. Pengfei Hou
Team CSM Hou
June 15, 2021

Introduction

We are Team CSM Hou, also known as The Rocky Road. Our goal in Field Session was to assist Dr. Pengfei Hou, a postdoctoral researcher in the geology department at Mines, with his current research project analyzing the complexity of stratigraphic columns in submarine fans. Dr. Hou's hypothesis is that within a submarine fan, the stratigraphic complexity decreases while traveling away from the sediment source. The goal of our part of this project is to provide Dr. Hou with software that will help him achieve his research goal of quantifying complexity in 1D stratigraphic records. Specifically, our purpose was to write a program that took in stratigraphy data in the form of a time series and used several types of algorithms from complex systems science to detect and measure complexity in the data. Ultimately, this took the form of a data processing program: taking in data, processing it using various algorithms, then creating and displaying several graphs regarding various trends and outputs extracted from the data. This is controlled through a simple UI where the user chooses a file with data and views graphical and numerical outputs about the complexity found in the data.

The concept of complexity in stratigraphy has been inconsistent and qualitative. In contrast, the field of complex system science has developed ways to quantitatively measure complexity. The goal of this project was to create an easy-to-use program that applied algorithms from complex system science to datasets with stratigraphy data. These algorithms work to detect and measure the complexity present in the stratigraphic samples. These samples were either gathered in the field or simulated using models. Having quantifiable results will allow geologists to compare and contrast between stratigraphical samples complexity more rigorously than when using traditional qualitative methods.

Geology Background

The study of stratigraphy focuses on sediment deposition. Over time, different materials such as silt, sand, or gravel get moved by water and wind and form different layers. This eventually solidifies into rock after millions of years, creating a sedimentary rock formation with distinct layers. A vertical slice of this rock, seeing all the layers that have built up over time due to various geologic events, is called a stratigraphic column, and stratigraphy refers to studying these columns. Studying the characteristics, thickness, grain size, and stacking patterns of these stratigraphic columns can reveal information about how the earth was millions of years ago.

While layers of sedimentary rocks can be formed by rivers or the wind through erosion and transportation of sediments, these rock layers, also known as strata, can be irregular since the turbulent nature of these mediums usually do not allow their environments to remain stagnant for long. In contrast, strata formed underwater are less prone to erosion and more continuous. A type of deposit known as a submarine fan can occur at the bottom of underwater slopes and is caused by gravity creating a current when acting on sediment suspended in a fluid. This current creates a constant flow that moves sediment downhill. When the slope levels out, any sediment being carried in the current disperses and fans out. Over time, this will create a submarine fan. Stratigraphic columns from these fans is what our client is analyzing using algorithms from complex systems science.

Requirements

The functional requirements for this project were to implement a set of algorithms that detect and measure complexity, then make the algorithms accessible to use for an audience of geologists. This breaks down into a couple parts. First, the algorithms themselves had to be implemented in two groups- the measurement algorithms inform whether complexity is present, and the detection algorithms quantify the degree of complexity in the sample. To make the outputs more useful, the numerical results needed to be visualized in graphs and plots. Finally, to make the software more accessible to users, a graphic user interface needed to be created so our client did not have to work directly with the code himself while performing his research.

While the functional requirements detailed what the client wanted the software to be able to do, non-functional requirements inform any specifics on how the client wants it to be done. For our project, it was requested that our software be created in either Matlab, Python, or both since those are languages the client is most familiar with. Although our client is familiar with some programming languages, their colleagues may not be as informed. A consequence of this is a second non-functional requirement was to make the software be usable by users without extensive programming knowledge.

System Architecture

Figure 1, below, is a simple representation of our program (it is further subdivided below in Figure 2). These components are as follows:

