UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Backend for Automated Ocean Analysis Program

Author:

Tuan Huynh

Supervisor:

Prof. Jifeng Peng, PhD

Anchorage AK, April 2016

Computer Science & Engineering Department
UNIVERSITY of ALASKA ANCHORAGE

thuynh3@alaska.edu

Version 1.0

Abstract

The target of this project is to automate the process of data collection and analysis for lagrangian coherent structures and publish it onto a website. By automating this process it saves time for researchers by having all the data ready whenever they need it. This paper will discuss the process of creating the backend for the project, the design methodology, the testing methods and a user manual. The motivation for working on this project is the many applications it can potentially have. Knowing where the ocean flows can prove beneficial in a lot of aspects from research on sea life to search and rescue missions in the ocean.

Acknowledgments

This project had been much harder than anticipated.
I would first like to thank the advisor Dr. Jifeng Peng for allowing me to join your project. Your knowledge of fluid dynamics really helped in this project where I knew nothing of the math involved.

Next, I would like to my teammate Jon Rendulic for letting me join the project as well as the help he gave me to catch up to where the project was. Your help in fixing those bugs allowed us to progress as we did.

Finally I'd like to thank all the professors I had the privilege of learning under. They were all good people that made classes fun and challenging and really pushed me to better myself.

Contents

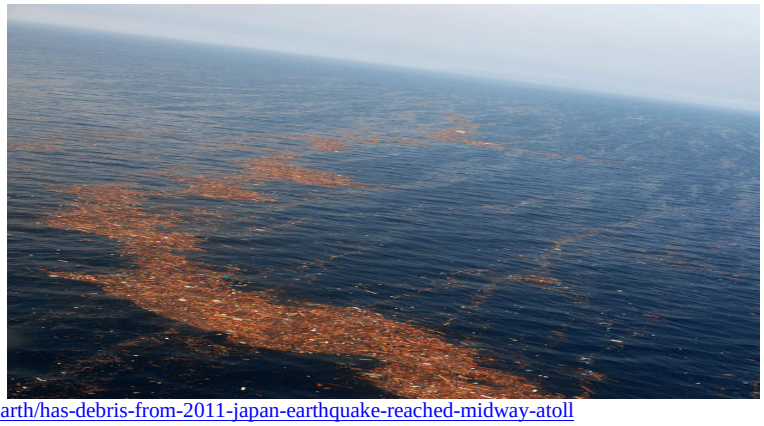
List of Figures

Chapter 1

Introduction

1.1 Introduction

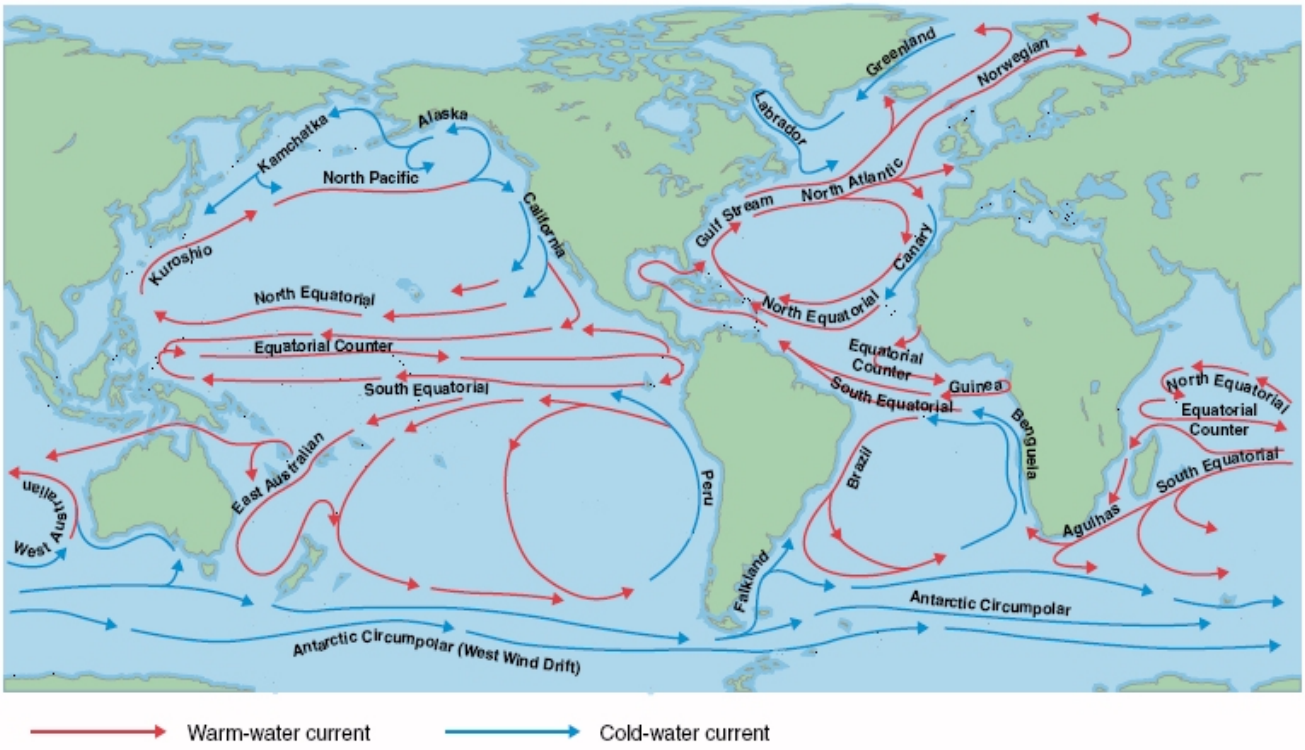
The ocean covers over 70% of Earth [1]. In 2011, the Tohoku earthquake hit Japan sending tons of debris into the ocean (Figure 1.1). As it drifted farther into the ocean, it separates and makes it harder for cleanup crews to find all of the trash floating around. What if they could get a general estimate of where the debris would float?



Taken from: http://earthsky.org/earth/has-debris-from-2011-japan-earthquake-reached-midway-atoll

Figure 1.1: Debris from Japan after the 2011 Tohoku earthquake

Consuming such a large amount of the Earth, it is of interest to know what happens to things in the ocean, natural or otherwise. This project's goal is a way to automate fluid mechanics analysis of ocean data and provide graphical representation of the data for ease of interpretation. By analyzing ocean currents (Figure1.2), it may help in tracing where debris like from the Tohoku earthquake may flow and prevention of it from dispersing into wider areas of the ocean or used to study ocean life and their migration patterns. We are considering using BSD license for the project.



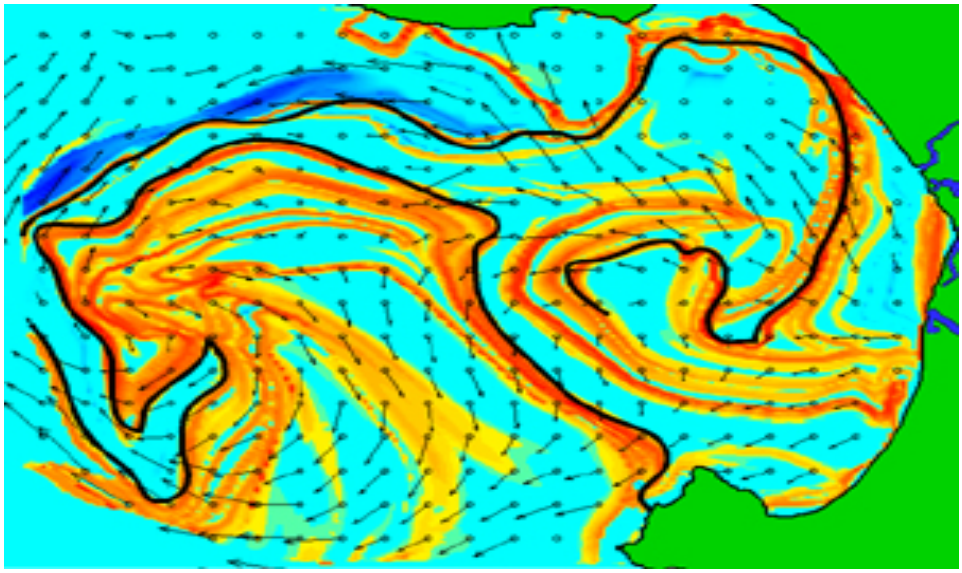
Taken from: https://cimss.ssec.wisc.edu/sage/oceanography/lesson3/concepts.html

Figure 1.2: Warm and Cold water currents

1.2 Application

The project is two parts; the first part, which is the focus of this paper, is the backend. A server will be automatically collecting data from a National Oceanic and Atmospheric Administration (NOAA) website and use math functions to get data that will show Lagrangian Coherent Structures (LCS). The data is sent to the second part of the project, an application that will plot it onto a map to see the LCS. The analysis of the data, fluid flow, is normally described using Eulerian or Lagrangian specifications. The Eulerian approach looks at specified area of a velocity field over a period of time. The Lagrangian approach follows the movement of a fluid over the passage of time [2]. The application will take the lagrangian approach and find the LCS (Figure 1.3) which are regions of a flow field that tend to have similar characteristics for the same period of time [3]. These LCS shape the fluid flow in their local

area [4] and with this information, we can estimate the fluid flow for an area. Luckily, we were provided with a program that used a graphical user interface to calculate the LCS data. We will automated it so the server can run all the data through the algorithms from the program. The application will plot the data calculated on a map. With luck, we are planning to have a website that will map the data with user specified variables. The backend of the project currently runs on a Linux server and is written in MATLAB since the LCS calculation program we were given was written in it as well. MATLAB's many tools for data manipulation for which was the program's main purpose was a large factor as well (Figure1.4). The map that the data plotting application is planning to use will be Google Maps (Figure 1.5).



Taken from: http://www.ams.org/news/math-in-the-media/mmarc-12-2009-media

Figure 1.3: A flow field with an LCS in black

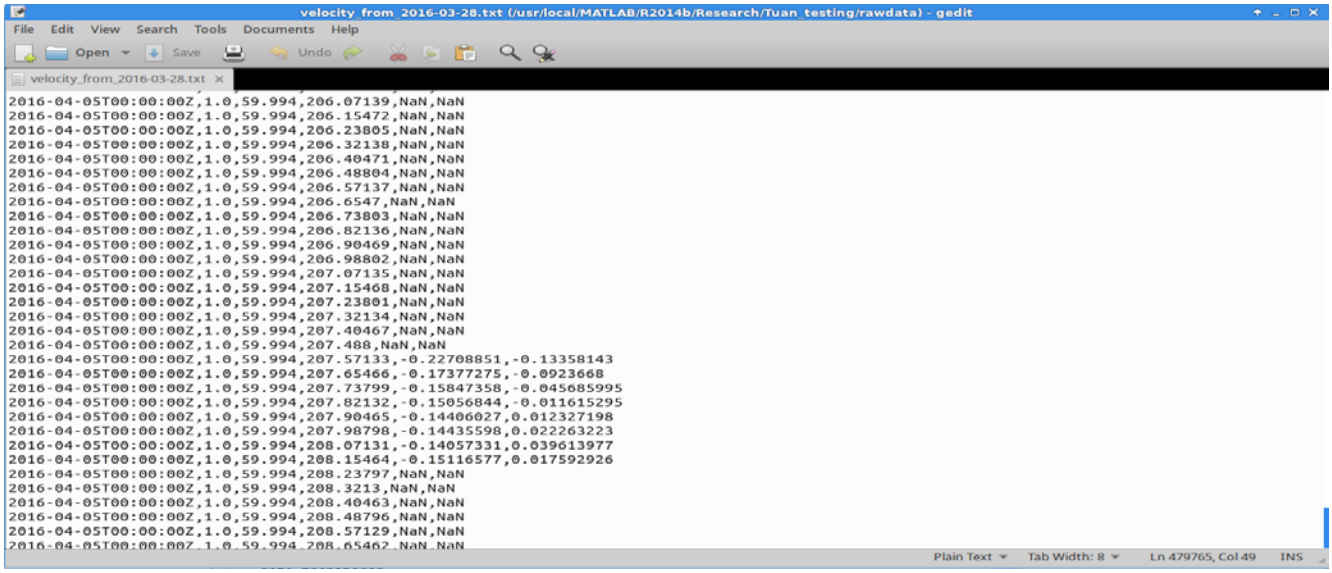


Figure 1.4: Real ocean current data before being processed.

