# Detection of Atmospheric Gravity Waves Through High-Altitude Ballooning

A Thesis
Presented in Partial Fulfillment of the Requirements for the
Degree of Master of Science
with a
Major in Chemical Engineering
in the
College of Graduate Studies
University of Idaho
by

Jacquelin Martinez-Alvarez

Major Professor: Matthew Bernards, Ph.D.
Committee Members: James Moberly, Ph.D.; Russell Qualls, Ph.D.
Department Administrator: Dev S. Shrestha, Ph.D.

August 2021

## Authorization to Submit Thesis

This thesis, of Jacquelin Martinez-Alvarez, submitted for the degree of Master of Science with a Major in Chemical Engineering and titled "Detection of Atmospheric Gravity Waves Through High-Altitude Ballooning," has been reviewed in final form. Permission, as indicated by the signatures and dates below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____ Date: _____
Matthew Bernards, Ph.D.

Committee Members: _____ Date: _____
James Moberly, Ph.D.

_____ Date: _____
Russell Qualls, Ph.D.

Department Administrator: _____ Date: _____
Dev S. Shrestha, Ph.D.

## Abstract

Atmospheric gravity waves (AGWs) are transverse waves that propagate in all directions throughout the atmosphere. They arise from a multitude of sources such as: orography, fronts, and convection. AGWs are excellent transporters of energy and momentum, which influence atmospheric turbulence, temperature, and chemistry. Wave transport and the successive deposition of energy is an important component in atmospheric dynamics as they have a large impact on the spatial and temporal characteristics of the middle and upper atmosphere. As a result, they play a great role in daily weather and long-term climate fluctuations. However, the horizontal resolutions of global climate models lack the ability to resolve the scales of the important part of AGWs, and because of this, their contributions to global climate simulations are missed. Thus, their parameterization and quantification are essential for improving weather forecast models.

This thesis focuses on the detection of AGWs through high-altitude ballooning. In December 2020, radiosondes were sent as payloads through the lower and upper atmosphere, up to about 30 km, collecting sounding profiles in South America, before, during and after a total solar eclipse. Half of the radiosondes were launched from Tolten, while the other half were launched from Villarrica, providing an opportunity to detect AGWs around the Andes mountains and during the total solar eclipse. Each radiosonde was launched an hour apart over 48 consecutive hours. Through the collection of wind and temperature profiles, and using Euler's equations of motion, a wavelet-based method was used to decompose vertical profiles of horizontal wind into gravity wave packets. To parameterize the waves, intrinsic frequency, and horizontal and vertical propagation directions were determined using Stokes parameters. The detected AGWs were characterized into three main categories: AGWs caused by significant wind shears, mountain waves, and solar eclipse induced waves. These results provide parameters for multiple sources of AGWs that can be useful for climate change mitigation strategies through improved climate models, which contribute to the quality of life for millions.

## Acknowledgements

I would like to thank everyone in the Department of Chemical and Biological Engineering. I do not think I would have survived without the support of faculty, staff, and my fellow graduate students. Thank you to Dr. James Moberly and Dr. Russell Qualls for being part of my committee and taking the time to invest in my education. This work would not have been possible without the help of my collaborators, Jen Fowler, Carl Spangrude, Hannah Woody, Keaton Blair, and everyone else from the University of Montana, Oklahoma State University, and University of Kentucky. A special thank you to our Summer Interns: Alex Chambers, Malachi Mooney-Rivkin, Carlos Muñoz, Lauren Perla, Roslyn McCormack, Leah Davidson, Sebastian Garcia, and Kathryn Reece. I will miss spending my Summers launching balloons and eating waffles with you all. Thank you for all your hard work and dedication to this project. I feel lucky to have had such an amazing group of students. Lastly and most importantly, I would like to thank Dr. Matthew Bernards. Thank you for your mentorship, time, and support. You have been a major factor to my success, from editing my documents, numerous hours spent rehearsing presentations, to allowing me to talk your ear off, letting me cry in your office, helping me network, and overall helping me get through graduate school. You have invested more than I could ever repay you. Thank you for believing in me during the times I did not believe in myself. I am forever grateful!

## Dedication

Para Mi Familia y Amorcito. Si se pudo!

# Table of Contents

## List of Tables

## List of Figures

# Chapter 1: Introduction

## Motivation

The Global Forecast System (GFS) is the weather forecast system used by the National Centers for Environmental Prediction (NCEP). The entire globe is covered by GFS at a base horizontal resolution of 28 km between grid points. When extended to a one – two week forecast, the horizontal resolution drops to about 70 km between grid points. Atmospheric gravity waves (AGWs) range from hundreds of meters to a few kilometers; GFS does not have the horizontal resolution required to account for many important AGWs. It is highly important to incorporate AGWs in weather prediction models, as they are great transporters of momentum and energy through dissipation and propagation into vertical and horizontal regions. However, scientists have struggled to accurately parameterize and characterize such waves for their inclusion in these simulations. The motivation behind this work is to detect and characterize AGWs by developing a robust set of numerical models, for their future inclusion in systems like the GFS. It is through the inclusion of AGWs that things such as storms, hurricanes, etc. can be better predicted and warnings can be sent out earlier to keep people safe. This work will focus on the detection of AGWs induced by total solar eclipses.

## Introduction to the Atmosphere

The standard atmosphere is established from global averages of temperature, density, chemical composition of air, etc. [1]. It is divided into four main regions: troposphere, stratosphere mesosphere, and thermosphere. Each region is separated by transition zones. The transitional area between the troposphere and stratosphere is known as the tropopause. The area separating the stratosphere and mesosphere is the stratopause, and the mesopause separates the mesosphere and thermosphere, as shown in Figure 1.1.

Most weather-creating actions and day-to-day experiences happen in the troposphere and stratosphere with air density, pressure, and temperature being the three major weather inducers. The relationship between the three can be described by the Ideal Gas Law (IGL), shown in Equation 1.1.

$$p = \rho RT \tag{1.1}$$

The IGL states that pressure, p, is proportional to density, $\rho$, temperature, T, and the gas constant, R, for air. When one variable is fixed, relationships between the other two are apparent, for example, when density is constant, pressure increases with temperature. These relationships are important to understand moving forward as the properties of each region of the atmosphere changes with increasing altitude.

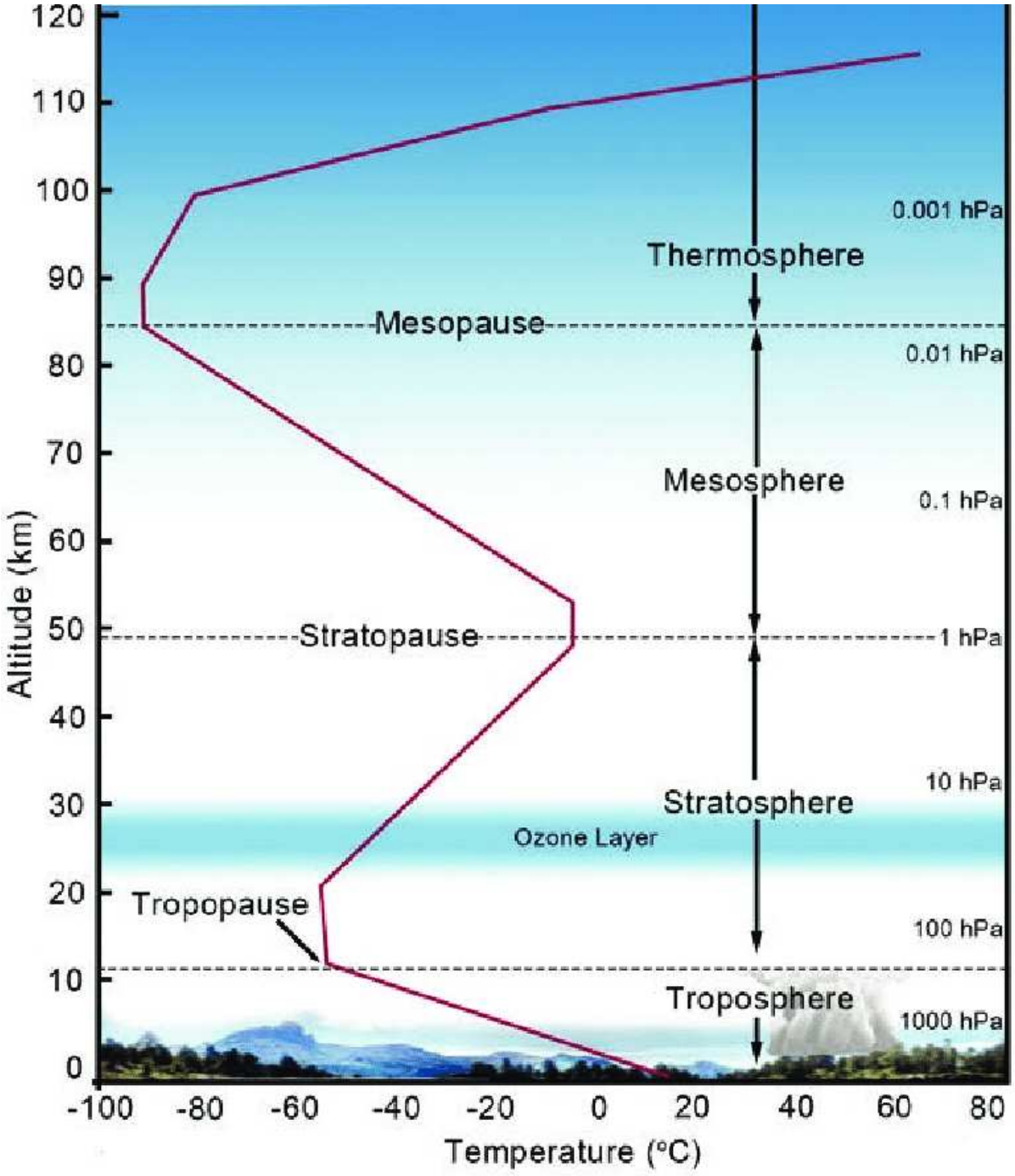



**Figure 1.1.** The standard atmosphere [2].

As seen in Figure 1.1, the lowest region of the atmosphere is the troposphere, rising from the surface to approximately 12 kilometers (km). Within this region, as altitude increases, the pressure decreases from approximately 1000 hectopascal (hPa) to 200 hPa, and temperature decreases from 20 °C to -60 °C. Described by the IGL, a pressure decrease in the troposphere also results in a drop in air

density. As a result, this layer is known to be the most turbulent and convective portion of the atmosphere. As a warm air parcel rises through the troposphere, it continues to rise until it cools down and becomes denser than the surrounding air. Once this occurs, it falls back down, thus creating a natural convective mixing effect.

The area following the troposphere is referred to as the stratosphere. This area extends from approximately 12 km to 50 km. Pressure ranges from 100 mb to 1 mb, however, unlike the troposphere, temperature increases from -60 °C to 0 °C. This is due to tropopause inversion layers (TIL) [3]. Temperature strongly increases just above a sharp local cold point tropopause [4] due to the sun's radiation being absorbed in the stratosphere's ozone layer. As a warm air parcel moves up, it experiences the surrounding air increasing in temperature, and will rise until it meets an equivalent ambient temperature.

Beyond the stratosphere lies the mesosphere, covering 50 km to 85 km, and the thermosphere, covering 85 km up. Similar to the troposphere, the temperature in the mesosphere decreases with altitude, and similar to the stratosphere, the temperature in the thermosphere increases with altitude. Overall, air density decreases as altitude increases. The combination of all four regions described and the variation of temperatures and pressures within them is what makes up a stably stratified, or stably layered, atmosphere.

**Introduction to Atmospheric Gravity Waves**

The atmosphere is almost always a stably stratified fluid [5]. Consider the atmosphere at rest and an air parcel in equilibrium with its environment. When that air parcel comes across a disturbance it is displaced by some distance. It is assumed that during this displacement the air in the parcel does not mix with its surroundings and there is no net heat transfer. The air parcel will oscillate adiabatically due to the gravitational force trying to restore equilibrium. This phenomenon is known as atmospheric gravity waves (AGWs). AGWs can only exist when the atmosphere is stably stratified [6]. Disturbances arise from many sources such as mountains, storm fronts, convection, wind shear, eclipses, etc. A representative image of AGWs can be seen in Figure 1.2.

The frequency associated with this oscillation is called the Brunt-Väisälä frequency, $N$. When squared, $N^2$, this frequency is a measure of the atmospheric static stability [3]. Static stability is defined as the stability of the atmosphere in a hydrostatic equilibrium with respect to vertical displacements [7]. It is important because it greatly influences the generation, propagation, and dissipation of AGWs. It acts as a pathway between the background atmosphere and AGWs. For example, as a parcel of air moves it is subject to minor changes in the forces that act on it and its

speed. If the force causes further changes that tend to restore the parcel of air to its original speed and orientation, without external machine or human input, the parcel of air is said to be statically stable. If such changes cause further changes that tend to drive the parcel of air away from its original speed and orientation, the parcel of air is said to be statically unstable.

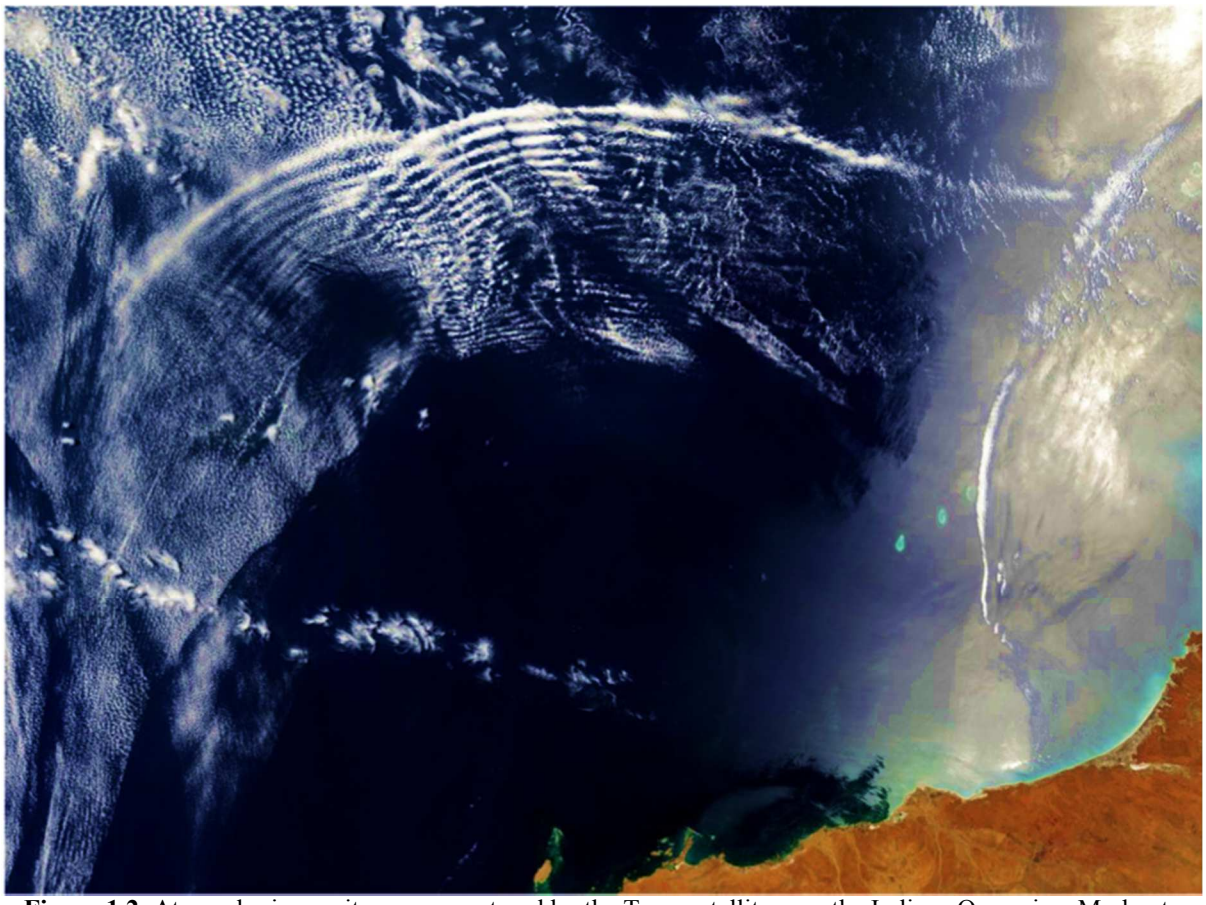


**Figure 1.2**. Atmospheric gravity waves captured by the Terra satellite over the Indiana Ocean in a Moderate Resolution Imaging Spectroradiometer (MODIS) image [8].

When a fluid is bounded above and below, wave propagation happens in the horizontal plane. Those that propagate vertically will often be referred to as a standing wave, however, the atmosphere has no upper bound. In this case, AGWs are able to propagate horizontally and vertically. Waves that have the ability to propagate vertically, in fluid with only one bound, are referred to as internal waves. Internal AGWs are transverse waves, meaning that the paths of parcel oscillations are parallel to the phase lines in the plane of reference [6]. Their phase is a function of height. Vertical propagation plays a huge rule in the atmosphere, especially because AGWs are great transporters of energy and momentum. Consider a parcel of air encountering a mountain range, that mountain will force the

parcel of air to move upward. It moves from a dense atmospheric layer to a thinner one. The heavier parcels of air will be forced downward by gravity resulting in a periodic oscillation [9]. When this happens, that transmission of motion throughout the atmosphere can be directed horizontally and vertically. AGWs will transport horizontal momentum vertically over many density scale heights. As mentioned before, density in the atmosphere decreases with altitude, thus, to conserve wave energy, the amplitude of AGWs grows with altitude. Small perturbations in the lower atmosphere will become larger in the middle and upper atmosphere, and deposit momentum and energy in unoriginal regions.

Momentum and energy transported from the troposphere to the stratosphere, and above, affect global circulation models. Spatial and temporal characteristics of the middle and upper atmosphere are impacted, and many numerical models or simulations do not contain the horizontal resolutions needed to account for AGWs. To date, the highest model resolution is 20 km [10]. This is much higher than the wavelengths of some important AGWs, as they often fall below the range of 20 km, with typical wavelengths anywhere from hundreds of meters to a few kilometers.

## High-Altitude Weather Ballooning

To date, the most accessible way of recording high temporal resolution lower and middle atmospheric data needed for AGW parameterization is through high-altitude weather ballooning. Weather balloons are manufactured from latex and require helium or hydrogen as a lift gas. They come in many different sizes ranging from 200 grams to 1500 grams. When inflated they range in diameters of 2.5 feet to 8 feet, however, during a flight their diameters can increase up to four times that size. When released into the atmosphere, most weather balloons reach heights of up to 30-40 kilometers. The main component of high-altitude weather ballooning is the payload. For atmospheric science studies, the payload is typically a 20 cm by 4 cm by 6 cm radiosonde package, which is capable of gathering data over horizontal distances up to 300 km. An example of a radiosonde is shown in Figure 1.3.

**Figure 1.3**. Radiosonde package with meteorological sensors located on the boom (long silver piece). Arduino-type signal-processing electronic enclosed inside of a Styrofoam housing.

Radiosondes measure the vertical profile of meteorological variables such as temperature, pressure, relative humidity, air density, wind speed and wind direction. A radiosonde rising through a gravity wave will experience elliptical motions. The elliptical motion is described by the polarization relations for gravity waves, which are described in more detail in Chapter 2. Parameters are calculated based on the elliptical characteristics of wind data gathered from these radiosondes.

The electronic unit consists of three major sections: meteorological sensors, signal-processing electronics, and a radio transmitter to send data back to a receiver. The data are transmitted to a ground-based receiving and processing station with an accuracy for temperature, relative humidity, and pressure of ± 1°C, ± 1%, and ± 3 hPa, respectively [11]. However, this accuracy is based on the following key requirements. First, the train connecting the balloon and radiosonde should be long enough to avoid any wake effect. Second, surface values are to be collected for sensor calibration. Most importantly, the ascent of the apparatus should be maintained at approximately a 5 m/s rise rate. This is the optimal rise rate for which these electronic units are calibrated. See Figure 1.4 for a representation of the overall system being sent into the atmosphere. The de-reeler slowly unwinds to create more than 30 meters between the balloon and radiosonde to avoid wake affects from the balloon.

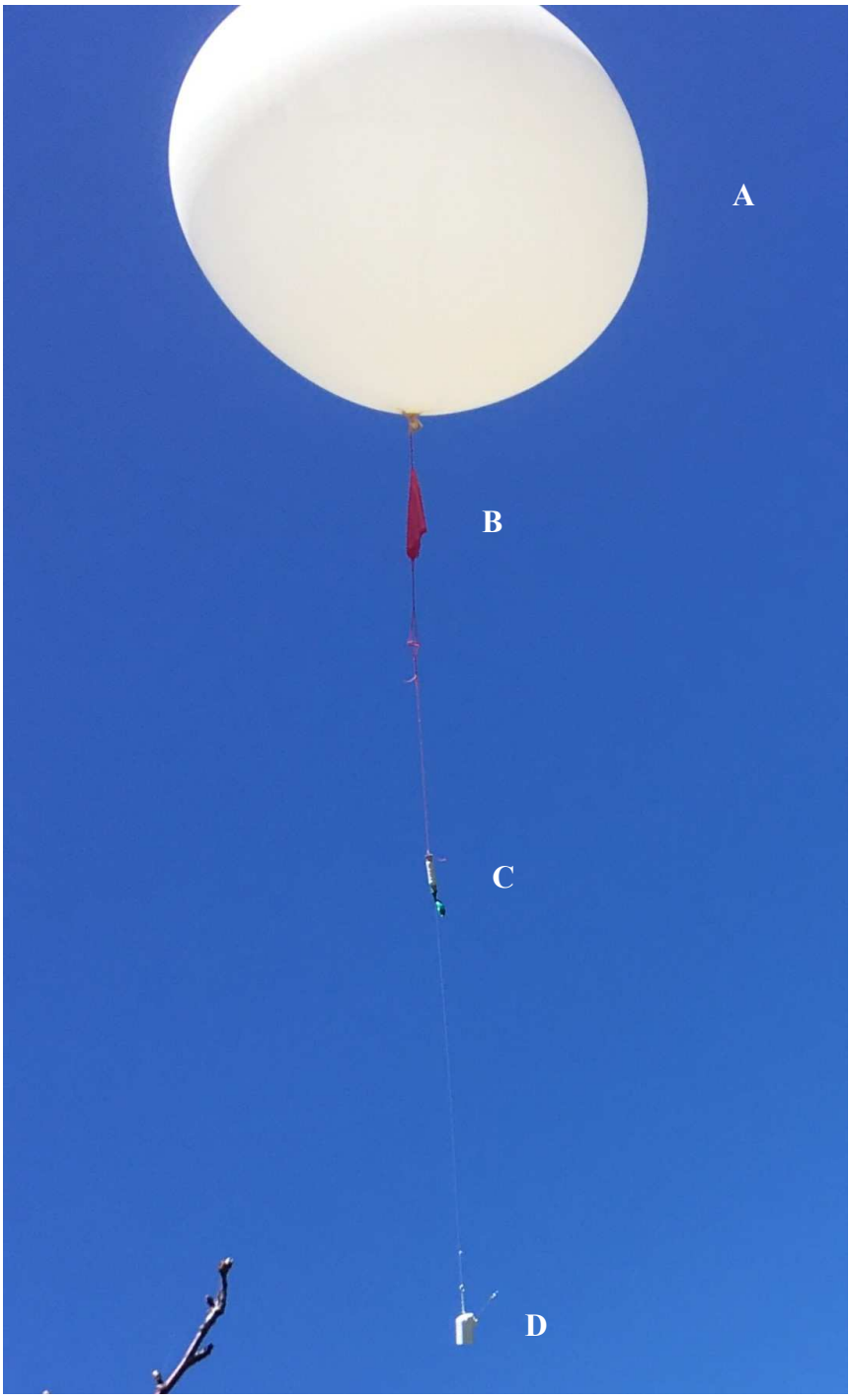



**Figure 1.4.** High-altitude ballooning set-up. The components are (A) latex weather balloon, (B) parachute, (C) de-reeler, (D) radiosonde.

## Common GW Detection Methods Used

There are two methods most commonly used for extracting wave parameters from radiosonde profiles: the wavelet method and hodograph method. This thesis focuses on the wavelet method and a more detailed explanation will be seen in Chapter 2. Briefly, the wavelet method performs a wavelet transform and applies Stokes parameters to the transformed data to extract wave parameters, such as intrinsic frequency, propagation direction, wavelength, etc.

In the hodograph method, zonal and meridional wind data is plotted on an x-y axis. The plots will form an ellipse in the presence of low frequency gravity waves [12]. To fit the data, a least-square fit method is used, however, other ways of data fitting can be done. Wave parameters are extracted from the fit. The horizontal propagation direction is the tilt angle from the least-square fit. The ratio of the semi-major to semi-minor axes is proportional to the intrinsic frequency scaled by the Coriolis frequency, and the vertical wavelength is found from the vertical extent of the ellipse. The hodograph method has been widely used for AGW analysis, however there are limitations to this method. Low frequency waves are targeted by the hodograph method and ellipse identification is typically done by hand. This introduces possible human error, leading to inconsistencies.