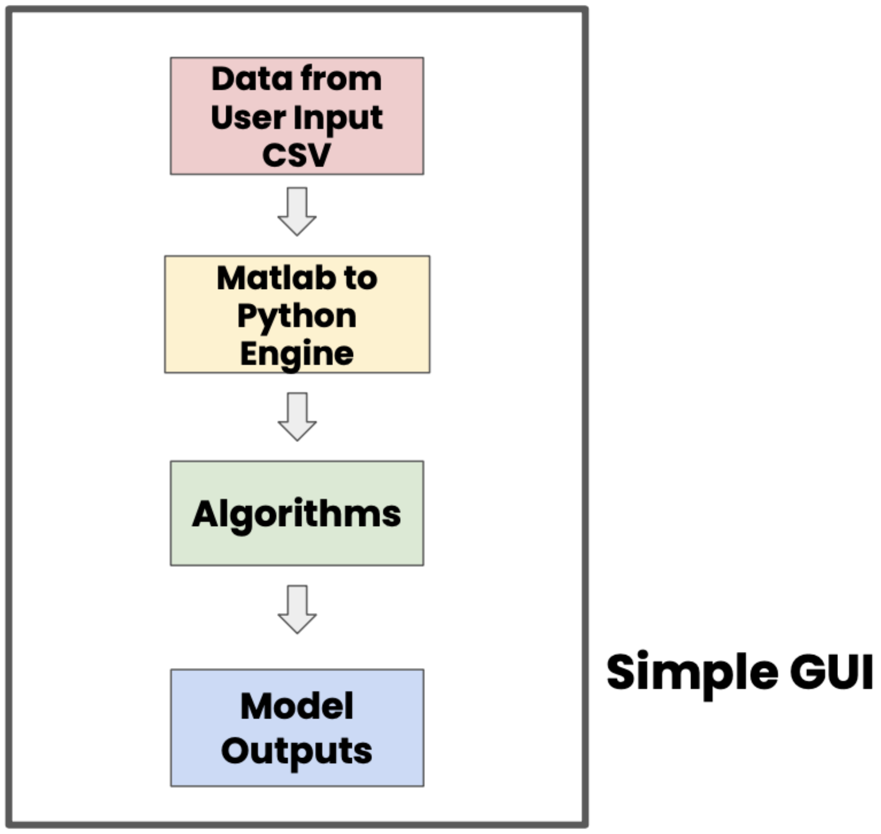


Figure 1: A simple visualization of the structure of our program and the program flow

Data from User Input CSV: The user provides data in the form of a comma-separated-values (CSV) file upon running the program. This data is either empirical (from actual stratigraphic samples) or generated via computational models.

Matlab to Python Engine: We use an API to communicate between instances of Matlab and Python, passing data back and forth to run our various algorithms.

Algorithms: These algorithms are almost entirely written in Matlab, and so are run through the Matlab/Python interface described prior. We use the Matlab/Python API to pass our data, which is originally parsed in Python, to the algorithms that are written in Matlab.

Model outputs: The algorithms provide us with a set of numerical results. In addition to returning those to the user, our team created visualizations and graphs using Python.

The rest of this section contains a more detailed breakdown of our system architecture. While the simplified architecture above is sufficient for most readers, the detailed architecture is included for completeness. A diagram of the detailed architecture can be viewed in Figure 2.

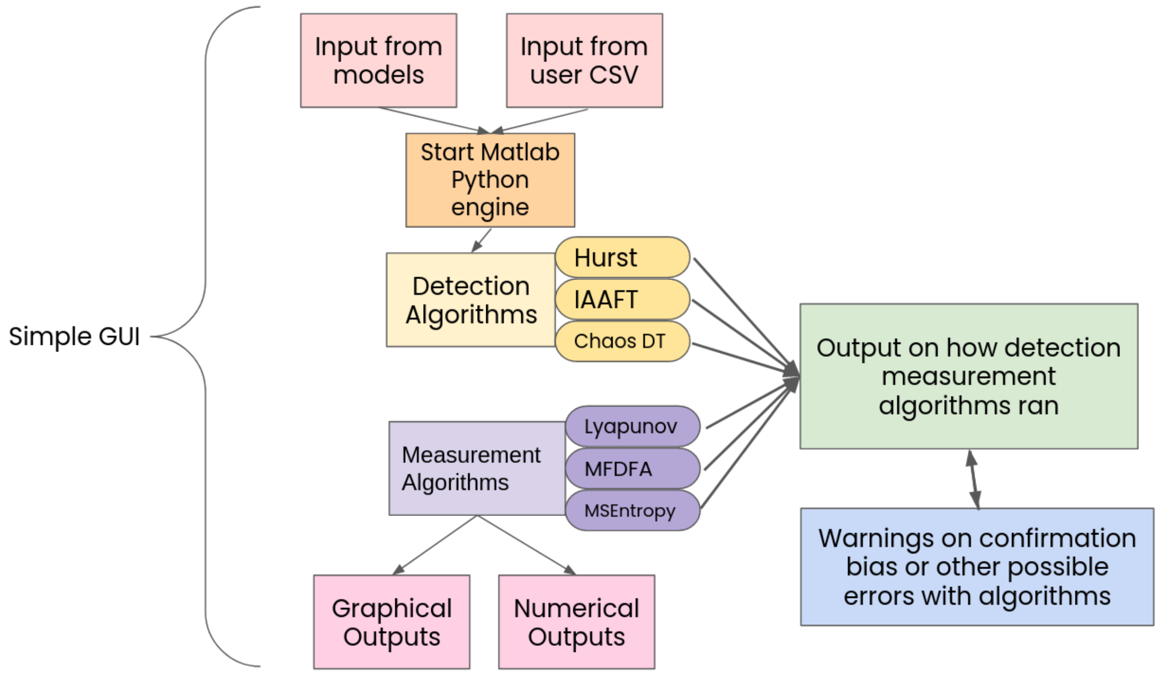


Figure 2: A detailed visualization of our program flow

Input from Models/Input from User CSV: As described previously, data may be collected empirically or from models. Either way, they are fed into the program as the beginning step. This step occurs as soon as the program is launched.