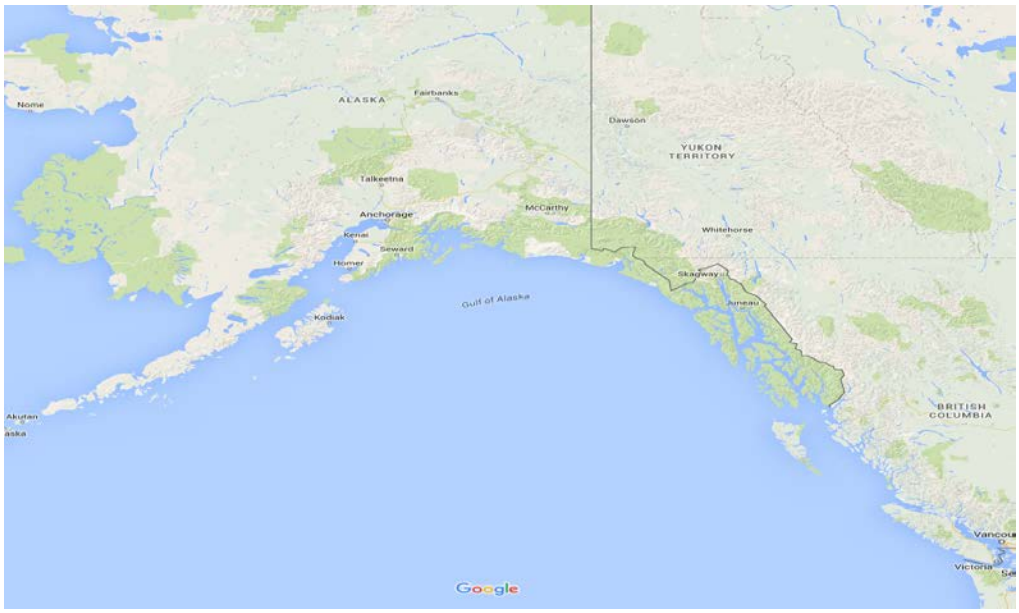


Figure 1.5: Google Maps

1.3 Motivation

The motivation for working on the project as mentioned earlier is because study of the ocean important. With this program, we can potentially track fluids and particles in ocean. Examples of this program's use include; when there is a natural disaster that sends debris into the ocean, we can now track where it will flow and help prevent its spread. Companies that do ocean drilling are likely to need to do analysis of potential dangers that can occur in case of an oil spill. A program like this can track the potential effected area if a spill does happen and be quicker in preventing it from spreading. Last year the U.S government banned the use of microbeads, which are 5mm plastic non-dissolvable beads. These were used often in facial washes products and went down our drains into the country's waterway and the ocean [5]. These tiny beads would be incredible hard for cleanup teams to find and remove from the ocean, but if a program like to this was ran on where our water leads to the ocean, we could find paths of where these things would go. A more productive effort in removing them from the ocean if that was the case instead of combing locations without knowledge of the likelihood of where they would be. This project is just in its early stages, there are much more avenues for growth. As of now, the focus of the project was on analysis ocean data but algorithm that we use is actually for fluids. This means that in future iterations it could be expanded to include analysis on wind currents. This can be used for things like tracking ash from a volcanic eruption or pollutants. Creating a functional and flexible backend is essential for use and future additions of the application.

Chapter 2

System Integration and Modeling/ Methodology

2.1 Introduction

The ocean data analysis project had already started development when I joined, the application was chosen to be built for use on a Linux Ubuntu Server and to be written in MATLAB, a proprietary programming language [6] and software created by MathWorks (Figure 2.1) The Ubuntu server was already set up by the time I had joined and the project was moving on to the programming portion. MATLAB was chosen as the primary language for the project because that was what the LCS program which we were using to do the heavy work in the backend of the program was written in and MATLAB worked well with large number computations.
The requirements for the project were:

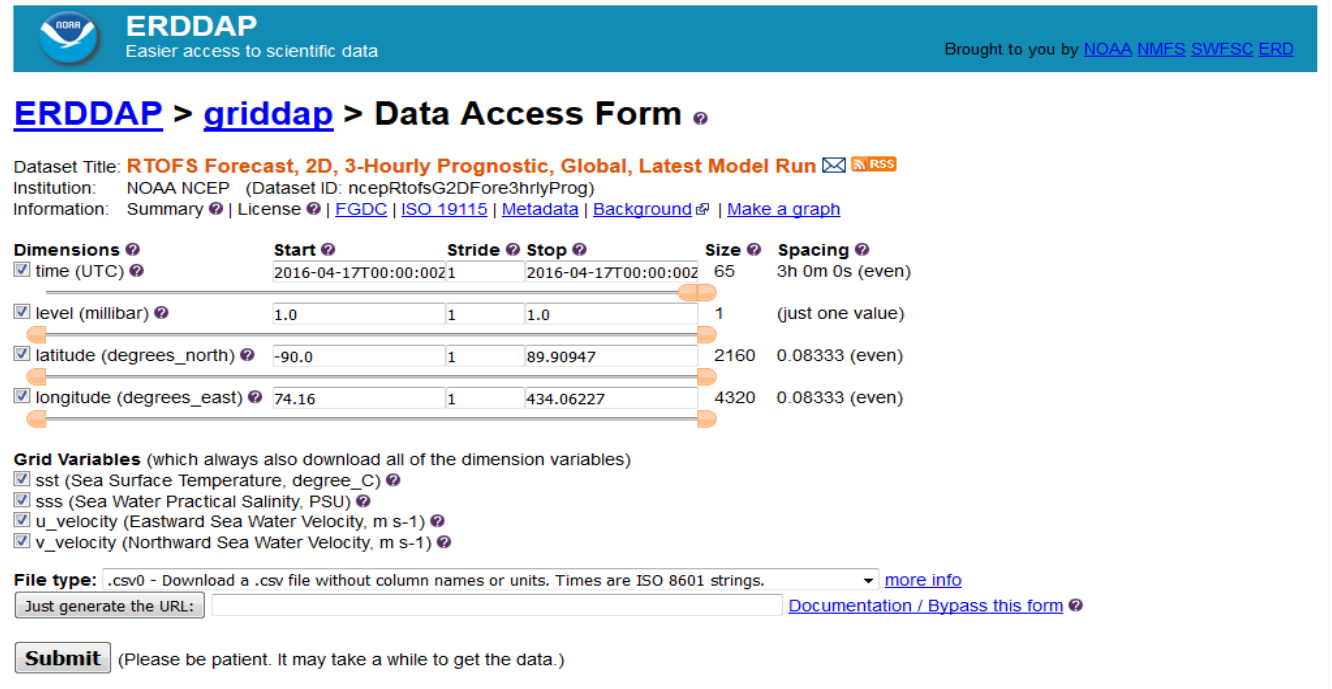
1. Automatic downloading of ocean data
2. Process the data through the LCS Calculation Program
3. Graph the data and hopefully put onto a website

As a team of two I was tasked with the backend of the system, I was responsible for tasks 1 and 2 and this chapter will discuss how these tasks were completed.

Figure 2.1: Logo for MathWorks and Linux

2.2 Application

To automate the downloading on Linux we used a program called Crontab. It is a textfile runs commands at specified times in the background [7]. Crontab will run a MATLAB script which downloads the data at a time specified by the user. The data that we needed to download was available from NOAA, http://coastwatch.pfeg.noaa.gov/erddap/griddap/ncepRtofsG2DFore3hrlyProg.html (Figure 2.2).



Taken from: http://coastwatch.pfeg.noaa.gov/erddap/griddap/ncepRtofsG2DNow3hrlyProg.html

Figure 2.2: National Oceanic and Atmospheric Administration data website

Using the NOAA site presented some unforeseen problems. NOAA used coordinates that were different from standard formats which required conversions. The application uses decimal degrees which are values from -180 to 180 for longitude and -90 to 90 for latitude and converts them into NOAA's coordinate system so that it can download from the website. The other problem was that it would sometimes fail to let you download the site if you tried to pull data from the current day. This was a problem that we were unable to fix and had to work around by having the program download from a day before the current. We could not figure out when the website updated so you could download from the current date. The MATLAB script that downloads from the website will pull in needed information from a settings file that the user will need to initialize at least once before they are able to use the program. Once the settings are in place the script will download a csv file into a folder specified by the settings and label the file "velocity_from" and then a time stamp.

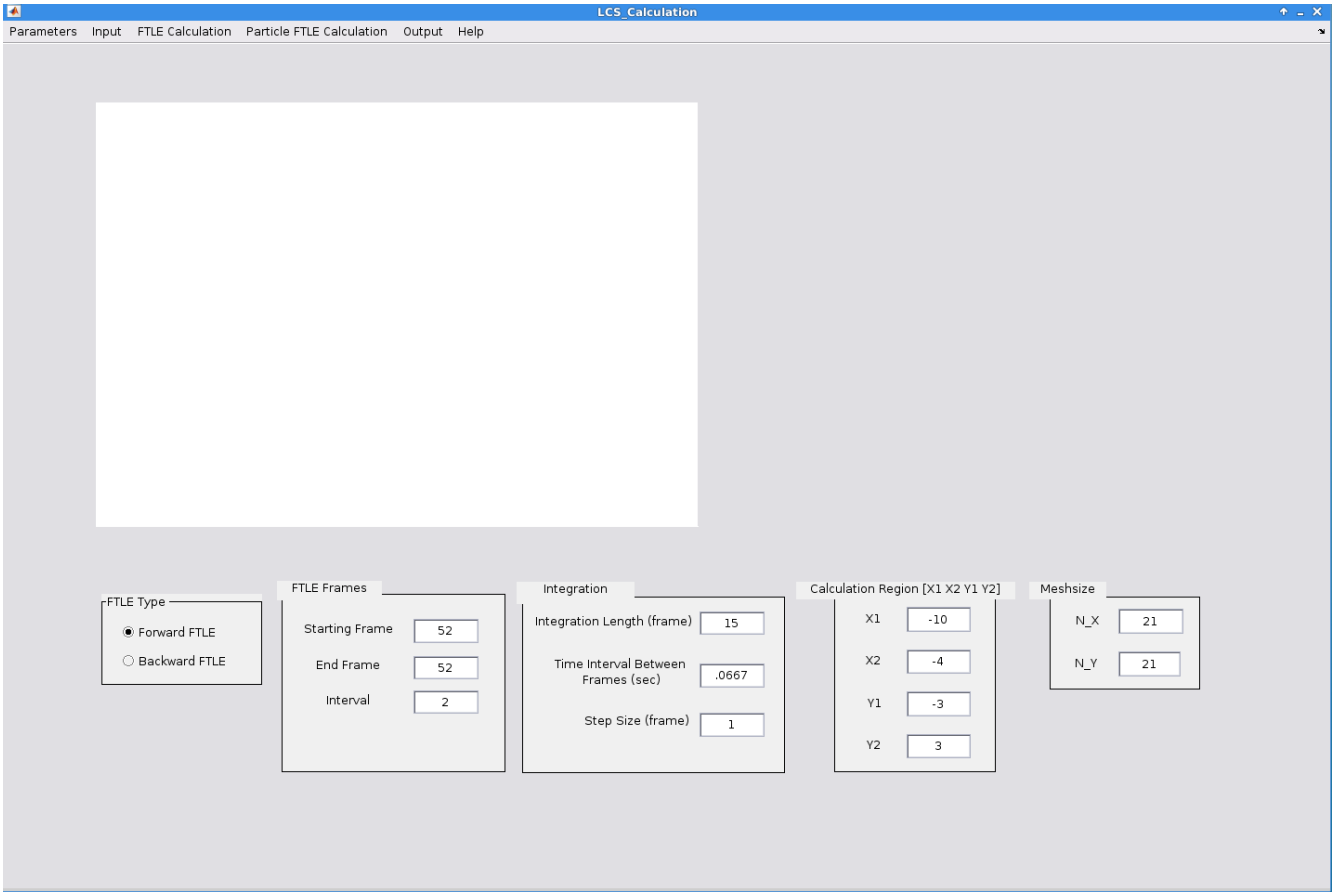


Figure 2.3: The LCS Calculation program

The data from NOAA needs to be processed through a LCS program that was given to us (Figure 2.3). This program originally functioned as a GUI only but we managed to strip it down so that we can pipe in the values it needs without the GUI. As the data runs through the LCS program it will create time stamped folders and store files in it that are used during steps of the LCS calculations. Unfortunately the program that calculates the LCS from the data uses math that is beyond the scope of my knowledge and the documentation of the program did not give clear instructions on the units for some of the input values. The files that come out of the LCS program are then handed to the plotting program for the

project which is handled by another team member.
Finally a way to change the coordinates for downloading data and values used in the LCS program will be created since making the user go into the code to change values would be unprofessional. So a settings GUI must be created for ease to the user and preventing them from changing things they might not understand.

2.3 Gantt Chart

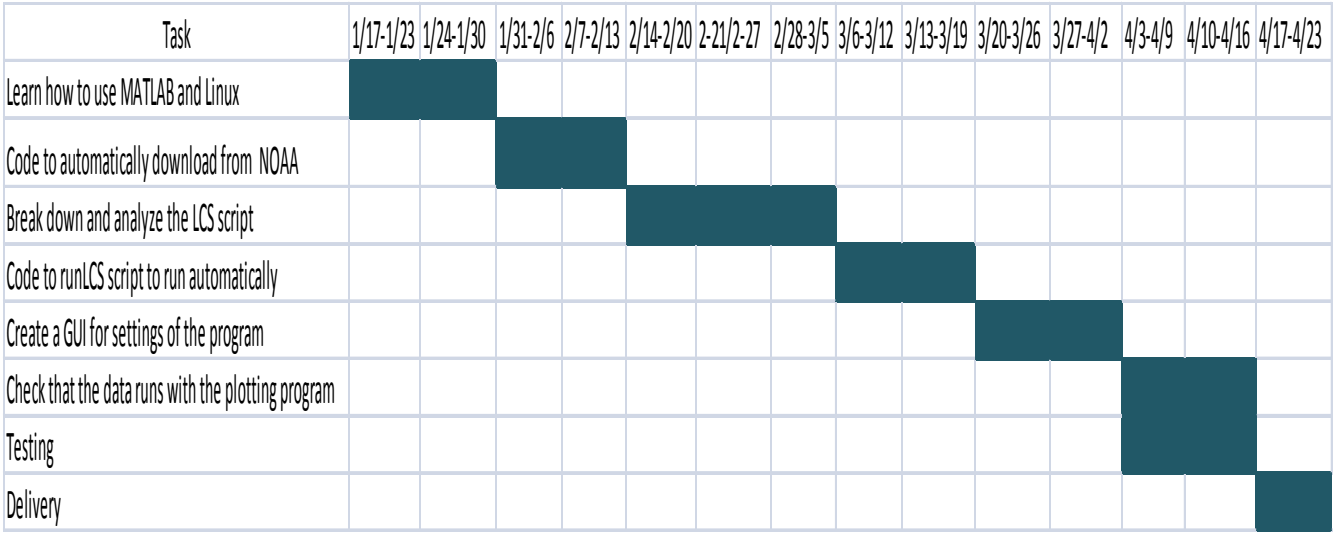


Figure 2.4: Estimated schedule

Here is the tentative schedule of how development of the backend will proceed. As this is my first time using MATLAB and Linux there will be a slow start as I learn how to use them. A critical point in the project will be the breakdown of the LCS program and testing to make the program is functional. Because the LCS program uses math that none of the team members are familiar with it will be hard to discern if we are correctly breaking down the program. The LCS program is quite a chunk of code with many variables we aren't sure if we need along with many large functions that are hard to decipher. Once we do manage decompose the program and retrieve the parts that we need the rest of the project will be easily completed.