## Research Goal

Disturbances in a stably stratified atmosphere can occur from a solar eclipse. The obstruction of solar irradiance, during a solar eclipse, subsequently inducing AGWs was first proposed in 1970 by Chimonas *et al*. [13]. Since then, many scientists have tried to study the effects of solar eclipses on the earth's atmosphere [14-17]. However, the natural occurring phenomena can make data collection difficult, as they only occur in limited places and times. The objective of this research is to develop a set of methods in an effort to detect AGWs during a solar eclipse. Various balloon-launches will be performed to collect sounding profiles. Launches will occur through-out the Summer of 2020 in Moscow, Idaho, with the goal of preparing for a consecutive 48-hour long campaign in Chile for the total solar eclipse on December 14, 2020. A mathematical model based off the wavelet method, to improve detection of AGWs, will be implemented in MATLAB. It is also of interest to characterize AGWs and identify their sources to find correlations between wave parameters associated with significant wind shears, topography induced waves, and solar eclipse induced waves.

# Chapter 2: Theory

## Euler's Equations of Motion

Atmospheric observations are made in a coordinate system fixed to the earth. The reference frame being used for this analysis is an Eulerian reference frame. This frame of reference is stationary relative to the flow, which is useful for extracting values such as velocity, density, temperature, etc. at fixed points in space as time passes. It is assumed that the atmosphere is irrotational, frictionless, and adiabatic. Thus, Euler's equations of motion for an irrotational, frictionless, and adiabatic atmosphere are used to derive the equations for identifying gravity waves.

*Conservation of mass*

In fluid mechanics, fluid flows into and away from regions. Euler's equation for the conservation of mass explicitly accounts for the flow of mass. Using a cartesian coordinate system, one can consider the control volume of an infinitesimal, rectangular cuboid that is fixed in space, as shown in Figure 2.1. The horizontal plane is x and y, while z lies in the vertical direction. The coordinates have unit vectors of $\hat{x}, \hat{y}, \hat{z}$, and velocities of $u$, $v$, $w$, respectively.

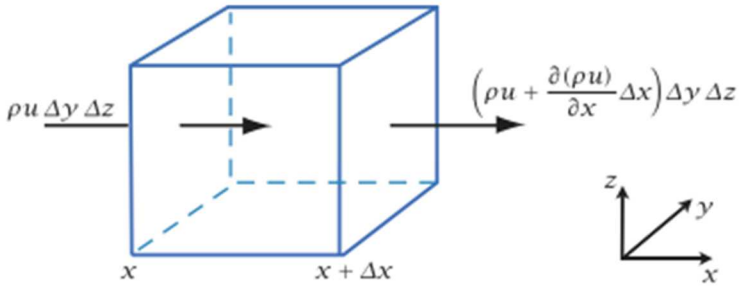



**Figure 2.1**. Control volume representing mass flow in and out of a rectangular cuboid. [19]

Figure 2.1 is well known in all transport books, as it aids in the derivation of the mass continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \vec{U}\right) = 0 \tag{2.1}$$

where $\vec{U}$ is the velocity field.

The mass continuity equation can also be derived from a material perspective. Consider a small parcel of air. The conservation of mass is represented by Equation 2.2.

$$\frac{D}{Dt}(\rho \Delta V) = 0 \tag{2.2}$$

Where $\frac{D}{Dt}$ is the material derivative defined by $\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{U} \cdot \nabla$, and V is the volume.

If both the density and the volume of the parcel can vary, then

$$\Delta V \frac{D\rho}{Dt} + \rho \frac{D\Delta V}{Dt} = \Delta V \left( \frac{D\rho}{Dt} + \nabla \rho \cdot \vec{U} \right) = 0 \tag{2.3}$$

The volume element is again arbitrary; thus Equation 2.3 becomes Equation 2.4, which is equivalent to Equation 2.1.

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \vec{U} = 0 \tag{2.4}$$

*Conservation of momentum*

As described in chapter 1, AGWs are great transporters of momentum. Euler's conservation of momentum equation describes how fluid velocity responds to internal and external forces. Consider the same infinitesimal parcel of air described above. Let *m(x, y, z, t)* be the momentum per unit volume of the fluid, where the total momentum is given by Equation 2.5.

$$\int_V m \, dV \tag{2.5}$$

The rate of change of the momentum for a fluid mass is described by the material derivative, which is equal to the force, F, acting on it as described by Newton's second law.

$$\frac{D}{Dt} \int_V \rho \vec{U} \, dV = \int_V F \, dV \tag{2.6}$$

Equation 2.6 can be rearranged into Equation 2.7.

$$\int_V \left( \rho \frac{D\vec{U}}{Dt} - F \right) dV = 0 \tag{2.7}$$

However, the volume is arbitrary, thus Equation 2.7 is transformed to Equation 2.8.

$$\rho \frac{D\vec{U}}{Dt} = F \tag{2.8}$$

It is difficult to directly quantify force in this term as not all forces are external to the fluid. This is where contact forces such as pressure and viscosity come into play. At the boundary of the fluid the pressure is the normal force per unit area.

$$d\vec{F_p} = -p \, dS \tag{2.9}$$

Where $\widehat{F_p}$ is the pressure force, and dS is a surface element.

Equation 2.9 can be rearranged to show the pressure force on a volume of fluid is the integral of the pressure over its surface.

$$\widehat{F_p} = -\int_S \; p \, dS \tag{2.10}$$

Applying Gauss's theorem, which relates the flux of a vector field through a closed surface to the divergence of the field in the volume enclosed [18], Equation 2.10 is rearranged into the following:

$$\widehat{F_p} = -\int_V \; \nabla p \, dV \tag{2.11}$$

The force is non-zero only if the pressure varies in space, therefore it is known as the pressure-gradient force [18].

In large-scale atmospheric flow, the viscous effects are negligible. Therefore, when combining external and internal forces, the force is described as:

$$F = \; \widehat{F_p} + \widehat{F_g} \tag{2.12}$$

Where $\widehat{F_g}$ is the force due to gravity.

The conservation of momentum equation is described in Equation 2.13.

$$\rho \frac{D\vec{U}}{Dt} = -\nabla p + \rho \vec{g} \tag{2.13}$$

Equation 2.13 can be broken up into each individual component. The momentum equations for the flow in the x and y direction, u and v, result in the following.

$$\frac{Du}{Dt} = -\frac{1}{\rho}\frac{\partial \rho}{\partial x} \tag{2.14}$$

$$\frac{Dv}{Dt} = -\frac{1}{\rho}\frac{\partial \rho}{\partial y} \tag{2.15}$$

Unlike the velocities in the x and y direction, the vertical component, *w*, is parallel to the gravitational force, thus resulting in Equation 2.16.

$$\frac{Dw}{Dt} = -\frac{1}{\rho}\frac{\partial \rho}{\partial z} - g \tag{2.16}$$

Equations 2.14 - 2.16 are some of the most important equations for the remainder of this document as they will be used in the wavelet analysis for detecting AGWs. These velocities can also be thought of

in terms of north, east, south, west, up and down velocities, where *u* is the velocity of the fluid in the east to west direction, *v* is the velocity of the fluid in the north to south direction, and *w* is the velocity of the fluid upward and downward.

*Conservation of energy*

The conservation of energy is described as Equation 2.17.

$$\frac{Dp}{Dt} - c_s^2 \frac{D\rho}{Dt} = 0 \tag{2.17}$$

Where $c_s^2$ is the speed of sound.

Equation 2.17 shows that density and pressure are related, which will become important for completely ruling out acoustic waves in the atmosphere.

## Wave fundamentals

Waves can be described using various coordinate systems; however, this thesis uses cartesian coordinates of x, y, z, as these are the standards in atmospheric science. A wave is created by periodic oscillations of fluid particles. These fluid particles oscillate on surfaces perpendicular to the direction of travel of the wave. In general, characteristics that describe a wave are frequency, length, amplitude, period, phase, and speed, as shown in Figure 2.2. A single wave is referred to as a periodic disturbance described by a single frequency without considering the number of cycles of the disturbance [19]. When referring to waves, it considers a group made up of different frequencies, amplitudes, lengths, and periods, otherwise known as a wave-packet.

Wave frequency, ω, can be thought of as the number of waves that pass a fixed point in a given amount of time. The more waves that pass, the higher the frequency is, and the more energy it contains. A low-frequency wave has less energy than a high-frequency wave with the same amplitude, or height of crest. Wavelength, λ, is the distance between any two identical points on the wave, or between following crests. The amplitude, A, of a wave is the difference in height between the crest and resting position. The higher the crest, the greater the amplitude is. Wave period, τ, is the time required to make one oscillation, or to go from point a to point b, as shown in Figure 2.2. This should not be confused with the wavelength or frequency. Although they are related, the wavelength is the distance of one propagation and the frequency is the inverse of the wave period.

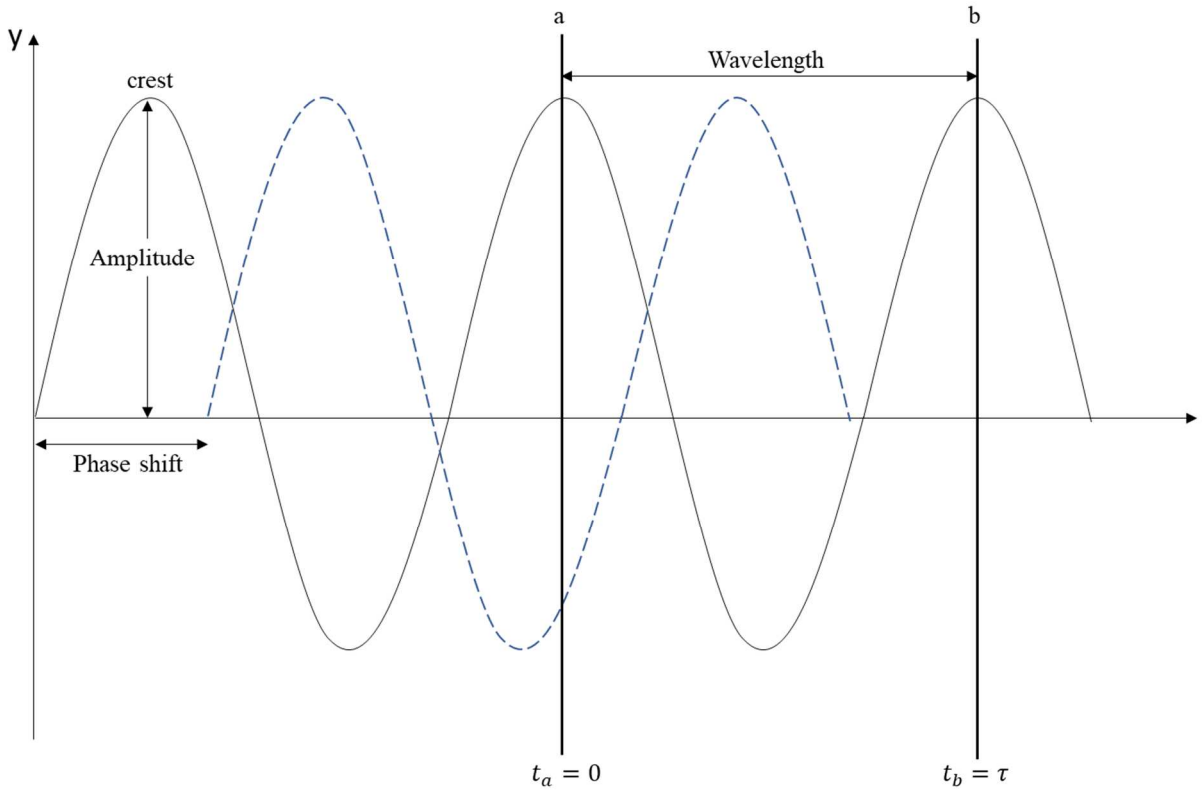



**Figure 2.2**. Characteristics describing a wave.

Furthermore, another characteristic of waves is the wavenumbers, which are denoted as k, l and m for x, y and z coordinates, respectively. The wavenumbers are defined as follows:

$$k = \frac{2\pi}{\lambda_x} \tag{2.18}$$

$$l = \frac{2\pi}{\lambda_y} \tag{2.19}$$

$$m = \frac{2\pi}{\lambda_z} \tag{2.20}$$

Where $\lambda_x$, $\lambda_y$, and $\lambda_z$ are the wavelengths in the x, y and z direction, respectively.

Wavenumbers are essential to describe the direction a wave will travel. This is known as a wave vector, and it takes the form shown in Equation 2.21.

$$\vec{k} = k\hat{x} + l\hat{y} + m\hat{z} \tag{2.21}$$

Equation 2.21 is also referred to as the propagation direction vector.

Wave propagation is mostly described in a fluid bounded horizontal plane, however there are cases when a fluid is not bounded, thus resulting in horizontal and vertical propagation. In polar coordinates, this wave propagation is known as the wave phase, or phase angle, ϕ.

Consider a wave in a one-dimensional plane described by $Acos(kx - \omega t)$. One oscillation of the wave is a cycle of 2π radians, and each point in that cycle is a phase point. In a two-dimensional plane, phase angle is the angle between the radius vector, $\vec{r}$, and horizontal axis, as shown in Figure 2.3.

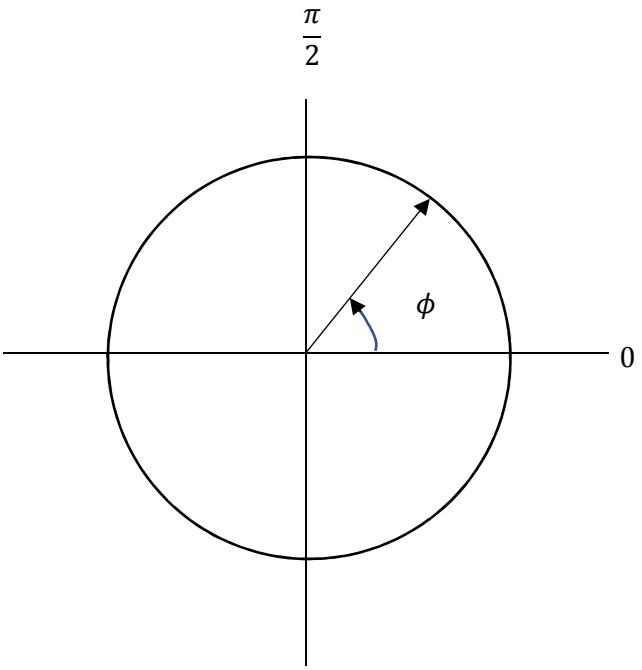



**Figure 2.3**. An illustration of phase angle in polar coordinates.

The phase angle takes the form of

$$\phi = \vec{k} \cdot \vec{r} - \omega t = kx + mz - \omega t \tag{2.22}$$

Where

$$\vec{r} = x\vec{x} + z\vec{z} \tag{2.23}$$

These relations are important for mathematically understanding propagation in cartesian coordinates.

In a two-dimensional (x, z) plane, the wave considered above is written as

$$f(x,z,t) = \mathbb{R}Ae^{i\phi} = Acos(kx + mz - \omega t) \tag{2.24}$$

Where $\mathbb{R}$ is the real part of the complex number, however, for the remainder of the document, only the exponential notation will be used. Lines of constant phase angles connect the same points in the wave field.

Another characteristic of waves is their speed. Consider following a point on a wave such as a crest. That crest goes from point a, at time $t$, to point b, at time $t + \Delta t$. The speed of a wave moving through a fluid at which a point of constant phase moves in the direction of a traveling wave is called the phase speed, c. The speed can be obtained by differentiating Equation 2.22, with respect to t while holding z constant.

$$\left.\frac{d\phi}{dt}\right|_{\phi,z} = k\frac{dx}{dt} - \omega = 0 \tag{2.25}$$

The phase speed in the x-direction can be defined as

$$c = \frac{\omega}{k} \tag{2.26}$$

Long waves travel faster than short waves, and when comprised together, they move at different phase speeds. The total energy of the wave-packet remains constant but disperses horizontally. The relation between phase speed and direction is known as the dispersion relation.

The speed and direction of energy transport is determined by the group velocity, $u_g$. As mentioned before, waves tend to move in wave-packets. Within this wave-packet there are waves moving at a speed proportional to its wavelength. The packet is made up of linear waves with varying wavelengths, and the dominant wave defining the packet has a local phase and frequency. The phase is a function of time and distance, and can be described with the following:

$$\frac{\partial \phi}{\partial \mathrm{x}} = -k \tag{2.27}$$

$$\frac{\partial \phi}{\partial \mathrm{t}} = \omega \tag{2.28}$$

Equation 2.27 and 2.28 are combined to form

$$\frac{\partial \mathrm{k}}{\partial \mathrm{t}} + \frac{\partial \omega}{\partial x} = 0 \tag{2.29}$$

Considering $\omega$ as a function of k yields the following:

$$\frac{\partial \mathrm{k}}{\partial \mathrm{t}} + \frac{\partial \omega}{\partial k}\frac{\partial k}{\partial x} = 0 \tag{2.30}$$

The group velocity is defined as

$$u_g = \frac{\partial \omega}{\partial k} \tag{2.31}$$

thus Equation 2.30 becomes:

$$\frac{\partial k}{\partial t} + u_g \frac{\partial k}{\partial x} = \frac{D_g k}{Dt} = 0 \tag{2.32}$$

The dominant wave number is constant when moving at the group velocity.

The relations developed are important for analyzing AGWs, as they will be distinguished based on the characteristics described above.

**Boussinesq Approximation**

AGWs are not the only type of wave found in the atmosphere. Acoustic waves are also very much present, and this can make AGW detection difficult. However, through the Boussinesq approximation, acoustic waves are eliminated.

Taking the conservation of momentum, Equation 2.13, the substitutions of $\rho = \rho_0 + \rho_1$ and $p = p_0 + p_1$ are made. This substitution assumes that density and pressure are a function of a slowly varying background, $\rho_0$ and $p_0$, and perturbation, $\rho_1$ and $p_1$, which will be discussed in further detail in the Linear Theory section below. Additionally, the background state is assumed to be in hydrostatic equilibrium because there is a balance between the pressure upward and the weight of the atmosphere. This yields Equation 2.33.

$$\left(1 + \frac{\rho_1}{\rho_0}\right)\frac{D\vec{U}}{Dt} = -\frac{1}{\rho_0}\nabla p_1 + \frac{\rho_1}{\rho_0}\vec{g} \tag{2.33}$$

It is assumed that $\left|\frac{\rho_1}{\rho_0}\right| < 1$, which allows for only density fluctuations to be considered when they occur in combination with gravity. This condition is satisfied when the fluid is confined to a layer whose thickness, $\Delta z$, is significantly less than the isothermal scale height of the atmosphere, $H_s$. The isothermal scale height is defined as

$$H_s = \frac{R_d T}{g} \tag{2.34}$$

Where $R_d$ is the universal gas constant for dry air.

For example, consider a vertical variation of density in an isothermal atmosphere.

$$\rho = \rho_s e^{-\frac{z}{H_s}} \tag{2.35}$$

Taking the derivative of Equation 2.35 results in Equation 2.36.

$$\frac{\partial \rho}{\partial z} = -\frac{\rho}{H_s} \tag{2.36}$$

By replacing the derivative with a differential, Equation 2.36 becomes Equation 2.37.

$$\frac{\Delta \rho}{\rho} = -\frac{\Delta z}{H_s} \tag{2.37}$$

Identifying $\Delta\rho$ as the density perturbation, $\rho_1$, yields the following:

$$\left|\frac{\rho_1}{\rho_0}\right| = \frac{\Delta z}{H_s} \tag{2.38}$$

Through this relation it is much more apparent that if $\Delta z < H_s$, then $\left|\frac{\rho_1}{\rho_0}\right| < 1$, thus proving the assumption.

The Boussinesq approximation allows one to neglect fluctuation changes in density due to pressure variations, thus a fluid can be treated as incompressible. However, when treating the fluid as incompressible $c_s \to \infty$ in Equation 2.17 and the conservation of energy can be broken into two terms. The Euler equations are then expressed as

$$\frac{D\vec{U}}{Dt} = -\frac{1}{\rho_0}\nabla p + \frac{\rho_1}{\rho_0}\vec{g} \tag{2.39}$$

$$\nabla \cdot \vec{U} = 0 \tag{2.40}$$

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} = 0 \tag{2.41}$$

Through the approximations made, sound waves are eliminated, and gravity waves are governed by Equation 2.39 – 2.41.

## Linear Theory

Differential equations are great for mathematical representation of variables, however, from a computational standpoint they can often be tedious to solve. Linear systems on the other hand are much more rapidly solved, as well as more understandable than nonlinear systems. Linear theory allows analytical solutions of wave equations. It predicts nonphysical behavior like significant wind shears, large temperature gradients, unrealistically stable flows, and more. Hines *et al.,* was the first to

publish, in 1960, on linear gravity wave theory where he explains the origins of irregular winds and turbulence in the middle and upper atmosphere [20].

Simply put, linear theory separates variables into constant or slowly varying background values and small first-order perturbations of these values. Using a two-dimensional reference plane (*x, z*) consider some variable $q$ is expanded into a background state $q_0$ and a first-order perturbation $q_1$, dependent on time and position. This yields Equation 2.42.

$$q(x,z,t) = q_0(z) + q_1(x,z,t) \tag{2.42}$$

The background state is steady or slowly varying vertically and horizontally uniform. The first-order perturbation is assumed to be much smaller than the background state and has no effect on the background state.

The governing equations expressing the momentum in the x and z directions are as follow:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial \rho}{\partial x} \tag{2.43}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho}\frac{\partial \rho}{\partial z} - g \tag{2.44}$$

The governing continuity equation is,

$$\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z} = 0 \tag{2.45}$$

and the conservation of mass is represented by Equation 2.46

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + w\frac{\partial \rho}{\partial z} = 0 \tag{2.46}$$

Equations 2.43 - 2.46 are linearized using the approach in Equation 2.42, which yield the following equations

$$\frac{\partial u_1}{\partial t} + u_0\frac{\partial u_1}{\partial x} + w_1\frac{\partial u_0}{\partial z} = -\frac{1}{\rho_0}\frac{\partial \rho_1}{\partial x} \tag{2.47}$$

$$\frac{\partial w_1}{\partial t} + u_0\frac{\partial w_1}{\partial x} = -\frac{1}{\rho_0}\frac{\partial \rho_1}{\partial z} - \frac{\rho_1}{\rho_0}g \tag{2.48}$$

$$\frac{\partial u_1}{\partial x} + \frac{\partial w_1}{\partial z} = 0 \tag{2.49}$$

$$\frac{\partial \rho_1}{\partial t} + u_0\frac{\partial \rho_1}{\partial x} + w_1\frac{\partial \rho_0}{\partial z} = 0 \tag{2.50}$$

The next step is to assume wave-like solutions of the form:

$$u_1(x,z,t) = u'(z)e^{-i(kx-\omega t)} \tag{2.51}$$

$$w_1(x,z,t) = w'(z)e^{-i(kx-\omega t)} \tag{2.52}$$

$$\rho_1(x,z,t) = \rho'(z)e^{-i(kx-\omega t)} \tag{2.53}$$

$$p_1(x,z,t) = p'(z)e^{-i(kx-\omega t)} \tag{2.54}$$

Where $u'$, $w'$, $\rho'$, and $p'$ are used to describe the real and imaginary part of $u_1, w_1, \rho_1$, and $p_1$, respectively. For example, $u' = u_r + iu_i$.

Equations 2.47 – 2.50 become:

$$-i\omega u' + iu_0 ku' + w'\frac{du_0}{dz} = -\frac{i}{\rho_0}kp' \tag{2.55}$$

$$-i\omega w' + iu_0 kw' = -\frac{1}{\rho_0}\frac{d\rho'}{dz} - \frac{\rho'}{\rho_0}g \tag{2.56}$$

$$iku' + \frac{dw'}{dz} = 0 \tag{2.57}$$

$$-i\omega\rho' + iu_0 k\rho' + w'\frac{dp_0}{dz} = 0 \tag{2.58}$$

The frequency included in the equations above is the apparent frequency measured by the observer in a fixed coordinate system. It is of interest to measure the frequency relative to the flow, or the intrinsic frequency, Ω, defined as:

$$\Omega = \omega - \mathrm{u}_0 k = \widehat{\omega} \tag{2.59}$$

where one horizontal background wind speed direction is included.

By substituting Equation 2.59 into Equations 2.55 – 2.58, the following are obtained

$$i\Omega u' - w'\frac{du_0}{dz} = \frac{i}{\rho_0}kp' \tag{2.60}$$

$$i\Omega w' = \frac{1}{\rho_0}\frac{d\rho'}{dz} + \frac{\rho'}{\rho_0}g \tag{2.61}$$

$$iku' + \frac{dw'}{dz} = 0 \tag{2.62}$$

$$i\Omega\rho' - w'\frac{p_0}{g}N^2 = 0 \tag{2.63}$$

where $N^2$ is the squared Brunt-Väisälä frequency defined as:

$$N^2 = -\frac{g}{\rho_0}\frac{dp_0}{dz} \tag{2.64}$$

Equations 2.60 – 2.63 are known as the polarization equations [20]. It is through these equations that the amplitude and phase of a wave, as described in the wave fundamentals section, can be obtained. Since $w'$ appears in Equations 2.60 – 2.63, the phases of other variables are relative to $w'$.

Assume a solution of the form

$$w'(z) = Ae^{imz} = A(\cos(mz) + isin(mz)) \tag{2.65}$$

The real part of $w'$ is $Acos(mz)$ and the imaginary part is $Asin(mz)$. Thus, taking the same approach to expand Equation 2.62 into real and imaginary parts yields Equation 2.66.

$$iku_R - ku_i = -Amcos(mz) - iAmsin(mz) \tag{2.66}$$

Considering only the real part yields the following

$$u_R = -A\frac{m}{k}\,cos(mz) \tag{2.67}$$

Depending on the sign of m, $u_R$ is either in phase with or 180° out of phase with $w'$.

