
Simulating the Trajectory for a Directionally Stable Sailboat

Daisy Zhang
dz298@cornell.edu

Guided By - Prof. Andy Ruina
Sibley School of Mechanical and Aerospace Engineering
Cornell University

November 25, 2020

Contents

1	Introduction	3
2	The Sailboat Model	3
2.1	The Parameters of the Boat	3
2.2	Forces and Moments on the Boat	3
2.2.1	Forces and Moments on Sail, Keel and Rudder	4
2.2.2	Force and Moment on Hull	5
2.2.3	Net Force and Moment on the Boat	6
3	The Polar Diagram of the Sailboat	7
3.1	Draw the Polar Diagram with the 2D Model	7
3.1.1	Create the Polar by Solving All Possible Velocities First	7
3.1.2	Create the Polar by fmincon	8
3.2	Other Factors Affecting the Polar Diagram	10
4	Trajectory Simulation in the Dynamic Ocean	13
4.1	Overview of the Trajectory Simulation Program	13
4.2	Real-time Weather Data	14
4.2.1	Ocean Current	15
4.2.2	Ocean Surface Wind	16
4.3	Automatic Data Retrieval	17
4.4	The Boat Velocity	18
5	Future Work	19
5.1	Further Improvement on the Program for Generating Polar Diagram	19
6	Conclusion	20
A	MATLAB Code for Polar Diagram Program	22
A.1	setBoatParam.m	22
A.2	polar_diagram.m	25
A.3	root_finding.m	27
A.4	heeling_angle.m	28
A.5	FM_on_boat.m	29
A.6	rhs.m	31
A.7	C_LD.m	32
A.8	optimize.m	32
B	MATLAB Code for Trajectory Simulation in Ocean	35
B.1	main.m	35
B.2	rhs.m	37
B.3	polar_plot.m	38
B.4	seamotion.m	38
B.5	windmotion.m	40
B.6	stats.m	41
B.7	find_wind_HTTPS.m	42
B.8	find_wind_FTP.m	43
B.9	makescale.m	44
B.10	plot_google_map.m	49
B.11	read_nc_file_struct.m	59

1 Introduction

The goal of our research is to develop a low-cost autonomous sailboat as a new candidate for autonomous monitoring the oceans. Such a sailboat is propelled by wind and powered by solar energy. We minimize the boat's size to reduce its cost. Because the area of solar panels on board scales with the boat size, the reduced boat size limits the available solar power. In addition, if we want the boat to travel in northern Atlantic Ocean, the problem of insufficient power will exacerbate in overcast days. A boat with limited power supply is unable to power sensors and a microprocessor while steering the sail and rudder constantly. A potential solution to save energy is that the boat will only adjust its sail and rudder servos intermittently. For the most of the time, it will charge its battery without steering.

The limited time of steering poses challenges on navigation. With the intermittent steering, it is hard for the boat to follow a desired trajectory. A directionally stable boat could be our solution to this problem. Because such a boat can return to a desired heading relative to wind regardless of interrupt, a constant steering is unnecessary to keep the direction. The design of a directionally stable boat is proposed and its effectiveness is proved in Jesse Miller's report titled 'Directional Stability of Proposed Control Surface Concepts' [6].

The design of a directionally stable boat replaces the traditional water rudder with an airfoil attached to the back of the sail. We denote the mechanism as a tail. The direction that the boat will return to is determined by the boat's sail and tail angle relative to the hull. The boat can change its sail and tail angles to change its desired heading. It can also keep these angles fixed to keep the desired heading. If we build our sailboat as a directionally stable boat, during a ocean monitoring mission, the boat can steer only occasionally to navigate. By the directional stability, the sailboat can sail to its destination and collect data for ocean monitoring while still fulfilling our power budget.

In our research, one of the key problems to solve is how does the sailboat perform in a real-time ocean environment. To know the boat's performance in the dynamic environment, it is crucial to know how the boat performs in still water. We have developed polar diagrams at various wind speeds for the given boat in a MATLAB simulation. Then we could analyze the sailboat's navigation performance by running a trajectory simulation in MATLAB with the generated polar diagram. The simulation can simulate the course of the sailboat in a real-time ocean environment.

In this paper, we present the sailboat model used in our two simulations. Then we detail how we designed and optimized the program to plot the polar diagram. Finally we describe how we developed the simulation program to generate the course of the boat on the ocean.

2 The Sailboat Model

2.1 The Parameters of the Boat

We have created a MATLAB program to generate the polar diagram of the sailboat. A two dimensional model of the sailboat has been developed to run the program. It is similar to the sailboat built by Cornell Autonomous Sailboat Team, which is a prototype of our sailboat. Table 1 lists the values of parameters of the boat.

Figure 1 shows a labeled diagram of the sailboat model. There are four main components consisting the boat: the keel, sail, rudder and hull. Except the hull, all components are treated as airfoils. The rudder is also what we refer to as a tail alternatively. It is attached to the back of the sail to increase the boat's directionally stability. Forces and moments on the four components are explained in details in the subsequent sections.

2.2 Forces and Moments on the Boat

All forces on the two-dimensional boat model are shown in Figure 2. The vectors \vec{i} , \vec{j} and \vec{k} are unit vectors in the fixed inertial frame. Each airfoil on the boat experiences a lift and drag. In the free body diagram, they are labeled as L_{sail} and D_{sail} for sail, L_{rudder} and D_{rudder} for rudder, and L_{keel} and D_{keel} for keel. R_{hull} is the hull resistance on the boat.

The gray dashed arrow labeled "Heading" is the direction the boat points to. The other gray arrow labeled "Course" is the direction the boat travels.

Boat Mass	7 kg
Sail CE Distance in front of COM	0 m
Sail Reference Area	0.2 m ²
Keel CE Distance in front of COM	0 m
Keel Reference Area	0.1 m ²
Rudder CE Distance in front of COM	-0.5m
Rudder Reference Area	0.05 m ²

Table 1: Parameters of the sailboat model. CE is the abbreviation for center of effort. The center of effort is the geometric center of an object. The CE of an airfoil is where the aerodynamic force is located. COM is the abbreviation for center of mass.

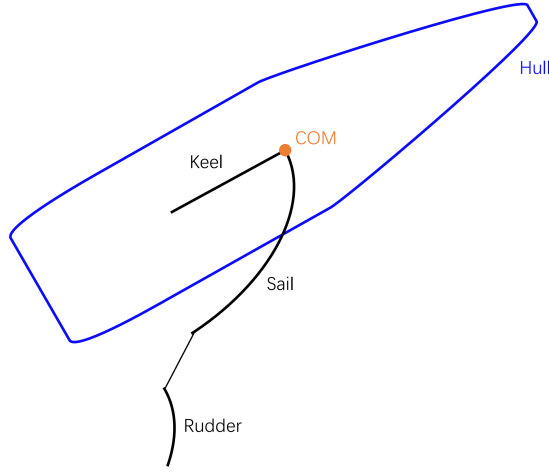


Figure 1: Visualization of the two dimensional sailboat model.

The vectors v_{wind} , v_{boat} , $v_{app.wind}$ represent true wind, boat velocity and apparent wind respectively. True wind is the speed and direction of wind observed by a stationary anemometer. Apparent wind is the speed and direction of wind measured by an anemometer on boat. As is reflected in the free body diagram,

$$v_{app.wind} = v_{wind} - v_{boat} \quad (1)$$

2.2.1 Forces and Moments on Sail, Keel and Rudder

The sail, keel and rudder on the sailboat are modeled as airfoils. The net force exerted on each of the airfoils by the surrounding fluid can be broken down to lift and drag.

$$\vec{F}_{airfoil} = \vec{L} + \vec{D} \quad (2)$$

The lift force can be written as:

$$\vec{L} = \frac{1}{2} C_L \rho A \|\vec{v}\| \vec{k} \times \vec{v} \quad (3)$$

Where C_L is the lift coefficient, ρ is the density of the surrounding fluid of the airfoil, A is the airfoil's reference area from table 1, \vec{v} is the velocity of the airfoil relative to the surrounding fluid and \vec{k} is the unit vector in z-direction. The direction of lift force is aligned with $\vec{k} \times \vec{v}$, which is perpendicular to \vec{v} .

The drag force can be written as:

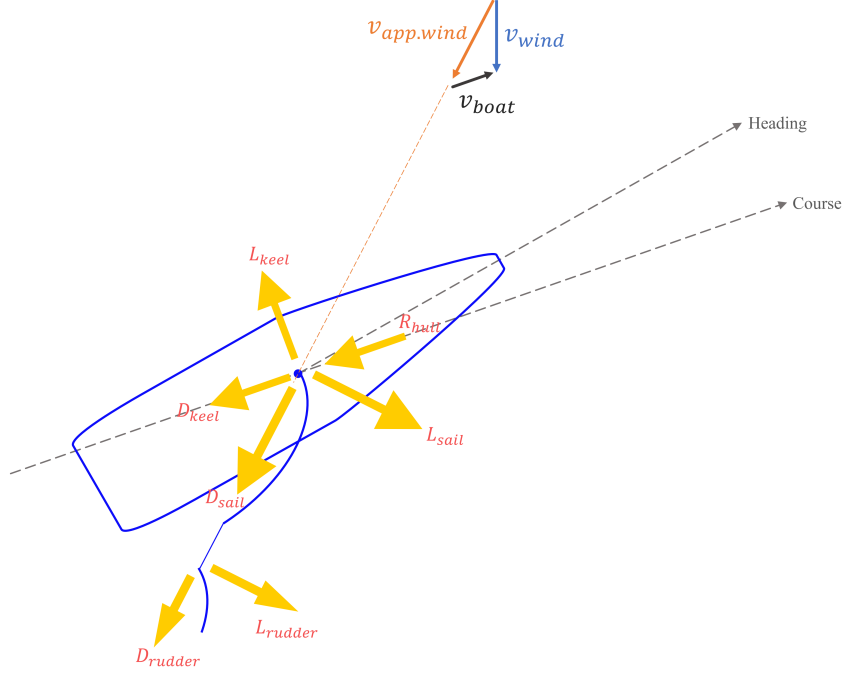


Figure 2: Free body diagram of the sailboat.

$$\vec{D} = -\frac{1}{2}C_D\rho A\|\vec{v}\|\vec{v} \quad (4)$$

Where C_D is the drag coefficient. The drag force is in the opposite direction of \vec{v} .

The values of lift and drag coefficients depend on the angle of attack of the airfoil. Their values are interpolated from data of the lift and drag coefficients of NACA 0015 airfoil[7]. However, interpolation is relatively computationally slow. So for faster calculation, we approximate the lift and drag coefficients as:

$$C_L = C_0 \sin 2\alpha \quad (5)$$

$$C_D = C_p + C_0(1 - \cos(2\alpha)) \quad (6)$$

Where C_p is the parasite drag coefficient of the airfoil, C_0 is constant specified by the developer to match the approximation to real airfoil data as close as possible, and α is the angle of attack of the airfoil. For a better approximation of the NACA 0015 airfoil, $C_p = 0.02$, and $C_0 = 0.9$. [Figure 3](#) compares the lift and drag coefficients of NACA 0015 airfoil and approximation [7]. From the diagram, it can be said that the approximation is very close to real airfoil lift and drag coefficients.

The moment of each airfoil about the center of mass of the boat (COM) is:

$$\vec{M} = \vec{d}_{\text{airfoil to COM}} \times \vec{F} \quad (7)$$

Where the direction of $\vec{d}_{\text{airfoil to COM}}$ is the position of the airfoil in relation to COM, its magnitude can be known from Table 1.

2.2.2 Force and Moment on Hull

Because of the complex shape of the hull, the hull is not modeled as an airfoil. We need experimental data to determine the hull resistance of our boat model. Cornell Autonomous Sailboat Team performed an experiment to determine the dependence of the hull resistance on the boat velocity[6]. The experimental

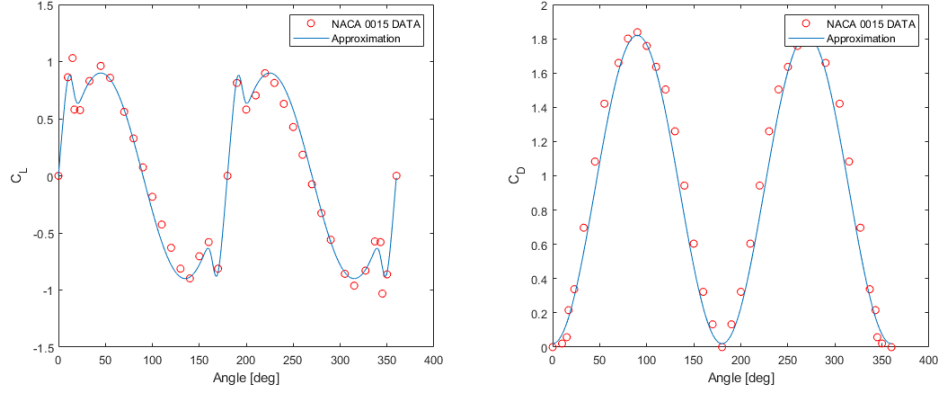


Figure 3: Comparison of lift and drag coefficient approximation with NACA 0015 airfoil coefficients data.

result showed a cubic relation between the hull resistance and the boat velocity. The cubic regression is as follows:

$$\vec{R}_{hull} = -C_{reg} \|\vec{v}_{boat}\|^2 \vec{v}_{boat} \quad (8)$$

Where $C_{reg} = -6.5 \frac{\text{kg}}{\text{m}^2}$ is the regression coefficient, \vec{v}_{boat} is the boat velocity relative to water. We assume that the hull resistance acts at the center of mass of the boat.

From the hull resistance equation, the resistance only depends on the boat velocity. It doesn't take hull shape, water-plane area or other boat parameters into consideration. It is because Cornell Autonomous Sailboat team only used one sailboat model to perform the experiment. The equation is only valid for this model. It cannot be applied to calculate the hull resistance for other boats. Because we use the same model with that used by Cornell Autonomous Sailboat team for our simulation, the equation is enough for our analysis. If we want to optimize the sailboat model to achieve better performance, we need to update the equation so that it can describe the hull resistance universally. The update will be discussed in later section. The hull damping moment is determined in Jesse Miller's report titled 'Directional Stability of Proposed Control Surface Concepts'.

$$\vec{M}_{damp} = -C_{damp} \vec{\omega}_{boat} \quad (9)$$

Where $\vec{\omega}_{boat}$ is the boat's angular velocity and $C_{damp} = 2 \text{ frackg} \cdot \text{m}^2 \text{rad} \cdot \text{s}$.

2.2.3 Net Force and Moment on the Boat

Given the forces and moments determined in section 2.2.1 and section 2.2.2, the net force on the boat and the net moment about the center of mass can be written as:

$$\vec{F}_{net} = \vec{L}_{sail} + \vec{D}_{sail} + \vec{L}_{keel} + \vec{D}_{keel} + \vec{L}_{rudder} + \vec{D}_{rudder} + \vec{R}_{hull} \quad (10)$$

$$\vec{M}_{net} = \vec{M}_{sail} + \vec{M}_{keel} + \vec{M}_{rudder} \quad (11)$$

All the forces on the sailboat are visualized by Figure 2.

3 The Polar Diagram of the Sailboat

3.1 Draw the Polar Diagram with the 2D Model

3.1.1 Create the Polar by Solving All Possible Velocities First

A polar diagram shows a sailboat's potential speed in a range of directions over various wind speed. A polar diagram of a boat is useful to tell how fast the boat can go in a particular direction at a particular wind speed. We want to find the polar diagram for our sailboat model by simulation. One way to create a polar diagram is by calculating all possible velocities of the sailboat given the wind speed, plotting them in a polar coordinate system with the distance to the pole as speed and the angle as the direction, and then connecting all points with maximum speed at angle from 0 to π . The direction the boat travels varies from 0 to 2π but we can halve the range due to symmetry. The resulting curve in the polar axes is the polar diagram of the sailboat at the given wind speed. Note that in drawing the polar diagram, we assume that the boat is in still water so water velocity doesn't affect the boat's performance. This method will be explained in detailed in this section. We have implemented a MATLAB program using this method to generate the polar diagram. The code could be found in [Appendix A](#).

When calculating the sailboat's possible velocities for generating the polar diagram, we are considering a constant velocity motion with zero angular velocity. That's when the boat is at stable equilibrium. At stable equilibrium, the net force and net moment on boat are zero and angular velocity is zero too.

Given the net force and net moment to be zero, we want to solve for sailboat's velocity. There are seven terms in the net force [Equation 10](#): the lift and drag for each airfoil and the hull resistance. We can plug the parameters of the sailboat in [table 1](#) into each of the force equation. After doing this, we find that except the unknown velocity, the lift and drag coefficients in lift and drag force remain unknown as well. The lift and drag coefficients are approximated using [Equation 5](#) and [Equation 6](#). Values of them depend on the attack angle of the airfoil.

Since we are considering the boat at stable equilibrium, its angular velocity is zero. With zero velocity, the attack angle are determined by:

$$\begin{aligned}\vec{v}_{app.airfoil} &= \begin{cases} \vec{v}_{wind} - \vec{v}_{boat} & \text{For sail and rudder} \\ -\vec{v}_{boat} & \text{For keel} \end{cases} \\ \theta_{app.airfoil} &= \arctan \frac{\vec{j} \cdot \vec{v}_{app.airfoil}}{\vec{i} \cdot \vec{v}_{app.airfoil}} \\ &= \begin{cases} \theta_{app.wind} & \text{For sail and rudder} \\ \theta_{app.water} & \text{For keel} \end{cases} \\ \alpha &= \theta_{heading} + \theta_{airfoil} + \theta_{app.airfoil}\end{aligned}\tag{12}$$

Where $\theta_{heading}$ is the boat's heading, $\theta_{airfoil}$ is the airfoil's angle relative to the hull, $\vec{v}_{app.airfoil}$ is the velocity of the airfoil relative to its surrounding fluid. For a better illustration, for example, [Figure 4](#) shows the attack angle of the sail.

Given the attack angle equation, after boat parameters plugged in, the unknown variables in solving the net force and moment equation are the boat's heading, the airfoil angle relative to the hull, the apparent wind angle, and the boat's velocity.

Because the polar diagram is solved at given wind speed, we should specify the wind. Our boat is directionally stable so that its velocity is determined by its sail and tail angles. Therefore we should also specify the sail and tail angles. After these variables are assigned values, the only unknowns left are boat's velocity and boat's heading. Note the boat's heading is where the boat points to; it is different from the course heading which is where the boat sails to. Their difference is well illustrated in [Figure 2](#): the gray dashed arrow labeled Heading is the boat heading, the gray dashed arrow labeled Course is the course heading, in the direction of the boat's velocity.

With the net force and net moment set to zero, [Equation 10](#) and [Equation 11](#) are sufficient and necessary to solve for the boat velocity and the boat's heading.

Our MATLAB program initializes the range of the sail angle relative to the hull and the range of the tail angle relative to the sail from -90 degrees to 90 degrees. It leaves the wind speed to be specified by the user

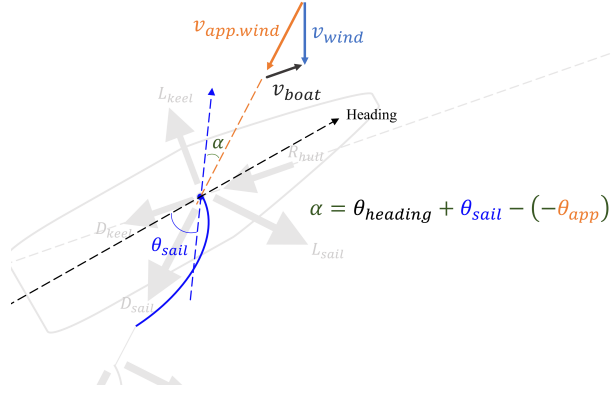


Figure 4: Angle of attack of sail.

while it assumes that the wind is blown from north, or 0 degree direction. The program does not initialize a wind direction range because the polar diagram is the same regardless of the wind direction.

At the beginning of the program, the sailboat is set to all possible combinations of sail and tail angles. For each combination, the velocity of the sailboat at the wind speed is solved by finding the roots of Equation 10 and Equation 11. This solved velocity is plotted as a point in a polar coordinate system with the distance to the pole as speed and the angle as the direction. After the iteration over all combinations of sail and tail angles, all possible velocities of the boat could be plotted. The program thus opens a new figure, plots points with maximum magnitude in each direction to generate the polar diagram.

For instance, when the wind speed is 1 m/s, all possible velocities of the sailboat are shown in Figure 5.a. Among all the possible velocities, the maximum speed in each direction is plotted in a new polar axes, as shown in Figure 5.b. The new figure is the polar diagram of the sailboat for wind speed = 1 m/s.

