UNIVERSITY OF HERTFORDSHIRE

Department of Computer Science

Modular BSc Honours in Computer Science

6COM1053 - Computer Science Project

Final Report

April 2021

Optimising Transatlantic Flight Paths

Erol GELBUL

Supervised by Stephen HUNT

Abstract

This report is about finding an algorithm to optimise the transatlantic flight route, departing from North America to Europe. The project aims to investigate all possible variables that affect this travel corridor, such as weather conditions, transatlantic wind maps, turbulence, airplane statistics etc. The project begins with an exploration phase with the calculation of the distance that needs to be traversed which is the great circle distance. Various trigonometry methodologies involving spherical and geodesic mathematics is introduced and one methodology is picked. After this flight path's distance is calculated, a simulation is prepared to optimise this flight path by altering the course of the plane using tailwind and avoiding headwind to reach the destination faster. Ultimately, the final algorithm outputted a flight path that aided the flight to reach its destination 6.2 minutes faster than the flight time when the plane traversed the regular great circle distance. The algorithm achieved $3,170.34 profit for the airline company.

Acknowledgements

I would like to thank my university for the MATLAB license, and special thanks to my supervisor, Stephen Hunt, for his time and help.

Abbreviations:

RHS:	Right Hand Side
LHS:	Left Hand Side
GP:	Gnomonic Projection
ATC:	Air Traffic Controller
FAA:	Federal Aviation Administration
DNF:	Did Not Finish

Contents

CHAPTER 1: Introduction

Today's technological advancements in communication, computation and many other areas are aiding the world to develop further. Any industry that touches upon science or a piece of technology is always on a path to better itself. For instance, the airline industry is constantly evolving, finding shortcuts, or benefiting from scientific discoveries to become something more. However, one area in this industry has been on a complete halt since the early 1980s. NAT-OTS is a system that has been in place for a long time, and it has worked well so far. The reason for its presence is the lack of radar over the Atlantic Ocean. This system calculates direct flight corridors over the Atlantic, following the great circle lines. With availability of technology this age-old system should be replaced. This project aims to develop an algorithm that will provide an optimised flight path to benefit from the technology present, instead of using direct flight paths, so that the need for a new system may come to attention. To achieve this, the following objectives should be achieved:

- Cover various methodologies on calculating the great circle distance.
- Create a system where many variables are considered when planning the flight path, such as:
  - Turbulence,
  - Atlantic Ocean wind maps, jet streams,
  - Weather conditions around the airports,
  - Different flight path simulations for distinct times of the year,
  - Fuel and aircraft statistics,
  - Algorithms to optimise the flight path.

The report begins with research information on subject areas in *Chapter 1.2*. Moving on from literature review, an exploration phase begins where student's personal findings are displayed on the topic of spherical trigonometry in *Chapter 2.1*. On top of that, two methodologies are shared: Haversine's formula and Vincenty's solution in the same sub chapter. The most suitable one is picked and used for calculating the distance between specific coordinates, in program code in *Chapter 2.3* where *Appendix I* is constructed. Using this distance value, a graph is plotted in MATLAB in *Chapter 4*. However, before that, *Chapter 3* acts as an explanation about the entire project, where all the potential variables are discussed and evaluated. *Chapter 4* introduces three algorithms which are designed to optimise the flight path. Ultimately, this chapter finalises with the production of *Appendix K*, a MATLAB program code. Finally, the report is concluded with the final chapter, where the results are discussed, and personal evaluation is given.

CHAPTER 1.2: Literature Review and Research

The most challenging section of the project starts with the exploration phase, where higher level of calculus is involved. For this part of the project, most of my study and experience comes from IB (International Baccalaureate) high level mathematics. Wolfram Web provides well-constructed trigonometry joined with calculus. Deakin's lecture notes from the RMIT University were especially useful to understand Geospatial methodologies. However, the perfect starting point for me was my read on Geometric Geodesy by R. H. Rapp from Ohio State University. These resources helped me understand how the calculation of great circles may be calculated using trigonometry and calculus.

Haversine's formula was straightforward, whereas Vincenty's formulae was a demanding solution for the great circle problem. It had a complex mathematical background, especially the inverse formula (direct formula was easier to understand compared to inverse). Reading Vincenty's own research paper from the 1975 helped me to get into the solution, although, the journal prepared by C.M. Thomas and W. E. Featherstone aided me to fully understand the concept so that I could briefly explain it without getting too deep into the concept, since it was an old solution.

The Federal Aviation Administration’s NAT Resource Guide was a useful guide to fully understand the reasoning behind why the aviation industry still use NAT-OTS. Reading this guide also supported my claim, that is, the industry needs a new system for transatlantic flights.

Reading two journals about climate, one from T. Woolings and M. Blackburn and the other one from R. Hall, R. Erdelyi, E. Jones and A. Scaife gave me confirmation that the flight paths over the Atlantic Ocean are getting unstable. These journals also helped me understand why jet streams are unreliable.

CHAPTER 2: Exploration

First main chapter is an introduction to the backbone of the project, great circles, which will be discussed, in-depth, in *Chapter 2.1*. *Chapter 2* from start to finish is merely an exploration and aims to achieve adequate understanding of the concept and acts as an exposition to the later chapters. Additionally, this chapter reveals the progress and evaluation of the project from the very start. To put it simply, the project's starting exploration is about a geodesic problem on a sphere. There are different methodologies available to calculate the distance between two points on the surface of a sphere. The question is which one would be more suitable for the purpose of this project? In other words, the main intent of this exploration is to identify which methodology would be the most compact and most practical tool to convert such high levels of mathematical calculations into program code. There are a lot of tools present, online, to determine the distance between any two points, however recognising the calculation steps of great circles will most definitely assist this project to reach clarity within the concept and it will be a resource to discover unnoticed variables that may be altered to optimise the results. Additional unseen variables will be disclosed in later chapters, such as the benefit of interpolation, updating the start and end points, and segmentation procedures. *Chapter 4: Optimising the Flight Path* highly benefits from the discoveries made in this chapter, therefore investigations undertaken in this chapter is necessary.

Chapter 2.1 concerns the mathematical calculations and analyses different methodologies in this field. Additionally, at the end of this chapter, the most optimal and advantageous methodology is chosen. *Chapter 2.2* will acquire the optimal method and convert it into program code. *Chapter 2.3* uses this program code to calculate the desired distance between two points specifically for this project's variables and discusses the outcomes. Furthermore, it briefly evaluates upcoming challenges and objectives.

CHAPTER 2.1: Mathematical Calculation Methods of Great Circles

The first approach for the problem of converting mathematical calculations into program code is to analyse all different variables that needs to be considered when calculating the great circle distance. Correspondingly, this chapter aims to eliminate any variables that may impose the optimisation process in the later chapters.

Chapter 2.1 begins with a brief explanation of the great circle distance and slightly touches upon similar possible explanations and concepts that may be useful for further investigation of the problem, such as gnomic projection. The chapter continues with the student's personal mathematical advancements and exploration within the problem to extrapolate individual set of rules and formulas. Furthermore, two methodologies (Haversine Formula, Vincenty Solution) are investigated and discussed. Ultimately, one method that is best fit is picked for *Chapter 2.2*.

CHAPTER 2.1.1: Great Circles

"A great circle is a section of a sphere that contains a diameter of a sphere." (Kern and Bland, 1938, p. 87) Since this project is about calculating the shortest flight path between two distinct points on the globe, it is only interested in calculating only a segment of this great circle. This part aims to explore and formulate that segment properly with regards to computation.

Before starting out on any decision-making progress on the mathematical depth of this problem, a necessary point needs to be evaluated. That is the potential use of cartesian coordinates vs spherical coordinates. Different coordinate systems offer distinct advantages and disadvantages within their use. Since the purpose of this project is to optimise the flight path and that research on its own is adequately complex, one single use of coordinate system must be designated. In other words, the program code will not grant the choice to switch between different coordinate system options since that will add another extra level of complexity. Although the most appropriate system must be adopted for the possible further development of the project.

“Cartesian coordinates are rectilinear two- or three-dimensional coordinates (and therefore a special case of curvilinear coordinates) which are also called rectangular coordinates.” (Stover and Weisstein) *Figure 1* below is an example of a cartesian coordinate system. If the program code uses this system the mathematical calculations, such as differentiation will be significantly simpler. However, use of only this system will not allow “Haversine Formula” and “Vincenty Solution” (later discussed in this chapter) to be used directly where colatitude (related to latitude) and longitude are involved. Although it is possible to convert cartesian coordinates into spherical coordinates as the code proceeds, it is impractical to implement the whole structure this way.

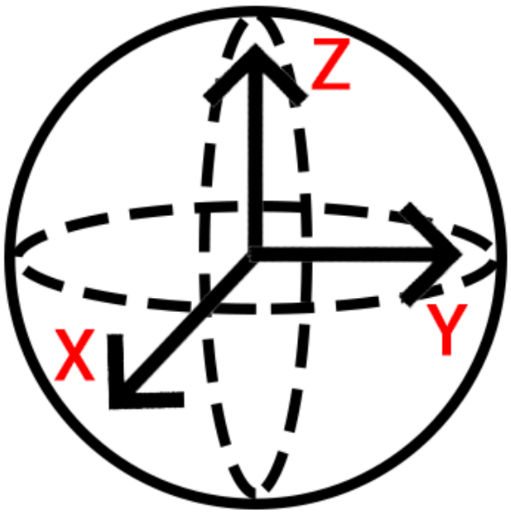


Figure 1: Cartesian Coordinates (Gelbul)

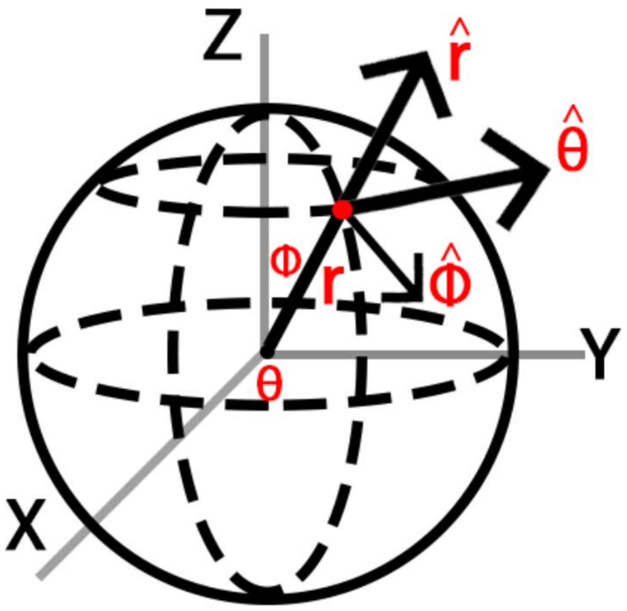


Figure 2: Spherical Coordinates (Gelbul)

Spherical coordinates on the other hand, are much more sophisticated. These coordinates are also called spherical polar coordinates (Arfken, 1985). *Figure 2* above displays how the spherical coordinates are attained (definitions and explanations are later shown in *Chapter 2.1.2*). In contrast to cartesian coordinates, they pose a higher level of complexity due to complicated differentiation nature. Nevertheless, it allows different methodologies to be used in the process and overall enhances the scope of the project. The upcoming chapter, *Chapter 2.1.2*, will delve into this contrast and aim to clarify the difference, moreover, suggest ideal paths to take when moving into program code.

CHAPTER 2.1.2: Exploration with Distinct Coordinate Systems

Instead of starting this exploration with great circles, it would be beneficial to neglect the existence of great circles and start from scratch, because the point of this exploration is to identify all possible variables and create the most simplistic environment for practical program code yet using only the most necessary piece of sophisticated work.

The problem at hand is a geodesic problem, which for this project's case, is the minimisation of the specific curvature of the Earth between two points. It is preferable to introduce this problem with a sphere, assuming that Earth is a perfect sphere, although it is an ellipsoid which is another type of sphere. (Stevens, Smith and Bianchetti)

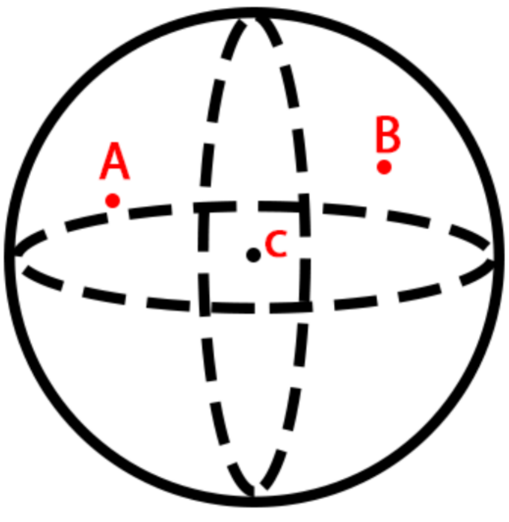


Figure 3: Points A and B on a sphere (Gelbul)

Figure 3 above displays a sphere with two random points on its surface. The geodesic problem-solving approach finds the optimal path between these two points by minimisation perception.

Start by giving the distance along the path as *D* functional. The aim is to minimise this functional.

$$D = \int_A^B dS$$

It is important to note that the sphere is a three-dimensional space, which means while applying the Pythagorean Theorem (see *Appendix A*) to *dS*, z-axis must be added to the equation, shown below.

$$dS = \sqrt{dx^2 + dy^2 + dz^2}$$

These coordinates are from the cartesian coordinate system. As previously discussed, the research deduced that spherical coordinates were going to be used. Therefore, these cartesian coordinates must be converted to spherical coordinates.

Figure 3 explains how spherical coordinates are achieved, however, below is a simplified version, *Figure 4*, is displayed.

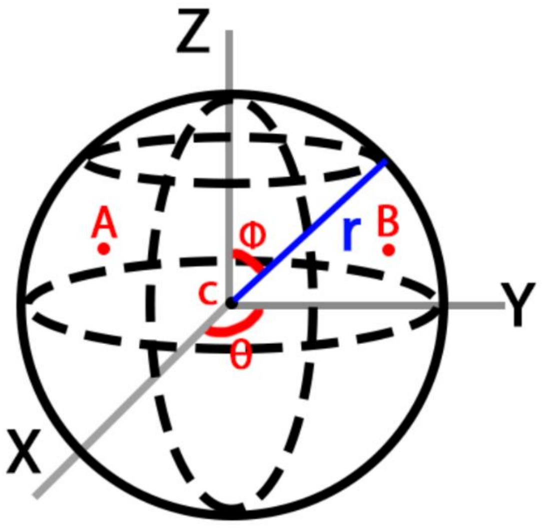


Figure 4: Spherical Coordinates 2 (Gelbul)

Notations from *Figure 4*:

r: radial distance (stays constant)
θ: angle from the positive x-axis
Φ angle relative to positive z-axis

Spherical coordinate conversion progress, giving the cartesian coordinates first:

$$x = rcos\theta\ \sin\Phi \qquad y = rsin\theta\ \sin\Phi \qquad z = rcos\theta$$

D differential requires *dx, dy, dz,* therefore, using the "Chain Rule" it is achievable.

$$dx = \frac{\partial x}{\partial\theta}d\theta + \frac{\partial x}{\partial\Phi}d\Phi \quad , \quad dy = \frac{\partial y}{\partial\theta}d\theta + \frac{\partial y}{\partial\Phi}d\Phi \quad , \quad dz = \frac{\partial z}{\partial\theta}d\theta + \frac{\partial z}{\partial\Phi}d\Phi$$

Now pop these values into the cartesian coordinates to get spherical coordinates:

$$dx = -rsin\theta\sin\Phi\ d\theta + \mathrm{r}\cos\theta\cos\Phi\ d\Phi$$

$$dy = rcos\theta\sin\Phi\ d\theta + \mathrm{r}\sin\theta\cos\Phi\ d\Phi$$

$$dz = -rsin\Phi\ d\Phi$$

The spherical values are ready, next step should be to pop these values into *dS*. This may include substantial amount of algebra, however, using the identity below in spherical coordinates will decrease the amount of work.

$$sin^2u + cos^2u = 1$$

Using this identity in *dS* gives:

$$dS = r\ d\Phi\ \sqrt{sin^2\Phi\left(\frac{d\theta}{d\Phi}\right)^2 + 1}$$

Now pop this into functional D:

$$D = \int_{\Phi A}^{\Phi B} r \sqrt{sin^2 \left(\frac{d\theta}{d\Phi}\right)^2 + 1} \;\; d\Phi$$

Designated ΦA and ΦB as the angular points. Euler-Lagrange equation (see *Appendix B*) could be applied at this point. Moreover, create a function called F:

$$F(\Phi, \theta, \theta') \quad while; \quad \theta' = \frac{d\theta}{d\Phi}$$

Integrating Euler-Lagrange gives:

$$\frac{\partial F}{\partial \theta} - \frac{d}{d\Phi}\left(\frac{\partial F}{\partial \theta'}\right) = 0$$

Function *F* can be eliminated since it does not contain θ. Remaining bit is now:

$$-\frac{d}{d\Phi}\left(\frac{\partial F}{\partial \theta'}\right) = 0$$

Next step is to integrate both sides of the equation (with respect to (w.r.t) Φ). RHS has a constant, named *G* because the integral of 0 (since it was eliminated function *F*) is a constant.

$$RHS \rightarrow \; \frac{dF}{d\theta'} = G$$

$$LHS \; \rightarrow \; -\frac{d}{d\Phi}\left(\frac{\partial F}{\partial \theta'}\right)$$

Expression for the function F is already known from functional D, above (while applying Euler-Lagrange). Function F can be used to assess $\frac{dF}{d\theta'}$, which would give.:

$$\therefore \quad LHS = \frac{\theta' sin^2\Phi}{\sqrt{(sin^2\Phi)\theta'^2 + 1}}$$

After continuous simplification (see *Appendix C*), θ' equates to:

$$\theta' = \frac{d\theta}{d\Phi}$$

$$\therefore \quad \frac{d\theta}{d\Phi} = \frac{G}{\sin\Phi\sqrt{sin^2\Phi - G^2}}$$

Need to isolate each variable. Additionally, integrate w.r.t. to Φ.

$$LHS = \theta$$

$$RHS = \int \frac{G}{sin\Phi\sqrt{sin^2\Phi - G^2}}\, d\Phi$$

Expression on the RHS is challenging to solve. However, analysing the divisor, G should be smaller than 1. If the magnitude of G was greater than 1, then the square root ($\sqrt{sin^2\Phi - G^2}$) would be imaginary. Since $sin^2\Phi$ can never go above 1. Overall, an imaginary square root would not satisfy the derivative, because the derivative must be real over the domain. Therefore, $G < 1$ (This will later be important).

From this point on, u-substitution may be applied to solve integral, by using a puppet variable $u = G\ cot\Phi \rightarrow du = -G\ csc^2\Phi\ \mathrm{d}\Phi$. Now all Φ's needs to be replaced by u.

Although, this would be pointless if the new expression did not involve sin . Therefore, it is prominent to express it by sin and Φ. *Figure 5* below, demonstrates how this can be achieved, by using a right triangle.

$$sin\Phi = \frac{G}{\sqrt{u^2 + G^2}}$$

$sin^2\Phi$ is required so:

$$sin^2\Phi = \frac{G^2}{u^2 + G^2}$$

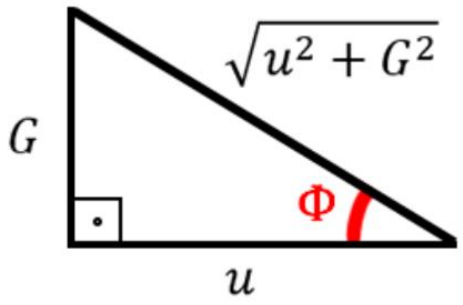


Figure 5: Attaining sin values using a right triangle. (Gelbul)

All is converted to Φ, now they can be popped into this integral:

$$RHS = \int \frac{G}{sin\Phi\sqrt{sin^2\Phi - G^2}}\, d\Phi$$

Important to note here:

$$LHS = \theta$$

Minor simplification is necessary and popping procedure:

$$\theta = -\int \frac{du}{\sqrt{1 - G^2 - u^2}}$$

$G < 1$ was previously established. Looking at the integral above, $1 - G^2$ then must be a positive number. Say a puppet variable is $\psi > 1$. $1 - G^2$ can be written as ψ^2 . This part is necessary for the next step which would make the process effortless by looking at an integration table. Using ψ^2:

$$\theta = -\int \frac{du}{\sqrt{\psi^2 - u^2}}$$

Checking an integration table to find the result will give:

$$\theta = \arccos\left(\frac{u}{\psi}\right) + \theta_1$$

Note: θ_1 is an integration constant.

In this case u-substitution was used to simplify the integral, and it is time to switch back to Φ. Since *u* was $u = G\ cot\Phi$, it can be replaced with the equation above. This would give:

$$\theta - \theta_1 = \arccos\left(\frac{G\ cot\Phi}{\psi}\right)$$

Moved the constant θ_1 to the *LHS*. Additionally, from $\left(\frac{G\ cot\Phi}{\psi}\right)$ another variable can be created. In other words:

$$\frac{G}{\psi} = \Omega$$

Finally, the equation of the geodesic on the sphere is achieved:

$$\theta - \theta_1 = \arccos(\Omega\ cot\Phi)$$

There are two constants: θ_1 and Ω. These variables can easily be calculated using the boundary conditions of A and B points on the sphere. Acquiring the values of these constants may be shown, however, it is not in the scope of this chapter.

At this point into the exploration, the equation $\theta - \theta_1 = \arccos(\Omega\ cot\Phi)$ is complicated to imagine in a three-dimensional space. Prior research into the project suggested that "gnomonic projection" should be applied if geodesic on a sphere was achieved. The next chapter will explain what gnomonic projection is and it will show why gnomonic projection cannot be used for this project.