2.4 Agile coding

The principles of agile coding will be applied in the process of writing the backend for the project. The methodology of agile coding is to write code that is easy to read and easy to understand.

- Write code to look similar to the code used throughout the rest of the program
- Try to keep the code simple
- Document your code
- Have people review your code

Following these guidelines helps make it easier for code to be used and read by other people [8]. By writing code to look like what everyone else working on the project is writing, it gives a uniformity to the overall code style of the project and an ease for fellow team members to use components not written by them. Trying to implement things as simple as possible makes it easier for someone to understand what you are trying to do. Documenting your code will give extra clarity on how the code works and how it should be used. It is great help for people who will end up using the program and future contributors [9]. Finally having team members review your code will help to point out different implementations you may not have noticed and areas where you may not have been as clear as you could have been. They will become more familiar with your code and how it is implemented so that they can better understand any issues or requirements that may come with using your code [10]. Code reviewing is important it is good to have your code easily available to team members through the use of services like Google Drive or GitHub (Figure 2.5).

Figure 2.5: GitHub Logo

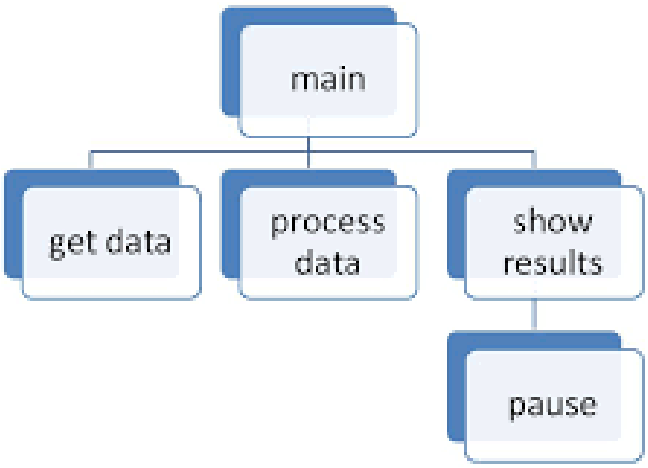
Chapter 3

Design and Testing/ User Interface

3.1 Introduction

Good design and regular testing during development lends to robust and easy to maintain software. The purpose of this program was focused on automated data processing. The choice was made to not use an object oriented approach in design because it did not fit the problem that was being solved. The project was designed as collections of functions that work independently and are connected together by a main function which calls them. The modular programming approach (Figure 3.1) to design was chosen since it emphasized parts of a machine that work independently designed to complete a specific task [11]. The LCS program that a large portion of the project was based around consisted of a series of functions which was modular so the rest of the backend was designed that way for uniformity structure. Building the rest of the backend based off of functions allowed for easier changes to parts of the program if need be. Since the backend of the program was built as functions this allowed for easier testing as parts could be tested separately to pinpoint where errors occur.

Just as important as design in testing is a clean and intuitive user interface (UI) as it helps them quickly understand and use it correctly. Users should not interact directly with the code of a program, which is why a UI is needed. The programmer only allows the user to change what is deemed safe and hides the inner workings of the software from the user.

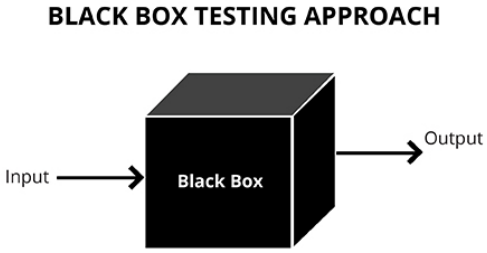


Taken from: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQt_hCgUxcZ0q5Day7otFHyYZfVrOcoX5IovHeboYNBlFviPro-

Figure 3.1: The project as a simple modular design

3.2 Testing

Tests check if the functionality of the program works correctly, one method of testing is black box testing. Black box testing checks if the code is functioning the way that is expected without looking at the code [12]. With the way the project was designed, black box testing was a useful method to use. The program was a collection of functions, what mattered most was if the input to the function produced the correct output (Figure 3.2).



Taken from: http://www.invensis.net/blog/wp-content/uploads/2015/05/Black-Box-Testing-Invensis.jpg

Figure 3.2: How Black Box Testing works

For each component of the program that was written it was tested as it was written and tested if worked properly with the rest of the program. For the downloading function, tests were made to ensure that it was downloading the correct data from the website. An error had been found that was due to the website the information was pulled from. The NOAA website would not allow downloads from the current day at certain periods of time. It was assumed the cause was the information from the website had not updated at the time of the query to download. Testing was done to try and download from the

current date, the times when it worked and the times that it didn't were not very consistent. Since the program was going to be automatic, having it fail due unpredictability of the website was unacceptable. A compromise was made to have the date which the data would be downloaded be one before the current.

For the data processing functions the tests were relatively simple. Data was sent through the functions and the outputs were analyzed to see if they matched the format that was needed for the next part of the program.

The tests for the LCS program was where problems beyond the scope of the team members appeared. As mentioned before the LCS calculation program was where most of the computation of the backend happened. It used formulas (Figure3.3) that were not familiar to either of the team members working on the project.

Calculation of Particle FTLE

The FTLE calculation described above consider dynamical systems of ideal fluid tracers, which has the same velocity as local flow velocity. A similar approach can be used to study dynamical systems of finite-size inertial particles. This code can also calculate particle FTLE in dynamical systems of small, rigid, spherical particles, whose dynamics are described by the linearized Maxey-Riley equation (Maxey and Riley, 1983):

$$\frac{d\mathbf{v}}{dt} - \frac{3R}{2}\frac{D\mathbf{u}}{Dt} = -A(\mathbf{v} - \mathbf{u})$$

with

$$R = \frac{2\rho_f}{\rho_f + 2\rho_p}, \quad A = \frac{R}{St}, \quad St = \frac{2}{9}\left(\frac{a}{L}\right)^2 Re.$$

Figure 3.3: Snippet from the LCS calculation read me

The documentation of the LCS calculation code did not provide units for variables and we weren't sure how to use some of the variables. This lead to guesses on what values the functions were expecting. For this reason testing became difficult for someone not familiar with fluid dynamics. Testing was moved to focus on automating the LCS functions to run on their own without the use of the GUI. This worked well as the program was able to replicate the files output from the GUI version of the code without using it. The functionality works, with more development we can correctly use the variables.

3.3 Graphical User Interface

A well-made user interface should be simple and clear. The user should be able to understand and locate what they need to do quickly. This was the mindset for designing the GUI for the backend of the program. The GUI (Figure3.4) gives user over the controls of the variables that can be changed in the

program. This includes values for calculating, what data to download and where to store it. On the side of most values is text that tells the user extra information about the values. Due to the fact that some of the values have bounds of allowable numbers this was needed or else incorrect information could be put. The program does not actually check if the values someone inserts are correct or not, it assumes that the user is following the instructions. There is also a default settings that is used to change the values to the ones that were used during development and tested to work.

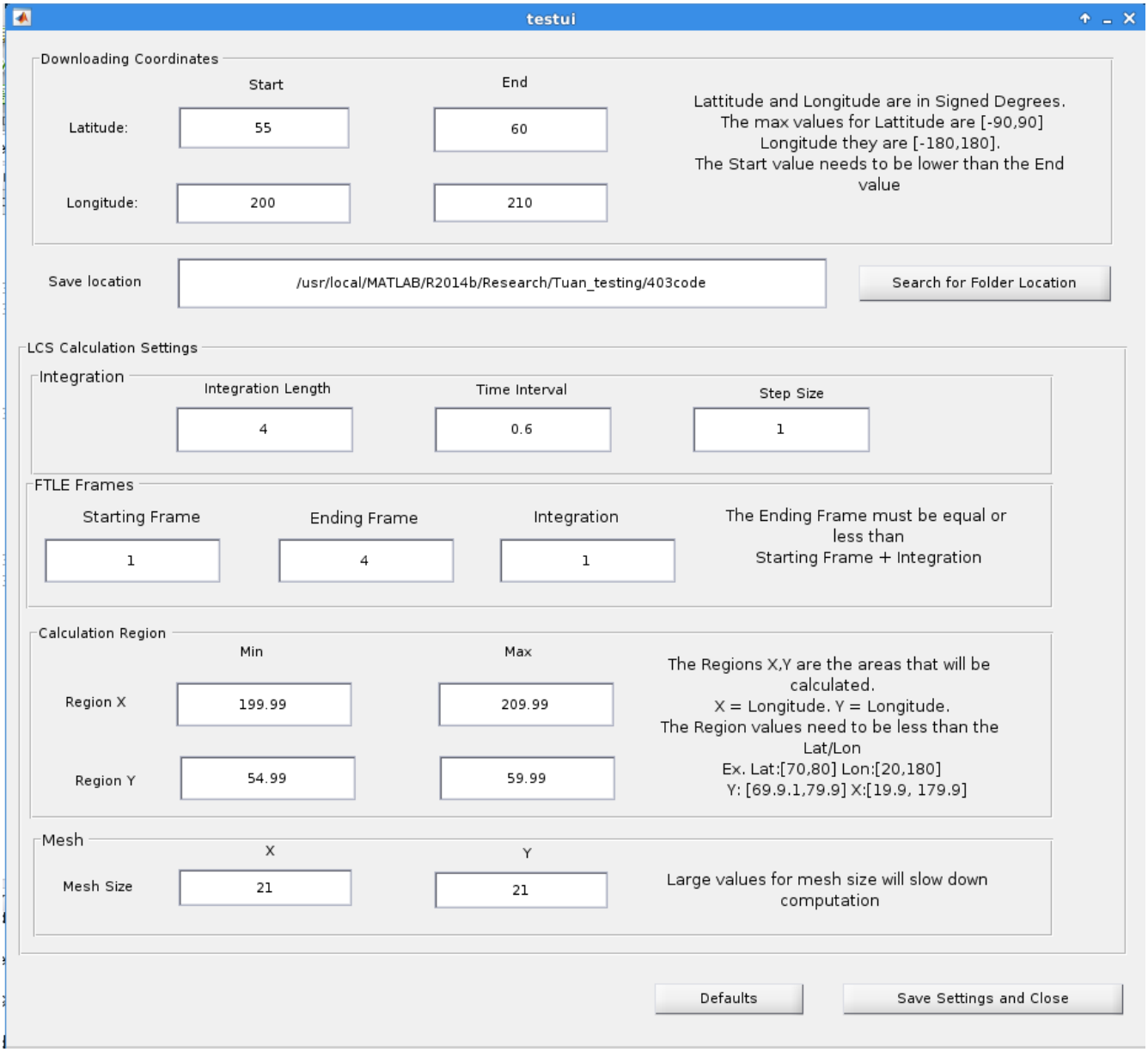
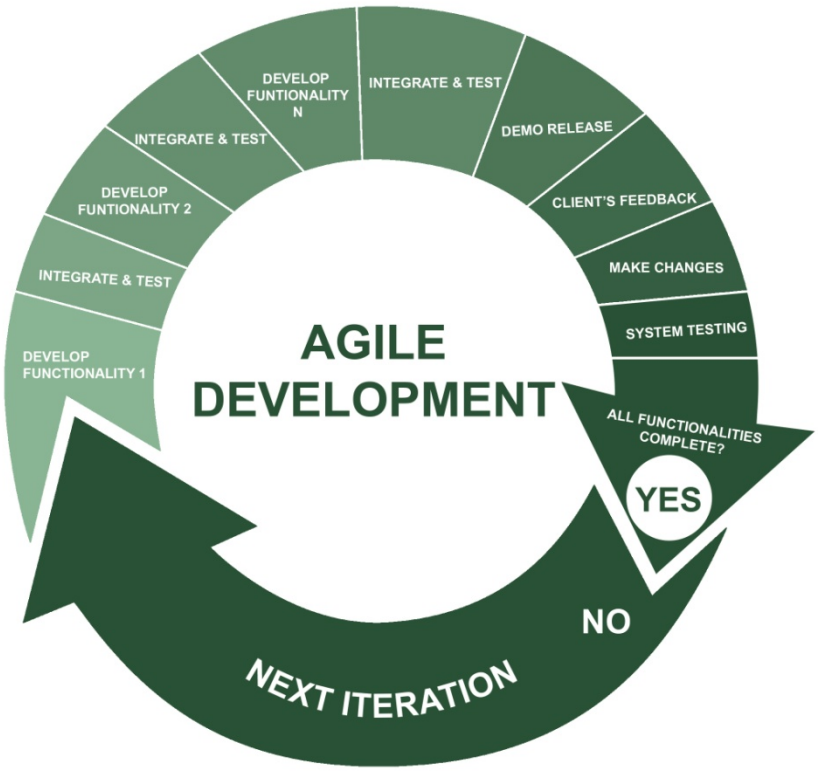


Figure 3.4: The GUI for the settings

3.4 Agile as a coding methodology and for project management

In the previous chapter agile coding was mentioned as the standard of how the project was to be written. This section pertains to how code for the project would proceed. Using agile for project management describes how to accomplish tasks to complete a project. The method to use agile are summarized as working with your customer to produce shippable software each iteration [13]. With agile, each iteration which tends to span about two weeks, you will write and test a piece of software that will contribute to the project (Figure3.4). Working in this way allows for flexibility if there is a change in requirements from the customer or if an issue with time and scope is found. This will give team members a better grasp of how long tasks will take to complete and let the team adjust better in their planning. With spending time with the customer often helps to create a better product that is suited for them. By using agile in the development process of this project bugs and tasks were found that would be out of the budgeted time and had to be dropped or adjusted.