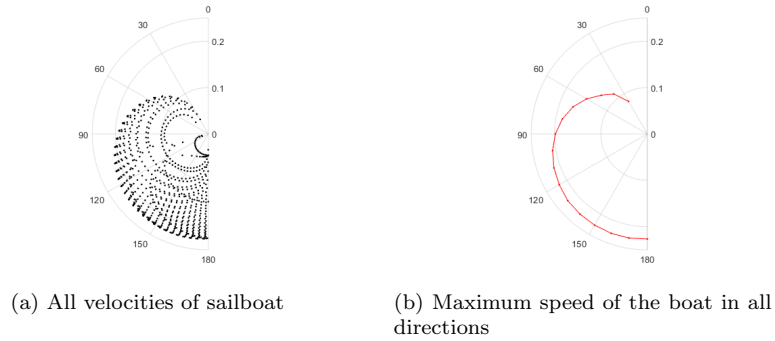


Figure 5: The polar diagram of the sailboat when wind speed is 1 m/s

When the range of the wind speed is expanded to from 10 m/s to 40 m/s, the polar diagram of the sailboat is shown in Figure 6.

3.1.2 Create the Polar by fmincon

The method of calculating all possible velocities first is inefficient and therefore time consuming. Thus We rewrote our MATLAB program so that it used optimization to solve for the polar diagram. The program optimizes the speed of the sailboat in the given direction at a given wind speed. Then, it draws a point in a polar coordinate system to represent the optimization result: the distance to the pole is the maximized speed, and the angle to the pole is the given direction. After iterating over all directions to optimize the speed, the program connects all points in the polar coordinate system. The resulting curve is the polar diagram of the sailboat at the given wind speed.

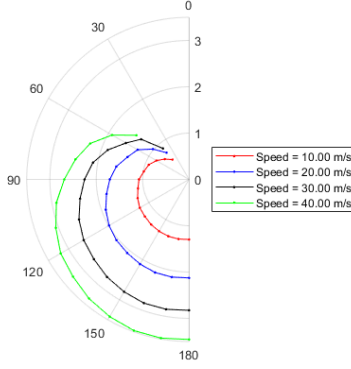


Figure 6: The polar diagram of the boat over the wind speed from 10 m/s to 40 m/s.

We used the MATLAB function `fmincon` to optimize the boat's speed. `fmincon` is a MATLAB function for solving non-linear optimization. It is used to find minimum of constrained nonlinear multivariable function. The function with constraints is specified by:

$$\min_{\vec{x}} f(\vec{x}) \text{ so that } \begin{cases} c(\vec{x}) \leq 0, \\ ceq(\vec{x}) = 0, \\ A \cdot \vec{x} \leq b, \\ Aeq \cdot \vec{x} = beq, \\ lb \leq \vec{x} \leq ub \end{cases} \quad (13)$$

We can then apply `fmincon` to our problem. \vec{x} is set to be a vector of length 2. It contains the sail angle relative to the hull and the tail angle of the sailboat relative to the sail. The program assumes the range of the two angles from -90 degrees to 90 degrees so lb is set to $-\frac{\pi}{2}$ and ub is set to $\frac{\pi}{2}$.

The objective function f is the negative magnitude of the boat velocity at the sail and rudder angle, because solving for the minimum of negative velocity magnitude is equivalent to solving the maximum speed. The net force Equation 10 and the net moment Equation 11 are used to construct f . As we discussed in section 3.1.1, the two equations are sufficient and necessary to solve for the boat's velocity with given sail and tail angles and wind speed. So with wind speed given, the two equations can be rewritten to be the boat's velocity magnitude function of sail and tail angles, which is our objective function.

For the constraints, A , Aeq and c are not specified because there's no linear constraints or inequality constraints in our case. As for ceq , the non-linear equality constraint function:

$$ceq = \arctan \frac{\vec{j} \cdot \vec{v}_{boat}}{\vec{i} \cdot \vec{v}_{boat}} - dir \quad (14)$$

Where dir is the given direction of the speed to be optimized, \vec{v}_{boat} is the boat's velocity, which is a function of sail and tail angles, \vec{x} . If the minimum of the function f is solved with dir varying from 0 to π , then the maximum speed of the boat in all directions, along with the corresponding sail and rudder angles can be solved.

To conclude, the optimization problem of how to solve for the polar diagram can be specified as:

$$\begin{aligned} & \forall dir \in [0, \pi], \\ & \min_{\text{sail angle, tail angle}} -\|\vec{v}_{boat}\| \text{ so that } \begin{cases} \arctan \frac{\vec{j} \cdot \vec{v}_{boat}}{\vec{i} \cdot \vec{v}_{boat}} - dir = 0, \\ -\frac{\pi}{2} \leq \text{sail angle} \leq \frac{\pi}{2}, \\ -\frac{\pi}{2} \leq \text{tail angle} \leq \frac{\pi}{2} \end{cases} \end{aligned} \quad (15)$$

With the `fmincon` method, the polar diagram of the sailboat with wind speed from 10 m/s to 40 m/s can be solved, as shown in Figure 7.

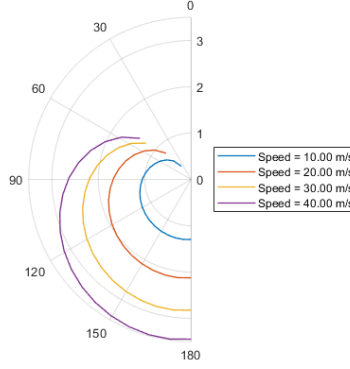


Figure 7: Solved by `fmincon`, the polar diagram of the boat over the wind speed from 10 m/s to 40 m/s.

We can prove the validity of this new method by overlapping the polar diagram over all possible velocities calculated and drawn by the method in section 3.1.1. Figure 8 shows the comparison. Points represent all possible velocities of the sailboat while curves are the same curves in Figure 7, the polar diagram solved by `fmincon`. At the same wind speed, points are in the same color with the curve. For example, when wind speed is 40 m/s, the curve associated with wind speed of 40 m/s is purple, and all points representing the boat's possible velocities at wind speed of 40 m/s. At every wind speed, all points representing possible sailboat velocities fall within or at the curve of polar diagram defined in Figure 7. Therefore, we can say that the new method of using `fmincon` is reliable on solving for the polar diagram.

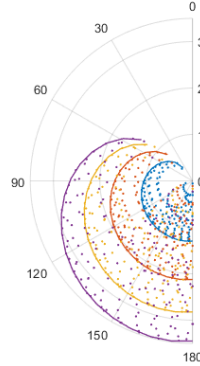


Figure 8: All possible velocities overlap over the polar solved by `fmincon` to attest the method's validity. Note that at the same wind speed, dots representing possible velocities are in the same color as the curve.

3.2 Other Factors Affecting the Polar Diagram

So far our force and moment analysis are only two dimensional. The yaw motion of our sailboat is considered in our force and moment analysis, but the pitch and roll of the sailboat have not yet taken into account in calculations.

The boat's pitch motion can be safely ignored in our analysis because our sailboat is designed to be long and narrow. The shape of the boat makes it have a large moment of inertia about its transverse axis so that the boat naturally resists rotational acceleration about this axis which causes pitch.

However, the boat's rolling, or heeling in other words, can only be ignored when wind speed is small. The moment caused by wind is small enough so that the boat's heeling does not affect its performance. If wind speed continues to rise and the resultant moment increases, at some sail and tail angles, the boat will sail at a significant heeling angle that cannot be ignored in impacting sailboat's performance or even the vessel's

safety. The excessive heeling will cause the boat broach or capsize, putting the vessel in danger. Therefore, the sailboat should never sail with the sail and tail angles which can cause excessive heeling. When we design the program to create the boat's polar diagram described in section 3.1, there should be one more constraint being considered: the sailboat's heeling angle should be less than 30 degrees.

The boat's heeling angle can be calculated through force and moment analysis on the boat. Figure 9 is the free body diagram of the sailboat at its stern view. F_a is the total aerodynamic force on the sail and tail; F_h is the total hydrodynamic force on the keel; w is the boat's weight and F_b is the buoyancy force. A and B are the center of effort of sail and keel respectively; G is the center of gravity of the boat; and BC is the buoyancy center. We assume the aerodynamic force on the sail and that on the tail act at the same vertical position on the boat, so F_a representing the sum of these two forces acts at point A .

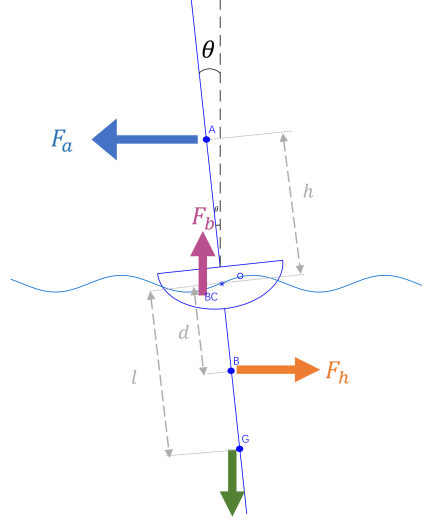


Figure 9: Free body diagram of the heeling sailboat.

The force and moment on the hull are ignored, and so do the hull's stability. Since the boat's center of mass is on the very long keel shown in Figure 9, the metacentric height of the boat, which is a measurement of the static stability of the boat, is very large no matter what heeling angle is. The change of metacentric height due to heeling does not have as much impact on our boat's stability as the boat with a shorter or no keel. Therefore in the calculation of heeling angle, the effect of hull stability can be discounted. Since we ignore forces on hull and the hull stability, we assume the buoyancy center rarely shifts when the boat heels. So the moment of buoyancy force about point O is assumed to be 0.

Based on our assumptions and the boat's free body diagram at its stern view, the net angular moment about point O can be written as:

$$M_{/O} = -F_a h \cos \theta - F_h d \cos \theta + w l \sin \theta \quad (16)$$

Where h is the vertical distance from O to A where F_a applies, d is the vertical distance from O to B where F_h applies, w is the boat's weight, l is the vertical distance from O to G where w applies, θ is the heeling angle to be solved.

Because we are considering a constraint of heeling angle for solving the polar diagram, and stable equilibrium is our assumption for solving for the polar diagram, we should also assume the boat is at stable equilibrium when we solve for the heeling angle. So we need to set the net moment about point O to be zero, that is Equation 16 equal to zero. The heeling angle of the sailboat can be solved.

$$\theta = \arctan \frac{F_a h + F_h d}{w l} \quad (17)$$

Where F_a is the magnitude of the sum of lift and drag of the sail and the tail of the boat, F_b is the magnitude of the sum of lift and drag of the keel. The two terms can be known when the net force equation is solved

for generating the polar diagram. d , w and l can be directly gained from the boat's parameters. So when we solve for the boat's polar diagram, we can also solve for possible heeling angles of the boat. The heeling effect on the boat's polar can be best observed when we compare all possible velocities of the sailboat with heeling effect considered or not considered. If heeling constraint is taken into account, all velocities associated with heeling angles larger than 30 degrees are eliminated from the polar. For instance, [Figure 10.a](#) shows all possible velocities of the boat without the effect of heeling at wind speed of 40 m/s and [Figure 10.b](#) shows all possible velocities with the effect of heeling taken into account at the same wind speed. With a high wind speed, the significance of heeling on boat is remarkably large.

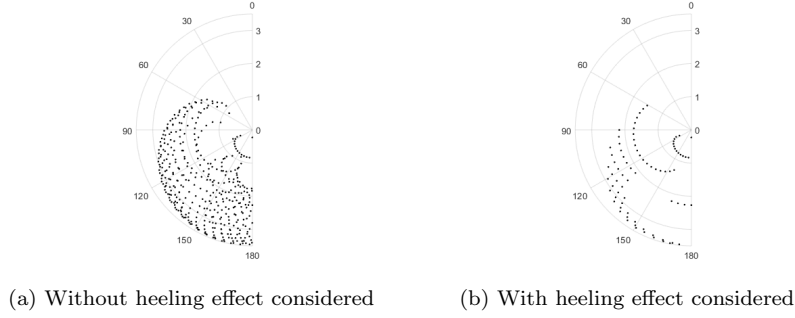


Figure 10: All velocities of sailboat when wind speed = 40 m/s without and with heeling effect considered

When the heeling constraint is taken into consideration, we can redefine the objective function with constraints in section 3.1.2 as:

$$\begin{aligned} & \forall dir \in [0, \pi], \\ & \min_{\text{sail angle, tail angle}} -\|\vec{v}_{boat}\| \text{ so that } \begin{cases} \theta = \arctan \frac{F_a h + F_h d}{wl} \leq \frac{\pi}{6} \\ \arctan \frac{\vec{i} \cdot \vec{v}_{boat}}{\vec{j} \cdot \vec{v}_{boat}} - dir = 0, \\ -\frac{\pi}{2} \leq \text{sail angle} \leq \frac{\pi}{2}, \\ -\frac{\pi}{2} \leq \text{tail angle} \leq \frac{\pi}{2} \end{cases} \end{aligned} \quad (18)$$

We apply `fmincon` to solve the problem. The polar diagram of the sailboat can be drawn as in [Figure 11](#). To verify its validity, all possible velocities with different wind speed are plotted again on the same axes as the polar diagram, as shown in [Figure 12](#).

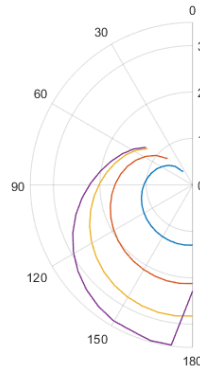


Figure 11: The polar diagram of the boat over the wind speed from 10 m/s to 40 m/s, with heeling effect taken into account.

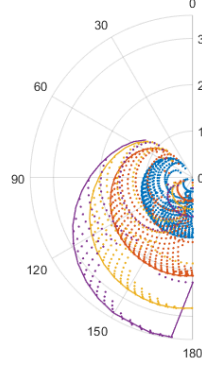


Figure 12: The polar diagram of the boat along with all possible velocities of the boat from wind speed to 10 m/s to 40 m/s.

4 Trajectory Simulation in the Dynamic Ocean

4.1 Overview of the Trajectory Simulation Program

From polar diagrams for our sailboat, we are able to know the boat's sailing performance in still water. Its performance in the ocean with moving water should also be evaluated because the boat is expected to monitor the oceans. We developed a MATLAB program to simulate its trajectory in Atlantic Ocean. The program code could be found in Appendix B.

User inputs to this program are coordinates of the start, coordinates of the destination, the beginning time of travel, and the time span of the travel. For example, we want to know the boat's course if we launch the boat at a port of New York City on January, 1, 2020 at 8 am and let it sail with no human intervention for one month. Our inputs to the program to generate the trajectory is the coordinates of the port in New York City, the coordinates of the place where we want the boat to arrive, January, 1, 2020 at 8am, and one month as the travel period. This program uses information provided in user inputs to obtain the weather data at the beginning location at the beginning time of travel, then based on the weather data to calculate our boat's initial velocity.

The boat's velocity in our program is the boat's velocity relative to ground. It is equal to the sum of the boat's velocity relative to water and the water velocity relative to ground. The water velocity is the velocity of ocean current. Our program can access to its value from ocean weather data. The boat's velocity relative to water is equivalent to the boat's velocity if the water is still. It can be known from the sailboat's polar diagram given the surface wind and the boat's intended direction. Our program can access to wind data from ocean weather data, like it does with ocean current data.

The boat's velocity could be written as:

$$\vec{v}_{boat} = \vec{v}_{current} + f(\vec{v}_{wind}, \theta_{course}) \quad (19)$$

Where $\vec{v}_{current}$ and \vec{v}_{wind} are the velocity of the ocean current and wind at the boat's position, θ_{course} is the boat's intended course heading. f is the function describing the velocity dependence of wind speed and course heading, extrapolated from the polar diagram of the sailboat. The boat's polar diagram can be drawn by methods described in section 3.

If weather remains unvaried so the boat keeps the calculated velocity during a certain period, we can predict the boat's future location after this period. This program thus can obtain the weather data at the predicted location at the time, calculate the boat's velocity and predict a new future location. The program continues to do so until the boat has collided with a land, the boat has arrived the destination or the user defined travel period has passed. The program connects all predicted locations one by one, the resulting zigzagged line is the simulated trajectory. For example, Figure 13 shows the simulated trajectory of the sailboat from January 1, 2019 to March 1, 2019. In this simulation, the boat always sails towards its destination regardless of ocean current effect.



Figure 13: A simulated trajectory of the boat model.

In this program, we assume weather stays unchanged and so do the boat's velocity during a certain period, so we can use the boat's velocity to predict the boat's future location. The position of the boat can be written as:

$$\vec{p}_{new} = \vec{p}_{old} + \int_T^{T+\delta T} \vec{v}_{boat} dt \quad (20)$$

Where T is the current time, δT is the period during which weather is unchanged, \vec{v}_{boat} is the boat's velocity from time T to $T + \delta T$, \vec{p}_{new} is the location of the boat at time $T + \delta T$ and \vec{p}_{old} is the location of the boat at time T .

This assumption will certainly fail if the period is large, such as one day. If the period is set very small such as one minute, our assumption will be very close to the real situation because weather changes every instant even at a infinitely small degree. Also due to the small period, each section of the zigzagged trajectory will become so small that the zigzagged trajectory appears to be a smooth curve like the boat's trajectory in reality. However, a very small period will make our program computationally slow. Therefore, we need to choose an appropriate period δT to approximate the real situation also to make our program run faster. We let δT to be one hour. The decision will be justified in subsequent section.

4.2 Real-time Weather Data

Our program needs ocean current data and ocean surface wind data to calculate the boat's velocity relative to ground, which is necessary to generate the trajectory of the boat. Therefore, our program should be able to fetch at least these two sets of data.

4.2.1 Ocean Current

The source of the ocean current data for our program is a product from Copernicus Marine environment monitoring service (CMEMS) Ifremer Centre [1]. This product includes various measurements such as hourly-mean water temperature, salinity, current from the top of the ocean to its bottom over the global ocean. The depth of these measurements varies from 0.5 meter below the ocean surface to 5500.0 meters below the surface. For our program, we only need ocean current data at a depth where the water flow through the boat's hull and keel. If the depth of the ocean current measurement is either too shallow or too deep, the ocean current at that depth will not flow along any parts of the boat so it cannot apply any force to the boat to affect its velocity. Our boat is small. The height of our boat, from the top of the sail to the bottom of the keel, will not exceed 2 meters. Therefore a depth of 0.5 meter of ocean current measurements is enough. The CMEMS product outputs data files in NetCDF format. We used MATLAB's `ncread` function to read data in the NetCDF file and store data in a struct variable that our program can access to.

Temporal resolution refers to the discrete resolution of a measurement regarding time. There are four temporal resolutions of ocean current data including hourly-mean, daily-mean, monthly-mean, and 6-hourly-instantaneous. Each indicates that the temporal resolution of data is one hour, one day, one month and six hours respectively. Hourly-mean, daily-mean and monthly-mean mean that data with each temporal resolution are the mean of all measurement taken during every hour, day or month, while 6-hourly-instantaneous data are measurements taken every six hours without filtering. We choose hourly-mean temporal resolution for our ocean current data. Since the temporal resolution of ocean current data is one hour, it is reasonable to choose one hour as the period δT , during which weather stays unchanged.

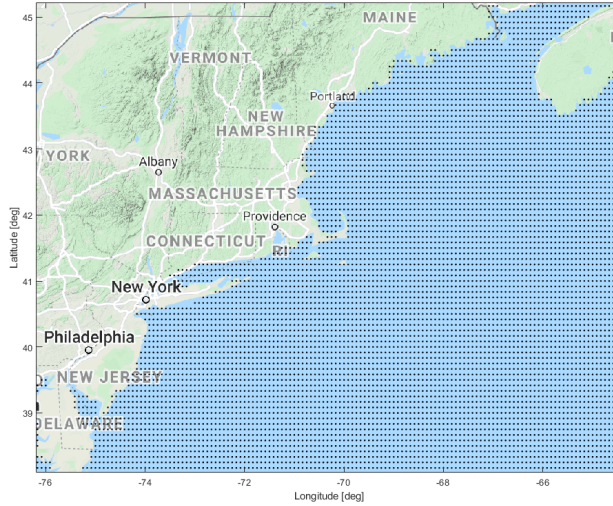


Figure 14: Ocean current data points in the equirectangular projection.

Spatial resolution is the linear spacing of a measurement. The spatial resolution of ocean current data is 0.083 longitude degree by 0.083 latitude degree. For better visualization, part of data points is displayed in black dots on the map in the equirectangular projection in Figure 14. The equirectangular projection maps meridians to vertical straight lines of equal spacing and latitude circles to horizontal straight lines of equal spacing. From this figure, there are constant linear spacing separating data points from each other. The distance of the spacing is 0.083 degree, the spatial resolution of the data set.

The temporal resolution and spatial resolution of data make all data points discrete. However, in our program, we want to know ocean current measurements at given location at given time while for the most of the time, the location and time fall outside the coverage of the discrete data set. For example, we want to fetch a ocean current measurement at x degree longitude and y degree latitude at t time, while none of the data points in our ocean current data set is measured at the same location and at the same time. One way to solve the problem of missing matched data points is that we will use interpolation to construct new

data points at the given location and time from known data points.

This interpolation is a multivariate interpolation in three dimensions. The ocean current data point is given by its latitude, longitude and time. These three variables consist of the three dimensions of the interpolation. We will first use the nearest-neighbor interpolation on time to reduce three dimensions to two. That is, we construct the data point at (x, y) coordinates at the given time t from known data points at the closest time to t in the original data set. Then we use bilinear interpolation to construct the data point at given coordinates. It is performed by using linear interpolation in longitude direction first, then using linear interpolation again in latitude direction. The bilinear interpolation can be better illustrated by [Figure 15](#) and [Equation 21](#).