Solving for $w'$ in Equations 2.60 – 2.63 gives

$$\frac{d^2w\prime}{dz^2} + \frac{1}{\rho_0}\frac{d\rho_0}{dz}\frac{dw\prime}{dz} + \left[\frac{k^2N^2}{\Omega^2} + \frac{k}{\Omega}\frac{d^2u_0}{dz^2} + \frac{k}{\Omega}\frac{1}{\rho_0}\frac{d\rho_0}{dz}\frac{du_0}{dz} - k^2\right] w' = 0 \tag{2.68}$$

It should be noted that the amplitude of AGWs grows with altitude in order to conserve energy. The second term in Equation 2.68 signifies the effect of varying atmospheric density on wave amplitude. To remove this height dependency, it is assumed that the atmospheric density decreases exponentially described by

$$\rho_0 = \rho_s e^{-\frac{z}{H_s}} \tag{2.69}$$

Where $\rho_s$ is the density at the ground surface, then Equation 2.68 becomes

$$\frac{d^2w\prime}{dz^2} - \frac{1}{H_s}\frac{dw'}{dz} + \left[\frac{k^2N^2}{\Omega^2} + \frac{k}{\Omega}\frac{d^2u_0}{dz^2} + \frac{k}{\Omega}\frac{1}{H_s}\frac{du_0}{dz} - k^2\right] w' = 0 \tag{2.70}$$

Using the integration factor,

$$e^{\int\left(\frac{1}{2H_s}\right)dz} \tag{2.71}$$

New variables, $\widehat{w}, \hat{u}, \hat{p}, \hat{\rho}$ are defined for $w'$, $u', p'$, and $\rho'$, respectively as seen in Equations 2.72 – 2.75.

$$w' = \widehat{w} e^{\left(\frac{z}{2H_s}\right)} \tag{2.72}$$

$$u' = \hat{u} e^{\left(\frac{z}{2H_s}\right)} \tag{2.73}$$

$$p' = \hat{p} e^{\left(\frac{z}{2H_s}\right)} \tag{2.74}$$

$$\rho' = \hat{\rho} e^{\left(\frac{z}{2H_s}\right)} \tag{2.75}$$

By substituting Equations 2.72 – 2.75 and replacing all derivatives with primes, Equation 2.70 becomes Equation 2.76.

$$\widehat{w}'' + \left[\frac{N^2}{(c-u_o)^2} + \frac{u_0''}{c-u_0} - \frac{1}{H_s}\frac{u_0'}{c-u_0} - \frac{1}{4H_s^2} - k^2\right]\widehat{w} = 0 \tag{2.76}$$

Replacing everything in the brackets by $m^2$ Equation 2.76 is simplified to Equation 2.77.

$$\widehat{w}'' + m^2\widehat{w} = 0 \tag{2.77}$$

If m is assumed to be constant, then the basis of linear theory is

$$\widehat{w} = Ae^{imz} + Be^{-imz} \tag{2.78}$$

The first term in the brackets of Equation 2.76, $\frac{N^2}{(c-u_o)^2}$, is the buoyancy term. This will dictate whether m is real or imaginary. When m is real, the amplitude of the vertical component of the wave perturbation velocity varies sinusoidally with height [19].

Through linear theory, one can solve for any perturbation variable. This thesis focuses on the perturbation velocities of the wind data gathered from the radiosonde. Zink and Vincent [21] simplify the polarization relations in a coordinate system aligned with the propagation direction of the wave. This gives the following relation

$$v'_{\perp} = -i\frac{f}{\Omega}u'_{||} \tag{2.79}$$

Where $v'_{\perp}$ is the perturbation velocity perpendicular to the wave vector, $u'_{||}$ is the perturbation velocity parallel to the wave vector, and $f$ is the Coriolis frequency, which accounts for the earth's rotation.

The vertical perturbation velocity is expressed as

$$w' = -i\frac{k_h}{m}u'_{||} \tag{2.80}$$

Where $k_h$ is the horizontal wave number.

Additionally, solving for temperature results in the following

$$\hat{T} = \frac{T'}{T_0} = i\frac{N^2}{g\Omega}\frac{k_h}{m}u'_{||} \tag{2.81}$$

Where $T'$ and $T_0$ are the perturbation and background temperatures, respectively.

By plotting $v'_{\perp}$ and $u'_{||}$ one would see those two sinusoids 90° out of phase form an ellipse. Therefore, gravity waves are said to be an elliptically polarized phenomena.

**Wavelet method**

A time series of perturbations consist of wave-like disturbances. As mentioned before, not all waves are identical. They vary in frequency and wavelength. They also do not all move in the same direction. More often than not, they propagate in every direction. It is important to remember that in the lower and middle atmosphere, a wave can break into smaller waves due to the turbulent-like disturbances happening when they cross the tropopause. Therefore, what sometimes may appear as a single wave is actually a wave-packet consisting of many wavelets. Methods to extract parameters from vertical wind and temperature profiles usually rely on the polarization relations of a monochromatic wave, a wave consisting of a single frequency, and single horizontal and vertical wavenumbers. Such a wave has neither a beginning nor an ending. It extends indefinitely in space and time. Realistically, waves do have a beginning, however they can extend over a large distance. This makes it difficult to solely analyze one complete wave.

Existing methods, such as spectral analysis, can only analyze parts of a AGW spectrum. Another method is decomposing the vertical profile into different wavelength bands [22-23]. The downfall to this is that wave detection is dependent on the height chosen. Narrow height bands favor the detection of small wavelengths, drowning the larger wavelengths in the background. Therefore, when larger bands are chosen, the larger wavelengths are detected and smaller wavelengths trend in the background, causing for an overlap in wave-packet detection [21].

The wavelet method is a great alternative to previous methods used. The benefit of it is that the window height is adaptable based on the wavelength being analyzed. Instead of decomposing vertical profiles of horizontal wind into wavelength bands, they are decomposed into gravity wave packets [21].

*Wavelet Transform [24]*

To isolate superimposed AGWs of different frequencies in a sounding, a wavelet transform is applied to the horizontal wind data. A wavelet transform is used to analyze time series that contain nonstationary power at different frequencies [25]. Consider a time series, $x_n$, where n = 0 ... N-1, with equal time spacing, $\delta t$, and a wavelet function, $\psi_0(\eta)$. The wavelet function most closely resembling AGWs is the Morlet wavelet, shown in Equation 2.82.

$$\psi_0(\eta) = \pi^{-\frac{1}{4}} e^{i\omega_0 \eta} e^{-\frac{\eta^2}{2}} \tag{2.82}$$

Where $\omega_0$ and $\eta$ are nondimensional frequency and nondimensional time respectively.

The Morlet wavelet is a continuous wavelet transform. It also has nonzero contributions meaning that its real and imaginary parts are 90° out of phase. The continuous wavelet transform of a discrete sequence $x_n$ is defined as the convolution of $x_n$with a scaled and translated version of $\psi_0(\eta)$

$$W_n(s) = \sum_{n'=0}^{N-1} x_{n'} \psi^* \left[\frac{(n'-n)\delta t}{s}\right] \tag{2.83}$$

Where * indicates the complex conjugate. A picture of amplitude versus the wavelet scale, s, can be constructed by varying s and translating along n. However, it is faster to calculate the wavelet transform in Fourier space.

The convolution is done N times for each scale. This allows the convolution theorem to do N convolutions in Fourier space using a discrete Fourier transform (DFT). The DFT of $x_n$ is seen in Equation 2.84.

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}} \tag{2.84}$$

Where k = 0…N-1 is the frequency index.

By the convolution theorem, the wavelet transform is the inverse Fourier transform of the product resulting in:

$$W_n(s) = \sum_{k=0}^{N-1} \hat{x}_k \, \widehat{\psi}^*(s\omega_k) e^{-i\omega_k n \delta t} \tag{2.85}$$

Where $\widehat{\psi}^*(s\omega_k)$ is the Fourier transform of a function $\psi\left(\frac{t}{s}\right)$ and $\omega_k$ is the angular frequency defined as

$$\omega_k = \begin{cases} \frac{2\pi k}{N\delta t} : \ k \leq \frac{N}{2} \\ -\frac{2\pi k}{N\delta t} : \ k > \frac{N}{2} \end{cases} \tag{2.86}$$

Equation 2.85 makes calculating the continuous wavelet transform at all n simultaneously possible. Normalization of the scale is necessary to ensure that the wavelet transforms at each scale are comparable to each other. The scale is normalized to have units of energy.

*Power Surface*

The wavelet transform can be divided into its real and imaginary parts. Its real part is referred to as the amplitude, $|W_n(s)|$, and the imaginary part is the phase. The wavelet power spectrum is the square of the real part of the wavelet transform, $|W_n(s)|^2$. The power spectrum is crucial in wave identification. The power spectrum is constructed from the wavelet coefficients of the vertical profiles of zonal and meridional wind components, U and V, respectively. By plotting vertical wavelength and power (y-axis) and altitude (x-axis), the surface can be scanned for local maxima. Those meeting a threshold corresponding to a perturbation amplitude of 0.1 m $s^{-1}$ are detected as wave packets. Anything lower than this threshold is deemed as noise.

As part of the power surface, the cone of influence is the region where edge effects become important. It is defined as the e-folding time for the autocorrelation of wavelet power at each scale. It is chosen to ensure that the wavelet power for a discontinuity at the edge drops by a factor of $e^{-2}$ and edge effects are negligible beyond this point [24].

*Stokes Parameters*

Wavelet transforms are a bandpass filter with a known response function, thus it is possible to reconstruct the time series using deconvolution or an inverse filter [24]. Continuous wavelet transforms can be reconstructed using a delta function, shown in Equation 2.87.

$$\delta = \frac{1}{C_\delta} \int_0^{+\infty} W_n \frac{ds}{s} \tag{2.87}$$

Where

$$C_\delta = \int_0^{+\infty} W_\delta \frac{ds}{s} \tag{2.88}$$

The reconstructed time series is the sum of the real part of the wavelet transform over all scales.

$$x_n = \frac{\delta j \delta t^{\frac{1}{2}}}{C_\delta \psi_0(0)} \sum_{j=0}^{j} \mathbb{R} \frac{\{W_n(s_j)\}}{s_j^{\frac{1}{2}}} \tag{2.89}$$

Where $s_j$ is the smallest scale of the form

$$s_j = s_0 2^{jsj}, \text{j} = 0,1,\ldots \text{J} \tag{2.90}$$

And J represents the largest scale of the form

$$J = \delta j^{-1} \log_2 \left(\frac{N\delta t}{s_0}\right) \tag{2.91}$$

The resulting reconstructed profiles contain the isolated wave packet as their real part and 90° phase-shifted version of the imaginary part. The vertical extent of each wave packet is the full width at half maximum of the horizontal wind variance of the reconstructed packet [21]. Wind and temperature perturbations associated with each wave are of interest to reconstruct, with a special interest in perturbation velocities, u' and v', shown in Equation 2.92 and 2.93, respectively.

$$u' = U(t)\cos(\omega t + \delta_1 t) \tag{2.92}$$

$$v' = V(t)\cos(\omega t + \delta_2 t) \tag{2.93}$$

Where U(t) and V(t) represent slowly varying amplitudes of the wave.

Through these reconstructed profiles, Stokes parameters can be calculated. Stokes parameters were first proposed in 1987 by Vincent and Fritts [26] to estimate gravity wave parameters. Stokes parameters are defined as:

$$I = u'^2 + v'^2 \tag{2.94}$$

$$D = u'^2 - v'^2 \tag{2.95}$$

$$P = UVcos\delta = 2u'v' \tag{2.96}$$

$$Q = UV'sin\delta \tag{2.97}$$

Waves with values of Q and P < 0.05 indicate weak wave activity and are discarded.

The degree of polarization, d, quantifies the contribution of coherent wave motion to the total velocity variance [12], and can be calculated as

$$d = \frac{(D^2+P^2+Q^2)^{\frac{1}{2}}}{I} \tag{2.98}$$

Waves with d < 0.5 indicate weak polarization, and d > 1 indicate a deviation of the data from the assumptions made to derive the Stokes parameters.

Additionally, other variables calculated from the Stokes parameters are the horizontal propagation direction and axial ratio (AR), Equation 2.99 and 2.100, respectively.

$$tan2\theta = \frac{P}{D} \tag{2.99}$$

$$\text{AR} = \cot(0.5 \sin^{-1}(\frac{Q}{Id})) \tag{2.100}$$

It is through these relations that waves are characterized in this work. A MATLAB code was designed to compute the wavelet transform and wave parameter calculations (See Appendix A).

# Chapter 3: Materials and Methods

Over the Summer of 2020, training launches were conducted using 600-gram Totex meteorological balloons. These balloon launches took place on the Tower Lawn at the University of Idaho, Moscow, Idaho (-46.73006, -117.0134). All, but one launch, took place during the day. These preliminary launches were aimed to train a team of students for the 48-hour campaign in Chile during the December 14, 2020 total solar eclipse, and to validate all of the operating procedures.

## Collecting ground measurements

Ground measurements were collected through a Lufft ground weather station (WS502-UMB Smart Weather Sensor). It is a compact weather sensor capable of measuring temperature, relative humidity, air pressure, wind direction, wind speed and radiation. Attached to the Lufft is a built-in housing for the CR300 data-logger used to interface with the Lufft and a battery to power the overall system. To output the data, PC400 software was used on a standard PC. It should be noted that both the Lufft and data-logger were set to a baudrate of 1200 Bd and SDI-12, which is an asynchronous serial communications protocol for intelligent sensors that monitor environmental data. A standard operating procedure for the Lufft set-up can be seen in Appendix B.

## Ground-based receiving and processing station

To receive data from radiosondes, a GS-U groundstation from GRAW was used. This radio receiver operates at a frequency range of 400 – 406 MHz. Data was continuously collected via the radiosonde and measurements were sent to the groundstation at intervals of one second. The GS-U was fully controlled via USB by GRAWMET software 5.15, a user interface specifically designed for the groundstation. It is through this software that sounding profiles were obtained. An SOP for running GRAWMET and initializing a radiosonde can be found in Appendix C.

## Helium fill

The sondes are pre-calibrated for an average rise rate of 5 m/s. The rise rate of the balloon is highly dependent on the amount of helium that goes into it. For example, too much helium will cause the balloon to rise at high speeds, but too little will result in slow travel. Knowing how much helium to add is complex as it is dependent on several variables such as drag force. To tackle this problem, the balloon and payload are treated as a statics problem, as seen in Figure 3.1 and Equation 3.1.

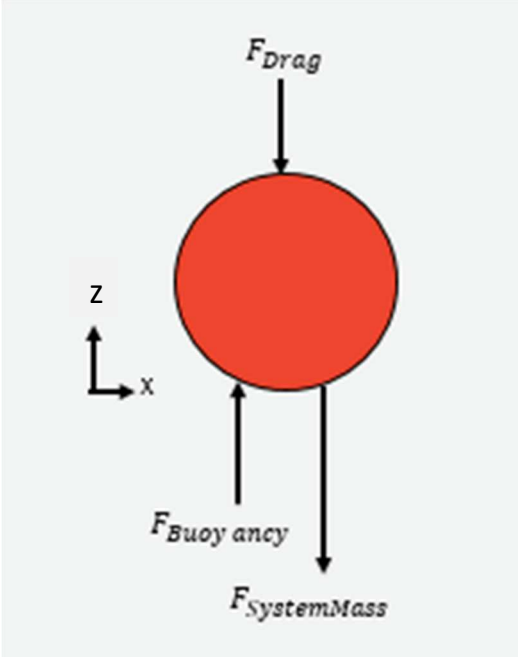



**Figure 3.1.** Force diagram of the balloon system.

The summation of forces is depicted as

$$0 = F_{Buoyancy} - F_{Drag} - F_{SystemMass} \tag{3.1}$$

Force due to buoyancy is

$$F_{Buoyancy} = Vg(\rho_{air}) \tag{3.2}$$

Where V is the volume of helium in the balloon at the time of launch, or ground conditions, g is the acceleration due to gravity, and $\rho_{air}$ is the density of air.

Force due to drag is

$$F_{Drag} = \frac{1}{2}\rho_{air}u^2 c_d A \tag{3.3}$$

Where $u$ is the rise velocity of the balloon and payload, $c_d$, is the drag coefficient, and A is the cross sectional area.

Force due to mass is,

$$F_{SystemMass} = (m_b + m_p + m_{He})g \tag{3.4}$$

Where $m_b$ is the mass of the balloon, $m_p$ is the mass of the payload, and $m_{He}$ is the mass of the helium.

Assuming helium behaves as an ideal gas, the volume of helium in the balloon can be expressed using the IGL.

To get everything in terms of mass of helium, the moles of helium (n) can be expressed as shown in Equation 3.5

$$n = \frac{m_{He}}{M_{He}} \tag{3.5}$$

Where $M_{He}$ is the helium molar mass.

Solving for volume in the IGL, and substituting Equation 3.5 yields Equation 3.6

$$V = \frac{m_{He}RT}{PM_{He}} \tag{3.6}$$

Where the mass of helium is defined as,

$$m_{He} = \rho_{He}V \tag{3.7}$$

Recall that in Equation 3.2, air density is needed. In many cases, air can be assumed to travel as a dry air parcel, however, for this system, relative humidity should be considered because the system of equations used to calculate mass is heavily dependent on a buoyancy calculation dictated by the difference in fluid densities.

To calculate density of moist air, and assuming it to be an ideal gas, Equation 3.8 can be used.

$$\rho_{air} = \frac{\left(\frac{P}{R_a T}\right)(1+x)}{(1+x\frac{R_w}{R_a})} \tag{3.8}$$

Where P is the pressure of the humid air, or ground pressure at launch, $R_a$ is the individual gas constant of air, $R_w$ is the individual gas constant of water vapor, and x is the humidity ratio.

The humidity ratio can be expressed in terms of a mass basis as follows:

$$x = \frac{m_w}{m_a} \tag{3.9}$$

Where $m_w$ is the mass of water vapor in the moist air and $m_a$ is the mass of dry air in the moist air.

The remaining unknown is the cross-sectional area. Assuming the balloon to be a perfect sphere, the cross-sectional area can be expressed with Equation 3.10.

$$A = \pi r^2 \tag{3.10}$$

Where r is the radius of the helium filled balloon.

The radius of the balloon can be expressed in terms of volume. It is known that volume of a sphere is:

$$V = \frac{4}{3}\pi r^3 \tag{3.11}$$

Which can be rearranged to form an expression for radius as:

$$r = \left(\frac{3}{4}\frac{V}{\pi}\right)^{\frac{1}{3}} \tag{3.12}$$

Taking these equations into consideration, results in the following:

$$0 = Vg\left(\frac{\left(\frac{P}{R_a T}\right)\left(1+\frac{m_w}{m_a}\right)}{\left(1+\left(\frac{m_w}{m_a}\right)\frac{R_w}{R_a}\right)}\right) - \frac{1}{2}\left(\frac{\left(\frac{P}{R_a T}\right)\left(1+\frac{m_w}{m_a}\right)}{\left(1+\left(\frac{m_w}{m_a}\right)\frac{R_w}{R_a}\right)}\right)v^2 C_d A - \left(m_b + m_p + \rho_{He} V\right)g \tag{3.13}$$

The unknown variables are volume, $V$, drag coefficient, $C_d$, and area, A. Through a trial-and-error method of filling balloons with helium and launching them to record their rise rate, a drag coefficient of 0.285 was determined to be the most optimal. The data supporting this value is provided in Table 3-1.

**Table 3-1**. Experimentally measured mean rise rates with varying drag coefficients.

| Launch | Drag Coefficient | Helium-fill Volume STP (SL) | Mean Rise Rate (m/s) | \|Error\| (%) |
|---|---|---|---|---|
| 1 | 0.23 | 1310.56 | 5.58 | 10.39 |
| 2 | 0.23 | 1309.89 | 5.29 | 5.48 |
| 3 | 0.23 | 1306.35 | 4.37 | 14.40 |
| 4 | 0.23 | 1302.87 | 4.21 | 15.8 |
| 5 | 0.3 | 1301.70 | 5.995 | 16.60 |
| 6 | 0.3 | 1590.14 | 5.38 | 7.06 |
| 7 | 0.285 | 1582.80 | 5.31 | 5.85 |
| 8 | 0.285 | 1488.61 | 4.98 | 0.4 |
| 9 | 0.285 | 1454.47 | 5.28 | 5.37 |

A Python helium fill code was developed to accurately calculate the helium needed for a particular temperature, pressure, relative humidity, payload mass, and balloon mass dependent on the day of launch. This code can be found in Appendix D.

## Wavelet Analysis

Wavelet analysis was performed on the radiosonde profile data collected during training exercises. However, before any analysis is done on the profile data, the files are "cleaned" to remove any blank rows of data where the radiosonde lost signal with the groundstation. If more than 10 minutes of consecutive data is missing, the profile is considered bad and analysis is not performed. If less than 10 minutes of consecutive data is missing, the rows with missing columns are manually deleted from the profiles. There is also the possibility that some flights require being rerun in GRAWMET, usually due to improper saving or bugs in the software, which impact the exported data. An SOP on how to rerun data in GRAWMET can be found in Appendix E. Other things to look for in data cleaning is ensuring start times and altitudes/coordinates are entered and saved properly. It should be noted that the MATLAB wavelet analysis code is designed to read radiosonde profiles of a specific form. Specifically, each profile needs to have a heading.

Radiosonde measurements are not always uniform, especially with varying rise rates and interrupted radio signals. In order to perform a wavelet analysis the profile data is interpolated so that the data is spatially uniform. Interpolation of the profile data should yield reasonable estimates for missing data points based on the known profile data. This step is included in the analysis code and the MATLAB can be seen in Appendix A under the "preprocess data" script.

With clean and spatially uniform data the Wavelet Transform is performed using the MATLAB code as detailed in Chapter 2. The wavelet transform uses the Morlet wavelet by default ($k_0$=6) and computes the wavelet transform. The outputs for this code will give the wavelet transform as well as period, scales, and the cone of influence. Most importantly, the wavelet coefficients are determined. These coefficients are the zonal and meridional wind components, denoted as U and V, respectively.

Once the wavelet transform is complete, a power surface is created by $S = |U|^2 + |V|^2$. The surface is scanned for local maxima on the wavelet surface that are inside the cone of influence. An example from one of the training flights is shown in Figure 3.2. After identifying local maxima, the extension of the corresponding wave packet in scale and height is recorded, denoted as $s_1$, $s_2$, $z_1$, and $z_2$, respectively. This is done by tracing a rectangle around each of the detected local maxima on

the constructed power surface by iterating in the four directions until either 25% of peak power is reached, or until the power surface begins increasing.

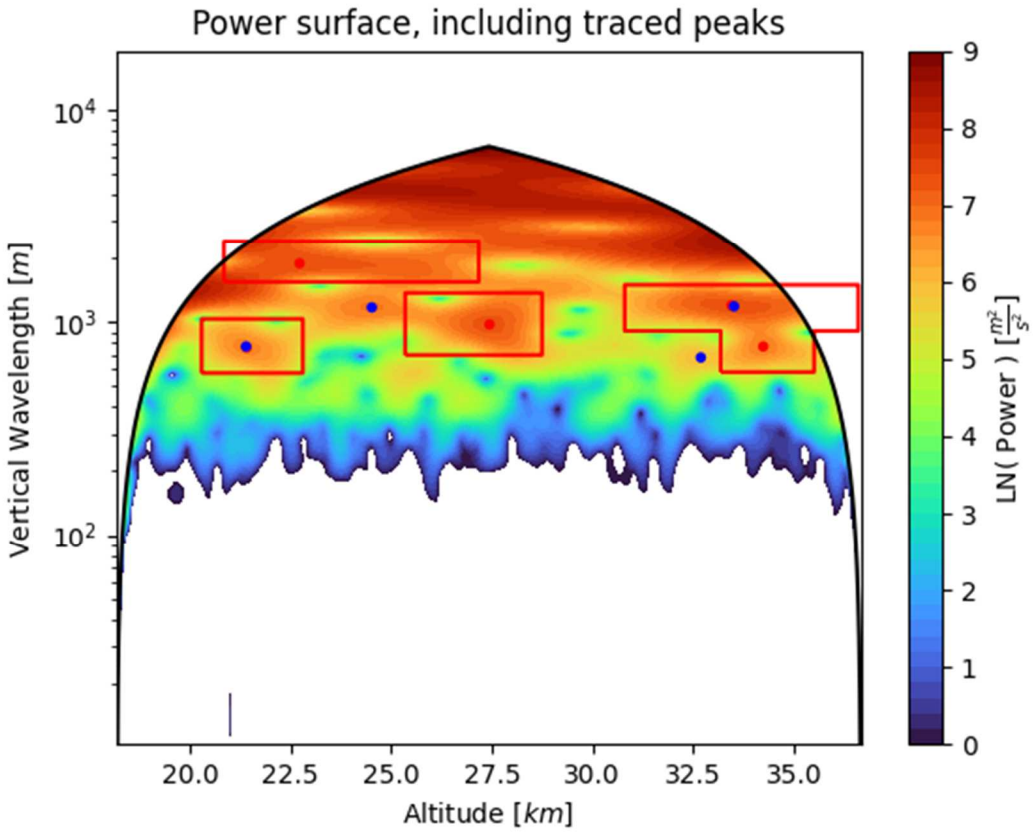



**Figure 3.2.** Power surface from one of the training profiles. The dome encompassing everything signifies the cone of influence. The rectangles trace local maxima found. Blue dots are waves detected that did not meet the criteria of AGWs and red dots meet the criteria. Power surface is used to compare different wavelet power spectra to a white-noise power spectrum.

