2018-01-01

Implementing Large Eddy Simulation To Numerical Simulation Of Optical Wave Propagation

Diego Alberto Lozano Jimenez
University of Texas at El Paso, dalozano4@outlook.com

IMPLEMENTING LARGE EDDY SIMULATION TO NUMERICAL SIMULATION OF OPTICAL WAVE PROPAGATION

DIEGO ALBERTO LOZANO JIMENEZ

Master's Program in Mechanical Engineering

APPROVED:

Vinod Kumar, Ph.D., Chair

V.S. Rao Gudimetla, Ph.D., Co-Chair

Arturo Bronson, Ph.D.

Heidi Taboada-Jimenez, Ph.D.

V. M. Krushnarao Kotteda., Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

IMPLEMENTING LARGE EDDY SIMULATION TO NUMERICAL SIMULATION OF OPTICAL WAVE PROPAGATION

by

DIEGO ALBERTO LOZANO JIMENEZ, BSME

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2018

Acknowledgements

This work supported by Air Force Office of Scientific Research (AFOSR) under Grant Number BAA-AFRL-AFOSR-2016-0007, and Department of Mechanical Engineering and Computational Science Program at University of Texas at El Paso. Simulations were running on XSEDE computational resources such as Bridges, Stampede, and Lonestar. I would also like to thank Dr. Stacie Williams, AFOSR Program Officer, for her hard work that helped provide the grant. Plane wave simulation and study was conducted at AFRL in Maui with Dr. Rao Gudimetla who was funded in part by an AFOSR Grant. I also want to give a special thanks to Dr. Rao Gudimetla for his patience and support throughout my study. None of this work could not have been possible without the help from Dr. Vinod Kumar and Dr. V. M. Krushnarao Kotteda.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

There is a growing interest in optical systems capable of propagating lasers over long distances in atmospheric conditions for a variety of applications. Improvement and understanding of laser propagation over a long path can have influences on the performance of long-range communication systems, active imaging, laser countermeasures, target tracking, and other related optical systems. The challenge is being able to predict atmospheric turbulence induced effects on scintillations, beam spreading, phase fluctuations, beam wander and jitter that can degrade and limit the performance of imaging and laser systems. The fluctuations are caused by temperature variations in the atmosphere associated with turbulent eddies. Atmospheric turbulence behavior is chaotic, and it is impossible to exactly determine the specific value of an optical parameter at all positions in time and space. That is why atmospheric turbulence-induced effects are modeled by numerical and theoretical statistical models to describe the random behavior.

In this study, we want to simulate long-range laser propagation in atmospheric turbulence. The numerical simulations are carried out to study the impact of strong atmospheric turbulence in spatial, temporal, and related spectral domains. The first section of this study will be concerned with modeling this numerical simulation in Kolmogorov and non-Kolmogorov spectrum. To validate our numerical simulation, we will compare the statistical parameter to theoretical approximation in both Kolmogorov and non-Kolmogorov spectrums. Once the code is validated, we want to integrate Large Eddy

Simulation (LES) turbulence modeling. LES simulations allowa us to study strong fluid turbulence and can predict advection-dissipation at various energy spectrums. Therefore, this this choice of modeling allows us to characterize turbulence at outer and inner regimes important for correcting the effects of optical turbulence over long-distance laser beam propagation. We conduct Computational Fluid Dynamics (CFD) simulations of atmospheric turbulence to realistic three-dimensional systems that better represent the geography of the summits between Maui and Hawaii, where long-range laser propagation measurements were conducted.

LES simulations are computationally expensive and need high wall time and computational power. To overcome that, we implement a conditional generative adversarial network (cGAN) that applies phase unwrapping to a wrapped phase screen. The cGAN framework establishes competition between two distinct players in the model, the discriminator and the generator. The network is trained on 512x512 wrapped and unwrapped images and learns the mapping to conduct phase unwrapping. This portion of the study focuses on employing neural networks to improve the accuracy of current simulation. The focus of this study is to accurately simulate laser propagation through the atmosphere r via ourknowledge of fluid mechanics and machine learning.

Chapter 2: The Scales of Turbulent Motion

When instabilities are introduced in ambient wind fields, eddies of varying sizes are created. The larger eddies will begin to subdivide onto smaller eddies until their energy is dissipated as heat. Energy dissipation is proportional to the kinematic viscosity, which is related to Reynolds number, Re, which describes the type of fluid flow [1]. To comprehend the complexity of atmospheric turbulence we analyze its behavior through incompressible Navier Stokes equations in nondimensional form.

$$\nabla u = 0 \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u \cdot \nabla u_i = -\nabla p + \frac{1}{Re}\Delta u + f_j \tag{2.2}$$

Reynolds number, $Re = \frac{UL}{\vartheta}$, is the only non-dimensional parameter in the Navier Stokes equations if the dimensionless body force f_j is absent. In case of atmospheric turbulence, Reynolds number lies between 10^6 and 10^7. This means that ϑ, the kinematic viscosity of the medium is relatively small and the flow is non-linear and advection dominated [2]. This results in producing approximately 10^{14} to 10^{16} nonlinear ordinary differential equations and they must be solved numerically. Implementing Computational Fluid Dynamics (CFD) would be ideal but the flow is very nonlinear, and the computational power needed to model the large geometry associated with long distance propagation is not feasible for our current computers. Therefore, atmospheric turbulence-induced effects in adaptive optics are modeled by numerical and theoretical

statistical models to describe its behavior. To characterize atmospheric turbulence, it is common to analyze the effects of turbulent eddies by modeling the random behaviors through the Kolmogorov statistical approach.

2.1 Kolmogorov Turbulence

The kolmogorov model states that the kinetic energy of a turbulent flow is transmitted within an inertial sub rage called the energy cascade theory [1]. The inertial subrange begins with eddies the size equal to that of the outer scale length L_o.The input energy is then progressively divided and redistributed into smaller eddies until the eddies are equal in size to the inner scale length l_o (Figure 1.1). Anything below this length is dissipated into heat. Eddies within this inertial range are assumed to be statistically homogeneous and isotropic.

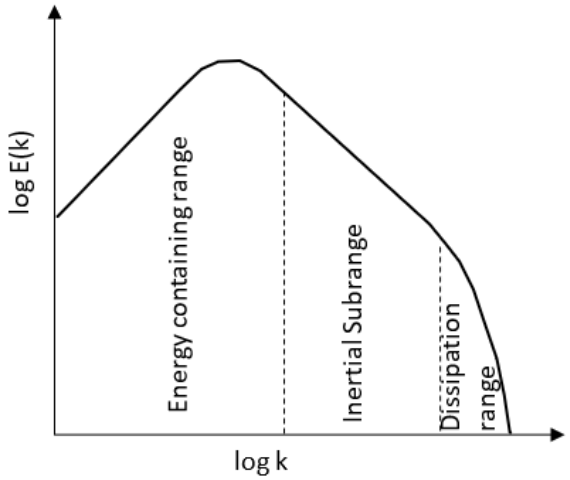


Figure 2.1: Description of Kolmogorov energy cascade theory

In this inertial subrange of local homogeneity, the Kolmogorov theory states that the time variation of two different velocities at two varying points statistically depend only on the displacement vector, r, between them [3]. This simplifies our analysis by implying that the magnitude of the displacement vector is most critical variable when considering the velocity

difference. The assumption led to the derivation of the structure-function which is the structure-function of wind velocity component parallel to the displacement.

$$D_v(r) = \langle [v_r(r_1 + r) - v_r(r_1)]^2 \rangle \tag{2.3}$$

If the separation displacement, r, is within the inertial subrange of turbulence, the velocity structure function follows the form

$$D_v(r) = C_v^2 r^{2/3} \quad l_o < r < L_o \tag{2.4}$$

C_v^2 is the velocity structure constant, which is a measure of the total amount of energy in the turbulent flow. Between the bounds of the inner and outer scale, the eddies may be considered isotropic and homogenous.

Tatarskii related the turbulence model to the index of refractivity through the concept of conservative passive additives. It can be shown [4], [5], that the refractive index has the same statistical properties as and obey the same structure-function laws as velocity and potential temperature.

$$D_n(r) = C_n^2 r^{2/3} = 8\pi \int_0^\infty \kappa^2 \Phi_n(\kappa) \left(1 - \frac{\sin(\kappa r)}{\kappa r}\right) d\kappa. \tag{2.5}$$

Now that turbulence structure function has been analytically derived we can use it to determine Kolmogorov's power spectral density function [6]. One can calculate the power spectral density given the structure-function in terms of the spectral density and inverting the equation to solve for the spectral density.

$$\phi_n(\kappa) = \frac{1}{4\pi^2\kappa^2}\int_0^\infty \frac{sin(\kappa r)}{\kappa r}\frac{d}{dr}\left[r^2\frac{d}{dr}D_n(r)\right]dr \tag{2.6}$$

κ is the angular spatial frequency in rad/m and is equal to $2\pi/\lambda$. Then by substituting equation (2.5) onto equation (2.6), we can obtain the spectral density function in terms of the Kolmogorov's inertial subrange.

$$\phi_n(\kappa) = 0.033C_n^2\kappa^{-11/3} \quad \frac{2\pi}{L_o} < \kappa < \frac{2\pi}{l_o} \tag{2.7}$$

$\phi_n(\kappa)$ is the Kolmogorov's power spectral density as a function of the spatial wavenumber κ, which is valid only in the inertial subrange. The probability density function is the probability measure associated with the atmospheric turbulence. Because there have been observations of atmospheric turbulence showing behavior that deviate from the Kolmogorov theory, it has become necessary to develop a statistical model that better represents the effects of atmospheric turbulence.

2.1.1 The Effects of Turbulence on Optical Propagation

Now that we have characterized the atmospheric induced effects on the refractive index, we can look at a propagation of a flat wavefront through turbulence. The phase shift produced by the refractive index fluctuation at an atmospheric layer thickness $\boldsymbol{\delta h}$ at a height $\boldsymbol{h}$ is

$$\varphi(x) = k\int_h^{h+\boldsymbol{\delta h}} dz\, n(x,z). \tag{2.8}$$

Other independent variables can affect the phase shift other than the refractive index, for layers thicker than the individual turbulent cell. That is why the Central Limit Theorem is implied, meaning that the phase shifts have a Gaussian statistic. Given the statistical properties of the refractive index, we can proceed to derive the statistical behavior of a wavefront, $\boldsymbol{U = e^{i\varphi(x)}}$. In order to do this, we need to express the coherence function $\boldsymbol{B_h(\rho)}$ in terms of the phase structure function [7].

$$B_h(r) = \langle U(x)U^*(x+r)\rangle$$

$$B_h(r) = \langle exp(i[\varphi(x) - \varphi(x+r)])\rangle \tag{2.9}$$

Since we have stated previously that we have implemented the Central Limit Theorem, $[\boldsymbol{\varphi(x) - \varphi(x+r)}]$, has Gaussian statistics with zero mean. We can then apply the relation $\langle \boldsymbol{exp(\alpha X)}\rangle = \boldsymbol{exp\left(\frac{1}{2}\alpha^2\langle X^2\rangle\right)}$.

$$B_h(\rho) = exp\left(-\frac{1}{2}\langle|\varphi(x) - \varphi(x+r)|^2\rangle\right)$$

$$B_h(\rho) = exp\left(-\frac{1}{2}D_\varphi(r)\right) \tag{2.10}$$

Due to the chaotic behavior of atmospheric turbulence, random processes like velocity, or temperature cannot be approximated as being stationary. We can still statistically analyze these random processes if they have stationary increments. Instead of

focusing on the random process over time or a point in space $x(r)$, we focus on the function $x(r + r_1) - x(r_1)$, which behaves like a stationary process [8].

The random process that has stationary increments and that can be described by such function are better analyzed through the phase structure function. It is customary when studying atmospheric turbulence to express a random process as a sum.

$$x(r) = x(r) + x_1(r) \tag{2.11}$$

The mean phase of our signal is defined by $m(r)$, and $x_1(r)$ is the stationary process with moments that are invariant under translations in space $\langle x_1(r) \rangle = 0$. The structure function for a random process can be defined by

$$D_x(r_1, r_2) = [m(r_1) - m(r_2)]^2 + \langle [x_1(r_1) - x_1(r_2)]^2 \rangle. \tag{2.12}$$

Previously we established that the random process has a slow varying mean, meaning that the difference in means is nearly zero. We can then approximate the structure-function to be

$$D_x(r_1, r_2) \cong \langle [x_1(r_1) - x_1(r_2)]^2 \rangle. \tag{2.13}$$

Let's consider a random process such that its ensemble average of $x(r + \rho) - x(r)$ is independent of displacement, r. Then it would be true that the ensemble average of $[x(r + \rho) - x(r)]^2$ is independent of displacement. We can then call this random process to

have stationary increments and can be characterized by the structure function. However, a stationary process can be considered a process with stationary increments in some special For instance, the phase fluctuations are a stationary process and its structure function is closely related to its covariance B_φ.

$$D_\varphi(\rho) = \langle(\varphi_1 - \varphi_2)^2\rangle = 2[\langle\varphi^2\rangle - \langle\varphi_1\varphi_2\rangle] = 2\big[\mathrm{B}_\varphi(0) - \mathrm{B}_\varphi(\rho)\big] \tag{2.14}$$

The covariance function is defined by the ensemble average.

$$\mathrm{B}_\varphi(\rho) = \langle\varphi(r)\varphi(r+\rho)\rangle \tag{2.15}$$

For an atmospheric layer thickness, δh, larger than the correlation scale of the fluctuations we can determine the phase covariance at an extended bound of $-\infty$ and ∞. This allows us to relate the refractive index variations $\mathrm{B}_n(\rho, z)$ to the phase covariance.

$$\mathrm{B}_\varphi(\rho) = k^2\delta h \int_{-\infty}^{\infty} dz\, \mathrm{B}_n(\rho, z). \tag{2.16}$$

Given both equations (2.14) and (2.16) we can derive an analytical approximation of the phase structure function in terms of the refractive index structure constant, $\boldsymbol{C_n^2}$.

$$D_\varphi(\rho) = 2\big[\mathrm{B}_\varphi(0) - \mathrm{B}_\varphi(\rho)\big] \tag{2.17}$$

$$= 2k^2\delta h \int_{-\infty}^{\infty} dz\, [\mathrm{B}_n(0,z) - \mathrm{B}_n(\rho,z)]$$

$$= 2k^2\delta h \int_{-\infty}^{\infty} dz\, \left[\left(\mathrm{B}_n(0,0) - \mathrm{B}_n(\rho,z)\right) - \left(\mathrm{B}_n(0,0) - \mathrm{B}_n(0,z)\right)\right]$$

$$= k^2\delta h \int_{-\infty}^{\infty} dz\, [D_n(\rho,z) - \mathrm{D}_n(0,z)]$$

Inserting equation (2.5) and integrative over the bounds gives

$$D_\varphi(\rho) = 2.914k^2\delta h C_n^2 \rho^{5/3}. \tag{2.18}$$

Now that we have the desired expression of the phase structure function in Kolmogorov turbulence, we can proceed to derive the coherence function.

$$B_h(\rho) = exp\left(-\frac{1}{2}\left[2.914k^2\delta h C_n^2 \rho^{5/3}\right]\right) \tag{2.19}$$

We need to consider that in adaptive optics most the time we are not looking vertically into the sky and need to take into account the zenith angle ***z***. The expression can then be integrated over the whole atmosphere.

$$B_h(\rho) = exp\left(-\frac{1}{2}\left[2.914k^2 \sec(z)\, \rho^{5/3} \int \delta h C_n^2(h)\right]\right) \tag{2.20}$$

In order to simplify the expression, we can introduce the Fried Parameter

$$r_o = \left[0.423k^2 \sec(z) \int \delta h C_n^2(h)\right]^{-3/5} \tag{2.21}$$

The coherence function and structure function can be simplified by now implementing Fried Parameter.

$$B_h(\rho) = exp\left[-3.44\left(\frac{\rho}{r_o}\right)^{5/3}\right] \quad , \quad D_\varphi(\rho) = 6.88\left(\frac{\rho}{r_o}\right)^{5/3} \tag{2.22}$$

Both the coherence function and the phase structure function both depend on the Fried Parameter which is a function of turbulence strength, zenith angle, and wavelength [7]. This is the importance of determining the phase structure function in our simulation. It will be used to determine a complex parameter that will characterize the effects of atmospheric turbulence on laser propagation such as the Fried parameter. As an example, observations with telescopes smaller than $\boldsymbol{r_o}$, are essentially diffraction-limited. Telescopes much larger than $\boldsymbol{r_o}$, are seeing-limited. More complex parameters are far off the score of this study as this is only an example to show the importance of the phase structure function.