CHAPTER 2.1.3: Gnomonic Projection

"A gnomonic projection (GP) is a bijection from the surface of a hemisphere (open) to a plane. It has the useful property that is maps great circle arcs to straight lines." (Stonelake, 2013)

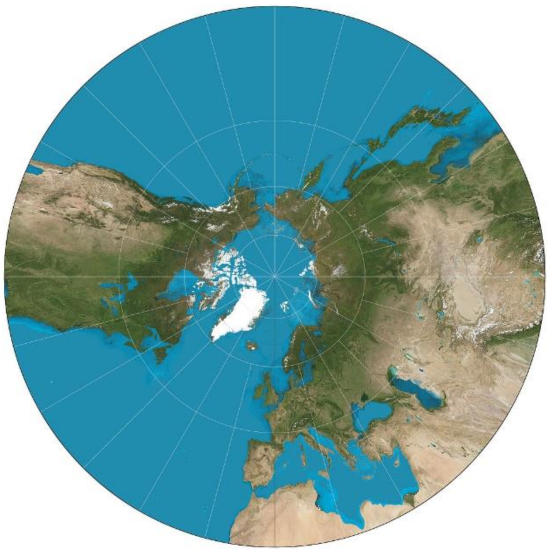


Figure 6: GP of Earth (Strebe, 2011)

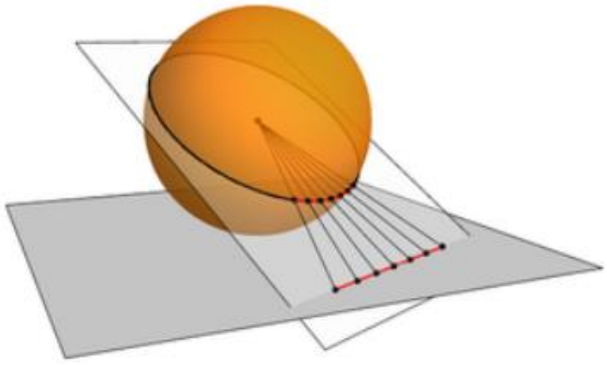


Figure 7: Great Circle Line (Marozols, 2009)

Analysing *Figure 7* above, to put it simply, if the sphere is situated on top of a plane, then the create circle arc cutting the sphere reflected onto the plane. The reflection itself can be achieved by another plane cutting through the sphere.

One of the advantages of the use of GP's would be the fact that it acquires the great circle arc and displays it in a straight line. Although this could be beneficial to the project, creating such three-dimensional space on program code will be complicated. Thus, this great circle arc appearing on the plane will need to be adjusted and amended for optimisation on *Chapter 4,* moreover, making the problem at hand more challenging.

The expandability and functionality of the project is also crucial. With, notice how the gnomonic projections can only inherit only one side of the hemisphere at a time. This may not be a problem for a flight, for instance, from London to New York as both cities are located at the Northern Hemisphere (see *Figure 6*). However, use of GP will limit potential calculations of flights between the Northern Hemisphere and Southern Hemisphere.

Overall, this chapter assisted the research with a sub-method to visualise the problem at hand. Even though, the method will not be used to the full extend, the next chapter will demonstrate a plane tangent to the sphere to provide better visualisation.

CHAPTER 2.1.4: Visualisation of the Equation

Chapter 2.1.2 finished with the equation: $\theta - \theta_1 = \arccos(\Omega\, cot\Phi)$ which was hard to visualise.

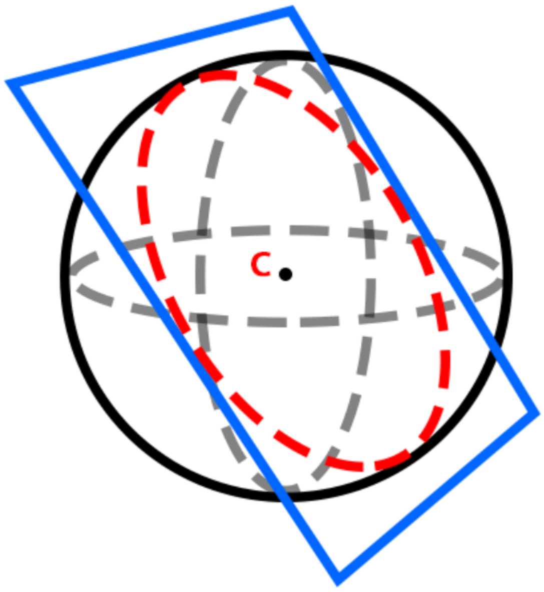


Figure 8: Plane Cutting a Sphere (Gelbul)

Inspiration from GP gave the idea to use a plane that is cutting the sphere in half and *Figure 8* demonstrates the three-dimensional visualisation. *Figure 8*'s plane is in blue, the circle formed from the cutting is in red, and rest of the sphere is from the previous representation. It is important to note that the plane passing through the sphere, intersects the sphere at the centre, *C*. In this case, the red circle formed is basically the great circle of the sphere.

CHAPTER 2.1.5: Great Circle Equation

To form some sort of an equation, it is best to start out from the equation which a plane passes through the origin. Even if the plane was not passing through the origin, the intersection will always form a circle, nonetheless. Hence, the equation of a plane passing through the origin is:

$$Ax + By + Cz = 0$$

Again, this equation does not represent spherical coordinates, they are representing cartesian coordinates. To convert the *x, y, z* into spherical coordinates similar process taken in *Chapter 2.1.2* must be applied. Expressing *x, y, z* in spherical coordinates:

$$x = rcos\theta\ sin\Phi$$

$$y = rsin\theta\ sin\Phi$$

$$z = rcos\Phi$$

The *r* is the radius that always stays constant. Now, popping these spherical coordinates into the equation of the plane:

$$\mathrm{A}\ \mathrm{r}\ cos\theta\ sin\Phi + \mathrm{B}\ \mathrm{r}\ sin\theta sin\Phi + \mathrm{C}\ \mathrm{r}\ cos\Phi = 0$$

Simplifying:

$$A\ cos\theta\ sin\Phi + \mathrm{B}\ sin\theta sin\Phi + \mathrm{C}\ cos\Phi = 0$$

$$A\ cos\theta\ sin\Phi + \mathrm{B}\ sin\theta sin\Phi = -\mathrm{C}\ cos\Phi$$

Divide both sides by sinΦ:

$$A\ cos\theta + \mathrm{B}\ sin\theta = -\mathrm{C}\ cot\Phi$$

For the *LHS* use trigonometric identities (see Appendix D):

Ultimately, using the trigonometric identities will give:

$$\theta - \theta_1 = arccos(\Omega \ \ cot\Phi)$$

The Ω constant this time is:

$$\Omega = \frac{-C}{\sqrt{A^2 + B^2}}$$

This equation, $\theta - \theta_1 = arccos(\Omega \ \ cot\Phi)$, is identical with the geodesic problem tackled in the *Chapter 2.1.2*.

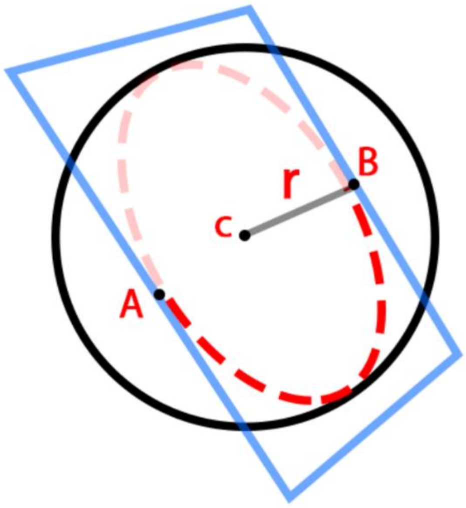


Figure 9: Geodesic and Great Circle (Gelbul)

In conclusion, addition of this concept assisted better with visualisation. Moreover, geodesic problem on the sphere is basically an arc of the great circle. While solving the geodesic on a sphere problem, the focus is to minimize this length. Depending on the boundary conditions given by the points, A and B, great circle segment can give the shortest path using the same methodology. Starting from *Chapter 2.1.1* to *Chapter 2.1.5* the idea was to explore the concepts from the root and explain the definitions such as, great circle distance, geodesic, spherical coordinates etc. to pinpoint the fundamental bullet points. The next two chapters will introduce two methodologies. Note that both chapters will be explanatory and will not include evaluation of either methodology.

CHAPTER 2.1.6: Haversine's Formula

Haversine's formula is a calculation method of great circle on a sphere. Exploration led to this chapter so far follows the exact pattern. However, instead of using raw spherical coordinates as seen previously, Haversine's formula uses longitudes and latitudes. Although, longitude and latitude are identical to the term spherical coordinates on a sphere. The haversine formula is $2\ r\ arcsin\sqrt{hav(\Delta\varphi) + cos\varphi_1 cos\varphi_2 hav(\Delta\lambda)}$. (Chen, Hsu and Chang, 2021) These variables are:

$$r = radius\ of\ the\ Earth$$

$$\Delta\varphi = (\varphi_2 - \varphi_1)$$

$$\varphi_1{:}\,Point\ A's\ Latitude \qquad \varphi_2{:}\,Point\ B's\ Latitude$$

$$\Delta\lambda = (\lambda_2 - \lambda_1)$$

$$\lambda_1{:}\,Point\ A's\ Longitude \qquad \lambda_2{:}\,Point\ B's\ Longitude$$

Longitude: responsible for east-west position of any point on a body, e.g., sphere. (Sajeevan, 2008) *Latitude*: responsible for north-south position of any point on a body, e.g., the Earth. (Sajeevan, 2008)

The hav (haversine) in the formula is $hav\ \alpha = sin^2\frac{\alpha}{2}$. (Bullock, 2007) Furthermore, the formula would be:

$$2\ r\ arcsin\sqrt{sin^2\frac{\Delta\varphi}{2} + cos\varphi_1 cos\varphi_2 sin^2\frac{\Delta\lambda}{2}}$$

Further analysis and the formula's usability will be discussed in *Chapter 2.1.9*.

CHAPTER 2.1.7: Vincenty's Solution

Thaddeus Vincenty's method uses an ellipsoidal model to calculate the distance between two points, developed in 1975 (Vincenty, 1975). The journal published by Vincenty in 1975 consists of two iterative methods: direct and inverse solutions for geodesics of any length. (Vincenty, 1975)

The inverse solution fails in the case of using immediate vicinity of any antipodal points on the ellipsoid. Points would be called antipodal if they are diametrically opposite. (Weisstein) The project specifically will not be working on any antipodal airport locations, however, a few of the transatlantic flights might occur departing and arriving at antipodal airports, therefore this should be noted.

The formulae below belong to Thaddeus Vincenty, all of the calculations below was implemented from his scientific survey review paper published in 1975, entitled "Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations". The variables/inputs are: (Vincenty, 1975)

- a and b are the semi-axis of the ellipsoid.
- $f = \left(\frac{a-b}{a}\right)$ (flattening)
- φ_1, φ_2 are the latitude values.
- L is the Δ of longitude.
- U reduced latitude.
- $\lambda = L$ (approximation)

Moreover, using trigonometric identities (see *Appendix E*).

$$tan\, U_{1/2} = (1 - f) \times tan\varphi_{1/2}$$

$$\cos U_{1/2} = \frac{1}{\sqrt{1 + tan^2 U_{1/2}}}$$

$$\sin U_{1/2} = tanU_{1/2} \times \cos U_{1/2}$$

Iterating until L (λ) becomes negligible. Such that it is a low value that will not affect the result. For instance, 0.001mm.

$$\sin \sigma = \sqrt{[(cosU_2 \ sin\lambda)^2 + (cosU_1 \times sinU_2 - sinU_1 \times cosU_2 \times cos\lambda)^2]}$$

$$cos\sigma = sinU_1 \times sinU_2 + cosU_1 \times cosU_2 \times cos\lambda$$

Normally, the foundation of this formulae in Vincenty's research involves around surplus number of trigonometric functions. However, Vincenty later devised utilisations to reduce the number of trigonometric functions. After the iteration above, the trigonometric identities used are limited to *sin* and *cos*, which makes the formulae more efficient. This implementation of formulae is beneficial for this project since it would mean less computation. Using both equations above, σ will be:

$$\sigma = atan\left(\frac{sin\ \sigma}{cos\sigma}\right)$$

$$sin\alpha = cosU_1 \times cosU_2 \times \frac{sin\lambda}{sin\sigma}$$

Using trigonometric identity will give: (see *Appendix F*)

$$cos2\sigma_2 = cos\sigma - 2 \times sinU_1 \times \frac{sinU_2}{cos^2\alpha}$$

$$C = \frac{f}{16} \times cos^2\alpha \times [4 + f \times (4 - 3 \times cos^2\alpha)]$$

$$\lambda' = L + (1 - C) \times f \times sin\alpha \ldots$$

$$\times \{\sigma + C \times sin\sigma \times [cos2\sigma_m + C \times cos \times (-1 + 2 \times cos^2\sigma_m)]\}$$

$$\left(u^2 = \frac{cos^2\alpha \times (a^2 - b^2)}{b^2}\right)$$

$$A = 1 + \frac{u^2}{16384} \times \{4096 + u^2 \times [-786 + u^2 \times (320 - 175 \times u^2)]\}$$

α_1 is the starting bearing (Vincenty, 1975):

$$\alpha_1 = atan\left(\frac{cosU_2 \times sin\lambda}{cosU_1 \times sinU_2 - sinU_1 \times cosU_2 \times cos\lambda}\right)$$

α_2 is the final bearing (Vincenty, 1975):

$$\alpha_2 = atan\left(\frac{cosU_1 \times sin\lambda}{-sinU_1 \times cosU_2 + cosU_1 \times sinU_2 \times cos\lambda}\right)$$

Name the distance between these two bearings as *d* (Vincenty, 1975):

$$d = b \times A \times (\sigma - \Delta\sigma)$$

CHAPTER 2.1.8: Earth's Radius

Before reaching the calculation and computation phase in *Chapter 2.2,* discussing Earth's radius is prominent. According to Goddard Space Flight Centre, Earth's radius directly on the equator is 6378km. (Sharp, 2017) Generally used value used is 6371km since it is the average radius.

The accuracy of this variable can cause crucial changes in results; therefore, Earth's radius was further investigated. As Vincenty laid out the fact that the Earth is not a perfect sphere but an Ellipsoid, where polar radius and the equator radius changes drastically, it is necessary to calculate the specific radius at the right latitude. RASC Calgary Centre's explanation of this concept introduces the following formula (McNish, 2005):

$$r = 6378 - 21 \times \sin(U')$$

See *Appendix G* for another version of the formula where equator radius should be provided in miles.

Where, U' is the latitude of the location. Since the aim is to find out the distance between two airports one of which is on the other side of the Atlantic Ocean, if both the latitude of both airports may be retrieved and then the average latitude is calculated, this value may be used for the U' variable. For the specific case of this project, *Heathrow Airport* (LHR) situated in London and *John F. Kennedy International Airport* (JFK) situated in New York City should be used.

According to latlong.net, LHR is 51.470020° and JFK is 40.641766°. (Airports, 2012) The average of these two values is 46.055893°. In this case, the average value should be plugged into U' to calculate the radius.

$$r = 6378 - 21 \times \sin(46.055893°)$$

$$r = 6362km$$

For the rest of the report all calculations regarding to great circle distances, the value r will be used.

CHAPTER 2.1.9 Evaluating Methodologies

This section will be evaluating and discussing which methodology, displayed earlier in this chapter, is the most suitable for the purpose of this project.

There are three main points to consider before picking any methodology: the depth of and the complexity of the program code that needs to be written, how fast the program code will run, and accuracy.

The complexity of the computational work must still be evaluated for the expandability of the project. The project may be a steppingstone for other flight paths other than the transatlantic route, therefore, it is prominent that the usability of the method is investigated. Student’s work in Chapter 2.1.2 is specific with complex mathematical calculations that has the lowest chance to mitigate unwanted calculation errors as it is straightforward unlike Vincenty’s method. One major issue with Vincenty’s inverse solution is that it does not give any result in the case of both locations’ being on or near antipodal points, which is very much possible for some two airports on the entire globe. Haversine’s formula on the other hand, also poses minor issues near antipodal points, however, these issues are due to floating points. Despite that, Haversine can provide results given the rare rounding errors, unlike Vincenty’s inverse method. However, it is worth mentioning again that for the purpose of this project, antipodal points do not cause an issue since the airports are not antipodal locations.

Second necessary aspect to consider is the speed of the program code that will be ran. For all three methodologies discussed previously, time tests were running on an Intel Core i7 PC. The longest test took 16µ ± 0.1n and the fastest test took 4µ ± 0.1n. While the time it takes for the calculations might appear insignificant, it matters when the dataset is immense. The value of time is emphasised later in *Chapter 3.1*. To relay the time test comparison between all three methodologies, *Table 1* is provided below. The test includes 8 different transatlantic flight paths (see *Appendix H* for the airport IATA-Codes). Each flight path distance was calculated 10 times time for each methodology. The mean of each calculation is given for each methodology and flight path. The test count was limited to 10 since the deviation was only 0.3 nanoseconds(n).

	Student's Great Circle Method	Haversine's Method	Vincenty's Method
JFK - LHR	8µ	4µ	14µ
BOS - LHR	9µ	4µ	13µ
BWI - LHR	8µ	4µ	16µ
MIA - LHR	8µ	4µ	14µ
LGA - LHR	8µ	4µ	13µ
EWR - LHR	9µ	5µ	16µ
DCA - LHR	8µ	4µ	16µ

Table 1: Time Test of three Methodologies (Gelbul)

Vincenty's distance calculation method took between 13-16 microseconds (≈14µ ± 0.1n) on program code. It is important to note that this calculation efficiency is remarkably fast compared to other calculation methods in the early 1970s. The efficiency in the methodologies used in his work is since the computational technology hardships in his time were prominent, therefore, Vincenty crafted his formulae with computationally economical due to computer parts beings relatively costly. Even though the method shows efficient code, due to Vincenty's inverse formulae's iterative nature, it is more than triple the amount of time that of Haversine's method. Another fact that is important to mention is that the student's method and Haversine's method was able to stay on a reliable time cap for each flight path, meaning that both methods only deviated 1µ at max, whereas Vincenty's does not appear reliable, with a 3µ change on different flight paths. As previously mentioned, this issue might not present a potential reliability issue for small datasets and test, however, a larger dataset will require reliability. (later discussed in *Chapter 3.1*)

The final point to investigate is accuracy and it plays a paramount role in the base of this project, because 10 kilometres of misrepresentation in the great circle may cost a lot of profit for the airline company. Therefore, accuracy can be used a tool solidify the profit expectation and probability for the airline companies. The first 10 tests on flight routes; JFK – LHR, BOS – LHR, BWI – LHR, alone, took a lot of time to conduct, using Box-Whisker Analysis for relative percentage error. It was unfeasible to complete the rest of the tests. This was caused by the processor on machine being insufficient. In a research paper titled "Comparison of Principal Geodetic Distance Calculation Methods for Automated Province Assignment in Turkey" by Istanbul Technical University students, they were comparing different geodesic distance calculation methods including Vincenty's formulae and Haversine's formula (*arcsin* version which is the one that was discussed in Chapter 2.1.6 and not the *arctan* version), and their percentage error. (Esenbuğa et al., 2016) Below is a figure from their research paper, relaying out the relative error percentage for the given methodologies.

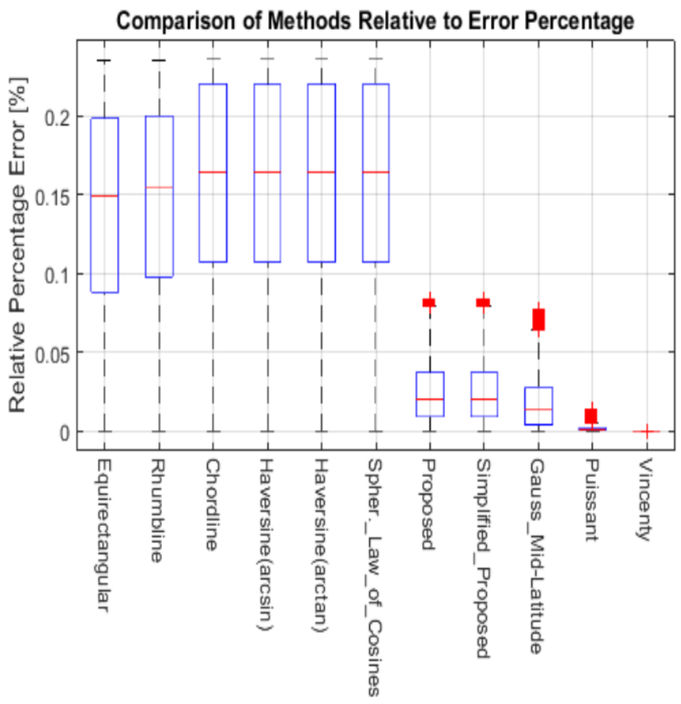


Figure 10: Box-Whisker Analysis for Relative Percentage Error for each method (Esenbuğa et al., 2016)

Figure 10 displays that Vincenty's method resulted with 0% relative error percentage. On the other hand, Haversine's *arcsin* version achieved ≈0.16% relative error percentage. To understand the difference, an example might be necessary. Distance from JFK to LHR is approximately 5536km (calculated previously using longitude and latitude of both coordinates using Vincenty's formulae). With ≈0.16% relative error, Haversine's is 8.85km inaccurate.

To conclude, student's approach was a good baseline to start off with the investigation in the field, however, it is not a well thought out method to be used for this project. Vincenty's method is the most accurate method of them all. Whereas, Haversine's method has shown 8.85km of misjudging, although, that means 99.84% accuracy which is not going to be a major issue. Haversine's computing time performed more than three times better than Vincenty's. In a system where time will be more essential than 0.16% inaccuracy potential, Haversine's method is more appropriate for this project.

CHAPTER 2.2: Implementing Great Circles Calculation to Program Code

This chapter starts out with the explanation of the Haversine's method in practice. Additionally, it refers to the antipodal points which results with an error and clarifies how that can occur, as well as that, it mentions about the Haversine's *atan* version, which was seen in the previous chapter. Then, a pseudocode is provided for the method. Finally, the code is provided and the answer from the first part is compared with the output given by the program code.

Haversine formula acquired from *Chapter 2.1.6*:

$$2\,r\,arcsin\sqrt{sin^2\frac{\Delta\varphi}{2}+cos\varphi_1 cos\varphi_2 sin^2\frac{\Delta\lambda}{2}}$$

The formula above comes from the relation between the distance of the two points and the angle for the sphere. That relation encompasses the *Central Angle* (see *Figure 11*).

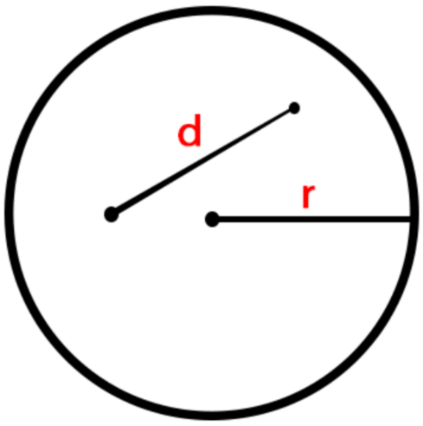


Figure 11: r is the radius, and d is the distance between 2 points.