To get the time series for U, V, and T the wavelet surface that is inside of the rectangle traced, or local maxima detected, is inverted. A sum of the columns of each is done, and the sum is multiplied by a reconstruction constant. The reconstruction constants are defined by Torrence and Compo for each of the wavelet types used [24]. For the default Morlet wavelet, the reconstruction factor is 0.776. Finally, using the time series just found for U, V, and T, values below 50% of the max power surface are filtered out, and Stokes parameters are found as described in Chapter 2. Other wave parameters as described by Murphy *et al*. [27] can also be calculated. Table 3-2 summarizes the intrinsic parameters and Table 3-3 lists the ground-based parameters.

**Table 3-2**. Intrinsic parameters [27]

| Parameter | Description | Equation |
|---|---|---|
| $k_h$ | Horizontal wave number | $k_h^2 = \frac{f^2 m^2}{N^2}\left(\frac{\hat{\omega}}{f^2} - 1\right)$ |
| $\lambda_h$ | Horizontal wavelength | $\lambda_h = \frac{2\pi}{k_h}$ |
| $k$ | Zonal wave number | $k = k_h sin\phi$ |
| $l$ | Meridional wave number | $l = k_h cos\phi$ |
| $\hat{c}_z$ | Intrinsic vertical phase speed | $\hat{c}_z = \frac{\hat{\omega}}{m}$ |
| $\hat{c}$ | Intrinsic horizontal phase speed | $\hat{c} = \frac{\hat{\omega}}{k_h}$ |
| $\hat{c}_x$ | Intrinsic zonal phase speed | $\hat{c}_x = \frac{\hat{\omega}}{k}$ |
| $\hat{c}_y$ | Intrinsic meridional phase speed | $\hat{c}_y = \frac{\hat{\omega}}{l}$ |
| $\hat{c}_{gz}$ | Intrinsic vertical group velocity | $\hat{c}_{gz} = -\frac{1}{\hat{\omega}m}(\hat{\omega}^2 - f^2)$ |
| $\hat{c}_{gx}$ | Intrinsic zonal group velocity | $\hat{c}_{gx} = \frac{kN^2}{\hat{\omega}m^2}$ |
| $\hat{c}_{gy}$ | Intrinsic meridional group velocity | $\hat{c}_{gy} = \frac{lN^2}{\hat{\omega}m^2}$ |
| $\hat{c}_{gh}$ | Intrinsic horizontal group velocity | $\hat{c}_{gh} = \sqrt{\hat{c}_{gx}^2 + \hat{c}_{gy}^2}$ |

**Table 3-3.** Ground-based parameters

| Parameter | Description | Equation |
|---|---|---|
| $c_{gx}$ | Zonal group velocity | $c_{gx} = \hat{c}_{gx} + \bar{u}$ |
| $c_{gy}$ | Meridional group velocity | $c_{gx} = \hat{c}_{gx} + \bar{v}$ |
| $c_{gz}$ | Vertical group velocity | $c_{gz} = \hat{c}_{gz}$ |
| $\omega$ | Ground-based frequency | $\omega = \hat{\omega} + \lvert k_h \rvert \lvert \bar{u}_h \rvert \cos(\phi_w - \phi)$ |
| $c_x$ | zonal phase velocity | $c_x = \hat{c}_x + \bar{u} + \frac{\bar{v}l}{k}$ |
| $c_y$ | Meridional phase velocity | $c_y = \hat{c}_y + \bar{v} + \frac{\bar{u}k}{l}$ |
| $c$ | Phase velocity | $c_x = \frac{\hat{\omega}}{k_h} + \bar{u}_h(cos\phi_w - \phi)$ |
| $\overline{u'w'}$ | Zonal momentum flux | $\overline{u'w'} = -\frac{\rho\hat{\omega}}{N^2} g u' \left(\frac{\overline{T'} + 90}{\bar{T}}\right)$ |
| $\overline{v'w'}$ | Meridional momentum flux | $\overline{v'w'} = -\frac{\rho\hat{\omega}}{N^2} g v' \left(\frac{\overline{T'} + 90}{\bar{T}}\right)$ |
| $KE$ | Kinetic energy | $KE = \frac{1}{2}\left[\overline{u'^2} + \overline{v'^2}\right]$ |
| $PE$ | Potential energy | $PE = \frac{g^2}{2N^2}\frac{\overline{T'^2}}{\bar{T}}$ |

Where $\bar{u}$ and $\bar{v}$ are the background zonal and background meridional wind components, respectively. Horizontal wind direction, degrees clockwise from north, is defined as $\phi_w$.

# Chapter 4: Total Solar Eclipse 2020

## Introduction

On August 21, 2017, data were collected during a total solar eclipse over the United States. The campaign, conducted by the University of Montana (UM), took place in Fort Laramie, WY (ARTSE2017). Three UM radiosonde launch sites were set-up through-out the path of totality, with no more than 55 km in-between each other. In total, 19 radiosondes were launched over the course of 48 hours, and the wavelet method was used to analyze the data. Wave-like structures with intrinsic angular frequencies in the range of $3.3 – 4.2x10^{-2}$ $s^{-1}$ within altitudes of 18-20 km were detected [28]. Further, a dominant gravity-wave structure around 19 km with an intrinsic frequency of approximately $4.2x10^{-2}$ $s^{-1}$ and a vertical wavelength of about 20 km was measured. The detected wave is believed to be solar eclipse induced; however definitive results were inconclusive. The campaign also targeted lower flight altitudes, missing key data for stratospheric AGW detection.

On July 2, 2019, the prior methods were modified to target higher temporal frequency and higher altitudes [12]. The eclipse being monitored occurred in South America over Chile and Argentina. Eclipse measurements were conducted outside of Andacollo, Chile at the Collowara Tourist Observatory. Radiosondes were launched every hour, from 23 hours before totality to two hours after, for a total of 25 radiosondes launched over the course of 27 hours. Three solar eclipse induced AGWs were detected and their parameters were compared using the wavelet method and hodograph method. The wavelet method revealed one high frequency wave at 26 km four hours after totality with an intrinsic frequency of 35, an intrinsic period of 40 min, and a propagation direction of 16°. The hodograph method revealed two solar eclipse induced AGWs 40 minutes after totality. The first AGW was detected at 23 km, with an intrinsic frequency scaled by the Coriolis parameter of 3.5, a period of 6.7 hours, and a propagation direction of 45°. The second wave was detected at 25 km, with an intrinsic frequency scaled by the Coriolis parameter of 5.9, a period of 4 hours, and a propagation direction of 49°. The findings during the 2019 campaign were the first unambiguous detection of eclipse induced GWs in the middle atmosphere.

On December 14, 2020, another total solar eclipse in South America occurred with the path of totality going right through Chile, as shown in Figure 4.1. This presented a unique opportunity to replicate methods in a similar location to the 2019 eclipse campaign. Two designated launch sites were set-up, one in Tolten and the other in Villarrica. Over the course of 48 hours, 50 radiosondes in Tolten and 50 in Villarrica were launched every hour. Launches started 24 hours before totality and continued for 24 hours after totality. The campaign goal was to detect eclipse induced AGWs in the stratosphere using the methods described above.

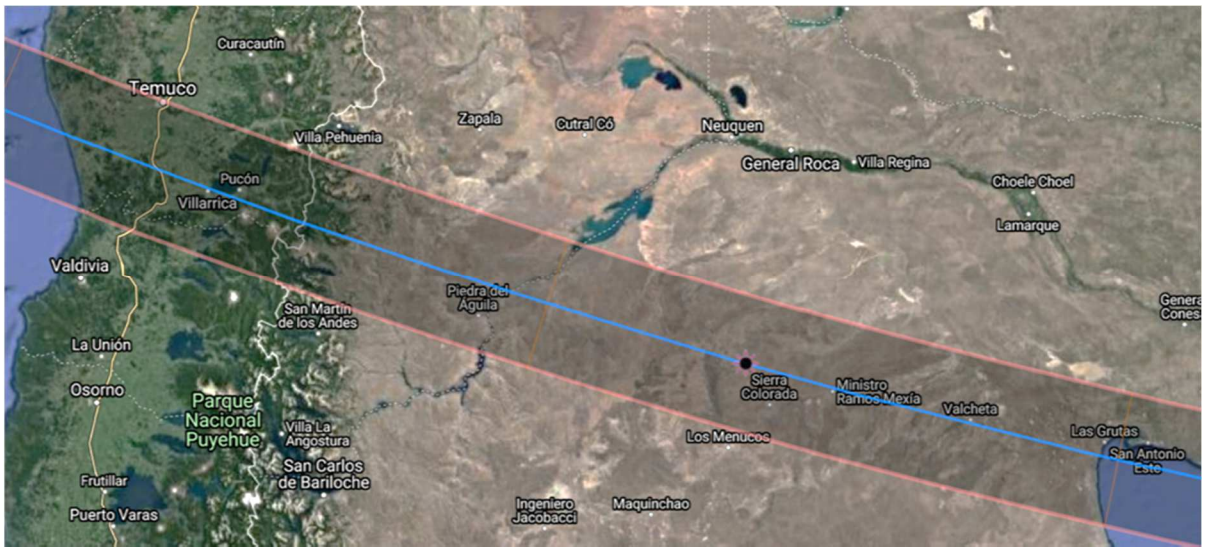



**Figure 4.1**. Total solar eclipse path of totality.

Efforts to collect atmospheric data was conducted by Montana Space Grant Consortium (MTSGC), Idaho Space Grant Consortium (ISGC), and Oklahoma Space Grant Consortium (OKSGC).

## Methods and Materials

### *Frequency allocation*

Radiosondes were launched simultaneously in different locations, thus, to avoid frequency interference between radiosondes in the atmosphere, a frequency allocation schedule was followed. Appendix F summarizes the Villarrica and Tolten frequency allocations and launch times. Frequencies were adjusted through GRAWMET when initializing each radiosonde. Each flight was identified as the first letter of the launch location and the flight number *i.e.,* the first flight in Villarrica is denoted as “V1”.

### *Equipment*

Each team transported their Lufft, ground stations, computers, 600-gram balloons, radiosondes, parachutes and de-reelers, as well as a designated launch box equipped with things like: tape, scissors, scale, gloves, tarp, etc. Helium was purchased in Chile and distributed between the two

launch sites. SOPs developed over the Summer, and described above, were followed for helium fill, initialization of radiosondes and running the GRAWMET software. Each profile was saved following the naming convention of: First letter of launch location_UTC_Date_Computer name. This was done in an effort to avoid any confusion between profiles being saved and distributed to other team members.

*Wavelet-analysis code modifications*

A major part and most of the work focused on in this thesis is the reanalysis of methods used for previous campaigns and their application to data collected in Chile in 2020. Originally, for the 2019 eclipse, the background was removed using a rolling mean. After further analysis, it was decided that the removal of the background was not needed, as the wavelet transform removes any background noise. The Morlet wavelet is an amplitude modulated sine wave that closely resembles AGWs. Signals in the data that closely resemble a Morlet wavelet will project onto it and through convolutions, any background noise is filtered out.

Additionally, analysis was originally done using a five- meter height sampling frequency (hsf), which sets the grid spacing to which all of the data is interpolated to. It is believed that when setting the hsf at a low value, such as a five-meter hsf, turbulence effects can be detected. To rule this out, data analysis was done using a ten-meter hsf.

Furthermore, additional checks were implemented. The original checks are followed:

Check 1: Intrinsic frequencies are between the Coriolis frequency and Brunt- Väisälä frequency. Anything outside of this range is non-physical.

Check 2: Stokes parameters, Q and P, must be greater than 0.05. Anything below indicates weak wave activity and should be discarded.

Check 3: Waves with a degree of polarization, d, must be greater than 0.5 and less than one. A degree of polarization less than 0.5 indicate weak polarization. A degree of polarization greater than one indicate a deviation of the data from the assumptions made to derive the Stokes parameters.

Check 4: Local maximum candidate from the power surface should be inside the COI. If it is not, the wave is discarded.

Check 5: Waves must meet a threshold corresponding to a perturbation amplitude of 0.1 m $s^{-1}$ to be detected as wave packets. Anything lower than this threshold is deemed as noise.

The additional checks implemented in this work are:

Check 6: The vertical wavelength should be less than half of the total vertical distance.

Check 7: The horizontal wavelength should be greater than the balloon drift distance.

Checks 6 and 7 were suggested by AGW expert, Dr. Jie Gong, as waves should follow these physical characteristics. It should be noted that all of these modifications are included in the MATLAB code in Appendix A.

*AGW analysis*

The wavelet method was used to extract wave parameters as described in Chapter 2, and following the code in Appendix A. Additionally, to isolate eclipse-induced waves, other potential sources were analyzed. Mountain waves were characterized as propagating in the opposite direction of the prevailing wind in the intrinsic frame of reference, as well as have ground-based horizontal phase speeds near zero [12]. The other major analysis done was to identify significant wind shears. These were distinguished as producing AGWs of the same average wind speed as the shear layer and propagation direction equal to the direction of the change in wind speed [29]. The Richardson-Index (RI) method looks for an RI number between 0 and 0.25. It is believed that between these two values, there is a fluid instability that contributes to the formation of AGWs [29]. Additionally, convective instabilities can be detected through a similar method. It is seen that when the RI number falls below zero, there are convective instabilities [30]. A code in Python, by UM student Hannah Woody, for calculating significant wind shear and convective instabilities was developed based on the RI, and is provided in Appendix G.

## 2020 data results

The 2020 solar eclipse was the first and only total solar eclipse in 2020. With one-hundred radiosondes launched during the campaign, one would expect to gather ample amount of data correlating AGWs with the solar eclipse, as this is a unique situation where one criterion can be completely isolated based on solar irradiance. However, during the entirety of the eclipse, another natural phenomenon, called an atmospheric river, occurred, shown in Figure 4.2. An atmospheric river transports water vapor and forms long, narrow regions in the atmosphere. When it makes landfall, it releases the water vapor in the form of rain. This phenomenon has made data analysis extremely difficult.

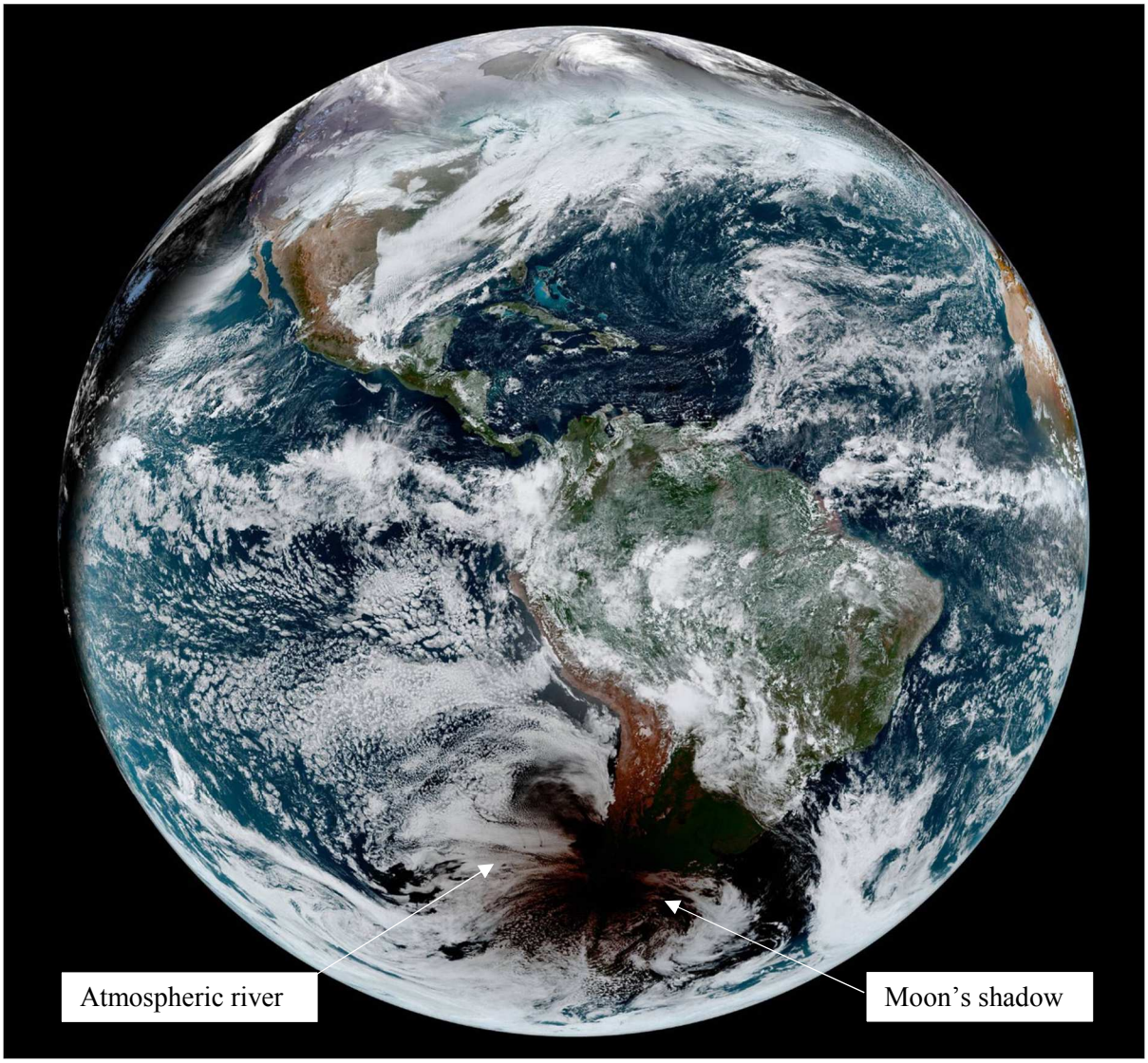



**Figure 4.2**. Image taken from Advanced Basline Imager (ABI) on Geostationary Operational Environmental Satellite 16 (GOES-16) operated by the National Oceanic and Atmospheric Administration (NOAA).

Due to such a strong storm, most of the data collected in Villarrica has large gaps of data missing, especially in the region of the stratosphere. Luckily Tolten was not impacted as much, with only a couple unrecoverable flight profiles. For this reason, data analysis has been solely focused using Tolten data sets. Through the wavelet analysis developed, many AGWs have been detected. The mean horizontal phase speed of all waves detected is 5.81 m/s with a standard deviation of 3.24 m/s (n=48). A mean period of approximately 13.43 hours corresponding to a mean intrinsic frequency over the Coriolis frequency, $\frac{\hat{\omega}}{f}$, of 1.94 (standard deviation of 0.67) was detected.

Totality occurred right after 17:00 UTC, thus special interest around the time of totality and the partial eclipse was taken into consideration for data analysis. To try and identify the potential sources of AGWs detected, certain characteristics known to be attributed to AGWs due to convective instabilities are evaluated. These characteristics are listed in Table 4-1.

**Table 4-1.** Characteristics of solar eclipse induced AGWs.

| Characteristic | Reference |
|---|---|
| High intrinsic frequency | [31] |
| Isotropic horizontal propagation direction | [31] |
| Horizontal wavelengths on the order of tens to hundreds of km | [32] |
| Spatially localized | [32] |
| Richardson-index number below zero | [30] |
| Higher horizontal phase speed | [30] |

Profiles centered around totality, flights 20-30, resulted in the following results:

**Table 4-2**. Results for profiles centered around totality.

| Flight | Alt of detection (km) | Horizontal wavelength (km) | $\hat{\omega}/f$ | Horizontal phase speed (m/s) | Propagation direction (deg N from E) |
|---|---|---|---|---|---|
| T20 | No gravity waves detected | | | | |
| T21 | 18.033 | 116.236 | 2.17 | 3.67 | -47.902 |
| | 19.618 | 117.425 | 1.67 | 2.85 | 11.522 |
| | 20.093 | 226.131 | 2.01 | 6.63 | 19.528 |
| | 22.338 | 132.714 | 1.72 | 3.33 | 152.981 |
| | 23.203 | 320.133 | 1.35 | 6.29 | 45.575 |
| | 26.898 | 100.65 | 2.70 | 3.96 | 168.460 |
| | 28.498 | 365.96 | 1.65 | 8.84 | 258.927 |
| T22 | 14.641 | 260.861 | 1.28 | 4.90 | 41.690 |
| | 17.696 | 193.04 | 2.45 | 6.92 | -13.565 |
| | 17.826 | 209.39 | 1.53 | 4.70 | 61.627 |
| | 19.371 | 91.845 | 2.00 | 2.68 | 210.482 |
| | 23.891 | 253.172 | 1.42 | 5.25 | -82.915 |

| | | | | | |
|---|---|---|---|---|---|
| | 27.036 | 84.792 | 5.55 | 6.85 | 254.029 |
| | 27.596 | 69.931 | 3.26 | 3.32 | -20.857 |
| T23 | 18.551 | 910.484 | 1.21 | 16.15 | -50.642 |
| | 18.716 | 107.459 | 3.19 | 5.00 | -1.757 |
| | 28.136 | 88.684 | 2.29 | 2.95 | -80.735 |
| T24 | 16.589 | 285.198 | 1.78 | 7.43 | 30.136 |
| | 19.069 | 78.907 | 2.15 | 2.47 | 235.182 |
| T25 | 18.478 | 512.583 | 1.55 | 11.62 | 31.643 |
| | 18.508 | 185.44 | 1.86 | 5.04 | 24.579 |
| | 25.768 | 68.452 | 3.62 | 3.61 | 113.99 |
| T26 | 19.409 | 387.511 | 1.63 | 9.21 | 264.353 |
| | 26.144 | 117.469 | 2.23 | 3.81 | 157.025 |
| T27 | 17.873 | 308.407 | 1.11 | 5.00 | 25.468 |
| T28 | 17.762 | 145.788 | 1.53 | 3.26 | -68.821 |
| | 18.402 | 101.368 | 2.58 | 3.81 | -39.911 |
| | 19.647 | 244.991 | 2.29 | 8.17 | -44.307 |
| | 21.327 | 126.121 | 1.73 | 3.17 | 46.429 |
| T29 | 19.27 | 79.953 | 3.9 | 4.53 | 30.664 |
| | 22.79 | 167.281 | 1.44 | 3.51 | 53.458 |
| | 25.085 | 224.133 | 3.17 | 10.35 | 232.284 |
| | 26.66 | 134.094 | 1.93 | 3.77 | -75.570 |
| T30 | 16.504 | 97.972 | 2.72 | 3.88 | -27.080 |
| | 19.024 | 718.621 | 1.23 | 12.86 | 240.946 |
| | 20.634 | 104.588 | 3.3 | 5.01 | 26.367 |
| | 25.589 | 414.338 | 2.4 | 14.46 | -55.331 |
| | 26.719 | 171.853 | 1.72 | 4.31 | -76.019 |
| | 31.739 | 74.551 | 4.04 | 4.36 | -53.832 |

To visualize the propagation direction and intrinsic frequency, Figure 4.3 plots the results outlined in Table 4-2. Each red arrow represents a gravity wave detected and it points in the direction of

propagation. The arrows are scaled by the gravity wave's intrinsic frequency, thus longer arrows are indicative of higher intrinsic frequencies.

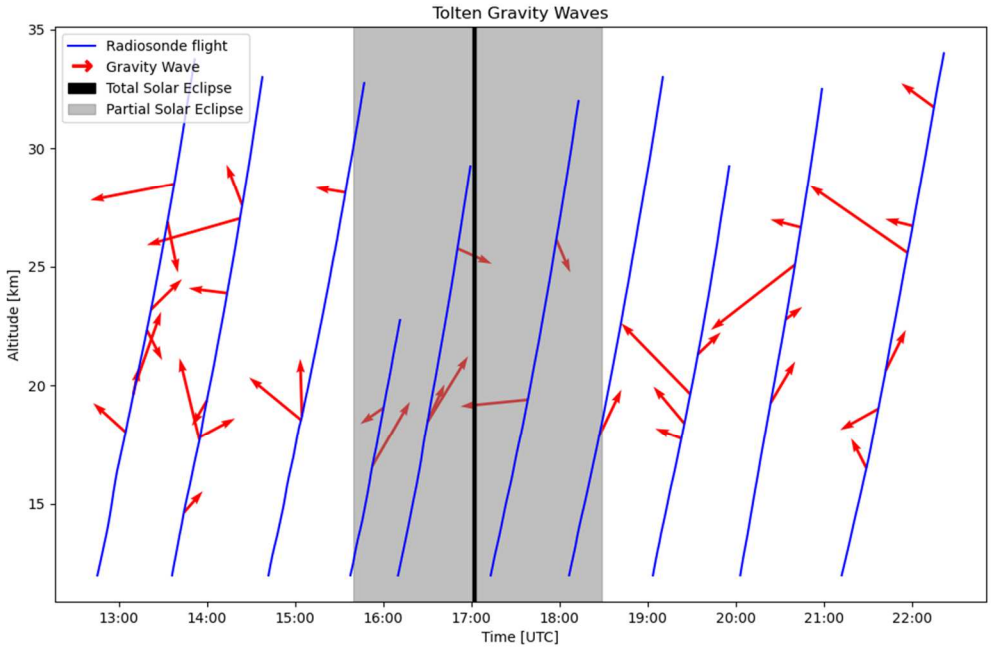



**Figure 4.3**. Tolten AGWs detected around totality. Each slanted line indicates a radiosonde flight, and each arrow is an AGW detected. Arrows are scaled by the AGW's intrinsic frequency, thus longer arrows have higher intrinsic frequencies.