2.2 NON-KOLMOGOROV TURBULENCE

Current analysis of atmospheric turbulence effects is based on theoretical and statistical models [3], [5], [9] derived from the Kolmogorov approach. Supporting experimental evidence [1], [10], [11] helped validate the classical theory of optical propagation. This is the reason Kolmogorov's power spectrum of refractive index fluctuation is widely employed in the study of optical wave propagation in the atmosphere. However, inaccuracies develop in the Kolmogorov model at various

conditions in atmospheric turbulence, which shows behavior that deviates from the theory [12]–[15]. Further investigation is needed to comprehend laser propagation in atmospheric turbulence that does not obey the Kolmogorov statistics.

The beta-model incorporates space-filling concepts for the turbulent eddies in the inertial ranges using fractal descriptions for the eddies. That is why we need to understand non-Kolmogorov descriptions of energy cascade theory and comprehend if it is better suited to model laser propagation over long distances. The spectral behavior of the refractive index fluctuations changes with altitude, which makes the Kolmogorov model inadequate for predicting atmospheric behaviors. Instead, we will implement a 3D power spectrum of refractive index fluctuations in terms of the arbitrary power law [16], which can be defined as

$$\phi_n(\kappa, \alpha) = a(\alpha)\beta\kappa^{-\alpha} \tag{2.23}$$

where κ is the spatial wavenumber, β is the general refractive index structure constant is like C_n^2 and has units of $m^{3-\alpha}$. $a(\alpha)$ is the function that maintains consistency between the index structure function and its power spectrum. To be able to use the 3D power spectrum, we need to determine the solution of $a(\alpha)$. Let's consider a structure function that describes a random media with an index of refraction fluctuations with the form of a power law [17].

$$D_n(r) = \beta r^{\gamma} \tag{2.24}$$

By using the relation derived from Tatarski [5], which relates the index power spectrum to the derivative of the structure-function, we can determine the power spectrum in terms of the arbitrary power law [18]. The structure function of the form of a power law is implemented onto equation (2.6) to derive the power spectrum of the turbulence in terms of the arbitrary power law

$$\phi_n(\kappa,\alpha) = 2^{\alpha-6}\beta(\alpha^2-5\alpha+6)\pi^{-2/3}\frac{\Gamma\left[\frac{\alpha-2}{2}\right]}{\Gamma\left[\frac{5-\alpha}{2}\right]}\kappa^{-\alpha} \tag{2.25}$$

Γ is Euler's gamma function and $a(\alpha)$ can be determine by equating equation (2.23) and equation (2.25) and algebraically solving for $a(\alpha)$.

$$(\alpha) = 2^{\alpha-6}(\alpha^2-5\alpha+6)\pi^{-2/3}\frac{\Gamma\left[\frac{\alpha-2}{2}\right]}{\Gamma\left[\frac{5-\alpha}{2}\right]} \tag{2.26}$$

Stribling [18] showed that values of α range from $3 < \alpha < 4$ and, when $\alpha = 11/3$, equation 9 reduces to the Kolmogorov power spectrum. When $\alpha \to 3$, values for the consistency function approach zero and the turbulence power spectrum vanishes. For large outer scale turbulence behavior, adjustment to power spectrum as proposed by von Karman can be used and is given by

$$\phi_n^{vK}(\kappa,\alpha) = a(\alpha)\beta(\kappa^2+\kappa_0^2)^{-\alpha/2}. \tag{2.27}$$

where $\kappa_0 = 2\pi/L_0$.

Chapter 3: Numerical Simulation of Laser Propagation in MATLAB

We will begin our simulation by first implementing Kolmogorov and when validated move towards a non-Kolmogorov model to simulate optical wave propagation. For simplicity, we will be propagating a plane wave over 10km to derive statistical parameters that will characterize atmospheric turbulence effects. The simulated laser propagation will have a wavelength of 0.532μm. Levels of turbulence intensity will be represented by a constant Rytov index and will vary through each simulation from a range of $0.1 \leq \sigma_\chi^2 \leq 1.0$. The Rytov index is the log-amplitude variance to describe the strength of scintillations and is related to C_n^2 by

$$\sigma_\chi^2 = 0.312 C_n^2 k^{7/6} L^{11/6} \tag{3.1}$$

The inner scale and outer scale length are 5mm and 10m, respectively. Due to computational constraints, the grid size will be equivalent to 10.24m with a 2048x2048 number of samples, *N*, and a 5mm grid spacing interval, δ. Within each simulation ,50 trials will run, and statistical parameters will be calculated from each trials output phase screen. These statistical parameters will then be averaged by the number of trials to consider the variation within each trial. Propagation will be simulated using Fresnel propagation in a vacuum [19] and introduce atmospheric turbulence effects through the phase screen approach.

$$U(x_2, y_2) = \frac{e^{ik\Delta z}}{i\lambda\Delta z} \iint_{-\infty}^{\infty} U(x_1, y_1) e^{i\frac{k}{2\Delta z}(x_1 x_2 + y_1 y_2)} dx_1 dy_1 \tag{3.2}$$

The phase screen approach introduces atmospheric turbulence by subdividing the propagation length into uniform slabs of distance Δz. Initially within the propagation range $0 < L < \Delta z$, the plane wave will propagate linearly using Fresnel propagation in a vacuum. Atmospheric turbulence effects will be implemented when the plane wave reaches $L = \Delta z$. The plane wave is altered by a turbulent-induced phase screen in the Fourier domain

$$U = U_0 e^{-i\Phi_\phi(\kappa)} \tag{3.3}$$

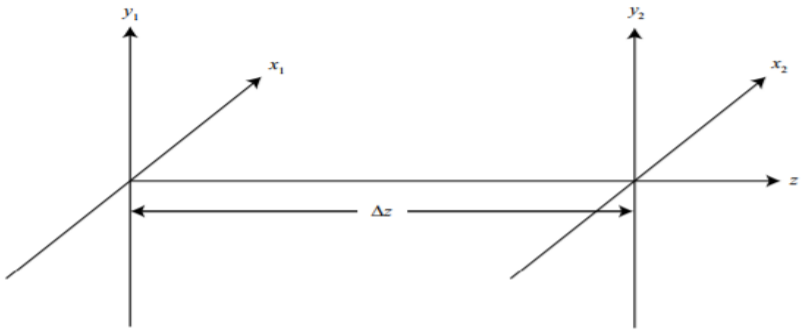


Figure 3.1: Schematic of Laser Propagation between Δz

The process will repeat itself between each slab subsection until the plane wave is transmitted throughout the whole propagation length. Each phase spatial power spectral density (PSD), $\Phi_\phi(\kappa, \alpha)$, can be expressed in terms of its refractive index spectrum, Φ_n^{vk}.

$$\Phi_\phi(\kappa, \alpha) = 2\pi^2 k^2 \Delta z \Phi_n^{vk}(\kappa, \alpha) \tag{3.4}$$

Phase screen simulations are generated through random numbers that are taken from a Gaussian distribution. These values are then implemented into two-dimensional arrays of phase values on a grid of sample points that have the same statistical values as turbulence-induced phase variations. The simulations are conducted using 15 screens over a 10km propagation path.

To derive statistical parameters of the phase fluctuations, the phase needs to be made continuous. Discontinuities are present in the phase at the output signal that wrap around $-\pi < \varphi < \pi$. The built-in MATLAB function can unwrap the phase at laminar flow but shows discontinuities at increased turbulence.

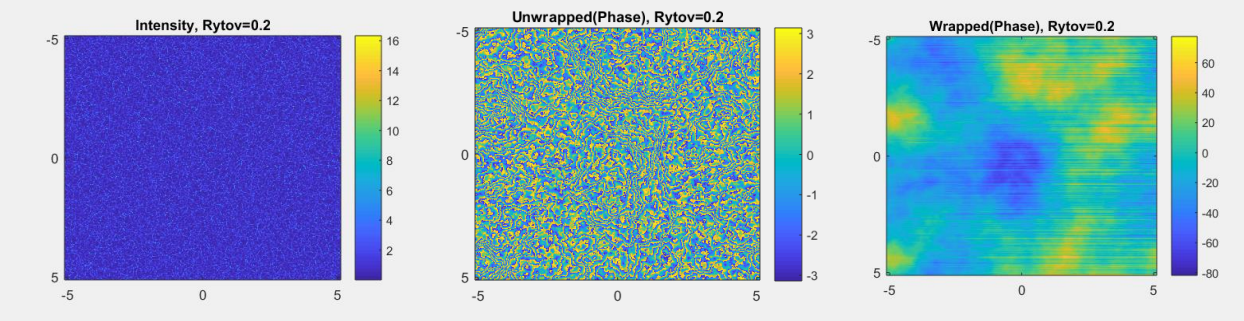


Figure 3.2: Wrapped and Unwrapped Phase Screens using the MATLAB embedded code

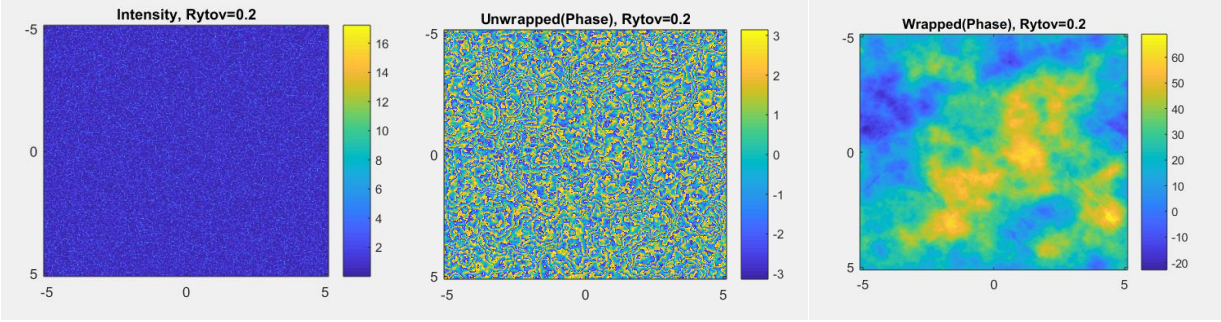


Figure 3.3: Wrapped and Unwrapped Phase Screens using the 2D unwrapping algorithm

A 2D unwrapping algorithm compiled by *Bruce Spottiswoode* [20] and edited by *Carey Smith* is used to make the phase fluctuations continuous. Once the phase is continuous, we are able to calculate statistical parameters like the phase variance. If the simulated phase variance values are validated with the theoretical approximation, we can then proceed to determine the phase structure function that obeys the same structure-function law as the refractive index. For a random process, the covariance is closely related to the autocorrelation by

$$\mathrm{B}_{\varphi}(\rho) = \mathrm{R}_{\varphi}(\rho) - m^2 \tag{3.5}$$

In this case for a random process like the phase shift, can be considered a stationary process the covariance and autocorrelation are identical. Determining the autocorrelation is often too difficult to compute in the spatial domain but is equivalent to simple multiplication in the frequency domain.

$$\mathrm{R}_{\varphi}(\rho) = \mathcal{F}^{-1}\{\mathcal{F}[\varphi(r)]\mathcal{F}[\varphi(r)]^{*}\} \tag{3.6}$$

The notation * is the complex conjugate of the function and $\rho = |r_1 - r_2|$ is the scalar separation. This will allow for a straight forward approach in determining the phase structure function in the 2-D case. The phase structure function is another statistical measure to describe the random behavior of phase values and is closely related to the phase autocorrelation.

$$D_{\varphi}(\rho) = 2\big[\mathrm{R}_{\varphi}(0) - \mathrm{R}_{\varphi}(\rho)\big] \tag{3.7}$$

3.1 STATISTICAL METRICS

To be able to validate our simulation theoretical calculations states that the intensity linearly increases with increase in Rytov index [21]. The theory states that the increase in variance is approximately four times the Rytov value for weak turbulence. range of $0.1 \leq \sigma_{\chi}^{2} \leq 0.5$, the simulated values do agree with theoretical calculations. However, once $\sigma_{\chi}^{2} > 0.5$, simulated values do not agree with the theoretical results and the intensity variance saturates at increased turbulence.

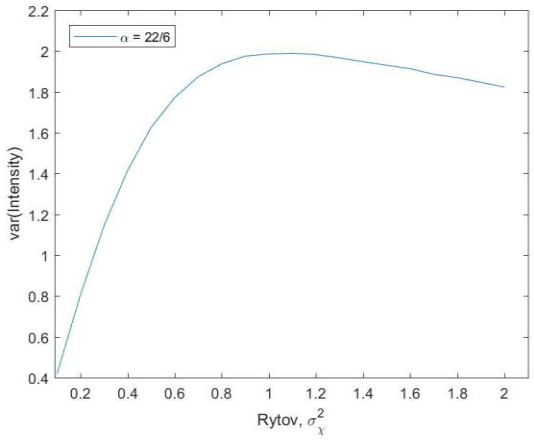


Figure 3.4: Intensity variance in the Kolmogorov spectrum

We wanted to continue our examination by determining the impact of α on the variance of intensity and a new parameter the zero-intensity value. This new parameter is any intensity value less than one percent of the maximum value of intensity, defined as the zero-intensity value.

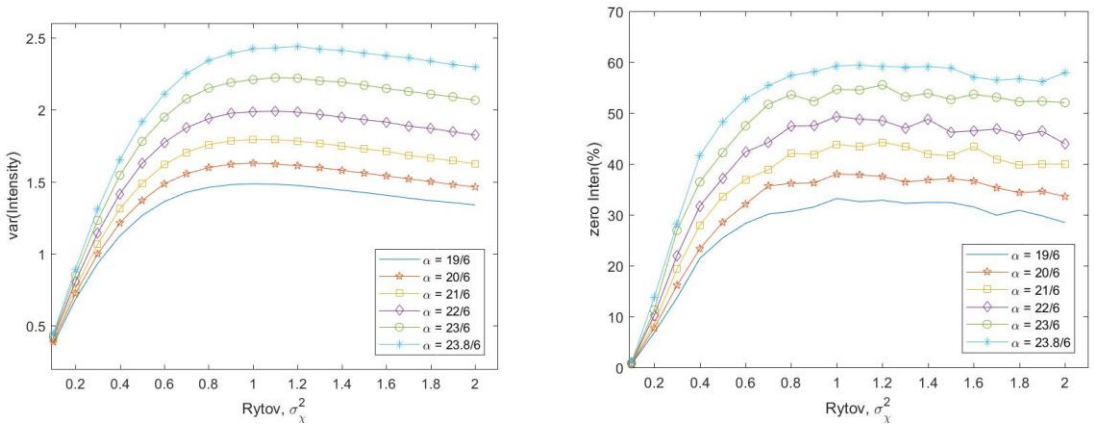


Figure 3.5. Variance of intensity, and number of zeros with respect to Rytov at varying value of alpha

To determine the impact of α on the statistical parameters, phase-screen simulations were performed from a range values of $3 < \alpha < 4$, at varying turbulence levels defined by the Rytov. We notice that the variance of intensity saturates for all values of α, but the point of saturation increases for larger values of α as shown in Figure 6 (Left). Like the variance of intensity, the zero-intensity value saturates for all values of α, but different points for larger values of α as shown in Figure 6 (Right). A noticeable difference between the variance of intensity and the zero-intensity values is that the zero-intensity values appear to fluctuate at different values of α and Rytov. This portion of the study was only to verify if our current numerical simulation is correct. This study mainly focuses on the phase fluctuations at the receiver plane.