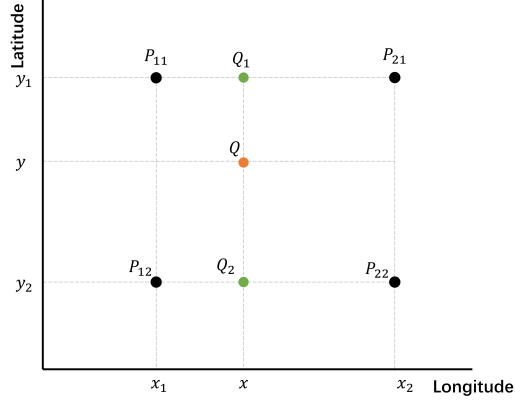


Figure 15: The four black dots show the known data points and the orange dot is the point at which we want to interpolate..

$$\begin{aligned}
 f(Q_1) &\approx \frac{x - x_1}{x_2 - x_1} f(P_{11}) + \frac{x_2 - x}{x_2 - x_1} f(P_{21}) \\
 f(Q_2) &\approx \frac{x - x_1}{x_2 - x_1} f(P_{12}) + \frac{x_2 - x}{x_2 - x_1} f(P_{22}) \\
 f(Q) &\approx \frac{y - y_2}{y_1 - y_2} f(Q_2) + \frac{y_1 - y}{y_1 - y_2} f(Q_1)
 \end{aligned} \tag{21}$$

The four black dots P_{11} , P_{12} , P_{21} , P_{22} represent the known data points; the two green dots Q_1 and Q_2 are constructed data points after first interpolation in longitude direction; the orange dot Q is the data point at which we want to interpolate. For every point, f is the ocean current velocity at that data point.

We use linear interpolation in longitude direction and latitude direction because we assume these four data points are on the same plane. Our assumption is different from the reality because these four data points are on the Earth's curved surface. However, despite of the difference, our assumption still holds. These four data points are the four closest points to the point at which we want to interpolate. Because the spatial resolution of the data set is 0.083 degree by 0.083 degree, $y_1 - y_2$ and $x_2 - x_1$ are also 0.083 degree latitude and 0.083 longitude respectively. The 0.083 degree by 0.083 degree grid is so small compared to the globe that its curvature is negligible. Therefore, we can safely make our assumption and do linear interpolation on these data points.

In this way, we successfully approximate the ocean current data at the given coordinates at the given time from known data points. We can use the interpolated ocean current velocity to calculate the boat's velocity in our program.

4.2.2 Ocean Surface Wind

The source of wind data for our program is also a product from CMEMS Ifremer Centre [2]. The spatial resolution of the data set is 0.25 longitude degree by 0.25 latitude degree. Its temporal resolution is 6-hourly-mean. Again, because the temporal resolution is six hours, it's reasonable to set our period during which

weather stays unchanged to be one hour. We want to know wind measurements at given coordinates at given time. Therefore, like how we obtain ocean current measurements, we use nearest-neighbor interpolation in time direction on known data points, then use linear interpolation in longitude and latitude directions to construct the data point at the given coordinates and given time.

There is only one vertical level of wind measurements in this data set, which is at 10 meters above the ocean surface. The height of our sailboat above water is around 1 meter. The height of wind measurements is too high above the top of our boat's sail. After we perform interpolations to gain the wind measurement at the given location and given time, the interpolated data point is also at the same vertical height as other known data points. We cannot use the boat's speed obtained at this wind speed measurement in our simulation. It is because our boat doesn't experience the same wind as the wind at 10 meters above the surface. The distribution of the wind speed in vertical direction makes the wind speed at 10 meters above the ocean surface differ tremendously from that at the surface level. Therefore, the wind profile power law is applied here to extrapolate the wind speed at the boat's height:

$$u = u_r \left(\frac{z}{z_r} \right)^\alpha \quad (22)$$

Where u is the wind speed to be extrapolated, z is the height of u , u_r is the reference wind speed, z_r is the height of u_r , α is an empirically derived coefficient. Here, we want to know the wind speed at the boat's height based on the known wind measurements. So z will be the height of the boat, z_r will be 10 m, and u_r is the wind speed from the data set. $\alpha = 0.11$ because our boat is at sea [5].

With Equation 22, we can know the wind speed at the boat's height and then gain the boat's speed from the boat's polar diagram in the intended direction at this wind speed.

4.3 Automatic Data Retrieval

We hope our simulation can achieve automatic data retrieval such that when data are not available in the local directory, this program can autonomously connect to the web server to download necessary data set and process it. Automatic data retrieval will be important if we want to use our simulation to aid the boat's navigation. With this feature, if the boat is launched for a mission, our program is able to fetch latest real-time weather data or latest weather forecast from online data set and use them to predict the boat's trajectory. The trajectory prediction can help the boat make decisions in navigation.

We need ocean current measurements and surface wind measurements. These two measurements' data sets are both from CMEMS Ifremer Centre. Automated data extraction from CMEMS Ifremer Centre can be achieved by two methods. Our program can either automatically download dataset files via the FTP server or with subsetting.

FTP servers are to facilitate file transfers across the internet. The CMEMS's FTP server contains all of its data products in the form of NetCDF files. The permission to download files from the CMEMS's FTP server can be granted by access to the server with Copernicus Marine Service account. MATLAB provides the `ftp` function to allow a program to connect to an FTP server, login a FTP account on the server, navigate to different folders on the server, then upload and download files. By calling the function, our program is able to connect to the CMEMS's FTP server, navigate to the folder containing the desired data and download them to the local computer. Details of implementing automatic data retrieval via FTP server are in B.8. However, files downloaded via the FTP server are unfiltered. These files contain all measurements of the product, and cover all the geographical area and vertical levels that the product covers. For example, even though we only want the ocean current measurement at the depth of 0.5 m at specified coordinates, if the file containing the desired data is downloaded via the FTP server, it also contains other measurements, such as water temperature and salinity, at various vertical level and across the entire ocean. The unfiltered files are large in size and thus can result in a large waste of time and space for downloading and storing files.

Subsetting is the other way to download data files from CMEMS. The CMEMS web portal generates a template command line for users to download data that they are interested in, which can be executed from the shell of user's operating system. The template command is as below, for instance, which is downloading ocean current measurements at the depth of 0.5 m on the day of December, 3rd of 2020 across the entire ocean:

```
python -m motuclient --motu http://nrt.cmems-du.eu/motu-web/Motu --service-id
```

```
GLOBAL_ANALYSIS_FORECAST_PHY_001_024-TDS --product-id global-analysis-forecas
t-phy-001-024 --longitude-min -180 --longitude-max 179.9166717529297 --latitu
de-min -80 --latitude-max 90 --date-min "2020-12-03 00:00:00" --date-max "202
0-12-04 00:00:00" --depth-min 0.493 --depth-max 0.4942 --variable uo --variab
le vo --out-dir <OUTPUT_DIRECTORY> --out-name <OUTPUT_FILENAME> --user <USERN
AME> --pwd <PASSWORD>
```

The full instructions on how to use subsetting to download CMEMS products can be found in [this page](#). Subsetting is achieved with the help of a Python package called "motuclient" [3]. This method is able to filter out unnecessary variables. It can also filter data sets with geospatial, vertical and temporal criterias. It is particularly useful in automating data extraction from data product of large volumes. Our program runs in MATLAB while this command uses Python and gets executed in the shell. Therefore, in our implementation, the program calls `system(command)`, the MATLAB function to start a new shell process and execute `command`, which is the template command above, in the shell process and return to the MATLAB program. Details of implementation are in [B.7](#).

4.4 The Boat Velocity

The boat's velocity is the sum of the boat's velocity relative to water and the ocean current velocity. One is read from the boat's polar diagram and the other is gained from the data set containing the ocean current measurements. The unit of each velocity is in meters per second. A unit conversion is necessary, because the resulting boat velocity is used to extrapolate the boat's future coordinates by [Equation 20](#), while coordinates use angular units: degrees, minutes and seconds.

Meanwhile, the resulting boat's velocity is in the local tangent plane which is defined by one local eastern axis tangent to the circle of the boat's latitude, one local northern axis tangent to the boat's meridian and one vertical axis normal to the surface. For trajectory calculation, the velocity needs to be converted from the local tangent plane coordinates to the Earth's coordinates. A sketch of the Earth's coordinate system and the local tangent plane is shown in [Figure 16](#). Let the boat be located at point Q . Its velocity is the sum of y' along the north axis and x' along the south axis of the local tangent plane. The velocity needs to be converted from the local tangent plane to the Earth's coordinates system P_2 .

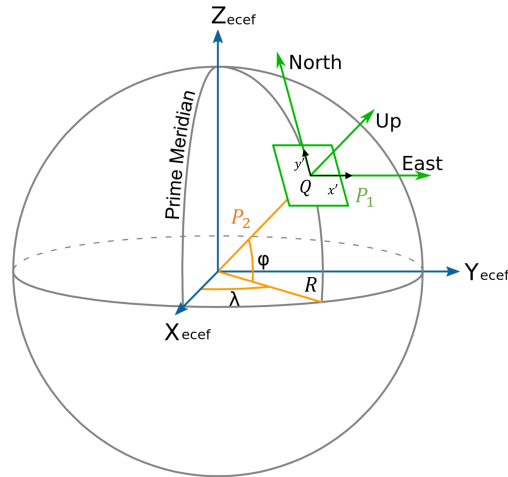


Figure 16: Sketch of the Earth's coordinate system along with the local tangential plane, from [8].

I assume the Earth is spherical so that the conversion happens from the local tangent plane to a spherical coordinate system, the radius of which is the Earth's radius R .

Let the velocity in the Earth's coordinates be (λ', φ', R') and let the boat's location be $Q = (\lambda, \varphi, R)$. Since the boat travels on the surface of the ocean, R' is always 0.

The local tangent coordinates and the global Earth's coordinates can be related with the following formula.

$$\begin{aligned} x' &= \sin(\lambda')R \cos(\varphi) \\ y' &= R \sin(\varphi') \end{aligned} \quad (23)$$

Because x' and y' are so small compared to the Earth's radius, λ' and φ' are small enough to be eligible for the small-angle approximation. Therefore, Equation 23 can be rewritten as:

$$\begin{aligned} x' &= \lambda' R \cos(\varphi) \\ y' &= R \varphi' \end{aligned} \quad (24)$$

We thus can convert the velocity of the boat in local plane to the global coordinate system.

5 Future Work

Now we have developed a program to generate our boat's polar diagram, and a program to simulate its trajectory in the ocean. These two programs can evaluate the boat's performance in still water and dynamic oceans. However, these programs have aspects to improve for a more robust performance evaluation.

The current trajectory simulation can be made more robust. For example, so far the simulation has only considered the effects of surface wind and ocean current on the sailboat. The effects of waves on the sailboat can be added to the simulation. Ocean waves may have a large impact on the directionally stability of the sailboat and should be included in the simulation.

In addition, the program for generating polar diagrams can be made more applicable for various boat models.

5.1 Further Improvement on the Program for Generating Polar Diagram

The program for generating polar diagrams of the given boat uses net force and net moment equations to solve for polar diagrams. The hull force on the boat is described by Equation 8 relating the hull force to the boat's velocity, derived by the Cornell Autonomous Sailboat Team. However, the derived relation is only applicable to the hull of the sailboat built by the team, because the experiment to derive the force equation is only done on that hull. Our sailboat may be optimized for better performance. After the optimization, the sailboat's parameters will change significantly so that the optimized sailboat model will not be similar to the sailboat built by Cornell Autonomous Sailboat Team. The hull force described by Equation 8 will also no longer be appropriate for measuring the hull resistance. The hull force equation has to be changed so that it can universally describe hull resistance on all kinds of hull forms, but not only limited to one.

Therefore, according to Gerritsma's equations, the updated hull resistance can be written as [4]:

$$R_{hull} = R_F + R_R + R_I + R_H \quad (25)$$

Where R_F is frictional resistance induced by the fluid on the hull, R_R is residual resistance mainly due to the waves created by the shape of the hull, R_I is the resistance induced by leeway, and R_H is the heeling induced resistance. Given the sailboat's hull shape, the four resistances can be computed using the Gerritsma's equations.

In Gerritsma's equation, R_F is expressed as:

$$\begin{aligned} R_F &= \frac{1}{2} C_F \rho v^2 A \\ C_F &= \frac{0.075}{(\log R_n - 2)^2} \end{aligned} \quad (26)$$

Where ρ is water density, v is the hull velocity, A is the wetted surface of the hull, R_n is the Reynolds number for the hull. C_F is the frictional resistance coefficient.

While R_R is computed by:

$$\begin{aligned}
R_R = & mg(a_0 + a_1 C_p + a_2 (LCB) + a_3 \frac{B_{wl}}{T_c} + a_4 \frac{L_{wl}}{V^{\frac{1}{3}}}) \\
& + a_5 C_p^2 + a_6 C_p \left(\frac{L_{wl}}{V^{\frac{1}{3}}} \right) + a_7 (LCB)^2 \\
& + a_8 \left(\frac{L_{wl}}{V^{\frac{1}{3}}} \right)^2 + a_9 \left(\frac{L_{wl}}{V^{\frac{1}{3}}} \right)^3
\end{aligned} \tag{27}$$

Where C_p is the prismatic coefficient, LCB is the longitudinal center of buoyancy, B_{wl} is the waterline width, L_{wl} is the waterline length, T_c is the draught of the hull, V is the submerged volume of the hull. The coefficients a_0 to a_9 are non-constant variables in the equation, given for a Froude number $F_r = \frac{v}{\sqrt{gL_{wl}}}$ within the range from 0.125 to 0.450.

The complex equation for R_R makes it take much longer time to compute. It is worth noting that R_R can be simplified to the similar form as R_F . The value of R_R is given by the Froude number for the hull. Since v is the only non-constant variable in F_r , we can say that the value of R_R is given by the hull speed v , the same as R_F . Therefore, we can simplify the equation for R_R so that its form is the same as that of R_F , and C_F and C_R are both given by the Froude number for the hull, as shown in Figure 17.

$$R_R = \frac{1}{2} C_R \rho v^2 A \tag{28}$$

Where C_R is the residual resistance coefficient.

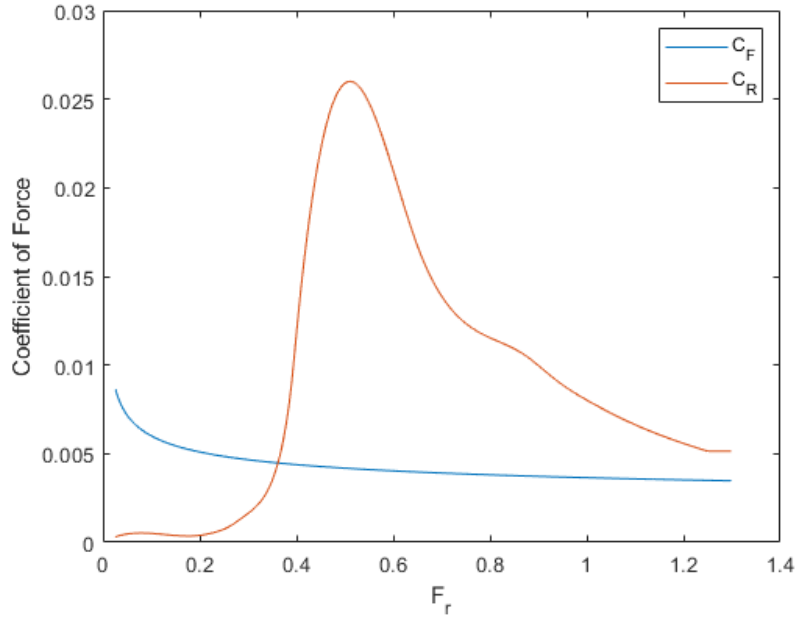


Figure 17: Frictional resistance coefficient and residual resistance coefficient versus Froude number.

Our program for generating polar diagrams can use the new equation Equation 25 to calculate the hull force on the boat and this force equation applies to various hull shapes.

6 Conclusion

We have created two programs to evaluate the performance of a directionally stability sailboat. We have developed a program to generate the boat's polar diagram. The `fmincon` MATLAB optimization function was used to solve for the sailboat's maximum speed in the various directions at the given wind speed, with

the constraint that the boat should never lean more than 30 degrees. The curve connecting data points representing the boat's maximum speed in each directions is the polar diagram of the boat. This program can be used to evaluate the boat's performance in still water. We have also developed a program to simulate the boat's trajectory in the ocean, given initial launch time and location. This program can be used to evaluate the boat's performance in dynamic oceans, as well as investigate the effects of wind and ocean current on the boat's navigation. The effects of waves on the boat can be included in the simulation for a more robust simulation, because waves may have a strong impact on the boat's directional stability. If we want to make a physical boat, these programs can inform us critical design decisions in the making of the boat. We should improve them so that these programs can encapsulate more realistic environment parameters and more comprehensive models.

References

- [1] CMEMS Ifremer Centre. *GLOBAL OCEAN 1/12 PHYSICS ANALYSIS AND FORECAST UPDATED DAILY*. 2016. URL: https://resources.marine.copernicus.eu/?option=com_csw&task=results?option=com_csw&view=details&product_id=GLOBAL_ANALYSIS_FORECAST_PHY_001_024 (visited on 07/19/2020).
- [2] CMEMS Ifremer Centre. *GLOBAL OCEAN WIND L4 NEAR REAL TIME 6 HOURLY OBSERVATIONS*. 2018. URL: http://marine.copernicus.eu/services-portfolio/access-to-products/?option=com_csw&view=details&product_id=WIND_GLO_WIND_L4_NRT_OBSERVATIONS_012_004 (visited on 03/09/2020).
- [3] CLS. *Motu Client Python Project*. 2019. URL: <https://github.com/clstoulouse/motu-client-python> (visited on 03/31/2020).
- [4] J. Gerritsma, J. A. Keuning, and A. Versluis. "Sailing Yacht Performance in Calm Water and in Waves". In: *12th International Symposium on Yacht Design and Construction HISWA* (1992).
- [5] S. A. Hsu, Eric A. Meindl, and David B. Gilhousen. "Determining the Power-Law Wind-Profile Exponent under Near-Neutral Stability Conditions at Sea". In: *Journal of Applied Meteorology* 33.6 (June 1994), pp. 757–765. ISSN: 0894-8763. DOI: 10.1175/1520-0450(1994)033<0757:DTPLWP>2.0.CO;2. eprint: [https://journals.ametsoc.org/jamc/article-pdf/33/6/757/3896591/1520-0450\(1994\)033<0757<_dtplwp>2.0.CO;2.pdf](https://journals.ametsoc.org/jamc/article-pdf/33/6/757/3896591/1520-0450(1994)033<0757<_dtplwp>2.0.CO;2.pdf). URL: [https://doi.org/10.1175/1520-0450\(1994\)033%3C0757:DTPLWP%3E2.0.CO;2](https://doi.org/10.1175/1520-0450(1994)033%3C0757:DTPLWP%3E2.0.CO;2).
- [6] Jesse Miller. "Directional Stability of Proposed Control Surface Concepts". In: (2015).
- [7] *NACA 0015 (naca0015-il)*. URL: <http://airfoiltools.com/airfoil/details?airfoil=naca0015-il>. (accessed: 06.22.2020).
- [8] Wikipedia, the free encyclopedia. *Local tangent plane coordinates*. [Online; accessed February 20, 2020]. 2020. URL: https://en.wikipedia.org/wiki/Local_tangent_plane_coordinates#/media/File:ECEF_ENU_Longitude_Latitude_relationships.svg.