The results indicate three potential AGWs induced by the solar eclipse, which are outlined in Table 4-3. The waves being considered occurred within a few hours of totality. These waves stand out due to their high intrinsic frequency and high horizontal phase speed, which are known characteristics of solar eclipse induced AGWs, as outlined in Table 4-1.

**Table 4-3.** Potential solar eclipse induced waves.

| Flight | Alt of detection (km) | Horizontal wavelength (km) | $\hat{\omega}/f$ | Horizontal phase speed (m/s) | Propagation direction (deg N from E) |
|---|---|---|---|---|---|
| T28 | 19.647 | 244.991 | 2.29 | 8.17 | -44.307 |
| T29 | 25.085 | 224.133 | 3.17 | 10.35 | 232.284 |
| T30 | 25.589 | 414.338 | 2.4 | 14.46 | -55.331 |

To further evaluate them, plots for T28 - T30 of the Richardson-Index number were examined and can be seen in Figure 4.4.

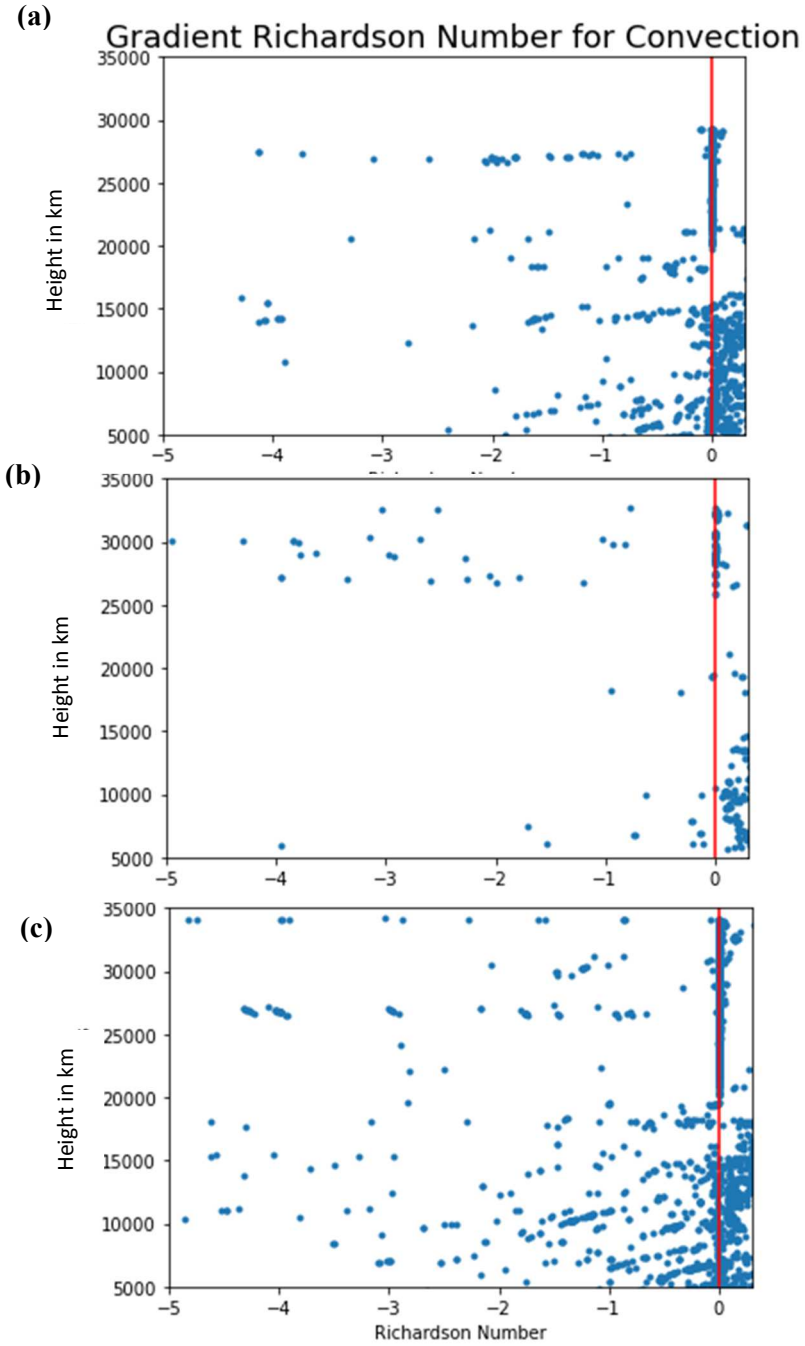



**Figure 4.4**. Richardson-Index number below zero indicates regions of convective instability. The profiles are denoted by **(a)** T28, **(b)** T29, and **(c)** T30.

The RI method looks at the gradient of the virtual potential temperature, and when convective instability occurs, the gradient falls below zero. Convective instability is obvious in T28 and T30, but less so in T29. As seen in Figure 4.4 (a) and (c) there are dense regions of the RI number that fall below zero. In Figure 4.4 (b) it is much less apparent, however there is a small region around 25 km – 30 km that falls below zero. With this in mind, other sources need to be ruled out.

Propagation direction may not matter in solar eclipse induced gravity waves, but it is useful in identifying other sources, such as mountain induced AGWs and wind shear induced AGWs. These type of AGWs will also have very distinct characteristics, which can be seen in Table 4-4.

**Table 4-4.** Characteristics of mountain and wind shear induced AGWs.

| Source | Characteristic | Reference |
|---|---|---|
| Mountain | Ground-based phase speed near zero | [31-33] |
| | Greater intermittency | [33] |
| | Horizontal propagation directions opposite to the flow of wind over topography | [30] |
| | Horizontal wavelengths on order of hundreds to thousands of km | [31] |
| | Dominate horizontal wavelengths at 400 km and 110 km in the Andes | [34] |
| | | |
| Windshear | Lower frequency | [31] |
| | Shear greater than 7.5 m/s every 1000 m | [35] |
| | Propagating away from shear direction | [36] |
| | Richardson-Index number between 0 and 0.25 | [29] |
| | Phase speed matching wind speed | [31] |

Flight T20 occurred about five hours before the eclipse, and no gravity waves were detected in this profile. The flights following were T21 and T22, which were completed before the partial eclipse began. These profiles and other profiles before T20 are great sources to distinguish mountain and windshear induced AGWs from solar eclipse induced AGWs. Within T21 and T22, fourteen AGWs were detected between both flights. Intrinsic frequencies and phase speeds are not as high compared to waves detected in T28 - T30. However, due to the atmospheric river, data analysis is still on-going to determine if the waves in T28 - T30 are solar eclipse induced. Mountain and windshear sources cannot be completely ruled out yet. Atmospheric river experts have been brought in to help with further data analysis.

## Conclusion

Since 1970, atmospheric gravity waves induced by solar eclipses have been of great interest to many atmospheric scientists. Data collection is tedious and can only be done during isolated windows, thus progress in the field is slow-moving. Until 2019, there were no conclusive results indicating atmospheric gravity waves are induced by solar eclipses. Since then, using a wavelet-based method, solar-eclipse induced waves are evident in the middle-atmosphere. In the data collected during the 2020 eclipse, promising candidates are being considered, and with further analysis there is potential to confirm solar eclipse induced AGWs from the 2020 field campaign.

# Chapter 5: Conclusion and Future Directions

## Conclusion

Training launches have led to an improved gravity wave analysis, which will be used for future campaigns in an effort to detect solar eclipse induced gravity waves. The 2020 campaign was a great opportunity to expand data sets during a total solar eclipse. Over 175 AGWs were detected based only on Tolten data. On-going analysis is being completed to identify the sources of these waves. Because many of the data sets in the 2020 campaign were overtaken by the atmospheric river, the data sets are also of great interest to many atmospheric scientists.

## Future Directions

UM detected three waves representing the first unambiguous detection of eclipse-induced gravity waves in the middle atmosphere during the 2019 solar eclipse. This drove the campaign for 2020, and with future solar eclipses happening over the next few years, it is of great interest to build on the data sets collected. The methods described in this thesis will work as training material for future campaign teams, however there is room for improvement on the methods presented.

With an increase in data sets, it would be valuable to compare results from other gravity wave analysis methods. The wavelet-method is promising; however, this method tends to favor higher frequency waves, compared to the hodograph method that favors lower frequency waves. Thus, developing an analysis method that does not favor certain waves over the other will allow for more accurate characterization of AGWs.

The main analysis is done in MATLAB. However, the limitation here is that MATLAB is not an open source, thus resulting in fewer people being familiar with the program. For educational purposes, the continuation of this campaign may thrive in an open environment like Python, as it is more user friendly. With this in mind, a Python script replicating the MATLAB script was created, but analysis comparing the two is on-going. Additionally, other team members are actively developing procedures for completing hodograph analyses of the collected data sets to complement the analysis in this thesis.

Finally, everything from data collection to analysis depends on sounding profiles. During the 2020 eclipse an unprecedented atmospheric river caused most of the profiles from the Villarrica launch site to come in with an astonishing amount of missing data. This posed a great challenge in the analysis as the team struggled to decide which profiles were usable and which needed to be thrown out. It would be beneficial to design a more reliable payload to avoid the loss of data during a storm or other natural phenomena.

# Chapter 6: References

[1] Fortune, P. (2014). Basic Meteorology: A Short Course.

[2] Lalic, B., Eitzinger, J., Marta, A., Orlandini, S., Firanj Sremac, A., & Pacher, B. (2018). Agricultural Meteorology and Climatology.

[3] Xiao, L., JiYao, X., & Jia, Y. (2020). Global static stability and its relation to gravity waves in the middle atmosphere. doi:10.26464/epp2020047

[4] Birner, T., Sankey, D., & Shepherd, T. G. (2006). The tropopause inversion layer in models and analyses. Geophysical Research Letters, 33(14). doi:https://doi.org/10.1029/2006GL026549

[5] Nappo, C. J. (2002). An Introduction to Atmospheric Gravity Waves (Vol. 85).

[6] Holton, J. R. (1979). An Introduction to Dynamic Meteorology.

[7] Peppier, R. A. (1988). A Review of static Stability Indices and Related Thermodynamic Parameters.

[8] Jacques Descloitres, M. R. R. T., NASA/GSFC. (2003). Retrieved from https://visibleearth.nasa.gov/images/69463/atmospheric-gravity-waves-and-internal-waves-off-australia

[9] Mann, A. (2019). Core Concept: To improve weather and climate models, researchers are chasing atmospheric gravity waves. Proceedings of the National Academy of Sciences, 116(39), 19218-19221. doi:10.1073/pnas.1912426116

[10] Murakami, H., Wang, Y., Yoshimura, H., Mizuta, R., Sugi, M., Shindo, E., . Kitoh, A. (2012). Future Changes in Tropical Cyclone Activity Projected by the New High-Resolution MRI-AGCM*. Journal of Climate, 25, 3237-3260. doi:10.1175/JCLI-D-11-00415.1

[11] T. Oakley, H. V., Li Wei. (2011). WMO Intercomparison of High Quality Radiosonde Systems (107).

[12] Colligan, T., Fowler, J., Godfrey, J., & Spangrude, C. (2020). Detection of stratospheric gravity waves induced by the total solar eclipse of July 2, 2019. Scientific reports, 10(1), 19428-19428. doi:10.1038/s41598-020-75098-2

[13] Chimonas, G., and Hines, C. O. (1970), Atmospheric gravity waves induced by a solar eclipse, *J. Geophys. Res.*, 75( 4), 875– 875, doi:10.1029/JA075i004p00875.

[14] Anderson, R. C., Keefer, D. R., & Myers, O. E. (1972). Atmospheric Pressure and Temperature Changes During the 7 March 1970 Solar Eclipse. Journal of Atmospheric Sciences, 29, 583. doi:10.1175/1520-0469(1972)029<0583:Apatcd>2.0.Co;2

[15] Eaton, F. D., Hines, J. R., Hatch, W. H., Cionco, R. M., Byers, J., Garvey, D., & Miller, D. R. (1997). SOLAR ECLIPSE EFFECTS OBSERVED IN THE PLANETARY BOUNDARY LAYER OVER A DESERT. Boundary-Layer Meteorology, 83(2), 331-346. doi:10.1023/A:1000219210055

[16] Goodwin, G. L., & Hobson, G. J. (1978). Atmospheric gravity waves generated during a solar eclipse. Nature, 275, 109. doi:10.1038/275109a0

[17] Marlton, G. J., Williams, P. D., & Nicoll, K. A. (2016). On the detection and attribution of gravity waves generated by the 20 March 2015 solar eclipse. Philos Trans A Math Phys Eng Sci, 374(2077). doi:10.1098/rsta.2015.0222

[18] Equations of Motion. (2017). In G. K. Vallis (Ed.), Atmospheric and Oceanic Fluid Dynamics: Fundamentals and Large-Scale Circulation (2 ed., pp. 3-54). Cambridge: Cambridge University Press.

[19] Nappo, C. (2010). An Introduction to Atmospheric Gravity Waves (Vol. 102).

[20] Hines, C. O. (1974). Internal Atmospheric Gravity Waves at Ionospheric Heights. In The Upper Atmosphere in Motion (pp. 248-328).

[21] Zink, F., & Vincent, R. A. (2001). Wavelet analysis of stratospheric gravity wave packets over Macquarie Island: 1. Wave parameters. Journal of Geophysical Research: Atmospheres, 106(D10), 10275-10288. doi:https://doi.org/10.1029/2000JD900847

[22] Gonella, J. (1972). A rotary-component method for analysing meteorological and oceanographic vector time series. Deep Sea Research and Oceanographic Abstracts, 19(12), 833-846. doi:https://doi.org/10.1016/0011-7471(72)90002-2

[23] Eckermann, S. D., & Vincent, R. A. (1989). Falling sphere observations of anisotropic gravity wave motions in the upper stratosphere over Australia. Pure and Applied Geophysics, 130, 509. doi:10.1007/bf00874472

[24] Torrence C, Compo G. A practical guide to wavelet analysis. *Bull. Am. Meteorol. Soc.* 1998;**79**(1):61–78. doi: 10.1175/1520-0477(1998)079<0061:APGTWA>2.0.CO;2

[25] Daubechies, I. (1990). The wavelet transform, time-frequency localization and signal analysis. IEEE Transactions on Information Theory, 36, 961. Retrieved from https://ui.adsabs.harvard.edu/abs/1990ITIT...36..961D

[26] Vincent R, Fritts D. A climatology of gravity wave motions in the mesopause region at Adelaide Australia. *J. Atmos. Sci.* 1987;**44**(4):748–760. doi: 10.1175/1520-0469(1987)044<0748:ACOGWM>2.0.CO;2

[27] Murphy, D. J., Alexander, S. P., Klekociuk, A. R., Love, P. T., and Vincent, R. A. (2014), Radiosonde observations of gravity waves in the lower stratosphere over Davis, Antarctica, J. Geophys. Res. Atmos., 119, 11,973– 11,996, doi:10.1002/2014JD022448.

[28] Fowler, J., & Stocker, K. (2017). Internal-Gravity Waves Observed During the August 21, 2017 Total Solar Eclipse by National Eclipse Radiosonde Campaign.

[29] Fritts, D. (1978). The Excitation of Radiating Waves and Kelvin-Helmholtz Instabilities by the Gravity Wave-Critical Level Interaction. Journal of Atmospheric Sciences, 36, 12-23. doi:10.1175/1520-0469(1979)036<0012:TEORWA>2.0.CO;2

[30] Tsuda, T. (2014). Characteristics of atmospheric gravity waves observed using the MU (Middle and Upper atmosphere) radar and GPS (Global Positioning System) radio occultation. Proceedings of the Japan Academy. Series B, Physical and biological sciences, 90(1), 12-27. doi:10.2183/pjab.90.12

[31] Fritts, D. C., & Alexander, M. J. (2003). Gravity wave dynamics and effects in the middle atmosphere. Reviews of Geophysics, 41(1). doi:https://doi.org/10.1029/2001RG000106

[32] Cao, B. (2017). Dynamical Process of Gravity Waves Propagation and Dissipation, and Statistical Characteristics of Their Momentum Flux in the Mesosphere and Lower Thermosphere. (Doctor of Philosphy). Embry-Riddle Aeronautical University

[33] Article, S., Kim, Y. J., Eckermann, S. D., & Chun, H. Y. (2003). An overview of the past, present and future of gravity□wave drag parametrization for numerical climate and weather prediction models. Atmosphere-Ocean, 41(1), 65-98. doi:10.3137/ao.410105

[34] Torre, A., & Alexander, P. (2005). Gravity waves above Andes detected from GPS radio occultation temperature profiles: Mountain forcing? Geophysical Research Letters - GEOPHYS RES LETT, 321. doi:10.1029/2005GL022959

[35] Thatcher, L., & Pu, Z. (2011). How Vertical Wind Shear Affects Tropical Cyclone Intensity Change: An Overview.

[36] Plougonven, R., & Zhang, F. (2014). Internal gravity waves from atmospheric jets and fronts. Reviews of Geophysics, 52(1), 33-76. doi:https://doi.org/10.1002/2012RG0004

# Appendix A - Wavelet-Analysis MATLAB Code

### Main script: runFile

```
warning('off', 'MATLAB:polyfit:RepeatedPointsOrRescale');
addpath('wave_matlab/'); % for wavelet analysis
addpath('dnsfnu/'); % for plot coordinates
addpath('acf/'); % for autocorrelation function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                User defined variables                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dataDirectory = 'C:\Users\Temp\Documents\MATLAB\GW Code December
2020\ToltenCleaned2\T26';
saveDirectory = 'C:\Users\Temp\Documents\MATLAB\GW Code December
2020\Stokes_parameters\ExtraChecksMarked';
% The w/f happened with no correction to wind direction, a window of
% 12km - 40km, and a heightSamplingFrequency of 5m.
% heightSamplingFrequency of 7m makes a 90 w/f show up.
showPowerSurfaces = true; % Do you want to show the wavelet transform
power surfaces?
save = true; % Do you want to save the data? It will save in
saveDirectory.
lowerCutOffAltitude = 12000; % Altitude where you want to start analysis
upperCutOffAltitude = 40000; % Altitude where you want to end analysis -
% a value of 40000 will go to the highest point in the profile.
latitude = 30.250; % Latitude of launch location, using absolute value for
Coriolis parameter. 30.25 for test data
heightSamplingFrequency = 5;
printData = true;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                End of user editing                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
textFiles = fullfile(dataDirectory, "*.txt");
files = dir(textFiles);
f1 = figure;
f2 = figure;
set(0, 'CurrentFigure', f1);
hold on;
set(0, 'CurrentFigure', f2);
hold on;
counter = 0;
first = true;
minLat = Inf;
minLon = Inf;
maxLat = -Inf;
maxLon = -Inf;
% Iterate over files in dataDirectory

for i=1:size(files)
    current = files(i).name;
    fprintf("Current file: %s\n", current);
    current = fullfile(dataDirectory, current);
    try
        % All analysis logic is in doAnalysis
```

```
        [latitudeArray, longitudeArray, altitude, data, ~, ~, ~, ~] =
doAnalysis(current, ...
            save, saveDirectory, showPowerSurfaces, lowerCutOffAltitude,
upperCutOffAltitude,...
            latitude, heightSamplingFrequency, printData);
        altitude = seconds(altitude);
        % the rest of the code here is plotting and error checking.
        if isempty(data)
            continue;
        end
        % Get latitude of bounding box around all radiosondes
        mLat = min(latitudeArray);
        mLon = min(longitudeArray);
        mxLat = max(latitudeArray);
        mxLon = max(longitudeArray);
        if mxLon > maxLon
            maxLon = mxLon;
        end
        if mxLat > maxLat
            maxLat = mxLat;
        end
        if mLat < minLat
            minLat = mLat;
        end
        if mLon < minLon
            minLon = mLon;
        end
        % plot the latitude, longitude, and altitude in 3d
        set(0, 'CurrentFigure', f1);
        plot3(longitudeArray, latitudeArray, altitude);
        % 3D plot magnitudes of quiver have to be 0 or else the plot is
        % weirdly scaled. The 3D plot is just for altitude of detection
and
        % propagation direction.
        magnitudes = 0*data.w_over_f + 0.1;
        angle = data.propagation_dir_compass;
        quiver3(data.lon_of_detection, data.lat_of_detection, ...
            data.alt_of_detection_km*1000, magnitudes.*cosd(angle), ...
            magnitudes.*sind(angle), zeros(size(angle), 'like', angle),
0);
        set(0, 'CurrentFigure', f2);
        offset = 10;
        % Plot a vertical line with placeholder, each one offset by 10
from
        % each other on the x-axis.
        altitude = seconds(altitude);
        placeholder = zeros(size(altitude), 'like', altitude) +
(counter*offset);
        plot(placeholder, altitude/1000, 'k'); % plot altitude in km.
        hold on;
        % xlim([-15, 160]);
        % ylim([9, 60]);
        magnitudes = data.w_over_f;
        % The magnitudes of the red lines are the axial ratios of the
        % gravity waves - axial ratio = intrinsic frequency / coriolis
        % frequency.
        % The for loop below just plots the red lines in the direction of
```

```
        % propagation of the gravity wave
        for q=1:size(data.alt_of_detection_km)
            x1 = counter*offset;
            x2 = counter*offset + magnitudes(q)*cosd(angle(q));
            y1 = data.alt_of_detection_km(q);
            y2 = data.alt_of_detection_km(q) +
magnitudes(q)*sind(angle(q));
            [xf, yf] = ds2nfu([x1 x2], [y1 y2]);
            xf(xf > 1) = 1;
            yf(yf > 1) = 1;
            % arrow([x1 x2], [y1 y2]);
            % annotation(gcf, 'arrow', xf, yf, 'color', '#9a0200',
'LineWidth', 2)
            % plot([x1 x2], [y1 y2], 'color', '#9a0200', 'linewidth', 3);
        end
        scaleFactor = 15;
        if first
            indicesForFilenames = i
            offsets = counter*offset;
            first = false;
        else
            indicesForFilenames = cat(2, indicesForFilenames, i)
            offsets = cat(2, offsets, counter*offset);
        end
        counter = counter + 1;
    catch e

        if (strcmp(e.identifier, 'MATLAB:table:UnrecognizedVarName'))
            fprintf("Data file %s does not contain a variable needed for
analysis. Rerun sounding.\n", files(i).name);
            fprintf("----------------------------\n");
            continue;
        end
        rethrow(e);
    end
        fprintf("----------------------------\n");
end
set(0, 'CurrentFigure', f1);
% xlim([minLon maxLon])
% ylim([minLat maxLat])
xlabel('Longitude (deg)');
ylabel("Latitude (deg)");
zlabel("Altitude (m)");
title("Gravity wave detection altitudes and directions (deg cw from
North)");
set(0, 'CurrentFigure', f2);
allFiles = dir(textFiles);
filenames = allFiles(indicesForFilenames)';

for i=1:size(filenames, 2)
   filenames(i).name = filenames(i).name(1:3);
   filenames(i).name(1) = 'F';
end

xticks(offsets);
yticks([0 10 20 30])
```

```
xticklabels(["0", "10", "20", "30"])
xticklabels({filenames.name});
set(gca,'XTickLabelRotation', 45, 'fontsize', 16)
ylabel("Altitude of detection (km)")
xlabel("Flight number")
title("Propagation direction and detection altitude of gravity waves",
'fontsize', 16);
```

**Read the radiosonde data**

```
function [data] = readRadioSondeData(filename, firstDataLine,
numHeaderLines)

if nargin < 2

    firstDataLine = 21;
    numHeaderLines = 18;
end
opts = detectImportOptions(filename, 'NumHeaderLines', numHeaderLines);
opts.DataLines = [firstDataLine Inf];
data = readtable(filename, opts);
end
```

**Interpolate for spatial uniformity: preprocessData**

```
function [alt, u, v, temp, bvFreqSquared] = preprocessData(alt, u, v,
temp, pressure, time, heightSamplingFrequency)
% This function resamples the data and removes the background.
alt = seconds(alt);
potentialTemperature = (1000.0^0.286)*temp./(pressure.^0.286); % from
Jaxen
tt = timetable(alt, u, v, temp, potentialTemperature, time);
uiq = unique(tt.alt);
tt = retime(tt, uiq);
%tt = retime(tt, 'secondly', 'linear'); % interpolate to regular grid
dt = seconds(heightSamplingFrequency);
tt = retime(tt, 'regular', 'linear', 'TimeStep', dt);
% using linear interpolation.
alt = seconds(tt.alt);
u = tt.u';
v = tt.v';
temp = tt.temp';
time = tt.time';
potentialTemperature = tt.potentialTemperature;
% remove background winds with a moving mean.
altExtent = max(alt) - min(alt);
np = max(fix(altExtent/heightSamplingFrequency/4),  11);
% enforce uniform spatial sampling...
% u = averageToAltitudeResolution(u, alt, heightSamplingFrequency);
% v = averageToAltitudeResolution(v, alt, heightSamplingFrequency);
% temp = averageToAltitudeResolution(temp, alt, heightSamplingFrequency);
% potentialTemperature = averageToAltitudeResolution(potentialTemperature,
alt, heightSamplingFrequency);
% alt = averageToAltitudeResolution(alt, alt, heightSamplingFrequency);
bvFreqSquared = bruntVaisalaFrequency(potentialTemperature,
heightSamplingFrequency); % returns the squared BV frequency.
```

```
meanU = movmean(u, np);
meanV = movmean(v, np);
meanT = movmean(temp, np);
%These next three lines are where the background gets subtracted, however,
%we do not need to do this when applying the wavelet transform. We have
%decided to take the background subtraction out by comenting out the
lines.
%u = u - meanU;
%v = v - meanV;
%temp = temp - meanT;
end
```