Now that we verified our code, we can determine the phase variance at the receiver plane. The phase variance depends on the first moment of the spectrum, meaning that it is determined mainly by the largest eddies. This makes the phase variance sensitive to the small-wavenumber portion of the spectrum of irregularities. Therefore, we modeled our simulation by the von Karman PSD to attempt to represent large eddy irregularities [1]. As an example, during terrestrial experiments, values for L, k and C_n^2 are known. We can then establish a distribution of

outer scale length by determining the average phase variance at the output signal. Given a theoretical approximation, we can algebraically calculate κ_0 and determine L_0, the outer length. An approximation provided by Dr.V.S Gudimetla [22] helps us determine an approximate phase variance value that is expressed in terms of the non-Kolmogorov power spectrum, $\phi_n^{vK}(\kappa, \alpha)$.

$$\langle \varphi^2 \rangle = \frac{1}{2(\alpha/2-1)} \Gamma(\alpha-1) \sin\left((\alpha-3)\frac{\pi}{2}\right) k^2 L C_{n\kappa}^2 \kappa_0^{-\alpha+2} \tag{3.8}$$

Given that at $\alpha = 22/6$ the power spectrum in terms of an arbitrary power law reduces to the Kolmogorov power spectrum. The phase variance in terms of the von Karman power spectrum in the Kolmogorov regime can be reduced to an approximated averaged phase variance [1] $\langle \varphi^2 \rangle = 0.782 L k^2 C_n^2 \kappa_0^{-5/3}$.

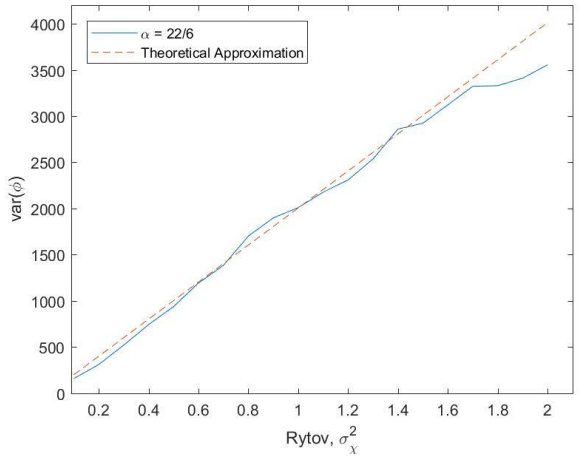


Figure 3.6: Phase variance in the Kolmogorov scale compared to Wheelon's approximation

When only looking at the Kolmogorov spectrum at an outer scale length of 10m, limitations do appear once it reaches $\sigma_\chi^2 \leq 1.6$ This can be attributed to the finite spatial dynamic range imposed by the maximum available grid size. This is because the von Kármán power spectrum has more influence in low spatial frequencies that are attributed to the outer scale length effects. The spatial grid needs to be larger so the sample frequency spacings can be low enough to accurately represent the outer scale effects. Another factor that can improve on the discrepancies is the number of trials within each simulation. To be able to reduce the variations of the mean phase variance, we would need to have a significantly larger number of trials. This would reduce the effects caused by any outliers that vary from the mean.

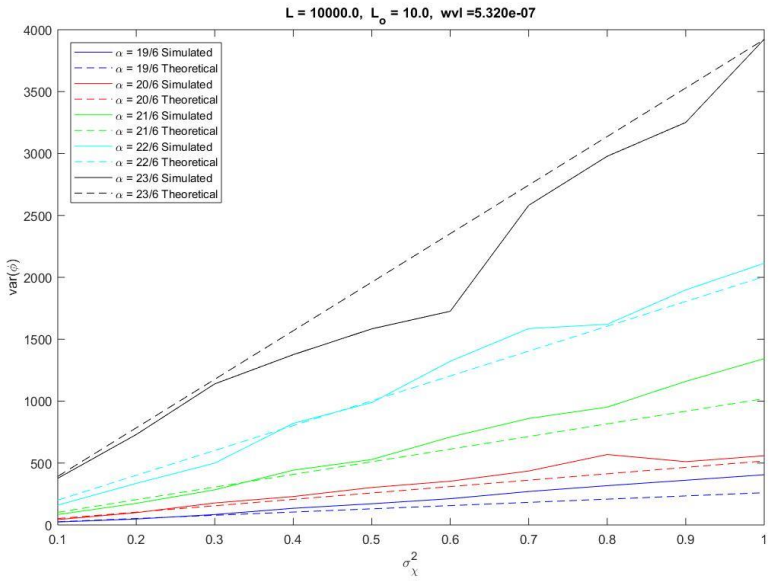


Figure 3.7: Phase Variance with respect to σ_χ^2 at varying value of α

When looking at varying levels of α the phase variance does not saturate and increases linearly with respect to Rytov. However, fluctuations do become more apparent at $\sigma_\chi^2 \geq 0.5$ and for greater values of α. This does show promising results as these values do not saturate. Aside

from the number of simulations within each trial, the fluctuations can also be attributed to the 2D Goldstein unwrapping function. At a high level of turbulence, the algorithm breaks down and show discontinuities in the image.

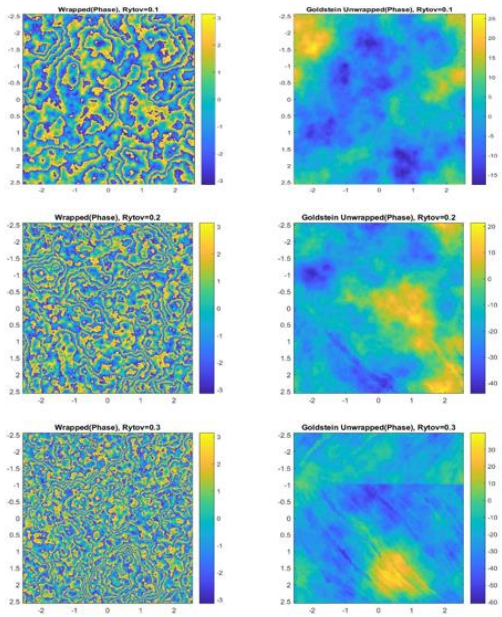


Figure 3.8: 2D Goldstein Unwrapping Algorithm breaking at high levels of atmospheric turbulence

We can continue our analysis by deriving parameters that better describe the strength of the phase fluctuations in turbulence bellow $\sigma_\chi^2 \leq 0.5$. Note that these simulations were obtained with 50 trials at relatively smaller spatial grid size (2048x2048), which may not be sufficient for higher levels of turbulence. The spatial grid needs to be larger to accurately represent the outer scale effects and increase the number of trials within each simulation to reduce any variations from the mean variance.

In order to calculate the phase structure function, we need to determine the phase covariance. But since the phase fluctuations is a stationary random process, we can determine the phase covariance by calculating the phase autocorrelation. The

autocorrelation values are calculated in the Fourier domain using equation (3.5). This results in a 2-D plane but only a vector from the center of the plane in the positive x-axis. This vector is now used in equation (3.6) to determine the phase structure function. This is done for simplicity and to compare to 1-D theoretical values of the phase structure function [22].

$$D_{\varphi}(\rho) = 2C_{n\kappa}^2 k^2 L\Gamma(\alpha-1)\sin\left((\alpha-3)\frac{\pi}{2}\right)\left[\frac{\kappa_0^{2-\alpha}}{\alpha-2} - \frac{\alpha 2^{-\alpha/2}}{\Gamma(1+\alpha/2)}\left(\frac{\rho}{\kappa_0}\right)^{\alpha/2-1} K_{1-\alpha/2}(\kappa_0\rho)\right] \tag{3.9}$$

In this equation $K_{1-\alpha/2}$ is the modified Bessel function of the second kind and Γ is the gamma function. This function allows us to determine the phase structure function various levels of α.

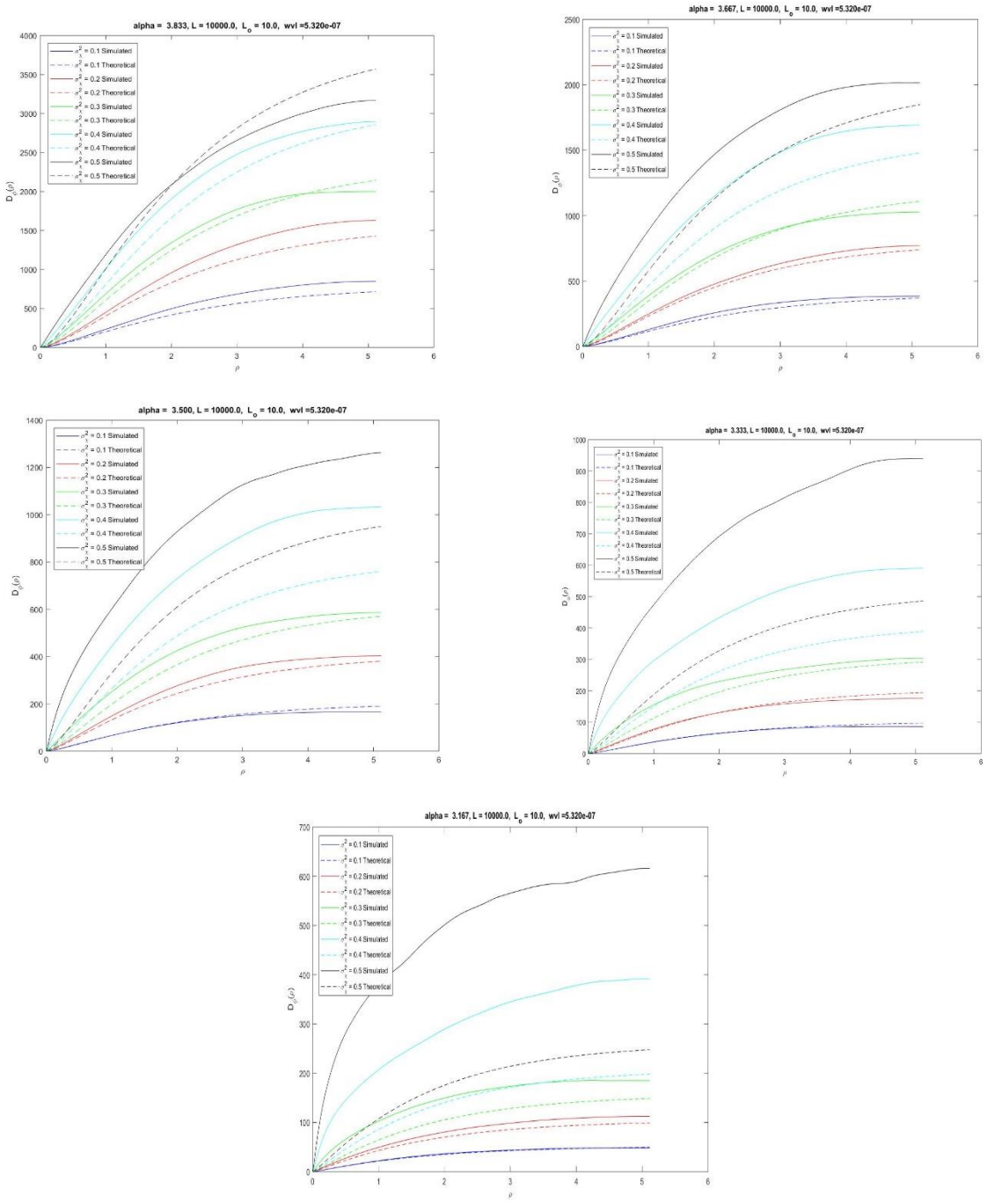


Figure 3.9: Phase Structure Function with respect to ρ at varying level of σ_χ^2; Left to Right, Top to Bottom: $\alpha = 23/6, \alpha = 22/6, \alpha = 21/6, \alpha = 20/6, \ \alpha = 19/6$

Progressing from phase variance to the phase structure function, due to the 2D Goldstein algorithm become more prominent at $\sigma_\chi^2 \geq 0.4$. Rytov values above this range

will still have simulated phase structure function that are larger in value than that of the theoretical, for all values of α. Within the range of $0.1 \leq \sigma_\chi^2 \leq 0.3$, the phase structure function function is approximately the same as the calculated theoretical values for all values of α, except $\alpha = 19/6$. However, at $\alpha = 23/6$ we seem to have a better approximation but worsen as we begin to decrease the value of α and reach $\alpha = 19/6$. Since the phase variance, autocorrelation and structure function are all closely related I believe these discrepancies are due to the unwrapping algorithm. Since there are more discontinuities it is a possibility that it is increasing the simulated value of the phase variance and so on. One method to overcome this problem is by increasing the grid size which has proven to produce better quality unwrapped imaged.

Currently, we can only characterize laser propagation at low turbulence. This does not mean the Kolmogorov and non-Kolmogorov model breaks apart at high turbulence, but further investigations are needed. We can increase the grid size from 2048x2048 to be 4096x4096. However, this increase in grid size is computationally expensive and would require significant amount of time to run these simulations. The ideal goal would be to have grid size of 8192x8192 to keep the grid spacing relatively small and still incorporate the outer scale effects.

With High Performance Computing (HPC), we can begin simulation with a screen size large enough that will allow for deep turbulence effects for a laser propagating over a long path. We can also train a deep neural network that can learn phase unwrapping. This will accelerate our algorithm if we are successful in training the model and integrating it with our current model. We can also avoid the discrepancies caused by the 2D Goldstein Algorithm at high atmospheric turbulence. Our attention will shift into training a deep neural network with the wrapped and unwrapped phase screes, to learn phase unwrapping.

Chapter 4: Accelerating Numerical Simulation with Neural Networks

Machine learning enables statistics and computer science to learn how to conduct a specific task without being programed to do so. As for example we can train machine learning algorithm to recognize the difference between a dog and a cat. It does this by taking in statistical patterns within the data to allow it to make decisions whether that data given is either a dog or a cat. Other machine learning application are face recognition, or spam filters in emails. Machine learning has the characteristic in laying in several different fields, such as artificial intelligence, statistics, computer science and some others [23]. For these simple reasons, TensorFlow library is one of the most interesting tools used during these days, even for solving engineering problems. Many machine learning libraries are available for making the predictions. TensorFlow is based on graphs-based computation to model and be implemented in Python.