Start Matlab Python Engine: We open the Matlab/Python interface after the data is successfully loaded into the program. It is important to note that this step only happens once- the Matlab API is slow to launch when compared to the rest of our program. This single instance of the Matlab/Python engine is used for all computations.

Detection Algorithms: Our algorithms are subdivided into two types, detection and measurement. The distinction between these two types is discussed later. For now, it is only important to know that the detection algorithm results should be viewed before the measurement algorithm's are trusted.

Detection Algorithms Output and Warnings: These two components are very closely related. After the detection algorithms run, a graphical output is shown to inform the user which types of complexity were detected on each time series. It is then up to the discretion of the user whether to proceed, but a warning is given that certain results may not be reliable due to mathematical properties of certain algorithms.

Measurement Algorithms: If the user chooses to continue with the program after the previous step, the measurement algorithms are run. The measurement algorithms quantify the degree of complexity present in the various time series using different metrics. These are run after the user has a chance to view the results of the detection algorithms.

Graphical Outputs: After the measurement algorithms are run, the user may optionally save graphs of the measurement results as images. These outputs are PNGs stored in a directory within our program's folder. The graphs are created using the Python library matplotlib.

Numerical outputs: After the measurement algorithms are run, the user may optionally save data corresponding to the results. These outputs are stored in CSV files, also stored in a directory within our program's folder. These are also generated based on the measurement algorithms, using Numpy for some calculations.

Simple GUI: All of the above is controlled via a simple user interface created with PySimpleGUI. This lets the user load in a data file, complete computations, export some results as CSVs and export the rest of the results as images. The GUI is designed to be easy to use and understand with no prior experience, and is composed of a series of small windows for selecting a file to load, outputting detection results, and selecting options. The GUI allows for saving the results as a CSV and displays which detection tests passed for each algorithm in a simple numerical format.

Technical Design

Our project was highly unusual in that it focused heavily on research and had somewhat less of an emphasis on programming and software engineering. This research focus informed many aspects of our design by imposing domain-specific constraints from geology and complex systems science onto the required functionality of our software. We will begin with a discussion on three different types of complexity that our algorithms focus on, then move into discussing how these algorithms are further divided into two subcategories of measurement and detection.

Our algorithms can be broadly split into three categories, distinct from the detection/measurement split as discussed before. The three categories can be seen on the left side of Figure 3. There is no singular formal definition for “complexity” in complex systems science. As such, we have algorithms that test for complexity as defined by three different definitions that each look at different metrics to determine complexity. In particular, chaotic complexity refers to a small change in the input resulting in an exponentially large change in the output, such that any change greater than 0 will eventually result in completely different systems at some point in the future, similar to the butterfly effect. Multifractal complexity refers to nested fractal structure, where the output is composed of two or more different fractal patterns overlaid on each other with different frequencies. Entropic complexity refers to how quickly the current state of the system will stop being useful when attempting to predict future states for the system. Higher entropy implies a more difficult to predict system.