**Wavelet transform**

```
classdef WaveletTransform
    %WaveletTransform
    %   This class acts as a logical grouping of variables for a wavelet
    %   transform. The transform and inverse transform rely on the
    %   same parameters (s0, dj, dt) so keeping the parameters grouped
with
    %   the transform will reduce the mental overhead incurred by trying
to
    %   pass the same set of parameters to the forward and backward
    %   transform.

    properties
        s0;
        dj;
        dt;
        u_wind_component;
        v_wind_component;
        powerSurface;
        uWavelet;
        vWavelet;
        waveletScales;
        fourierWavelength;
        tempWavelet;
        coi;
        alt;
        sig95;
    end

    methods
        function obj = WaveletTransform(u_wind_component,
v_wind_component, temperature, dt)

            pad = 1; % Use padding to help with edge effects.
            dj = 0.125/8; % Value of dj determines scale resolution of the
transform.
            s0 = 2*dt; % Minimum resolvable scale. dt is just the sampling
rate -
            % either in time or space.
            obj.u_wind_component = u_wind_component;
            obj.v_wind_component = v_wind_component;
            obj.s0 = s0;
            obj.dj = dj;
```

```
            obj.dt = dt;
            [obj.uWavelet, ~, obj.waveletScales, ~] =
wavelet(u_wind_component, dt, pad, dj, s0); % wave, period, scale, COI
            lag1 = acf(u_wind_component', 1);
            [sigU, ~] = wave_signif(u_wind_component, dt,
obj.waveletScales, 0, lag1);
            [obj.vWavelet, ~, ~, obj.coi] = wavelet(v_wind_component, dt,
pad, dj, s0); % coi is the same for all transforms of the same data.
            lag1 = acf(v_wind_component', 1);
            [obj.tempWavelet, ~, ~, ~] = wavelet(temperature, dt, pad, dj,
s0);
            [sigV, ~] = wave_signif(v_wind_component, dt,
obj.waveletScales, 0, lag1);
            obj.sig95 = (sigU + sigV)'*(ones(1,size(u_wind_component,
2)));
            obj.powerSurface = abs(obj.uWavelet).^2 +
abs(obj.vWavelet).^2;
            obj.sig95 = obj.powerSurface ./ obj.sig95;
            obj.fourierWavelength = 1.03 * obj.waveletScales; % magic
number from Torrence and Compo.
        end

        function [u_wind_reconstructed, v_wind_reconstructed,
tempReconstructed, meanVerticalWavelength, windVariance] =
invertWindowedTransform(obj, windowedWaveletTransform)
            % Reconstruct the wave packet at altitudes of interest and
            % scales of interest by adding up the wavelet coefficients at
            % the scales of interest. This is an implementation of
equation
            % 11 in Torrence and Compo, 1998. The difference is that we
            % keep both the real and imaginary parts of the reconstruction
            % to use in later analysis.
            scale_index_1 = windowedWaveletTransform.scale_index_1;
            scale_index_2 = windowedWaveletTransform.scale_index_2;
            alt_index_1 = windowedWaveletTransform.alt_index_1;
            alt_index_2 = windowedWaveletTransform.alt_index_2;
            constant_coef = obj.dj * sqrt(obj.dt) / (0.776*pi^(1/4)); %
Constant from Torrence and Compo, 1998, given in
            % table 2 and applied in equation 11.
            windowed_scales =
obj.waveletScales(scale_index_1:scale_index_2);
            windowed_u_wavelet = obj.uWavelet(scale_index_1:scale_index_2,
alt_index_1:alt_index_2);
            u_wind_reconstructed = constant_coef*sum(windowed_u_wavelet ./
sqrt(windowed_scales)', 1); % sum over all scales.
            windowed_v_wavelet = obj.vWavelet(scale_index_1:scale_index_2,
alt_index_1:alt_index_2);
            windowedTempWavelet =
obj.tempWavelet(scale_index_1:scale_index_2, alt_index_1:alt_index_2);
            tempReconstructed = constant_coef*sum(windowedTempWavelet ./
sqrt(windowed_scales)', 1);
            v_wind_reconstructed = constant_coef*sum(windowed_v_wavelet ./
sqrt(windowed_scales)', 1);
            meanVerticalWavelength =
mean(obj.fourierWavelength(scale_index_1:scale_index_2)); % the dominant
wavelength is taken
            % to be the mean of the wavelengths over the scale range of
```

```
            % interest. Same as in Murphy, 2014.
            windVariance = abs(u_wind_reconstructed).^2 +
abs(v_wind_reconstructed).^2;
        end

        function [uWindInverted, vWindInverted] =
invertWaveletTransform(obj)
            % This inverts the entire wavelet transform.
           constantCoef = obj.dj * sqrt(obj.dt) / (0.776*pi^(1/4)); %
Magic from T&C.
           uWindInverted = constantCoef*sum(real(obj.uWavelet) ./
sqrt(obj.waveletScales)', 1); % sum over scales
           vWindInverted = constantCoef*sum(real(obj.vWavelet) ./
sqrt(obj.waveletScales)', 1);

        end

        function [a, b, c, d] = clipWindowedTransformToValue(obj,
localMaxRow, localMaxCol)
            % clipWindowedTransformToValue either clips the windowed
            % transform to thresholdValue, or clips it to the next point
            % where the surface starts rising again.
            maxValue = obj.powerSurface(localMaxRow, localMaxCol);
            thresholdValue = 0.25*maxValue;
            column = obj.powerSurface(:, localMaxCol); % extract whole
column
            row = obj.powerSurface(localMaxRow, :); % extract whole row
            % need the locations of the first minima on each side of the
            % local max in both row and column. These locations will
define
            % the windowed power surface before we clip it to
            % thresholdValue.
            [a, b] = findMinimaClosestToIndex(abs(column-thresholdValue),
localMaxRow);
            [c, d] = findMinimaClosestToIndex(abs(row-thresholdValue),
localMaxCol);
            % index into this window and make sure there's only
            % one peak inside this window.
            if b > size(obj.powerSurface, 1)
                x = size(obj.powerSurface, 1);
                fprintf("%d , %d\n", x, b);
            end
            if d > size(obj.powerSurface, 2)
                x = size(obj.powerSurface, 2);
                fprintf("%d , %d\n", x, d);
            end

        end

    end
end
```

### Windowed Wavelet Transform

```
classdef WindowedWaveletTransform
    % Container class to store indices
    % of the windowed wavelet transform.
    properties
        scale_index_1;
        scale_index_2;
        alt_index_1;
        alt_index_2;
    end

    methods
        function obj = WindowedWaveletTransform(scale_index_1 ,
scale_index_2, alt_index_1, alt_index_2)
            %UNTITLED Construct an instance of this class
            %   Detailed explanation goes here
            obj.scale_index_1 = scale_index_1;
            obj.scale_index_2 = scale_index_2;
            obj.alt_index_1 = alt_index_1;
            obj.alt_index_2 = alt_index_2;

        end
    end
end
```

### Data analysis: doAnalysis

```
function [latitudeArray, longitudeArray, altNonFiltered, dataBlock,
waveletTransform, clippedAlt, gWaveLocations, windSpeed] = doAnalysis(f,
save, saveDir, showPowerSurfaces, lowerCutOffAltitude,
upperCutOffAltitude, latitude, heightSamplingFrequency, printData)
% does g-wave analysis for a radiosonde sounding.
if nargin < 6
    upperCutOffAltitude = 40000; % flights never reach 40km.
end
if nargin < 5
    lowerCutOffAltitude = 12000;
end
if nargin < 4
    showPowerSurfaces = false;
end
if nargin < 3
    saveDirExists = false;
else
    saveDirExists = true;
end
if nargin < 2
    save = false;
end
if save
   [~, saveFileName, ~] = fileparts(f);
   % default save in same directory.
   saveFileName = strcat(saveFileName, '_gravity_wave_parameters.csv');
   % if a data directory is provided, use it.
   if saveDirExists
      saveFileName = fullfile(saveDir, saveFileName);
```

```
   end
end
data = readRadioSondeData(f);
[maxAlt, mai] = max(data.Alt);
[~, lai] = min(abs(data.Alt(1:mai) - lowerCutOffAltitude));
[~, mai] = min(abs(data.Alt(lai:mai) - upperCutOffAltitude));
mai = mai + lai;
if maxAlt < lowerCutOffAltitude && upperCutOffAltitude == 40000
    % filter data to altitude bounds
    fprintf("Flight %s did not reach %d m\n", f, lowerCutOffAltitude);
    latitudeArray = [];
    longitudeArray = [];
    altNonFiltered = [];
    dataBlock = [];
    waveletTransform = [];
    clippedAlt = [];
    gWaveLocations = [];
    windSpeed = [];
    return
elseif upperCutOffAltitude ~= 40000 && data.Alt(mai) <
(upperCutOffAltitude)
    fprintf("Flight %s did not reach %d m\n", f, upperCutOffAltitude);
    latitudeArray = [];
    longitudeArray = [];
    altNonFiltered = [];
    dataBlock = [];
    waveletTransform = [];
    clippedAlt = [];
    gWaveLocations = [];
    windSpeed = [];
    return
elseif mai == lai + 1
    latitudeArray = [];
    longitudeArray = [];
    altNonFiltered = [];
    dataBlock = [];
    waveletTransform = [];
    clippedAlt = [];
    gWaveLocations = [];
    windSpeed = [];
    return
end
mai = mai - 1;
% Prepare data

latitudeArray = data.Lat_( 1:mai);
longitudeArray = data.Long_(1:mai);
latitudeArray = latitudeArray(~isnan(latitudeArray));
longitudeArray = longitudeArray(~isnan(longitudeArray));
altNonFiltered = data.Alt(1:mai);
altNonFiltered = altNonFiltered(~isnan(altNonFiltered));

data = data(lai:mai, :);
data = rmmissing(data);
ws = data.Ws;
windSpeed = ws;
```

```
wd = data.Wd;
pressure = data.P;
alt = data.Alt;
temp = data.T+ 273.15;
time = data.Time;

u = -ws .* sind(wd); % from MetPy
v = -ws .* cosd(wd); %

% heightSamplingFrequency = 5;
fprintf("height sampling frequency %d\n", heightSamplingFrequency);
[alt, u, v, temp, bvFreqSquared] = preprocessData(alt, u, v, temp, ...
  pressure, time, heightSamplingFrequency);
clippedAlt = alt;
% calculate constant values to use in analysis
coriolisFreq = coriolisFrequency(latitude);
% finally, do the wavelet transform.
wt = WaveletTransform(u, v, temp, heightSamplingFrequency);
waveletTransform = wt;
% get local maxima that (could) correspond to gravity wave packets
% "Peaks were identified as a function of scale and altitude" - MAL2014.

[rows, cols] = find(imregionalmax(wt.powerSurface, 8)); % 8 for 8-
connectivity

%LMaxFinder = vision.LocalMaximaFinder('MaximumNumLocalMaxima', 10000,
'Threshold', 300);
%idx = LMaxFinder(wt.powerSurface);
%rows = idx(:,2);
%cols = idx(:,1);

% Sort by most power... works now but could use improvement!
pow = zeros(size(rows));
newrows = pow;
newcols = pow;
for k = 1:size(pow)
    pow(k) = wt.powerSurface(rows(k), cols(k));
end
[pow,ind] = sort(pow, 'descend');
for r = 1:size(pow,1)
   newrows(r) = rows(ind(r));
   newcols(r) = cols(ind(r));
end
rows = newrows;
cols = newcols;


if showPowerSurfaces
    f1 = figure;
    set(0, 'CurrentFigure', f1);
    colormap parula;
    [X, Y] = meshgrid(alt, wt.fourierWavelength);
    % Log scale is a temporary testing mechanism...
    surf(X, Y, log(wt.powerSurface), 'edgecolor', 'none');
    % imagesc(alt, wt.fourierWavelength, (wt.powerSurface));
```

```
    [~, titleName, ~] = fileparts(f);
    titleString = sprintf("%s", titleName);
    title(titleString, 'Interpreter', 'none');
    hold on;
    plot(alt, wt.coi, 'k')
    %legend('cone of influence');
    %scatter3(alt(cols(1)), wt.fourierWavelength(rows(1)),
max(wt.powerSurface(:)), 'k*')
    ylabel('vertical wavelength (m)', 'FontSize', 14);
    xlabel('altitude (m)', 'FontSize', 14)
    c = colorbar('FontSize', 14);
    c.Label.String = 'power surface, (m^2/s^2)';
    %c.Label.Interpreter = 'latex';
    set(gca, 'YDir', 'normal');
    set(gca, 'YScale', 'log');
    view([0,90]);
    xlim([alt(1), alt(length(alt))]);
    ylim([wt.fourierWavelength(1),
wt.fourierWavelength(length(wt.fourierWavelength))]);

end
gWaveDetected = false;
first = true;
xValuesForPolygon = 1:size(wt.powerSurface, 2); % get n columns
yValuesForPolygon = wt.coi; % y values that we must check
gWaveLocations = [];
while size(rows) > 0
    % filter local maxima to COI.
    % create a polygon with the COI and query whether or not the x and y
    % values (cols(i) and waveletScales(i)) are within that polygon.
    if ~inpolygon(cols(1), wt.waveletScales(rows(1)), xValuesForPolygon,
yValuesForPolygon)
        % if local maxima candidate is not inside COI polygon, skip it.
        rows = rows(2:end);
        cols = cols(2:end);
        continue;
    end
    % clip the wavelet transform to a box (s1, s2, a1, a2) that
    % corresponds to where the power surface equals 1/4Smax
    [s1, s2, a1, a2] = wt.clipWindowedTransformToValue(rows(1), cols(1));
    if s1 == 0 || s2 == 0 || a1 == 0 || a2 == 0
        % Too close to an edge.
        % This is Keaton -- why are we checking just two edges? And why
are
        % we even doing this check???
        rows = rows(2:end);
        cols = cols(2:end);
        continue
    end

    wwt = WindowedWaveletTransform(s1, s2, a1, a2); % helper object to
ease passing parameters in to invertWaveletTransform
    % invert the wavelet transform in the windowed transform to get a
wave
    % packet.
    % "Wavelet coefficients in the vicinity of the peak were used to
```

```
     % reconstruct temperature and wind perturbations associated with each
     % wave around the wave altitude ``z'' and to estimate the vertical
     % wavelength, lambda_z" - MAL2014.
     [ui, vi, tempi, lambda_z, horizWindVariance] =
wt.invertWindowedTransform(wwt);
     lambda_z = wt.fourierWavelength(rows(1));  % Peak value not mean
     % ^ u reconstructed, v reconstructed, temp reconstructed, vertical
     % wavenumber.
     % estimateParametersFromWavePacket thresholds wave candidates based
on
     % criteria laid out in Murphy et al, 2014.
     [maxVal, ~] = max(horizWindVariance);
     fwhm = find(horizWindVariance >= 0.5*maxVal);
     ui = ui(fwhm);
     vi = vi(fwhm);
     tempi = tempi(fwhm);
     [~, ~, Q, theta, axialRatio, degreeOfPolarization] =
estimateParametersFromWavePacket(ui, vi, tempi);

     if showPowerSurfaces
        set(0, 'CurrentFigure', f1);
        scatter3(alt(cols(1)), wt.fourierWavelength(rows(1)),
max(wt.powerSurface(:)), 'b*');
     end

     if theta == 0
         % theta = 0 when wave packet does not pass filtering criteria.
         rows = rows(2:end);
         cols = cols(2:end);
        continue;
     end
     % theta = azimuthFromUnitCircle(rad2deg(theta));
     theta = rad2deg(theta);
     % all equations below from Murphy et al, 2014:
     % "Radiosonde observations of gravity waves in the lower stratosphere
     %   over Davis, Antartica", table 2.
     intrinsicFreq = coriolisFreq*axialRatio; %1/s
     bvMean = mean(bvFreqSquared(a1:a2)); % Get the mean squared Brunt-
Vaisala frequency over the height range of the gravity wave packet
     if ~((sqrt(bvMean) > intrinsicFreq) && (intrinsicFreq >
coriolisFreq))
         % if the intrinsic frequency is greater than the buoyancy
         % frequency or if it's less than the coriolis frequency, the
         % gravity wave is not physical, so skip it.
         rows = rows(2:end);
         cols = cols(2:end);
         continue
     end
    %gWaveDetected = true;
     %fprintf("m:%f\n", lambda_z);
     m = 2*pi / lambda_z; % vertical wavenumber (1 / meters)
     %k_h =
sqrt(((coriolisFreq^2*m^2)/(bvMean))*(intrinsicFreq^2/coriolisFreq^2 -
1)); % horizontal wavenumber (1 / meters)
```

```
     k_h =
sqrt(((coriolisFreq^2*m^2)/(bvMean))*(intrinsicFreq^2/coriolisFreq^2 -
1));  % CHANGED FOR NOW, PEAK BV!
     intrinsicVerticalGroupVel = -(1 / (intrinsicFreq*m))*(intrinsicFreq^2
- coriolisFreq^2); % m/s
     zonalWaveNumber = k_h*sin(theta);% 1/m
     meridionalWaveNumber = k_h*cos(theta); % 1 / m
     intrinsicVerticalPhaseSpeed = intrinsicFreq / m; % m/s
     intrinsicHorizPhaseSpeed = intrinsicFreq / k_h; % m/s
     %intrinsicZonalGroupVel = zonalWaveNumber * bvMean / (intrinsicFreq *
m^2); % m/s
     intrinsicZonalGroupVel = zonalWaveNumber * bvMean / (intrinsicFreq *
m^2);  % CHANGED FOR NOW, PEAK BV!
     %intrinsicMeridionalGroupVel = meridionalWaveNumber * bvMean /
(intrinsicFreq * m^2); % m/s
     intrinsicMeridionalGroupVel = meridionalWaveNumber * bvMean /
(intrinsicFreq * m^2);  % CHANGED FOR NOW, PEAK BV!
     intrinsicHorizGroupVel = sqrt(intrinsicZonalGroupVel^2 +
intrinsicMeridionalGroupVel^2); % m/s
     lambda_h = 2*pi / k_h; % horizontal wavelength (m)
     altitudeOfDetection = alt(cols(1));%mean(alt(a1:a2)); % mean of the
altitude range as the central altitude of the gravity wave.
     [~, detectionIndex] = min(abs(altNonFiltered - altitudeOfDetection));
     latitudeOfDetection = latitudeArray(detectionIndex);
     longitudeOfDetection = longitudeArray(detectionIndex);
     Period_minutes = (2*pi/intrinsicFreq)/60;

     %These next if statments are additional checks suggested from Dr. Jie
     %Gong

     %This check makes sure the vertical wavelength < (deltaz)/2
     if m > (alt(length(alt))-alt(1))
         disp("2nd check")
         set(0, 'CurrentFigure', f1);
         scatter3(alt(cols(1)), wt.fourierWavelength(rows(1)),
max(wt.powerSurface(:)), 'g*');
         rows = rows(2:end);
         cols = cols(2:end);
         continue
     end

     %This check makes sure that the horizontal wavelength >> balloon
drift
     %distance
     %The following equation is the distance formula
     %We included a conversion for the latitude and longitude to represent
     %kilometers
     if lambda_h/1000 < sqrt(((max(latitudeArray)-
min(latitudeArray))*110.574)^2+((max(longitudeArray)-
min(longitudeArray))*111.320*cos(latitudeOfDetection*pi/180))^2)
        disp("3rd check")
        %disp(lambda_h/1000)
        %disp(sqrt(((max(latitudeArray)-
min(latitudeArray))*110.574)^2+((max(longitudeArray)-
min(longitudeArray))*111.320*cos(latitudeOfDetection*pi/180))^2))
        set(0, 'CurrentFigure', f1);
```

```
        scatter3(alt(cols(1)), wt.fourierWavelength(rows(1)),
max(wt.powerSurface(:)), 'g*');
        rows = rows(2:end);
        cols = cols(2:end);
        continue
    end
    %Suggested checks end here

    gWaveDetected = true;

    xCoordsOfWindow = [a1 a1 a2 a2];
    yCoordsOfWindow = [s2 s1 s1 s2];

    if showPowerSurfaces
        set(0, 'CurrentFigure', f1);
        scatter3(alt(cols(1)), wt.fourierWavelength(rows(1)),
max(wt.powerSurface(:)), 'r*');
        plot(polyshape(alt(xCoordsOfWindow).',
wt.fourierWavelength(yCoordsOfWindow)), 'FaceAlpha', 0, 'EdgeColor', 'r');
    end

    % Check for other peaks inside the same rectangle as current peak
    k = 1;
    while k <= size(rows)
        if inpolygon(cols(k), rows(k), xCoordsOfWindow, yCoordsOfWindow)
            % Remove peak from list if inside the rectangle of current
            % peak
            rows = cat(1,rows(1:k-1),rows(k+1:end));
            cols = cat(1,cols(1:k-1),cols(k+1:end));
            k = k - 1;
        end
        k = k + 1;
    end


    %data = [altitudeOfDetection/1000, coriolisFreq, latitudeOfDetection,
longitudeOfDetection, lambda_z/1000, lambda_h/1000,
azimuthFromUnitCircle(theta), theta, axialRatio, Period_minutes,
intrinsicVerticalGroupVel, intrinsicHorizGroupVel,
intrinsicVerticalPhaseSpeed, intrinsicHorizPhaseSpeed,
degreeOfPolarization, Q, max(alt), min(alt), bvMean];
    data = [altitudeOfDetection/1000, coriolisFreq, latitudeOfDetection,
longitudeOfDetection, lambda_z/1000, lambda_h/1000,
azimuthFromUnitCircle(theta), theta, axialRatio, Period_minutes,
intrinsicVerticalGroupVel, intrinsicHorizGroupVel,
intrinsicVerticalPhaseSpeed, intrinsicHorizPhaseSpeed,
degreeOfPolarization, Q, max(alt), min(alt), bvMean];
    header = {'alt_of_detection_km' 'coriolisFreq' 'lat_of_detection'
'lon_of_detection' 'vert_wavelength_km' 'horiz_wavelength_km'
'propagation_dir_compass' 'propagation_dir_unit_circle' 'w_over_f'
'period_min' 'int_vert_group_vel_ms' 'int_horiz_group_vel_ms'
'int_vert_phase_spd_ms' 'int_horiz_phase_spd_ms' 'degreeofpolarization'
'stokes_param_Q' 'max_alt_m' 'min_alt_m' 'mean_squared_Brunt_Vaisala'};

    if first
```

```
            dataArray = data;
            gWaveLocations = [cols(1) rows(1)];
            first = false;
        else
            dataArray = [dataArray; data];
            gWaveLocations = [gWaveLocations; cols(1) rows(1)];
        end

        if printData
           fprintf("Alt: %f, L_z: %f, L_h: %f, Theta (compass): %f, w/f: %f,
period (hours): %f, Vert. group vel: %f, Horiz. group vel: %f, Vert. phase
spd: %f, Horiz. phase spd: %f, Brunt-Vaisala: %f\n",
altitudeOfDetection/1000, lambda_z/1000, lambda_h/1000,
azimuthFromUnitCircle(theta), axialRatio, (2*pi/intrinsicFreq)/3600,
intrinsicVerticalGroupVel, intrinsicHorizGroupVel,
intrinsicVerticalPhaseSpeed, intrinsicHorizPhaseSpeed, bvMean);
        end
end
if save && ~isfile(saveFileName) && gWaveDetected
            % check if the file exists - if it doesn't, write a header to
            % the file to ease further analysis.
            % writecell([header; num2cell(data)], saveFileName);
            % PROP DIR> azimunth from unti cirlc
            dataBlock = array2table(dataArray, 'VariableNames', header);
            writetable(dataBlock, saveFileName);

elseif save && isfile(saveFileName) && gWaveDetected
            % if the file does exist, append to it.
            % There is no overwrite functionality, so it's possible to run
            % the analysis multiple times and write duplicates to a file.
            % This must be taken care of in later analysis.
            dlmwrite(saveFileName, dataArray, 'delimiter', ',', '-append');
            dataBlock = array2table(dataArray, 'VariableNames', header);
end
if ~gWaveDetected
    fprintf("No gravity waves detected.\n");
    dataBlock = [];
elseif gWaveDetected
    dataBlock = array2table(dataArray, 'VariableNames', header);
end
end
```

**Coriolis Frequency**

```
function [f] = coriolisFrequency(latitude)
%Calculate Coriolis frequency at latitude.

f = 2*7.2921e-5*sind(latitude);
end
```

**Brunt- Väisälä Frequency**

```
function [N2] = bruntVaisalaFrequency(potentialTemperature,
heightSamplingFrequency)
% Calculates BV frequency for a whole sounding, using
% N^2 = g/theta * dtheta / dz, where theta is potential temperature.
```

```
% Validate vertical profiles of bvfreq w/ papers
g = 9.8;
N2 = (g ./ potentialTemperature) .* gradient(potentialTemperature,
heightSamplingFrequency);
end
```

**Find minima closest to index**

```
function [id1, id2] = findMinimaClosestToIndex(array, index)
%   Gets the two closest minima on either side of index in array.
%   Returns the indices into the array of the minima.
[~, locs] = findpeaks(-array); % find the maxima (the negative minima).
locs = sort(locs);
id2 = 0;
id1 = 0;
if size(locs, 1) == 1
    sz = size(locs, 2);
else
    sz = size(locs, 1);
end
for i=2:sz
    if (locs(i-1) < index) && (locs(i) > index)
        id1 = locs(i-1);
        id2 = locs(i);
    end
end
end
```