TensorFlow is a second-generation system created by Google Brain that can run neural network algorithms at small portable processing systems, as well as, large scalable processing units, such as compositions of CPUs, GPUs, and hybrid systems [24][25]. Its applications are many, such as image processing, sound detection, speech recognition, natural language processing and engineering problems [26]. The tensorflow mechanism works as any high-performance computing platform, where data is given to the machine as a form of a matrix, then the platform must analyze it by mapping into small sections where it can parallelize in multiple processing units known as High-Performance computing clusters[27][28]. TensorFlow can be scalable through a range of processors, even though, having recognized that parallelism has its limits and cannot always be

accomplished. TensorFlow second-generation has greater capabilities than its older version, DistBelief[29]. Such as flexible programming, better performance and supports the training of various neural network models[26][30].

4.1 INTRODUCTION TO TENSORFLOW

TensorFlow is based on graph-based computation and it better understood with a given example. When trying to solve a simple calculation such as $a = (1 + c)/(c - b)$, we break down the function into the following concepts.

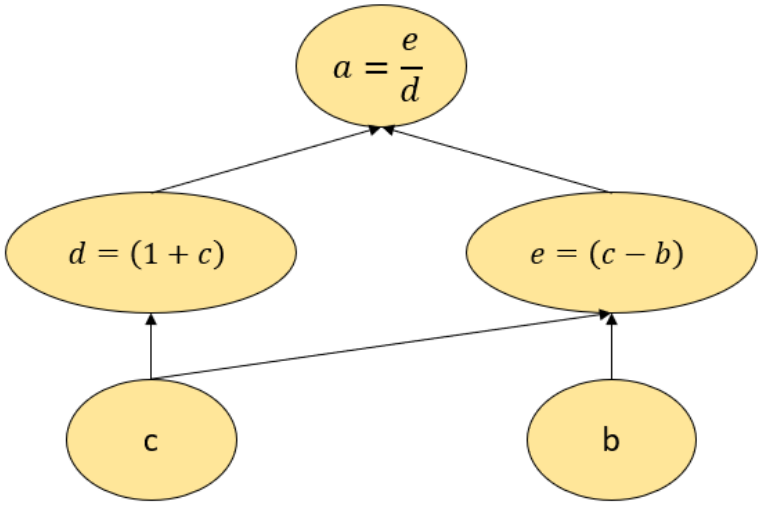


Figure 4.1: Graphical representation of a simple function

This may be over complicating a simple function, but the computations can now be done in parallel. By splitting up the computation across CPUs and GPUs can give a significant speed up time when conducting calculations. This is something that is necessary when implementing machine learning especially in the case of complicated networks. Whether it is a simple function or a complex network the idea remains similar. As a TensorFlow developer, we create the full set

of graph operations, initialize the variables, start a session, feed data into the graph and compute the output of the graph.

4.2 SUPERVISED ALGORITHMS IN TENSORFLOW

Neural networks can be trained unsupervised but, in this study, we will be mainly focusing on the supervised method. These algorithms are based on training data including inputs and outputs, where the user gives the algorithm an enormous amount of data that must be analyzed, and the algorithm predicts future data. This approach task may seem laborious, but at the end is feasible and easy to adapt. All machine learning algorithms depend on datasets. For machine learning applications the data must be normalized. The other requirement is that data must have partitions, which are training, and test sets. Then the algorithm must have hyperparameters that hold constant during the procedure of the algorithm. There is also need of telling TensorFlow what can or cannot be modified by initializing variables and placeholders. Then, there is also a need for defining the model that the graph is going to use to predict the values. The model will come to the knowledge of knowing if the predicted values are correct compared to the expected values. The last thing is to validate if the model chosen is best for solving the given problem.

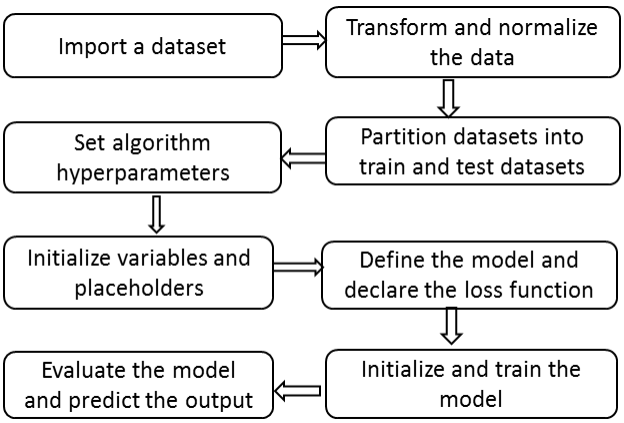


Figure 4.2: TensorFlow supervised algorithm flow chart.

4.3 SIMPLE NEURAL NETWORK

A simple neural network is composed of a variety of nodes that are connected together and communicate with one another. Each node takes in multiple inputs and conducts the dot product with each input weight. It then applies an activation function to the product of the inputs and generates an output.

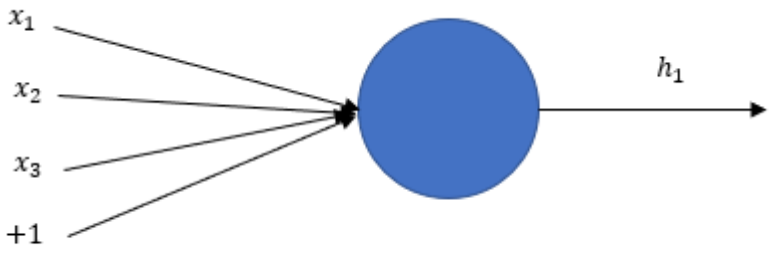


Figure 4.3: Neural network node.

The circle represents the nodes with three inputs and a bias. It begins by taking the dot product of the inputs and weights added by a bias.

$$[x_1 \quad x_2 \quad x_3] \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + [b] = x_1 w_1 + x_2 w_2 + x_3 x_3 + b \tag{4.1}$$

This product then used as an input to an activation function. An activation function acts as a switch and depending on what the function is, it can range a value from 0 to 1 or from -1 to 1. It is common practice that most activation function is either sigmoid or ReLu (Rectified Linear Unit). However, we will later see that there is a more complex activation function.

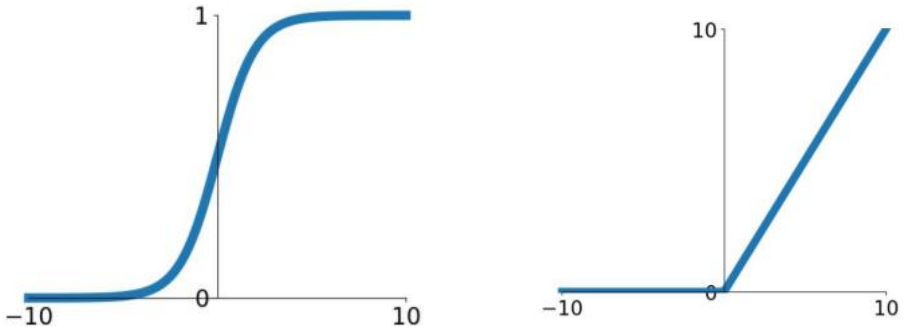


Figure 4.4: Sigmoid Function (Left), ReLu Function (Right)

After the activation function is calculated the output is then determined, denoted by ***h***.

$$h_1 = f(x_1 w_1 + x_2 w_2 + x_3 x_3 + b) \tag{4.2}$$

In this case, the activation function is denoted by $\boldsymbol{f}(*)$**.** When it comes to neural network it easier to remember that there are two components. The is the linear components which is the linear algebra and the non-linear component which is the activation function.

Now to better explain machine learning, it would be better to go over a simple neural network. A common neural network consists of an input layer, a hidden layer, and an output layer.

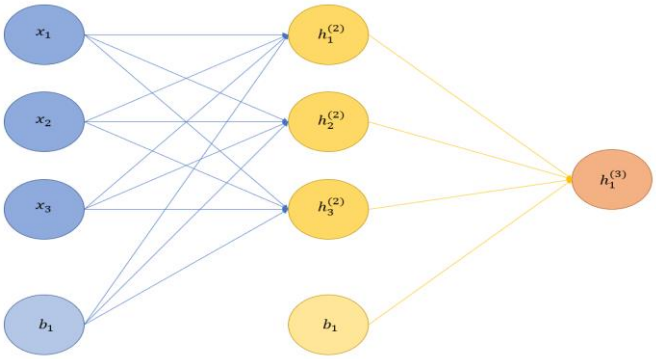


Figure 4.5: Simple Neural Network with an input layer, hidden layer and an output layer

In order to identify the weights, we will use the following notation, $\boldsymbol{w_{ij}^{(l)}}$. The notation $\boldsymbol{l}$ refers to the connection layer and the $\boldsymbol{i}$ notation refers to the node number of the connection in the layer $\boldsymbol{l+1}$. While the $\boldsymbol{j}$ notation refers to the node number of the connection layer $\boldsymbol{l}$. So, for the connection between node 1 in layer 1 and node 3 in layer 2, the weight notation would be $\boldsymbol{w_{31}^{(1)}}$. Since the bias is not really a node with an activation function, it has no inputs. The notation for the bias weight is $\boldsymbol{b_i^{(l)}}$, where $\boldsymbol{i}$ is the node number of the connection in the layer $\boldsymbol{l+1}$. So, the bias notation for the connection between node 1 in layer 1 and node 3 in layer 2, is $\boldsymbol{b_3^{(1)}}$. Finally, the node output notation is $\boldsymbol{h_j^{(l)}}$, where $\boldsymbol{j}$ refers to the node number in layer $\boldsymbol{l}$. The output of node 2 in layer 2 would be $\boldsymbol{h_2^{(2)}}$.

The feed-forward pass is straightforward. To calculate any of the hidden nodes including the output layer would be like calculating one node but now we have multiple nodes.

$$h^{(l+1)} = f\left(W^{(l)}h^{(l)} + b^{(l)}\right) \tag{4.3}$$

The value $\boldsymbol{W^{(l)}}$ denotes the matrix form of the weights, and the input layer could be also written as $\boldsymbol{h^{(1)}}$. To make things easier we can expand the equation and solve for the hidden layer in our simple neural network.

$$h^{(2)} = f\left(\begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_2^{(1)} \end{bmatrix}\right) \tag{4.4}$$

In this case, it is simply matrix multiplication followed by an activation function to solve the hidden layer and we would repeat this process to solve the output layer. When building a neural network, the key goal is to determine the correct weights for a given problem. At the beginning of constructing a neural network, the weights are initialized through a Gaussian random number generator. The goal is to get the neural network to adjust its weights as to get better predictions at the output node. This is done by varying the weights to minimize the error between the true data set and the predicted value. To do so we need to implement backpropagation and gradient descent.

$$w = w_o - \alpha * \nabla error \tag{4.5}$$

The gradient descent method uses the gradient to make a change in the new value of the weight $\boldsymbol{w}$. It takes the current position $\boldsymbol{w_o}$, and adjust it by the gradient error weighted by a step size $\boldsymbol{\alpha}$. The most important component in machine learning would be the loss function.

To optimize a neural network revolves around minimizing the loss function. As mentioned before when we are conduction supervised training, we want the network to learn how to make predictions off the given data set. For a given training pair, we will denote the input data

as $\boldsymbol{x^z}$ and the desired output to be $\boldsymbol{y^z}$. So, an $\boldsymbol{L^2}$ the loss for a training pair in a neural network can be given as

$$J(w,b,x,y) = \frac{1}{2}\left\|y^z - y_{pred}(x^z)\right\|^2 \tag{4.6}$$

Let it be noted that $\boldsymbol{y_{pred}(x^z)}$ is the output layer of out neural network and in our case can be written as $\boldsymbol{h_1^{(3)}(x^z)}$. To generalize the cost function to include more than one training pair we want to minimize the loss function over all m, training pairs.

$$\boldsymbol{J(w,b) = \frac{1}{m}\sum_{z=0}^{m}\frac{1}{2}\left\|y^z - y_{pred}(x^z)\right\|^2} \tag{4.7}$$

To determine the new weights, the gradient decent for every weight and bias in a neural network becomes

$$w_{ij}^{(l)*} = w_{ij}^{(l)} - \alpha\frac{\partial}{\partial w_{ij}^{(l)}}\boldsymbol{J(w,b)} \tag{4.8}$$

$$b_i^{(l)*} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}\boldsymbol{J(w,b)} \tag{4.9}$$

The way neural networks work is that we initialize the weights. Then we feedforward and calculate the predicted value. We then take the predicted value and the true value and determine the loss function. Through backpropagation and gradient decent we adjust the weights and bias. We can see the importance of the activation function from equation (4.8) and equation (4.9). Whenever the output of an activation function is near zero there would no information to adjust the weights when it is conducting gradient decent. This is how a neural network learns and

begging to exclude unnecessary values. When the weights are not being adjusted we know that our neural network has converged. This may be a simple neural network, but the overall concept remains the same for complex networks.

4.4 Convolutional Neural Network

To be able to comprehend complex neural network we need to introduce the idea of a convolutional neural network. Like our previous example of a simple neural network, we are now working with images or matrices. The weights are now introduced as a filter that convolves with the image. The weight is dependent on the kernel size and stride. It is common practice to keep the weights or filters to be square matrices. However, in order to work well each convolutional stage usually has multiple filters.

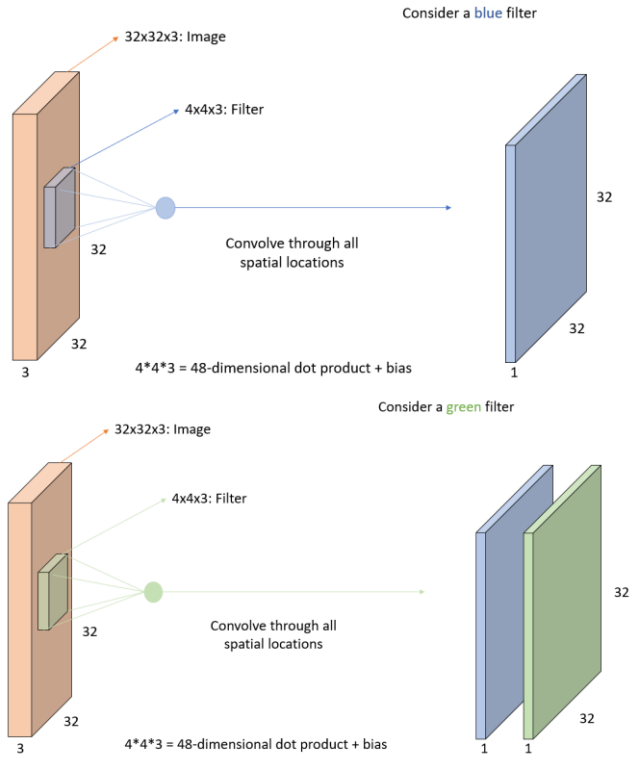


Figure 4.6: Convolution Layer with only 2 filters

Each filter will convolve fully with the image and produce their own 2D output. Each filter has its own set of weights that will be adjusted to train the network. In this example, the filter will convolve fully with the colored image of multiple channels producing an output. This will be repeated depending on the number of filters producing a new 3D matrix. Each convolution output for each node will also be passed through an activation function.

The kernel size is related to the size of the filter/weight and in our example, it is a [4,4] kernel. We can also control how much the filter moves or the stride, whether it be a single space or more. In cases where we might want to downsize an image to reduce the number of parameters and to do so, we can increase the stride. However, if we want to maintain the same dimension of an image but want to increase the stride as to not overlap filter areas, padding may be involved. Dummy nodes have zero value with a rectified linear unit activation of the previous layer it won't have much value when introduced. This is implemented only when we want to maintain the same dimensions as the previous image.