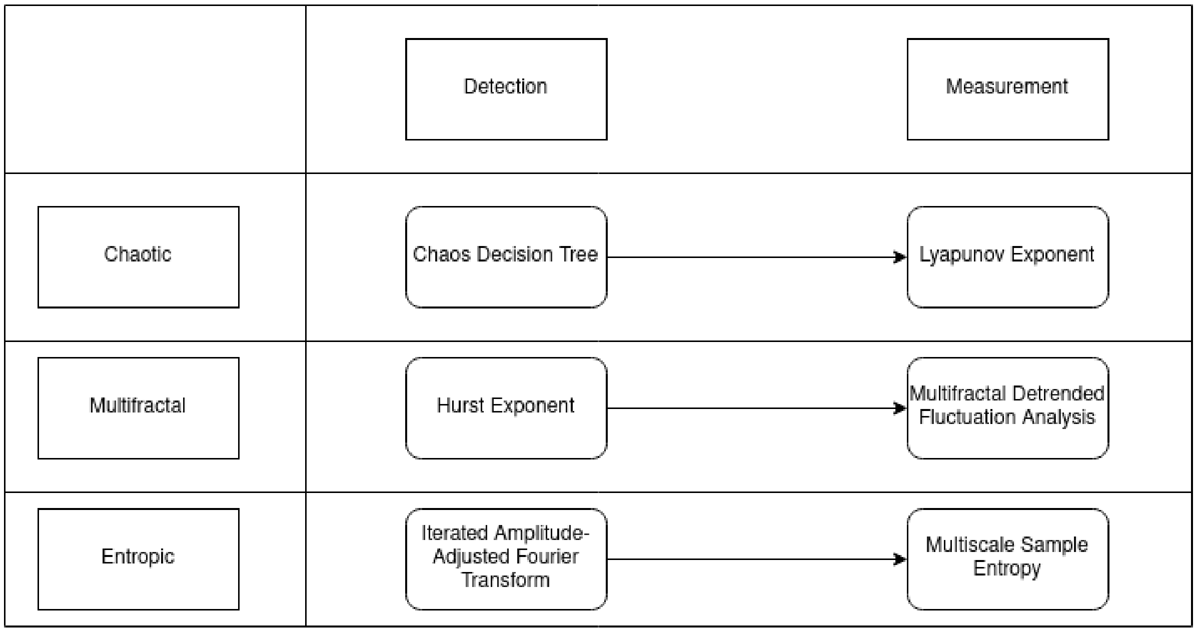
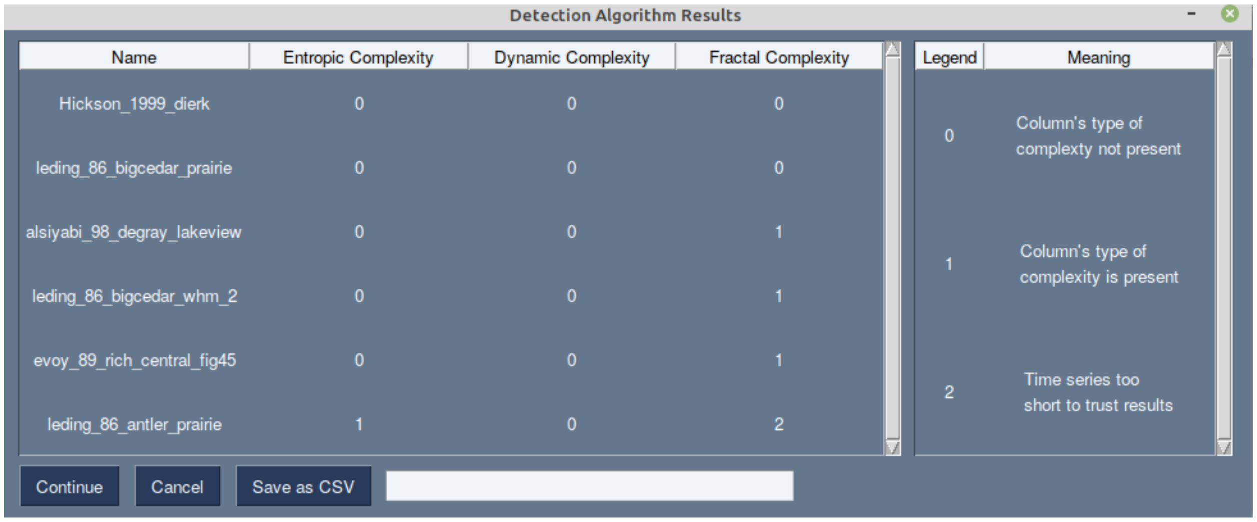

Detection
Measurement
Chaotic
Chaos Decision Tree
Lyapunov Exponent
Multifractal
Hurst Exponent
Multifractal Detrended Fluctuation Analysis
Entropic
Iterated Amplitude-Adjusted Fourier Transform
Multiscale Sample Entropy

Figure 3: The relationship between the detection and measurement algorithms and different types of complexity

As seen in the top row of Figure 3, we have two algorithms for each category of complexity, where one is used for detection and the other for measurement. With the detection algorithm, we may see if a particular type of complexity is present in the data. This calculation is followed by a measurement algorithm that calculates how much complexity is present. It is important to determine if complexity is present before blindly measuring it, otherwise, the results from the measurement algorithms may be misinterpreted.

To make our software more usable for non-programmers, we created a graphic user interface (GUI) that allows our client to run algorithms on their datasets without having to interact with the code directly. This was important to our client as they want their geologist colleagues to be able to use our software for their own research in the future. Figure 4 shows the screen that displays the results of the detection algorithms and is designed so that the user can save the results for reference later. This was important to the client as the detection algorithm results inform how accurate the measurement algorithms will be.



Name	Entropic Complexity	Dynamic Complexity	Fractal Complexity
Hickson_1999_dierk	0	0	0
leding_86_bigcedar_prairie	0	0	0
alsiyabi_98_degray_lakeview	0	0	1
leding_86_bigcedar_whm_2	0	0	1
evoy_89_rich_central_fig45	0	0	1
leding_86_antler_prairie	1	0	2

Legend	Meaning
0	Column's type of complexty not present
1	Column's type of complexity is present
2	Time series too short to trust results

Figure 4: Detection algorithm results screen of the GUI

In order to help the end users be able to better interpret the results we obtained from running the various complexity algorithms, we created a variety of visualizations. These visualizations were created using Python's matplotlib library. The different measurement algorithms generate outputs in a variety of different ways. As an example the largest lyapunov algorithm outputs an individual value for each time series that corresponds directly to the measure of chaotic complexity present in each time series. This output was visualized using a bar graph which can be seen in Figure 5.