A MATLAB Code for Polar Diagram Program

A.1 setBoatParam.m

```
1 function [p,z0]=setBoatParam
2 %{
3 % Initializes various boat parameters in p structure
4 %
5 % Input
6 %     None
7 % Output
8 %     p: structure storing boat parameters, created by setBoatParam
9 %     z0: initial conditions
10 %
11 % Date: Oct. 19 2020
12 % Author: Daisy Zhang
13 %}
14 %% Initialized parameters
15 %initial conditions
16 x0=0; y0=0; th0=1; xdot0=0; ydot0=0; thdot0=0; %initial pose and (d/dt)pose
17 z0=[x0,y0,th0,xdot0,ydot0,thdot0]';
18
19
20 %%p.accuracy defines the accuracy with which lift and drag coeff. are
21 %%interpolated
22 %1: most accurate but slower (using pchip interpolation)
23 %2: less accurate but fastest (approx. Clift Cdrag as sinusoidal)
24 p.accuracy=1;
25
26 %sail
27 p.d_sail=0; %distance from C.O.M. to sail [m] (positive is infront COM)
28 p.SA_sail=-.2; %surface area sail [m^2]
29 p.angle_sRelb=0; %angle of sail relative to boat [rad]
30 p.angle_sRelw=0.09;
31
32
33 %keel
34 p.d_keel=0; %distance from C.O.M. to keel [m] (positive is infront of COM)
35 p.SA_keel=0.1; %surface area keel [m^2]
36
37 %rudder
38 p.d_rudder=-0.5; %distance from C.O.M. to rudder [m] (positive is infront COM)
39 p.d_rRels=-0.5; %distance from sail center to rudder [m] (positive is infront of sail)
40 p.SA_rudder=0.05; %surface area rudder [m^2]
41 p.angle_rRelb=0*pi/180; %angle of rudder relative to boat [rad]
42 p.angle_rRels=0; %angle of rudder relative to sail [rad]
43 p.rudderType=3; %1: water rudder 2: air rudder 3: tail
44
45 %hull
46 p.SA_hull = 1;
47
48
49 p.v_airMag=-1; %magnitude of air velocity
50 p.v_airAngle=0; %direction of wind [rad]
51 p.v_a=p.v_airMag*[cos(p.v_airAngle),sin(p.v_airAngle)]; %x-y velocity componenets of air [m/
52 s]
53 p.mass=7; %mass of boat [kg]
54 p.I=(1/12)*p.mass; %moment of inertia of boat about COM [kg*m^2]
55 p.rho_air=1.2; %density of air [kg/m^3]
56 p.rho_water=1000; %density of water [kg/m^3]
57
58 %% Tabulated Data for NACA 0015 airfoil:
59 angle = [0,10,15,17,23,33,45,55,70,80,90,100,110,120,130,...
60 140,150,160,170,180,190,200,210,220,230,240,250,260,...
61 270,280,290,305,315,327,337,343,345,350,360]';
62
```

```

63 lift = [0,0.863,1.031,0.58,.575,.83,.962,.8579,.56,.327,...
64         .074,-.184,-.427,-.63,-.813,-.898,-.704,-.58,-.813,0,...
65         .813,.58,.704,.898,.813,.63,.427,.184,-.074,-.327,...
66         -.56,-.8579,-.962,-.83,-.575,-.58,-1.031,-.863,0]';
67
68 drag = [0,.021,.058,.216,.338,.697,1.083,1.421,1.659,1.801,...
69         1.838,1.758,1.636,1.504,1.26,.943,.604,.323,.133,0,...
70         .133,.323,.604,.943,1.26,1.504,1.636,1.758,1.838,1.801,...
71         1.659,1.421,1.083,.697,.338,.216,.058,.021,0]';
72
73 p.paraDrag=0.02; %parasitic drag used to limit max(Cl/Cd)=5
74 % drag=drag+p.paraDrag; %adjusted drag
75
76 p.CO=.9; % nominal lift coefficient for sinusoidal lift/drag approx.
77
78 %%% Fit a pchip piecewise-polynomial curve fit to the data
79 % This is good because it preserves local maximum and minimum
80 % Another option would be to replace pchip() with spline(), which would
81 % produce a more smooth curve, but would add new peaks to the data.
82 p.lift = pchip(angle, lift);
83 p.drag = pchip(angle, drag);
84
85 p.lift_f = fit(angle, lift, 'linearinterp');
86 p.drag_f = fit(angle, drag, 'linearinterp');
87
88 p.Ra = 0.5; % distance between center and COE of sail
89 p.Rh = 0.1; % distance between center and COE of keel
90 p.l = 0.5; % distance between COM and center
91
92 p.heelinglimit = pi/12; % the limit of the heeling angle
93
94 %%% Updated parameters for the sailboat prototype
95 p.epsilon_sea = 1.19e-6;
96 p.g = 9.81;
97
98 p.d_sail=0.117;
99 p.SA_sail=0.64994;
100 p.m_sail = 6.39;
101 p.rho_sail = p.m_sail/p.SA_sail;
102
103 p.d_keel=0.103;
104 p.SA_keel=0.38276;
105 p.m_keel = 6.39;
106 p.rho_keel = p.m_keel/p.SA_keel;
107 p.c_keel = 0.28219 ;% mean chord length keel
108
109 p.d_rudder= -2.78;
110 p.d_rRels=p.d_rudder-p.d_sail;
111 p.SA_rudder=0.13224;
112 p.m_rudder = 1.4;
113 p.rho_rudder = p.m_rudder/p.SA_rudder;
114
115 p.LWL = 1.512; % water line length
116 p.BWL = 0.198; % water line breadth
117 p.Tc = 0.13; % hull draft
118 p.T = 1.545; % total draft
119 r = p.BWL/2;
120 p.SA_hull = 0.6673;
121 p.immersed_SA_hull = 0.38569; % immersed hull area
122 p.immersed_vol_hull = 0.01788; % immersed hull volume
123 p.immersed_w_hull = p.rho_water*p.immersed_vol_hull*p.g; % immersed hull weight
124 p.Cp = 0.5487423121; % prismatic coefficient -> assume
125 p.LCB = 0; % longitude center of buoyancy
126 p.Aw = 0.14285; % waterline area
127
128
129 p.mass=25.478;
130 p.I=15.073404;

```

```

131
132 p.Ra = 0.758;
133 p.Rh = 0.691;
134 p.l = 1.054;
135
136 % Residuary resistance polynomial coefficients (Gerritsma et al, 1992)
137 Fr_1 = (0.125:0.025:0.45)';
138 a0 = [-6.735654 -0.382870 -1.503526 11.29218 22.17867 25.90867 40.97559 45.83759
139      89.20382...
140      212.6788 336.2354 566.5476 743.4107 1200.62]';
141 a1 = [38.36831 38.17290 24.40803 -14.51947 -49.16784 -74.75668 -114.2855 -184.7646...
142      -393.0127 -801.7908 -1085.134 -1609.632 -1708.263 -2751.715]';
143 a2 = [-0.008193 0.007243 0.0122 0.047182 0.085998 0.153521 0.207226 0.357031 0.617466...
144      1.087307 1.644191 2.016090 2.435809 3.208577]';
145 a3 = [0.055234 0.026644 0.067221 0.085176 0.150725 0.188568 0.250827 0.338343 0.460472...
146      0.538938 0.532702 0.265722 0.013553 0.254920]';
147 a4 = [-1.997242 -5.295332 -2.448582 -2.673016 -2.878684 -0.889467 -3.072662 3.871658
148      11.54327 ...
149      10.80273 -1.223173 -29.24412 -81.16189 -132.0424]';
150 a5 = [-38.86081 -39.55032 -31.91370 -11.41819 7.167049 24.12137 53.01570 132.2568
151      331.1197...
152      667.6445 831.1445 1154.091 937.4014 1489.269]';
153 a6 = [0.956591 1.219563 2.216098 5.654065 8.600272 10.48516 13.02177 10.86054 8.598136...
154      12.39815 26.18321 51.46175 115.6006 196.3406]';
155 a7 = [-.002171 .000052 .000074 .007021 .012981 .025348 .035934 .066809 .104073...
156      .166473 .238795 .2888046 .365071 .528225]';
157 a8 = [0.272895 .824568 .244345 -.094934 -.327085 -0.85494 -0.715457 -1.719215 -2.815203
158      -3.026131...
159      -2.45047 -.0178354 1.838967 1.379102]';
160 a9 = [-0.017516 -0.047842 -0.015887 0.006325 0.018271 0.048449 0.039874 0.095977...
161      0.15596 0.165055 0.139154 0.018446 -0.062023 0.013577]';
162
163 p.a0 = fit(Fr_1,a0,'cubicinterp');
164 p.a1 = fit(Fr_1,a1,'cubicinterp');
165 p.a2 = fit(Fr_1,a2,'cubicinterp');
166 p.a3 = fit(Fr_1,a3,'cubicinterp');
167 p.a4 = fit(Fr_1,a4,'cubicinterp');
168 p.a5 = fit(Fr_1,a5,'cubicinterp');
169 p.a6 = fit(Fr_1,a6,'cubicinterp');
170 p.a7 = fit(Fr_1,a7,'cubicinterp');
171 p.a8 = fit(Fr_1,a8,'cubicinterp');
172 p.a9 = fit(Fr_1,a9,'cubicinterp');
173
174 Fr_2 = (0.475:0.025:0.75)';
175 c0 = [180.1004 243.9994 282.9873 313.4109 337.0038 356.4572 324.7357 301.1268 292.0571...
176      284.4641 256.6367 304.1803]';
177 c1 = [-31.50257 -44.52551 -51.51953 -56.58257 -59.19029 -62.85395 -51.31252 -39.79631...
178      -31.85303 -25.14558 -19.31922 -30.11512]';
179 c2 = [-7.451141 -11.15456 -12.97310 -14.41978 -16.06975 -16.85112 -15.34595 -15.02299 ...
180      -15.58548 -16.15423 -13.08450 -15.85429]';
181 c3 = [2.195042 2.179046 2.274505 2.326117 2.419156 2.437056 2.334146 2.059657 1.847926
182      1.703981...
183      2.152824 2.863173]';
184 c4 = [2.689623 3.857403 4.343662 4.690432 4.766793 5.078768 3.855368 2.545676 1.569917
185      0.817912 0.348305 1.524379]';
186 c5 = [.00648 .009676 .011066 .012147 .014147 .014980 .013695 .013588 .014014 .014575 .011343
187      .014031]';
188 p.c0 = fit(Fr_2,c0,'cubicinterp');
189 p.c1 = fit(Fr_2,c1,'cubicinterp');
190 p.c2 = fit(Fr_2,c2,'cubicinterp');
191 p.c3 = fit(Fr_2,c3,'cubicinterp');
192 p.c4 = fit(Fr_2,c4,'cubicinterp');
193 p.c5 = fit(Fr_2,c5,'cubicinterp');
194
195 RR = [];
196 Fr_lst = [];
197 for v = 0.1:0.01:5
198     Fr = v/sqrt(p.g*p.LWL);

```



```

192 % Hull residual resistance
193 if Fr >= 0.125 && Fr < 0.45
194     R_R = p.immersed_w_hull/1e3*(p.a0(Fr)+p.a1(Fr)*p.Cp+p.a2(Fr)*p.LCB+p.a3(Fr)*(p.BWL/p
    .Tc)...
195         +p.a4(Fr)*(p.LWL/(p.immersed_vol_hull^(1/3)))+p.a5(Fr)*p.Cp^2+p.a6(Fr)*p.Cp*...
196         (p.LWL/(p.immersed_vol_hull^(1/3)))+p.a7(Fr)*p.LCB^2+p.a8(Fr)*...
197         (p.LWL/(p.immersed_vol_hull^(1/3)))^2+p.a9(Fr)*(p.LWL/(p.immersed_vol_hull^(1/3)
    ))^3);
198 elseif Fr >= 0.45 && Fr < 0.475
199     if abs(Fr-0.45) <= abs(Fr-0.475)
200         R_R = p.immersed_w_hull/1e3*(p.a0(.45)+p.a1(.45)*p.Cp+p.a2(.45)*p.LCB+p.a3(.45)
    *(p.BWL/p.Tc)...
201         +p.a4(.45)*(p.LWL/(p.immersed_vol_hull^(1/3)))+p.a5(.45)*p.Cp^2+p.a6(.45)*p.Cp
    *...
202         (p.LWL/(p.immersed_vol_hull^(1/3)))+p.a7(.45)*p.LCB^2+p.a8(.45)*...
203         (p.LWL/(p.immersed_vol_hull^(1/3)))^2+p.a9(.45)*(p.LWL/(p.immersed_vol_hull
    ^((1/3)))^3);
204     else
205         R_R = p.immersed_w_hull/1e3*(p.c0(.475)+p.c1(.475)*(p.LWL/p.BWL)+p.c2(.475)*(p.
    Aw/(p.immersed_vol_hull^(2/3)))+...
206         p.c3(.475)*p.LCB+p.c4(.475)*(p.LWL/p.BWL)^2+p.c5(.475)*(p.LWL/p.BWL)*(p.Aw/(p.
    immersed_vol_hull^(2/3)))^3);
207     end
208 elseif Fr >= 0.475 && Fr < 0.75
209     R_R = p.immersed_w_hull/1e3*(p.c0(Fr)+p.c1(Fr)*(p.LWL/p.BWL)+p.c2(Fr)*(p.Aw/(p.
    immersed_vol_hull^(2/3)))+...
210     p.c3(Fr)*p.LCB+p.c4(Fr)*(p.LWL/p.BWL)^2+p.c5(Fr)*(p.LWL/p.BWL)*(p.Aw/(p.
    immersed_vol_hull^(2/3)))^3);
211 elseif Fr >= 0.75
212     R_R = p.immersed_w_hull/1e3*(p.c0(.75)+p.c1(.75)*(p.LWL/p.BWL)+p.c2(.75)*(p.Aw/(p.
    immersed_vol_hull^(2/3)))+...
213     p.c3(.75)*p.LCB+p.c4(.75)*(p.LWL/p.BWL)^2+p.c5(.75)*(p.LWL/p.BWL)*(p.Aw/(p.
    immersed_vol_hull^(2/3)))^3);
214 else
215     R_R = 0;
216 end
217 RR = [RR R_R/(0.5*p.rho_water*norm(v)^2*p.immersed_SA_hull)];
218 Fr_lst = [Fr_lst Fr];
219 end
220 % fit residual resistance to Fn
221 p.fRR = fit(Fr_lst',RR','cubicinterp');
222
223 % Cr and Fn values read from (Dubrovsky et al, 2006)
224 Cr = [0.55,0.55,0.8,0.9,1.2,1.4,1.8,2.3,4.5,5.6,7.1,9.2,12]/1000;Fn =
    [0.15,0.2,0.24,0.25,0.27,0.28,0.3,0.32,0.36,0.37,0.38,0.39,0.4];
225 CroverLCB = [0.1,0.335];Fn4CroverLCB = [0.24,0.3];
226 delta_Cr = fit(Fn4CroverLCB',CroverLCB','poly1');
227 LCB = [1.5,-2]/100;Fn4LCB = [0.18,0.26];
228 delta_LCB = fit(Fn4LCB',LCB','poly1');
229 Cr = Cr+delta_Cr(Fn)'.*delta_LCB(Fn)'./1000+0.16*(p.BWL/p.Tc-2.5)/1000;
230
231 % extend ranges of Cr and of Fn by reading values from the plots generated
232 % from formulas in (Gerritsma et al, 1992)
233 Cr = [0,Cr,0.026,0.0168,0.0106,0.0091908,0.0078313,0.0068829,0.0059881,0.005127];
234 Fn = [0,Fn,0.51151,0.649,0.86983,0.93474,1.0126,1.0801,1.158,1.2515];
235 p.fRR=fit(Fn',Cr','cubicinterp');
236 end

```

A.2 polar_diagram.m

```

1 function [speed,dir,heel_angles,angles,labels] = polar_diagram(p,sail_angle,rudder_angle)
2 %{
3 % Draw the polar diagram of the given boat.
4 %
5 % Input
6 %     p: structure storing boat parameters, created by setBoatParam
7 %     function
8 %     sail_angle: list of sail angles, size: 1xn

```

```

9 % rudder_angle: list of rudder angles, size: 1xm
10 % Output
11 % speed: nxm matrix storing the speed of the boat.
12 % speed(i,j) = speed of the boat with sail angle = sail_angle(i)
13 % rudder_angle = rudder_angle(j)
14 % dir: nxm matrix storing the direction of the boat.
15 % dir(i,j) = boat direction, sail = sail_angle(i),rudder =
16 % rudder_angle(j)
17 % heel_angles: cell array containing all possible heel angles of boat
18 % angles: cell arrays s.t.
19 % angles{1,:} = all boat rudder angle rel. to sail
20 % angle{2,:} = all boat sail angle rel. to boat
21 % angle{3,:} = all boat rudder angle rel. to boat
22 % angle{4,:} = all boat course direction
23 % labels: cell arrays containing the wind speed information string
24 % for later labeling
25 %
26 % Date: Oct. 18 2020
27 % Author: Daisy Zhang
28 %}
29
30 h=waitbar(0,'Sailing...');
31 t=1;
32 final = figure(3);
33 p3 = polaraxes;
34
35 sail_angle_mat = zeros(length(sail_angle),length(rudder_angle));
36 rudder_angle_mat = zeros(length(sail_angle),length(rudder_angle));
37
38 n =3;
39 tend = length(sail_angle)*length(rudder_angle)*n;
40 spd = linspace(35,37,n);
41 best_dir = cell(1,n);
42 best_spd = cell(1,n);
43 color = {'r','b','k','g','m','c','y'};
44 heel_angles = cell(1,n);
45 angles = cell(4,n);
46
47 for i = 1:n
48     speed = zeros(length(sail_angle),length(rudder_angle));
49     dir = zeros(length(sail_angle),length(rudder_angle));
50     h1 = figure;
51     ph1 = polaraxes;
52     hold on
53     p.v_airMag = spd(i);
54     p.v_airAngle = pi;
55     p.v_a=p.v_airMag*[cos(p.v_airAngle),sin(p.v_airAngle)];
56     hold on
57     for idx = 1 : length(rudder_angle)
58         p.angle_rRelb = rudder_angle(idx);
59         for jdx = 1:length(sail_angle)
60             waitbar(t/tend,h);
61             t=t+1;
62             p.angle_sRelb = sail_angle(jdx);
63             p.angle_rRels = p.angle_rRelb-p.angle_sRelb;
64             ANS = root_finding(p); % calculate velocity of the boat
65             v_boat_x= ANS(2);
66             v_boat_y = ANS(3);
67             % calculate heel angle
68             try
69                 heeling = heeling_angle(ANS,p);
70             catch
71                 heeling = 0;
72             end
73             heel_angles{i} = [heel_angles{i},heeling];
74             angles{1,i} = [angles{1,i} p.angle_rRels];
75             angles{2,i} = [angles{2,i} p.angle_sRelb];
76             angles{3,i} = [angles{3,i} p.angle_rRelb];

```

```

77     angles{4,i} = [angles{4,i} atan2(v_boat_y,v_boat_x)];
78     v = [ v_boat_x;v_boat_y];
79     speed(idx,jdx) = norm(v); % store speed
80     dir(idx,jdx) = atan2(v_boat_y,v_boat_x);% store direction
81     sail_angle_mat(idx,jdx) = sail_angle(jdx);
82     rudder_angle_mat(idx,jdx) = rudder_angle(idx);
83     if dir(idx,jdx)<=pi && dir(idx,jdx)>=0 && heeling < p.heelinglimit
84         % plot direction in [0,pi/2] since the polar diagram is symmetric
85         % and only plot if the velocity is safe for sailing
86         if norm(v) ~=0
87             polarplot(atan2(v_boat_y,v_boat_x),speed(idx,jdx),'k.');
```

```

88         else
89             polarplot(atan2(v_boat_y,v_boat_x),speed(idx,jdx),'rx');
```

```

90         end
91         ph1.ThetaLim = [0 90];
92         ph1.ThetaZeroLocation = 'top';
93     end
94 end
95 end
96 figure(3)
97 sail_angle_sol = [];
98 rudder_angle_sol = [];
99 for idx = 1 : length(sail_angle)
100     for jdx = 1:length(rudder_angle)
101         same_dir_config = round(rad2deg(dir))/10 == round(rad2deg(dir(idx,jdx))/10);
102         spd_in_same_dir = speed(same_dir_config);
103         sail_in_same_dir = sail_angle_mat(same_dir_config);
104         rudder_in_same_dir = rudder_angle_mat(same_dir_config);
105         [M,I] = max(spd_in_same_dir); % find the max speed in approximately the same
direction
106         sail_angle_sol = [sail_angle_sol sail_in_same_dir(I)];
107         rudder_angle_sol = [rudder_angle_sol rudder_in_same_dir(I)];
108         direction = deg2rad(round(rad2deg(dir(idx,jdx))/10)*10);
109         best_dir{i} = [best_dir{i} direction];
110         best_spd{i} = [best_spd{i} M];
111     end
112 end
113 [sorted_dir, sorted_I] = sort(best_dir{i});
114 best_dir{i} = sorted_dir;
115 sorted_spd = best_spd{i};
116 best_spd{i} = sorted_spd(sorted_I);
117 assignin('base','best_dir',best_dir)
118 assignin('base','best_spd',best_spd)
119 sail_angle_sol = [sail_angle_sol(sorted_I)];
120 rudder_angle_sol = [rudder_angle_sol(sorted_I)];
121 pp = polarplot(best_dir{i},best_spd{i},'color',color{i},'marker','.');
```

```

122 %plot max speed in
all directions
123 p3.ThetaZeroLocation = 'top';
124 p3.ThetaLim = [0 180];
125 pp.DataTipTemplate.DataTipRows(1).Label = 'Direction [deg]';
126 pp.DataTipTemplate.DataTipRows(2).Label = 'Speed [m/s]';
127 row = dataTipTextRow('Sail Angle [deg]',rad2deg(sail_angle_sol));
128 row1 = dataTipTextRow('Rudder Angle [deg]',rad2deg(rudder_angle_sol));
129 pp.DataTipTemplate.DataTipRows(end+1) = row;
130 pp.DataTipTemplate.DataTipRows(end+1) = row1;
131 hold on
132 end
133 close(h);
134 title(sprintf('Polar Diagram'))
135 labels = {};
136 for i = 1:n
137     labels{i} = sprintf('Speed = %.2f m/s',spd(i));
138 end
139 legend(labels{1:n});
140 end