Another important component of the convolutional neural network is called pooling. Some convolutional neural network will implement pooling in order to downsample an image. The reason to downsample is to reduce the number of parameters in a neural network. We also want to make feature detection more complex by making it tolerant to scale and orientation changes. Like convolutional filters, pool filter moves along an image taking only the largest value within the image. However, pool filters do not attain any weights and are better explained through figure (4.7).

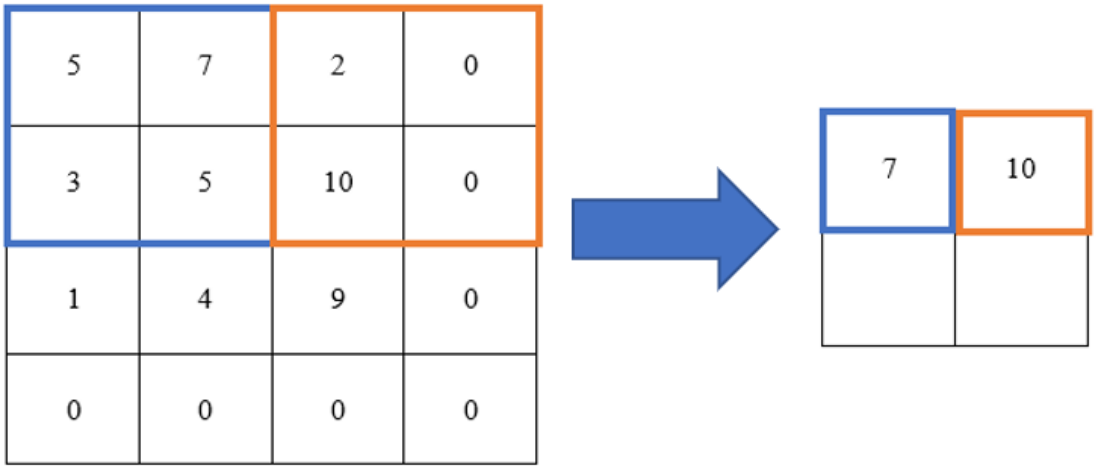


Figure 4.7: Max Pooling of a 4x4 matrix to a 2x2 matrix

Also note that this max pooling filter has a stride of two, causing the matrix to be downsized. When introducing padding the output height and width is equivalent to $\boldsymbol{H = H_o/stride}, \boldsymbol{W = W_o/stride}$. Let's consider a more complex network called a U-Net model for tumor detection and segmentation [31], [32]. This will allow us to see the overall picture of fully convolutional neural networks.

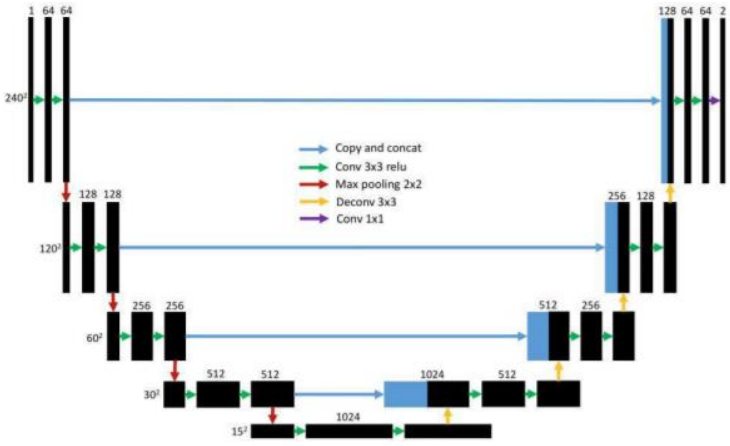


Figure 4.8: U-Net Model for Image Segmentation [32]

This fully convolutional neural network consists of down sampling and upsampling. With the input and output image being of the same size. The down-sampling path has 5 convolutional portions. Every block has two convolutional layers with a filter size of 3×3, and a stride of 1. The activation function is a rectifier activation. For each block of the down-sampling, the number of convolutional layers increases by a factor of 2. After each convolutional block, a down-sampling of max pooling is used with a stride 2×2 is applied. The decrease in size is divisible by 2 until reach the last block of 15x15. In the up-sampling path, to increase each block, a deconvolutional layer with a filter size of 3×3 and stride of 2×2 is applied. It doubles the size of feature maps. In every up-sampling block, two convolutional layers are also implemented. The number of filter maps is reduced by a factor of 2. However, feature maps of the down-sample path are concatenated with the up-sample path before being convoluted. The loss function is a 'Soft' Dice loss function introduce in [32]. Like our previous example of a simple neural network, the U-Net model will adjust itself through backpropagation and gradient descent. With a general concept of machine learning we can begin to dive into the type of architecture we implemented on our current numerical simulation and why.

4.5 Reasoning Behind Implementing a Deep Neural Network

To analyze turbulence effects, laser propagation is simulated through the phase screen approach and modeled by Kolmogorov [19] and non-Kolmogorov statistical theories [18]. Our main objective is to characterize atmospheric turbulence effects through statistical analysis of the phase fluctuations at the observer's plane. To do so we need to calculate the phase screen with the arctangent function, and the values are restricted to an interval of $(-\pi, \pi)$. Overcoming this problem requires 2D phase unwrapping to restore the true phase values, assuming that the unwrapped phase map is a continuous surface. To unwrap the phase screens a 2D Goldstein

unwrapping algorithm is implemented that is compiled by *Bruce Spottiswoode* and edited by *Carey Smith* [20]. However, there are some complication when using the current unwrapping algorithm.

As we mentioned before when calculating the phase variance and phase structure function, we saw discrepancies at Rytov values greater than 0.5. This may be due to the grid size not being sufficiently large, such that sample frequency spacings can be low enough to accurately represent the outer scale effects. It could also be due to the Goldstein algorithm to the current numerical simulation breaking down when we begin to implement deep atmospheric turbulence When we plot the unwrapped phase screens as shown in figure(14c) we see fractures and discontinuities in the image. To accurately develop a numerical simulation of laser propagation we need to overcome this problem.

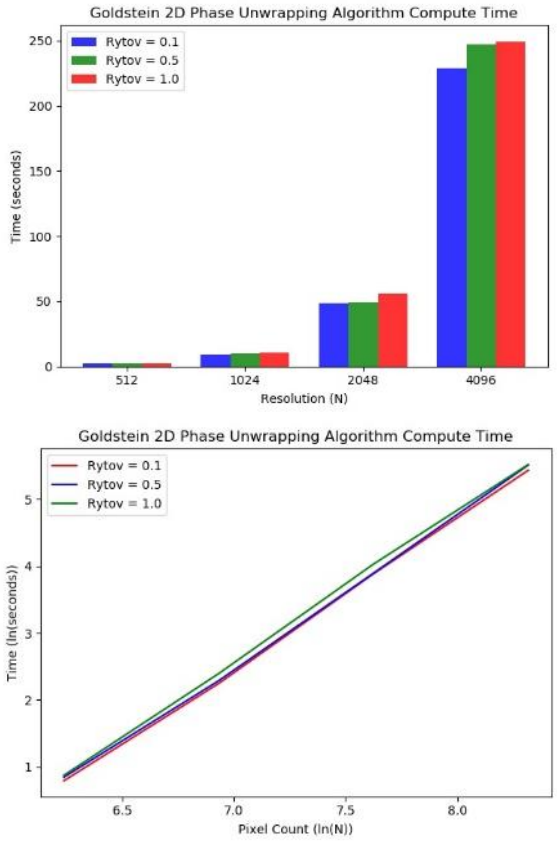


Figure 4.9: Goldstein Algorithm Compute Time relative to spatial grid size

The algorithm does not scale well for large spatial grids (Figure 1). Another reason to implement neural networks is to accelerate the current simulation. It would be beneficial to reduce the computational time between each simulation. This will allow us to compute statistical parameters at larger grid sizes at a reasonable time. These statistical parameters need to be as accurate as possible to better represent the effects of atmospheric turbulence. When plotted on a log-log scale we can see that the algorithm compute time scales exponentially with the grid size. To accelerate our numerical simulations of wave propagation, we train a deep neural network to approximate the Goldstein 2D phase unwrapping.

4.6 CONDITIONAL GENERATIVE ADVERSARIAL NETWORK

Convolutional neural networks have been the standard to solve a variety of classification and regression problems [32]–[35]. However, convolutional neural networks can't be applied to a problem domain naively, because the specified loss may produce undeniable results. For example, it is common that implementing an L2 norm loss to a convolutional model can produce a blurred image [36], as a local average is often a good approximation of an image. Implement a loss function that produces desired images is complicated and problem-specific. Generative adversarial networks (GAN) [37] are a deep neural network architecture that effectively learns a loss function simultaneously, alongside a useful mapping function. A standard (or vanilla) GAN comprises two components, the generator, and the discriminator, and pits them against each other.

For this study, we implement cGAN [38] to learn phase unwrapping, which is similar to a vanilla GAN but applies a conditioning variable. This conditioning variable is seen by both generator and discriminator which establish some arbitrary condition for a generation. We can state the objective of a using a min-max value function.

$$\mathcal{L}_{cGAN}(G,D) = \min_{G}\max_{D}\left(\mathbb{E}_{x,y}[logD(x,y)] + \mathbb{E}_{x,z}\left[log\left(1 - D\big(x,G(x,z)\big)\right)\right]\right) \tag{4.10}$$

By minimizing this loss function, we train the discriminator, $D(x,y)$, to maximize the expectation of target data distribution, y, and minimize expectation of the generated model data distribution, $G(x,z)$. At the same time, we need to train the generator to maximize the probability that the discriminator is fooled into choosing the generated data or fake image. In this

instance, the conditioning variable is the input image, x, which will be seen by both the generator and the discriminator. In our problem, the wrapped phase screen with be the input image, and the unwrapped phase screens will be the target image (Figure 4.10).

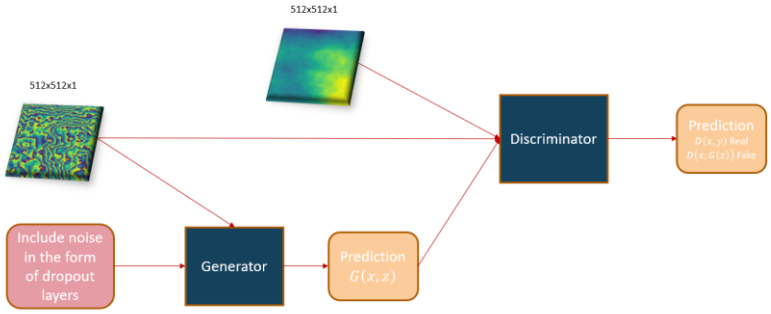


Figure 4.10: Conditional Generative Adversarial Network for Phase Unwrapping

In a cGAN, it is common to input a noise variable, z, to the generator to impose stochasticity on the outputs. Past cGan models try to solve this problem by implementing a Gaussian noise, z, to the generator in addition to, x. The strategy is ineffective, as the generator learns to ignore the random noise. The cGan model that implemented in this work for phase unwrapping applies noise in the form of dropout layers in several layers of the generator [38].

Before going in depth into each architecture involved in a cGAN, it is important to understand why we need to preprocess the dataset before training. Before training a model, we need to normalize the data set before feeding it into the neural network. Depending on the activation function we need to normalize the data set to be between [0,1] or in our case [-1,1]. This is because the activation functions only work within a small range. As an example, our generator final activation function is a $\tanh(x)$. Any information that is not within the range of [-1,1] will be lost and not included in our network. That is why normalizing and batch

normalization are important. This makes sure that values do not stray far from the range of the activation function. Thus, the gradient descent can reduce the oscillations when approaching the minimum point and converge faster. It also acts as a form of regulation and prevents overfitting.

The task of the generator is to be able to take the input image and perform the requires transforms to produce the target image. In our case, we would want the generator to be able to perform phase unwrapping. It would input the unwrap phase and be able to transform the 2D matrix and be able to produce an unwrapped phase. Most generators consist of the encoder (down-sampling) and decoder (up-sampling) layers. However, the generator we are implementing resembles a U-Net model and implements skip-connections.

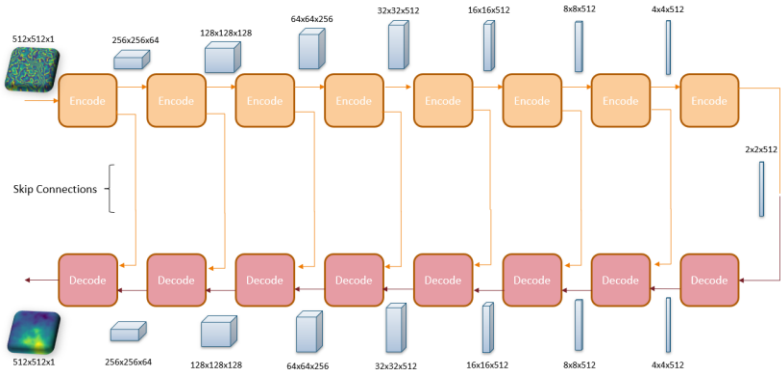


Figure 4.11: Generator Architecture

The generator takes the input image and reduces it with a series of encoders into a much smaller representation. The idea is to not only reduce the number of parameters in the networks but have a higher representation of data in the final encoder layer. Then the

decoder will do the exact opposite and propagate this information to higher resolution layers. The skip connections have shown to help recover spatial resolution that may have been lost within the network [39].

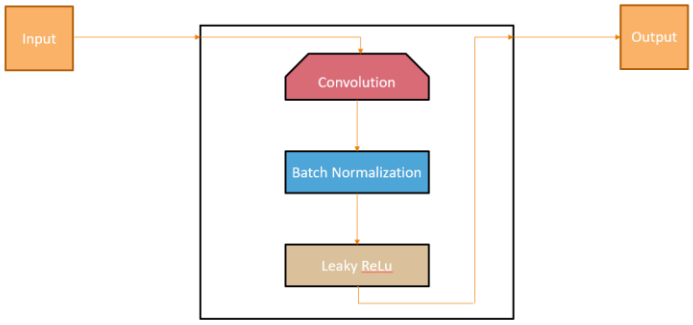


Figure 4.12: Encoder Structure

The encoders consist of convolution, batch normalization, and an activation function. The convolution consists of a kernel size of 4x4 and a stride of 2. This decreases the input image by a factor of 2. We then implement batch normalization before the activation function. The encoder activation function is a Leaky Rectified Linear Unit (Leaky ReLu). Like a ReLu, a Leaky ReLu does not saturate for values less than zero. This allows for more information to be included in the network for gradient descent. The output channel after each encoder should increase by a factor of 2 before reaching 512. After reaching a channel of 512 it will remain at that value for the remainder of the encoders.

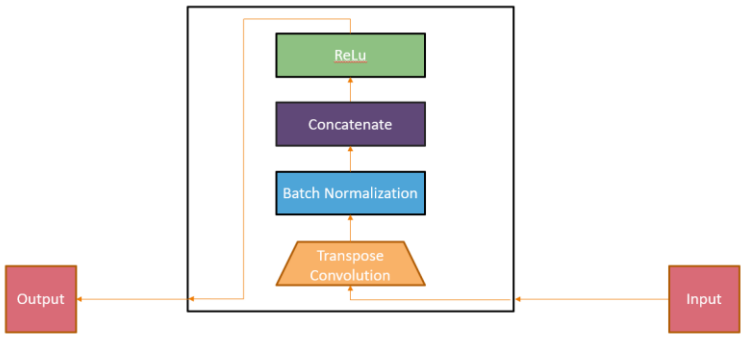


Figure 4.13: Decoder Structure

The decoder consists of transpose convolution with a kernel size of 4x4 and a stride of 2. It increases the input image by a factor of 2. Then the image will be normalized before being concatenated with an encoder image of similar height and width. The activation function for a decoder is a ReLu. The first four decoder blocks will have a filter channel of 512 before reducing by a factor of 2. The images are a simplification of the generator model. Other things that are not mentioned in the image is that the last activation function of our generator will be a $\tanh(x)$. We need to also note that the first three decoder blocks include the drop out layers. This is the architecture of the generator which will try to fool the discriminator and make predictions of phase unwrapping.