Taken from: http://www.strategicstaff.com.php53-8.ord1-1.websitetestlink.com/wp-content/uploads/2013/08/AGILE.jpg

Figure 3.5: Example of an agile iteration

Chapter 4

User Manual

4.1 Introduction

The code for this program is available at:
https://github.com/thuynh3uaa/Backend-for-Ocean-Anaylsis-Program.git

This is a program that will download ocean data and run calculations on it and output files that have coordinates and values that you can use to plot and see LCS. This program is a part of a bigger project that has a program that will plot using the files output from this program. These LCS are structures in flowing fluids that effect the area around them and alter how the fluids flow in their vicinity. This has many applications in real life from monitoring things like plankton in the ocean to tracing where debris flows after a wreckage.

4.2 Getting Started

You will a Linux system as this program was written for it, it will work on a Windows but you will need to make some changes in the code mentioned in the **4.4**. A copy of MATLAB will be needed to run the code, because it is a proprietary software, you will need to either purchase or obtain a copy. You will also need a program that will automatically run things, for Linux Ubuntu there is crontab that will automatically run tasks at specified times.

The program currently consists of nine files:

- timeblockFormatDataFcn.m
- testui.m
- testui.fig
- normalToNOAA.m
- Main.m
- lcscaltesting.m
- downloadDataFcn.m
- csv.FormatFcn.m
- settings.mat

These files will need to all be in the same folder to work. The settings.mat is not particularly important if you have it, the program will create its own after the first run if it does not appear.

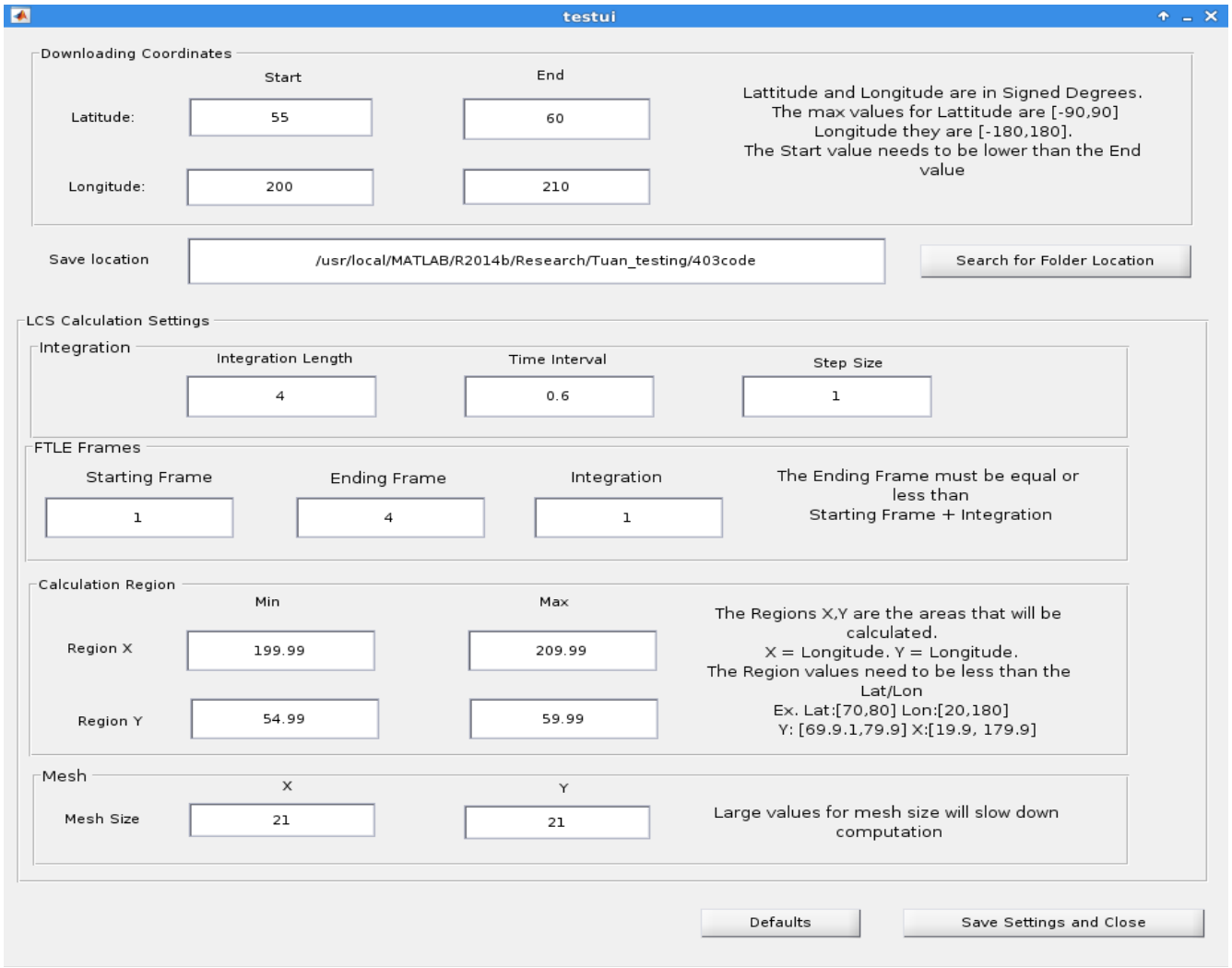


Figure 4.1: Settings Menu

4.3 How to Use

All the files will need to be in the same folder. During your first run of the program you will use MATLAB to open and run testui.m. When you run it the settings GUI (Figure 4.1) will appear.

The GUI controls the settings for the program. If there is a problem opening the GUI then delete the settings.mat file. The settings are separated into two main sections. There is the Downloading Coordinates and LCS Calculation Settings sections.

Downloading Coordinates - Lets you change the values of which areas of Earth you want to pull data from. The units for latitude and longitude are signed degrees, this means that values go from -180 to 180 for longitude, if the number is negative it means east of the equator, if it is positive then it signifies west of the equator. Latitude goes from -90 to 90, negative denotes south of the equator and positive means north of the equator. The start and end values represent the leftmost and the rightmost coordinates for longitude and the lowest to highest coordinates for latitude. The program will not work correctly if the value for start is higher than the value for end.

Save Location - Specifies where the data that is downloaded and formatted will be stored. You can type in the file location or you can search for it using the search for folder button, it will bring up a window to make it easier to search.

LCS Calculation - Deals with variables that are used when calculating LCS from the downloaded data.

Integration - This section doesn't not work correctly and needs further development. Integration Length is the time duration in frames for integration. Time Interval is measured in seconds and works if the value is less than one. The step size should normally be set to one.

FTLE Frames - The variables here deal with how many days worth of data are being downloaded, usually they will not be changed. Currently the program downloads eight days worth of data that is then split into 64 frames. Normally the starting frame will be set to 1. The ending frame and the integration values can be set to whatever you like, but the ending frame must be equal or greater than the integration plus starting frame.

Calculate Region Size - The area inside of the data you collected that you wish to examine. The x value corresponds to longitude and the y value corresponds to the latitude. Usually you will want to calculate on the entire region of the data you put in so it is normal to go .1 less the values you put in the download coordinates section at the top of the window.

Mesh - Akin to the resolution of the plot after running the calculations on the data. As you increase the values of the mesh size you will get a more refined image but computation time will increase. Low values around the twenties are not very precise so values over a hundred are recommended.

At the bottom are two buttons the default and the save and close. Pressing the default button will put

into the boxes the values that were used during. These numbers are guaranteed to produce a plot. The save and close button will save the values written in the boxes to a file called settings.mat. The program uses the values stored in settings.mat to work.

Once you have gone through the settings menu and changed the variables to your liking, you now need to open up your automated task running program. This example will be written using crontab.

In crontab write
30 06 * * * cd /home/person/desktop/research/; /home/person/desktop/research/Main.m

Crontab uses 24 hour days instead of AM/PM. The 30 signifies minutes, 06 signifies hour, the 3rd * signifies day, etc. For more information look up the crontab wiki:
https://help.ubuntu.com/community/CronHowto

With that the program will be set up, it will automatically download data and calculate LCS from it using the values stored in settings.mat

4.4 Details of components

Each component of the program can be run on its own. This means that you can download data for any day you want or compute a certain dataset if you wish as long as you understand which inputs are needed for the functions. Refer to the documentation of the functions for more detail.

Main.m
This is the main file of the backend that runs the program. It will call the functions of the program using the settings.mat file's values and the previous day's date.

downloadDataFcn.m
This function downloads data from the NOAA website. It takes in six parameters, folder to download to, date of the data you wish to download from in the format YYYY-MM-DD, starting latitude, ending latitude, starting longitude, ending longitude.
If you wish to change the amount of days worth of data you want to download, go to line
day8= dl_date + days(8);
and change the number in days to the number desired. The max for the days of data is assumed to be eight.

For Windows

```
fileName = [downloadLocation '/velocity_from_' dl_date_str '.txt'];
fullFileName = websave(fileName,url,options);
disp('Download Finished');
cd ([homedir])
```

Change '/velocity_from_' to '\velocity_from_' inside the downloadDataFcn.

timeblockFormatDataFcn.m
This function calls the csvFormatFcn to remove commas from the csv file downloaded from the

downloadDataFcn and then breaks that file down into separate files that hold the values of each three hour block from the data. There will be 65 files normally. The parameters for the function are filename of the data and the date of the data. The date of the data is used to create a folder holding the three hour block files.

csvFormatFcn.m
Removes the commas from a csv file and returns an array of values used for the LCS calculation program. If you wish to use a different file format for data or additional data columns you will need to modify the timeblockFormatDataFcn.m as well as modify csvFormatFcn.m or create a new function to handle the data.

normalToNOAA.m
Function that converts normal signed degree coordinates into NOAA's coordinates for downloading.

lcscaltesting.m
This file needs more development. The units for the variables are not clearly known at the time and we aren't sure which variables we can use and what we can cut. This is the main function that holds the computation functions for LCS data. The parameter for the function is the folder that holds the three hour block files. It will read them in and then perform computations using those files and values from the settings.mat file. A folder called Mat Files will be made in the folder that contains the time block files. The files in that folder are used for plotting the LCS.

testui.m
This file controls the functionality of the UI. This is where the values of the settings.mat file are set. If you wish you change the UI then in MATLAB type in guide and select testui. If you want to add more values to the UI you will also need to add them into the settings.mat. Changes will need to be made in the testui_OpeningFcn, saveCloseBtn_Callback, and defaultBtn_Callback.

Chapter 5

Summary and Conclusion

5.1 Summary

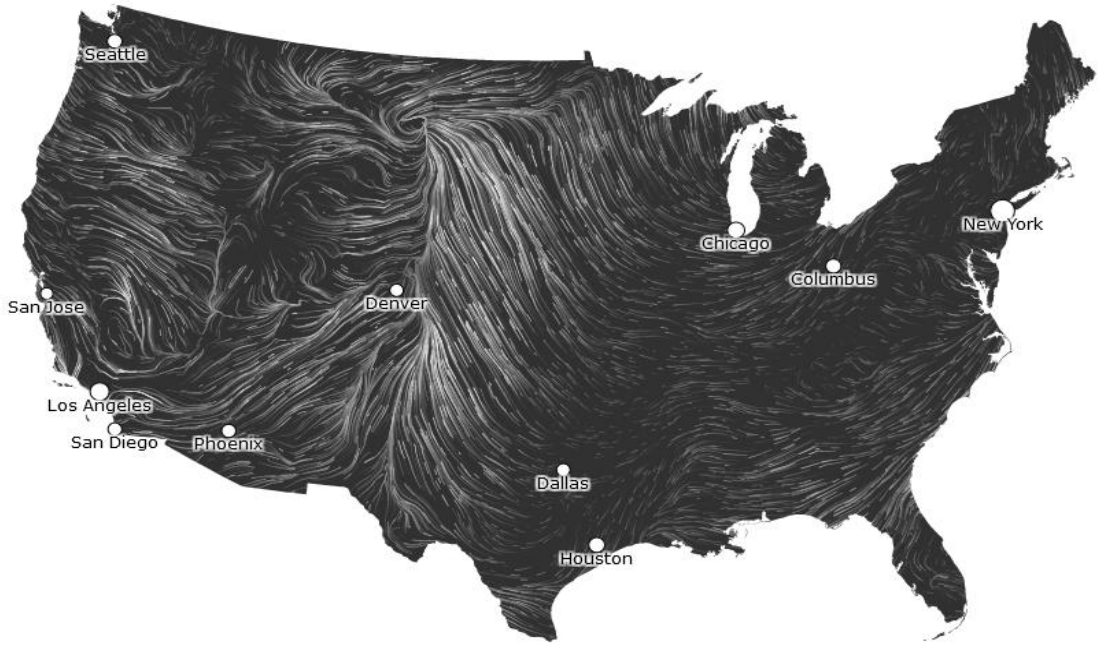
The project has been modified from the original planning. We were able to successful implement automated downloading for the data. Automating the LCS calculation program was also successful and data is being plotted onto a map. The requirements set out at the beginning of the project were accomplished, but in a different way than was planned. The automation of the LCS calculation program does need a bit further development, the plotting is not done in Google Maps, and we were not able to get to the website portion of the project to publish the data on maps.