Central angle θ equates to $\frac{d}{r}$. This central angle is calculated using the Haversine formula, which calculates the Haversine of θ. Haversine of θ is the square root section of the formula above, which is:

$$hav\ (\theta) = sin^2 \frac{\Delta\varphi}{2} + cos\varphi_1 cos\varphi_2 sin^2 \frac{\Delta\lambda}{2}$$

This was formed using the trigonometric identity:

$$hav(\theta) = sin^2 \left(\frac{\theta}{2}\right)$$

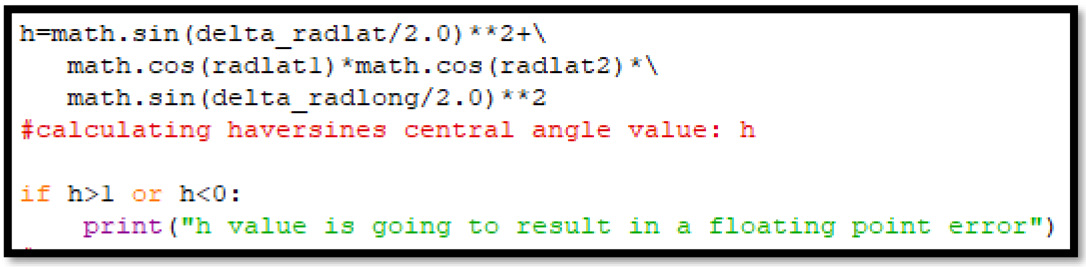
$$sin^2 \left(\frac{\theta}{2}\right) = \frac{1 - cos\theta}{2}$$

To find the distance between these two points, inverse haversine is plugged in to the expression above, which gives:

$$d = r\ \times archav(h)$$

$$\therefore 2\ r\ \text{arcsin}\ (\sqrt{h})$$

archav is inverse haversine. If the value h gets closer to 1, that would mean that the points are antipodal or at least very near antipodal. Moreover, if the value is greater than 1, it will result in a floating point error. This should be taken into consideration as a check-up within the program for safety. See *Code Snippet 1* below.



```
h=math.sin(delta_radlat/2.0)**2+\
   math.cos(radlat1)*math.cos(radlat2)*\
   math.sin(delta_radlong/2.0)**2
#calculating haversines central angle value: h

if h>1 or h<0:
    print("h value is going to result in a floating point error")
```

Code Snippet 1: Safety check, outputs error message

Now that the methodology is discussed and safety concerns were checked, the next step is to form a pseudocode.

```
INPUTS: φ_1, λ_1, φ_2, λ_2, R←6362000

OUTPUT: Haversine

[1] h ← $sin^2\left(\frac{1}{2}\varphi_1 - \frac{1}{2}\varphi_2\right) + \left(cos\varphi_1 cos\varphi_2 sin^2\left(\frac{1}{2}\lambda_1 - \frac{1}{2}\lambda_2\right)\right)$

[2] c ← $2\text{asin}(\sqrt{h})$

[3] IF $(h < 0)$ $OR$ $(h > 1)$THEN [4] ← PRINT ERROR

[4] Haversine ← $R \cdot c$
```

The pseudocode above displays the arcsin version. If arctan version had to be used, then [2] must be changed to:

$$[2]\ c \leftarrow 2\text{atan}\left(\frac{\sqrt{h}}{\sqrt{1-h}}\right)$$

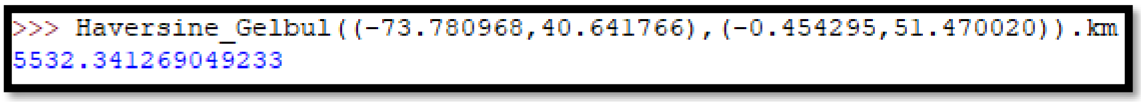
Below are the values required:

- JFK’s latitude is 40.641766° and longitude is -73.780968° . (Airports, 2012)
- LHR’s latitude is 51.470020° and longitude is -0.454295°. (Airports, 2012)
- $r = 6362km$ (*Chapter 2.18*)

Converting degrees to radians for both airports (this is not a necessity, however inputting these values into the formula must stay consistent throughout the calculation procedure):

- JFK: 0.7093rad (φ_1), -1.2877rad (λ_1)
- LHR: 0.8983rad (φ_2), -0.0079rad (λ_2)

The program code converts degrees to radians by itself. See *Appendix I* for the full code with comments.



```
>>> Haversine_Gelbul((-73.780968,40.641766),(-0.454295,51.470020)).km
5532.341269049233
```

Code Snippet 2: Calling the class *Haversine_Gelbul* with the coordinates of both airports.

After running the python script to calculate the Haversine distance, see *Code Snippet 2* above, the code outputted 5532.3km. Using Vincenty’s formulae resulted with 5537,1km. The difference is 4.8km to be precise, more discussion on this outcome will be discussed in the next chapter.

CHAPTER 2.3: Exploration Results

Chapter 2 on its own was the first step to explore the mathematical difficulties and challenges that were coming up. Additionally, it revealed what would be practical for the project, for instance which methodology was the best fit for calculating transatlantic flight paths and how flexible would that methodology be. The reason why Chapter 2 comes before

Chapter 3 is because this chapter was responsible for introducing the complexity of the project and it explored what background preparation was required for the actual formation of a program that was going to be built later (in Chapter 4).

One important thing to mention from the findings of this chapter is that Earth is not a perfect sphere, it is an ellipsoid shape. This information required the student to research a method that included the consideration of Earth being an ellipsoid, whereas Vincenty considered that in his formulae. However, from the time test taken conducted between methodologies, revealed that Vincenty's formulae takes 3x more to compute than Haversine's formula. Vincenty's solution also displayed unreliable time variance between different tests. Another downside of Vincenty's formulae was that it was overly complex. However, Haversine's formula was straightforward. On the other hand, Haversine's only downside was that it did not consider the Earth as an ellipsoid, but a sphere. This problem could be improved but not fully solved, by amending Earth's radius to a more proper value, since Earth's radius is longer on the equator and short on the poles. This was achieved by averaging the latitudes of both airports and creating a more suitable radius for the calculation. The expected relative percentage error for Haversine's was ≈0.16%, being 8.85km inaccurate. Although only 4.8km relative error was identified. Nearly, half of the expected error. Ultimately, Haversine appears to be the most suitable method for this project.

Next chapter will be about the variables that come into play when transatlantic flights occur. One of which is the departure airport and destination airport. Chapter 3 will be mentioning why this project will not be needing both airports' coordinates, since the project will only be concerned with the distance where cruising altitude occurs, this will be satisfied with the findings of this chapter. However, the code and the calculations given in this chapter will still be used for other coordinates.

CHAPTER 3: Transatlantic Flight Paths and Variables

According to Statista, 91 million passengers flew from the United States over the Atlantic in 2020 (Statista, 2021). Chapter 3 delves into how these flights are operated and what are the critical variables that needs to be acknowledged to make efficient and beneficial changes. First and foremost, the upcoming section briefly explains how the flight paths are generated and what specific flight path, heading, and time this project will be focusing on and why. The succeeding chapter then mentions all the possible variables that may come into play. Finally, one variable is chosen and discussed for simulation in *Chapter 4*.

CHAPTER 3.1: Regular Flight Paths

Completing a full journey over the Atlantic Ocean is not an elementary event. Flights over the Atlantic Ocean are increasing every year according to Statista. Correspondingly increasing the amount of traffic. The only problem that complicates this journey is the lack of radar available over the Atlantic. Normally, air traffic controllers (ATC) are located on the ground. Shortage of radar is compensated by North Atlantic Organized Track System (NAT-OTS). (FAA The North Atlantic NAT Resource Guide, 2021) To explain briefly, everyday four Air Traffic Planning Centres generate routes over the Atlantic. 2 of them situated in North America, 1 of them in Ireland, and the other one in Santa Maria Oceanic Centre (FAA The North Atlantic NAT Resource Guide, 2021). When flying over the Atlantic Ocean, pilots must report their location every 14 minutes to ensure safety. Daily generation of flight paths has been around to this day since the early 1950s. (FAA The North Atlantic NAT Resource Guide, 2021) See *Figure 12* below for a prepared NAT-OTS pathing by Shanwick Air Traffic Planning Centre.

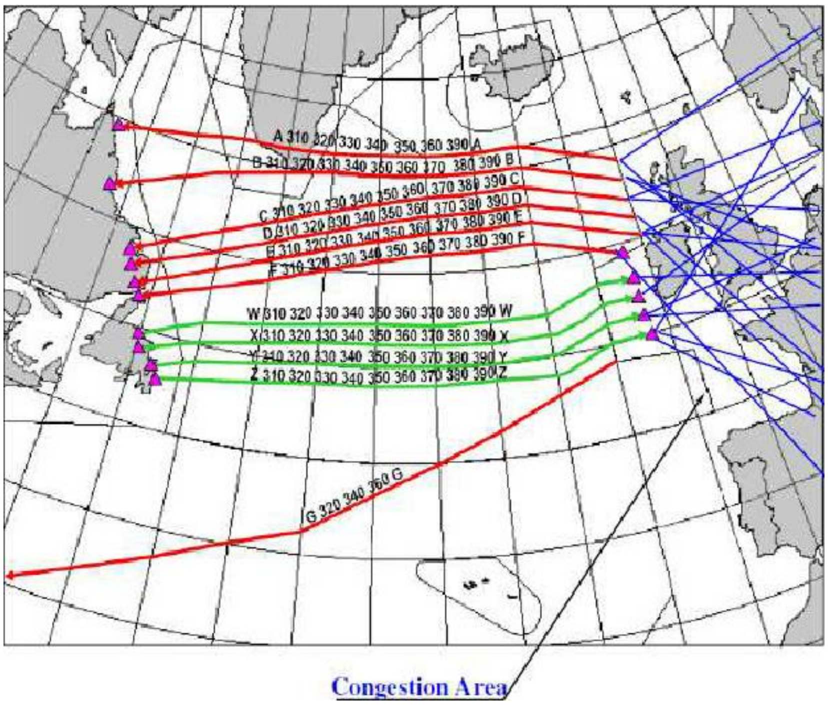

A 310 320 330 340 350 360 390 A
B 310 320 330 340 350 360 370 380 390 B
C 310 320 330 340 350 360 370 380 390 C
D 310 320 330 340 350 360 370 380 390 D
E 310 320 330 340 350 360 370 380 390 E
F 310 320 330 340 350 360 370 380 390 F
W 310 320 330 340 350 360 370 380 390 W
X 310 320 330 340 350 360 370 380 390 X
Y 310 320 330 340 350 360 370 380 390 Y
Z 310 320 330 340 350 360 370 380 390 Z
G 320 340 360 G
Congestion Area

Figure 12: Example of NAT-OTS. (Rodionova et al., 2014)

Planes traversing the Atlantic, are separated by ≈60km in distance, ≈40km laterally, and vertically 305m at minimum. (FAA The North Atlantic NAT Resource Guide, 2021) Although this method did not have any fatal incidents yet, it might cause problems in the future. *Chapter 3.3* explains the reason behind this problem, and that is climate change. *Figure 12* displays pre-planned straight routes that do not act on changing weather conditions. Student's proposed solution is to have a system in place that acts upon changing weather condition or at least is not a straight route that benefits from the weather. This topic will be discussed in depth in *Chapter 3.3*.

This project specifically focuses on a flight from JFK International Airport to London Heathrow Airport. The time and date of this flight is fixed (more on *Chapter 3.3*).

CHAPTER 3.2: Variables

The project started out with the idea to include all the variables into the simulation. Although, creating a multi-layered variable system caused the focus of the project to shift drastically to other areas which required substantial amount of research. Therefore, the following variables were removed from the equation: flight occurring at different times of the year, weather conditions over and near the airports, wind difference caused by other aircrafts in the vicinity, delays in flights caused by airline issues and turbulence factors.

Previously mentioned case of using different start and end points for the transatlantic flights in *Chapter 2* is decided on this section. This is caused by the availability of ATCs near the airports. According to FAA, NAT-OTS flight corridors for a generic JFK to LHR starts at the waypoint PORTI (47°N, 0.5°W) and ends at the waypoint XETBO (52°N, 14°W). (FAA The North Atlantic NAT Resource Guide, 2021) Consequently, the distance must be calculated again with these inputs.

Beforehand, there is another point that needs to be amended. The simulation will be fully based on cruising speed, in other words, air speed will be used, and not ground speed. This slight change alters the distance that needs to be traversed.

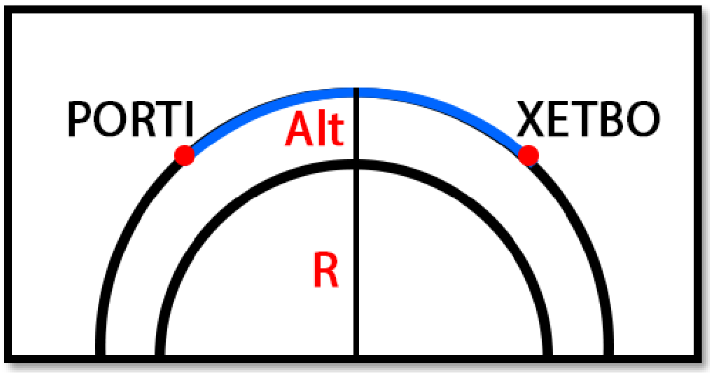
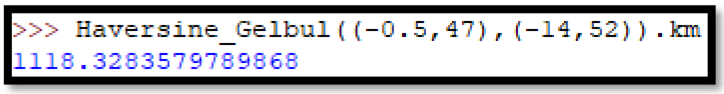


Figure 13: Actual flight route being traversed (Gelbul)

Figure 13 above represents the actual flight path in blue. *R* is the radius of Earth, and *Alt* represent the cruising altitude. According to FAA's transatlantic flight regulations, cruising altitude for a typical aircraft ranges from 34,000 – 42,000 feet (10.3 – 12.8km) however, FAA states that the average tends to be 35,000 feet (10.7km). Therefore, the radius of the Haversine calculation should be:

$$6362.0 + 10.7 = 6372.7km$$

Replacing this value for Earth's radius and inputting the PORTI and XETBO coordinates outputs 1118.3km as the distance (see *Code Snippet 3*).



```
>>> Haversine_Gelbul((-0.5,47),(-14,52)).km
1118.3283579789868
```

Code Snippet 3: Calling the class *Haversine_Gelbul.* (Gelbul)

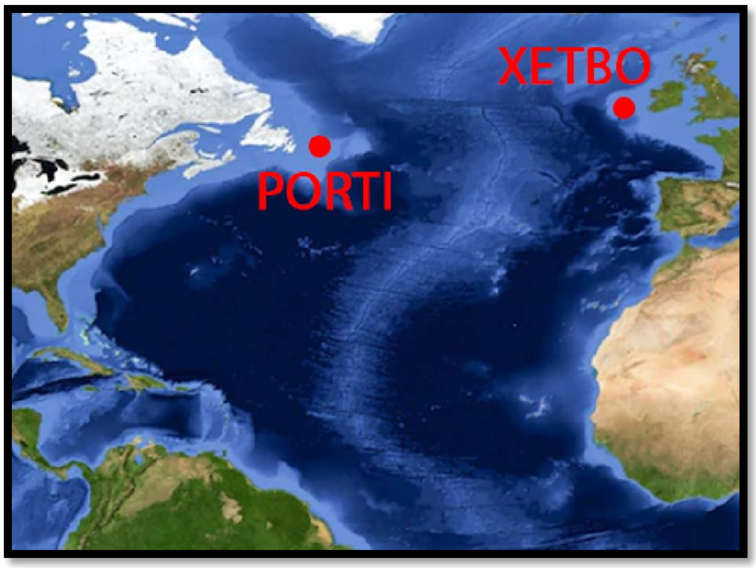


Figure 14: Waypoints PORTI and XETBO on the map. (Gelbul)

The distance now is 1118.3km which is considerably smaller compared to the first calculation (Previously calculated distance was 5532.3km). See *Figure 14* above.

According to statistics provided by the FAA the most common commercial airplane model that crosses is the Airbus A330 (for 2019, no valid data was available for 2020 for this statistic). (FAA The North Atlantic NAT Resource Guide, 2021) The cruising speed is another important variable to consider because the simulations will be using this variable. Airbus reported that their A330-300 performs between 820-910km/h (540-565mph) over the

Atlantic with a cruising altitude of 35,000 feet. (A330-300, n.d.) Airbus provides information about the maximum weight their airplane is capable of handling. Therefore, a random transatlantic flight was picked, and its statistic were analysed. The aircraft's total weight including all the passengers on board, luggage weight and fuel was an estimate of 148,680kg. (A330-300, n.d.)

The next and final variable to explore is the weather conditions over the Atlantic Ocean which is discussed in the following chapter since it is the chosen variable that will form the core problem to tackle for the algorithms in *Chapter 4*.

CHAPTER 3.3: Atlantic Ocean Weather

As previously mentioned in *Chapter 3.2*, the cruising altitude for commercial airplanes are estimated to be 35,000 feet, therefore, Atlantic Ocean weather conditions was researched around this altitude.

"A traditional view (e.g. Krishnamurti, 1961) is that there are two main jet streams in the troposphere of the Northern Hemisphere, which are relatively thin 'ribbons' of high velocity air moving eastwards near the tropopause." (Hall et al., 2014) International Journal of Climatology explains jet streams as fast currents of air, blowing from west to east. (Hall et al., 2014) Jet streams fall perfectly into variables that alter the commercial flights over the Atlantic Ocean, since jet streams can reach their most effective state around 30,000-40,000 feet according to International Journal of Climatology. (Hall et al., 2014) The concept of jet streams can be a useful tool for airplanes to benefit from. For instance, acquiring a tailwind assist from a jet stream while going over the Atlantic from JFK to LHR, it may drastically reduce the flight time. However, there are cases where the usage of tailwind almost caused an incident on the NAT-OTS. On February 19, 2019, a Boeing 787 was benefitting from a jet stream that assisted the aircraft to reach 1288km/h (321km/h over the normal limit). (Cappucci, 2019) This was an unexpected event, and could easily cause a potential collision with the aircraft that was in front on the NAT-OTS corridor, since each aircraft has only about a 60km distance (*Chapter 3.1*). This unreliability is caused by the ramifications of the climate change that has been occurring since early 2000s. According to the Journal of Climate Volume 25 and the International Journal of Climatology, "increased sheer over the North Atlantic jet stream has increased over the last four decades".

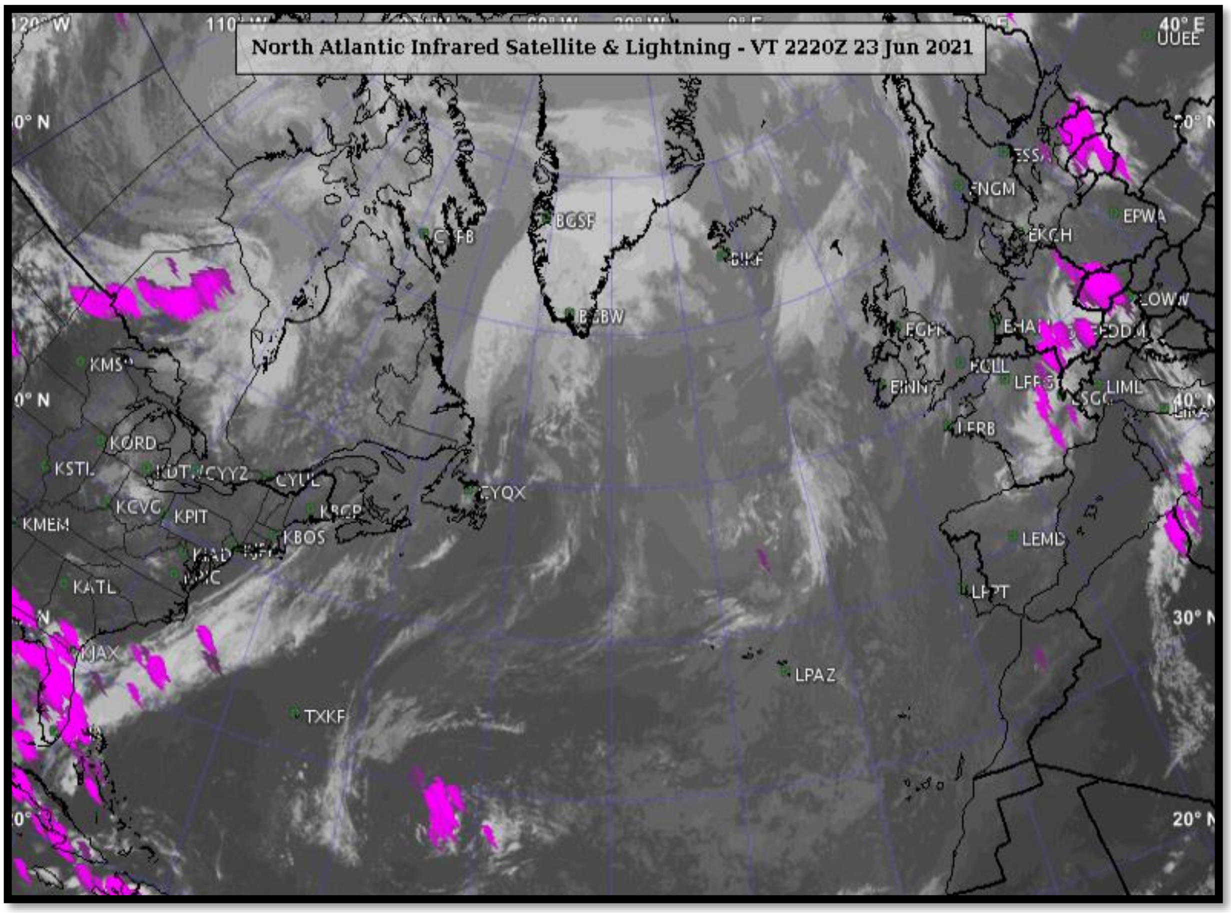


Figure 15: North Atlantic Infrared Satellite & Lightning – 23 Jun 2021 (Jeppesen)

Another point to note is about jet streams, apart from them displaying erratic changes and being unreliable, is the fact that jet streams demonstrate different results at different times of the year, and it is "constantly shifting" depending on the position of the sun. Clear air turbulence incident reports have been increasing according to the FAA over the North Atlantic Ocean.

See *Appendix J* for wind maps, by WAFC London, over the Atlantic Ocean at distinct universal coordinate time on each image. The solution for this unreliable and changing problem will be discussed in the following chapter.