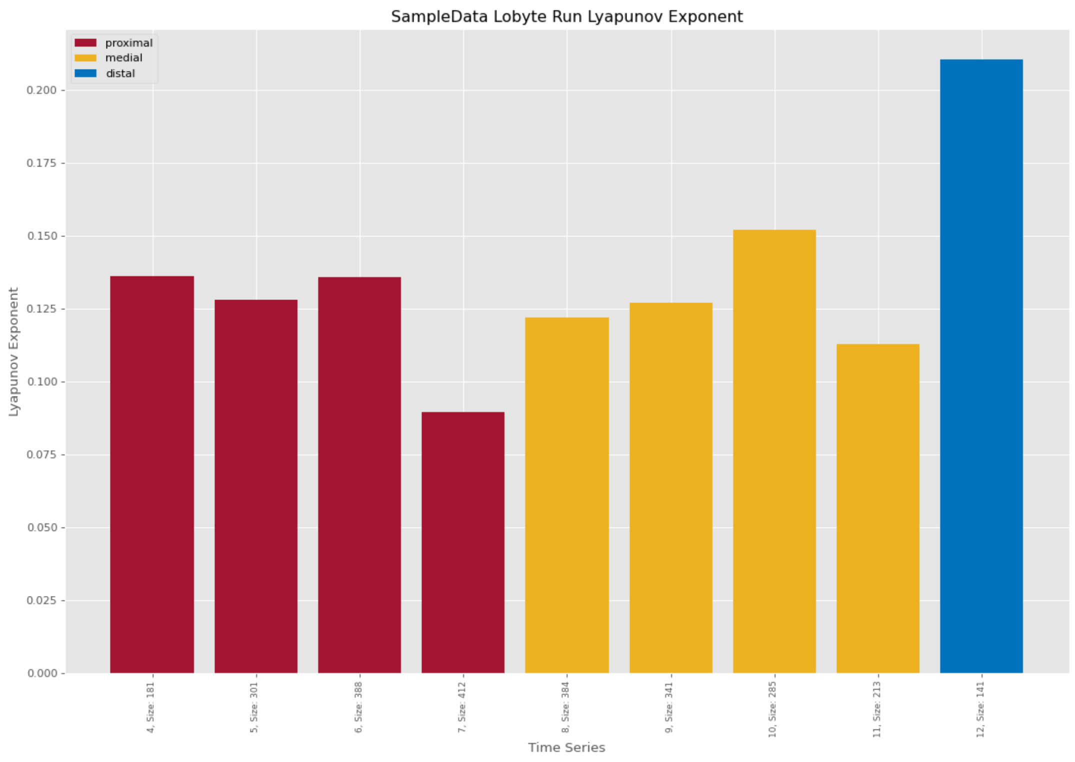


Figure 5: A bar graph showing the Lyapunov Exponents of the Lobyte3D data set

To create visualizations that were better suited for the client to be able to test their hypothesis we specified that there must be a specific ID column present in the input file. This column helped to denote the different time series groups when graphing. In the case of our client this meant that the different time series were labeled to be proximal, medial, or distal. In some cases we even combined the results from all of the time series within one group and displayed those grouped results. The graph below shows the grouped averaged lines from the application of the Multiscale Sample Entropy algorithm. The interpretation of this graph is largely up to the user. Lines that are above other lines are generally considered to represent time series with a higher order of entropic complexity.

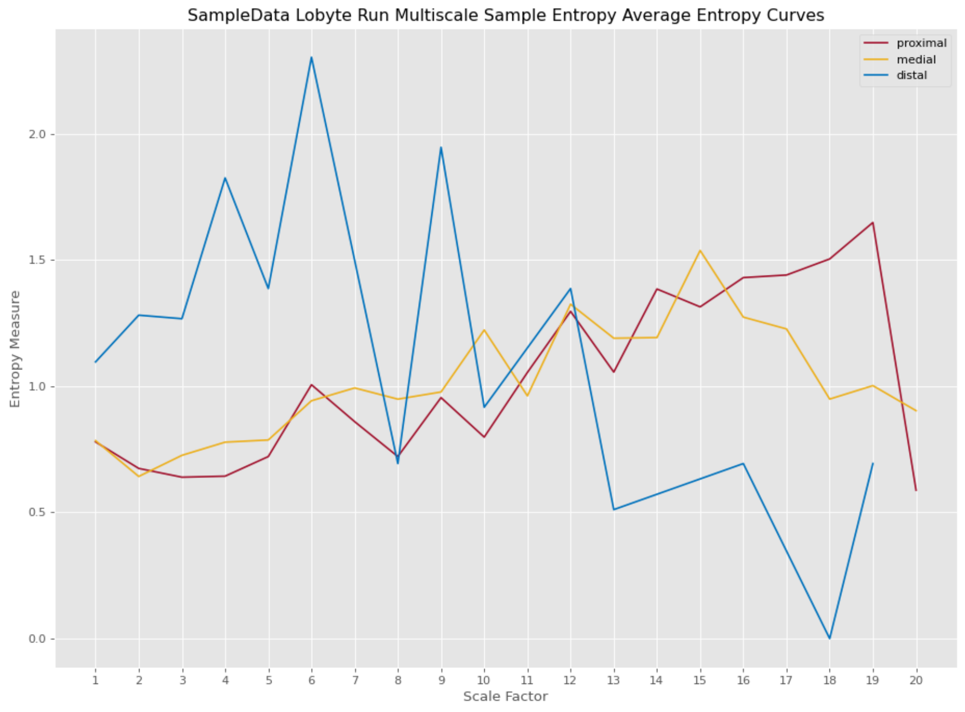


Figure 6: A line graph showing the average multiscale sample entropy of the Lobyte data set

Quality Assurance

1. Python linter:

   Python has a long-standing tradition of maintaining a style guide known as PEP 8. Applying a style guide to our project provided consistency and normalized any variations in code styling between different team members on the project. To ensure consistency, the linter Pylint was used to analyze Python code in our project. A consistent style has made our code more maintainable for any future developers who may work on this project.