The discriminator will take two images as an input. It will take the input image which is out conditioning variable and an unknown image. This unknown image is either the target image or the predicted image that is produced by the generator. The generator will take this two pairs of images and concatenate them together and determine if it was produced by the generator or not.

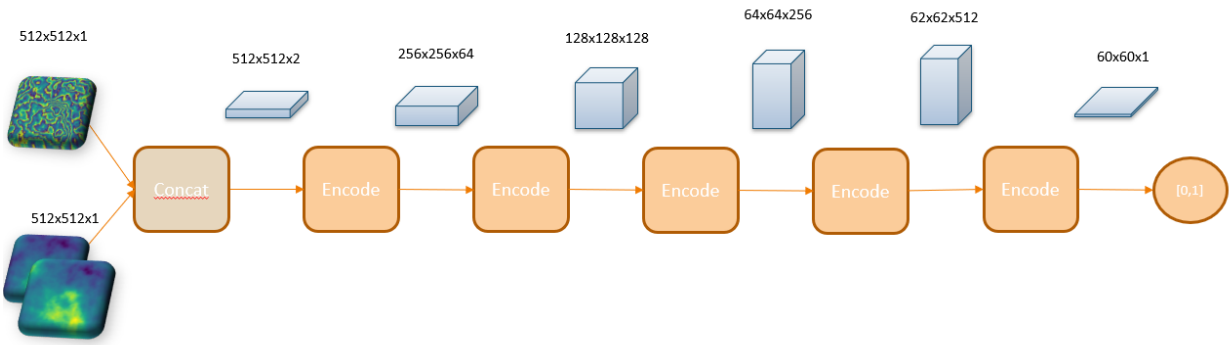


Figure 4.14: Discriminator Architecture

The architecture of the discriminator built up of only encoders. The encoders are similar to those of the generator which have convolution, batch normalization, and a Leaky ReLu activation function. However, the last two encoder portions have a stride of 1. The output should be a 60x60x1 image where each pixel value ranges from 0 to 1. The 0 denotes that it fake and 1 represents that is the target image. Each pixel value corresponds to how believable the image is to the target image. This type of architecture is called a PatchGAN [38].

To train this network, there are two steps, training the discriminator and training the generator. To train the discriminator, the generator will generate a fake image. The discriminator looks at the input/target pair and we will specify to the model that this is a real image which will be denoted by 1. Then the discriminator will look at the input/fake pair and we will specify to the model that this is the fake pair which is denoted by 0. The weights of the discriminator are then adjusted based on the classification error of the input/output pair and the input/fake pair (Figure 4.15: Left).

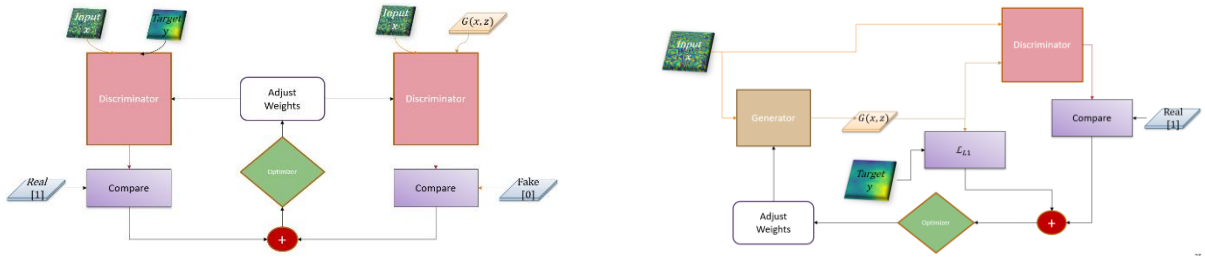


Figure 4.15: (**Left)** Training for the discriminator (**Right)** Training for the generator

We will train the generator to take the input/fake pair to fool the discriminator to thinking it is a real image and denote it with 1 (Figure 4.15: Right). Not only do we train the generator to fool the discriminator we also want it to be near the ground truth through an L1 loss.

$$G^* = \mathcal{L}_{cGAN}(G, D) + \beta \mathcal{L}_{L1}(G) \tag{4.11}$$

The advantage of the cGAN architecture is bootstrapped training. As the discriminator reduces its loss contribution, the generator loss term becomes dominant. This forces parameter changes to reduce the generator loss until the decimator contribution is again dominant. In turn, the discriminator must reduce its loss contribution yet further. This iterative minimization of mutually adversarial losses forces the generator to become arbitrarily effective at the desired mapping.

4.7 Phase Unwrapping with Images

To begin this study, we had to generate the dataset which we did by converting the numerical simulation in MATLAB into an executable file. This executable file will

output an HDF5 file containing the unwrap and wrapped matrices. These values contain negative and positive float values. We would then call on executable file through python where we converted the matrices into images. This was only done as a proof of concept and make sure that this will be a feasible problem.

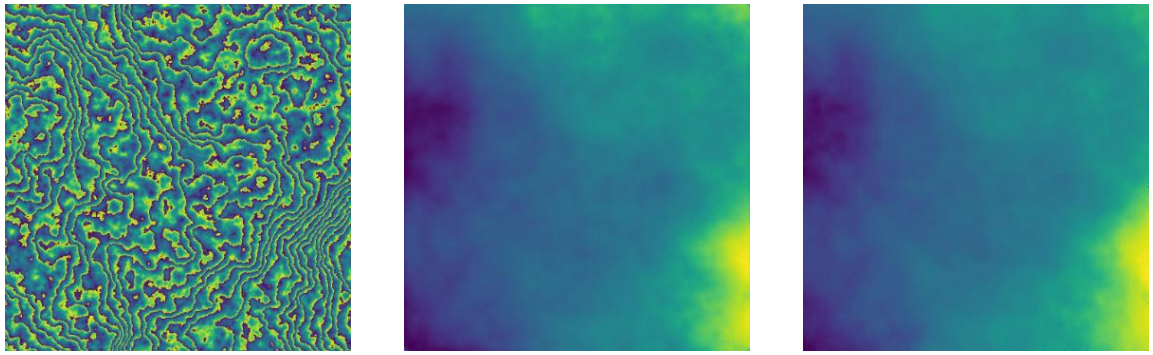


Figure 4.16: Training Image Set: (Left) Input Image, (Middle) Predicted Image, (Right) Target Image

The dataset generated for this study comprised of 4000 training images and 1000 validation images. The grid size of these images is 512x512. The cGAN model was trained for 3.5 hours, which yielded 45 epochs of training. A qualitative comparison of the resultant model output is shown in the figures below. These images are taken at random from the validation partition of the dataset and were therefore never used to train the model. Thus, these images constitute a qualitative estimate of the model's generalization performance.

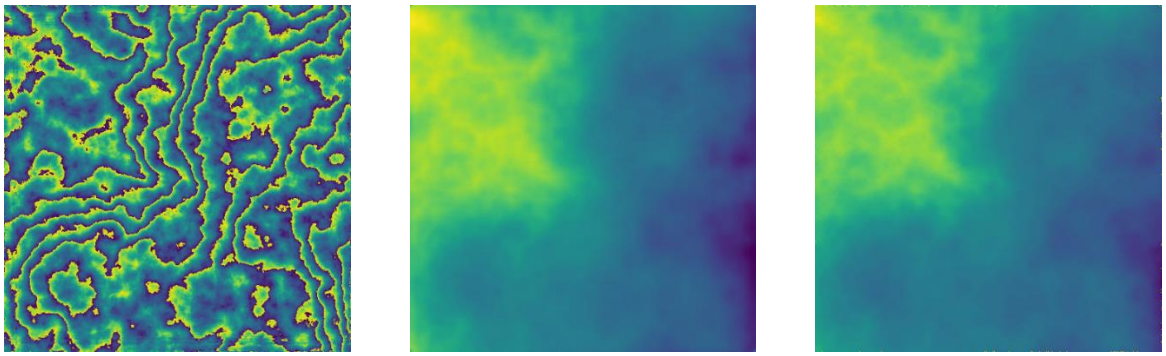


Figure 4.17: Validation Image Set: (Left) Input Image, (Middle) Predicted Image, (Right) Target Image

The obtained results suggest that conditional generative adversarial networks are a promising approach to learn phase unwrapping. It may be possible to make predictions for larger image sizes but would need further investigation to verify. We would need to test this current neural network onto the actual float matrices of the wrap and unwrapped phase screens. The objective is to integrate our model to our current numerical simulation of wave propagation and calculate statistical parameters on the unwrapped phase screens. So instead of working with images, we would need to test our model on the actual data set.

4.8 PHASE UNWRAPPING WITH MATRICES

Instead of converting the matrices into images we want to train out a model with the float matrices. But before feeding the data into the neural network we need to determine the sample data set mean and standard deviation. We need to normalize the data set before feeding it into our model.

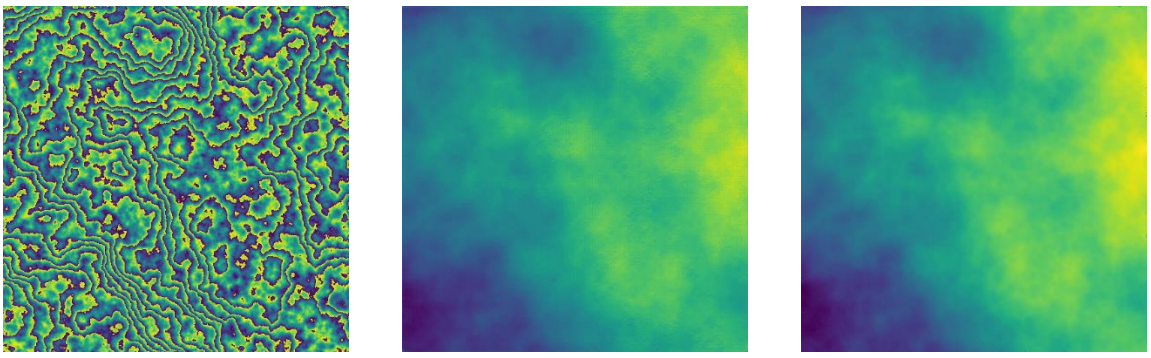


Figure 4.18: Validation Matrix Set: (Left) Input Matrix, (Middle) Predicted Matrix, (Right) Target Matrix

The dataset generated for this study comprised 4800 training matrices and 1200 validation matrices. The grid size of these images is 512x512. The cGAN model was trained for approximately 40 hours, which yielded 250 epochs of training. A comparison of the resultant model prediction and the target image are shown is shown in Figure (4.18). These images are taken at random from the training partition of the dataset like before. It seems that the predicted image is approximately like the target image, being able to show the target image pattern. But looking at the predicted matrix closely, we see that there is a checkerboard pattern in the image.

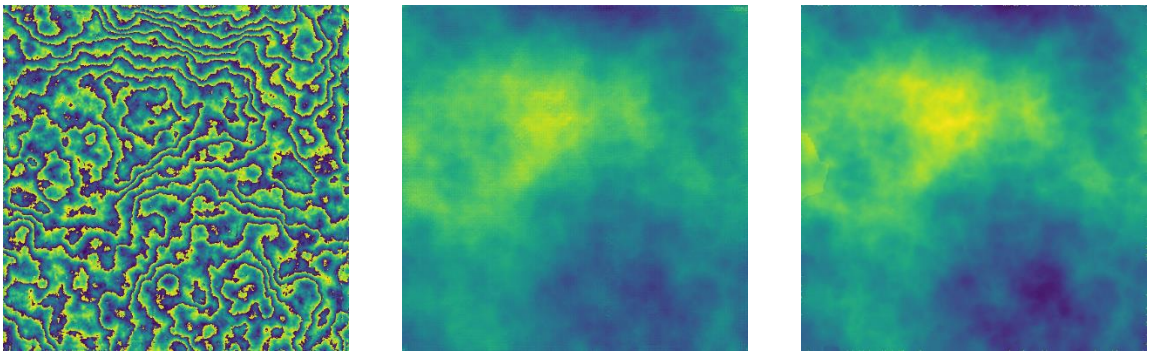


Figure 4.19: Validation Matrix Set: (Left) Input Image, (Middle) Predicted Image, (Right) Target Image

Even though the predicted image almost resembles the target image for the validation set, the checkerboard artifacts become more prominent than in the training set. This is due to the transpose convolution operation that we use to go from a low resolution to a higher one.

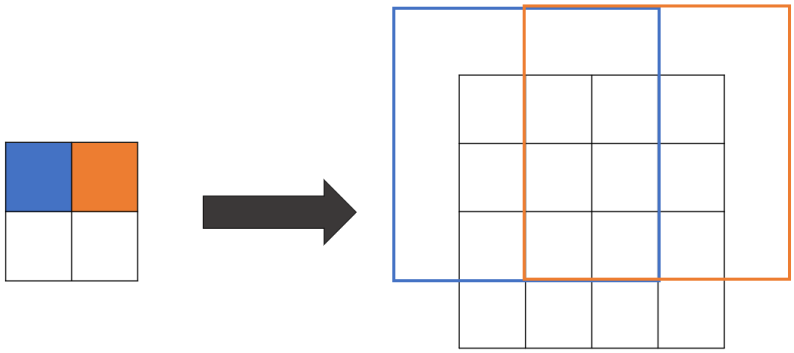


Figure 4.20: 4x4 Transpose convolution with a stride of 2

Transpose convolution work by swapping the forward and backward passes of a convolution. This can take a small 2x2 image and increase the resolution to a 4x4 image. The problem with transpose convolution is it can have uneven overlap between pixels. This uneven overlap occurs more when the kernel size is not divisible by the stride. The uneven overlap can be compounded when transposed convolutional layers are stacked. In practice to avoid these artifacts a convolution of stride 1 is used on the last layers, but in other models, it isn't enough [40]. To improve on the image resolution [41], instead of up-sampling with transpose convolution we resize the image followed by a convolutional layer. We will be replacing our transpose convolutional layers with nearest neighbor interpolation. Then follow it with a convolutional layer with a kernel size of [2,2] and a stride of 1.