5.2 Implications

This project of ocean data analysis is just one of many that will come along as we try to understand more about Earth. This was just a stepping stone that will help to build up a larger application. We are helping researchers cut down on menial labor and speed up their process for data analysis. Right now it is only designed for ocean current data but with a bit more development it can easily work with wind current data.

5.3 Recommendations

The current project is workable but it can use improvements. Personally, I suggest that the code could be rewritten in a free language. MATLAB has many tools and features but the fact that it is a proprietary software may dissuade people from attempting to contribute. A in background fluid dynamics or at least high level math skills are recommended as this project is fairly math focused, having someone with background in these areas to help will also work. The LCS calculation program will need some work done to it, we are fairly certain that it can be trimmed down more but we weren't sure which variables and methods could be cut out. The addition of analysis of wind data (Figure 5.1) can be added with further development. Threading or parallel processing should definitely be implemented into the program. The LCS calculation does a lot of computing but uses none of these techniques for faster execution. For larger data sets we can see the computation time becoming quite long, we were not able to reach the point for optimizing the code but it should be done. And finally a website should be made that allows users see plots of their choosing.



Taken from: http://hint.fm/wind/

Figure 5.1: A wind map

5.4 Conclusion

We have made a small contribution towards helping researchers with finding lagrangian coherent structures. There are many applications for this project. With plans for a web service and future addition of wind currents data, there is still much to do on this project.

Bibliography

[1] National Oceanic and Atmospheric Administration. How much water is in the ocean?. 2016. http://oceanservice.noaa.gov/facts/oceanwater.html

[2] Peacock, T., & Haller, G. (2013). Lagrangian coherent structures: The hidden skeleton of fluid flows. In Physics Today 66 (pp. 41-47). http://georgehaller.com/reprints/PhysToday.pdf

[3] Kasten, J., Hotz, I., & Hege, H. C. (2012). On the elusive concept of lagrangian coherent structures. In Topological Methods in Data Analysis and Visualization II (pp. 207-220). Springer Berlin Heidelberg. https://www.zib.de/hotz/publications/paper/kasten11_LCS.pdf

[4] Shadden, S. Lagrangian Coherent Structures. 2016. http://shaddenlab.berkeley.edu/uploads/shadden11.pdf

[5] Imam, J. Microbead ban signed by President Obama. 2015. http://www.cnn.com/2015/12/30/health/obama-bans-microbeads/

[6] MathWorks. 2016. http://www.mathworks.com/products/matlab/

[7] Ubuntu. CronHowto. 2015. https://help.ubuntu.com/community/CronHowto

[8] Mawdsley, J. Coding Conventions: Make Them Agile. 2006 http://www.drdobbs.com/architecture-and-design/coding-conventions-make-them-agile/193003844?pgno=1

[9] Holscher, E., & Howard, T. A beginners guide to writing documentation. 2013. http://docs.writethedocs.org/writing/beginners-guide-to-docs/

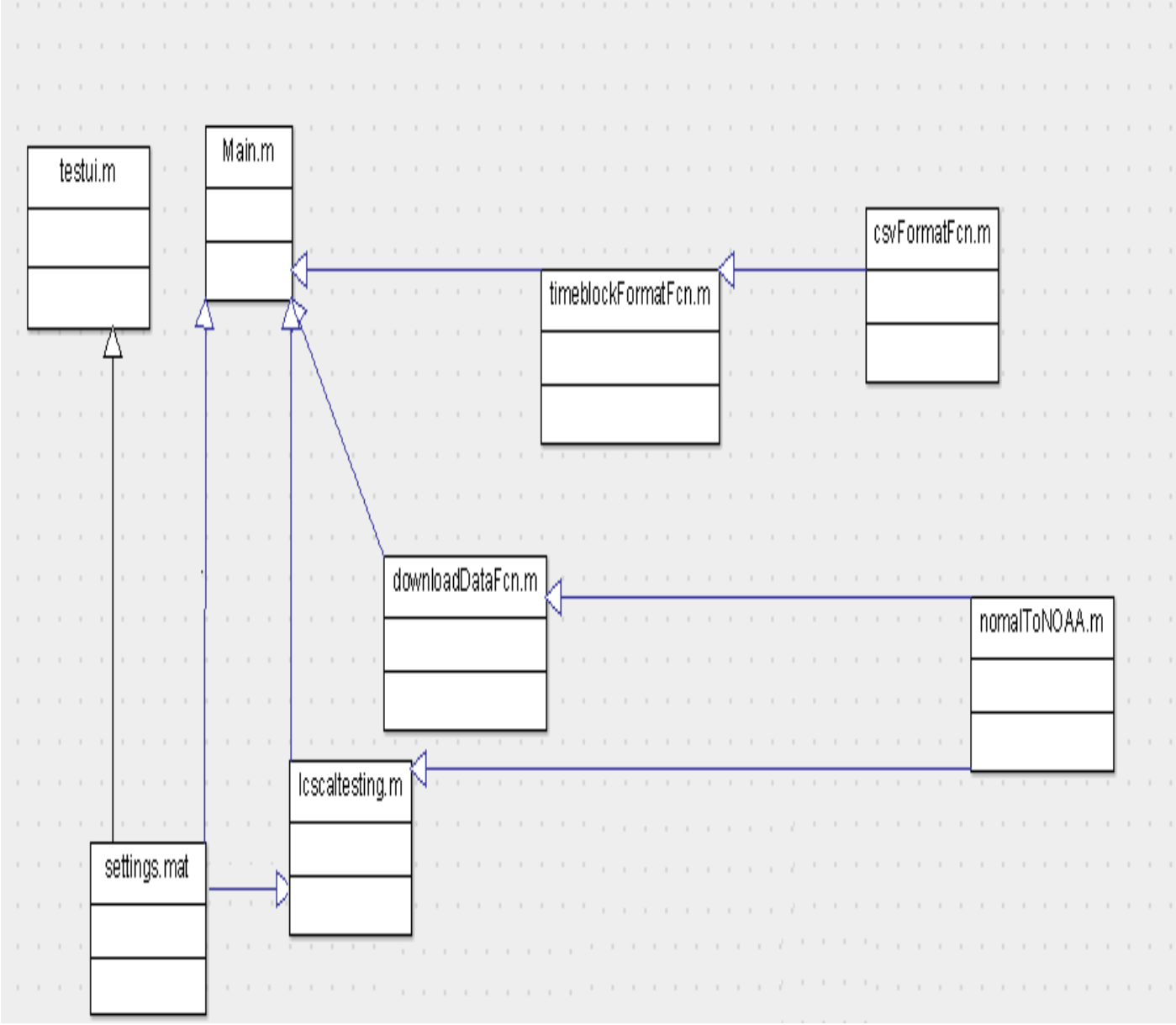
[10] Radigan, D. Why code review matters (and actually saves time!). 2016 https://www.atlassian.com/agile/code-reviews

[11] C2. Modular Programming. 2014. http://c2.com/cgi/wiki?ModularProgramming

[12] Guru99. What is Black Box Testing?. 2016 http://www.guru99.com/black-box-testing.html

[13] Agilemanifesto. Principles behind the Agile Manifesto. 2016. http://agilemanifesto.org/principles.html

Appendix A: UML



Appendix B: Source Code

The source code for the original LCS Calculation Program can be found here:

http://dabirilab.com/software

The code for this program is available at:

https://github.com/thuynh3uaa/Backend-for-Ocean-Anaylsis-Program.git

Main.m

```
function Main
%Script that runs the function for downloading, formatting, and LCS
%calculations
homefolder = pwd;
load('settings.mat');
cd (settings_folderLocation)
%Run data download from NOAA Function
%Takes in 4 parameters, Lattitude & Longitude Start and End from the
%Settings File.
dl_date = datetime('now','Format','yyyy-MM-dd');
dl_date = dl_date - days(1);
[filename,current_date_str] = downloadDataFcn(settings_folderLocation, dl_date,...
    settings_latStart,settings_latEnd,...
    settings_longStart, settings_longEnd);
disp(filename);
fileLocationForLCS = timeblockFormatDataFcn(filename, current_date_str);
cd (homefolder)
lcscaltesting(fileLocationForLCS);
cd (homefolder)
```

normalToNOAA.m

```
function x = normalToNOAA(y)
if sign(y) == -1
    x = 360 + y;
else
    temp = 360 + y;
    if temp > 434
        x = temp - 360;
    else
        x = temp;
    end
end
```

downloadDataFcn.m

```
function [downloadedFileName, dl_date_str] = downloadDataFcn(downloadLocation,dl_date,
lat_s, lat_e, lon_s, lon_e)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
global homedir
homedir = pwd;

dl_date_str = datestr(dl_date,'yyyy-mm-dd');
day8= dl_date + days(8);
day8str = datestr(day8,'yyyy-mm-dd');

%%
if lat_e == 90
    lat_e = 89.90947;
end

if lon_s == -180
    lon_s = 74.16;
else
    lon_s = normalToNOAA(lon_s);
end

if lon_e == 180
    lon_e = 434.06227;
else
    lon_e = normalToNOAA(lon_e);
end
%%
lat_start = num2str(lat_s);
lat_stop = num2str(lat_e);
lon_start = num2str(lon_s);
lon_stop = num2str(lon_e);

tz = 'T00:00:00Z';
urlhead =
'http://coastwatch.pfeg.noaa.gov/erddap/griddap/ncepRtofsG2DFore3hrlyProg.csv0?';
url_u_vel = ['u_velocity[(' dl_date_str  '):1:(' day8str  ')][(1.0):1:(1.0)][(' lat_start
'):1:(' lat_stop ')][(' lon_start '):1:(' lon_stop ')]'];
url_v_vel = ['v_velocity[(' dl_date_str  '):1:(' day8str  ')][(1.0):1:(1.0)][(' lat_start
'):1:(' lat_stop ')][(' lon_start '):1:(' lon_stop ')]'];
url = [urlhead url_u_vel ',' url_v_vel];
```

```
downloadedFileName = downloadFromWeb(url,dl_date_str, downloadLocation);
end

function fullFileName = downloadFromWeb(url, dl_date_str, downloadLocation)
global homedir
options = weboptions('Timeout', 60);
% downloadDestination = exist('rawdata' , 'dir');
% if downloadDestination == 0
%     mkdir rawdata
%     cd rawdata
% else
%     cd rawdata
% end
fileName = [downloadLocation '/velocity_from_' dl_date_str '.txt'];
fullFileName = websave(fileName,url,options);
disp('Download Finished');
cd ([homedir])
%FormatData(current_date_str)
end
```

timeblockFormatDataFcn.m

```
function fileLocationForLCS = timeblockFormatDataFcn(filename,current_date_str)
[DateTime, Lat, Lon, u_vel, v_vel] = csvFormatFcn(filename);

%formatting files after removing comma from csv%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t = datetime(DateTime, 'InputFormat', 'uuuu-MM-dd''T''HH:mm:ss''Z', 'Format', 'yyyy-MM-dd
HH:mm:ss');
ds = dataset(t,Lat,Lon,u_vel,v_vel);
start = datetime([current_date_str ' 00:00:00'], 'Format', 'yyyy-MM-dd HH:mm:ss');
%start_stamp = datestr(start);
h = exist('formatted_data' , 'dir');
if h == 0
    mkdir formatted_data
    cd formatted_data
else
    cd formatted_data
end

j = exist(current_date_str, 'dir');
if j == 0
    mkdir([current_date_str])
    cd ([current_date_str])
else
    cd([current_date_str])
end
%g = 1;
for n = 0:8

   for i = 0:7
        hr = i * 3;


        temp_ds = ds(ds.t == start + days(n) + hours(hr),{'Lat', 'Lon', 'u_vel',
'v_vel'});
        day = sprintf('%02d' , n);
        hrs = sprintf('%02d' , hr);
        formattedName = ['data from ' current_date_str ' D' day ' Hr' hrs '.txt'];

        %c = int2str(g);
        %formattedName = ['velocity' c '.txt'];
        %g = g + 1;
        export(temp_ds,'file',formattedName,'Delimiter', ' ', 'WriteVarNames', false);
        if n == 8
            break;
        end
   end
```

```
end
display('Download Testing End');
fileLocationForLCS = pwd;
end
```

csvFormatFcn.m

```
function [DateTime, Lat, Lon, u_vel, v_vel] = csvFormatFcn(filename)
% pathHead = '/usr/local/MATLAB/R2014b/Research/Tuan_testing/rawdata/velocity_from_';
% dl_date = datetime('now','Format','yyyy-MM-dd');
% current_date_str = datestr(dl_date,'yyyy-mm-dd');
% filename = [pathHead current_date_str '.txt'];



%% Initialize variables.
%filename = '/usr/local/MATLAB/R2014b/Research/Tuan testing/veltest1.txt';
delimiter = ',';

%% Format string for each line of text:
%   column1: text (%s)
%      column3: double (%f)
%   column4: double (%f)
%      column5: double (%f)
%   column6: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%*s%f%f%f%f%[^¥n¥r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'EmptyValue' ,NaN,
'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
DateTime = dataArray{:, 1};
Lat = dataArray{:, 3};
Lon = dataArray{:, 2};
u_vel = dataArray{:, 4};
```

```
v_vel = dataArray{:, 5};


disp('Script Done');
%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;
end
```