```

A.3 root finding.m

```

1 function ANS = root_finding(p)
2 %{
3 % Calculate the velocity and the heading of the boat by root finding
4 %
5 % Input
6 %   p: structure storing boat parameters, created by setBoatParam
7 %   function
8 % Output
9 %   ANS(2,3) = the velocity of the boat
10 %   ANS(1) = the heading of the boat
11 %
12 % Date: Oct. 19 2020
13 % Author: Daisy Zhang
14 %}
15
16 options = optimoptions('fsolve','Display','off','TolFun',1e-20);
17 for vb_x_guess = linspace(-3,3,20)
18     for vb_y_guess = linspace(-3,3,20)
19         for t_guess = linspace(0,2*pi,10)
20             z0 = [t_guess;vb_x_guess;vb_y_guess];
21             [z,~,exitflag,~] = fsolve(@(z) rhs(z,p),z0,options);
22             if exitflag > 0
23                 vb_x=z(2);
24                 vb_y=z(3);
25                 t = z(1);
26                 ANS = [t;vb_x;vb_y];
27                 return
28             end
29         end
30     end
31 end
32 end

```

A.4 heeling angle.m

```

1 % Main script for drawing the polar diagram for the given sailboat
2 % Date: Oct. 18 2020
3 % Author: Daisy Zhang
4
5 function ANS = heeling_angle(z,p)
6 %{
7 % Calculate the heeling angle of the boat
8 %
9 % Input
10 %   p: structure storing boat parameters, created by setBoatParam
11 %   function
12 %   z: current pose and (d/dt)pose
13 % Output
14 %   ANS = heel angle
15 %
16 % Date: Oct. 19 2020
17 % Author: Daisy Zhang
18 %}
19 options = optimoptions('fsolve','TolFun',1e-20,'Display','off');
20 for heel_guess = linspace(0,2*pi,40)
21     z0 = heel_guess;
22     [heel,~,exitflag,~] = fsolve(@(heel_angle) rolling_moment(heel_angle,z,p),z0,
        options);
23     if exitflag > 0
24         ANS = heel;
25         return
26     end
27 end
28 end
29
30
31 function M = rolling_moment(heel,z,p)
32 %{

```

```

33 % Calculate the rolling moment of the boat
34 %
35 % Input
36 %     heel: the boat's heeling angle
37 %     p: structure storing boat parameters, created by setBoatParam
38 %     function
39 %     z: current pose and (d/dt)pose
40 % Output
41 %     M: the rolling moment
42 %
43 % Date: Oct. 19 2020
44 % Author: Daisy Zhang
45 %}
46 [L_sail,D_sail,L_keel,D_keel,L_rudder,D_rudder] = FM_on_boat(z,p);
47 n = [sin(z(1)), -cos(z(1))];
48 F_a = L_sail+D_sail+L_rudder+D_rudder;
49 F_h = L_keel+D_keel;
50 M = abs(dot(F_a,n))*p.Ra*cos(abs(heel))+abs(dot(F_h,n))*p.Rh*cos(abs(heel))-p.mass*p.g*p.l*
    sin(abs(heel));
51 end

```

A.5 FM_on_boat.m

```

1 function [L_sail,D_sail,L_keel,D_keel,L_rudder,D_rudder,R_hull,M_rudder,M_keel,M_sail,M_hull
    ] = FM_on_boat(z,p)
2 %{
3 % Calculate force and moment on the boat
4 %
5 % Input
6 %     p: structure storing boat parameters, created by setBoatParam
7 %     function
8 %     z: current pose and (d/dt)pose
9 % Output
10 %     L_sail: lift force on sail
11 %     D_Sail: drag force on sail
12 %     L_keel: lift force on keel
13 %     D_keel: drag force on keel
14 %     L_rudder: lift force on rudder
15 %     D_rudder: drag force on rudder
16 %     R_hull: resistance on hull
17 %     M_rudder: the area moment on rudder
18 %     M_keel: the area moment on keel
19 %     M_sail: the area moment on sail
20 %     M_hull: the area moment on hull
21 %
22 % Date: Oct. 19 2020
23 % Author: Daisy Zhang
24 %}
25
26 vb_x = z(2);
27 vb_y = z(3);
28 t = z(1);
29 v_boat = [vb_x,vb_y];
30 omega = 0;
31 %% SAIL
32 %%Calculates lift drag forces on a sail at a fixed angle relative to the
33 %%boat
34 %sail velocity relative to air
35 v_sail=v_boat+p.d_sail*omega*[-sin(t),cos(t)]-p.v_a;
36
37 %angle of attack of sail in air
38 alpha_sail= (t+p.angle_sRelb-atan2(v_sail(2),v_sail(1)));
39 [c_lift,c_drag]=C_LD(alpha_sail,p);
40
41 %lift and drag on sail
42 L_sail=.5*p.rho_air*p.SA_sail*norm(v_sail)*...
43     c_lift*[-v_sail(2),v_sail(1)];
44

```

```

45
46 D_sail=.5*p.rho_air*p.SA_sail*norm(v_sail)*...
47   c_drag*(-v_sail);
48
49 %Moment
50 M_sail=p.d_sail*cos(t)*(L_sail(2)+D_sail(2))-...
51   p.d_sail*sin(t)*(L_sail(1)+D_sail(1));
52
53 %% RUDDER
54 %%%Calculates the forces on the rudder given it is in the air
55 %velocity of rudder relative to air
56
57
58 v_rudder=v_boat+p.d_rudder*omega*[-sin(t+p.angle_sRelb),cos(t+p.angle_sRelb)]-p.v_a;
59
60 %angle of attack of rudder in air
61 alpha_rudder= (t+p.angle_rRels+p.angle_sRelb-atan2(v_rudder(2),v_rudder(1)));
62
63
64 [c_lift,c_drag]=C_LD(alpha_rudder,p);
65
66 %lift and drag on rudder
67 L_rudder=.5*p.rho_air*p.SA_rudder*norm(v_rudder)*...
68   c_lift*[-v_rudder(2),v_rudder(1)];
69 D_rudder=.5*p.rho_air*p.SA_rudder*norm(v_rudder)*...
70   c_drag*(-v_rudder);
71
72 %Moment
73 rx=p.d_rRels*cos(t+p.angle_sRelb)+p.d_sail*cos(t);
74 ry=p.d_rRels*sin(t+p.angle_sRelb)+p.d_sail*sin(t);
75 M_rudder=rx*(L_rudder(2)+D_rudder(2))-ry*(L_rudder(1)+D_rudder(1));
76
77 %% KEEL
78 %%%Calculates lift drag forces on keel
79
80 %velocity of keel in water
81 v_keel=v_boat+p.d_keel*omega*[-sin(t),cos(t)];
82
83 %angle of attack of keel in water
84 alpha_keel= (t-atan2(v_keel(2),v_keel(1)));
85
86 [c_lift,c_drag]=C_LD(alpha_keel,p);
87
88 %lift and drag on keel
89 L_keel=.5*p.rho_water*p.SA_keel*norm(v_keel)*...
90   c_lift*[-v_keel(2),v_keel(1)];
91
92 D_keel=.5*p.rho_water*p.SA_keel*norm(v_keel)*...
93   c_drag*(-v_keel);
94
95 %Moment
96 M_keel=p.d_keel*cos(t)*(L_keel(2)+D_keel(2))-...
97   p.d_keel*sin(t)*(L_keel(1)+D_keel(1));
98
99 %% HULL
100 %%%Calculates hull resistance and rotational damping
101
102 % Reynolds number hull
103 Rn_hull = norm(v_boat)*0.7*p.LWL/p.epsilon_sea;
104
105 % Hull frictional resistance
106 R_F = 0.5*p.rho_water*norm(v_boat)^2*p.immersed_SA_hull*(0.075/(log10(Rn_hull)-2)^2);
107
108 % Froude number
109 Fr = norm(v_boat)/sqrt(p.g*p.LWL);
110
111 % Hull residual resistance
112 if Fr < 1.25

```

```

113 R_R = p.fRR(Fr)*(0.5*p.rho_water*norm(v_boat)^2*p.immersed_SA_hull);
114 else
115 R_R = p.fRR(1.25)*(0.5*p.rho_water*norm(v_boat)^2*p.immersed_SA_hull);
116 end
117 % solve for heeling angle
118 n = [sin(t),-cos(t)];
119 F_a = L_sail+D_sail+L_rudder+D_rudder;
120 F_h = L_keel+D_keel;
121 heeling = atan((abs(dot(F_a,n))*p.Ra+abs(dot(F_h,n))*p.Rh)/p.mass/p.g/p.l);
122
123 q = 0.5*p.rho_water*norm(v_boat)^2;
124 CH = 1/1e3*(6.747*(p.Tc/p.T)+2.517*(p.BWL/p.Tc)+3.71*(p.BWL/p.T));
125
126 %angle of attack of keel in water
127 alpha_hull= (t-atan2(v_boat(2),v_boat(1)));
128
129 [c_lift,c_drag]=C_LD(alpha_hull,p);
130
131 %lift and drag on keel
132 L_hull=.5*p.rho_air*(p.SA_hull-p.immersed_SA_hull)/2*norm(v_boat)*...
133     c_lift*[-v_boat(2),v_boat(1)];
134
135 D_hull=.5*p.rho_air*(p.SA_hull-p.immersed_SA_hull)/2*norm(v_boat)*...
136     c_drag*(-v_boat);
137
138 % Aspect ratio of keel and hull combined
139 ARE = p.T^2/(p.SA_keel+p.immersed_SA_hull);
140
141 % Side force of heel and keel combined
142 FH = abs(dot(F_h+L_hull+D_hull,n));
143
144 % Induced resistance
145 R_i = FH^2/(pi*q*p.immersed_SA_hull*ARE);
146
147
148 % Heeling resistance
149 R_H = q*p.immersed_SA_hull*CH*Fr^2*heeling;
150
151
152 R_hull = -v_boat/norm(v_boat)*(R_F+R_H+R_R+R_i)+L_hull+D_hull;
153 %hull damping moment
154 M_hull=-2*omega;
155 end
156 function [c_lift,c_drag]=C_LD_hull(alpha,p)
157 %%%calculates local hull coeff. of lift and drag at a given angle of attack
158 %%(alpha)
159
160     c_lift=p.CO*sin(2*alpha);
161     hull_paraDrag = 0.15;
162     c_drag=hull_paraDrag+p.CO*(1-cos(2*alpha));
163
164 end

```

A.6 rhs.m

```

1 function F = rhs(z,p)
2 %{
3 % Calculate force and moment on the boat
4 %
5 % Input
6 %     p: structure storing boat parameters, created by setBoatParam
7 %     function
8 %     z: current pose and (d/dt)pose
9 % Output
10 %     F: F(1,2) is the net force on boat, F(3) is the net moment
11 %
12 % Date: Oct. 19 2020
13 % Author: Daisy Zhang

```

```

14 %}
15
16 [L_sail,D_sail,L_keel,D_keel,L_rudder,D_rudder,R_hull,M_rudder,M_keel,M_sail,M_hull] =
    FM_on_boat(z,p);
17
18 F(3)= M_rudder+M_keel+M_sail+M_hull;
19 Ftot=L_sail+D_sail+L_keel+D_keel+L_rudder+D_rudder+R_hull;
20 F(1) = Ftot(1);
21 F(2) = Ftot(2);
22 end

```

A.7 C_LD.m

```

1 function [c_lift,c_drag]=C_LD(alpha,p)
2 %{
3 % calculates local coeff. of lift and drag at a given angle of attack
4 %
5 % Input
6 %     alpha: angle of attack
7 %     p: structure storing boat parameters, created by setBoatParam
8 % Output
9 %     c_lift: lift coefficient
10 %     c_drag: drag_coefficient
11 %
12 % Date: Oct. 19 2020
13 % Author: Daisy Zhang
14 %}
15 %     c = 1/15;
16 %     attenuator = (pi-1)*(exp(-(sin(alpha).^4/c^2))) + 1;
17 c_lift=p.C0*sin(2*alpha);
18 %     c_lift = c_lift.*attenuator; % fits to the NACA0015 data better but
19 %                                     % is calculation heavy
20 c_drag=p.paraDrag+p.C0*(1-cos(2*alpha));
21
22 end

```

A.8 optimize.m

```

1 %{
2 % Drawing the polar diagram for the given sailboat
3 % by optimizing the maximum speed in all directions.
4 % Date: Oct. 18 2020
5 % Author: Daisy Zhang
6 %}
7
8 figure
9 pa = polaraxes;
10 [p,z0]=setBoatParam;
11 h=waitbar(0,'Sailing...');
12 t=1;
13 dir_lst = -pi/2:0.1:pi/2;
14 wind_lst = 10:10:40;
15 tend = length(dir_lst)*length(wind_lst);
16 i=1;
17 spd = cell(1,length(wind_lst));
18 direct = spd;
19
20 colors = ['k','r','m','b','g'];
21
22 % optimize the maximum speed on all directions
23 for wind = wind_lst
24     p.v_airMag=-wind;
25     p.v_airAngle=0;
26     p.v_a=p.v_airMag*[cos(p.v_airAngle),sin(p.v_airAngle)];
27     for dir = dir_lst
28         p.angle_rRelb = dir;
29         waitbar(t/tend,h);

```



```

30     t=t+1;
31     [x,fval] = optimize_fun(dir,p); % find the sail and tail angles for max speed
32     if ~isnan(x)
33         p.angle_sRelb = x(1);%p.angle_rRelb = x(2);
34         p.angle_rRels = p.angle_rRelb-p.angle_sRelb;
35         ANS = root_finding(p);
36     else
37         ANS = [0;0;0];
38     end
39     spd{i} = [spd{i} norm(ANS(2:3))];
40     direct{i} = [direct{i} atan2(ANS(3),ANS(2))];
41     polarplot(atan2(ANS(3),ANS(2)),norm(ANS(2:3)),'marker','.', 'color',colors(i));
42     hold on
43     pa.ThetaLim = [0 180];
44     pa.ThetaZeroLocation = 'top';
45 end
46 i = i+1;
47 end
48 labels = {};
49 for j = 1:i-1
50     labels{j} = sprintf('Speed = %.2f m/s',wind_lst(j));
51 end
52 legend(labels{1:i-1});
53
54 close(h)
55 figure(2);
56 p2 = polaraxes('ThetaZeroLocation','top');
57 figure(3);
58 p3 = polaraxes('ThetaZeroLocation','top');
59
60 % Fit the speed data points to corresponding directions by cubic interpolation
61 for j = 1:i-1
62     [s_direct,s_index] = sort(direct{j});
63     spd_j = spd{j};
64     s_spd = spd_j(s_index);
65     rmv_out_spd = filloutliers(s_spd,'pchip','movmedian',10);
66     figure(2);
67     polarplot(direct{j},spd_j,[colors(j) '.'],s_direct,rmv_out_spd,'rx');
68     p2.ThetaZeroLocation = 'top';
69     hold on
70     figure(3)
71     f = fit(s_direct,rmv_out_spd,'cubicinterp');
72     polarplot(s_direct,f(s_direct))
73     p3.ThetaZeroLocation = 'top';
74     hold on
75 end
76
77
78 function [x,fval] = optimize_fun(dir,p)
79 %%% optimize the VMG magnitude by finding
80 %%% sail/tail angles
81 opts = optimoptions('fmincon','Display','iter','Algorithm','interior-point','MaxIterations',
82     ,5e02);
83 p.angle_rRelb = dir;
84
85 for s_angle_guess = -pi/2:0.1:pi/2
86     x0 = [s_angle_guess];
87     [x,fval,eflag] = runobjconstr(x0,p,opts);
88     if eflag > 0
89         return
90     elseif eflag == 0
91         x = nan; fval = nan;
92     end
93 end
94
95 function [x,f,eflag,outpt] = runobjconstr(x0,p,opts)
96 %%% optimize the objective by equality and inequality constraints

```

```

97
98 if nargin == 1 % No options supplied
99     opts = [];
100 end
101
102 xLast = []; % Last place computeall was called
103 myf = []; % Use for objective at xLast
104 myc = []; % Use for nonlinear inequality constraint
105 myceq = []; % Use for nonlinear equality constraint
106
107 fun = @(x) objfun(x,p); % the objective function, nested below
108 cfun = @(x) constr(x,p); % the constraint function, nested below
109
110 % Call fmincon
111 [x,f,eflag,outpt] = fmincon(fun,x0,[],[],[],[],[-pi/2],[pi/2],cfun,opts);
112
113 function y = objfun(x,p)
114     if ~isequal(x,xLast) % Check if computation is necessary
115         [myf,myc,myceq] = computeall(x,p);
116         xLast = x;
117     end
118     % Now compute objective function
119     y = -norm(myf(2:3));
120 end
121
122 function [c,ceq] = constr(x,p)
123     if ~isequal(x,xLast) % Check if computation is necessary
124         [myf,myc,myceq] = computeall(x,p);
125         xLast = x;
126     end
127     % Now compute constraint functions
128     c = myc; % In this case, the computation is trivial
129     ceq = myceq;
130 end
131
132 end
133
134
135 function [f1,c1,ceq1] = computeall(x,p)
136 %%% produce the objective, inequality constraint and equality constraint
137
138 p.angle_sRelb = x(1);
139 p.angle_rRelb = p.angle_rRelb - p.angle_sRelb;
140 f1 = root_finding(p);
141 c1 = [heeling_angle(f1,p) - pi/6 - atan2(f1(3),f1(2))];
142 ceq1 = [];
143 end

```

B MATLAB Code for Trajectory Simulation in Ocean

B.1 main.m

```
1 %{\n2 % Main script for the simulation trajectory program\n3 %\n4 % Date: Oct. 28 2020\n5 % Author: Daisy Zhang\n6 %}\n7\n8 clear % clear all variables including global variables\n9 clear global\n10\n11 % % =====The code below will open a world map. User can =====\n12 % % follow the instructions to select the start location and destination\n13 % % =====\n14 % worldmap world\n15 % load coastlines\n16 % plotm(coastlat, coastlon)\n17 % h = gcf;\n18 % h.Units = 'normalized';\n19 % h.Position = [0.1,0.1,0.8,0.8];\n20 % title('Click to Select Your Starting Point, then Click to Select Your End Point')\n21 % [lat, lon] = inputm(2);\n22 % close gcf\n23 % x0 = lon(1); % initial lon * same way measuring lon as wind dataset*\n24 % y0 = lat(1); % initial lat\n25 % y_des = lat(2); x_des = lon(2);\n26 % % =====The code above will open a world map. User can =====\n27 % % follow the instructions to select the start location and destination\n28 % % =====\n29\n30 % Start location coordinates\n31 % x: longitude AND y:latitude\n32 x0 = -59.8683;y0 = 39.0402; % equivalent to 39.0402 degree N, 59.8683 W\n33 % destination coordinates equivalent to 34.20 degree N, 34.20 W\n34 x_des = -34.204359486664400; y_des = 34.201656541126450;\n35\n36 %p is a structure containing all necessary information for the tour\n37 p = struct;\n38 p.des = [x_des,y_des]; % store the destination to p structure\n39\n40 % Initial conditions\n41 spd = 0;\n42 theta0 = 0;\n43 z0 = [x0,y0]';\n44 zdot0 = [spd*cos(theta0),spd*sin(theta0)]';\n45\n46 dataStart = datetime('19900101','InputFormat','yyyyMMdd'); % dataset starting date\n47 start = datetime('20190101','InputFormat','yyyyMMdd'); % simulated tour start date\n48\n49 % hours btw 1990-01-01 00:00:00 and 2019-01-01 00:00:00\n50 global DIFF\n51 DIFF = hours(start-dataStart);\n52\n53 % maximum travel duration (in hours)\n54 % if users want to change the tour end date, just replace 20190302 to the\n55 % desired date\n56 p.tf = hours(datetime('20190302','InputFormat','yyyyMMdd')-start);\n57 % tour state time i.e. the boat will be launched "t0" hours after "start".\n58 p.t0 = 0;\n59\n60 % trajectory(T+deltaT) = trajectory(T) + velocity*deltaT\n61 % fps = 3600/(deltaT) i.e. times of updates per hour
```

```

62 p.fps =1;
63 p.tspan = linspace(p.t0+DIFF,p.tf+DIFF,(p.tf-p.t0)*p.fps);
64
65 % struct containing fields of data for ocean current
66 % fields include:
67 % - depth: the depth of the measured ocean current [m]
68 % - latitude, longitude: as the name specified [deg]
69 %           e.g. -30 in longitude = 30 degrees W; -30 in latitude = 30
70 %           degrees S.
71 % - vo, uo: at depth, water current speed in v direction, u direction respectively [m/s]
72 % - times: the time of the ocean current being measured [hours after 1950-01-01]
73 % other detailed information can be found in the research report
74 % "Simulating the Trajectory for a Directionally Stable Sailboat" by Daisy
75 % Zhang
76 [~, outstrct_w]=read_nc_file_struct('./global-analysis-forecast-phy-001-024-3dinst-
    uovo_1595967643151.nc');
77
78 % struct containing fields of data for ocean current
79 % fields include:
80 % - lat, lon: as the name specified [deg]
81 %           e.g. -30 in lon = 30 degrees W; -30 in lat = 30
82 %           degrees S.
83 % - northward_wind, eastward_wind: at 10 m above sea surface, wind speed in north direction,
84 %           and east direction respectively [m/s]
85 % - time: the time of the wind velocity being measured [hours after 1990-01-01]
86 % other detailed information can be found in the research report
87 % "Simulating the Trajectory for a Directionally Stable Sailboat" by Daisy
88 % Zhang
89 [~, outstrct_a]=read_nc_file_struct('CERSAT-GLO-BLENDED_WIND_L4-V6-
    OBS_FULL_TIME_SERIE_1581555950466.nc');
90
91 h=waitbar(0,'Sailing...');
92 % =====Core of the program =====
93 [t,zarray]=myOde(@rhs,p.tspan,z0,p,h,outstrct_w,outstrct_a,NaN,NaN);
94 close(h);
95 % plot the simulated trajectory on map, along with other information
96 stats(t,zarray,p)
97 % =====
98
99 function [t,zarray] = myOde(rhs,tspan,z0,p,h,outstrct_w,outstrct_a,init_strct_in_use,
    init_wind_file)
100 % ODE solver that can solve systems of equations of the form y' = f(t,y)
101 % INPUT:
102 %   - rhs: the right-hand-side function
103 %   - tspan: time span
104 %   - z0: initial conditions
105 %   - p: sturcture storing tour information
106 %   - h: the handle to the figure showing intergation progress
107 %   - outstrct_w: struct storing ocean current data
108 %   - outstrct_a: struct storing wind data
109 %   - init_strct_in_use: NaN when the program doesn't use automation
110 %           data retrieval; otherwise, init_strct_in_use is the
111 %           strct containing wind data for the beginning of the
112 %           simulation
113 %   - init_wind_file: NaN when the program doesn't use automation
114 %           data retrieval; otherwise, init_wind_file is the name
115 %           of the NetCDF file containing wind data for the
116 %           beginning of the simulation
117 % OUTPUT:
118 %   - t: tspan
119 %   - zarray: numerical solution to the integration of rhs over tspan with
120 %           initial conditions z0
121 global DIFF
122 numInt = 1; % number of iterations
123 t = tspan;
124 dt = (tspan(2) - tspan(1))*3600;
125 zarray = z0;
126 z = zarray(:,end);