2. Branching code reviews:

   Code reviews are a standard practice in industry that help prevent obvious mistakes by having the programmer talk over their code with another party knowledgeable about the project. This knowledgeable third party may be able to spot simple mistakes that go against the project requirements that the programmer may have lost sight of during implementation, helping the project reach its goals sooner. We performed code reviews in which at least one other member of our team looked over our code after adding any new features. We also always made sure to review code prior to performing merges. This helped to ensure that every member of the team had a good idea of how our code functioned as a unit and minimized misunderstandings.

3. Manual UI testing:

   UI testing ensures that every object the user can interact with on the interface performs as intended. For example, no button or menu should crash the program or be non-responsive. While we lacked the time necessary to programmatically test the UI, we performed manual UI testing by going through the intended program flow and ensuring that everything was working as expected. This type of testing only checks to make sure all functionality is in place, not with whether the client finds the UI intuitive and easy-to-use.

4. Matlab/Python integration testing:

   We use a Python module that interfaces directly with Matlab code. While we can trust the correctness of the results returned from the Matlab code since the module was created by Mathworks, we cannot blindly trust we are giving the Matlab code the correct inputs. Because of the differences in data types between Matlab and Python, incorrect inputs can throw uncatchable errors in the Python wrapper. Interface testing was done to ensure that when our program is given a valid dataset as input, nothing the user can do will cause the Matlab interface to throw an error.

5. User acceptance testing on the UI:

   Aside from wanting the UI to work flawlessly, it is also desirable for the UI to be as intuitive as possible for the user. In order to accomplish this, we garnered feedback directly from the client on how they would like the UI to function. Doing this allowed us to make the necessary modifications required in order to give the client the product they desired.

The lack of unit tests in the quality assurance portion of this project is not an oversight. Instead, it was deemed unnecessary as our team did not implement the computational algorithms ourselves. Instead, the implementations are sourced from previous research groups who have released their code for use in other research projects such as this one. Since the algorithms were tested by the original creators, our team opted to not implement unit tests for that part of our project. Instead, we focused on integration testing of the Matlab/Python interface to ensure we were properly passing data back and forth from our source code in Python to the algorithms from various research groups in Matlab.

Results

The goal of this project was to create a program that contained six algorithms, three of which would detect complexity and another three that would measure the amount of complexity present in a stratigraphic column. Our program met all of the requirements laid out by our client, but we did not accomplish adding the models as part of our data input, rather our client provided those to us in a csv. Our initial goal was to create a single numeric value that defined the complexity measurement in a sample based on all algorithms, but we found this to be unrealistic. Instead, we created a radar chart which shows this graphically. We were unable to implement our stretch goals of a website front end that provides access to the complexity algorithms and database backend due to time constraints. Additionally, we did not have time to migrate our code from being a set of python scripts to being an executable, which would be more accessible to geologists. The website and database could be something worked on by a future group, field session or not. As our project is an aid to a research project, it is nearly impossible to test for “correctness” as there are currently no correct datasets or expected results for a project like this. We also, because of time constraints, did not write tests for our GUI. Rather we performed manual UI testing.

Lessons Learned:

- Matlab and Python have an easily accessible interface that was very useful in combining algorithms found in Matlab into Python as Python has an easier to understand framework for a graphical user interface.
- There is a large benefit to understanding the things behind what we are writing code for. Complex systems science is new to all of us, and so working with these algorithms presented some great challenges as we weren’t always sure if they were working properly. A lot of in-depth reading of research papers was required before we even began to write any code, and this proved to be very useful once we began developing in earnest. Being able to understand results and comprehend if they are logical is something we had, beforehand, taken for granted.
- The field of complex systems science is still relatively new and has many applications outside of just geology. There exist time series in the stock market, weather, and internet traffic on which these algorithms may be usefully applied.
- It is important to have an open line of communication with the client or user to make sure code functions in a way that is suitable to them and accomplishes what they would like.

- We also realized the importance of the parts of the Scrum process, especially the parts that kept the team focused on the next tasks at hand in order to always have us on a productive path.