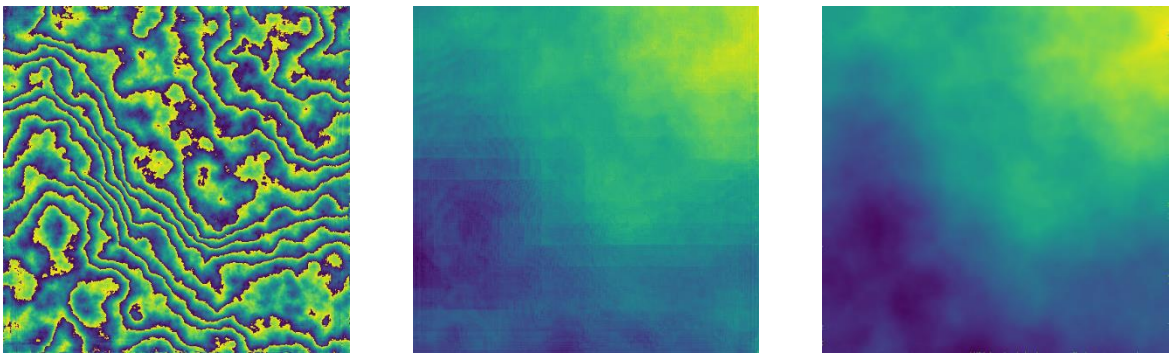


Figure 4.21: Modified cGAN Training Set: (Left) Input Matrix, (Middle) Predicted Matrix, (Right) Target Matrix

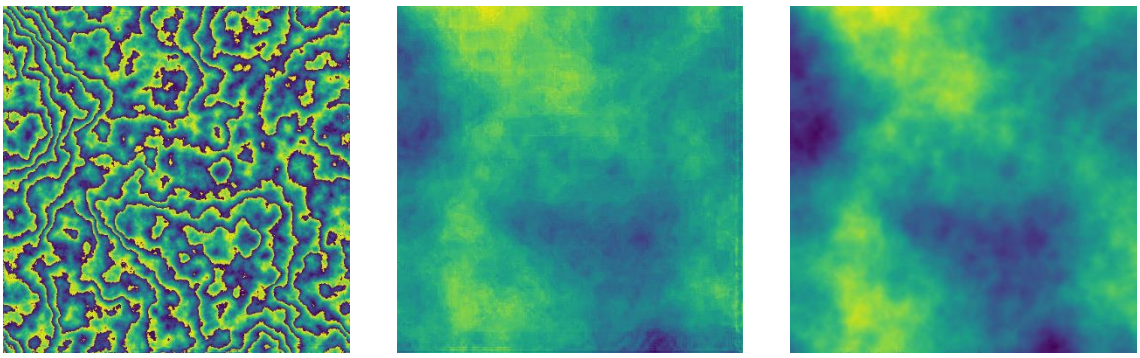


Figure 4.22: Modified cGAN Validation Set: (Left) Input Matrix, (Middle) Predicted Matrix, (Right) Target Matrix

However, when we modify our cGAN to include the resize method using nearest neighbor interpolation and a convolutional layer we see more checkerboard artifacts. This may be due to the kernel size and stride chosen for the convolutional layer. There may be pixel overlap and aren't really negating the effects caused by transpose convolution in the backwards pass. More work needs to be conducted to remove any artifacts in the matrix, so we can make actual prediction with float matrices. Being able to conduct phase unwrapping using neural networks will not only accelerate our current simulation but allow us to conduct larger grid sizes that isn't currently feasible. We can still conduct phase unwrapping with the current algorithm with some restrictions. This still allows us to implement CFD to model laser propagation in atmospheric turbulence.

Chapter 5: Governing Equations of Fluid Mechanics

To begin implantation of CFD on our current simulation this section will go over the basics of fluid mechanics, to plant the base of our overall research. We will consider a fluid, whether it be liquid or gas, to be moving in the domain Ω included in three-dimensional space, $\mathbb{R}^3$. We will describe the fluid flow with the Navier Stokes equation which is derived by the mass conservation and fluid motion equation. In this study, we will only be looking at air and the effects of atmospheric turbulence on laser propagation. We will consider air to be an incompressible Newtonian fluid.

$$\frac{\partial u}{\partial t} + \bar{v} \cdot \nabla \bar{v} = -\nabla p + \nabla \cdot (2\upsilon D\bar{v}) + f \tag{5.1}$$

$$\nabla \bar{v} = 0 \tag{5.2}$$

In this scenario $\bar{v}$ is the velocity of the fluid, p is pressure and $D\bar{v} = (1/2)(\nabla \bar{v} + \nabla \bar{v}^T)$ is the deformation tensor. Eq.1 is the momentum equation, derived from Newton's law while, equation (5.2), is determined from the mass conservation equation of incompressible fluids. We will go over the derivation of both equation and how coupled yield the Navier Stokes equation.

Before deriving the governing equations, we need to define the Lagrange and Euler coordinate systems. However, in fluid mechanics, we will be dealing mainly with the Euler description. Euler coordinates are denoted by $\boldsymbol{x}$, which represents the spatial

coordinate. A spatial coordinate specifies the location of a point in space. Lagrangian coordinates also known as material coordinates, are denoted by $\boldsymbol{X}$. The material coordinate specifies the materials point or location. In Eularian coordinates, the nodes of the mesh are fixed in $\boldsymbol{x}$, and the material moves through the mesh. In a Lagrangian coordinate the mesh is fixed in $\boldsymbol{X}$, and the mesh moves with the material.

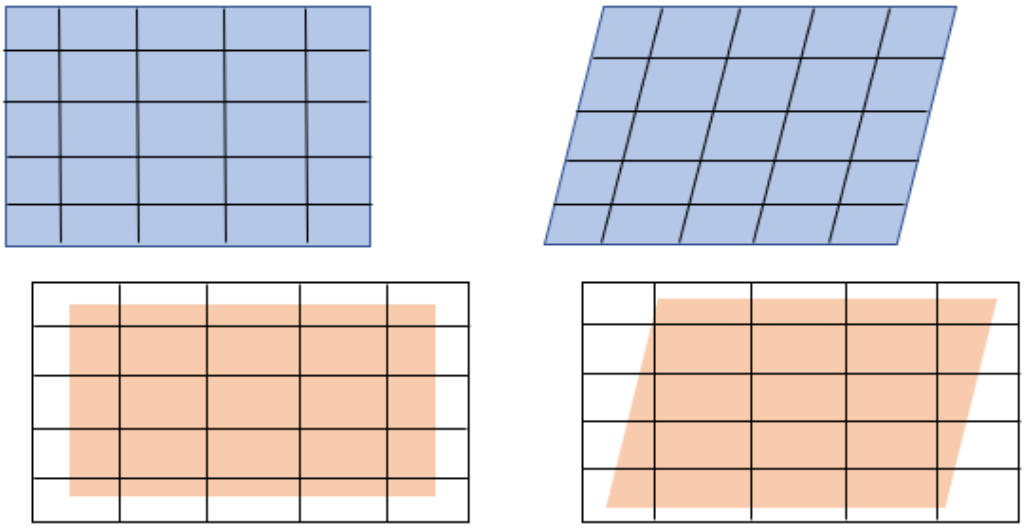


Figure 5.1: Two-Dimensional shearing of a block: Top (Lagrangian) and Bottom (Eularian)

The deformation and/or motion of a body is described by the function $\phi(\boldsymbol{X}, t)$. This function gives the spatial position of the material coordinate point as a function of time.

$$\boldsymbol{x} = \phi(\boldsymbol{X}, t) \tag{5.3}$$

This is also the map between the initial and current configuration. The displacement u of a material point is the difference between the current position and the original position.

$$u(\boldsymbol{X}, t) = \phi(\boldsymbol{X}, t) - \boldsymbol{X} \tag{5.4}$$

The velocity of the material point can be easily determined by taking the time derivative of the motion or deformation of the body.

$$v(\boldsymbol{X}, t) = \frac{\partial \phi(\boldsymbol{X}, t)}{\partial t} \tag{5.5}$$

All of these derivations were done with the material coordinate being fixed meaning that this is in the Lagrangian frame of reference. We can also express the material coordinates in terms of the spatial coordinates by inverting the mapping function.

$$\boldsymbol{X} = \phi^{-1}(\boldsymbol{x}, t) \tag{5.6}$$

This will allow us to describe the velocity as a function of the Eularian coordinates.

$$\bar{v}(\boldsymbol{x}, t) = v(\phi^{-1}(\boldsymbol{x}, t), t) \tag{5.7}$$

We have indicated Eularian frame of reference by placing a bar over the velocity symbol. The difference in the Eularian and Lagrangian meshes is seen in the behavior of the nodes. If the mesh Is Eularian, the Eularian coordinates of nodes, $\boldsymbol{x}$, are fixed. If the mesh is Lagrangian, the material points, $\boldsymbol{X}$, are time invariant.

5.1 MASS CONSERVATION EQUATION

Let us consider a fluid particle that is located at $\boldsymbol{x} = \boldsymbol{x}(t)$ and moves to $\boldsymbol{x} = \boldsymbol{x}(t + \delta t)$. This particle is moving along a trajectory with a given velocity $\bar{v}(\boldsymbol{x}, t)$.

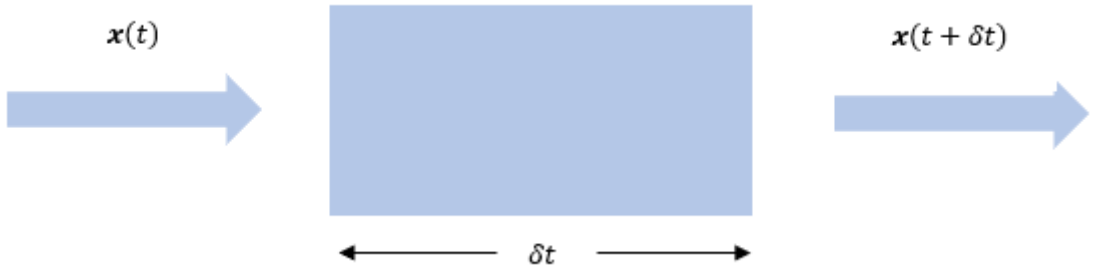


Figure 5.2: Fluid particle moving across a controlled volume

The mass conservation equation states that the mass of a particle remains constant along a given trajectory.

$$\frac{D(dm)}{Dt} = 0 \tag{5.8}$$

Recall that mass $dm(x, t)$ is equivalent to $\rho(x, t)dv(x, t)$, where dv is the local volume. All we need to do is compute the total derivative of mass.

$$\frac{D(dm)}{Dt} = \left(\frac{\partial \rho}{\partial t} + \bar{v} \cdot \nabla \bar{v}\right) dv + \rho(\nabla \bar{v}) dv \tag{5.9}$$

We can then combine equation (5.8) and (5.9) to obtain

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\bar{v}\rho) = 0. \tag{5.10}$$

However, for incompressible flow, the mass density remains constant during in motion. This reduced our conservation of mass to be

$$\nabla \cdot \bar{v} = 0. \tag{5.11}$$

5.2 THE EQUATION OF MOTION

To solve conservation of momentum by applying Newton law $\sum F = ma$, to a given fluid particle. We assume that the external forces exerted on a point in the particle $\boldsymbol{x}$, at time t by the fluid can be described by a density function $f_{ext}(x,t)$. The sum of applied forces is equal to

$$\rho a(x,t)dv = (\nabla \cdot \sigma + f_{ext})dv \tag{5.12}$$

We aim to determine the acceleration of the particle denoted by a, which is equal to

$$a(x,t) = \frac{D\bar{v}}{Dt}(x,t). \tag{5.13}$$

The by computing the total derivative of velocity we obtain

$$\frac{D\bar{v}}{Dt}(x,t) = \frac{\partial \bar{v}}{\partial t} + (\bar{v} \cdot \nabla)\bar{v}. \tag{5.14}$$

Applying Newtown's law, equation (5.12), to our particle yields

$$\rho\left(\frac{\partial \bar{v}}{\partial t} + (\bar{v} \cdot \nabla)\bar{v}\right) dv = (\nabla \cdot \sigma + f_{ext}) dv \tag{5.15}$$

Since the local volume dv, is not zero we can divide each side and determine the momentum equation

$$\rho\left(\frac{\partial \bar{v}}{\partial t} + (\bar{v} \cdot \nabla)\bar{v}\right) = \nabla \cdot \sigma + f_{ext}. \tag{5.16}$$

The stress tensor σ, is often determined by experiments and is often convenient to split the stress tensor as

$$\sigma = \mathbb{D} - p\mathbb{I}. \tag{5.17}$$

The expression is composed of the dynamic pressure p and $\mathbb{D}$ the deviatoric part of σ. Furthermore, since the stress tensor is symmetric, and through experimentation [42], the deviatoric component is $\mathbb{D} = 2\mu Du$. Consequently, this will alternate the expression of the momentum equation to be

$$\rho\left(\frac{\partial \bar{v}}{\partial t} + (\bar{v} \cdot \nabla)\bar{v}\right) = \nabla \cdot (2\mu Du - p\mathbb{I}) + f_{ext}. \tag{5.18}$$

For incompressible flow, we assume that density is constant, $\rho = \rho_o$. When we divide the momentum equation by density, we will denote the ratio p/ρ by p. This ratio is better known as the density of pressure per unit of mass and volume. We will also the denote the ratio of f_{ext}/ρ as f, which is still called the external force. We can then take the momentum and mass conservation equation while noting that $\nabla \cdot (p\mathbb{I}) = \nabla p$, to derive the Navier Stokes equation.

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla \vec{v} = -\nabla p + \nabla \cdot (2\upsilon D\vec{v}) + f \tag{5.19}$$

This is the governing equation of fluid motion which we will use to conduct our CFD simulations.

Chapter 6: Computational Fluid Dynamics for Laser Propagation

Computational fluid dynamic (CFD) is the numerical simulation of fluid motion. The approach is geared toward modeling turbulence at a degree of accuracy to determine the coupled fluid dynamic behaviors. There are various numerical models that have been developed over the years to describe a wide variety of fluid dynamic phenomenon. They range from least to best in accuracy when simulating turbulent flow, but computational requirements can restraint researchers from using the most sophisticated models.

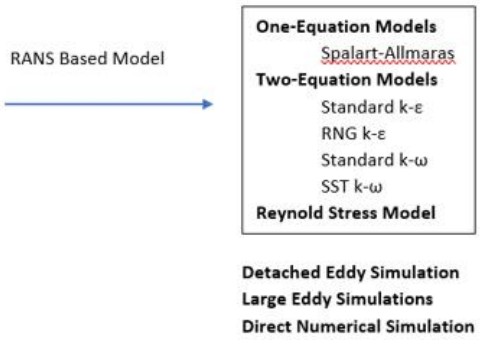


Figure 6.1: Various Models of CFD

The most accurate approach for simulating turbulent flows is the Direct Numerical Simulation (DNS) model in which the full Navier-Stokes equation is numerically solved. There is no modeling involved as it is solved directly by a finite mesh to capture all the scales that are present in a turbulent flow. Therefore, DNS is computationally prohibitive and is not the most commonly used method to simulate turbulent flow. In the other hand for modest computing requirement, the Reynolds-averaged Navier-Stokes (RANS) approach solves the ensemble-average of the Navier-Stokes equations. This approach requires extensive modeling to model the effect of all the scales of instantaneous turbulent motion. It is also the least accurate approach when trying to model turbulent flow.

An alternative approach would be Large-Eddy Simulations (LES) which directly solves the large-scale motions in turbulent flow while the small-scale motions are modeled. This results in a significant reduction of computational cost and does not adopt the conventional ensemble averaging that is used in the RANS approach. LES is more accurate than the RANS approach by directly solving the large eddies which are responsible for the momentum transfer and contain the most energy in a turbulent flow. The smaller eddies, or Sub-Grid Scale (SGS), are assumed to be more isotropic and therefore easier to model. Therefore, LES is a promising method of accurately modeling turbulent flow with reasonable computing power.