```

```

127 i = 1;
128 while i <= length(tspan)
129     time = tspan(i);
130     ifcollide = 0; % the boat doesn't collide with land
131     for k=1:numInt
132         % % =====The code below will allow data automation retrieval=====
133         % %           Read find_wind_HTTPS doc before uncommenting the code below
134         % [file_name,outstrct] = find_wind_HTTPS(init_wind_file,init_strct_in_use,time,
135             double(i==1));
136         % init_wind_file = file_name;
137         % init_strct_in_use = outstrct;
138         % % =====The code above will allow data automation retrieval=====
139         [zdot,ifcollide]=rhs(time,z,p,h,outstrct_w,outstrct_a);
140         z=z+(1/numInt)*dt*zdot;
141     end
142     if ifcollide % if a collision happens
143         idx = find(tspan==time);
144         t = tspan(1:idx)-DIFF; % the time when the collision happens
145         break % break the loop
146     else
147         zarray(:,end+1) = z;
148         i = i+1;
149     end
150 end
151 t = t-DIFF;
152 end

```

B.2 rhs.m

```

1 function [zdot,ifcollide] = rhs(t,z,p,h,outstrct_w,outstrct_a)
2 %{
3 % rhs is the abbreviation for "right hand side" function. i.e. y' = f(t,y)
4 % It is the system of differential equations to be solved.
5 % INPUT:
6 % t: hours since 1990-01-01 00:00:00 [hour]
7 % z: current coordinates at t;
8 %     z(1): current longitude coordinate [deg]
9 %     z(2): current latitude coordinate [deg]
10 % p: struct created in main
11 % h: the handle to the figure showing intergation progress
12 % outstrct_w:struct containing fileds of data for ocean current
13 % outstrct_a:struct containing fileds of data for wind
14 % OUTPUT:
15 % zdot: boat's velocity at t [deg/s]
16 % ifcollide: 1 if there's collistion between boat and any land; 0 otherwise
17
18 % Date: Oct. 29 2020
19 % Author: Daisy Zhang
20 %}
21
22 global DIFF
23
24 pos = z(1:2);
25
26 [vcx,vcy,ifcollide_sea] = seamotion(pos(1),pos(2),t,outstrct_w); % water velocity rel.to
27     ground [m/s]
28
29 if ~ifcollide_sea % if not collide
30     [vax,vay,ifcollide_wind] = windmotion(pos(1),pos(2),t,outstrct_a);% wind velocity rel.to
31     ground [m/s]
32 else
33     vax = 0; vay = 0;ifcollide_wind = 1;
34 end
35
36 ifcollide = ifcollide_sea||ifcollide_wind;
37
38 if ~ifcollide % if no collision happens
39     wind_spd = norm([vax,vay]); % wind speed at 10 m above sea surface

```

```

38     surface_wind_spd = (.1)^(0.11)*wind_spd; % wind speed at sea surface; check research
    report for the equation
39     [vx,vy] = polar_plot(surface_wind_spd,p,pos(1),pos(2)); % boat velocity rel. to water [m
    /s]
40     zdot = [vcx+vx,vcy+vy]; % boat velocity rel. to ground [m/s]
41     R = 6.371*10^6; % radius of Earth
42     zdot = [rad2deg(zdot(1)/(R*cos(deg2rad(pos(2)))));rad2deg(zdot(2)/R)]; % convert boat
    velocity in [m/s] to [deg/s]
43 else % if any collision happens
44     zdot = [0,0];
45 end
46
47 waitbar((t-DIFF-p.t0)/(p.tf-p.t0),h); % update the handle bar
48 end

```

B.3 polar_plot.m

```

1 function [vx,vy] = polar_plot(true_wind_spd,p,x,y)
2 %{
3 % Calculate boat velocity rel. to water by wind and polar diagram
4 % INPUT:
5 %   true_wind_spd: true wind speed [m/s]
6 %   p: struct created in main
7 %   x: current longitude
8 %   y: current latitude
9 % OUTPUT:
10 %   vx: boat speed rel. to water in x direction (along the local longitude axis)
11 %   vy: boat speed rel. to water in y direction (along the local latitude axis)
12
13 % Date: Oct. 29 2020
14 % Author: Daisy Zhang
15 %}
16
17 % for now, the boat is assumed to always travel to the destination at a
18 % constant speed (0.1 m/s).
19 % Later the function should be able to calculate the boat's velocity given
20 % wind, and polar diagram generated by the other program.
21 theta = atan2(p.des(2)-y,p.des(1)-x);
22 vx = 0.1*cos(theta);
23 vy = 0.1*sin(theta);
24 end

```

B.4 seamotion.m

```

1 function [u,v,ifcollide] = seamotion(x,y,t,outstrct)
2 %{
3 % Output water current velocity and flag of any collision at the given time
4 % and location
5 % INPUT:
6 %   x: current longitude [deg]
7 %   y: current latitude [deg]
8 %   t: hours since 1990-01-01 00:00:00 [hour]
9 %   outstrct: struct containing fields of data for ocean current
10 % OUTPUT:
11 %   u: zonal water current speed at (x,y) when t [m/s]
12 %   v: meridional current speed at (x,y) when t [m/s]
13 %   ifcollide: 1 if there's collision between boat and any land; 0 otherwise
14
15 % Date: Oct. 29 2020
16 % Author: Daisy Zhang
17 %}
18
19 ifcollide = 0;
20 lat = length(outstrct.latitude);
21 lon = length(outstrct.longitude);
22
23 temp = datetime('19900101','InputFormat','yyyymmdd');

```

```

24 hour_diff = hours(temp+hours(t)-datetime('19500101','InputFormat','yyyyMMdd')); % hours
    since 1950-01-01 00:00:00
25
26 u = inf; v = inf;
27
28 % find the index of current time in file
29 for i = 1:length(outstrct.time)
30     if hour_diff == outstrct.time(i) || (hour_diff > outstrct.time(i) && ...
31         hour_diff < outstrct.time(i+1))
32         time = i;
33     end
34 end
35
36 % find the indexes of latitude and longitude in file, then do bilinear
37 % interpolation
38 for j = 1:lat
39     for i = 1:lon
40         if outstrct.latitude(j) <= y && outstrct.latitude(j+1) >= y ...
41             && outstrct.longitude(i) <= x && outstrct.longitude(i+1) >= x
42             u = grid_interpol(x,y,time,outstrct,i,j,outstrct.uo);
43             v = grid_interpol(x,y,time,outstrct,i,j,outstrct.vo);
44             break
45         end
46     end
47     if ~isinf(u) && ~isinf(v) % if u and v are found, break the loop
48         break
49     end
50 end
51
52 if isnan(u)||isnan(v) % if any u or v is NaN, collision happens
53     fprintf('Land Collision\n')
54     u = 0;
55     v = 0;
56     ifcollide = 1;
57 end
58
59 function u = grid_interpol(x,y,t,outstrct,i,j,uo)
60 % Bilinear interpolate on the outstrct to find speed at the point (x,y)
61 % at given time
62 % INPUT:
63 %   x: longitude coordinate [deg]
64 %   y: latitude coordinate [deg]
65 %   t: the index of time approximating to current time so that
66 %       outstrct.time(t) < current time < outstrct.time(t+1)
67 %   outstrct: the struct containing data
68 %   i: the index of longitude so that
69 %       outstrct.longitude(i) < x < outstrct.longitude(i+1)
70 %   j: the index of latitude so that
71 %       outstrct.latitude(i) < y < outstrct.latitude(i+1)
72 %   uo: the matrix to be interpolated
73
74 lo = round([outstrct.longitude(i) outstrct.longitude(i+1)],2);
75 la = round([outstrct.latitude(j) outstrct.latitude(j+1) ],2);
76
77 % create a rectilinear 2d grid for interpolation
78 u_mat = [uo(i,j,1,t) uo(i+1,j,1,t);uo(i,j+1,1,t) uo(i+1,j+1,1,t)];
79
80 [Lo,La] = meshgrid(lo,la);
81
82 x = round(x,2);
83 y = round(y,2);
84
85 [Loq,Laq] = meshgrid(lo(1):0.01:lo(2),la(1):0.01:la(2));
86
87 % interpolation for 2-D gridded data in meshgrid format
88 u_mat_q = interp2(Lo,La,u_mat,Loq,Laq,'linear');
89 Loq = round(Loq,2);
90 Laq = round(Laq,2);

```

```

91 tol = 0.005; % difference tolerance
92 u = u_mat_q((abs(Loq-x)<tol)&(abs(Laq-y)<tol));
93 end
94
95 end

```

B.5 windmotion.m

```

1 function [u,v,ifcollide] = windmotion(x,y,t,outstrct)
2 %{
3 % Output water current velocity and flag of any collision at the given time
4 % and location
5 % INPUT:
6 %   x: current longitude coordinate[deg]
7 %   y: current latitude coordinate [deg]
8 %   t: hours since 1990-01-01 00:00:00 [hour]
9 %   outstrct: struct containing wind data info
10 % OUTPUT:
11 %   u: zonal speed of the wind at (x,y) when t
12 %   v: meridional speed of the wind at (x,y) when t
13 %   ifcollide: 1 if there's collision between boat and any land; 0 otherwise
14
15 % Date: Oct. 29 2020
16 % Author: Daisy Zhang
17 %}
18
19 ifcollide = 0;
20
21 lat = length(outstrct.lat);
22 lon = length(outstrct.lon);
23
24 u = inf; v = inf;
25
26 now = datetime('19900101','InputFormat','yyyyMMdd')+hours(t); % now date time
27 outstrct.time = datetime('19000101','InputFormat','yyyyMMdd')+hours(outstrct.time); % data
    date time
28
29 % find the index of current time in file
30 for i = 1:length(outstrct.time)-1
31     if now == outstrct.time(i) || (now >= outstrct.time(i) && ...
32         now <= outstrct.time(i+1))
33         time = i;
34     end
35 end
36
37 % find the indexes of latitude and longitude in file, then do bilinear
38 % interpolation
39 for j = 1:lat
40     for i = 1:lon
41         if outstrct.lat(j) <= y && outstrct.lat(j+1) >= y ...
42             && outstrct.lon(i) <= x && outstrct.lon(i+1) >= x
43             u = grid_interpol(x,y,outstrct,i,j,time,outstrct.eastward_wind);
44             v = grid_interpol(x,y,outstrct,i,j,time,outstrct.northward_wind);
45             break
46         end
47     end
48     if ~isinf(u) && ~isinf(v) % if u and v are found, break the loop
49         break
50     end
51 end
52 if isnan(u)||isnan(v) % if any u or v is NaN, collision happens
53     fprintf('Land Collision\n')
54     u = 0;
55     v = 0;
56     ifcollide = 1;
57 end
58
59

```



```

60 function u = grid_interpol(x,y,outstrct,i,j,t,uo)
61 % Bilinear interpolate on the outstrct to find speed at the point (x,y)
62 % at given time
63 % INPUT:
64 %   x: longitude coordinate [deg]
65 %   y: latitude coordinate [deg]
66 %   outstrct: the struct containing data
67 %   i: the index of longitude so that
68 %       outstrct.longitude(i) < x < outstrct.longitude(i+1)
69 %   j: the index of latitude so that
70 %       outstrct.latitude(i) < y < outstrct.latitude(i+1)
71 %   t: the index of time approximating to current time so that
72 %       outstrct.time(t) < current time < outstrct.time(t+1)
73 %   uo: the matrix to be interpolated
74
75 lo = [outstrct.lon(i) outstrct.lon(i+1)];
76 la = [outstrct.lat(j) outstrct.lat(j+1)];
77
78 lo = round(lo,2);
79 la = round(la,2);
80
81 % create a rectilinear 2d grid for interpolation
82 u_mat = [uo(i,j,t) uo(i+1,j,t);uo(i,j+1,t) uo(i+1,j+1,t)];
83
84 [Lo,La] = meshgrid(lo,la);
85 x = round(x,2);
86 y = round(y,2);
87 [Loq,Laq] = meshgrid(lo(1):0.01:lo(2),la(1):0.01:la(2));
88
89 % interpolation for 2-D gridded data in meshgrid format
90 u_mat_q = interp2(Lo,La,u_mat,Loq,Laq,'linear');
91 Loq = round(Loq,2);
92 Laq = round(Laq,2);
93 tol = 0.005;
94 u = u_mat_q((abs(Loq-x)<tol)&(abs(Laq-y)<tol));
95 end
96 end

```

B.6 stats.m

```

1 function stats(t,zarray,p)
2 %{
3 % Plot the simulated trajectory (or other info if there exists)
4 % INPUT:
5 %   t: tspan
6 %   zarray: numerical solution to the integration of differential equations
7 %   p: struct created in main
8
9 % Date: Oct. 29 2020
10 % Author: Daisy Zhang
11 %}
12
13 figure(2);
14 hold on;
15 zarray = zarray';
16 % plot the simulated trajectory
17 h1=plot(zarray(:,1),zarray(:,2),'linewidth',2,'DisplayName','Trajectory');
18 % plot start point
19 h2=plot(zarray(1,1),zarray(1,2),'rs','markersize',5,'linewidth',2,'DisplayName','Start');
20 text(zarray(1,1),zarray(1,2),'Start')
21
22 % labels & title
23 xlabel('Latitude','fontsize',16);
24 ylabel('Longitude','fontsize',16);
25 h=legend([h1,h2]);
26 title(sprintf('Trajectory over %0.2f Days',(t(end)-t(1))/24))
27 set(h,'fontsize',14,'location','best');
28 axis equal;

```

```

29 axis([-100 25 0 80])
30
31 % Plots a google map on the current axes using the Google Static Maps API
32 plot_google_map('MapType','terrain')
33 axis([-100 25 0 80])
34
35 makescale;
36 set(h,'fontsize',14,'location','best');
37 hold off
38 end

```

B.7 find_wind_HTTPS.m

```

1 function [file_name,outstrct] = find_wind_HTTPS(file,strct_in_use,t,iffirst)
2 %{
3 % Achieve wind data automatic retrieval via HTTPS server
4 % INPUT:
5 %   file: wind data file name which is used for the simulation right now
6 %   strct_in_use: strct containing wind data info in use for the simulation
7 %               right now
8 %   t: hours since 1990-01-01-00:00:00 [hour]
9 %   iffirst: 1 if no wind data set being automatically downloaded on file, 0 otherwise
10
11 % OUTPUT:
12 %   file_name: the name of the file just being downloaded
13 %   outstrct: strct containing applicable wind data info
14
15 % Date: Oct. 28 2020
16 % Author: Daisy Zhang
17 %}
18
19 YOUR_CMEMS_USERNAME = ''; % your username to cmems account
20 YOUR_CMEMS_PASSWORD = ''; % your password to cmems
21 YOUR_LOCAL_PATH = 'C:\Users\daisy\Desktop\SimTemp CurrentOnly'; % the path for storing
    downloaded data file
22 if iffirst
23
24     temp = datetime('19900101','InputFormat','yyyyMMdd');
25     now = temp+hours(t);
26     begin_date = datetime([year(now),month(now),day(now)]);
27
28     % 5 months of data is around 1 GB, the maximum size available for
29     % download each time. Can input smaller number for better download success rate.
30     end_date = begin_date + calmonths(5);
31
32     % the file format of data file in the dataset
33     formatOut = 'yyyyMMdd HH:MM:ss';
34     begin_date_str = datestr(begin_date,formatOut);
35     end_date_str = datestr(end_date,formatOut);
36     % the file format for renaming the downloaded data
37     formatOutfile = 'yyyymmddHHMMss';
38     begin_date_str_file = datestr(begin_date,formatOutfile);
39     end_date_str_file = datestr(end_date,formatOutfile);
40     % the name of the downloaded file
41     file_name = [begin_date_str_file, end_date_str_file, '.nc'];
42     command = ['python -m motuclient --motu http://nrt.cmems-du.eu/motu-web/Motu --service-
id WIND_GLO_WIND_L4_NRT_OBSERVATIONS_012_004-TDS --product-id CERSAT-GLO-BLENDED_WIND_L4
-V6-OBS_FULL_TIME_SERIE --longitude-min -100 --longitude-max 10 --latitude-min 0 --
latitude-max 60 --date-min "' begin_date_str '" --date-max "' end_date_str '" --variable
    eastward_wind --variable northward_wind --out-dir "' YOUR_LOCAL_PATH '" --out-name "'
    begin_date_str_file end_date_str_file '.nc' --user "' YOUR_CMEMS_USERNAME '" --pwd "'
    YOUR_CMEMS_PASSWORD '"'];
43
44     [status,cmdout] = system(command) % open a new cmd window and execute command
45
46     [~,outstrct] = read_nc_file_struct(file_name);
47 else
48     temp = datetime('19900101','InputFormat','yyyyMMdd');

```

```

49     now = temp+hours(t);
50     begin_date = datetime(file(1:14),'InputFormat','yyyyMMddHHmmss');
51     end_date = datetime(file(15:28),'InputFormat','yyyyMMddHHmmss');
52     if now <= end_date && now >= begin_date % if time is within the time span of current
        available file
53         outstrct = strct_in_use;
54         file_name = file;
55     else % otherwise
56         begin_date = end_date;
57         % 5 months of data is around 1 GB, the maximum size available for
58         % download each time. Can input smaller number for better download success rate.
59         end_date = begin_date + calmonths(5);
60
61         % the file format of data file in the dataset
62         formatOut = 'yyyy-mm-dd HH:MM:ss';
63         begin_date_str = datestr(begin_date,formatOut);
64         end_date_str = datestr(end_date,formatOut);
65         % the file format for renaming the downloaded data
66         formatOutfile = 'yyyymmddHHMMss';
67         begin_date_str_file = datestr(begin_date,formatOutfile);
68         end_date_str_file = datestr(end_date,formatOutfile);
69         file_name = [begin_date_str_file, end_date_str_file, '.nc'];
70         command = ['python -m motuclient --motu http://nrt.cmems-du.eu/motu-web/Motu --
        service-id WIND_GLO_WIND_L4_NRT_OBSERVATIONS_012_004-TDS --product-id CERSAT-GLO-
        BLENDED_WIND_L4-V6-OBS_FULL_TIME_SERIE --longitude-min -100 --longitude-max 10 --
        latitude-min 0 --latitude-max 60 --date-min "' begin_date_str '" --date-max "'
        end_date_str '" --variable eastward_wind --variable northward_wind --out-dir "'
        YOUR_LOCAL_PATH '" --out-name "' begin_date_str_file end_date_str_file '.nc' --user "'
        YOUR_CMEMS_USERNAME '" --pwd "' YOUR_CMEMS_PASSWORD '"'];
71         [status,cmdout] = system(command) % open a new cmd window and execute command
72         [~,outstrct] = read_nc_file_struct(file_name);
73         delete(file);
74
75     end
76 end
77 end
78 % file_name = file;
79 % outstrct = strct_in_use;