Future Work

While we did complete a great deal of our functional and non-functional requirements, there were a few stretch goals and late term ideas that could be implemented by a later group. The late term idea that we ran out of time to implement would be instead of using a Matlab engine to run our Matlab code in Python, switching to Matlab and Python modules to make for more ease of use for our end users as well as making installation of our program a slight bit easier. Some code modifications could be done to allow for algorithm expansion, such as making the existing algorithms more modular. This might look like splitting our algorithms into subfiles that can be counted and detected dynamically, so that adding in a new algorithm actually would take no new code outside of the algorithm itself. Our graphical representations could also be improved by making more size and color options for the user.

A future group could also work on removing certain time series if they are deemed not complex or to have more user input on which algorithms to run based off of detection results. This is something that was considered, but ultimately dropped due to it being unnecessary with such a small number of algorithms. If there were many algorithms, the time required to run them could become increasingly substantial, and so the user may wish to remove time series that are not likely to be complex to save time and make the outputs more clear. Choosing a subset of measurement algorithms to run, if the user is only interested in some but not all of the results, is another time-saving measure that could be implemented in the future as the number of algorithms grows.

On a more abstract level, a group of more qualified geologists or complex system scientists could concatenate our results from the measurement algorithms to possibly determine a singular value for complexity for each time series to allow for easier comparison. A final suggestion for future work would be to reach our stretch goal of creating a website to run our algorithms, with a database backend so that a user could pick from existing datasets or input their own. Alternatively we could also create an executable file that users can just run.

Appendix

Readme -

Authors:
This program was created by Noah Fields, Ryker Fish, Sarah Mewhinney, and Paxton Poole.

Purpose:

This program is intended to accept stratigraphic data in the form of time series, with at minimum three columns: a Name column for identifying each individual time series, an ID column for splitting each time series into particular categories (ie upstream, downstream), and a data column containing the time series themselves.

Installation:

Download all files from the Github repo. Dependencies are as follows:

Python requirements:

- Python 3.7 or 3.8 (higher version numbers will not work as of this writing)
- matlab.engine (see Matlab's website)
- Matplotlib
- Numpy
- nolitsa (click here)
- Pandas
- PySimpleGUI
- SciPy
- Tkinter
- Seaborn

Matlab requirements:

- Matlab R2014b or above (suggested R2021a)
- Chaotic Systems Toolbox
- Curve fitting toolbox
- Econometrics toolbox
- Signal processing toolbox
- Statistics and Machine Learning toolbox
- Symbolic math toolbox

To install the code, do the following steps (written for Linux and Mac):

- Install the prerequisites listed above
- In a bash terminal, clone the repo and navigate (cd) into the root directory of the repo. This is the directory containing the src directory, setup.py file, and the setpath.m file.
- Use the following command to install the code:
- python -m pip install -e .
    - Note the period at the end of the command! Depending on your version of Python, you may need to type "python3" or "python3.x", where x is the additional versioning number, instead of just "python".
    - This command means “install the python module in this directory to my computer”
- Open matlab and navigate to where the setpath.m file is in the root directory of the cloned github repo
- Run the setpath.m file
- If a permissions error occurs here, you may have to change ownership of matlab’s files to your user account.
- Once this is complete, you should be able to run the program by calling python on the main.py file, ex. “python main.py”
    - To get any future updates, run a git pull

NOTE: If you have issues installing Matlab packages that begins with “Access Denied” or “Missing Permissions”, please see this page. That specific page is for Linux, and your steps may vary for other environments.

How to use:

Navigate to the folder you have downloaded strat-complex to. A terminal is suggested. From here, use Python to run either main.py for a GUI or commandline.py for a CLI version of the program. Images and CSVs will be output to the folder once it is completed.

Note that if you want the name and ID to be the same, you must have two separate columns for this. Simply make another column in your CSV that is a duplicate of the name column to refer to as the ID.

Recommendations:

Review the algorithm requirements file to see requirements, details, and limitations for each algorithm. In general, a large time series (1000+) is recommended.

Algorithm Information Document-

This project is composed of various applied math algorithms borrowed from the field of complex systems science. This document details certain requirements imposed on the data or limitations on the algorithm so users can better understand the cases in which each algorithm applies and can be trusted.

Multifractal Detrended Fluctuation Analysis (MFDFA):

Authors of section: Noah Fields, Paxton Poole

Requirements:

- Documentation recommends user's err on the side of caution when trying to interpret results from time series that have less than 1000 elements.[1]
- The algorithm works best when the data is "noise-like". This can generally be accomplished by detrending the series (subtracting the mean, i.e. normalizing to mean 0), and therefore is not a significant impediment in most cases. [2]
    - The currently implemented algorithm automatically detrends the data for us, so this isn't a concern unless a different implementation is used.

Meaning of output:

- The algorithm returns a multitude of Hurst exponents from the generalized Hurst exponent. If these exponents are different then the time series is shown to have a multifractal nature. This means that there are multiple fractal relations within the dataset. MFDFA accomplishes this essentially by finding the generalized Hurst exponent over varying time scales to see if it is different. If it is different, meaning that there are relations over multiple time scales, then one could say the data is more complex. In order to try and quantify this complexity we calculate the standard deviation of the different Hurst exponents as well as the maximum difference of the Hurst exponents.