CHAPTER 4: Optimising the Flight Path

The issue with NAT-OTS is that it is a fixed, straight flight corridor generation system that is from the 1980s. With today's technology and communication advancements, these flight corridors can turn into interpolated and reactive flight paths computed by complicated piece of technology. *Chapter 4* will start out with introducing the simulation environment and it will briefly explain the optimisation interface in MATLAB. Afterwards, the weather data will be discussed, and the implementation will be explained. Following the description of the program, the three algorithms produced will be shown and evaluated.

CHAPTER 4.1: The Interface

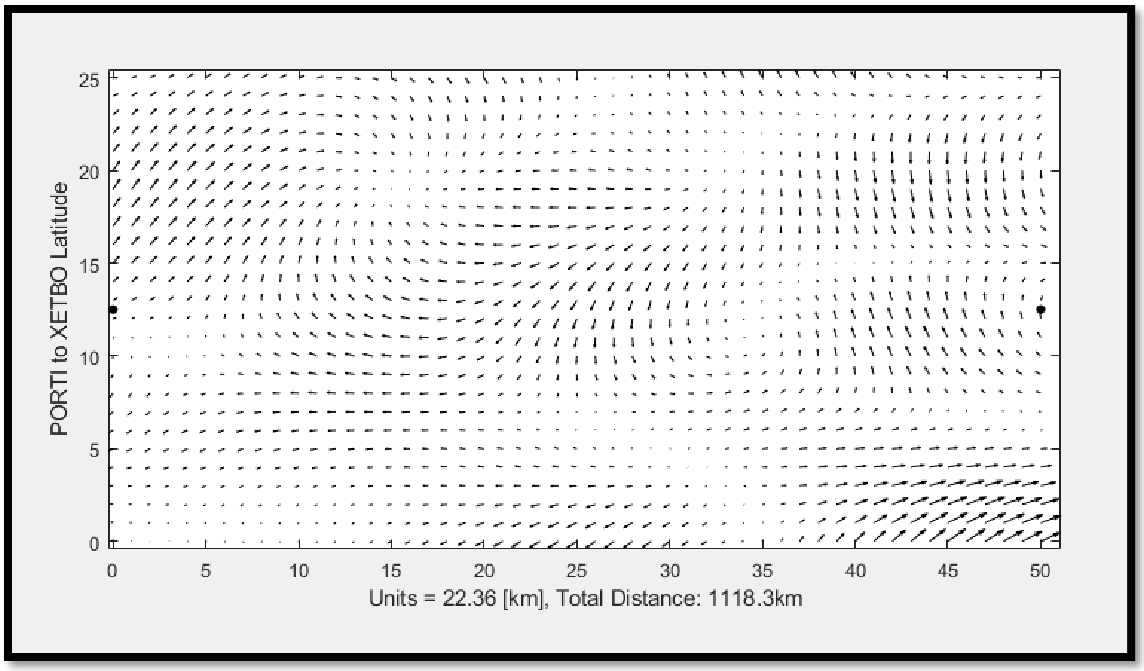


Figure 16: MATLAB plotted graph. (Gelbul)

The main idea was to mimic a world map with exact latitudes and longitudes on the graph and plot the start and end waypoints (PORTI and XETBO), however this task caused multiple problems and the results were inaccurate. If the world map was to be projected, then the great circle shown would not be a straight line, but it would rather be a curved line. On top of that, the implementing a wind map would mean that the headwind and tailwind calculations would have to be proportionate to the curvature of the great circle. Simulating wind maps over a great circle curvature caused unexpected interpolation to occur, displaying inaccurate results, nonetheless. Therefore, *Figure 16*, is a showcase of the plotted graph, where the x-axis displays the great circle distance on a straight line, divided into 50 points, where each point is separated by 22.36 km. The total distance between the two points plotted in black is 1118.3 km, calculated in *Chapter 3*. The y-axis represents the extension of the wind map laterally.

CHAPTER 4.2: Wind Map

As climate change develops more imbalance and unreliability over the Atlantic (*Chapter 3.3*) the new system requires an optimisation process where the weather conditions are random at 35.000 feet. A system that can adapt and understand the weather conditions or avoid hazardous flight routes. Thus, a random weather map is required to perform tests on a newly designed optimal flight path.

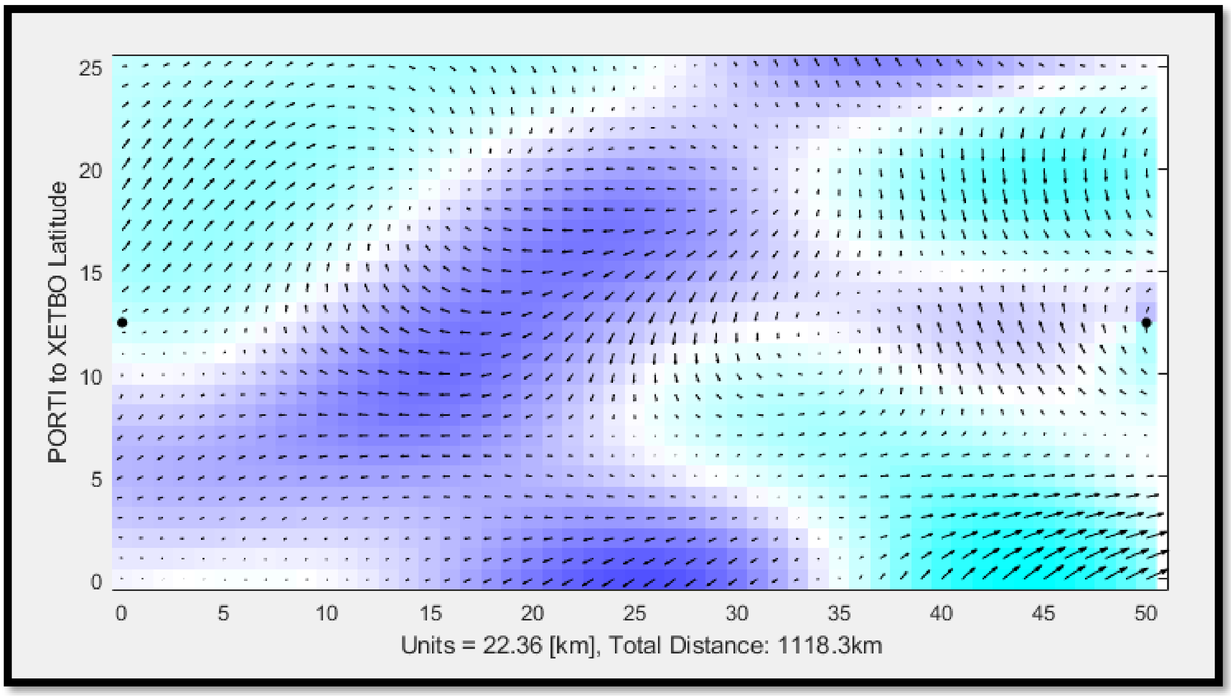
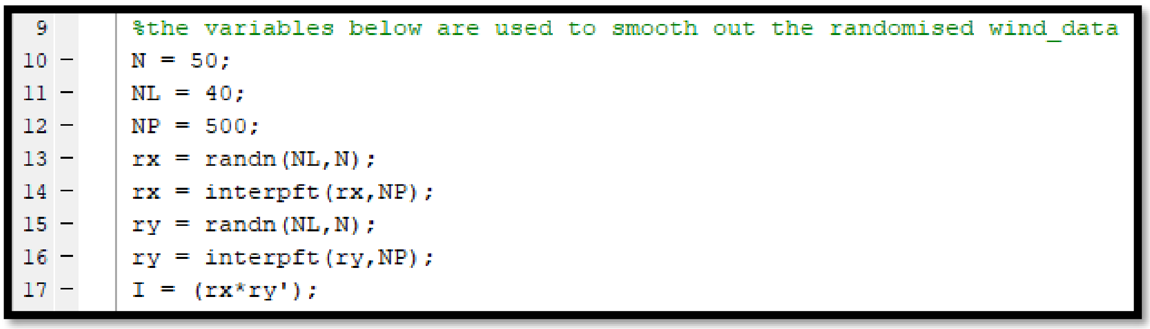


Figure 17: MATLAB plotted graph with wind data. (Gelbul)

Figure 17 above represents the random wind data over the Atlantic. This wind data can be randomised by changing the *rng(value)*, which is the random seed generator for the wind maps. Also, another technique used for this generation was to call the wind map twice on both x and y axis. The fineness and the smoothness of the wind can be altered by changing the values in the Wind_Data subscript, see *Figure 18* (All MATLAB files are in *Appendix K*).



```
%the variables below are used to smooth out the randomised wind_data
N = 50;
NL = 40;
NP = 500;
rx = randn(NL,N);
rx = interpft(rx,NP);
ry = randn(NL,N);
ry = interpft(ry,NP);
I = (rx*ry');
```

Figure 18: Wind_Data subscript’s values. (Gelbul)

The colour code generated for the wind is calculated beforehand. The cyan coloured areas are where tailwind is available to increase speed. White areas represent neutral and blue areas are headwind is present and must be avoided.

The Wind_Data subscript and the calculation method for tailwind and headwind was inspired by Mapping and Geospatial Data Analysis Using MATLAB by Alan Hwang, (Hwang, 2009) see *Figure 19*.

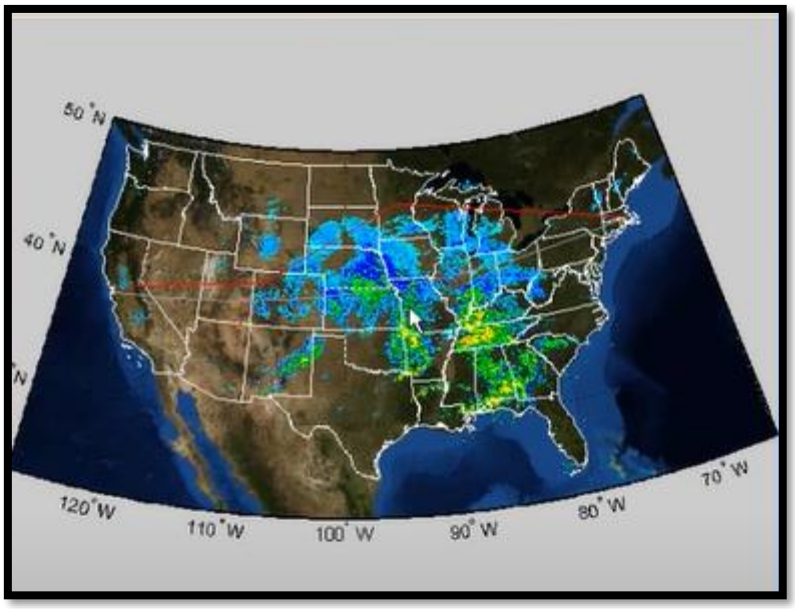


Figure 19: Weather avoidance over North America. (Hwang, 2009)

Chapter 4.3 and 4.4 will merely explain the algorithms used to reach XETBO from PORTI faster using the wind maps and great circle distance. Chapter 4.5 will ultimately explain the chosen algorithm with details about how the code works.

CHAPTER 4.3: K-Nearest Neighbour Inspired Algorithm

The following algorithm was inspired by the KNN algorithm. The plotted graph inside MATLAB is converted into a rectangular grid by using *meshgrid*. As it can be seen in *Figure 20*. This is a way to determine each point's specific wind data on the graph. Each point was given a value between 0 and 1. If the value were closer to 0 that would mean the point had headwind (unwanted wind, blue) properties; if the value was closer to 1, point had tailwind (preferred, cyan) properties.

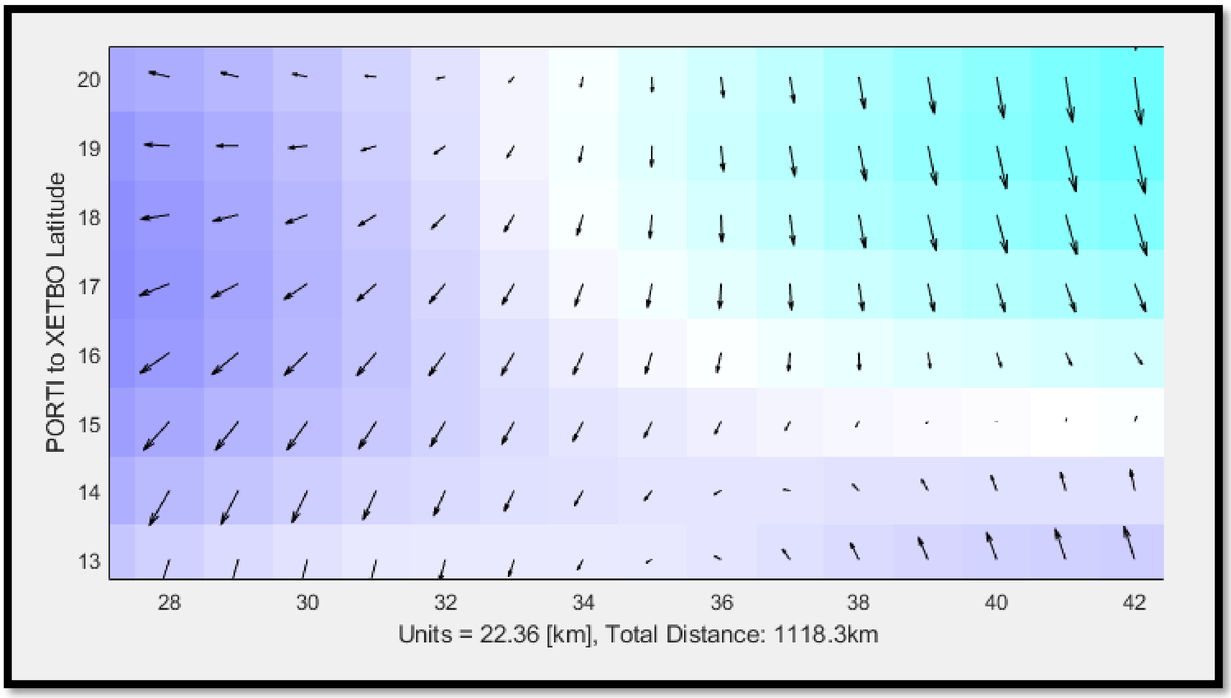
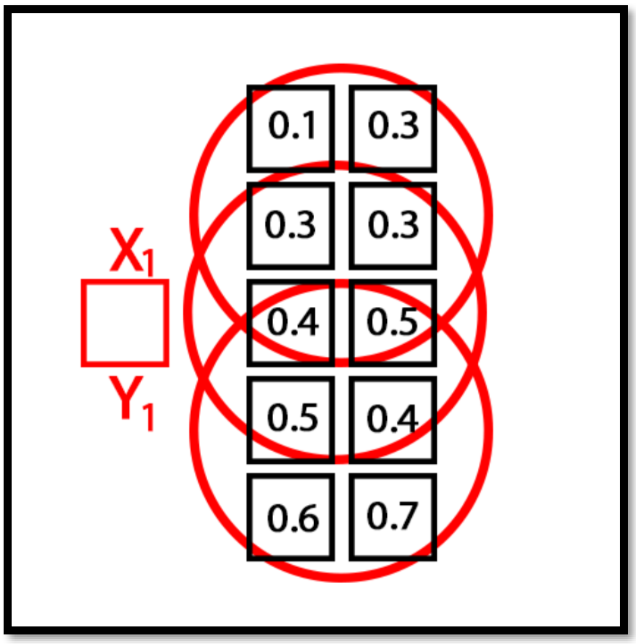


Figure 20: Grid conversion for each x and y value. (Gelbul)

If the plane's starting location was X_1, Y_1, then 9 points (towards the positive x) would be selected, see *Figure 21*. The top four, middle six, and bottom four values are grouped. The mean of these values is calculated.



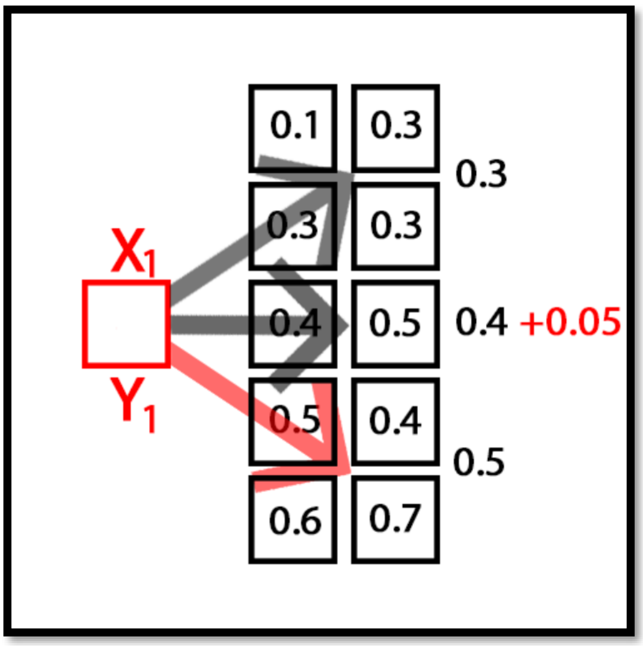


Figure 21 & 22: Calculating next grid set and decision making. (Gelbul)

Afterwards, different cases are examined:

- **Case 1:** There is a group with the highest mean (the other two are lower than this group and not equal). Then the flight path is updated to that point c and the other two blue points a and b are dismissed, see *Figure 23*.

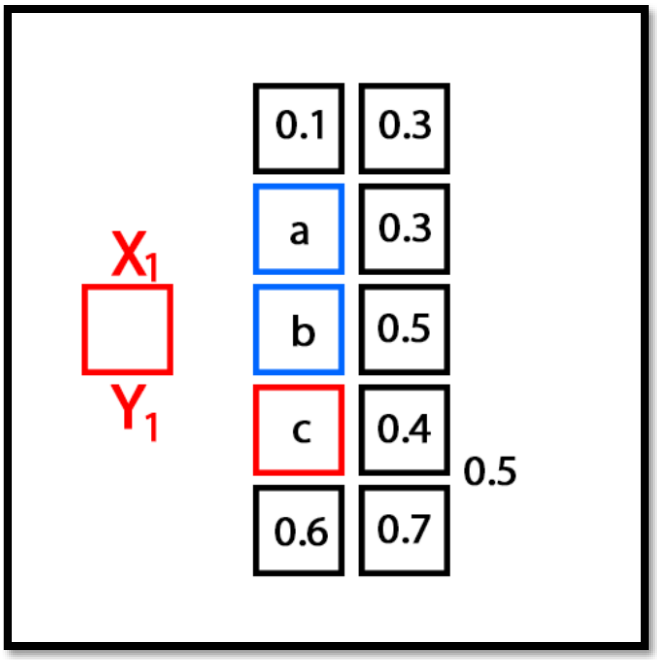


Figure 23: Flight path updated to point *c*. (Gelbul)

- **Case 2:** The top group and the bottom group have the same mean, the flight path will be updated to the middle group, b. The reason for this is that staying in the centre no matter how beneficial the tailwind is going to be is not worth moving. This is not a major issue to consider since having a mid-section that is lower than the top and bottom section alone is a very rare occasion.
- **Case 3:** This case introduces a precautionary action. *Figure 22* displays a +0.05 for the mid-section, meaning that if the any value of the other two group is equal to the mid-section, then the flight path should tend to stay on the middle group that is preferably closer to the great circle (later discussed). After conducting 50 tests for each of the values; 0.01, 0.025, 0.05, and 0.1: 0.5 gave the best possible result (see *Table 2*). The rest of the potential values to be added negatively affected the result.

	Flight Time Change (min)
0.01	DNF
0.025	-10.3
0.05	0.08
0.1	-8.8

Table 2: Mid-section additional value effect. (Gelbul)

For the program to complete the flight path a control device was placed in using the y-axis. The starting point would be the y_1 value and if this value stayed as y_1 the mid-section would be needing the value addition just like in *Figure 22*. If the new y value was higher than y_1, then the bottom section would require the addition, moreover, if the y value was lower than y_1, then the top section would require the addition. This was necessary to keep the flight path closer to the great circle and go over too much. A working example is provided in *Figure 24*.

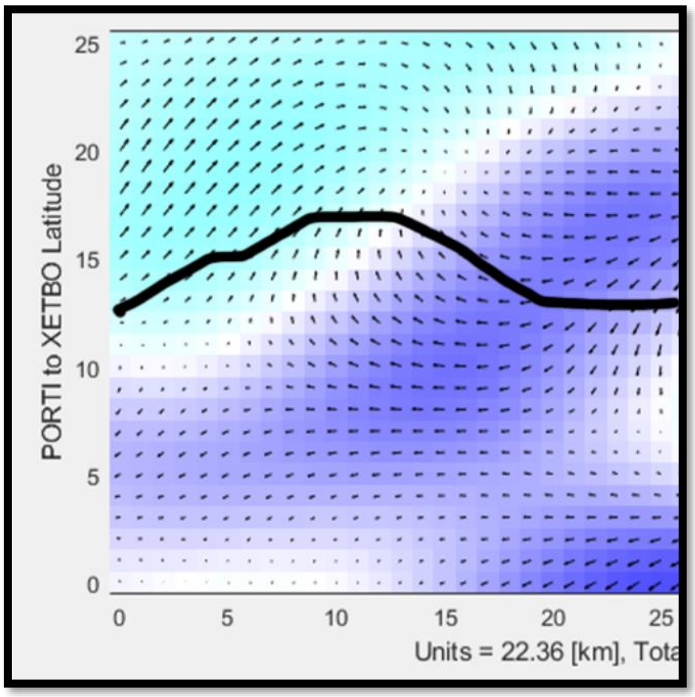


Figure 24: Working example of the algorithm, 0.2 minutes faster than normal. (Gelbul)

Unfortunately, after 50 trials of randomised wind maps, the program code gave -3.81 minutes traversing time on average, the best time being 0.3 minutes faster than normal.

Another reason why this algorithm was impractical is the fact that it does not measure and evaluate the graph but instead limits itself by analysing the next step only. A perfect example of this case can be seen in *Figure 25*, where the algorithm will mostly likely move to Area A and will stay on course while missing Area B.