```

B.8 find_wind_FTP.m

```

1 function outstrct = find_wind_FTP(t)
2 %{
3 % Achieve wind data automatic retrieval via FTP server
4 % Extremely slow - do not recommend
5 % INPUT:
6 %   t: hours since 1990-01-01-00:00:00 [hour]
7
8 % OUTPUT:
9 %   outstrct: strct containing applicable wind data info
10
11 % Date: Oct. 28 2020
12 % Author: Daisy Zhang
13 %}
14
15 YOUR_CMEMS_USERNAME = ''; % your username to cmems account
16 YOUR_CMEMS_PASSWORD = ''; % your password to cmems
17 DATA_PATH = './Core/WIND_GLO_WIND_L4_NRT_OBSERVATIONS_012_004/CERSAT-GLO-BLENDED_WIND_L4-V6-
    OBS_FULL_TIME_SERIE'; % the path to data file in FTP driver
18 ftpobj = ftp('nrt.cmems-du.eu',YOUR_CMEMS_USERNAME,YOUR_CMEMS_PASSWORD,'System','WINDOWS','
    LocalDataConnectionMethod','passive');
19 cd(ftpobj,DATA_PATH);
20
21 temp = datetime('19900101','InputFormat','yyyyMMdd');
22 now = temp+hours(t);
23 if month(now)<10 % if the month is from Jan to Sep (01-09)
24     folder = sprintf('./%i/0%i',year(now),month(now));
25 else % otherwise (10-12)

```

```

26     folder = sprintf('./%i/%i',year(now),month(now));
27 end
28 cd(ftpobj,folder); % move to the folder containing the month of data
29 listing = dir(ftpobj); % listing all files inside the folder
30 file = NaN;
31 for i = 1:length(listing)-1
32     file_date = listing(i).name(1:10); % parse the date
33     file_date = datetime(file_date,'InputFormat','yyyyMMddHH');
34     file_date_next = listing(i+1).name(1:10); % parse the date
35     file_date_next = datetime(file_date_next,'InputFormat','yyyyMMddHH');
36     if file_date==now || (file_date<now && now < file_date_next)
37         file = listing(i).name;
38     end
39 end
40
41 if isnan(file)
42     file = listing(end).name;
43 end
44
45 file_name = mget(ftpobj,file); % download the file
46 [finfo outstrct] = read_nc_file_struct(file_name{1}); % convert the NetCDF file to struct
47 close(ftpobj);
48 end

```

B.9 makescale.m

```

1 function h = makescale(varargin)
2 %MAKESCALE creates a scale for map data.
3 %
4 %     MAKESCALE creates a scale on the current axis based on the current axis
5 %     limits. The scale is made to occupy 1/5th of the map. It is placed
6 %     in the southeast corner of the map. The units will either be in
7 %     millimeters, meters or kilometers, depending on the size of the map.
8 %
9 %     MAKESCALE(H_AXIS) creates a scale on the axis specified by the handle
10 %     H_AXIS based on the its axes limits. H_AXIS must be a scalar.
11 %
12 %     MAKESCALE('width', WIDTH) creates a scale made to occupy up to WIDTH
13 %     of the map width.
14 %     WIDTH is bounded to be between 0.1 and 0.9.
15 %     If a larger value is passed in, 0.9 will be used.
16 %     If a smaller value is passed in, 0.1 will be used.
17 %
18 %     MAKESCALE('location', LOCATION) places the scale in the location specified by
19 %     LOCATION. Acceptable values for location are as follows
20 %         'northeast'      'ne'
21 %         'northwest'     'nw'
22 %         'southeast'     'se'
23 %         'southwest'     'sw'
24 %         'north'         'n'
25 %         'south'         's'
26 %
27 %     MAKESCALE('units',UNITS) changes the units systems from SI to imperial
28 %     units. UNITS should be either 'si' or 'imp' The units displayed
29 %     are automatically switched between millimeters, meters, and
30 %     kilometers for the SI system, or between inches, feet, and statute
31 %     miles for the imperial system.
32 %
33 %     H = MAKESCALE(...) outputs H, a 3x1 containing the handles of the of
34 %     box, line, and text.
35 %
36 %     Any number of these input sets may be passed in any order.
37 %
38 %     The map scale will automatically be updated as the figure is zoomed,
39 %     panned, resized, or clicked on. It will not, however, be updated
40 %     upon using the commands "axis", "xlim", or "ylim" as these do not
41 %     have callback functionality.
42 %

```

```

43 % Example:
44 %     load conus
45 %     figure
46 %     plot(uslon,uslat);
47 %     makeScale
48 %
49 % Example: Placed in the south
50 %     load conus
51 %     figure
52 %     plot(uslon,uslat);
53 %     makeScale('south')
54 %
55 % Example: Half the size of the Window
56 %     load conus
57 %     figure
58 %     plot(uslon,uslat);
59 %     makeScale(0.5,'south')
60 %
61 % Example: Use Imperial Units
62 %     load conus
63 %     figure
64 %     plot(uslon,uslat);
65 %     makeScale(0.5,'south','units','imp')
66 %
67 % Example: Zooming In
68 %     load conus
69 %     figure
70 %     plot(uslon,uslat);
71 %     makeScale(0.5,'south')
72 %     zoom(2)
73 %
74 % Note: This assumes axis limits are in degrees. The scale is sized
75 %       correctly for the center latitude of the map. As the size of
76 %       degrees longitude change with latitude, the scale becomes invalid
77 %       with very large maps. Spherical Earth is assumed. Ideally, the map
78 %       will be proportioned correctly in order to reflect the relationship
79 %       between a degree latitude and a degree longitude at the center of
80 %       the map.
81 %
82 % By Jonathan Sullian - October 2011
83 % Modified by Zohar Bar-Yehuda for plot_google_map - April 2018
84 %
85 % Parse Inputs
86 [anum, latlim, lonlim, width, location, units, set_callbacks] = parseInputs(varargin{:});
87 if ~isreal(width)
88     error('MAKESCALE:MapWithVal','MAPWIDTH must be a real number')
89 end
90
91 % Bound the width
92 width = min(max(width,0.1),0.9);
93 earthRadius = 6378137;
94
95 % Get the distance of the map
96 mlat = mean(latlim);
97 if abs(mlat) > 90;
98     d = 0;
99 else
100     d = earthRadius.*cosd(mlat).*deg2rad(diff(lonlim));
101 end
102 dlat = diff(latlim);
103 dlon = diff(lonlim);
104
105 % Calculate the distance of the scale bar
106 % rnd2 = floor(log10(d/scale))-1;
107
108 % Make the text string
109 if strcmpi(units,'si')
110     dscale = round_scale(d*width);

```

```

111     if d*width > 1e3
112         dst = num2str(dscale/1e3);
113         lbl = ' km';
114     elseif d*width > width
115         dst = num2str(dscale);
116         lbl = ' m';
117     else
118         dst = num2str(dscale*1e3);
119         lbl = ' mm';
120     end
121 else
122     if d*width > 1/0.000621371192
123         dscale = round_scale(d*width*0.000621371192);
124         dst = num2str(dscale);
125         lbl = ' mi';
126         dscale = dscale/0.000621371192;
127     elseif d*width > 0.3048
128         dscale = round_scale(d*width/0.3048);
129         dst = num2str(dscale);
130         lbl = ' ft';
131         dscale = dscale*.3048;
132     else
133         dscale = round2(d*width/0.3048*12);
134         dst = num2str(dscale);
135         lbl = ' in';
136         dscale = dscale/12*.3048;
137     end
138 end
139
140 % Get the postions
141 d1 = [-0.02 0.05];
142 issouth = 0;
143 iseast = 0;
144 iswest = 0;
145 switch lower(location)
146     case {'southeast','se'}
147         issouth = 1;
148         iseast = 1;
149     case {'northeast','ne'}
150         iseast = 1;
151     case {'southwest','sw'}
152         issouth = 1;
153         iswest = 1;
154     case {'northwest','nw'}
155         iswest = 1;
156     case {'north','n'}
157     case {'south','s'}
158         issouth = 1;
159 end
160
161 if issouth
162     slat = latlim(1)+0.05*diff(latlim);
163 else
164     slat = latlim(end)-0.08*diff(latlim);
165 end
166
167 if iseast
168     slon = lonlim(end)-0.05*diff(lonlim);
169     slon = [slon slon-rad2deg(dscale./(earthRadius.*cosd(mlat)))];
170     slat = [slat slat];
171 elseif iswest
172     slon = lonlim(1)+0.05*diff(lonlim);
173     slon = [slon slon+rad2deg(dscale./(earthRadius.*cosd(mlat)))];
174     slat = [slat slat];
175     slon = fliplr(slon);
176 else
177     slon = mean(lonlim);
178     slon = slon + [-rad2deg(dscale./(earthRadius.*cosd(mlat)))/2 rad2deg(dscale./(

```

```

    earthRadius.*cosd(mlat))/2)];
179     slat = [slat slat];
180     slon = flipplr(slon);
181 end
182
183 % Get the box location
184 blat = [slat([2 1])+[d1(1)*dlat d1(2)*dlat] slat([1 2])+[d1(2)*dlat d1(1)*dlat]];
185 blat = blat([2:4 1]);
186 blon = [slon+[0.02*dlon -0.02*dlon] slon([2 1])+[-0.02*dlon 0.02*dlon]];
187
188 % Delete Old Scale
189 aold = gca;
190 axes(anum);
191 ch = get(anum,'Children');
192 isOldScale = strcmpi(get(ch,'Tag'),'MapScale');
193 delete(ch(isOldScale));
194
195 % Make the scale
196 washold = ishold;
197 hold on
198 hbox = patch(blon,blat,'w','DisplayName','box');
199 set( get( get( hbox, 'Annotation'), 'LegendInformation' ), 'IconDisplayStyle', 'off' );
200 set(hbox,'Tag','MapScale');
201 hline = plot(slon,slat,'k','LineWidth',3, 'DisplayName','line');
202 set( get( get( hline, 'Annotation'), 'LegendInformation' ), 'IconDisplayStyle', 'off' );
203 set(hline,'Tag','MapScale');
204 units_axis = get(gca,'Units');
205 set(gca,'Units','Inches')
206 pos = get(gca,'OuterPosition');
207 sz = mean(pos(4));
208 htext = text(mean(blon),mean(blat)+.01*dlat,[dst lbl],'HorizontalAlignment','center','
    FontSize',sz*2.3);
209 hzoom = zoom;
210 hpan = pan(gcf);
211 set(htext,'Tag','MapScale')
212 set(gca,'Units',units_axis);
213
214 % Output Handles
215 if nargout > 0
216     h = [hbox; hline; htext];
217 end
218
219 % Restore Hold Off
220 if ~washold
221     hold off
222 end
223
224 % Set Resizer/Zoom/Pan/Click Callbacks
225 if set_callbacks
226     set(gcf,'ResizeFcn',{@ChangeTextSize,gca,htext});
227     set(hzoom,'ActionPostCallback',{@remakeZoomPanClick,anum,location,width,units});
228     set(hpan,'ActionPostCallback',{@remakeZoomPanClick,anum,location,width,units});
229     set(anum,'ButtonDownFcn',{@remakeZoomPanClick,anum,location,width,units});
230 end
231 axes(aold);
232
233 % Output Handles
234 if nargout > 0
235     h = [hbox; hline; htext];
236 end
237
238 % Restore Hold Off
239 if ~washold
240     hold off
241 end
242
243
244 function x = round2(x,base)

```

```

245 x = round(x./base).*base;
246
247 function scale = round_scale(scale)
248 pow = floor(log10(scale));
249 scale = scale / 10^pow;
250 if scale >= 5
251     scale = 5;
252 elseif scale >= 2
253     scale = 2;
254 elseif scale >= 1
255     scale = 1;
256 end
257 scale = scale * 10^pow;
258
259 % Change the text font on figure resize.
260 function ChangeTextSize(~,~,anum,htext)
261 units = get(anum,'Units');
262 set(anum,'Units','Inches')
263 pos = get(anum,'OuterPosition');
264 sz = mean(pos(4));
265 set(htext,'FontSize',sz*2.3);
266 set(anum,'Units',units);
267
268 function remakeZoomPanClick(~,~,anum,location,width,units)
269 makescale(anum, 'location', location, 'width', width, 'units', units);
270
271 function [anum, latlim, lonlim, width, location, units, set_callbacks] = parseInputs(
    varargin)
272 % Default Values
273 anum = gca;
274 width = 0.2;
275 location = 'se';
276 units = 'si';
277 set_callbacks = 1;
278
279 % Loop through number of arguments in
280 ii = 1;
281 while ii <= length(varargin)
282     % Either a axis number, or a scale value
283     if ii == 1 && isscalar(varargin{ii}) && ishandle(varargin{ii})
284         % Check if it's an axis handle
285         pos = get(varargin{ii},'ActivePositionProperty');
286         if strcmpi(pos,'outerposition')
287             anum = varargin{ii};
288         end
289
290         ii = ii + 1;
291
292     elseif ischar(varargin{ii})
293         param_name = lower(varargin{ii});
294         if strcmpi(param_name, 'units')
295             units = lower(varargin{ii+1});
296             if ~ismember(units, {'si', 'imp'})
297                 error('MAKESCALE:UNITS', 'UNITS must be one of the following: si, imp')
298             end
299             ii = ii + 2;
300         elseif strcmpi(param_name, 'location')
301             location = lower(varargin{ii+1});
302             locs = {'northeast','ne','north','n','southeast','se','south',...
303                 's','southwest','sw','northwest','nw'};
304             if ~ismember(location,locs)
305                 locOut = 'northeast, ne, north, n, southeast, se, south, s, southwest, sw,
306                 northwest, nw';
307                 error('MAKESCALE:LOCS',[ 'LOCATION must be one of the following: ' locOut])
308             end
309             ii = ii + 2;
310         elseif strcmpi(param_name, 'width')
311             width = varargin{ii+1};

```



```

311         ii = ii + 2;
312         elseif strcmpi(param_name, 'set_callbacks')
313             set_callbacks = varargin{ii+1};
314             ii = ii + 2;
315         else
316             error('MAKESCALE:PARAM',[ 'Unrecognized parameter: ' varargin{ii}]);
317         end
318     end
319 end
320
321 % Get limits
322 latlim = get(anum,'YLim');
323 lonlim = get(anum,'XLim');

```

B.10 plot_google_map.m

```

1 function varargout = plot_google_map(varargin)
2 % function h = plot_google_map(varargin)
3 % Plots a google map on the current axes using the Google Static Maps API
4 %
5 % USAGE:
6 % h = plot_google_map(Property, Value,...)
7 % Plots the map on the given axes. Used also if no output is specified
8 %
9 % Or:
10 % [lonVec latVec imag] = plot_google_map(Property, Value,...)
11 % Returns the map without plotting it
12 %
13 % PROPERTIES:
14 %     Axis          - Axis handle. If not given, gca is used.
15 %     Height (640)  - Height of the image in pixels (max 640)
16 %     Width (640)   - Width of the image in pixels (max 640)
17 %     Scale (2)     - (1/2) Resolution scale factor. Using Scale=2 will
18 %                   double the resolution of the downloaded image (up
19 %                   to 1280x1280) and will result in finer rendering,
20 %                   but processing time will be longer.
21 %     Resize (1)    - (recommended 1-2) Resolution upsampling factor.
22 %                   Increases image resolution using imresize(). This results
23 %                   in a finer image but it needs the image processing
24 %                   toolbox and processing time will be longer.
25 %     MapType       - ('roadmap') Type of map to return. Any of [roadmap,
26 %                   satellite, terrain, hybrid]. See the Google Maps API for
27 %                   more information.
28 %     Alpha (1)     - (0-1) Transparency level of the map (0 is fully
29 %                   transparent). While the map is always moved to the
30 %                   bottom of the plot (i.e. will not hide previously
31 %                   drawn items), this can be useful in order to increase
32 %                   readability if many colors are plotted
33 %                   (using SCATTER for example).
34 %     ShowLabels (1) - (0/1) Controls whether to display city/street textual labels on the
35 %                   map
36 %     Style          - (string) A style configuration string. See:
37 %                   https://developers.google.com/maps/documentation/static-maps/?csw=1#
38 %     StyledMaps     http://instrument.github.io/styled-maps-wizard/
39 %     Language       - (string) A 2 letter ISO 639-1 language code for displaying labels in a
40 %                   local language instead of English (where available).
41 %                   For example, for Chinese use:
42 %                   plot_google_map('language','zh')
43 %                   For the list of codes, see:
44 %                   http://en.wikipedia.org/wiki/List\_of\_ISO\_639-1\_codes
45 %     Marker         - The marker argument is a text string with fields
46 %                   conforming to the Google Maps API. The
47 %                   following are valid examples:
48 %                   '43.0738740,-70.713993' (default midsize orange marker)
49 %                   '43.0738740,-70.713993,blue' (midsize blue marker)
50 %                   '43.0738740,-70.713993,yellowa' (midsize yellow

```

```

51 %         '43.0738740,-70.713993,tinyredb' (tiny red marker
52 %         with label "B")
53 % Refresh (1) - (0/1) defines whether to automatically refresh the
54 %               map upon zoom/pan action on the figure.
55 % AutoAxis (1) - (0/1) defines whether to automatically adjust the axis
56 %               of the plot to avoid the map being stretched.
57 %               This will adjust the span to be correct
58 %               according to the shape of the map axes.
59 % MapScale (0) - (0/1) defines whether to add a scale indicator to
60 %               the map.
61 % ScaleWidth (0.25) - (0.1-0.9) defines the max width of the scale
62 %                   indicator relative to the map width.
63 % ScaleLocation (sw) - (ne, n, se, s, sw, nw) defines the location of
64 %                   scale indicator on the map.
65 % ScaleUnits (si) - (si/imp) changes the scale indicator units between
66 %                   SI and imperial units.
67 % FigureResizeUpdate (1) - (0/1) defines whether to automatically refresh the
68 %                   map upon resizing the figure. This will ensure map
69 %                   isn't stretched after figure resize.
70 % APIKey - (string) set your own API key which you obtained from Google:
71 %         http://developers.google.com/maps/documentation/staticmaps/#api\_key
72 %         This will enable up to 25,000 map requests per day,
73 %         compared to a few hundred requests without a key.
74 %         To set the key, use:
75 %         plot_google_map('APIKey','SomeLongStringObtainedFromGoogle')
76 %         You need to do this only once to set the key.
77 %         To disable the use of a key, use:
78 %         plot_google_map('APIKey','')
79 %
80 % OUTPUT:
81 % h - Handle to the plotted map
82 %
83 % lonVect - Vector of Longitude coordinates (WGS84) of the image
84 % latVect - Vector of Latitude coordinates (WGS84) of the image
85 % imag - Image matrix (height,width,3) of the map
86 %
87 % EXAMPLE - plot a map showing some capitals in Europe:
88 % lat = [48.8708 51.5188 41.9260 40.4312 52.523 37.982];
89 % lon = [2.4131 -0.1300 12.4951 -3.6788 13.415 23.715];
90 % plot(lon, lat, 'r', 'MarkerSize', 20)
91 % plot_google_map('MapScale', 1)
92 %
93 % References:
94 % http://www.mathworks.com/matlabcentral/fileexchange/24113
95 % http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/
96 % http://developers.google.com/maps/documentation/staticmaps/
97 % https://www.mathworks.com/matlabcentral/fileexchange/33545-automatic-map-scale-generation
98 %
99 % Acknowledgements:
100 % Val Schmidt for the submission of get_google_map.m
101 % Jonathan Sullivan for the submission of makescale.m
102 %
103 % Author:
104 % Zohar Bar-Yehuda
105 %
106 % Version 2.0 - 08/04/2018
107 % - Add an option to show a map scale
108 % - Several bugfixes
109 % Version 1.8 - 25/04/2016 - By Hannes Diethelm
110 % - Add resize parameter to resize image using imresize()
111 % - Fix scale parameter
112 % Version 1.7 - 14/04/2016
113 % - Add custom style support
114 % Version 1.6 - 12/11/2015
115 % - Use system temp folder for writing image files (with fallback to current dir if
116 %   missing write permissions)
117 % Version 1.5 - 20/11/2014
118 % - Support for MATLAB R2014b

```

```

118 % - several fixes for complex layouts: several maps in one figure,
119 % map inside a panel, specifying axis handle as input (thanks to Luke Plausin)
120 % Version 1.4 - 25/03/2014
121 % - Added the language parameter for showing labels in a local language
122 % - Display the URL on error to allow easier debugging of API errors
123 % Version 1.3 - 06/10/2013
124 % - Improved functionality of AutoAxis, which now handles any shape of map axes.
125 % Now also updates the extent of the map if the figure is resized.
126 % - Added the showLabels parameter which allows hiding the textual labels on the map.
127 % Version 1.2 - 16/06/2012
128 % - Support use of the "scale=2" parameter by default for finer rendering (set scale=1
    if too slow).
129 % - Auto-adjust axis extent so the map isn't stretched.
130 % - Set and use an API key which enables a much higher usage volume per day.
131 % Version 1.1 - 25/08/2011
132
133 persistent apiKey useTemp
134 if isnumeric(apiKey)
135     % first run, check if API key file exists
136     if exist('api_key.mat','file')
137         load api_key
138     else
139         apiKey = '';
140     end
141 end
142
143 if isempty(useTemp)
144     % first run, check if we have write access to the temp folder
145     try
146         tempfilename = tempname;
147         fid = fopen(tempfilename, 'w');
148         if fid > 0
149             fclose(fid);
150             useTemp = true;
151             delete(tempfilename);
152         else
153             % Don't have write access to temp folder or it doesn't exist, fallback to
            current dir
154             useTemp = false;
155         end
156     catch
157         % in case tempname fails for some reason
158         useTemp = false;
159     end
160 end
161
162 hold on
163
164 % Default parametr
165 axHandle = gca;
166 set(axHandle, 'Layer','top'); % Put axis on top of image, so it doesn't hide the axis lines
    and ticks
167 height = 640;
168 width = 640;
169 scale = 2;
170 resize = 1;
171 maptype = 'roadmap';
172 alphaData = 1;
173 autoRefresh = 1;
174 figureResizeUpdate = 1;
175 autoAxis = 1;
176 showLabels = 1;
177 language = '';
178 markeridx = 1;
179 markerlist = {};
180 style = '';
181 mapScale = 0;
182 scaleWidth = 0.25;