**Extract window around local max**

```
function [windowedWaveletTransform] =
extractWindowAroundLocalMax(powerSurface, localMaxCol, localMaxRow)
% Returns a custom object that makes inverting the wavelet power surface
easier.
ofs = 99;
n_rows = size(powerSurface, 1);
n_cols = size(powerSurface, 2);

first_row = localMaxRow - ofs;
last_row = localMaxRow + ofs;
first_col = localMaxCol - ofs;
last_col = localMaxCol + ofs;
if first_row < 1
   first_row = 1;
end
if last_row > n_rows
   last_row = n_rows;
end
if last_col > n_cols
   last_col = n_cols;
end
if first_col < 1
   first_col = 1;
end
windowedWaveletTransform = WindowedWaveletTransform(first_row, last_row,
first_col, last_col);
end
```

**Estimate parameters from wave packet**

```
function [D, P, Q, theta, axialRatio, degreeOfPolarization] =
estimateParametersFromWavePacket(uWavePacket, vWavePacket, tempWavePacket)
%   Estimate gravity wave parameters from a wave packet using the Stokes
%   parameters method.
u = real(uWavePacket);
v = real(vWavePacket);
vWavePacketHilbertTransformed = imag(vWavePacket);
% Stokes parameters from Murphy et al, 2014, Appendix A.
I = mean(u.^2) + mean(v.^2);
D = mean(u.^2) - mean(v.^2);
P = mean(2*u.*v);
Q = mean(2*u.*vWavePacketHilbertTransformed);
degreeOfPolarization = sqrt((P^2 + Q^2 + D^2)) / I;
% perform filtering based on Stokes parameters and degree of polarization.
% TODO test DOP threshold values and examine 'depolarization' in Murphy et
% al
if abs(Q) < 0.05 || abs(P) < 0.05 || degreeOfPolarization < 0.5 ||
degreeOfPolarization > 1
    theta = 0;
    axialRatio = 0;
    degreeOfPolarization = 0;
    Q = 0;
    return;
else
    theta = 0.5 * atan2(P, D); % Zink, 2000 eqn 3.14 TODO investigate
atan2
    % Axial ratio
    axialRatio = abs(cot(0.5*asin(Q/(degreeOfPolarization*I))));
    % Murphy et al (2014), Koushik et al (2019), and Vincent (1989)
    % Look at phase b/t u' and T+90'
    % From the polarization relations, we know that u' and T' are +-90 deg
    % out of phase, depending on the sign of the horizontal wavenumber.
The
    % polarization relations are in a frame that is traveling with the
wave
    % in the same direction, so we have to rotate the zonal wind component
    % (u) to align with the wave's direction.
    rotationMatrix = [cos(theta) sin(theta); -sin(theta) cos(theta)];
    uv = [uWavePacket; vWavePacket];
    uvRotated = rotationMatrix * uv;
    % axialRatio = abs(mean(uv(1, :)) / mean(uv(2, :)));
    uPar = uvRotated(1, :);
    % vPerp = uvRotated(2, :);

    % Coherence function, from Zink 2000
    gamma = mean(uPar.*conj(tempWavePacket)) ./
sqrt(mean(abs(uPar).^2).*mean(abs(tempWavePacket).^2));

    phase = atan2(imag(gamma), real(gamma));
    if phase <= 0
        theta = theta + pi;
    end
    if axialRatio < 1
        % i.e. if the intrinsic frequency is less than the coriolis
        % frequency. This does not agree with theory, so the wave packet
```

```
        % is discarded.
        % Assing NaN to unphysical values?
        theta = 0;
        axialRatio = 0;
        degreeOfPolarization = 0;
        Q = 0;
        return;
    end
end
end
```

**Filter wave packet candidates**

```
function [goodMaximaRows, goodMaximaCols] =
filterWavePacketCandidates(waveletTransform, localMaximaRows,
localMaximaCols)
%filterWavePacketCandidates Filters wave packet candidates by calculating
the Stokes
%parameters and degree of polarization. If Q < 0.05 || P < 0.05, the
%candidate is discarded. If d < 0.5 | d > 1, the candidate is also
%discarded. Otherwise, the file name, bounding box (s1, s2, z1, z2), the
%position of the local max, and the parameters are stored.

goodMaximaRows = zeros('like', localMaximaRows);
goodMaximaCols = zeros('like', localMaximaCols);
k = 0;
for i=1:size(localMaximaRows, 1)
    % first, clip the local maxima to 1/4Smax.
    currentMaxRow = localMaximaRows(i);
    currentMaxCol = localMaximaCols(i);
    %windowedWaveletTransform =
extractWindowAroundLocalMax(waveletTransform.powerSurface, currentMaxCol,
currentMaxRow);
    [row_index_1, row_index_2, col_index_1, col_index_2] =
waveletTransform.clipWindowedTransformToValue(currentMaxRow,
currentMaxCol);
    if row_index_1 == 0 || row_index_2 == 0 || col_index_1 == 0 ||
col_index_2 == 0
       %fprintf("Maxima too close to power surface edge. Most likely noise
anyway.\n");
       continue;
    end
    oneQuarterMaxWindow = WindowedWaveletTransform(row_index_1,
row_index_2, col_index_1, col_index_2);
    [u, v, temp, ~] =
waveletTransform.invertWindowedTransform(oneQuarterMaxWindow);
    [theta, axialRatio, degreeOfPolarization] =
estimateParametersFromWavePacket(u, v, temp);
    if theta == 0 || axialRatio == 0 || degreeOfPolarization == 0
        %fprintf("Wave packet did not satisfy critera\n");
        continue;
    end
    k = k + 1;
    goodMaximaRows(k) = currentMaxRow;
    goodMaximaCols(k) = currentMaxCol;
end
```

```
goodMaximaRows = goodMaximaRows(1:k);
goodMaximaCols = goodMaximaCols(1:k);
end
```

**Average to altitude resolution**

```
function [averagedArray, averagedAlt] = averageToAltitudeResolution(array,
altitude, resolution)
%   Averages a time series of data to a given altitude resolution. Steps:
%   walk through the altitude array, keeping a running sum of the values
in
%   "array" at the same indices. When altBegin - altEnd > resolution,
%   divide the sum by the number of indices between altBegin and altEnd,
%   and store it in a new array - averagedArray.

averagedArray = zeros('like', array);
averagedAlt = zeros('like', altitude);
altBegin = altitude(1);
altEnd = altitude(1);
k = 0;
s = 0;
nPoints = 0;
for i=1:size(altitude, 1)
    altEnd = altitude(i);
    s = s + array(i);
    nPoints = nPoints + 1;
    if (altEnd - altBegin) >= resolution
        k = k + 1;
        averagedArray(k) = s / nPoints;
        averagedAlt(k) = altBegin + (altEnd - altBegin) / 2;
        s = 0;
        altBegin = altEnd;
        nPoints = 0;
    end
end
averagedArray = averagedArray(1:k);
averagedAlt = averagedAlt(1:k);
end
```

**Azimuth from unit circle**

```
function [azimuth] = azimuthFromUnitCircle(theta)
%   Theta is in degrees. This function converts theta from degrees
%   counterclockwise from east to degrees clockwise from north.
%   Detailed explanation goes here
azimuth = 450 - theta;
if (azimuth > 360)
    azimuth = azimuth - 360;
end
end
```

**Plot gravity waves**

```
function plotGravityWaveData(data, altitudeArray, ...
```

```
    waveletTransform, gWaveLocations, clippedAlt, axes1, axes2)
    placeholder = zeros(size(altitudeArray), 'like', altitudeArray);
    plot(axes2, placeholder, altitudeArray/1000, 'k'); % plot altitude in
km.
    hold(axes2, 'on');
    % The magnitudes of the red lines are the axial ratios of the
    % gravity waves - axial ratio = intrinsic frequency / coriolis
    % frequency.
    % The for loop below just plots the red lines in the direction of
    % propagation of the gravity wave
    offset = 0;
    if ~isempty(data)
        for q=1:size(data.alt_of_detection_km)
            x1 = offset;
            magnitude = data.axial_ratio(q);
            angle = data.propagation_dir(q);
            x2 = offset + magnitude*cosd(angle);
            y1 = data.alt_of_detection_km(q);
            y2 = data.alt_of_detection_km(q) + magnitude*sind(angle);
            plot(axes2, [x1 x2], [y1 y2], 'r', 'DisplayName', 'g-wave');
        end
    end
    xlim(axes2, [-5 5]);
    title(axes2, "Gravity wave altitude of detection and propagation
direction", 'FontSize', 8);
    ylabel(axes2, 'Altitude (km)');
    set(axes1, 'XTickMode', 'auto', 'XTickLabelMode', 'auto');
    set(axes1, 'YTickMode', 'auto', 'YTickLabelMode', 'auto');
    %set(app.UIAxes_2, 'XTickMode', 'auto', 'XTickLabelMode', 'auto');
    set(axes2, 'YTickMode', 'auto', 'YTickLabelMode', 'auto');
    if ~isempty(waveletTransform)
        imagesc(axes1, clippedAlt, waveletTransform.fourierWavelength,
log10(waveletTransform.powerSurface));
        hold(axes1, 'on');
        plot(axes1, clippedAlt, waveletTransform.coi, 'k');
    end
    if ~isempty(data) && ~isempty(gWaveLocations)
        s = scatter(axes1, clippedAlt(gWaveLocations(:, 1)),
waveletTransform.fourierWavelength(gWaveLocations(:, 2)), 'ro');
        legend(axes1, s, 'gravity wave');
    end
    if ~isempty(waveletTransform)
        ylim(axes1, [waveletTransform.s0 Inf]);
        ylabel(axes1, 'Wavelength (m)');
        xlabel(axes1, 'Altitude (m)');
    end

end
```

# Appendix B - Lufft Set-Up

Slide the Lufft sensor on top of the white dome on the end of the metal rod. (leave bolts semi tight to allow slight up down movement for when attaching the cable from datalogger box)

Screw datalogger box onto metal pole using configuration below (u-bolt/pole/thick bracket/box/thin bracket/washer/bolt):

Slide the metal rod with the Lufft sensor on the end (with the stopper firmly attached) and datalogger box attached, into the opening in the stand.

Make sure the "North" arrow is pointing north (the arrow is on top of the Lufft sensor) then tighten bolts on both top and bottom ring to secure metal pipe. (the bolts that secure the pipe have 2 nuts each. The inside ones are hard to adjust once the metal pipe is in the ring, so tighten all reasonably evenly and close to final position before sliding in the pipe)

Run the green tipped cord out of one hole in the bottom of the datalogger box, and along the side. This cord can be placed inside the side connectors of the box without being pinched. (see pictures on top of next page)

Connect the green-tipped cord to the bottom of the Lufft sensor - it is directional, so investigate the connection before forcing it on. (this is where you may want to slide the Lufft head up to see better when attaching the cable)

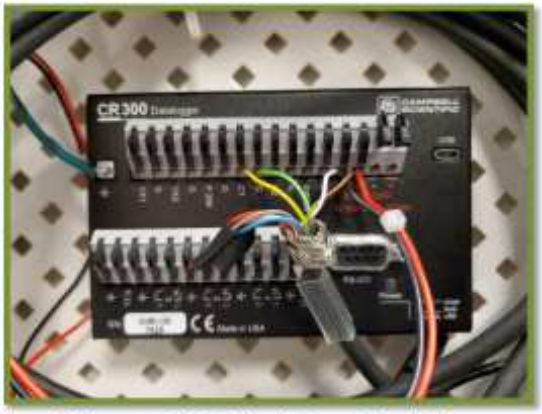
The wiring is as follows for the data cord: (picture to the right)

Yellow wire to port C1

Green wire to port G

White wire and black battery lead to port – BAT

Brown wire and red battery lead to port + BAT




Hook the solar panel to the Lufft datalogger by putting the leads through the unused hole in the bottom of the datalogger box and into the CHG +/- port on the right-hand side of the CR300 datalogger (see pictures below; red is positive and black is negative).

# Appendix C - SOP for Running GRAW and Initializing Radiosonde

- BEFORE STARTING GRAWMET:
  - Open the radiosonde-careful to not the end of the sensor. The initialization cable can be plugged into the sonde.
  - Do not switch on radiosonde until told so
- Turn on the Ground Station and connect before starting GRAW program
- Open Grawmet 5.15
- Weigh the payload, latitude, longitude and altitude record for future use
- Plug in the radiosonde (do not switch on, repeat do not turn on)
- Click on the tab to the left of the Start tab
- Open General settings (notebook icon)
- Click on the Program Settings tab > Communications tab
- Detect both Com Ports and Receiver
- Click the back arrow and exit
- Click Sounding/Simulation (do not use wizard)
- Click initialize radiosonde
  - *If initialization fails 3 times switch to a new sonde*
- Set sonde and Ground station to proper frequency (idaho team: 401.010)
- Enter ground station values recorded from Lufft
  - Check them with the kestrel
- Click start sounding
- Enter lat/long and altitude (both lat and long requires negative signs in chile)
- Confirm raw data is coming in
- Turn on radiosonde to on position
- Start filling balloon
- Unplug radiosonde from computer
- Hang it to acclimate same height as the lufft
  - *The radiosonde has 10 minutes to acclimate*
- Double check raw data is still coming in
- Get the sonde and attach it to the balloon
- Launch balloon

# Appendix D - Helium Fill Code

```
8   ##########################################################################################################################
9   """
10  This code sets up a dialog box that takes the numbers input and extracts them from the function: Helium Calculations.
11  "Helium Calculations" extraplates the user inputs and formats golbal variables used in the rest of the code.
12  It then runs through the calculations, and creates an ouput with the answers.
13  The output is produced in a messagebox popup as well as in the console. A predetermined excel sheet is opened
14  and a function finds the last row and writes the inputs,outputs,and timestamp to an excel document that saves.
15  """
16
17
18  """
19  ***NOTE: If planning to use Excel Push feature, download the excel template from BOX
20          and save in same folder as this code!
21          Otherwise, create new excel sheet and adjust file name and sheet name in lower section of code
22
23  Operational Instructions:
24      -Run Code
25      -Add values from LUFFT to the dialog box pop up
26      -Select Calculate
27      -Read Values from output dialog box
28      -Select OK to push data to Excel
29  """
30
31  ########Excel Sheet Name - Change for name you are using (USE TEMPLATE FROM BOX) - #########
32
33  excel = '2020LABCOATS_HeliumCalculations.xlsx'
34
35  #Idaho's Regulator is old and slightly faulty, correction factor of +100 PSI through trail and error
36  CorFact = 0  # Regulator Correction Factor (IDAHO's regulator requires 100 PSI more to make correct volume)
37
38  ##########################################################################################################################
39  from sympy.solvers import solve
40  from sympy import Symbol
41  import numpy as np
42  import xlwings as xw
43  import datetime
44  import sys
45  from tkinter import Tk, mainloop,Label,Entry,Button,W,messagebox
46  import metpy.calc as mpcalc     #IF USING ANACONDA, MUST DOWNLOAD METPY MODULE
47  from metpy.units import units
48
49  now = datetime.datetime.utcnow()
50  Date = (now.strftime("%Y-%m-%d"))
51  Time = (now.strftime("%H:%M:%S"))
52
```

```
52
53
54      ############CREATES GLOBAL VARIABLES FROM USER INPUT############
55
56      def helium_calculations():
57          global P,T,mb,mp,SondeNum,P_in,T_in,mb_in,mp_in,rh,TnkP
58          SondeNum = float(num1.get())        #RADIOSONDE NUMBER
59          P_in = float(num2.get())            #GROUND PRESSURE INPUT
60          P = P_in*100.0                      #PRESSURE CONVERT TO Pa
61          T_in = float(num3.get())            #GROUND TEMP INPUT
62          T = T_in+273.15                     #TEMP CONVERT TO K
63          rh = float(num4.get())              #GROUND HUMIDITY INPUT
64          TnkP = float(num5.get())            #INITIAL TANK PRESSURE
65          mb_in = float(num6.get())           #BALLOON MASS INPUT
66          mb = mb_in/1000.0                   #MASS CONVERT TO KG
67          mp_in = float(num7.get())           #PAYLOAD MASS INPUT
68          mp = mp_in/1000.0                   #MASS CONVERT TO KG
69
70
71      def output1():
72          global CalcTest,AdvCalc
73          CalcTest = 1
74          AdvCalc  = 0
75          return
76      def output2():
77          global AdvCalc,CalcTest
78          AdvCalc  = 1
79          CalcTest = 0
80          return
```

```
81
82      ############CREATES DIALOG BOX FOR GROUND DATA############
83      main = Tk()
84      main.title('Ground Condition Data')
85      Label(main,text = "Sonde Number:").grid(row=0)
86      Label(main, text = "Ground Pressure (hPa)").grid(row=1)
87      Label(main, text = "Ground Temperature (C):").grid(row=2)
88      Label(main, text = "Humidity (%)").grid(row=3)
89      Label(main, text = "Tank Pressure (PSI):").grid(row=4)
90      Label(main, text = "Balloon Mass (g):").grid(row=5)
91      Label(main, text = "Payload Mass (g):").grid(row=6)
92      Label(main, text = "Created for ISGC Ballooning Campaign").grid(row=7)
93
94      num1 = Entry(main)              #SN
95      num2 = Entry(main)              #GP
96      num3 = Entry(main)              #GT
97      num4 = Entry(main)              #GH
98      num5 = Entry(main)              #BM
99      num6 = Entry(main)              #PM
100     num7 = Entry(main)
101
102     num1.grid(row=0, column=1)      #SN
103     num2.grid(row=1, column=1)      #GP
104     num3.grid(row=2, column=1)      #GT
105     num4.grid(row=3, column=1)      #GH
106     num5.grid(row=4, column=1)      #BM
107     num6.grid(row=5, column=1)      #PM
108     num7.grid(row=6, column=1)
109
110
111     Calc=Button(main, text='Calculate', command=lambda:[helium_calculations(),output1(),main.destroy()]).grid(row=1, column=3, sticky=W, pady=4, padx=50)
112     AdvCalc=Button(main,text = 'Advanced Calculations', command=lambda:[helium_calculations(),output2(),main.destroy()]).grid(row=3,column=3,sticky=W,pady
113     Quit=Button(main, text='Quit', command=lambda:[main.destroy(),sys.exit()]).grid(row=5, column=3, pady=4, padx=10)
114     mainloop()
115
116
117
118     ############HELIUM FILL RATE/RISE RATE CALCULATIONS#############
119     m = Symbol('m')
120     Mhe = 0.004                 #HELIUM MOLAR MASS
121     Mar = 0.02896               #AIR MOLAR MASS
122     B = 8.314                   #GAS CONSTANT
123     g = 9.8                     #ACCELERATION DUE TO GRAVITY
124     V_tank = .049               #VOLUME OF TANK IN L
125     v = 5.0                     #RISE VELOCITY
126     Cd = 0.285                  #DRAG COEFFICIENT
127
128
129     Dhe = (P * Mhe)/(B * T)     #Density of Helium
130     mixingRatio = mpcalc.mixing_ratio_from_relative_humidity(rh,T*units.degK,P*units.Pa)   ##CALCULATE DENSITY OF AIR USING GC HUMIDITY##
131     Dar = mpcalc.density(P*units.Pa,T*units.degK,mixingRatio)
132     Dar = Dar.magnitude         #Density of Air
133
134
```

```
136
137     R = ((3 * m * B * T)/(4.0 * np.pi * Mhe * P))**(1.0/3.0)          #RADIUS @ LAUNCH ALT.
138     A = np.pi*(R**2)                                                  #CROSS-SECTIONAL AREA @ LAUNCH ALT.
139     V = (m * B * T)/(Mhe * P)                                         #VOLUME @ LAUNCH ALT
140
141     Drag = (0.5 * Dar * (v**2) * Cd * A)
142
143     Mass = solve(((V * g)*(Dar)) - Drag - ((mb + mp + m) * g), m)[0]   #Statics: Boyant Force - Drag Force - mg=0, m=0, Solve m=Mass
144     MolesHE = Mass/Mhe                                                #Find number of Moles in balloon needed
145     LSTP = MolesHE*22.4                                               #Convert the number of Moles to Liters
146     print ('Fill Program Volume SL: ')
147     print (LSTP)
148     print ('Mass: ')
149     print (Mass)
150     print('Moles')
151     print(MolesHE)
152
153     Vm = (MolesHE*B*T)/(P) #VOLUME OF GAS IN M^3
154
155     print ('Volume: ')
156     print (Vm)
157     Vf = Vm/0.0283168
158     Vi = Vm*61023.7 #Volume of Gas in^3
159     print('Vi:')
160     print(Vi)
161
162
163     Pm = (MolesHE*B*T)/V_tank       #Calculate Pressure using PV=nRT
164     Pf = Pm*.000145038              #Convert from Pa to PSI
165
166
167     Pf = Pf + CorFact         #Pressure in PSI to read from Regulator with regulator correction factor included (see line 48)
168     newTkP = TnkP-Pf          #Tank Pressure for IDAHO's regulator
169
170
171
172     print ('Fill PSI:' )
173     print (Pf)
174
175     R_out = ((3 * Mass * B * T)/(4.0 * np.pi * Mhe * P))**(1.0/3.0)         #RADIUS OF BALLOON AFTER FILL
176     R_out_inch = R_out*39.37                                                #RADIUS AFTER FILL - INCHES
177     A_out = np.pi*(R_out**2)                                                #CROSS-SECTIONAL AREA AFTER FILL
178     V_out = (Mass * B * T)/(Mhe * P)                                        #VOLUME OF BALLOON AFTER FILL
179     Circ_out = 2*np.pi*R_out_inch
180     Drag_out = (0.5 * Dar * (v**2) * Cd * A_out)                            #CIRCUMFERENCE OF BALLOON AFTER FILL
181
```

```
183     ###################################CREATES OUTPUT OF THE VALUES FROM ABOVE CALCULATIONS###########
184     #This section is so you can view the results, no new math included here
185
186     if CalcTest == 1:
187         output = "New Tank Pressure: %.3f PSI" %(newTkP)
188         output += "\n"
189         output += "\nDelta P: %.3f PSI" %(Pf)
190         output += "\n"
191         output += "\nFP Volume: %.3f SL" %(LSTP)
192
193         root = Tk()
194         root.withdraw()
195         messagebox.showinfo("Results",output)
196         root.destroy()
197
198     if AdvCalc == 1:
199         output = "Fill Program Volume: %.3f SL" %(LSTP)
200         output += "\n"
201         output += "\nFinal Tank Pressure: %.3f PSI" %(newTkP)
202         output += "\n"
203         output += "\nFill PSI: %.3f PSI" %(Pf)
204         output += "\nMass: %.3f kg" %(Mass)
205         output += "\nVolume: %.3f m^3" %(Vm)
206         output += "\n"
207         output += "\nRadius: %.3f inches" %(R_out_inch)
208         output += "\nCircumference: %.3f inches" %(Circ_out)
209         output += "\nDrag force: %.3f" %(Drag_out)
210
211         root = Tk()
212         root.withdraw()
213         messagebox.showinfo("Advanced Results",output)
214         root.destroy()
215
216
217
218
219     ############ MESSAGE WINDOW DISPLAYING OUTPUT CALCULATIONS############
220
221     ############OPENS AND IMPORTS DATA INTO EXCEL############
222
223     ### MUST CREATE EXCEL SHEET NAMED GROUND LAUNCH DATA, WITH SHEET NAME GROUND DATA OR USE TEMPLATE FROM BOX#########
224     def write_data_to_excel(workbooklocation,sheetname,columnletter):    #Function finds last row in excel sheet and moves to next row
225           wb = xw.Book(workbooklocation)
226           X = wb.sheets[sheetname].range(columnletter + str(wb.sheets[sheetname].cells.last_cell.row)).end('up').row + 1
227           cell = columnletter + str(X)
228           return cell
229
230     wb = xw.Book(excel)
231     sht = wb.sheets['Ground Data']
232
233     last_row = write_data_to_excel(excel,'Ground Data','A')
234     sht.range(last_row).value = [Date,Time,SondeNum,P_in,T_in,rh,TnkP,mb_in,mp_in,LSTP,Mass,Vm,Pf,newTkP,R_out_inch,Circ_out] #Writes data to empty row in excel
235     wb.save()    #Saves Excel Sheet
```

# Appendix E - Rerunning Profiles in Graw

- Open GRAW
- Click on Sounding/Simulation (do not start the Wizard)
- On the left tab click on simulation
- Open zip file of desired profile to be ran.
- Make sure all inputs are correct
- Click start
    - When reports are done, a message window at the bottom will indicate the completion of reports created
- Go to reports
- Click on profile data
- Save profile data as a txt file.
    - W#_L#_hhmmUTC_mmddyy_computer_Profile