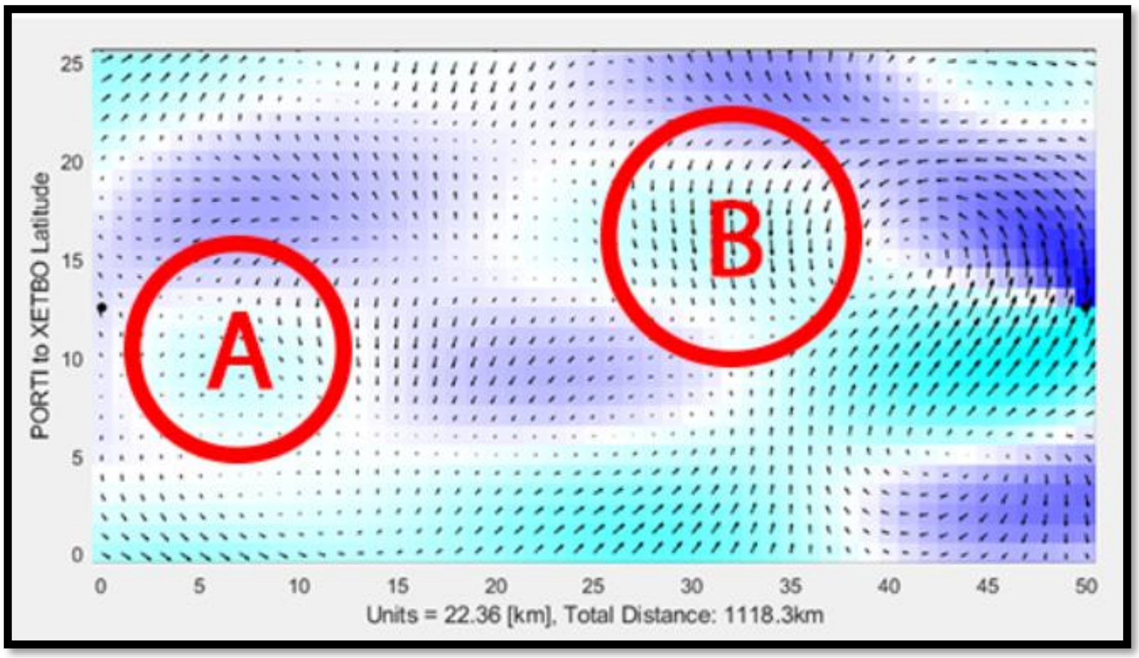


Figure 25: Algorithm cannot see the upcoming beneficial route. (Gelbul)

The following algorithm revealed in the next chapter focuses on analysing the graph all together and then decides on which route to take, the idea is to learn from the mistakes made in this algorithm.

CHAPTER 4.4: Headwind Avoidance Algorithm

The idea behind this algorithm is to see the bigger picture and relay what lies ahead by initially scanning the wind map fully and then outlining the areas to avoid.

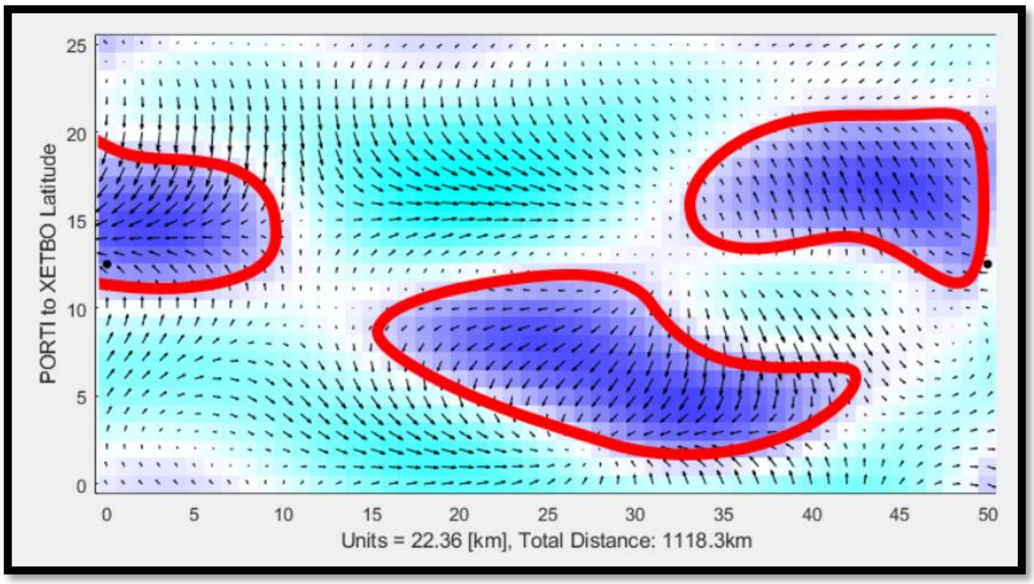


Figure 26: Mapping toolbox outlines. (Gelbul)

The same method used for the wind values in the previous algorithm was applied for this algorithm. Using KNN, any wind value less than 0.4 (closer to headwind) was grouped and outlined. Then Mapping Toolbox from MATLAB was used to define a buffer zone around the border of these areas, see *Figure 26*.

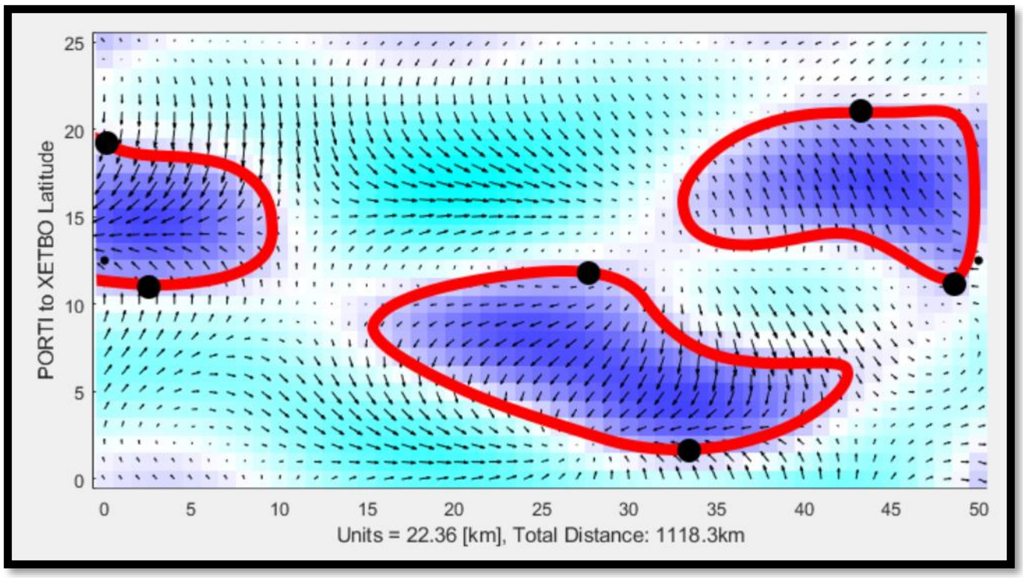


Figure 27: Locating high and low points. (Gelbul)

The next step was to locate the highest and lowest points of all the major headwind groups. For both points, this was measured using the grid system's y value (see *Figure 27*). Next step was to establish the flight path.

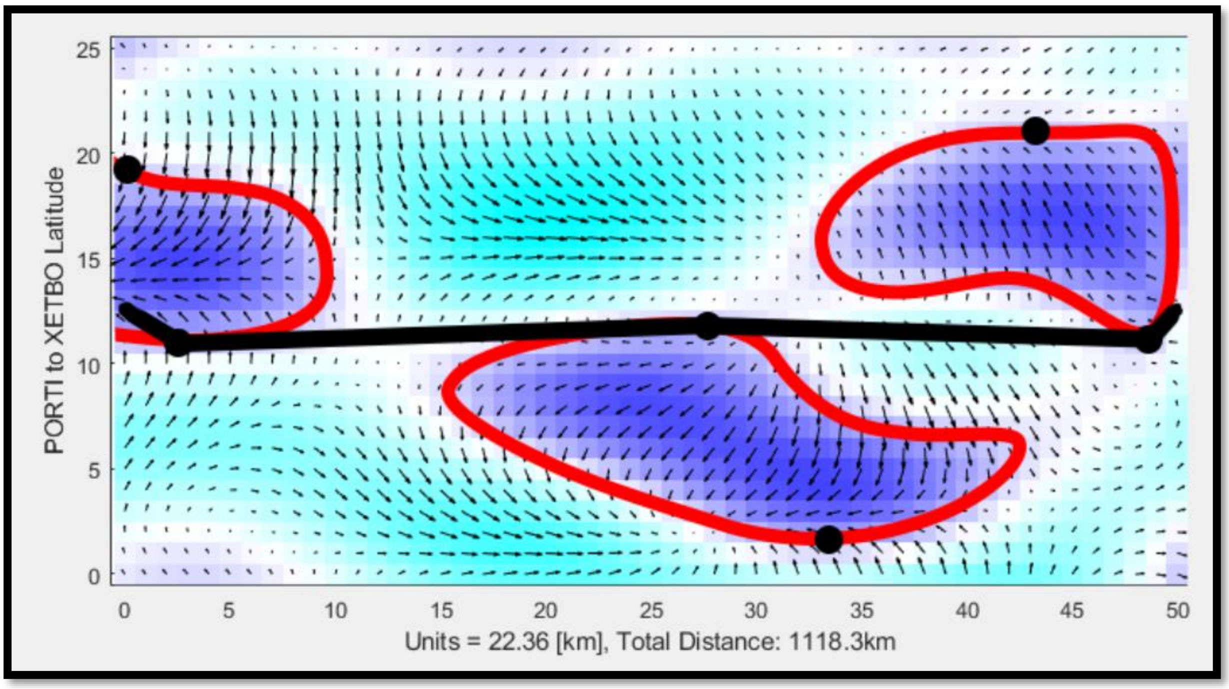


Figure 28: Forming the flight path. (Gelbul)

The program code gave -0.93-minute traversing time compared to a direct flight in 50 trials. The best flight time was 0.3-minute, similar result with the previous algorithm on this aspect. This was in fact an improvement compared to the previous algorithm (2.88 minutes more efficient algorithm). The program was avoiding the headwinds very well, which at the same time helped the flight path form around possible tailwinds. Although, this result concluded that the algorithm should focus on using the tailwinds more rather than avoiding them. However, the idea of analysing the whole wind map beforehand was an advantage to get a shorter flight time.

One major downside of this algorithm was the absence of creating a curve between the points. There was an attempt to implement this function, depending on the grid values in between points. This method worked out fine for lines between points which had a stable and one sided (either headwind or tailwind) property, however, areas where headwind turned into tailed or vice versa in between these points, the program code outputted error, unable to create a curvature. This issue could be avoided by creating additional waypoints and separating these changes, although this defeats the purpose of this algorithm in the first place. The next algorithm will be analysed and explain in-depth since its efficiency is incomparable to first two algorithms presented, in this chapter and *Chapter 4.3*.

CHAPTER 4.5: MATLAB's FMINCON Assisted Algorithm

The algorithm has one main script and three subscripts to call different functions to calculate various properties. The main script is named *Path_Optimsaition_Gelbul*, where the main code is being run. The other three subscripts are called: *Wind_Data*, *Straight_Line* and *Time_Calculator*. Their functionality will be discussed later. The interface and the wind map were already discussed earlier in *Chapter 4*.

The algorithm's basis starts with creating waypoints, inspired by the previous algorithm, however, these way points are initially placed on the great circle line (*Figure 29*).

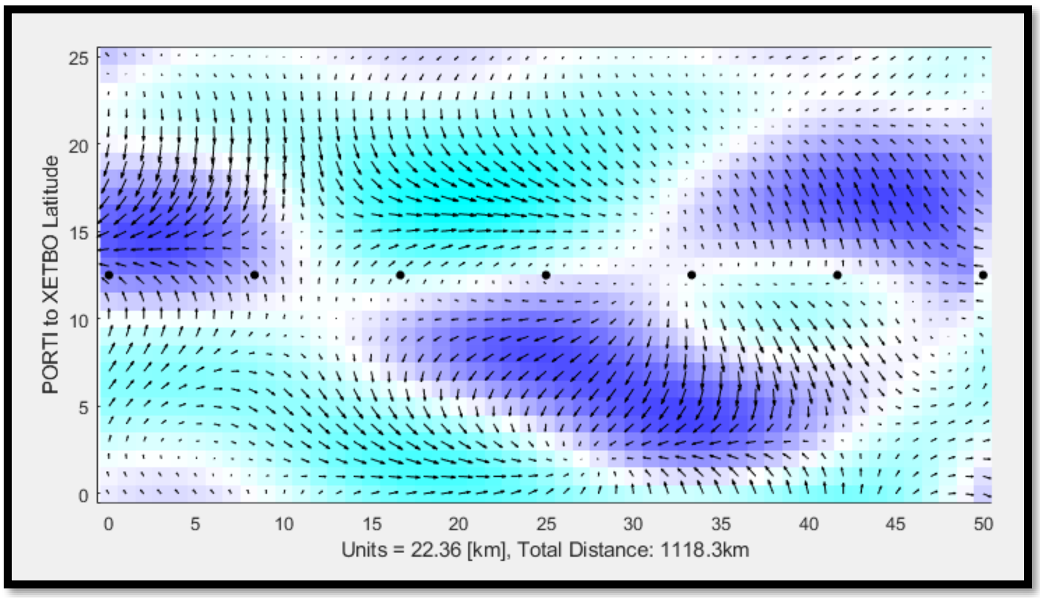
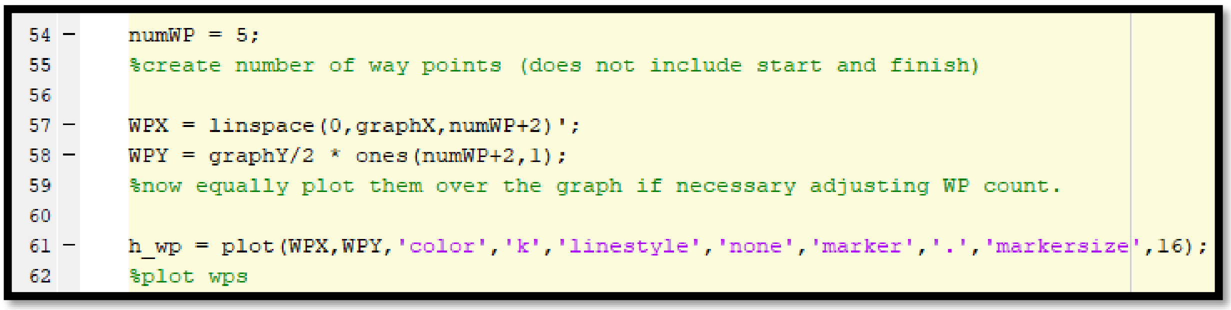


Figure 29: Plotting waypoints on the graph. (Gelbul)

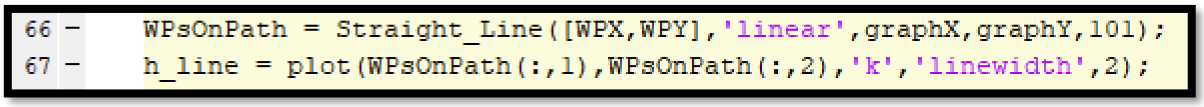
The first waypoint represents the PORTI waypoint, and the last waypoint is the XETBO waypoint, discussed in *Chapter 3*.



```
54 -    numWP = 5;
55      %create number of way points (does not include start and finish)
56
57 -    WPX = linspace(0,graphX,numWP+2)';
58 -    WPY = graphY/2 * ones(numWP+2,1);
59      %now equally plot them over the graph if necessary adjusting WP count.
60
61 -    h_wp = plot(WPX,WPY,'color','k','linestyle','none','marker','.','markersize',16);
62      %plot wps
```

Code Snippet 4: Setting waypoints. (Gelbul)

Code Snippet 4 outputs the plotted, 5 waypoints (plus start and finish) in *Figure 29*. Each of these waypoints are plotted with equal distance which can be seen in line 57. The y value for these waypoints will exactly be half of the y axis, right in the middle.



```
WPsOnPath = Straight_Line([WPX,WPY],'linear',graphX,graphY,101);
h_line = plot(WPsOnPath(:,1),WPsOnPath(:,2),'k','linewidth',2);
```

Code Snippet 5: Setting waypoints. (Gelbul)

Code Snippet 5 displays how the line is plotted, this is the great circle line, going through the waypoints (see *Figure 30*).

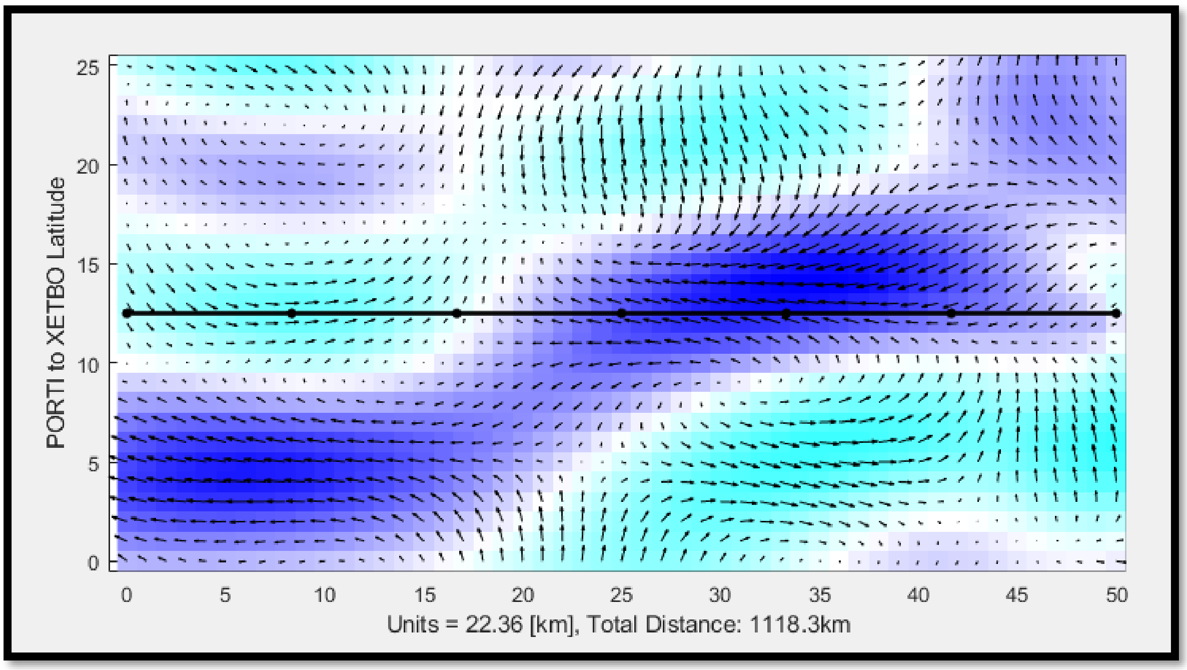


Figure 30: Plotting the straight line. (Gelbul)

Before getting into the optimisation process, it is useful to demonstrate how time variable is calculated. *Time_Calculator* subscript is used to complete these calculations.

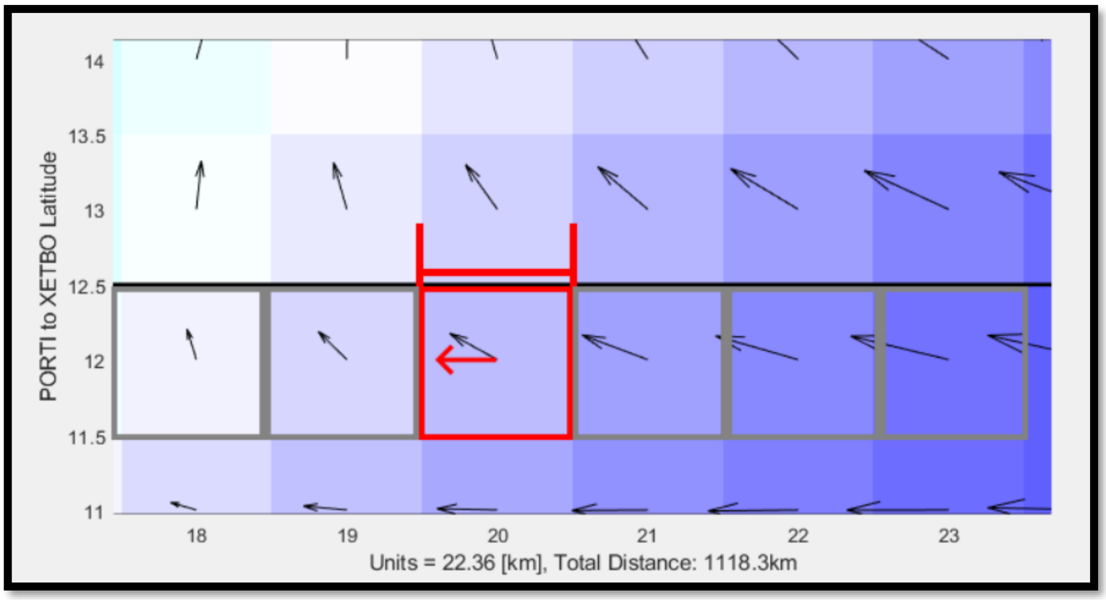
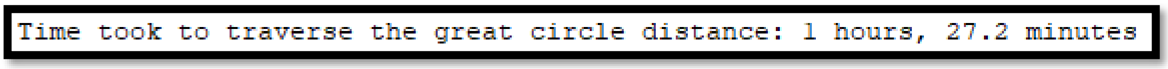


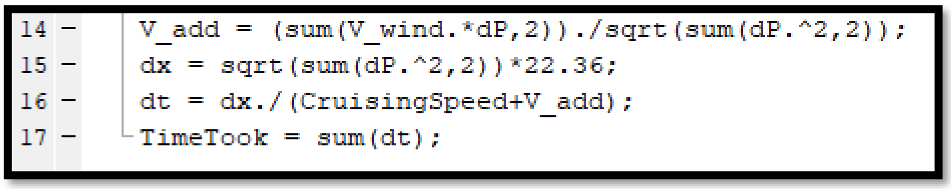
Figure 31: Zoomed in appearance of each grid and the wind position. (Gelbul)

This is part is inspired from the first algorithm in *Chapter 4.3*, where each wind data was positioned into a point, forming a grid. In *Figure 31*, the heading is towards the + x-axis, while the headwind is opposing the direction of travel (the red arrow). Moreover, for each grid, average velocity is calculated since in each grid the wind is constant. Afterwards, with the average speed calculated and knowing that each grid is 22.36km (the entire great circle for this experiment was 1118.3km), the time it takes to traverse one grid can be calculated using the formula $v = \frac{\Delta x}{\Delta t}$ (v is the average velocity, Δx is displacement and Δt is the time it takes to travel). Finally, the sum of each grid's time will give the total amount of time it takes to travel the line. See *Code Snippet 6* for the dot product and additions of each grid's time taken to complete using a simple calculus. When this calculation is completed, the program code outputs the result. See *Figure 32*, specifically calculated for the wind map seed: 757, resulted with 1 hour 27.2 minutes.



```
Time took to traverse the great circle distance: 1 hours, 27.2 minutes
```

Figure 32: Output for the time calculation. (Gelbul)



```
V_add = (sum(V_wind.*dP,2))./sqrt(sum(dP.^2,2));
dx = sqrt(sum(dP.^2,2))*22.36;
dt = dx./(CruisingSpeed+V_add);
TimeTook = sum(dt);
```

Code Snippet 6: *Time_Calculator* calculus. (Gelbul)

At this point, the waypoints are ready, calculation mechanisms have been discussed and variables are set. The following section uses the FMINCON from the optimisation toolbox in MATLAB to optimise the flight path.