Multiscale Sample Entropy (MSE):

Authors of section: Paxton Poole

Requirements:

- In the paper they use a very large dataset with around 30000 elements. The minimum sample size they used at scale 20 was 1500.[3]

Meaning of output:

- We calculate the sample entropy for different scale (τ) values ranging from 1-20. These entropy values are then plotted in a graph showing the entropy curve as a

[1] Espen A. F. Ihlen. Introduction to Multifractal Detrended Fluctuation Analysis in Matlab. Page 17.

[2] Jan W. Kantelhardt, Stephan A. Zscheigner, Eva Koscienly-Bunde, Shlomo Havlin, Armine Bunde, H. Eugene Stanley. Multifractal Detrended Fluctuation Analysis of Nonstationary Time Series. Page 3.

[3] Madalena Costa, Ary L. Goldberger, C. K. Peng. Multiscale Entropy Analysis of Complex Physiologic Time Series. Page 2.

simple line graph. Generally speaking, the entropy curves that are higher should correspond to more entropic time series.

Detrended Fluctuation Analysis (DFA):
Author of section: Paxton Poole

Requirements:

- DFA can theoretically be applied to time series with as little as 64 samples. It is important to note however that the variance greatly increases as the number of samples decreases. It is our recommendation that a time series for which DFA is to be applied has a minimum length of 100.

Meaning of output:

- DFA returns a number that is a generalization of the Hurst exponent. This output if it is arounds .5 corresponds to white noise. If it is less than .5 it is anti correlated. If it is somewhere between .5 and 1 then there is some self similarity present in the data. If it is around 1 there is pink noise present and if it is above 1 then the time series is unbounded. Since this algorithm is being used for complexity detection we want the output to be somewhere in the range of .5 - 1 for which DFA classifies the time series as correlated. This points towards there being fractal complexity in the time series and it is recommended to run MFDFA if this occurs.

Chaos Decision Tree (CDT):
Author of section: Noah Fields

Requirements:

- Various algorithms are present as subroutines of the CDT algorithm[4]. Some issues were found with particular subroutines - more investigation is needed but currently it is called with subroutines that we found worked effectively and had no issues.
- The algorithm is known to fail if the complexity is exceedingly small compared to the noise term. There is likely no solution to this and issues of this sort are likely present in other algorithms as well, if the complexity is nearly imperceptible.
- The size of the time series impacts the optimal coefficient for determining if complexity is present for the 0-1 test component of the algorithm. Larger time series are better for this purpose as for most such algorithms, though it is theoretically possible with even very small time series.

Meaning of Output:

[4] Daniel Toker, Friedrich T. Sommer, Mark D'Esposito. A Simple Method for Detecting Chaos in Nature. Page 3.

- The output of the CDT is very simple, generally just a text string regarding the detected nature of the input. The CDT first runs a surrogate test to check for stochasticity. If the data is deemed stochastic (and therefore not complex by our definition) the algorithm returns as such and terminates here. Otherwise, a downsampling algorithm is run followed by a modified 0-1 test for complexity, and - if complexity is detected - a permutation entropy algorithm not unlike the MSE algorithm above. The results from this entropy algorithm are also present in the output if this part is run, but for our purposes of detection the most important component is the output string, one of “stochastic”, “chaotic”, and “periodic”.

Largest Lyapunov Exponent:

Author of section: Ryker Fish

Requirements:

- Below inequality should be true- lower bound for size of data set[5]
- $N > 10^d$
  - N = # of data points
  - d = dimensionality of data set. Correlation dimension can be used (there exists a Matlab function to calculate this)
- To trust the calculation of dimension, another inequality should be tested.[6]
  - $d_{max} = \frac{2\,log\,N}{log(1/p)}$
  - $d <= d_{max}$
  - Above inequality should be true, otherwise calculation of d is suspect (Eckmann-Ruelle requirement)
- If both of the above inequalities are true, Lyapunov exponent calculation can be somewhat trusted. However, it is possible that $N > 30^d$ could be necessary depending on how complex the system is.[7]

Meaning of Results:

- Lyapunov exponents are a measure of a system’s predictability and sensitivity to changes in its initial conditions
- The presence of a positive exponent indicates chaos
  - A larger exponent indicates faster divergence

[5] Rosenstein et al. A Practical Method for Calculating Largest Lyapunov Exponent from Small Data Sets. Pg131
[6] J. P. Eckmann and D. Ruelle. Fundamental limitations for estimating dimensions and Lyapunov exponents in dynamical systems. Page 2.
[7] Wolf et al. Determining Lyapunov Exponents From a Time Series. Page 306.

IAAFT Test:

Author of Section: Sarah Mewhinney

Requirements:

- A fairly large dataset should be used to be able to use the statistical assumptions in the algorithm (100+ samples)

Meaning of Results:

- The result is an average mean square distance of the 39 (39 needed for 95% confidence interval for a two-tailed test) surrogate data sets and a value larger than 1.96 would be considered “complex”