# Appendix F - Frequency Allocations for Eclipse 2020

**Table F-1.** Frequency allocations for Villarrica during 2020 solar eclipse.

| GSU | Flight ID | UTC | Frequency (MHz) | Saving Convention |
|---|---|---|---|---|
| | | ***December*** | ***13th*** | ***2020*** |
| ID-1 | V1 | 16:02 | 401.610 | V1_UTC_12**13**20_Computer |
| ID-2 | V2 | 17:02 | 404.810 | V2_UTC_12**13**20_Computer |
| ID-1 | V3 | 18:02 | 401.610 | V3_UTC_12**13**20_Computer |
| ID-2 | V4 | 19:02 | 404.810 | V4_UTC_12**13**20_Computer |
| ID-1 | V5 | 20:02 | 401.610 | V5_UTC_12**13**20_Computer |
| ID-2 | V6 | 21:02 | 404.810 | V6_UTC_12**13**20_Computer |
| ID-1 | V7 | 22:02 | 401.610 | V7_UTC_12**13**20_Computer |
| ID-2 | V8 | 23:02 | 404.810 | V8_UTC_12**13**20_Computer |
| | | ***December*** | ***14th*** | ***2020*** |
| ID-1 | V9 | 00:02 | 401.610 | V9_UTC_12**14**20_Computer |
| ID-2 | V10 | 01:02 | 404.810 | V10_UTC_12**14**20_Computer |
| ID-1 | V11 | 02:02 | 401.610 | V11_UTC_12**14**20_Computer |
| ID-2 | V12 | 03:02 | 404.810 | V12_UTC_12**14**20_Computer |
| ID-1 | V13 | 04:02 | 401.610 | V13_UTC_12**14**20_Computer |
| ID-2 | V14 | 05:02 | 404.810 | V14_UTC_12**14**20_Computer |
| ID-1 | V15 | 06:02 | 401.610 | V15_UTC_12**14**20_Computer |
| ID-2 | V16 | 07:02 | 404.810 | V16_UTC_12**14**20_Computer |
| ID-1 | V17 | 08:02 | 401.610 | V17_UTC_12**14**20_Computer |
| ID-2 | V18 | 09:02 | 404.810 | V18_UTC_12**14**20_Computer |
| ID-1 | V19 | 10:02 | 401.610 | V19_UTC_12**14**20_Computer |
| ID-2 | V20 | 11:02 | 404.810 | V20_UTC_12**14**20_Computer |
| ID-1 | V21 | 12:02 | 401.610 | V21_UTC_12**14**20_Computer |
| ID-2 | V22 | 13:02 | 404.810 | V22_UTC_12**14**20_Computer |
| ID-1 | V23 | 14:02 | 401.610 | V23_UTC_12**14**20_Computer |
| ID-2 | V24 | 15:02 | 404.810 | V24_UTC_12**14**20_Computer |
| MT-1 | V25 | 15:32 | 400.810 | V25_UTC_12**14**20_Computer |
| ID-1 | V26 | 16:32 | 404.010 | V26_UTC_12**14**20_Computer |
| ID-2 | V27 | 17:32 | 402.810 | V27_UTC_12**14**20_Computer |

| | | | | |
|---|---|---|---|---|
| ID-1 | V28 | 18:32 | 400.010 | V28_UTC_12**14**20_Computer |
| ID-2 | V29 | 19:32 | 403.210 | V29_UTC_12**14**20_Computer |
| ID-1 | V30 | 20:32 | 400.010 | V30_UTC_12**14**20_Computer |
| ID-2 | V31 | 21:32 | 404.010 | V31_UTC_12**14**20_Computer |
| ID-1 | V32 | 22:32 | 403.210 | V32_UTC_12**14**20_Computer |
| ID-2 | V33 | 23:32 | 400.010 | V33_UTC_12**14**20_Computer |
| ID-1 | V34 | 00:32 | 403.210 | V34_UTC_12**14**20_Computer |
| ID-2 | V35 | 01:32 | 400.010 | V35_UTC_12**14**20_Computer |
| ID-1 | V36 | 02:32 | 403.210 | V36_UTC_12**14**20_Computer |
| | | ***December*** | ***15th*** | ***2020*** |
| ID-2 | V37 | 03:32 | 400.010 | V37_UTC_12**15**20_Computer |
| ID-1 | V38 | 04:32 | 403.210 | V38_UTC_12**15**20_Computer |
| ID-2 | V39 | 05:32 | 400.010 | V39_UTC_12**15**20_Computer |
| ID-1 | V40 | 06:32 | 403.210 | V40_UTC_12**15**20_Computer |
| ID-2 | V41 | 07:32 | 400.010 | V41_UTC_12**15**20_Computer |
| ID-1 | V42 | 08:32 | 403.210 | V42_UTC_12**15**20_Computer |
| ID-2 | V43 | 09:32 | 400.010 | V43_UTC_12**15**20_Computer |
| ID-1 | V44 | 10:32 | 403.210 | V44_UTC_12**15**20_Computer |
| ID-2 | V45 | 11:32 | 400.010 | V45_UTC_12**15**20_Computer |
| ID-1 | V46 | 12:32 | 403.210 | V46_UTC_12**15**20_Computer |
| ID-2 | V47 | 13:32 | 400.010 | V47_UTC_12**15**20_Computer |
| ID-1 | V48 | 14:32 | 403.210 | V48_UTC_12**15**20_Computer |
| ID-2 | V49 | 15:32 | 400.010 | V49_UTC_12**15**20_Computer |
| ID-1 | V50 | 16:32 | 403.210 | V50_UTC_12**15**20_Computer |

**Table F-2.** Frequency allocation for Tolten during 2020 solar eclipse.

| **GSU** | **Flight ID** | **UTC** | **Frequency (MHz)** | **Saving Convention** |
|---|---|---|---|---|
| | | ***December*** | ***13th*** | ***2020*** |
| OK-1 | T1 | 16:02 | 400.010 | T1_UTC_12**13**20_Computer |
| OK-2 | T2 | 17:02 | 403.210 | T2_UTC_12**13**20_Computer |
| OK-1 | T3 | 18:02 | 400.010 | T3_UTC_12**13**20_Computer |
| OK-2 | T4 | 19:02 | 400.010 | T4_UTC_12**13**20_Computer |
| OK-1 | T5 | 20:02 | 403.210 | T5_UTC_12**13**20_Computer |

| | | | | |
|---|---|---|---|---|
| OK-2 | T6 | 21:02 | 400.010 | T6_UTC_12**13**20_Computer |
| OK-1 | T7 | 22:02 | 403.210 | T7_UTC_12**13**20_Computer |
| OK-2 | T8 | 23:02 | 400.010 | T8_UTC_12**13**20_Computer |
| | | ***December*** | ***14th*** | ***2020*** |
| OK-1 | T9 | 00:02 | 400.010 | T9_UTC_12**14**20_Computer |
| OK-2 | T10 | 01:02 | 403.210 | T10_UTC_12**14**20_Computer |
| OK-1 | T11 | 02:02 | 400.010 | T11_UTC_12**14**20_Computer |
| OK-2 | T12 | 03:02 | 403.210 | T12_UTC_12**14**20_Computer |
| OK-1 | T13 | 04:02 | 400.010 | T13_UTC_12**14**20_Computer |
| OK-2 | T14 | 05:02 | 403.210 | T14_UTC_12**14**20_Computer |
| OK-1 | T15 | 06:02 | 400.010 | T15_UTC_12**14**20_Computer |
| OK-2 | T16 | 07:02 | 403.210 | T16_UTC_12**14**20_Computer |
| OK-1 | T17 | 08:02 | 400.010 | T17_UTC_12**14**20_Computer |
| OK-2 | T18 | 09:02 | 403.210 | T18_UTC_12**14**20_Computer |
| OK-1 | T19 | 10:02 | 400.010 | T19_UTC_12**14**20_Computer |
| OK-2 | T20 | 11:02 | 403.210 | T20_UTC_12**14**20_Computer |
| OK-1 | T21 | 12:02 | 400.010 | T21_UTC_12**14**20_Computer |
| OK-2 | T22 | 13:02 | 403.210 | T22_UTC_12**14**20_Computer |
| OK-1 | T23 | 14:02 | 400.010 | T23_UTC_12**14**20_Computer |
| OK-2 | T24 | 15:02 | 403.210 | T24_UTC_12**14**20_Computer |
| MT-2 | T25 | 15:32 | 402.410 | T25_UTC_12**14**20_Computer |
| OK-1 | T26 | 16:32 | 400.010 | T26_UTC_12**14**20_Computer |
| OK-2 | T27 | 17:32 | 401.610 | T27_UTC_12**14**20_Computer |
| OK-1 | T28 | 18:32 | 404.810 | T28_UTC_12**14**20_Computer |
| OK-2 | T29 | 19:32 | 401.610 | T29_UTC_12**14**20_Computer |
| OK-1 | T30 | 20:32 | 404.810 | T30_UTC_12**14**20_Computer |
| OK-2 | T31 | 21:32 | 401.610 | T31_UTC_12**14**20_Computer |
| OK-1 | T32 | 22:32 | 404.810 | T32_UTC_12**14**20_Computer |
| OK-2 | T33 | 23:32 | 401.610 | T33_UTC_12**14**20_Computer |
| OK-1 | T34 | 00:32 | 404.810 | T34_UTC_12**14**20_Computer |
| OK-2 | T35 | 01:32 | 401.610 | T35_UTC_12**14**20_Computer |
| OK-1 | T36 | 02:32 | 404.810 | T36_UTC_12**14**20_Computer |
| | | ***December*** | ***15th*** | ***2020*** |

| | | | | |
|---|---|---|---|---|
| OK-2 | T37 | 03:32 | 401.610 | T37_UTC_12**15**20_Computer |
| OK-1 | T38 | 04:32 | 404.810 | T38_UTC_12**15**20_Computer |
| OK-2 | T39 | 05:32 | 401.610 | T39_UTC_12**15**20_Computer |
| OK-1 | T40 | 06:32 | 404.810 | T40_UTC_12**15**20_Computer |
| OK-2 | T41 | 07:32 | 401.610 | T41_UTC_12**15**20_Computer |
| OK-1 | T42 | 08:32 | 404.810 | T42_UTC_12**15**20_Computer |
| OK-2 | T43 | 09:32 | 401.610 | T43_UTC_12**15**20_Computer |
| OK-1 | T44 | 10:32 | 404.810 | T44_UTC_12**15**20_Computer |
| OK-2 | T45 | 11:32 | 401.610 | T45_UTC_12**15**20_Computer |
| OK-1 | T46 | 12:32 | 404.810 | T46_UTC_12**15**20_Computer |
| OK-2 | T47 | 13:32 | 401.610 | T47_UTC_12**15**20_Computer |
| OK-1 | T48 | 14:32 | 404.810 | T48_UTC_12**15**20_Computer |
| OK-2 | T49 | 15:32 | 401.610 | T49_UTC_12**15**20_Computer |
| OK-1 | T50 | 16:32 | 404.810 | T50_UTC_12**15**20_Computer |

# Appendix G - Windshear and convective instabilities code

```
import numpy as np  # Numbers (like pi) and math
import matplotlib.pyplot as plt  # Easy plotting
import pandas as pd  # Convenient data formatting, and who doesn't want pandas
from numpy.core.defchararray import lower  # For some reason I had to import this separately
import os  # File reading and input
from io import StringIO  # Used to run strings through input/output functions
from scipy import interpolate  # Used for PBL calculations
import pywt  # Library PyWavelets, for wavelet transforms
from scipy.signal import argrelextrema
from array import *
from sympy import *

########## Function definitions, to be used later ##########

#The Ri Method looks for a Richardson Number between 0 and 0.25
#Between these two values there is a fluid instability (Kelvin-Helmzbolt) that can contribute to
#the creation of waves


def ri_shear(vpt, vt, u, v, hi,showPlots):
    g = 9.81
    gd_vpt = np.gradient(vpt, hi) #gradient of virtual potential temperature with height
    gd_u = np.gradient(u, hi) #gradient of wind shear "u" with height
    gd_v = np.gradient(v, hi) #gradient of wind shear "v" with height

    ri = ((g/vt) * gd_vpt) / (((gd_u)**2) + ((gd_v)**2) ) #Gradient RI equation

    ri_final = [hi for hi, ri in zip(hi, ri) if ri >= 0 and ri <= 0.25] #Ties height to to the ri calculated in the above equation
    # and returns ri's between the critical values



    if showPlots:
        plt.plot(ri, hi, '.') # plots calculated RI against the assoicated height
        plt.axvline(0.25, 0, 1, color = 'red') # plots line at critical value
        plt.axvline(0, 0, 1, color = 'red') # plots line at critical value
        plt.xlabel("Richardson Number")
        plt.ylabel("Height in Meters")
        plt.title("Gradient Richardson Number for Wind Shear", fontsize = 18)
        plt.xlim(-0.2, 0.30)
        plt.ylim(8000, 35000)
        plt.show()

    critical_list = []


    for i in ri_final:
        if i >= 10000:
            critical_list.append(i)

```

```
    criticalAlts = pd.DataFrame(critical_list)          #create dataframe to access values easier from critical list
    k=0                                                 #Start counter
    wsAlts = []                                         #Create list to append the "new" values in the loop
    while k < len(criticalAlts):                        #While count is below the size of Critical Altitude list
        if k == 0 :                                     #Initialize starting altitude on first run
            alt = criticalAlts.loc[k]                   #First altitude is first critical altitude value (grabbing the count as index position)
            wsAlts.append(int(alt))
            k=k+1
        else:                                           #If NOT FIRST RUN
            lastAlt = alt                               #Grab last "alt" value and make lastAlt to compare new alt to on each loop
            currentAlt = criticalAlts.loc[k]            #Grab new altitude for position of loop
            deltaAlt = int(currentAlt - lastAlt)        #Compare New altitude vs last runs altitude (make int for boolean in next line)
            if deltaAlt >=1000:                         #If change in Alt is BIGGER than 1000 meters
                alt = currentAlt                        #Make "alt" the current altitude
                wsAlts.append(int(alt))                 #Append the Wind Shear Altitudes list with the NEW altitude value
            k=k+1
    # print("Wind shear altitudes")
    # print(wsAlts)
    # print(wsAlts, file=open("ws_"+file, "a"))


    return wsAlts # returns list of heights at which ri is critical, ignore noise from lower atmosphere where windshear is common



def ri_conv(vpt, vt, u, v, hi,showPlots):
    g = 9.81
    gd_vpt = np.gradient(vpt, hi) #gradient of virtual potential temperature with height
    gd_u = np.gradient(u, hi) #gradient of wind shear "u" with height
    gd_v = np.gradient(v, hi) #gradient of wind shear "v" with height

    ri = ((g/vt) * gd_vpt) / (((gd_u)**2) + ((gd_v)**2) ) #Gradient RI equation
    conv_final = [hi for hi, ri in zip(hi, ri) if ri <= 0] #Ties height to to the ri calculated in the above equation
    # and returns ri's between the critical values



    if showPlots:
        plt.plot(ri, hi, '.') # plots calculated RI against the assoicated height
        plt.axvline(0, 0, 1, color = 'red') # plots line at critical value of zero
        plt.xlabel("Richardson Number")
        plt.ylabel("Height in Meters")
        plt.title("Gradient Richardson Number for Convection", fontsize = 18)
        plt.xlim(-5, 0.30)
        plt.ylim(5000, 35000)
        plt.show()

    convective_list = []


    for i in conv_final:
        if i >= 5000:
            convective_list.append(i)
```

```
    convectiveAlts = pd.DataFrame(convective_list)   #create dataframe to access values easier from convective list
    k=0                                               #Start counter
    cvAlts = []                                       #Create list to append the "new" values in the loop
    while k < len(convectiveAlts):                     #While count is below the size of Convective Altitude list
        if k == 0 :                                   #Initialize starting altitude on first run
            alt = convectiveAlts.loc[k]                #First altitude is first convective altitude value (grabbing the count as index position)
            cvAlts.append(int(alt))
            k=k+1
        else:                                         #If NOT FIRST RUN
            lastAlt = alt                             #Grab last "alt" value and make lastAlt to compare new alt to on each loop
            currentAlt = convectiveAlts.loc[k]          #Grab new altitude for position of loop
            deltaAlt = int(currentAlt - lastAlt)      #Compare New altitude vs last runs altitude (make int for boolean in next line)
            if deltaAlt >=1000:                        #If change in Alt is BIGGER than 200 meters
                alt = currentAlt                      #Make "alt" the current altitude
                cvAlts.append(int(alt))               #Append the Convective Altitudes list with the NEW altitude value
            k=k+1
    # print("Convective altitudes")
    # print(cvAlts)
    # print(cvAlts, file=open("cv_"+file, "a"))


    return cvAlts # returns list of heights at which ri is critical, ignore noise from lower atmosphere where windshear is common


 #returns just the list of heights found to meet the requirements for a kelvin-helmzbolt instability

#NOTE: This method simply gives heights at which there MIGHT be an instabilty, it is possible to have a significant Ri
# number with no activity due to other factors in the atmosphere


# The "Wind Shear" method, this method looks for areas of local extrema for wind shear and then looks another 200 indices ahead
# (Around 1000m - with a rise rate of 5m/s) for that wind shear. If there is a difference of 7.5m/s or more it is
# considered a significant windshear and meets the criteria for instability to create waves

# This method has been used in previous MSGC campaigns but is rarely referenced, and the only support for this method
# was found in the following paper (google title and author for PDF download)
# How Vertical Wind Shear Affects Tropical Cyclone Intensity Change: An Overview by Levi Thatcher and Zhaoxia Pu (University of Utah)
# the author of this paper was emailed to ask where they had gotten the measurement of 7.5 and stated it was
# the work of a student researcher and is not a "hard line" but rather a reference point


def windshear_max(hi, ws):
    lmx = np.array(argrelextrema(ws, np.greater)) #looks for local maximums in windshear values
    lmx_final = lmx[lmx <= (len(ws) - 506)] #cuts out 200 indices, so that later when we add 200, its not above the array

    ws_upper = ws[lmx_final + 506] #adds 200 indices of windshears to the local maximums
    ws_lower = ws[lmx_final] #pulls the windshears with local maximums

    ws_math = ws_upper - ws_lower #takes the difference of 200 incides of windshear

    hi_final = [hi for hi, ws in zip(hi[lmx_final], ws_math) if abs(ws) >= 7.5] #ties the indices of height with indices
    # of significant windshears (different of or greater than 7.5) so we can list just the heights at which these
    # shears are occuring
    # print("windshearMax")
    # print(hi_final)
    return hi final #returns the height of significant windshear
```

```
def windshear_min(hi, ws):
    lmn = np.array(argrelextrema(ws, np.less)) #looks for local minimum in windshear values
    lmn_final = lmn[lmn <= (len(ws) - 506)] #cuts out 200 indices, so that later when we add 200, its not above the array

    ws_upper = ws[lmn_final + 506] #adds 200 indices of windshears to the local minimums
    ws_lower = ws[lmn_final] #pulls the windshears with local minimums

    ws_math = ws_upper - ws_lower #takes the difference of 200 incides of windshear

    hi_final = [hi for hi, ws in zip(hi[lmn_final], ws_math) if abs(ws) >= 7.5] #ties the indices of height with indices
    # of significant windshears (different of or greater than 7.5) so we can list just the heights at which these
    # shears are occuring
    # print("windshearMin")
    # print(hi_final)
    return hi_final #returns the height of significant windshear


########## USER INPUT SECTION ##########
def getUserInputFile(prompt):
    print(prompt)
    userInput = ""
    while not userInput:
        userInput = input()
        if not os.path.isdir(userInput):
            print("Please enter a valid directory:")
            userInput = ""
    return userInput


def getUserInputTF(prompt):
    print(prompt+" (Y/N)")
    userInput = ""
    while not userInput:
        userInput = input()
        if lower(userInput) != "y" and lower(userInput) != "n":
            userInput = ""
    if lower(userInput) == "y":
        return True
    else:
        return False



#freq_val = input('What is the Brut Vasla Frequency You would like to investigate?:')
dataSource = getUserInputFile("Enter path to data input directory: ")
showPlots = getUserInputTF("Do you want to display plots for analysis?")
saveData = getUserInputTF("Do you want to save the output data?")
if saveData:
    savePath = getUserInputFile("Enter path to data output directory: ")
else:
    savePath = "NA"
# MATLAB code has lower and upper altitude cut-offs and latitude
# I've changed these to be read in from the data

```

```
228     # For debugging, print results
229     print("Running with the following parameters:")
230     print("Path to input data: /"+dataSource+"/")
231     print("Display plots: "+str(showPlots))
232     print("Save data: "+str(saveData))
233     print("Path to output data: "+savePath+"\n")
234
235     ########## FILE RETRIEVAL SECTION ##########
236
237     # Need to find all txt files in dataSource directory and iterate over them
238
239     # However, I also want to check the GRAWMET software to see if it can output
240     # the profile in either a JSON or CSV file format, as that would likely be
241     # much easier.
242
243
244     for file in os.listdir(dataSource):
245         if file.endswith(".txt"):
246             saveName = (file.split(".",2))[0]
247             #Used to fix a file reading error
248             contents = ""
249             #Check to see if this is a GRAWMET profile
250             isProfile = False
251             f = open(os.path.join(dataSource, file), 'r')
252             print("\nOpening file "+file+":")
253             catch1 = f
254             for line in f:
255                 if line.rstrip() == "Profile Data:":
256                     isProfile = True
257                     contents = f.read()
258                     catch2= contents
259                     print("File contains GRAWMET profile data")
260                     break
261             f.close()
262             if not isProfile:
263                 print("File "+file+" is either not a GRAWMET profile, or is corrupted.")
264
265             if isProfile:  # Read in the data and perform analysis
266
267                 # Fix a format that causes a table reading error
268                 contents = contents.replace("Virt. Temp", "Virt.Temp")
269                 contents = contents.split("\n")
270                 contents.pop(1)  # Remove units from temp file
271                 index = -1
272                 for i in range(0, len(contents)):  # Find beginning of footer
273                     if contents[i].strip() == "Tropopauses:":
274                         index = i
275                 if index >= 0:  # Remove footer, if found
276                     contents = contents[:index]
277                 contents = "\n".join(contents)  # Reassemble string
278                 del index
279
280                 # Read in the data
281                 print("Constructing a data frame")
282                 data = pd.read_csv(StringIO(contents), delim_whitespace=True)
283                 del contents
```

```
            # Find the end of usable data
            badRows = []
            for row in range(data.shape[0]):
                if not str(data['Rs'].loc[row]).replace('.', '', 1).isdigit():  # Check for nonnumeric or negative rise rate
                    badRows.append(row)
                elif row > 0 and np.diff(data['Alt'])[row-1] <= 0:
                    badRows.append(row)
                else:
                    for col in range(data.shape[1]):
                        if data.iloc[row, col] == 999999.0:  # This value appears a lot and is obviously wrong
                            badRows.append(row)
                            break
            if len(badRows) > 0:
                print("Dropping "+str(len(badRows))+" rows containing unusable data")
            data = data.drop(data.index[badRows])

            ########## PERFORMING ANALYSIS ##########



            hi = np.array(data['Alt'] - data['Alt'][0])  # height above ground in meters

            Alt = np.array(data['Alt']) # Raw altitude data from profile

            epsilon = 0.622  # epsilon, unitless constant

            ws = np.array(data['Ws']) # Raw ws data from profile

            virtcon = 0.61 # Constant value used for Virtual Potential Temperature

            # vapor pressure
            dwPt = data['Dewp.'] # Raw dewpoint data from profile

            e = np.exp(1.8096+(17.269425*dwPt)/(237.3+dwPt)) # Used in water vapor mixing ratio, in totally arid conditions a constant can be used

            # water vapor mixing ratio
            rvv = (epsilon * e) / (data['P'] - e)  # unitless

            # potential temperature
            pot = (1000.0 ** 0.286) * (data['T'] + 273.15) / (data['P'] ** 0.286)  # kelvin

            # virtual potential temperature
            vpt = pot * (1 + (virtcon * rvv))  # kelvin

            # absolute virtual temperature
            vt = (data['T'] + 273.15) * ((1 + (rvv / epsilon)) / (1 + rvv))  # kelvin

            # u and v (east & north?) components of wind speed
            u = -data['Ws'] * np.sin(data['Wd'] * np.pi / 180) #Breaks wind speed values into directions
            v = -data['Ws'] * np.cos(data['Wd'] * np.pi / 180) #Breaks wind speed values into directions

```

```
            # call the functions from the first section that you want to use


            HeightWindRichardson = pd.Series(ri_shear(vpt, vt, u, v, hi,showPlots))
            HeightConvectiveRI = pd.Series(ri_conv(vpt, vt, u, v, hi,showPlots))
            HeightWindMethodOne = pd.Series(windshear_max(hi, ws))
            HeightWindMethodTwo = pd.Series(windshear_min(hi, ws))

            headers = ['WindshearAlt','ConvectiveAlt','Wind_MAX','Wind_MIN']
            Results = (pd.concat([HeightWindRichardson,HeightConvectiveRI,HeightWindMethodOne,HeightWindMethodTwo],
                                 ignore_index=True,axis=1))
            Results.columns=headers
            Results = Results.fillna('-')

            print(Results)

            if saveData:
                npResults = Results.to_numpy()
                npResults = np.insert(npResults,[0],headers,axis=0)
                np.savetxt("%s/WindShearResults_%s.txt"%(savePath,saveName),npResults,fmt='%s',delimiter='\t\t')
                del npResults
                # Results.to_csv("%s/WindShearResults_%s.txt" %(savePath,saveName),header=True,index=None,sep=str("\t"))
                #I am working on a fix to make them all spaced evenly in the text file



           # print("Heights of Significant Wind Shear: " + str(HeightWindMethodOne) + str(HeightWindMethodTwo))
           #I have choosen to only print the two methods due to noise in the RI method, heights of unstable RI can be
           # determined by the graph that it produces and looking for areas of consistently critical RI numbers.


            ########## FINISHED ANALYSIS ##########

            print("Finished analysis.")

print("\nAnalyzed all .txt files in folder /"+dataSource+"/")
```