FMINCON requires an “objective function” in line 80 (See *Appendix K* for the full code). *Time_Calculator* subscript’s function is valid for that use. FMINCON is applied to each, and every waypoint established previously, and it computes the best-fit position for these points by calculating the time it takes for each grid to be traversed, using *Time_Calculator*.

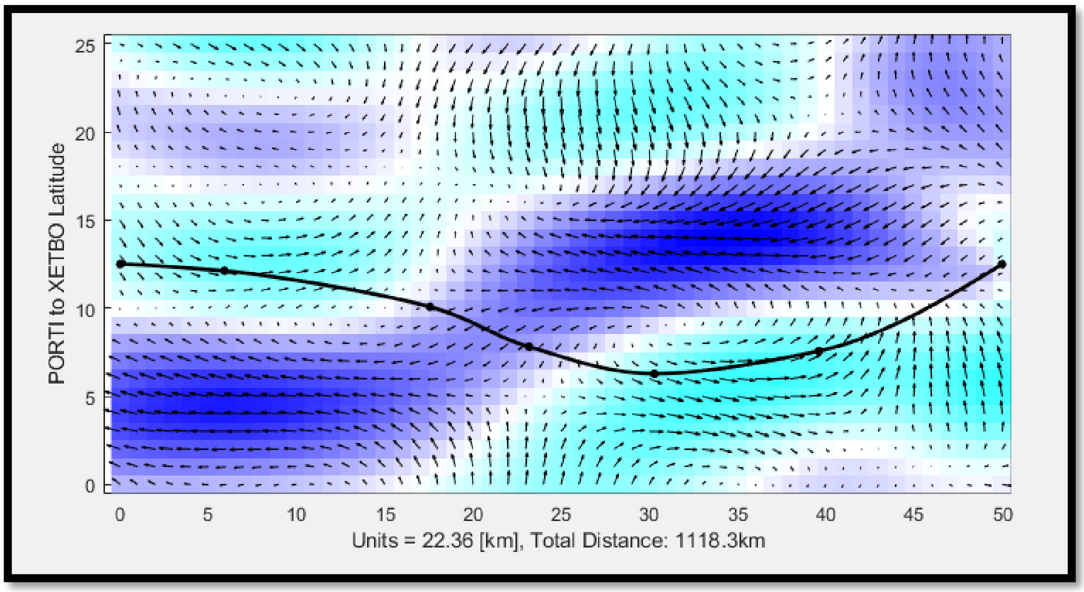
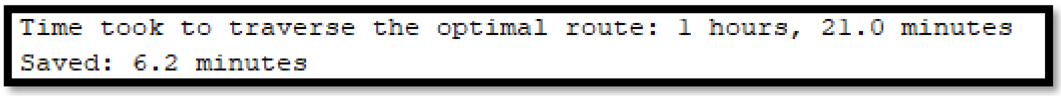


Figure 33: Output for the optimal path using FMINCON. (Gelbul)

After the Optimisation process is done, the altered flight path (see *Figure 33*) is fed into the Time_Calculator once again to print out the time took to traverse the flight path. Next, this flight time is subtracted from the previously given straight great circle's flight time.



```
Time took to traverse the optimal route: 1 hours, 21.0 minutes
Saved: 6.2 minutes
```

Code Snippet 7: Outputting results from FMINCON. (Gelbul)

This specific test with the *rng*(757) wind map seed saved 6.2 minutes from a 1 hour 27.2 minutes flight (*Code Snippet 7*). Additionally, 50 tests with distinct map seeds were ran and the average time savings were 4.1 minutes, which is a serious improvement compared to algorithms in *Chapter 4.4* and *Chapter 4.3*. The outcomes will be discussed in the following chapter.

CHAPTER 5: Discussion and Evaluation

The project started out as a broad research that was meant to explore many areas in aviation. The focus was transatlantic flights, however, I wanted to delve into every subcategory about the airline industry to construct a multi-layered, complex, and well-designed piece of work. My open approach to each variable slowly decayed due to the entangled network of business. For instance, one of my main goals was to tackle turbulence in aviation and how the airline companies attempt to overcome this issue daily. After researching this broad topic, I was still assured that I could mention it in my report. Although, the weather conditions over the Atlantic Ocean were a tremendous topic to cover on its own, therefore, it took most of my time and energy. With my focus completely shifting towards wind maps over the Atlantic, many other objectives that I have previously set out complete diminished. Another example is the weather conditions over the airports (JFK and LHR), which was impossible to cover in a single paper if every single weather condition had to be discussed. I was able to cover Atlantic Ocean wind maps, although, they had to be randomised and were not specifically prepared for any time of the year, which was another objective I wanted to complete. Fuel and aircraft statistics were another concept I wanted to investigate, instead of cycling through different aircraft types I picked the most common aircraft type which was Airbus A330-300. About fuel consumption, I will still showcase the profit margin that will be made from the algorithm I produced at the end of this chapter.

Those were the objectives I failed to complete. Nonetheless, I succeeded with the most prominent topics that I wanted to demonstrate. Such as, the great circle distance calculations. I believe I was able to provide distinct methodologies to calculate the great circle distance, while these methodologies where all suitable for the purpose of this project. I exercised in spherical trigonometry thoroughly and adequately. I examined each methodology that was available through geodesy books. I am satisfied with the amount of calculus I was able to deliver to this paper. Especially, the first exploration that I accomplished from scratch is a very sophisticated piece of work that is a strong starting point for the project and assists the following methodologies (Haversine and Vincenty). I could have included surplus number of methodologies, for instance the Law of Cosines. However, instead I presented Haversine’s formula by itself since I believe it is sufficient for the trigonometry available and helps the reader grasp the concept far better than Law of Cosines. Vincenty’s formulae was hard to explain and work with and I believe the amount of space I left for Vincenty’s was perfect, not a lot and not too little. Determining the best methodologies for this project was crucial, moreover, factors such as time cost for the computation and accuracy had to be decided carefully. Time cost for the computation to complete is necessary because according to FAA, on an average summer night, there are more than 1500 transatlantic flights happening from North America to Europe (FAA, n.d.).

That would equate to calculating 1.500 flight routes with the possible of optimisation including millions of combinations of each flight daily. In other words, the system designed to compute this problem must be fast, so the code written must be fast. Correspondingly, the accuracy should be acceptable, since FAA has strict regulations on how transatlantic flights must occur, with specific required distance, laterally and vertically as well. In this case, Haversine's formula was the best-fit methodology. After calculating this distance in program code next step was to start building the optimisation program.

Right after my research, I started to build a program from scratch using Eclipse an IDE, supported by IBM. The idea was to include all possible variables as previously mentioned, therefore, using an IDE was a good start for a big project. However, the problem with using Eclipse was the work, time and energy needed to develop certain packages from scratch, such as plotting a basic graph with a mesh, creating an environment where each variable is interrelated etc. The first algorithm provided in *Chapter 4.3* was created in Eclipse initially. After facing problems while constructing the second algorithm in *Chapter 4.4,* I decided to separate certain events such as, calculating the great circle distance, calculating time, plotting a graph etc. in different environments. That is the reason why great circle is calculated in Python code, and rest of the wind map application is situated in MATLAB. Unfortunately, switching coding environments added extra work on top of the project. Although, it was the right decision to reduce the variables and focus on one main variable which is the wind maps. This brought more clarity to the aim of the project, and having a platform, MATLAB, where the major specialty is maths, exponentially reduced the program code complexity.

About the time management in general, the start of the implementation of the main features were simple, however, as time went on and the relation between each variable became more challenging to investigate, the project started slowing down. When I faced these challenges, I decided on the path I am going to follow very late. Instead, I waited out to overcome the issues I faced, even though the issues were troublesome to overcome. If I saw that I would not be able to complete these issues and decided earlier, I could have spent more time on going further into wind maps and jet streams in general. Although, for the level of this project, which I believe it is high, the degree of difficulty for wind maps is considerably satisfying. Apart from this, whenever I had a task, a task that was feasible, at hand that needed to be completed by a specific date, I would work on it and complete it. While working on this project I experienced hardships due to Covid-19 which caused a major setback for the project's future.

Ultimately, I would like to showcase the final algorithm's, *Chapter 4.5*, results. Following the algorithm, the flight from POTRI to XETBO, a 1118.3km distance flight that takes 1 hour and 27.2 minutes while cruising at a speed of 820 km/h, resulted 6.2 minutes faster, completing at 1 hour 21.0 minutes. That time saving is equivalent to 7.11% of the actual flight. According to Airbus, A330-300 burns around 5700kg/h. According to FAA, one gallon of jet fuel cost was on average $ 1.43. 6.2 minutes is 10.3% of an hour. 1-gallon equals to 3.78kg, therefore, it costs around $5.40 per kg of jet fuel.

$$\frac{10.3}{100} \times 5700 \times 5.40 = 3{,}170.34$$

This specific flight was $3,170.34 cheaper for the airline company. From another standpoint, on an average summer night the airline industry profited $4,755,510 for flight occurring only from North America to Europe.

References:

1. Airbus. n.d. *A330-300*. [online] Available at: <https://www.airbus.com/aircraft/passenger-aircraft/a330-family/a330-300.html> [Accessed 30 January 2021].
2. Arfken, G. "Spherical Polar Coordinates." §2.5 in *Mathematical Methods for Physicists, 3rd ed.* Orlando, FL: Academic Press, pp. 102-111, 1985.
3. Bullock, R., 2007. Great Circle Distances and Bearings Between Two Locations. [online] p.2. Available at: <https://dtcenter.org/sites/default/files/community-code/met/docs/write-ups/gc_simple.pdf> [Accessed 24 February 2021].
4. Cappucci, M., 2019. Flight reaches 801 mph as a furious jet stream packs record-breaking speeds. *The Washington Post,*.
5. C.F.F. Karney, 2013. Algorithms for geodesics, J. Geodesy 87: 43-55. https://dx.doi.org/10. 1007/s00190-012-0578-z. Addendum: http://geographiclib.sf.net/geod-addenda.html.
6. Chen, C., Hsu, T. and Chang, J., 2004. A Novel Approach to Great Circle Sailings: The Great Circle Equation. *The Journal of Navigation*, [online] 57(2), pp.311-325. Available at: <http://ntour.ntou.edu.tw:8080/ir/bitstream/987654321/41783/2/div-class-title-a-novel-approach-to-great-circle-sailings-the-great-circle-equation-div.pdf> [Accessed 24 February 2021].
7. Esenbuğa, Ö., Akoğuz, A., Çolak, E., Varol, B. and Erol, B., 2016. Comparison of Principal Geodetic Distance Calculation Methods for Automated Province Assignment in Turkey. *16th International Multidisciplinary Scientific GeoConference SGEM,* Section 9 Geodesy and Mine Surveying.
8. Federal Aviation Administration. 2021. *The North Atlantic NAT Resource Guide*. [online] Available at:

<https://www.faa.gov/about/office_org/headquarters_offices/avs/offices/afx/afs/afs400/afs410/media/nat.pdf> [Accessed 15 January 2021].

9. Hall, R., Erdélyi, R., Hanna, E., Jones, J. and Scaife, A., 2014. Drivers of North Atlantic Polar Front jet stream variability. *Journal of Climatology*, 35(8), pp.1697-1720.
10. Hwang, Alan, 2009. Mapping and Geospatial Data Analysis Using MATLAB, MathWorks.
11. Kern, W. and Bland, J., 1938. *Solid Mensuration With Proofs*. 2nd ed. New York: John Wiley & Sons.
12. Kifana, B. and Abdurohman, M., 2021. Great Circle Distance Methods for Improving Operational Control System Based on GPS Tracking System. *International Journal*, 4(4), pp.647-662.
13. LatLong. 2012. *Airports*. [online] Available at: <https://www.latlong.net/> [Accessed 13 March 2021].
14. McNish, L., 2005. *Latitude and Longitude*. [online] RASC Calgary Centre. Available at: <https://calgary.rasc.ca/latlong.htm> [Accessed 3 February 2021].
15. Olga Rodionova, Mohammed Sbihi, Daniel Delahaye, Marcel Mongeau. North Atlantic Aircraft Trajectory Optimization. IEEE Transactions on Intelligent Transportation Systems, Institute of Electrical and Electronics Engineers (IEEE), 2014, 15 (5), pp 2202-2212. 10.1109/TITS.2014.2312315 hal-00981337
16. Panou, G., Delikaraoglou, D. and Korakitis, R., 2013. Solving the geodesics on the ellipsoid as a boundary value problem. *Journal of Geodetic Science*, 3, pp.40-47.
17. Sajeevan, G., 2008. Commentary. *Latitude and Longitude - A misunderstanding*, 94(5), pp.568-569.

18. Sharp, T., 2017. *How Big is Earth?*. [online] Space.com. Available at: <https://www.space.com/17638-how-big-is-earth.html> [Accessed 28 January 2021].
19. Statista.com. 2021. *Number of Atlantic air traffic passengers traveling to or from the United States from 2006 to 2020*. [online] Available at: <https://www.statista.com/statistics/193551/atlantic-air-traffic-passengers-travelling-to-or-from-the-us/> [Accessed 23 March 2021].
20. Stevens, Joshua and Smith, Jennifer M. and Bianchetti, Raechel A. *Mapping Our Changing World*, Editors: MacEachren, Alan and Peuquet, Donna J. University Park, PA: Department of Geography, The Pennsylvania State University, 2012.
21. Stonelake, B., 2013. *Gnomonic Projections onto Various Polyhedra*. [online] Oregon: Southern Oregon University, p.4. Available at: <http://webpages.sou.edu/~stonelakb/math/pdf/GP%20v3.pdf> [Accessed 20 February 2021].
22. Stover, Christopher and Weisstein, Eric W. "Cartesian Coordinates." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/CartesianCoordinates.html (Accessed: 10 January 2021)
23. Vincenty, T., 1975. Survey Review. *Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations*, 23(176), pp.88-93.
24. Weisstein, Eric W. "Antipodal Points." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/AntipodalPoints.html.
25. Woollings, T. and Blackburn, M., 2012. The North Atlantic Jet Stream under Climate Change and Its Relation to the NAO and EA Patterns. *Journal of Climate*, 25(3), pp.886-902.

Bibliography:

1. Robinson, A. H., Morrison, J. L., Muehrcke, P. C., Jon Kimerling, A. and Guptill, S. C., Elements of Cartography,. John Wiley, 2002, 6th edn.
2. Deakin R. E. and Hunter M. N., 2010, Geometric Geodesy - Part B, Lecture Notes, School of Mathematical & Geospatial Sciences, RMIT University, Melbourne, Australia
3. Sjöberg L. E., 2007, Precise determination of the Clairaut constant in ellipsoidal geodesy, Surv. Rev., 39, 81-86.
4. Sjöberg L. E. and Shirazian M., 2012, Solving the direct and inverse geodetic problems on the ellipsoid by numerical integration, J. Surv. Eng., 138, 9-16.
5. Thomas C. M. and Featherstone W. E., 2005, Validation of Vincenty's formulas for the geodesic using a new fourth-order extension of Kivioja's formula, J. Surv. Eng., 131, 20-26.
6. Kivioja L. A., 1971, Computation of geodetic direct and indirect problems by computers accumulating increments from geodetic line elements, Bull. Geod., 99, 55-63.
7. Rapp R. H., 1993, Geometric Geodesy - Part II, Department of Geodetic Science and Surveying, Ohio State University, Columbus, Ohio, USA.
8. Lambert, W.D. and Swick, C.H.,1935, Formulas and Tables for the Computation of Geodetic Positions on the International Ellipsoid: U.S. Coast and Geodetic Survey Special Publication No. 200.
9. U.S.C.& G.S., Formulas and Tables for the Computation of Geodetic Positions, U.S. Coast and Geodetic Survey Special Publication No. 8.

10. NationsOnline. 2016. *List of major Canadian and US Airports with Airport-links and IATA 3-Letter Codes Major international airports in the United States and Canada*. [online] Available at: <http://nationsonline.org> [Accessed 20 January 2021].
11. Baldwin MP, Gray LJ, Dunkerton TJ, Hamilton K, Haynes PH, Randel WJ, Holton JR, Alexander MJ, Hirota I, Horinouchi T, Jones DBA, Kinnersley JS, Marquardt C, Sato K, Takahashi M. 2001. The Quasi-Biennial Oscillation. *Rev. Geophys.* **39**: 179– 229.
12. Berckmans J, Woollings T, Demory M-E, Vidale P-L, Roberts M. 2013. Atmospheric blocking in a high resolution climate model: influences of mean state, orography and eddy forcing. *Atmos. Sci. Lett.* **4**: 34– 40, doi: *10.1002/asl2.412.*
13. Butler AH, Thompson DWJ, Heikes R. 2010. The steady-state atmospheric circulation response to climate change-like thermal forcings in a simple general circulation model. J. Clim. **23**: 3474– 3496

Image Credits:

Figure 6: Strebe (2011) *Gnomonic Projection SW* [Online Image] Available from: https://commons.wikimedia.org/wiki/File:Gnomonic_projection_SW.jpg [Accessed 15/02/21]

Figure 7: Marozols (2009) *Gnomonic* [Online Image] Available from: https://commons.wikimedia.org/wiki/File:Gnomonic.png [Accessed 15/02/21]

Figure 10: Esenbuğa, Ö., Akoğuz, A., Çolak, E., Varol, B. and Erol, B., 2016. Comparison of Principal Geodetic Distance Calculation Methods for Automated Province Assignment in Turkey. *16th International Multidisciplinary Scientific GeoConference SGEM*, Section 9 Geodesy and Mine Surveying.

Figure 11: Rodionova, Olga & Sbihi, Mohammed & Delahaye, Daniel & Mongeau, Marcel. (2014). North Atlantic Aircraft Trajectory Optimization. Intelligent Transportation Systems, IEEE Transactions on. 15. 2202-2212. 10.1109/TITS.2014.2312315.

Figure 15: North Atlantic Infrared Satellite & Lightning, VT 2220Z, Copyright Jeppesen

Figure 19: Hwang, Alan, 2009. Mapping and Geospatial Data Analysis Using MATLAB, MathWorks.

All other figures, graphs, images, and charts belong to Erol Gelbul. If you would like to use any of them, follow this reference:

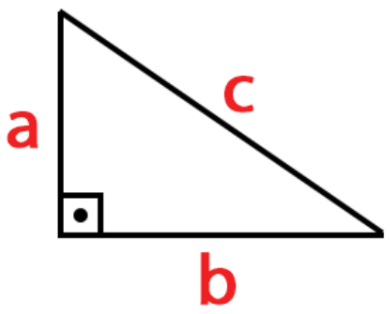
Gelbul, Erol. *Optimising Transatlantic Flight Paths* University of Hertfordshire, United Kingdom, 2021.

Additional Information and Facts:

- *All the mathematical calculations are made with TI-84 Plus CE-T from Texas Instruments.*
- *License for MATLAB was provided by the University of Hertfordshire.*
- *Word Count: 10603*

Appendix A: PYTHAGOREAN THEOREM

A triangle with a right angle (90°): $a^2 + b^2 = c^2$



Appendix B: EULER LEGURANGE DIFFERENTIAL EQUATION

Say A is an integral:

$$A = \int f(t, y, \dot{y})\, d\, t$$

Where:

$$\dot{y} \equiv \frac{d\, y}{d\, t}$$

Then:

$$\frac{\partial f}{\partial y} - \frac{d}{dt}\left(\frac{\partial f}{\partial \dot{y}}\right) = 0$$

Moreover, A is a stationary value only if Euler-Lagrange is satisfied.

Appendix C: SIMPLIFYING ALGEBRA

(…from where this was left off.)

$$LHS = \frac{\theta' sin^2\Phi}{\sqrt{(sin^2\Phi)\theta'^2 + 1}} \quad \rightarrow \quad \frac{\theta' sin^2\Phi}{\sqrt{\theta'^2(sin^2\Phi) + 1}} = G$$

$$\rightarrow \frac{\theta'^2 sin^4\Phi}{\theta'^2(sin^2\Phi) + 1} = G^2 \quad \rightarrow \quad \theta'^2 sin^4\Phi = G^2[\theta'^2(sin^2\Phi) + 1]$$

$$\rightarrow \ \theta'^2 = \frac{G^2}{sin^4\Phi - \ G^2 sin^2\Phi} \quad \rightarrow \quad \theta' = \frac{d\theta}{d\Phi}$$

Appendix D: TRIGONOMETRIC IDENTITY 1

$$\sqrt{A^2 + B^2}\cos(\theta - \theta_1) = -C\ cot\Phi \quad \text{where:} \quad \theta_1 = arctan\left(\frac{B}{A}\right)$$

Note: A and B are NOT the points A and B on the sphere. They are the constants from the equation.