```

```

183 scaleLocation = 'se';
184 scaleUnits = 'si';
185
186 % Handle input arguments
187 if nargin >= 2
188     for idx = 1:2:length(varargin)
189         switch lower(varargin{idx})
190             case 'axis'
191                 axHandle = varargin{idx+1};
192             case 'height'
193                 height = varargin{idx+1};
194             case 'width'
195                 width = varargin{idx+1};
196             case 'scale'
197                 scale = round(varargin{idx+1});
198                 if scale < 1 || scale > 2
199                     error('Scale must be 1 or 2');
200                 end
201             case 'resize'
202                 resize = varargin{idx+1};
203             case 'maptype'
204                 maptype = varargin{idx+1};
205             case 'alpha'
206                 alphaData = varargin{idx+1};
207             case 'refresh'
208                 autoRefresh = varargin{idx+1};
209             case 'showLabels'
210                 showLabels = varargin{idx+1};
211             case 'figureresizeupdate'
212                 figureResizeUpdate = varargin{idx+1};
213             case 'language'
214                 language = varargin{idx+1};
215             case 'marker'
216                 markerList{markerIdx} = varargin{idx+1};
217                 markerIdx = markerIdx + 1;
218             case 'autoaxis'
219                 autoAxis = varargin{idx+1};
220             case 'apikey'
221                 apiKey = varargin{idx+1}; % set new key
222                 % save key to file
223                 funcFile = which('plot_google_map.m');
224                 pth = fileparts(funcFile);
225                 keyFile = fullfile(pth,'api_key.mat');
226                 save(keyFile,'apiKey')
227             case 'style'
228                 style = varargin{idx+1};
229             case 'mapscale'
230                 mapScale = varargin{idx+1};
231             case 'scalewidth'
232                 scaleWidth = varargin{idx+1};
233             case 'scalelocation'
234                 scaleLocation = varargin{idx+1};
235             case 'scaleunits'
236                 scaleUnits = varargin{idx+1};
237             otherwise
238                 error(['Unrecognized variable: ' varargin{idx}])
239         end
240     end
241 end
242 if height > 640
243     height = 640;
244 end
245 if width > 640
246     width = 640;
247 end
248
249 % Store paramters in axis handle (for auto refresh callbacks)
250 ud = get(axHandle, 'UserData');

```

```

251 if isempty(ud)
252     % explicitly set as struct to avoid warnings
253     ud = struct;
254 end
255 ud.gmap_params = varargin;
256 set(axHandle, 'UserData', ud);
257
258 curAxis = axis(axHandle);
259 if max(abs(curAxis)) > 500 || curAxis(3) > 90 || curAxis(4) < -90
260     warning('Axis limits are not reasonable for WGS1984, ignoring. Please make sure your
261         plotted data in WGS1984 coordinates,')
262     return;
263 end
264 % Enforce Latitude constraints of EPSG:900913
265 if curAxis(3) < -85
266     curAxis(3) = -85;
267 end
268 if curAxis(4) > 85
269     curAxis(4) = 85;
270 end
271 % Enforce longitude constrains
272 if curAxis(1) < -180
273     curAxis(1) = -180;
274 end
275 if curAxis(1) > 180
276     curAxis(1) = 0;
277 end
278 if curAxis(2) > 180
279     curAxis(2) = 180;
280 end
281 if curAxis(2) < -180
282     curAxis(2) = 0;
283 end
284
285 if isequal(curAxis,[0 1 0 1]) % probably an empty figure
286     % display world map
287     curAxis = [-200 200 -85 85];
288     axis(curAxis)
289 end
290
291
292 if autoAxis
293     % adjust current axis limit to avoid stretched maps
294     [xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
295     xExtent = diff(xExtent); % just the size of the span
296     yExtent = diff(yExtent);
297     % get axes aspect ratio
298     drawnow
299     org_units = get(axHandle,'Units');
300     set(axHandle,'Units','Pixels')
301     ax_position = get(axHandle,'position');
302     set(axHandle,'Units',org_units)
303     aspect_ratio = ax_position(4) / ax_position(3);
304
305     if xExtent*aspect_ratio > yExtent
306         centerX = mean(curAxis(1:2));
307         centerY = mean(curAxis(3:4));
308         spanX = (curAxis(2)-curAxis(1))/2;
309         spanY = (curAxis(4)-curAxis(3))/2;
310
311         % enlarge the Y extent
312         spanY = spanY*xExtent*aspect_ratio/yExtent; % new span
313         if spanY > 85
314             spanX = spanX * 85 / spanY;
315             spanY = spanY * 85 / spanY;
316         end
317         curAxis(1) = centerX-spanX;

```

```

318     curAxis(2) = centerX+spanX;
319     curAxis(3) = centerY-spanY;
320     curAxis(4) = centerY+spanY;
321     elseif yExtent > xExtent*aspect_ratio
322
323         centerX = mean(curAxis(1:2));
324         centerY = mean(curAxis(3:4));
325         spanX = (curAxis(2)-curAxis(1))/2;
326         spanY = (curAxis(4)-curAxis(3))/2;
327         % enlarge the X extent
328         spanX = spanX*yExtent/(xExtent*aspect_ratio); % new span
329         if spanX > 180
330             spanY = spanY * 180 / spanX;
331             spanX = spanX * 180 / spanX;
332         end
333
334         curAxis(1) = centerX-spanX;
335         curAxis(2) = centerX+spanX;
336         curAxis(3) = centerY-spanY;
337         curAxis(4) = centerY+spanY;
338     end
339     % Enforce Latitude constraints of EPSG:900913
340     if curAxis(3) < -85
341         curAxis(3:4) = curAxis(3:4) + (-85 - curAxis(3));
342     end
343     if curAxis(4) > 85
344         curAxis(3:4) = curAxis(3:4) + (85 - curAxis(4));
345     end
346     axis(axHandle, curAxis); % update axis as quickly as possible, before downloading new
    image
347     drawnow
348 end
349
350 % Delete previous map from plot (if exists)
351 if nargin <= 1 % only if in plotting mode
352     curChildren = get(axHandle, 'children');
353     map_objs = findobj(curChildren, 'tag', 'gmap');
354     bd_callback = [];
355     for idx = 1:length(map_objs)
356         if ~isempty(get(map_objs(idx), 'ButtonDownFcn'))
357             % copy callback properties from current map
358             bd_callback = get(map_objs(idx), 'ButtonDownFcn');
359         end
360     end
361     ud = get(axHandle, 'UserData');
362     delete(map_objs);
363     delete(findobj(curChildren, 'tag', 'MapScale'));
364     % Recover userdata of axis (cleared in cleanup function)
365     set(axHandle, 'UserData', ud);
366 end
367
368 % Calculate zoom level for current axis limits
369 [xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
370 minResX = diff(xExtent) / width;
371 minResY = diff(yExtent) / height;
372 minRes = max([minResX minResY]);
373 tileSize = 256;
374 initialResolution = 2 * pi * 6378137 / tileSize; % 156543.03392804062 for tileSize 256
    pixels
375 zoomlevel = floor(log2(initialResolution/minRes));
376
377 % Enforce valid zoom levels
378 if zoomlevel < 0
379     zoomlevel = 0;
380 end
381 if zoomlevel > 19
382     zoomlevel = 19;
383 end

```

```

384
385 % Calculate center coordinate in WGS1984
386 lat = (curAxis(3)+curAxis(4))/2;
387 lon = (curAxis(1)+curAxis(2))/2;
388
389 % Construct query URL
390 preamble = 'http://maps.googleapis.com/maps/api/staticmap';
391 location = ['?center=' num2str(lat,10) ', ' num2str(lon,10)];
392 zoomStr = ['&zoom=' num2str(zoomlevel)];
393 sizeStr = ['&scale=' num2str(scale) '&size=' num2str(width) 'x' num2str(height)];
394 maptypeStr = ['&maptype=' maptype ];
395 if ~isempty(apiKey)
396     keyStr = ['&key=' apiKey];
397 else
398     keyStr = '';
399 end
400 markers = '&markers=';
401 for idx = 1:length(markerlist)
402     if idx < length(markerlist)
403         markers = [markers markerlist{idx} '%7C'];
404     else
405         markers = [markers markerlist{idx}];
406     end
407 end
408
409 if showLabels == 0
410     if ~isempty(style)
411         style = [style '&style='];
412     end
413     style = [style 'feature:all|element:labels|visibility:off'];
414 end
415
416 if ~isempty(language)
417     languageStr = ['&language=' language];
418 else
419     languageStr = '';
420 end
421
422 if ismember(maptype,{'satellite','hybrid'})
423     filename = 'tmp.jpg';
424     format = '&format=jpg';
425     convertNeeded = 0;
426 else
427     filename = 'tmp.png';
428     format = '&format=png';
429     convertNeeded = 1;
430 end
431 sensor = '&sensor=false';
432
433 if ~isempty(style)
434     styleStr = ['&style=' style];
435 else
436     styleStr = '';
437 end
438
439 url = [preamble location zoomStr sizeStr maptypeStr format markers languageStr sensor keyStr
440     styleStr];
441
442 % Get the image
443 if useTemp
444     filepath = fullfile(tempdir, filename);
445 else
446     filepath = filename;
447 end
448 try
449     urlwrite(url,filepath);
450 catch % error downloading map

```

```

451     warning(['Unable to download map form Google Servers.\n' ...
452             'Matlab error was: %s\n\n' ...
453             'Possible reasons: missing write permissions, no network connection, quota exceeded,
           or some other error.\n' ...
454             'Consider using an API key if quota problems persist.\n\n' ...
455             'To debug, try pasting the following URL in your browser, which may result in a more
           informative error:\n%s'], lasterr, url);
456     varargout{1} = [];
457     varargout{2} = [];
458     varargout{3} = [];
459     return
460 end
461
462 [M, Mcolor] = imread(filepath);
463 Mcolor = uint8(Mcolor * 255);
464 %M = cast(M,'double');
465 delete(filepath); % delete temp file
466 width = size(M,2);
467 height = size(M,1);
468
469 % We now want to convert the image from a colormap image with an uneven
470 % mesh grid, into an RGB truecolor image with a uniform grid.
471 % This would enable displaying it with IMAGE, instead of PCOLOR.
472 % Advantages are:
473 % 1) faster rendering
474 % 2) makes it possible to display together with other colormap annotations (PCOLOR, SCATTER
   etc.)
475
476 % Convert image from colormap type to RGB truecolor (if PNG is used)
477 if convertNeeded
478     imag = zeros(height,width,3, 'uint8');
479     for idx = 1:3
480         cur_map = Mcolor(:,idx);
481         imag(:, :, idx) = reshape(cur_map(M+1),height,width);
482     end
483 else
484     imag = M;
485 end
486 % Resize if needed
487 if resize ~= 1
488     imag = imresize(imag, resize, 'bilinear');
489 end
490
491 % Calculate a meshgrid of pixel coordinates in EPSG:900913
492 width = size(imag,2);
493 height = size(imag,1);
494 centerPixelY = round(height/2);
495 centerPixelX = round(width/2);
496 [centerX,centerY] = latLonToMeters(lat, lon ); % center coordinates in EPSG:900913
497 curResolution = initialResolution / 2^zoomlevel / scale / resize; % meters/pixel (EPSG
   :900913)
498 xVec = centerX + ((1:width)-centerPixelX) * curResolution; % x vector
499 yVec = centerY + ((height:-1:1)-centerPixelY) * curResolution; % y vector
500 [xMesh,yMesh] = meshgrid(xVec,yVec); % construct meshgrid
501
502 % convert meshgrid to WGS1984
503 [lonMesh,latMesh] = metersToLatLon(xMesh,yMesh);
504
505 % Next, project the data into a uniform WGS1984 grid
506 uniHeight = round(height*resize);
507 uniWidth = round(width*resize);
508 latVect = linspace(latMesh(1,1),latMesh(end,1),uniHeight);
509 lonVect = linspace(lonMesh(1,1),lonMesh(1,end),uniWidth);
510 [uniLonMesh,uniLatMesh] = meshgrid(lonVect,latVect);
511 uniImag = zeros(uniHeight,uniWidth,3);
512
513 % Fast Interpolation to uniform grid
514 uniImag = myTurboInterp2(lonMesh,latMesh,imag,uniLonMesh,uniLatMesh);

```



```

515
516 if narginout <= 1 % plot map
517     % display image
518     hold(axHandle, 'on');
519     cax = caxis;
520     h = image(lonVect,latVect,uniImag, 'Parent', axHandle);
521     caxis(cax); % Preserve caxis that is sometimes changed by the call to image()
522     set(axHandle, 'YDir', 'Normal')
523     set(h, 'tag', 'gmap')
524     set(h, 'AlphaData', alphaData)
525
526     % add a dummy image to allow pan/zoom out to x2 of the image extent
527     h_tmp = image(lonVect([1 end]),latVect([1 end]),zeros(2),'Visible','off', 'Parent',
528     axHandle, 'CDataMapping', 'scaled');
529     set(h_tmp, 'tag', 'gmap')
530
531     uistack(h, 'bottom') % move map to bottom (so it doesn't hide previously drawn
532     annotations)
533     axis(axHandle, curAxis) % restore original zoom
534     if narginout == 1
535         varargout{1} = h;
536     end
537     set(h, 'UserData', onCleanup(@() cleanupFunc(axHandle)));
538
539     % if auto-refresh mode - override zoom callback to allow automatic
540     % refresh of map upon zoom actions.
541     figHandle = axHandle;
542     while ~strcmpi(get(figHandle, 'Type'), 'figure')
543         % Recursively search for parent figure in case axes are in a panel
544         figHandle = get(figHandle, 'Parent');
545     end
546
547     zoomHandle = zoom(axHandle);
548     panHandle = pan(figHandle); % This isn't ideal, doesn't work for contained axis
549     if autoRefresh
550         set(zoomHandle, 'ActionPostCallback', @update_google_map);
551         set(panHandle, 'ActionPostCallback', @update_google_map);
552     else % disable zoom override
553         set(zoomHandle, 'ActionPostCallback', []);
554         set(panHandle, 'ActionPostCallback', []);
555     end
556
557     % set callback for figure resize function, to update extents if figure
558     % is stretched.
559     if figureResizeUpdate && isempty(get(figHandle, 'ResizeFcn'))
560         % set only if not already set by someone else
561         set(figHandle, 'ResizeFcn', @update_google_map_fig);
562     end
563
564     % set callback properties
565     set(h, 'ButtonDownFcn', bd_callback);
566
567     if mapScale
568         makescale(axHandle, 'set_callbacks', 0, 'units', scaleUnits, ...
569         'location', scaleLocation, 'width', scaleWidth);
570     end
571 else % don't plot, only return map
572     varargout{1} = lonVect;
573     varargout{2} = latVect;
574     varargout{3} = uniImag;
575 end
576
577 % Coordinate transformation functions
578
579 function [lon,lat] = metersToLatLon(x,y)
580 % Converts XY point from Spherical Mercator EPSG:900913 to lat/lon in WGS84 Datum
581 originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244

```

```

581 lon = (x ./ originShift) * 180;
582 lat = (y ./ originShift) * 180;
583 lat = 180 / pi * (2 * atan( exp( lat * pi / 180)) - pi / 2);
584
585 function [x,y] = latLonToMeters(lat, lon )
586 % Converts given lat/lon in WGS84 Datum to XY in Spherical Mercator EPSG:900913"
587 originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
588 x = lon * originShift / 180;
589 y = log(tan((90 + lat) * pi / 360 )) / (pi / 180);
590 y = y * originShift / 180;
591
592
593 function ZI = myTurboInterp2(X,Y,Z,XI,YI)
594 % An extremely fast nearest neighbour 2D interpolation, assuming both input
595 % and output grids consist only of squares, meaning:
596 % - uniform X for each column
597 % - uniform Y for each row
598 XI = XI(1,:);
599 X = X(1,:);
600 YI = YI(:,1);
601 Y = Y(:,1);
602
603 xiPos = nan*ones(size(XI));
604 xLen = length(X);
605 yiPos = nan*ones(size(YI));
606 yLen = length(Y);
607 % find x conversion
608 xPos = 1;
609 for idx = 1:length(xiPos)
610     if XI(idx) >= X(1) && XI(idx) <= X(end)
611         while xPos < xLen && X(xPos+1)<XI(idx)
612             xPos = xPos + 1;
613         end
614         diffs = abs(X(xPos:xPos+1)-XI(idx));
615         if diffs(1) < diffs(2)
616             xiPos(idx) = xPos;
617         else
618             xiPos(idx) = xPos + 1;
619         end
620     end
621 end
622 % find y conversion
623 yPos = 1;
624 for idx = 1:length(yiPos)
625     if YI(idx) <= Y(1) && YI(idx) >= Y(end)
626         while yPos < yLen && Y(yPos+1)>YI(idx)
627             yPos = yPos + 1;
628         end
629         diffs = abs(Y(yPos:yPos+1)-YI(idx));
630         if diffs(1) < diffs(2)
631             yiPos(idx) = yPos;
632         else
633             yiPos(idx) = yPos + 1;
634         end
635     end
636 end
637 ZI = Z(yiPos,xiPos,:);
638
639
640 function update_google_map(obj,evd)
641 % callback function for auto-refresh
642 drawnow;
643 try
644     axHandle = evd.Axes;
645 catch ex
646     % Event doesn't contain the correct axes. Panic!
647     axHandle = gca;
648 end

```

```

649 ud = get(axHandle, 'UserData');
650 if isfield(ud, 'gmap_params')
651     params = ud.gmap_params;
652     plot_google_map(params{:});
653 end
654
655
656 function update_google_map_fig(obj, evd)
657 % callback function for auto-refresh
658 drawnow;
659 axes_objs = findobj(get(gcf, 'children'), 'type', 'axes');
660 for idx = 1:length(axes_objs)
661     if ~isempty(findobj(get(axes_objs(idx), 'children'), 'tag', 'gmap'));
662         ud = get(axes_objs(idx), 'UserData');
663         if isfield(ud, 'gmap_params')
664             params = ud.gmap_params;
665         else
666             params = {};
667         end
668
669         % Add axes to inputs if needed
670         if ~sum(strcmpi(params, 'Axis'))
671             params = [params, {'Axis', axes_objs(idx)}];
672         end
673         plot_google_map(params{:});
674     end
675 end
676
677 function cleanupFunc(h)
678 ud = get(h, 'UserData');
679 if isstruct(ud) && isfield(ud, 'gmap_params')
680     ud = rmfield(ud, 'gmap_params');
681     set(h, 'UserData', ud);
682 end

```

B.11 read_nc_file_struct.m

```

1 function [finfo outstrct]=read_nc_file_struct(fnam)
2 %FILENAME: read_nc_file_struct
3 %
4 %DESCRIPTION:
5 %This file contains one Matlab program that is a high level NetCDF reader. It will read any
6 %NetCDF file
7 %inputted. If the file is properly formatted, CF or another standard format, it will apply
8 %offsets and scaling
9 %automatically and will replace the fill values with NaNs. Otherwise you will need to do
10 %this manually.
11 %This program will output all of the variables in the file into a
12 %structure.
13 %
14 %USAGE: To run this program, use the following command:
15 % >> [finfo outstrct] = read_nc_file_struct('filename');
16 %
17 %INPUTS:
18 %fnam = The filename of the file you would like to read, with the full path,
19 %as a string
20 %
21 %OUTPUTS:
22 %1. finfo = File information, as a structure, with the Global and Variable Attributes
23 %
24 %2. outstrct is a structure containing all of the variables within the
25 %specified file. The first field in the structure is the filename
26 %
27 %3. This function will output Variable and Global Attributes to the screen. If you do
28 %not want this feature comment line 38.
29 %
30 %NOTES:
31 %This code was written with Matlab Version 7.13.0.564 (R2011b)

```

```

29 %
30 %19 April 2012
31 %=====
32 % Copyright (c) 2012, California Institute of Technology
33 %=====
34
35
36 % ncdisp(fnam)          %outputs Variable and Global Attributes to the screen
37
38 finfo=ncinfo(fnam); %gets Variable and Global Attribute information from the file
39 minargs=1;          %make sure that the right number of fields are inputted and outputted
40 maxargs=2;
41 outstrct=struct('filename',fnam);          %first field in the structure is the filename
42 % nargoutchk(minargs,maxargs) %check if more than just finfo is declared but make sure
    that the number of variables declared is not more than what is in the file
43
44 for i=1:length(finfo.Variables)          %loop over the number of variables in the file
45     eval(['a=ncread(fnam,'' finfo.Variables(i).Name ''');']) %reads in variable data
46     outstrct=setfield(outstrct,finfo.Variables(i).Name,a);
47     clear a
48 end
49
50 end          %ends function

```