testui.m

```
function varargout = testui(varargin)
% TESTUI MATLAB code for testui.fig
%      TESTUI, by itself, creates a new TESTUI or raises the existing
%      singleton*.
%
%      H = TESTUI returns the handle to a new TESTUI or the handle to
%      the existing singleton*.
%
%      TESTUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TESTUI.M with the given input arguments.
%
%      TESTUI('Property','Value',...) creates a new TESTUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before testui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to testui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help testui

% Last Modified by GUIDE v2.5 06-Apr-2016 01:43:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @testui_OpeningFcn, ...
                   'gui_OutputFcn',  @testui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before testui is made visible.
function testui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to testui (see VARARGIN)

% Choose default command line output for testui
handles.output = hObject;

%Check if saving files has been initiated yet
settingsFile = 'settings.mat';
settingsExist = exist(settingsFile, 'file');

if settingsExist == 2
    load(settingsFile);
    set(handles.latSEbox, 'string', num2str(settings_latStart))
    set(handles.latEEbox, 'string', num2str(settings_latEnd))
    set(handles.lonSEbox, 'string', num2str(settings_longStart))
    set(handles.lonEEbox, 'string', num2str(settings_longEnd))
    set(handles.folderEbox, 'string', settings_folderLocation)
    set(handles.integEbox, 'string', num2str(settings_integrationL))
    set(handles.timeinEbox, 'string', num2str(settings_timeInter))
    set(handles.stepEbox, 'string', num2str(settings_stepSize))
    set(handles.regionxminEbox, 'string', num2str(settings_regionXMin))
    set(handles.regionxmaxEbox, 'string', num2str(settings_regionXMax))
    set(handles.regionyminEbox, 'string', num2str(settings_regionYMin))
    set(handles.regionymaxEbox, 'string', num2str(settings_regionYMax))
    set(handles.meshxEbox, 'string', num2str(settings_meshX))
    set(handles.meshyEbox, 'string', num2str(settings_meshY))
    set(handles.startFEbox, 'string', num2str(settings_startF))
    set(handles.endFEbox, 'string', num2str(settings_endF))
    set(handles.frameIntEbox, 'string', num2str(settings_frameInt))

else
    %Defaults that were used during testing of project
    set(handles.latSEbox, 'string', num2str(55))
    set(handles.latEEbox, 'string', num2str(60))
    set(handles.lonSEbox, 'string', num2str(200))
    set(handles.lonEEbox, 'string', num2str(210))
    set(handles.folderEbox, 'string', 'Please set a folder to download to')
    set(handles.integEbox, 'string', num2str(4))
    set(handles.timeinEbox, 'string', num2str(0.6))
    set(handles.stepEbox, 'string', num2str(1))
    set(handles.regionxminEbox, 'string', num2str(199.99))
```

```
    set(handles.regionxmaxEbox, 'string', num2str(209.99))
    set(handles.regionyminEbox, 'string', num2str(54.99))
    set(handles.regionymaxEbox, 'string', num2str(59.99))
    set(handles.meshxEbox, 'string', num2str(21))
    set(handles.meshyEbox, 'string', num2str(21))
    set(handles.startFEbox, 'string', num2str(1))
    set(handles.endFEbox, 'string', num2str(4))
    set(handles.frameIntEbox, 'string', num2str(1))
end

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes testui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = testui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function latSEbox_Callback(hObject, eventdata, handles)
% hObject    handle to latSEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of latSEbox as text
%        str2double(get(hObject,'String')) returns contents of latSEbox as a double


% --- Executes during object creation, after setting all properties.
function latSEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latSEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function lonSEbox_Callback(hObject, eventdata, handles)
% hObject    handle to lonSEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lonSEbox as text
%        str2double(get(hObject,'String')) returns contents of lonSEbox as a double


% --- Executes during object creation, after setting all properties.
function lonSEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lonSEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function latEEbox_Callback(hObject, eventdata, handles)
% hObject    handle to latEEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of latEEbox as text
%        str2double(get(hObject,'String')) returns contents of latEEbox as a double


% --- Executes during object creation, after setting all properties.
function latEEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latEEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function lonEEbox_Callback(hObject, eventdata, handles)
% hObject    handle to lonEEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lonEEbox as text
%        str2double(get(hObject,'String')) returns contents of lonEEbox as a double


% --- Executes during object creation, after setting all properties.
function lonEEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lonEEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function folderEbox_Callback(hObject, eventdata, handles)
% hObject    handle to folderEbox (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of folderEbox as text
%        str2double(get(hObject,'String')) returns contents of folderEbox as a double


% --- Executes during object creation, after setting all properties.
function folderEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to folderEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in folderSaveBtn.
function folderSaveBtn_Callback(hObject, eventdata, handles)
% hObject    handle to folderSaveBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
g = uigetdir();

if g ~= 0
set(handles.folderEbox, 'string',g);

end


% --- Executes on button press in saveCloseBtn.
function saveCloseBtn_Callback(hObject, eventdata, handles)
% hObject    handle to saveCloseBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
settings_latStart = str2double(get(handles.latSEbox, 'string'));
settings_latEnd = str2double(get(handles.latEEbox, 'string'));
settings_longStart = str2double(get(handles.lonSEbox, 'string'));
settings_longEnd = str2double(get(handles.lonEEbox, 'string'));
settings_folderLocation = get(handles.folderEbox, 'string');
settings_integrationL = str2double(get(handles.integEbox, 'string'));
settings_timeInter = str2double(get(handles.timeinEbox, 'string'));
settings_stepSize = str2double(get(handles.stepEbox, 'string'));
```

```
settings_regionXMin = str2double(get(handles.regionxminEbox, 'string'));
settings_regionXMax = str2double(get(handles.regionxmaxEbox, 'string'));
settings_regionYMin = str2double(get(handles.regionyminEbox, 'string'));
settings_regionYMax = str2double(get(handles.regionymaxEbox, 'string'));
settings_meshX = str2double(get(handles.meshxEbox, 'string'));
settings_meshY = str2double(get(handles.meshyEbox, 'string'));
settings_startF = str2double(get(handles.startFEbox, 'string'));
settings_endF = str2double(get(handles.endFEbox, 'string'));
settings_frameInt = str2double(get(handles.frameIntEbox, 'string'));

save('settings.mat', 'settings_latStart', 'settings_latEnd', 'settings_longStart',...
    'settings_longEnd', 'settings_folderLocation', 'settings_integrationL',...
    'settings_timeInter','settings_stepSize', 'settings_regionXMin',...
    'settings_regionXMax', 'settings_regionYMin', 'settings_regionYMax',...
    'settings_meshX', 'settings_meshY', 'settings_startF', 'settings_endF',...
    'settings_frameInt');
close(handles.figure1);



function integEbox_Callback(hObject, eventdata, handles)
% hObject    handle to integEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of integEbox as text
%        str2double(get(hObject,'String')) returns contents of integEbox as a double


% --- Executes during object creation, after setting all properties.
function integEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to integEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function timeinEbox_Callback(hObject, eventdata, handles)
% hObject    handle to timeinEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of timeinEbox as text
%        str2double(get(hObject,'String')) returns contents of timeinEbox as a double


% --- Executes during object creation, after setting all properties.
function timeinEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to timeinEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function stepEbox_Callback(hObject, eventdata, handles)
% hObject    handle to stepEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of stepEbox as text
%        str2double(get(hObject,'String')) returns contents of stepEbox as a double


% --- Executes during object creation, after setting all properties.
function stepEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to stepEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function regionxminEbox_Callback(hObject, eventdata, handles)
% hObject    handle to regionxminEbox (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of regionxminEbox as text
%        str2double(get(hObject,'String')) returns contents of regionxminEbox as a double


% --- Executes during object creation, after setting all properties.
function regionxminEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to regionxminEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function regionxmaxEbox_Callback(hObject, eventdata, handles)
% hObject    handle to regionxmaxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of regionxmaxEbox as text
%        str2double(get(hObject,'String')) returns contents of regionxmaxEbox as a double


% --- Executes during object creation, after setting all properties.
function regionxmaxEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to regionxmaxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function regionyminEbox_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to regionyminEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of regionyminEbox as text
%        str2double(get(hObject,'String')) returns contents of regionyminEbox as a double


% --- Executes during object creation, after setting all properties.
function regionyminEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to regionyminEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function regionymaxEbox_Callback(hObject, eventdata, handles)
% hObject    handle to regionymaxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of regionymaxEbox as text
%        str2double(get(hObject,'String')) returns contents of regionymaxEbox as a double


% --- Executes during object creation, after setting all properties.
function regionymaxEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to regionymaxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function meshxEbox_Callback(hObject, eventdata, handles)
% hObject    handle to meshxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of meshxEbox as text
%        str2double(get(hObject,'String')) returns contents of meshxEbox as a double


% --- Executes during object creation, after setting all properties.
function meshxEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to meshxEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function meshyEbox_Callback(hObject, eventdata, handles)
% hObject    handle to meshyEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of meshyEbox as text
%        str2double(get(hObject,'String')) returns contents of meshyEbox as a double


% --- Executes during object creation, after setting all properties.
function meshyEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to meshyEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in defaultBtn.
function defaultBtn_Callback(hObject, eventdata, handles)
% hObject    handle to defaultBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    %Defaults that were used during testing of project
    set(handles.latSEbox, 'string', num2str(55))
    set(handles.latEEbox, 'string', num2str(60))
    set(handles.lonSEbox, 'string', num2str(200))
    set(handles.lonEEbox, 'string', num2str(210))
    set(handles.folderEbox, 'string', 'Please set a folder to download to')
    set(handles.integEbox, 'string', num2str(4))
    set(handles.timeinEbox, 'string', num2str(0.6))
    set(handles.stepEbox, 'string', num2str(1))
    set(handles.regionxminEbox, 'string', num2str(199.99))
    set(handles.regionxmaxEbox, 'string', num2str(209.99))
    set(handles.regionyminEbox, 'string', num2str(54.99))
    set(handles.regionymaxEbox, 'string', num2str(59.99))
    set(handles.meshxEbox, 'string', num2str(21))
    set(handles.meshyEbox, 'string', num2str(21))
    set(handles.startFEbox, 'string', num2str(1))
    set(handles.endFEbox, 'string', num2str(4))
    set(handles.frameIntEbox, 'string', num2str(1))




function startFEbox_Callback(hObject, eventdata, handles)
% hObject    handle to startFEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of startFEbox as text
```

```
%        str2double(get(hObject,'String')) returns contents of startFEbox as a double


% --- Executes during object creation, after setting all properties.
function startFEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to startFEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function endFEbox_Callback(hObject, eventdata, handles)
% hObject    handle to endFEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of endFEbox as text
%        str2double(get(hObject,'String')) returns contents of endFEbox as a double


% --- Executes during object creation, after setting all properties.
function endFEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to endFEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function frameIntEbox_Callback(hObject, eventdata, handles)
% hObject    handle to frameIntEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of frameIntEbox as text
%        str2double(get(hObject,'String')) returns contents of frameIntEbox as a double


% --- Executes during object creation, after setting all properties.
function frameIntEbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frameIntEbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

lcscaltesting.m

```
% In  LoadOriginalDataMenuItem_Callback()
% orginaldata_pathname - main program will put this script into the folder with the data,
this just grabs the name
% of the folder it's in
% originaldata_filename - this will grab every filename inside the folder. Right now the
program is assumed to only run one time
% so it doesn't check if files that won't work are in the folder. Will
% throw error if so
%
% In ConvertToMATFilesMenuItem_Callback()
% mkdir MatFiles - This makes the folder for the MAT files
% directory_name = 'MatFiles' - This tells where to put the MAT files
%
% In function SaveFTLECalculationMenuItem_Callback()
% output_file = '/FTLE.txt'; - Name of what the FTLE file will be, keep /
% output_path = directory_name - Name of the directory FTLE will be put in

function lcscaltesting(varargin)
global settings_startF settings_endF settings_frameInt settings_integrationL
settings_meshY settings_meshX
global settings_timeInter settings_stepSize settings_regionXMax settings_regionYMax
settings_regionXMin settings_regionYMin

load('settings.mat','settings_startF', 'settings_endF', 'settings_frameInt',
'settings_integrationL', 'settings_meshY',...
    'settings_meshX', 'settings_timeInter', 'settings_stepSize', 'settings_regionXMax',
'settings_regionYMax',...
    'settings_regionXMin', 'settings_regionYMin' );
b = char(varargin);
 cd (b)
%% % LCS_Calculation M-file for LCS_Calculation.fig
% %      LCS_Calculation, by itself, creates a new LCS_Calculation or raises the existing
% %      singleton*.
% %
% %      H = LCS_Calculation returns the handle to a new LCS_Calculation or
% %      the handle to
% %      the existing singleton*.
% %
% %      LCS_Calculation('CALLBACK',hObject,eventData,handles,...) calls the local
% %      function named CALLBACK in LCS_Calculation.M with the given input arguments.
% %
% %      LCS_Calculation('Property','Value',...) creates a new LCS_Calculation or raises
the
% %      existing singleton*.  Starting from the left, property value pairs are
% %      applied to the GUI before LCS_Calculation_OpeningFunction gets called.  An
```