Appendix E: TRIGONOMETRIC IDENTITIES

$$sin^2\varphi + \ cos^2\varphi = 1$$

$$1 + tan^2\varphi = sec^2\varphi$$

Appendix F: TRIGONOMETRIC IDENTITY 2

$$cos^2\alpha = 1 - sin^2\alpha$$

Appendix G: EARTH’S RADIUS IN MILES

$$r = 3963 - 13 \times \ \sin\ (Latitude)$$

Appendix H: IATA-CODES

LHR – London Heathrow

JFK – John F. Kennedy International

BOS – Logan International Airport

BWI – Baltimore/Washington International

MIA – Miami International Airport

LGA – LaGuardia International

EWR – Newark Liberty International

DCA – Ronald Reagan Washington National

Referencing:

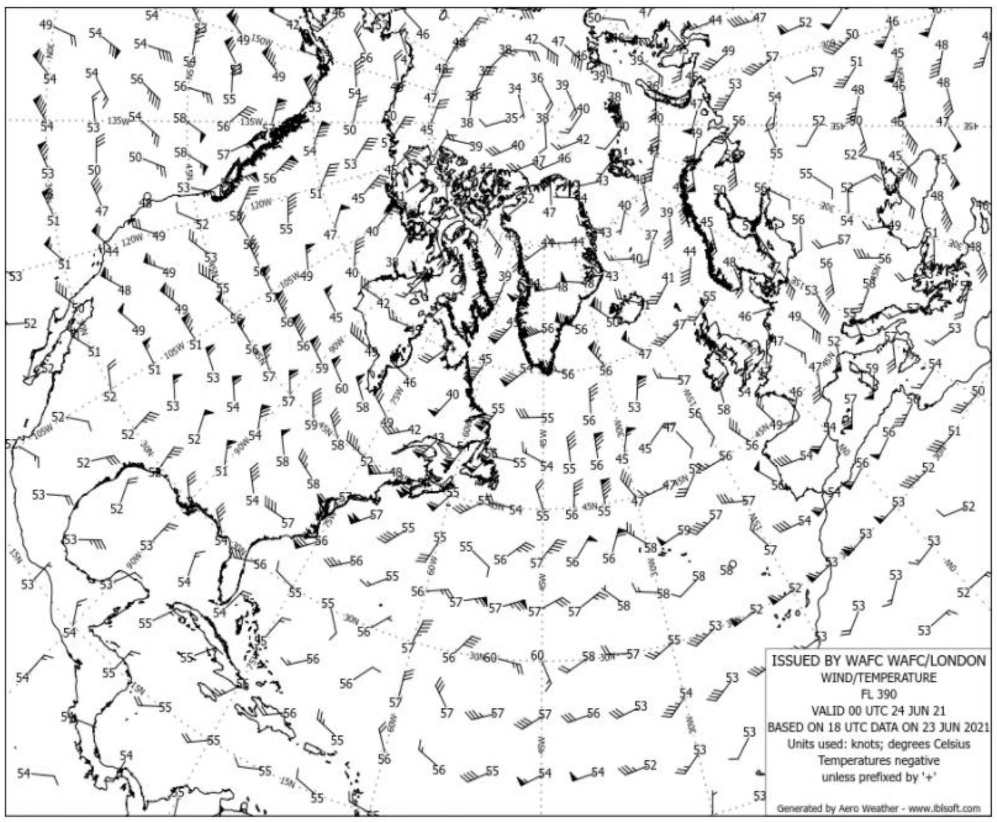
NationsOnline. 2016. *List of major Canadian and US Airports with Airport-links and IATA 3-Letter Codes Major international airports in the United States and Canada*. [online] Available at: <http://nationsonline.org> [Accessed 20 January 2021].

Appendix I: HAVERSINE DISTANCE PYTHON CODE

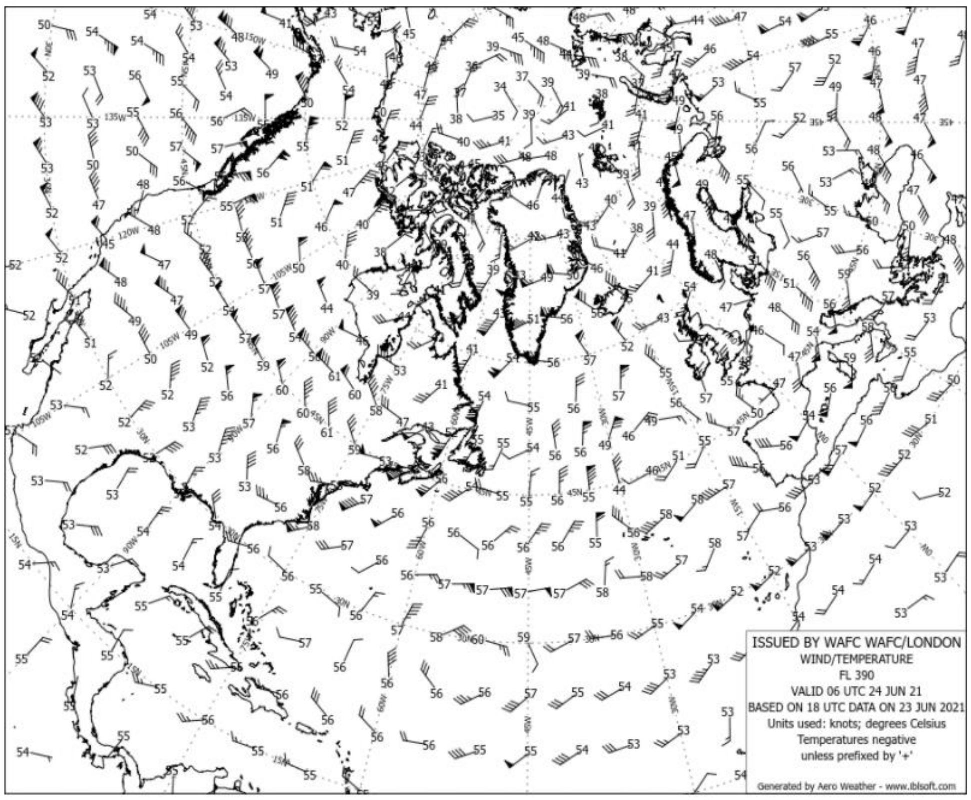
```
import math
#import libraries

class Haversine_Gelbul:

    def __init__(self,location1,location2):

        long1,lat1 = location1
        long2,lat2 = location2
        #coordinates of both of the locations are inserted either
        #as a tuple or as a list

        R = 6362000
        #the radius of Earth calculated using Chapter 2.1.6 where
        #the latitude of both airports were averaged to achieve a
        #more practical radius

        radlat1 = math.radians(lat1)
        radlat2 = math.radians(lat2)
        #converting the values to radians, again, this is not a
        #necessity, although we will be using radians all the time
        #for consistency

        delta_radlat=math.radians(lat2-lat1)
        delta_radlong=math.radians(long2-long1)
        #calculating the difference of both the values, starting from
        #the second value for each location and substracting the first


        h=math.sin(delta_radlat/2.0)**2+\
           math.cos(radlat1)*math.cos(radlat2)*\
           math.sin(delta_radlong/2.0)**2
        #calculating haversines central angle value: h

        if h>1 or h<0:
            print("h value is going to result in a floating point error")
        #this is a precaution if the points are antipodal or near antipodal
        #in other words might result in a floating point error

        c=2*math.atan2(math.sqrt(h),math.sqrt(1-h))
        #calculate haversine distance

        self.meters=R*c
        self.km=self.meters/1000.0
        #output result in kilometers

if __name__ == "__Haversine_Gelbul__":
    main()
```

Appendix J: WIND DATA OVER THE ATLANTIC AT FL390 AND UNIVERSAL COORDINATED TIMES

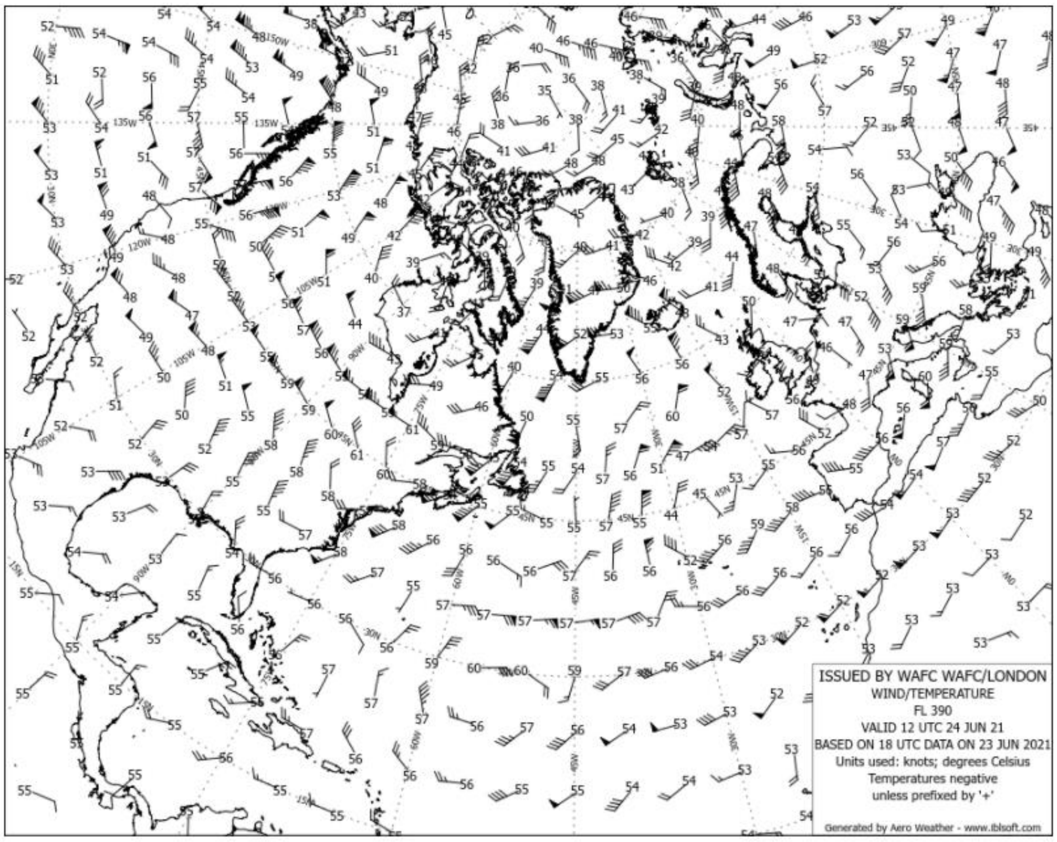
FL390 at 00Z by WAFC London


ISSUED BY WAFC WAFC/LONDON
WIND/TEMPERATURE
FL 390
VALID 00 UTC 24 JUN 21
BASED ON 18 UTC DATA ON 23 JUN 2021
Units used: knots; degrees Celsius
Temperatures negative
unless prefixed by '+'
Generated by Aero Weather - www.iblsoft.com

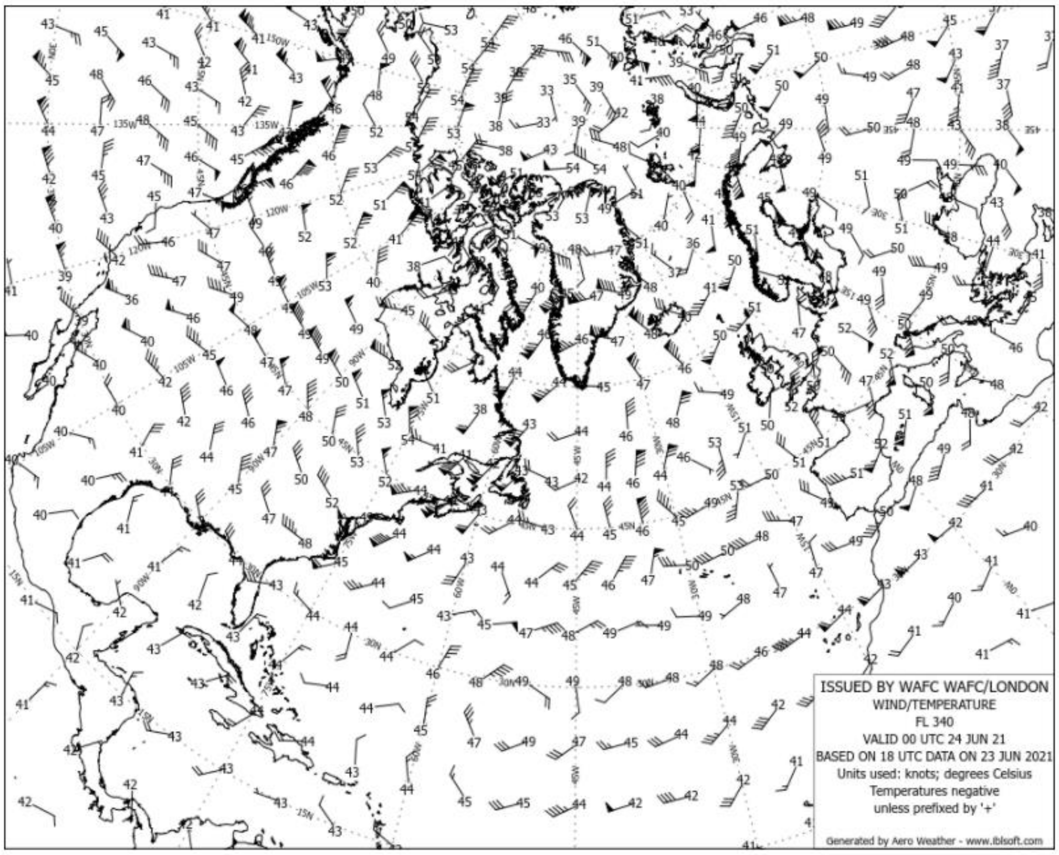
FL390 at 06Z by WAFC London


ISSUED BY WAFC WAFC/LONDON
WIND/TEMPERATURE
FL 390
VALID 06 UTC 24 JUN 21
BASED ON 18 UTC DATA ON 23 JUN 2021
Units used: knots; degrees Celsius
Temperatures negative
unless prefixed by '+'
Generated by Aero Weather - www.iblsoft.com

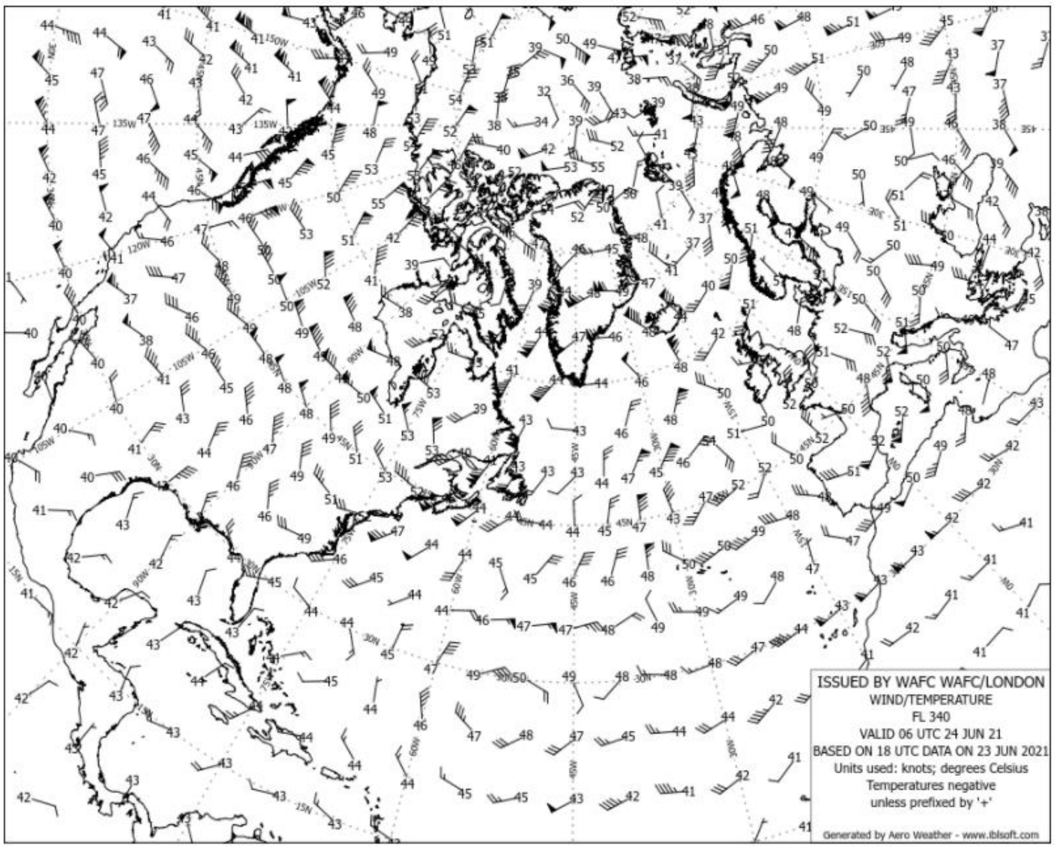
FL390 at 12Z by WAFC London


ISSUED BY WAFC WAFC/LONDON
WIND/TEMPERATURE
FL 390
VALID 12 UTC 24 JUN 21
BASED ON 18 UTC DATA ON 23 JUN 2021
Units used: knots; degrees Celsius
Temperatures negative
unless prefixed by '+'
Generated by Aero Weather - www.iblsoft.com

FL340 at 00Z by WAFC London


ISSUED BY WAFC WAFC/LONDON
WIND/TEMPERATURE
FL 340
VALID 00 UTC 24 JUN 21
BASED ON 18 UTC DATA ON 23 JUN 2021
Units used: knots; degrees Celsius
Temperatures negative
unless prefixed by '+'
Generated by Aero Weather - www.iblsoft.com

FL340 at 06Z by WAFC London


ISSUED BY WAFC WAFC/LONDON
WIND/TEMPERATURE
FL 340
VALID 06 UTC 24 JUN 21
BASED ON 18 UTC DATA ON 23 JUN 2021
Units used: knots; degrees Celsius
Temperatures negative
unless prefixed by '+'
Generated by Aero Weather - www.iblsoft.com

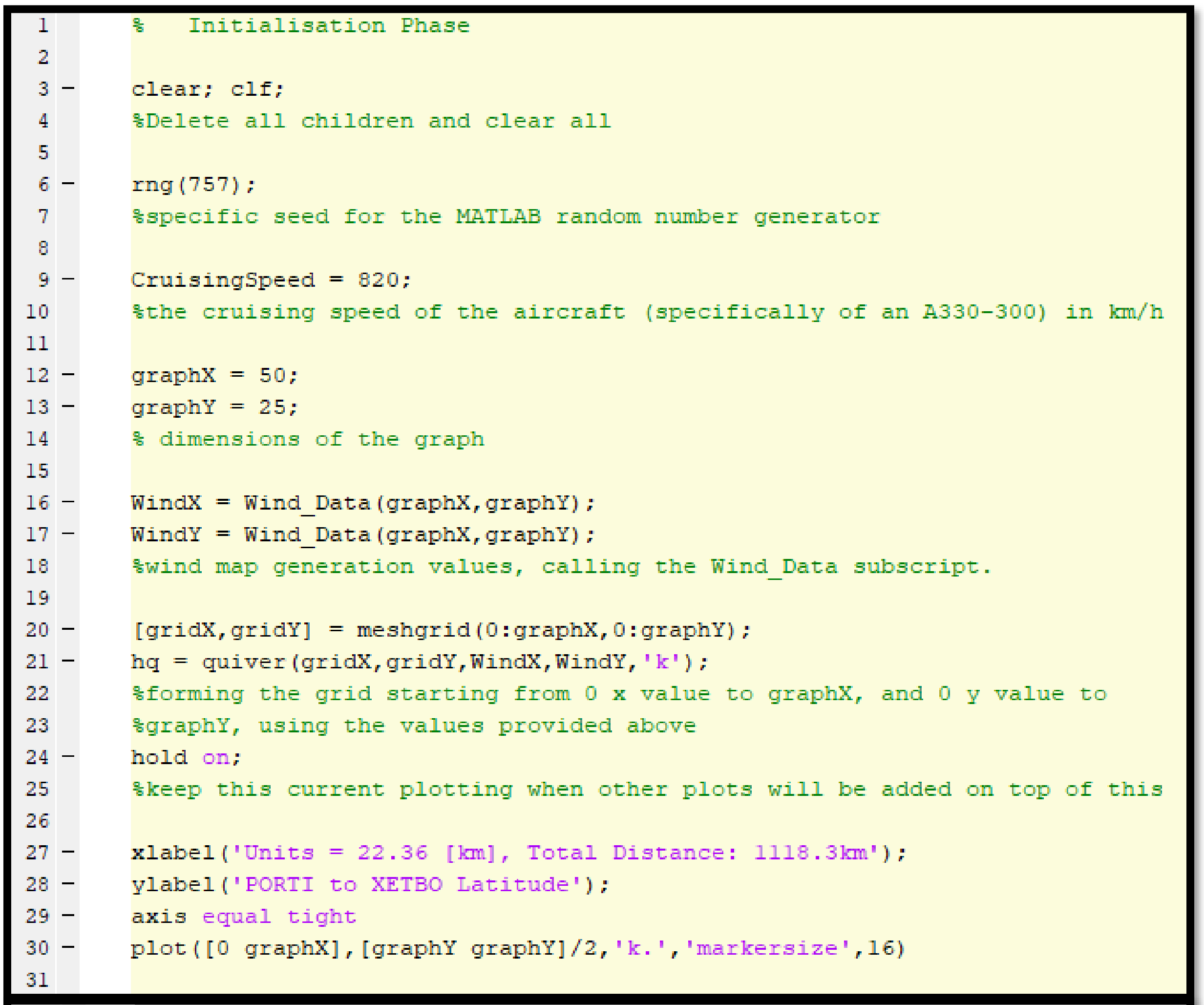
Referencing: *All the images above in Appendix J, belongs to:*

WAFC London, issued by: WAFC/London, Valid Until: 24 Jun 21, Based on 18 UTC Data,