```
% %      unrecognized property name or invalid value makes property application
% %      stop.  All inputs are passed to LCS_Calculation_OpeningFcn via varargin.
% %
% %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
% %      instance to run (singleton)".
% %
% % See also: GUIDE, GUIDATA, GUIHANDLES
%
% % Edit the above text to modify the response to help LCS_Calculation
%
% % Last Modified by GUIDE v2.5 07-Mar-2009 21:59:45
%
% % Begin initialization code - DO NOT EDIT
% gui_Singleton = 1;
% gui_State = struct('gui_Name',       mfilename, ...
%                    'gui_Singleton',  gui_Singleton, ...
%                    'gui_OpeningFcn', @LCS_Calculation_OpeningFcn, ...
%                    'gui_OutputFcn',  @LCS_Calculation_OutputFcn, ...
%                    'gui_LayoutFcn',  [] , ...
%                    'gui_Callback',   []);
% if nargin & isstr(varargin{1})
%     gui_State.gui_Callback = str2func(varargin{1});
% end
%
% if nargout
%     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
% else
%     gui_mainfcn(gui_State, varargin{:});
% end
% % End initialization code - DO NOT EDIT
%
%
% % --- Executes just before LCS_Calculation is made visible.
% function LCS_Calculation_OpeningFcn(hObject, eventdata, handles, varargin)
% % This function has no output args, see OutputFcn.
% % hObject    handle to figure
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
% % varargin   command line arguments to LCS_Calculation (see VARARGIN)
%
% % Choose default command line output for LCS_Calculation
% handles.output = hObject;
%
% % Update handles structure
% guidata(hObject, handles);
%
% % UIWAIT makes LCS_Calculation wait for user response (see UIRESUME)
```

```
% % uiwait(handles.figure1);
%
%
% % --- Outputs from this function are returned to the command line.
% function varargout = LCS_Calculation_OutputFcn()
% % varargout  cell array for returning output args (see VARARGOUT);
% % hObject    handle to figure
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Get default command line output from handles structure
% varargout{1} = handles.output;
%
%
% % --- Executes during object creation, after setting all properties.
% function starting_frame_CreateFcn()
% % hObject    handle to starting_frame (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
% function starting_frame_Callback()
% % hObject    handle to starting_frame (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of starting_frame as text
% %        str2double(get(hObject,'String')) returns contents of starting_frame as a
double
%
%
% % --- Executes during object creation, after setting all properties.
% function end_frame_CreateFcn()
% % hObject    handle to starting_frame (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
```

```
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
% function end_frame_Callback()
% % hObject    handle to end_frame (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of end_frame as text
% %        str2double(get(hObject,'String')) returns contents of end_frame as a
% %        double
%
%
% % --- Executes during object creation, after setting all properties.
% function interval_CreateFcn()
% % hObject    handle to interval (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
% function interval_Callback()
% % hObject    handle to interval (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of interval as text
% %        str2double(get(hObject,'String')) returns contents of interval as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function integration_length_CreateFcn()
% % hObject    handle to integration_length (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
```

```
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function integration_length_Callback()
% % hObject    handle to integration_length (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of integration_length as text
% %        str2double(get(hObject,'String')) returns contents of integration_length as a
double
%
%
% % --- Executes during object creation, after setting all properties.
% function time_interval_CreateFcn()
% % hObject    handle to time_interval (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function time_interval_Callback()
% % hObject    handle to time_interval (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of time_interval as text
% %        str2double(get(hObject,'String')) returns contents of time_interval as a
double
%
%
% % --- Executes during object creation, after setting all properties.
% function meshsize_x_CreateFcn()
% % hObject    handle to meshsize_x (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
```

```
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function meshsize_x_Callback()
% % hObject    handle to meshsize_x (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of meshsize_x as text
% %        str2double(get(hObject,'String')) returns contents of meshsize_x as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function meshsize_y_CreateFcn()
% % hObject    handle to meshsize_y (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function meshsize_y_Callback()
% % hObject    handle to meshsize_y (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of meshsize_y as text
% %        str2double(get(hObject,'String')) returns contents of meshsize_y as a double
%
%
% % --- Executes during object creation, after setting all properties.
```

```
% function region_x_min_CreateFcn()
% % hObject    handle to region_x_min (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function region_x_min_Callback()
% % hObject    handle to region_x_min (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of region_x_min as text
% %        str2double(get(hObject,'String')) returns contents of region_x_min as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function region_x_max_CreateFcn()
% % hObject    handle to region_x_max (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function region_x_max_Callback()
% % hObject    handle to region_x_max (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of region_x_max as text
% %        str2double(get(hObject,'String')) returns contents of region_x_max as a double
```

```
%
%
%
% % --- Executes during object creation, after setting all properties.
% function region_y_max_CreateFcn()
% % hObject    handle to region_y_max (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function region_y_max_Callback()
% % hObject    handle to region_y_max (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of region_y_max as text
% %        str2double(get(hObject,'String')) returns contents of region_y_max as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function region_y_min_CreateFcn()
% % hObject    handle to region_y_min (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function region_y_min_Callback()
% % hObject    handle to region_y_min (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
```

```
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of region_y_min as text
% %        str2double(get(hObject,'String')) returns contents of region_y_min as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function step_size_CreateFcn()
% % hObject    handle to step_size (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc
%     set(hObject,'BackgroundColor','white');
% else
%     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
% end
%
%
%
% function step_size_Callback()
% % hObject    handle to step_size (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of step_size as text
% %        str2double(get(hObject,'String')) returns contents of step_size as a double
%
%
% % --- Executes on button press in forward_FTLE.
% function forward_FTLE_Callback()
% % hObject    handle to forward_FTLE (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hint: get(hObject,'Value') returns toggle state of forward_FTLE
%
%
% % --- Executes on button press in backward_FTLE.
% function backward_FTLE_Callback()
% % hObject    handle to backward_FTLE (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hint: get(hObject,'Value') returns toggle state of backward_FTLE
```

```
%
% % --- Executes during object creation, after setting all properties.
% function figure1_CreateFcn()
% % hObject    handle to figure1 (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns called
%
% % Add the current directory to the path, as the pwd might change thru' the
% % gui. Remove the directory from the path when gui is closed
% % (See figure1_DeleteFcn)
% setappdata(hObject, 'StartPath', pwd);
% addpath(pwd);


% % ------------------------------------------------------------------------

LoadOriginalDataMenuItem_Callback

function LoadOriginalDataMenuItem_Callback()

global originaldata_filename originaldata_pathname find_txt find_wk1
%[originaldata_filename, originaldata_pathname, filterindex] =
uigetfile({'*.txt';'*.wk1'}, 'MultiSelect', 'on');
originaldata_pathname = [pwd '/'];
getFolder = dir(fullfile(originaldata_pathname, 'data from*'));
%getFolder = getFolder(~ismember({getFolder.name},{'.','..'}));
originaldata_filename = {getFolder.name};
filterindex = 1;
if iscell(originaldata_filename)==1
    find_txt=max(size(strfind(originaldata_filename{1},'.txt')));
    find_wk1=max(size(strfind(originaldata_filename{1},'.wk1')));
else
    find_txt=max(size(strfind(originaldata_filename,'.txt')));
    find_wk1=max(size(strfind(originaldata_filename,'.wk1')));
end

if find_txt==0 & find_wk1==0
    warndlg('Selection failed. Unknown file format.','error');
end

if filterindex==0
    warndlg('Selection failed. Matlab sometimes fails to a open large number of files.
Please select less files.','error');
end
originaldata_filename=sort_char_array(originaldata_filename);

% % ------------------------------------------------------------------------
```

```
%%
ConvertToMATFilesMenuItem_Callback

function ConvertToMATFilesMenuItem_Callback()

global originaldata_filename originaldata_pathname directory_name find_txt find_wk1
mkdir MatFiles
directory_name = 'MatFiles';
%directory_name = uigetdir('','Select directory of the folder to which MAT files are to
be saved');
%directory_name = '/Users/mb00k/Desktop/LCS_TEST/Mat Files';

% prompt = {'Enter the starting frame number:'};
% dlg_title = 'Convert Original Data to Matlab Files';
% num_lines = 1;
% def = {'1'};

% MatFileStartingFrameString = inputdlg(prompt,dlg_title,num_lines,def);
MatFileStartingFrame = 1; %str2num(MatFileStartingFrameString{1});

n_originaldata=length(originaldata_filename);
for i=1:n_originaldata
    i_frame=i+MatFileStartingFrame-1;
    filename=[originaldata_pathname originaldata_filename{i}];

    if find_txt==1
        A = dlmread(filename);
    end
    if find_wk1==1
        A = wk1read(filename);
    end
    A = sortrows(A,1);
    B=sortrows(A,2);
    ndata=max(size(A));

    if i==1
        ysize=max(find(A(:,1)==A(1,1)));
        xsize=max(find(B(:,2)==B(1,2)));
        xcoor=B(1:xsize,1);
        ycoor=A(1:ysize,2);
        [xloc yloc]=meshgrid(xcoor,ycoor);
        save([directory_name '/xgrid'],'xcoor');
        save([directory_name '/ygrid'],'ycoor');
    end

    for j=1:xsize
```

```
        u(1:ysize,j)=A(1+ysize*(j-1):ysize*j,3);
        v(1:ysize,j)=A(1+ysize*(j-1):ysize*j,4);
    end

    expression=['u' num2str(i_frame) '=u'';'];
    eval(expression);
    expression=['v' num2str(i_frame) '=v'';'];
    eval(expression);
    expression=['save(''' directory_name '/U_T' num2str(i_frame) ''','''u'
num2str(i_frame) ''');'];
    eval(expression);
    expression=['save(''' directory_name '/V_T' num2str(i_frame) ''','''v'
num2str(i_frame) ''');'];
    eval(expression);
    expression=['clear u' num2str(i_frame) ' v' num2str(i_frame) ';'];
    eval(expression);
end
```

ComputeFTLEMenuItem_Callback

```
function ComputeFTLEMenuItem_Callback()
% % hObject    handle to ComputeFTLEMenuItem (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
global nx ny f_start f_end f_int t_length tstep dt
global settings_startF settings_endF settings_frameInt settings_integrationL
settings_meshY settings_meshX
global settings_timeInter settings_stepSize settings_regionXMax settings_regionYMax
settings_regionXMin settings_regionYMin

load('/usr/local/MATLAB/R2014b/Research/Tuan_testing/403code/settings.mat');
% FTLE Type

forward_FTLE = 1;
backward_FTLE = 0;


if settings_regionYMax == 90
    settings_regionYMax = 89.90947;
end

if settings_regionXMin == -180
   settings_regionXMin = 74.16;
else
```

```
    settings_regionXMin = normalToNOAA(settings_regionXMin);
end

if settings_regionXMax == 180
    settings_regionXMax = 434.06227;
else
    settings_regionXMax = normalToNOAA(settings_regionXMax);
end

% FTLE frames Settings
starting_frame = num2str(settings_startF);%'1';
end_frame = num2str(settings_endF);%'4';
interval = num2str(settings_frameInt);%'1';

% Integration Settings
integration_length = num2str(settings_integrationL);%'4';
time_interval = num2str(settings_timeInter);%'0.6';
step_size = num2str(settings_stepSize);%'1';

% Calculation Region [X1 X2 Y1 Y2] Settings
region_x_min = num2str(settings_regionXMin);%'199.99';
region_x_max = num2str(settings_regionXMax);%'209.99';
region_y_min = num2str(settings_regionYMin);%'54.99';
region_y_max = num2str(settings_regionYMax);%'59.99';

% Mesh Size
meshsize_x = num2str(settings_meshX);%'21';
meshsize_y = num2str(settings_meshY);%'21';




f_start=str2double(starting_frame);
f_end=str2double(end_frame);
f_int=str2double(interval);
t_length=str2double(integration_length);
tstep=str2double(time_interval);
dt=str2double(step_size);

region_x1=str2double(region_x_min);
region_x2=str2double(region_x_max);
region_y1=str2double(region_y_min);
region_y2=str2double(region_y_max);

nx=str2double(meshsize_x);
ny=str2double(meshsize_y);
```

```
forward_calculation=double(forward_FTLE);
backward_calculation=double(backward_FTLE);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%some variables repeated recalled by subroutines are set as global
%variables to save computation time
global xloc yloc xcoor ycoor sxcoor sycoor umesh vmesh directory_name sigma

load([directory_name '/xgrid']);
load([directory_name '/ygrid']);
[xloc yloc]=meshgrid(xcoor,ycoor);
velo_x1=min(xcoor);
velo_x2=max(xcoor);
velo_y1=min(ycoor);
velo_y2=max(ycoor);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sxcoor=linspace(region_x1,region_x2,nx);                % define the initial region of the
fluid to track
sycoor=linspace(region_y1,region_y2,ny);
[xmesh ymesh]=meshgrid(sxcoor,sycoor);
sigma_x=xmesh';
sigma_y=ymesh';
sigma=zeros(nx,ny);
sigma0=sigma;

CalcFTLE=zeros(nx,ny)+1;
LeftDomain=zeros(nx,ny);
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Forward FTLE Calculation%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if forward_calculation==1
    for iT=f_start:f_int:f_end                          % define the time when the FTLE is
calculated

        % calculate FTLE at frame # iT
        t_proc = ['T = ', ...
            num2str( iT )];
        disp( t_proc );

        t_start=iT;                         % starting frame
```

```
t_end=t_start+t_length;
T_start=(t_start-1)*tstep;          % initial time
T_end=(t_end-1)*tstep;              % end time
T_span=T_end-T_start;               % integration time length

sigma_x=xmesh';                     % initial position
sigma_y=ymesh';