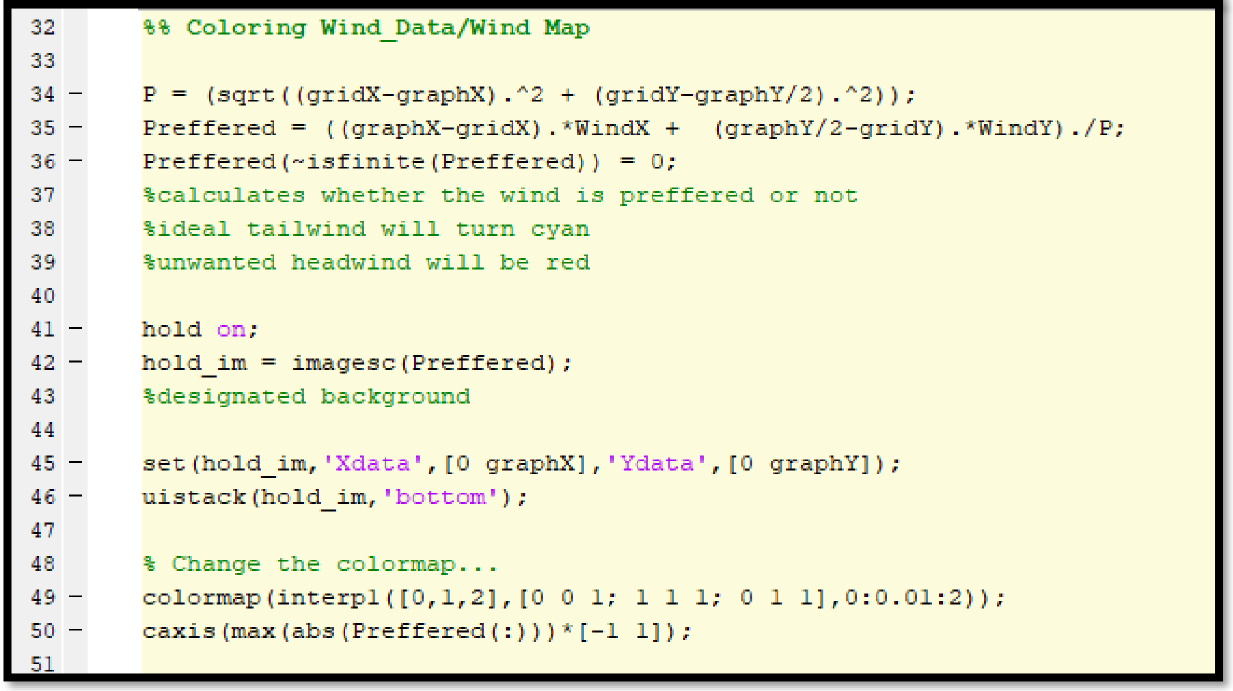
Generated by: Aero Weather

Appendix K: MATLAB CODE WITH OTHER SUBSCRIPTS

Path_Optimisation_Gelbul.m

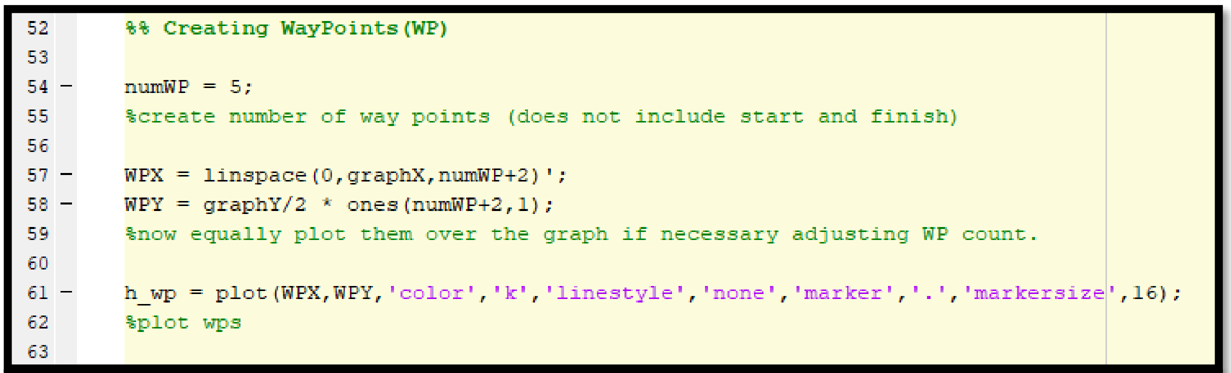


```
%   Initialisation Phase

clear; clf;
%Delete all children and clear all

rng(757);
%specific seed for the MATLAB random number generator

CruisingSpeed = 820;
%the cruising speed of the aircraft (specifically of an A330-300) in km/h

graphX = 50;
graphY = 25;
% dimensions of the graph

WindX = Wind_Data(graphX,graphY);
WindY = Wind_Data(graphX,graphY);
%wind map generation values, calling the Wind_Data subscript.

[gridX,gridY] = meshgrid(0:graphX,0:graphY);
hq = quiver(gridX,gridY,WindX,WindY,'k');
%forming the grid starting from 0 x value to graphX, and 0 y value to
%graphY, using the values provided above
hold on;
%keep this current plotting when other plots will be added on top of this

xlabel('Units = 22.36 [km], Total Distance: 1118.3km');
ylabel('PORTI to XETBO Latitude');
axis equal tight
plot([0 graphX],[graphY graphY]/2,'k.','markersize',16)

```

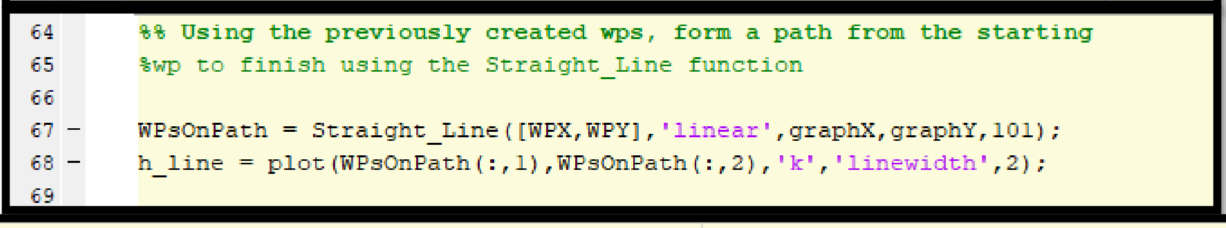


```
%% Coloring Wind_Data/Wind Map

P = (sqrt((gridX-graphX).^2 + (gridY-graphY/2).^2));
Preffered = ((graphX-gridX).*WindX +  (graphY/2-gridY).*WindY)./P;
Preffered(~isfinite(Preffered)) = 0;
%calculates whether the wind is preffered or not
%ideal tailwind will turn cyan
%unwanted headwind will be red

hold on;
hold_im = imagesc(Preffered);
%designated background

set(hold_im,'Xdata',[0 graphX],'Ydata',[0 graphY]);
uistack(hold_im,'bottom');

% Change the colormap...
colormap(interp1([0,1,2],[0 0 1; 1 1 1; 0 1 1],0:0.01:2));
caxis(max(abs(Preffered(:)))*[-1 1]);

```

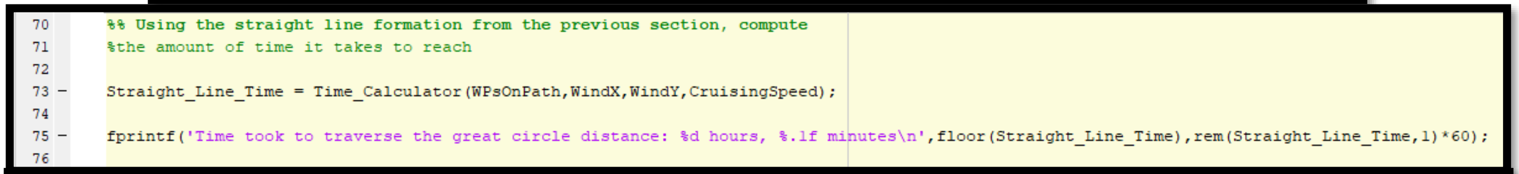
Path_Optimisation_Gelbul.m continued…



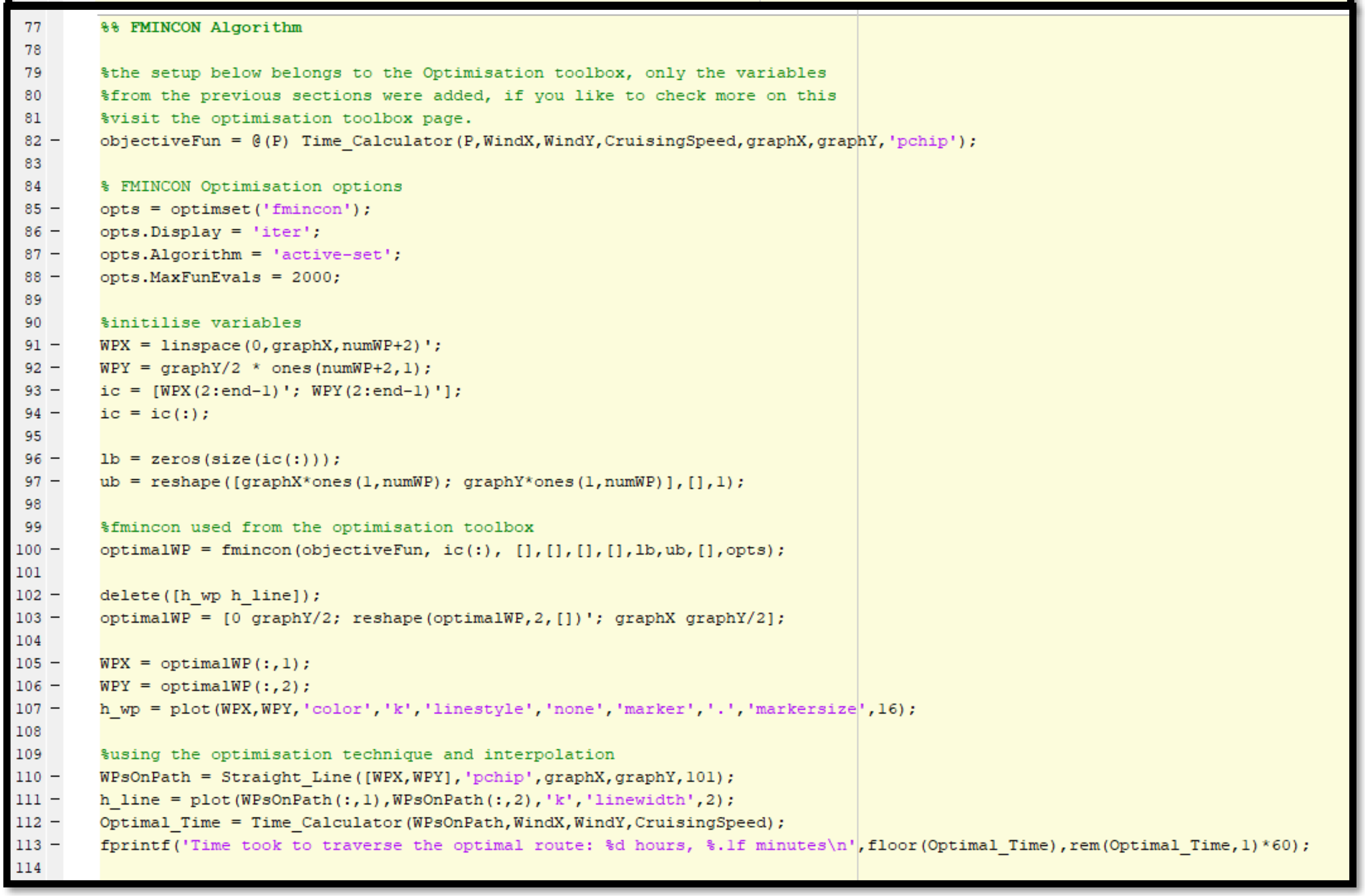
```
%% Creating WayPoints(WP)

numWP = 5;
%create number of way points (does not include start and finish)

WPX = linspace(0,graphX,numWP+2)';
WPY = graphY/2 * ones(numWP+2,1);
%now equally plot them over the graph if necessary adjusting WP count.

h_wp = plot(WPX,WPY,'color','k','linestyle','none','marker','.','markersize',16);
%plot wps

```



```
%% Using the previously created wps, form a path from the starting
%wp to finish using the Straight_Line function

WPsOnPath = Straight_Line([WPX,WPY],'linear',graphX,graphY,101);
h_line = plot(WPsOnPath(:,1),WPsOnPath(:,2),'k','linewidth',2);

```

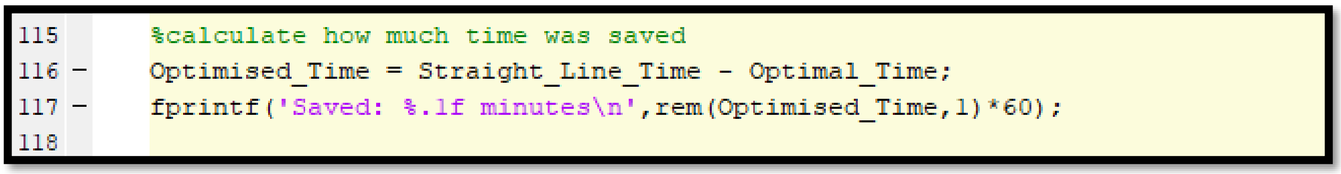


```
%% Using the straight line formation from the previous section, compute
%the amount of time it takes to reach

Straight_Line_Time = Time_Calculator(WPsOnPath,WindX,WindY,CruisingSpeed);

fprintf('Time took to traverse the great circle distance: %d hours, %.1f minutes\n',floor(Straight_Line_Time),rem(Straight_Line_Time,1)*60);

```



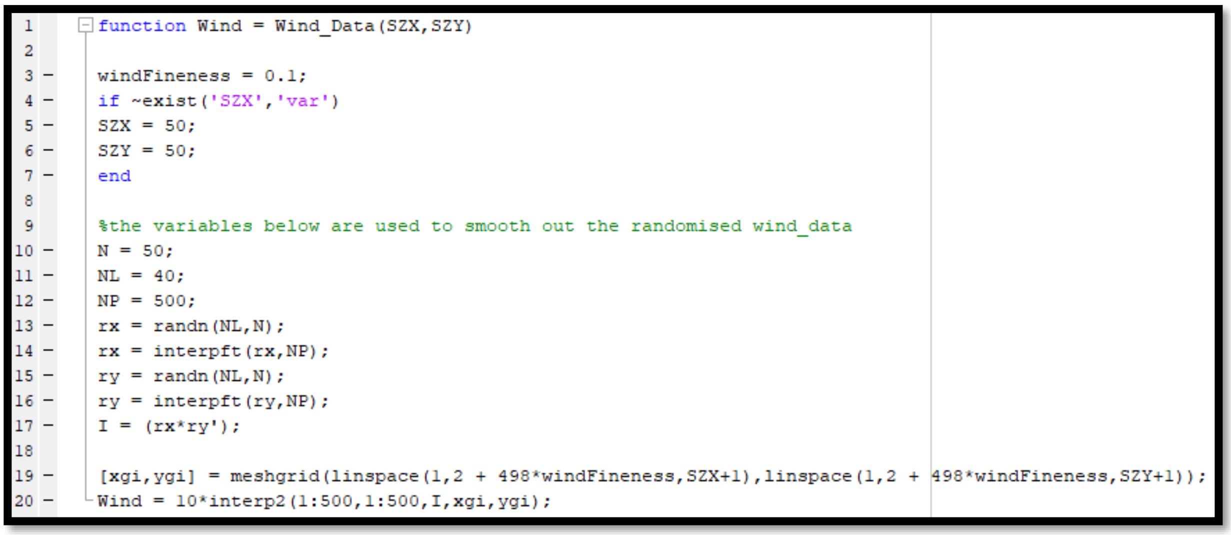
```
%% FMINCON Algorithm

%the setup below belongs to the Optimisation toolbox, only the variables
%from the previous sections were added, if you like to check more on this
%visit the optimisation toolbox page.
objectiveFun = @(P) Time_Calculator(P,WindX,WindY,CruisingSpeed,graphX,graphY,'pchip');

% FMINCON Optimisation options
opts = optimset('fmincon');
opts.Display = 'iter';
opts.Algorithm = 'active-set';
opts.MaxFunEvals = 2000;

%initilise variables
WPX = linspace(0,graphX,numWP+2)';
WPY = graphY/2 * ones(numWP+2,1);
ic = [WPX(2:end-1)'; WPY(2:end-1)'];
ic = ic(:);

lb = zeros(size(ic(:)));
ub = reshape([graphX*ones(1,numWP); graphY*ones(1,numWP)],[],1);

%fmincon used from the optimisation toolbox
optimalWP = fmincon(objectiveFun, ic(:), [],[],[],[],lb,ub,[],opts);

delete([h_wp h_line]);
optimalWP = [0 graphY/2; reshape(optimalWP,2,[])'; graphX graphY/2];

WPX = optimalWP(:,1);
WPY = optimalWP(:,2);
h_wp = plot(WPX,WPY,'color','k','linestyle','none','marker','.','markersize',16);

%using the optimisation technique and interpolation
WPsOnPath = Straight_Line([WPX,WPY],'pchip',graphX,graphY,101);
h_line = plot(WPsOnPath(:,1),WPsOnPath(:,2),'k','linewidth',2);
Optimal_Time = Time_Calculator(WPsOnPath,WindX,WindY,CruisingSpeed);
fprintf('Time took to traverse the optimal route: %d hours, %.1f minutes\n',floor(Optimal_Time),rem(Optimal_Time,1)*60);

```

Path_Optimisation_Gelbul.m continued…



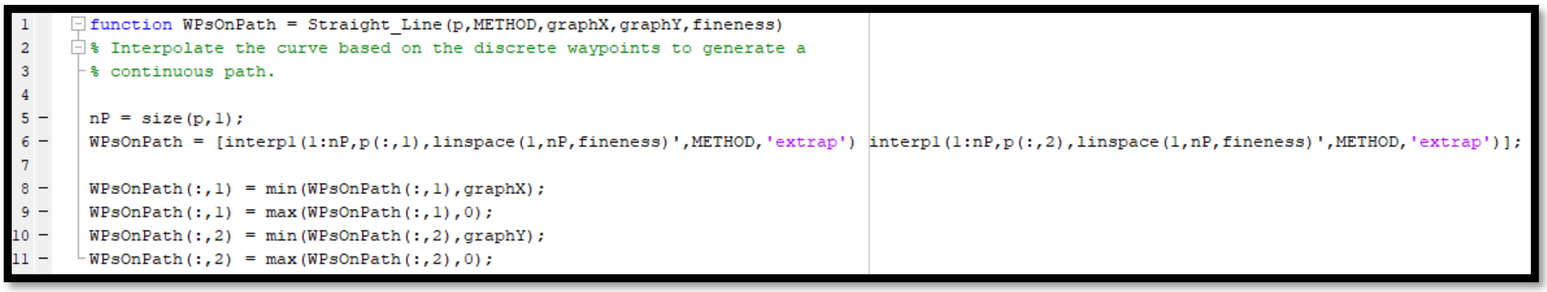
```
    %calculate how much time was saved
    Optimised_Time = Straight_Line_Time - Optimal_Time;
    fprintf('Saved: %.1f minutes\n',rem(Optimised_Time,1)*60);

```

Wind_Data.m



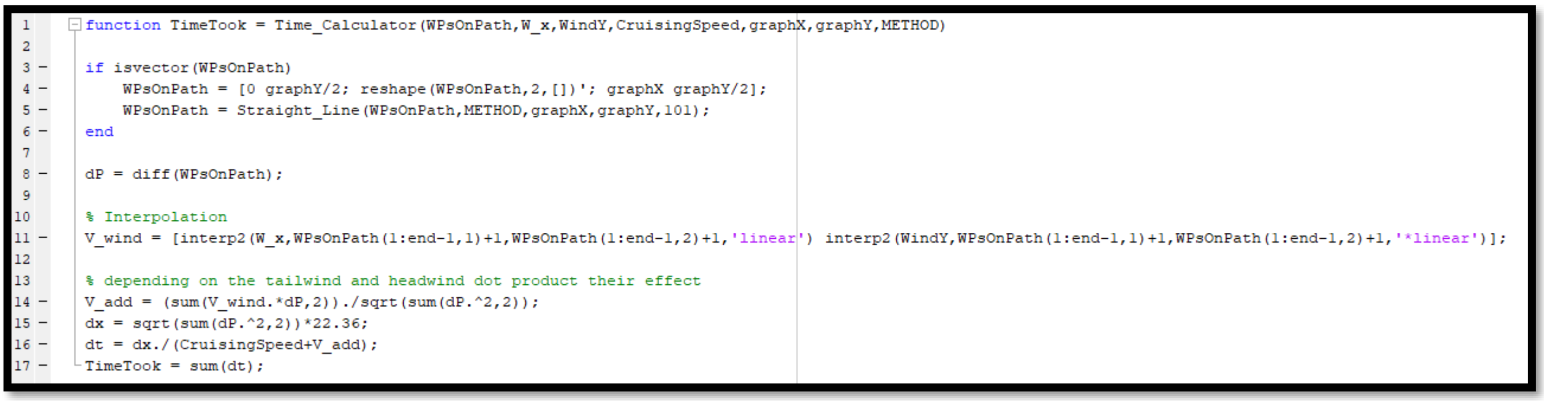
```
function Wind = Wind_Data(SZX,SZY)

windFineness = 0.1;
if ~exist('SZX','var')
SZX = 50;
SZY = 50;
end

%the variables below are used to smooth out the randomised wind_data
N = 50;
NL = 40;
NP = 500;
rx = randn(NL,N);
rx = interpft(rx,NP);
ry = randn(NL,N);
ry = interpft(ry,NP);
I = (rx*ry');

[xgi,ygi] = meshgrid(linspace(1,2 + 498*windFineness,SZX+1),linspace(1,2 + 498*windFineness,SZY+1));
Wind = 10*interp2(1:500,1:500,I,xgi,ygi);
```

Straight_Line.m



```
function WPsOnPath = Straight_Line(p,METHOD,graphX,graphY,fineness)
% Interpolate the curve based on the discrete waypoints to generate a
% continuous path.

nP = size(p,1);
WPsOnPath = [interp1(1:nP,p(:,1),linspace(1,nP,fineness)',METHOD,'extrap') interp1(1:nP,p(:,2),linspace(1,nP,fineness)',METHOD,'extrap')];

WPsOnPath(:,1) = min(WPsOnPath(:,1),graphX);
WPsOnPath(:,1) = max(WPsOnPath(:,1),0);
WPsOnPath(:,2) = min(WPsOnPath(:,2),graphY);
WPsOnPath(:,2) = max(WPsOnPath(:,2),0);
```

Time_Calculator.m



```
function TimeTook = Time_Calculator(WPsOnPath,W_x,WindY,CruisingSpeed,graphX,graphY,METHOD)

if isvector(WPsOnPath)
    WPsOnPath = [0 graphY/2; reshape(WPsOnPath,2,[])'; graphX graphY/2];
    WPsOnPath = Straight_Line(WPsOnPath,METHOD,graphX,graphY,101);
end

dP = diff(WPsOnPath);

% Interpolation
V_wind = [interp2(W_x,WPsOnPath(1:end-1,1)+1,WPsOnPath(1:end-1,2)+1,'linear') interp2(WindY,WPsOnPath(1:end-1,1)+1,WPsOnPath(1:end-1,2)+1,'*linear')];

% depending on the tailwind and headwind dot product their effect
V_add = (sum(V_wind.*dP,2))./sqrt(sum(dP.^2,2));
dx = sqrt(sum(dP.^2,2))*22.36;
dt = dx./(CruisingSpeed+V_add);
TimeTook = sum(dt);
```