CalcFTLE=zeros(nx,ny)+1;
LeftDomain=zeros(nx,ny);

for t_integration=0:dt:t_length-dt;
    %integrate the flow trajectory from t0 to t1;
    t0=t_start+t_integration;
    t1=t0+dt;

    iiT=t0;
    %%%%%%%%%%%% read the velocity files
    expression=['load(''' directory_name '/U_T' num2str(iiT) '.mat'');'];
    eval(expression);
    expression=['load(''' directory_name '/V_T' num2str(iiT) '.mat'');'];
    eval(expression);
    expression=['umesh=u' num2str(iiT) ';'];
    eval(expression);
    expression=['vmesh=v' num2str(iiT) ';'];
    eval(expression);
    expression=['clear u' num2str(iiT) ' v' num2str(iiT) ';'];
    eval(expression);

    clear X Y T
    %%%%%% if the point is inside, calculate it's trajectory
    [X,Y,T]= RungeKutta_f(t0,t1,tstep,sigma_x,sigma_y);
    sigma_x_new=X;
    sigma_y_new=Y;

    for ix=1:nx
        for iy=1:ny
            clear X Y T
            %%%%%% if the point is inside, calculate it's trajectory
            if LeftDomain(ix,iy)==0;

                % of the point is convected out of the domian,
                % calculate the FTLE at this point and adjacent points.
                if (sigma_x_new(ix,iy)-velo_x1)*(sigma_x_new(ix,iy)-velo_x2)>=0
| ...
                        (sigma_y_new(ix,iy)-velo_y1)*(sigma_y_new(ix,iy)-
velo_y2)>=0
```

```
                            LeftDomain(ix,iy)=1;

                            if CalcFTLE(ix,iy)==1

sigma(ix,iy)=calculate_FTLE(ix,iy,sigma_x,sigma_y,T_span);
                                CalcFTLE(ix,iy)=0;

                                if ix>1 & CalcFTLE(ix-1,iy)==1
                                    sigma(ix-1,iy)=calculate_FTLE(ix-
1,iy,sigma_x,sigma_y,T_span);
                                    CalcFTLE(ix-1,iy)=0;
                                end
                                if  ix<nx & CalcFTLE(ix+1,iy)==1

sigma(ix+1,iy)=calculate_FTLE(ix+1,iy,sigma_x,sigma_y,T_span);
                                    CalcFTLE(ix+1,iy)=0;
                                end
                                if  iy>1 & CalcFTLE(ix,iy-1)==1
                                    sigma(ix,iy-1)=calculate_FTLE(ix,iy-
1,sigma_x,sigma_y,T_span);
                                    CalcFTLE(ix,iy-1)=0;
                                end
                                if  iy<ny & CalcFTLE(ix,iy+1)==1

sigma(ix,iy+1)=calculate_FTLE(ix,iy+1,sigma_x,sigma_y,T_span);
                                    CalcFTLE(ix,iy+1)=0;
                                end

                            end
                        end
                        %%%%%% if the point is outside, don't calculate it's trajectory.
keep its position
                    else
                        sigma_x_new(ix,iy)=sigma_x(ix,iy);
                        sigma_y_new(ix,iy)=sigma_y(ix,iy);
                    end
                end
            end

            % update the position for all the points in the domain
            for ix=1:nx
                for iy=1:ny
                    if LeftDomain(ix,iy)==0
                        sigma_x(ix,iy)=sigma_x_new(ix,iy);
                        sigma_y(ix,iy)=sigma_y_new(ix,iy);
                    end
                end
```

```
            end

            c_frame = (['Frame = ',num2str(iT)]);
            c_proc = strcat( 'process accomplished :  ', ...
                num2str( 100*t_integration/t_length,' %03.0f' ), '/100' );
            disp( c_proc );


%             axes(handles.axes1);
%             Z = sigma0';
%             [C,h] = contourf(xmesh,ymesh,Z,10);
%             axis([region_x1 region_x2 region_y1 region_y2]);
%
%             xlabel('X');
%             ylabel('Y');
%             title('FTLE plot');
%             text_output=strvcat(c_frame,c_proc);
%             text(region_x2,region_y2,text_output);
%             pause(0.2);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        end

        % calculate FTLE for all the points inside the domain at the last
        % time step.
        c_proc = strcat( 'process accomplished :  ', ...
            num2str( 100,' %03.0f' ), '/100' );

        for ix=1:nx
            for iy=1:ny
                if CalcFTLE(ix,iy)==1
                    sigma(ix,iy)=calculate_FTLE(ix,iy,sigma_x,sigma_y,T_span);
                    CalcFTLE(ix,iy)=0;
                end
            end
        end

        sigma0=sigma;
        expression=['clear FTLE' num2str(iT)];
        eval(expression);
        expression=['global FTLE' num2str(iT)];
        eval(expression);
        expression=['FTLE' num2str(iT) '=sigma;'];
        eval(expression);

         expression=['save(''' directory_name '/Z_T' num2str(iT) '.mat'');'];
         eval(expression);
```

```
        %     figure
%         axes(handles.axes1);
%         Z = sigma0';
%         [C,h] = contourf(xmesh,ymesh,Z,10);
%         axis([region_x1 region_x2 region_y1 region_y2]);
%
%         xlabel('X');
%         ylabel('Y');
%         title('FTLE plot');
%         text_output=strvcat(c_frame,c_proc);
%         text(region_x2,region_y2,text_output);

    end
end

% % ------------------------------------------------------------------
%%
SaveFTLECalculationMenuItem_Callback

function SaveFTLECalculationMenuItem_Callback()

global sigma directory_name
%[output_file,output_path] = uiputfile('FTLE.txt','Save file name');
output_file = '/FTLE.txt';
output_path = directory_name;
output_FTLE(sigma,output_file,output_path)
%%

%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Function sort_char_array%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [sortedArr,finalIndex] = sort_char_array(orgArr)

% SORTN - Sort cell array or char array in textual and numerical mode.
%               Primary sort by first letter, secondary by second letter and so on.
%               Numbers are considered an numerical values.
%
%               Sorting Rules:
%                           'a' comes before 'b' and so on.
%                           Numbers come before letters
%                           Numbers are compared according to their value
%                           Short string comes before a long one (when first chars are
identical)
%                           Ignore characters that are not letters or
```

```
%                           numbers.
%                           SORTN is NOT case sensitive.
%
%               The main interest in this function is that it can sort
%               numbers with text, so string like 'a60' comes before 'a100'
%
%[SORTED_CELL_ARRAY , INDEXES] = SORTN(CELL_ARRAY) - Sorts the cells array
%                or char array and returns sorted cell array and indexes
%                vector of new arranged cell array.
%
%EXAMPLE:
%               cell_arr = {'abcde54f';
%                                       'aabde54f'}
%
%              [sorted,indexes]=sortn(cell_arr)
%
%               sorted =
%                                   'aabde54f'
%                                   'abcde54f'
%               indexes =
%                                       2
%                                       1
%
%
%
%               cell_arr = {'abcde154f';
%                                       'abcde54f'}
%
%              [sorted,indexes]=sortn(cell_arr)
%
%               sorted =
%
%                                   'abcde54f'
%                                   'abcde154f'
%               indexes =
%
%                                   2
%                                   1
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if ischar(orgArr)   %function works with cell array, so if char input, convert to cell
array
    tempList = lower(orgArr);
    orgArr = cellstr(orgArr);
else
```

```
    tempList = char(lower(orgArr));
end

finalIndex = (1:length(orgArr))';   %create the index vector to be changed
g = {};
for i=1:size(tempList)  %go over all strings and break each one to single letters and
numbers
    tempCellArr = breakString(tempList(i,:));   %break one string from the cell array
    g = AssignCellArray(tempCellArr, g);    %add the breaked string to cell matrix
end
%[g,indexes] = SortByColumn2(g,5);
for i=size(g,2):-1:1    %go over cell matrix of breaked strings and sort is from last
column to the first

    [g,indexes] = SortByColumn(g,i);    %sort the i column

    orgArr = orgArr(indexes);   %update cell array order by the indexes
    finalIndex = finalIndex(indexes);   %update final index order

end

sortedArr =orgArr ;


function [sortedArr,indexes] = SortByColumn(arr,column);
%this function sort each column of the breaked cell-matrix
%sorting is done by converting each textual number to its real value
%and later assign every letter its ascii value plus the maximum of all
%numbers.
%the sorting is pure numeric (Matlab sort) so numbers are always before
%letters.

tempColumn = zeros(size(arr,1),1);
for i=1:length(tempColumn)
    if ~isempty(str2num(char(arr(i,column)))); %finds all numbers including non-real
numbers
        tempColumn(i) = str2num(char(arr(i,column)))+1; %assign numbers to the vactor
    end
end
maxNum = max(tempColumn);
u2 = isletter(char(arr(:,column))); %find all letters
tempColumn(u2) = char(arr(u2,column)) + maxNum; %assign numerical representation of
letters that is bigger then the numbers

[tempSort,indexes] = sort(tempColumn);
sortedArr = arr(indexes,:);
```

```
function newArr = AssignCellArray(assignmentVector, assignTo)
%this function helps adding cell-arrays to the breaked strings cell-matrix


assignToSize = size(assignTo,2);
assignmentVectorSize = length(assignmentVector);
if assignToSize==0
    newArr = assignmentVector;

elseif (assignToSize==assignmentVectorSize)
    newArr = [assignTo;assignmentVector];
elseif (assignToSize>assignmentVectorSize)
    assignmentVector((assignmentVectorSize+1):assignToSize)={' '};
    newArr = [assignTo;assignmentVector];
else
    assignTo(:,(assignToSize+1):assignmentVectorSize)={' '};
    newArr = [assignTo;assignmentVector];
end


function groups = breakString(stringToBreak)
%this function breaks a string to its single letters and numbers,
%producing cell array for a single string.

if ~ischar(stringToBreak)   %convert to char array
    tempString = char(stringToBreak);
else
    tempString=stringToBreak;
end
counter = 1;
i=1;
groups = {};
while i<=length(tempString) %go over all chars in the string
    if  ~isletter(tempString(i)) & isempty(str2num(tempString(i)))   %tempString(i)=='
'   %Skip spaces
        i=i+1;
        continue;
    end
    if isempty(str2num(tempString(i)))  %in case of a letter - just add it to the array
        groups(counter)={tempString(i)};
        counter = counter + 1;
        i = i + 1;
    else
        groups(counter) = {tempString(i)};
        i = i+1;
```

```
        while (i<=length(tempString) & ~isempty(str2num(tempString(i))))    %in case of
number,
            %add it to the array and search for following numbers
            groups(counter)={[char(groups(counter)) tempString(i)]};
            i = i + 1;
        end
        counter = counter + 1;
    end
end
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Function RungeKutta_f%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [X,Y,T]= RungeKutta_f(i_start,i_end,h,xstart,ystart)

% global umesh vmesh

n=1;
t=i_start;
tend=i_end;

delta_i=i_end-i_start;

[ux,uy] = velocity(xstart,ystart,i_start);
x = xstart + delta_i*h*ux;
y = ystart + delta_i*h*uy;

X=x;
Y=y;
T=t+1;


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Function velocity%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ux,uy] = velocity(x,y,iiT)

global xloc yloc umesh vmesh

ux = interp2(xloc,yloc,umesh',x,y,'linear');
uy = interp2(xloc,yloc,vmesh',x,y,'linear');

index=find(isnan(ux)==1);
ux(index)=0;

index=find(isnan(uy)==1);
```

```
uy(index)=0;
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Function calculate_FTLE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function out=calculate_FTLE(ix,iy,flowmap_x,flowmap_y,t_span)

global sxcoor sycoor

[nx ny]=size(flowmap_x);

if (ix-1)*(ix-nx)<0 & (iy-1)*(iy-ny)<0
    A11=(flowmap_x(ix+1,iy)-flowmap_x(ix-1,iy))/(sxcoor(ix+1)-sxcoor(ix-1));
    A12=(flowmap_x(ix,iy+1)-flowmap_x(ix,iy-1))/(sycoor(iy+1)-sycoor(iy-1));
    A21=(flowmap_y(ix+1,iy)-flowmap_y(ix-1,iy))/(sxcoor(ix+1)-sxcoor(ix-1));
    A22=(flowmap_y(ix,iy+1)-flowmap_y(ix,iy-1))/(sycoor(iy+1)-sycoor(iy-1));
    A=[A11 A12;A21 A22];
    B=A'*A;
    delta=max(eig(B));
    out=log(delta)/(2*t_span);
else
    out=0;
end
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Function output%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function out=output_FTLE(sigma,file,path)

global sxcoor sycoor f_start f_end f_int

% f_start=str2double(get(handles.starting_frame,'String'));
% f_end=str2double(get(handles.end_frame,'String'));
% f_int=str2double(get(handles.interval,'String'));

%     expression=['save(''sigma_f'',''sigma'');'];
%     eval(expression);
```

```
[xmesh ymesh]=meshgrid(sxcoor,sycoor);
xloc1=reshape(xmesh(:,:),[],1);
yloc1=reshape(ymesh(:,:),[],1);

filename=[path file];
fid = fopen(filename,'wt');
fprintf(fid,'TITLE = FTLE FIELD¥n');
fprintf(fid,'VARIABLES = X, Y, FTLE¥n');
for iT=f_start:f_int:f_end
    expression=['global FTLE' num2str(iT) ';'];
    eval(expression);
    expression=['sigma0=FTLE' num2str(iT) ';'];
    eval(expression);

    [nx,ny]=size(sigma0);

    % FTLE FIELD
    Z = sigma0';

    Z1=reshape(Z,[],1);
    dummy=0*Z1;
    data=[xloc1 yloc1 Z1];

    fprintf(fid,['ZONE T=''A' num2str(iT) ''' I=' num2str(ny) ', J=' num2str(nx) ',
F=POINT¥n']);
    fprintf(fid,'%12.8f %12.8f %12.8f ¥n',data');
end
fclose(fid);
```