6.1 FINITE VOLUME METHOD

To evaluate the partial differential equations in fluid motion we need to implement the finite volume method. Let's consider the governing equations for a two-dimensional, steady flow

$$\frac{\partial}{\partial x}(\rho uu) + \frac{\partial}{\partial y}(\rho vu) = \frac{\partial}{\partial x}\left(\mu \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right) + S_u \tag{6.1}$$

$$\frac{\partial}{\partial x}(\rho uv) + \frac{\partial}{\partial y}(\rho vv) = \frac{\partial}{\partial x}\left(\mu \frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu \frac{\partial v}{\partial y}\right) + S_v \tag{6.2}$$

$$\frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0 \tag{6.3}$$

where $S_u = -\partial p/\partial x$ and $S_v = -\partial p/\partial y$ and they are the pressure gradient terms which is what we initially try to solve. There are complications with solving the governing equations because the convective terms contain non-linear quantities and all three equations are coupled with each other. We would initially solve for the pressure gradient by implementing a checker-board pressure field grid, but this results in nonrealistic representation of the flow. To solve this problem we need to implement a staggered grid for the velocity components [43].

The idea is to solve for scaler variables at ordinary nodal points like before but to calculate velocity components on staggered grids centered around the cell faces. H. K. Versteeg and W. Malalasekera can determine the coefficients for pressure-velocity coupling to be [44]

$$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} + \left(p_{I-1,J}p_{I,J}\right)A_{i,J} + b_{i,J} \tag{6.4}$$

$$a_{I,j}v_{I,j} = \sum a_{nb}v_{nb} + \left(p_{I,J-1}p_{I,J}\right)A_{I,j} + b_{I,j} \tag{6.5}$$

where $b_{i,J} = \bar{S}\Delta V_u$ the momentum source term, and $A_{i,J}$ is the cell face area. Coefficients $a_{i,J}$ and a_{nb} are calculated through any differencing methods suitable for convection-diffusion problem. The unbroken grid lines are numbered by capital letters and the dashed lines are noted by lowercase letters. Scalar nodes are located at the intersection of two grid lines and identified by two capital letters. Velocity components are located at the intersection of a line defining a cell boundary and grid line which is why they noted by both lowercase and capital notation. Then we can continue to determine coefficients for the convective flux per unit mass F and diffusive conductance D at u and v control volume faces [44].

u-control volume	v-control volume	
$F_w = (\rho u)_w = \frac{F_{i,J} + F_{i-1,J}}{2}$	$F_w = (\rho u)_w = \frac{F_{i,J} + F_{i,J-1}}{2}$	(6.6)
$F_e = (\rho u)_e = \frac{F_{i+1,J} + F_{i,J}}{2}$	$F_e = (\rho u)_e = \frac{F_{i+1,J} + F_{i+1,J-1}}{2}$	(6.7)
$F_s = (\rho v)_s = \frac{F_{I,j} + F_{I-1,j}}{2}$	$F_s = (\rho v)_s = \frac{F_{I,j-1} + F_{I,j}}{2}$	(6.8)
$F_n = (\rho v)_n = \frac{F_{I,j+1} + F_{I-1,j+1}}{2}$	$F_n = (\rho v)_n = \frac{F_{I,j} + F_{I,j+1}}{2}$	(6.9)
$D_w = \frac{\Gamma_{I-1,J}}{x_i - x_{i-1}}$	$D_w = \frac{\Gamma_{I-1,J-1} + \Gamma_{I,J-1} + \Gamma_{I-1,J} + \Gamma_{I,J}}{4(x_I - x_{I-1})}$	(6.10)
$D_e = \frac{\Gamma_{I,J}}{x_{i+1} - x_i}$	$D_e = \frac{\Gamma_{I,J-1} + \Gamma_{I+1,J-1} + \Gamma_{I,J} + \Gamma_{I+1,J}}{4(x_{I+1} - x_I)}$	(6.11)
$D_s = \frac{\Gamma_{I-1,J} + \Gamma_{I,J} + \Gamma_{I-1,J-1} + \Gamma_{I,J-1}}{4(y_J - y_{J-1})}$	$D_s = \frac{\Gamma_{I,J-1}}{y_{jk} - y_{j-1}}$	(6.12)
$D_n = \frac{\Gamma_{I-1,J+1} + \Gamma_{I,J+1} + \Gamma_{I-1,J} + \Gamma_{I,J}}{4(y_{J+1} - y_J)}$	$D_n = \frac{\Gamma_{I,J}}{y_{j+1} - y_j}$	(6.13)

To solve for the velocity and pressure for a given control volume we need to implement an algorithm presented by Patankar and Spalding [45]. Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) begins by guessing the pressure field. The initial guess will be implemented onto the discretized momentum equation (6.4) and equation (6.5) so solve for the velocity components noted by (*).

$$a_{i,J}u^*_{i,J} = \sum a_{nb}u^*_{nb} + \left(p^*_{I-1,J} - p^*_{I,J}\right)A_{i,J} + b_{i,J} \tag{6.14}$$

$$a_{I,j}v^*_{I,j} = \sum a_{nb}v^*_{nb} + \left(p^*_{I,J-1} - p^*_{I,J}\right)A_{I,j} + b_{I,j} \tag{6.15}$$

Now we have to define a correction parameter noted by (‘) which is the difference between the correct parameter and the guessed parameter.

$$p' = p - p^* \qquad u' = u - u^* \qquad v' = v - v^* \tag{6.16}$$

Subtracting the momentum equation with the correct pressure field by the discretized equation with the guessed pressure field yields the correction values. Then the main approximation for the SIMPLE algorithm is implemented by dropping the summation value in the discretized equation for the correction velocity fields,

$$u'_{i,J} = \left(p'_{I-1,J} - p'_{I,J}\right)d_{i,J} \tag{6.17}$$

$$v'_{I,j} = \left(p'_{I,J-1} - p'_{I,J}\right)d_{I,j} \tag{6.18}$$

where $d_{i,J} = \frac{A_{i,J}}{a_{i,J}}$ and $d_{I,j} = \frac{A_{I,j}}{a_{I,j}}$. Now we can use the correction values to solve for equation. (6.17) and 5.(18).

$$u_{i,J} = u^*_{i,J} + d_{i,J}\left(p'_{I-1,J} - p'_{I,J}\right) \tag{6.19}$$

$$v_{I,j} = v^*_{I,j} + d_{I,j}\left(p'_{I,J-1} - p'_{I,J}\right) \tag{6.20}$$

There is a similar expression exist for $u_{i+1,J}$ and $v_{I,j+1}$ that contains the similar form as equation (6.14) and equation (6.15). Now we need to consider the constraints in the velocity field that satisfy the continuity equation from equation (6.3).

$$\left[(\rho uA)_{i+1,J} - (\rho uA)_{i,J}\right] + \left[(\rho vA)_{I,j+1} - (\rho vA)_{I,j}\right] = 0 \tag{6.21}$$

Substitution of the corrected velocities onto the discretized continuity equation gives

$$a_{I,J}p'_{I,J} = a_{I+1,J}p'_{I+1,J} + a_{I-1,J}p'_{I-1,J} + a_{I,J+1}p'_{I,J+1} + a_{I,J-1}p'_{I,J-1} + b'_{I,J} \tag{6.22}$$

where the coefficients are given below.

Table 6.1: Expanded coefficients for discretized continuity equation

$a_{I,J}$	$a_{I+1,J}$	$a_{I-1,J}$	$a_{I,J+1}$		$a_{I,J-1}$	$b'_{I,J}$
$a_{I+1,J}$ $+ a_{I-1,J}$ $+ a_{I,J+1}$ $+ a_{I,J-1}$	$(\rho dA)_{i+1,J}$	$(\rho dA)_{i,J}$	$(\rho dA)_{I,j+1}$		$(\rho uA)_{I,j}$	$(\rho u^*A)_{i,J}$ $- (\rho u^*A)_{i+1,J}$ $+ (\rho v^*A)_{I,j}$ $- (\rho v^*A)_{I,j+1}$

By solving the equation, we can determine the correction field p' at all points and use it to determine the correct pressure field p. The pressure correction equation diverges unless some under-relaxation term is used in the iterative process [44].

$$p^{new} = p^* + \alpha_p p' \tag{6.23}$$

The pressure under-relaxation factor is α_p and ranges between zero and one. Taking $0 < \alpha_p < 1$ allows us to add to the guessed field p^* by a fraction of the correction field p' that will allow us to move the iterative process forward. The velocity components are also under-relaxed.

$$u^{new} = \alpha_u u + (1 - \alpha_u) u^{n-1} \tag{6.24}$$

$$v^{new} = \alpha_v v + (1 - \alpha_v) v^{n-1} \tag{6.25}$$

The velocity under-relaxation factor is α_u and α_v for u- and v-velocity. The corrected velocity components without relaxation are u and v. The values obtained in the previous iteration are represented by u^{n-1} and v^{n-1}. Optimum values for the under-relaxation parameters are dependent on the flow and must be determined for each individual problem. The SIMPLE algorithm provides a method for calculating pressure and velocities. The method is iterative, and when other scalars are coupled to the momentum equations the calculation needs to be done sequentially [44].

6.2 LARGE EDDY SIMULATION

In LES simulations the large-scale motions are represented directly, and the smaller scales are modeled through sub-grid scale modeling. The velocity component u is split between the filtered component $\bar{u}$ and a residual subgrid-scale component $\acute{u} = u - \bar{u}$ [46]. Then we derive the governing equations employed in LES from a filtered Navier Stokes equation to compute the evolution of the filtered velocity field. This filtered Navier Stokes equation has the same form as the unfiltered equation except for a residual stress tensor arising from the residual motions. This

residual stress tensor must be modeled by the various sub-grid scale models. Then the filtered equation can then be solved numerically for the filtered velocity.

The filtering operator is defined as

$$\bar{u}(x,t) = \int_D G_\Delta(r,x)u(x-r,t)dr \tag{6.26}$$

where D is the fluid domain and G_Δ is the filter function that determines the scale of the resolved eddies. The filter is called uniform if it does not depend on x, and isotropic if it depends only on the displacement vector r. The are many filter functions with varying properties and is dependent on what effects is needed from the filtered continuity equation. In Ansys Fluent, the filter function filters out the eddies whose scales are smaller than the filter width or grid spacing [47]. This simplifies the integration over the flow domain to be

$$\bar{u} = \frac{1}{V}\int_v u(x-r,t)dr, \quad r \epsilon v \tag{6.27}$$

where V is the volume of the computational cell. This will help us resolve the filtered Navier Stokes equation to be

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}.(\rho\overline{\boldsymbol{u_i}}) = 0 \tag{6.28}$$

$$\frac{\partial(\rho\overline{\boldsymbol{u_i}})}{\partial t} + \boldsymbol{\nabla}.\left(\rho\overline{\boldsymbol{u_i}}\,\overline{\boldsymbol{u_j}}\right) = -\boldsymbol{\nabla}\bar{p} + \boldsymbol{\nabla\sigma_{ij}} + \boldsymbol{\nabla\tau_{ij}} \tag{6.29}$$

where $\sigma_{ij} = \left[\mu\left(\nabla\overline{u_\iota} + \nabla\overline{u_j}\right)\right] - \frac{2}{3}\mu\nabla\overline{u_l}\delta_{ij}$ is the stress tensor due to molecular viscosity and $\tau_{ij} = \rho\overline{\boldsymbol{u_\iota u_j}} - \rho\overline{\boldsymbol{u_\iota}}\,\overline{\boldsymbol{u_j}}$ is the subgrid-scale stress. The next step will be deciding which sub-grid scale model will be used to derive the residual stress.

6.3 WALL-ADAPTING LOCAL EDDY-VISCOSITY (WALE) MODEL

In LES scales smaller than the smaller than the grid size are not directly resolved but are represented by the subgrid-scale stress. Most sub-grid scale modeling are based on the Boussinesq's eddy viscosity assumption [48] to model the subgrid scale tensor.

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\mu_t\overline{S_{\iota J}} \tag{6.30}$$

μ_t is the subgrid-scale turbulent viscosity and $\overline{S_{\iota J}} = \frac{1}{2}\left(\nabla\overline{u_\iota} + \nabla\overline{u_j}\right)$ is the rate-of-strain tensor for the resolved scale. In various sub-grid scales these components are modeled differently for their respected operation or goal. In the WALE model [47] the eddy viscosity is modeled by

$$\mu_t = \rho L_s^2 \frac{\left(S_{ij}^d S_{ij}^d\right)^{3/2}}{\left(\overline{S_{\iota J}}\,\overline{S_{\iota J}}\right)^{5/2} + \left(S_{ij}^d S_{ij}^d\right)^{5/4}} \tag{6.31}$$

where $L_s = \min(\kappa d, C_w V^{\frac{1}{3}})$ where C_w is 0.325 which has been found to yield good results for wide range of flow [47] .This method of modeling the eddy viscosity allows for proper behavior near the boundary walls since we are assuming that the cutoff length plays no important role in its behavior [49]. In this model as the sub-grid scale stress approaches the wall it should show behavior of order $O(y^3)$ if y is the direction normal to the wall. However, the behavior of $\overline{S_{\iota J}}$ is of order $O(1)$ in the same limit, which results in defining a new operator S_{ij}^d.

$$S_{ij}^{d} = \frac{1}{2}\left(\bar{g}_{ij}^{2} + \bar{g}_{ji}^{2}\right) - \frac{1}{3}\delta_{ij}\bar{g}_{kk}^{2} \tag{6.32}$$

In this case S_{ij}^{d} is based on the traceless symmetric part of the square gradient velocity tensor $\bar{g}_{ij}^{2} = \frac{\partial u_i}{\partial x_j}$ [49]. This method of modeling the sub-grid scale should provide a local eddy-viscosity, able to switch off during the early stages of transition and offer a proper wall scaling to get a good prediction of friction coefficient.

6.4 Implementing LES on Current Numerical Simulation

To generate atmospheric turbulence over long distance laser propagation, a simplified 3D geometry was generated. This 3D model mimics the elevation profile along the propagation path from the Mauna Loa NOAA observatory to the AFRL AEOS telescope on Haleakala. These summits are the U.S. Air Force sites for conducting the optical turbulence studies [50]. The Maui Space Surveillance System (MSSS) is an electro-optical facility located at the summit of Haleakala an elevation of 3.055 km. The NOAA observatory located on top of the Mauna Loa summit reaches an elevation of approximately 3.397 km. The distance between both observatories reaches approximately 150 km length. Due to computational restriction, we were limited by the simplified geometry dimensions and made it as large as possible with a very coarse mesh to obtain a numerical simulation of fluid motion.

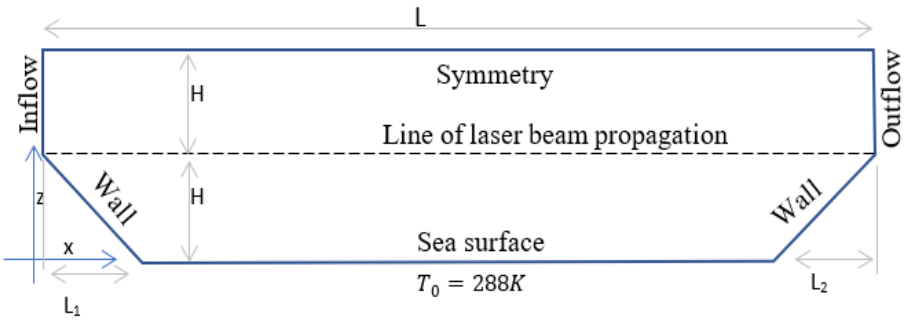


Figure 6.2: Schematic of computational domain

Table 6.2 Geometry of computational domain and inlet velocity

Geometry	Velocity
H=3m, L=150m, L1=10m, L2=20m, W=0.1m	4 m/s

Both observatories are assumed to be on the same horizontal axis 3 km above sea level. The dimensions L_1 and L_2 is an approximate distance of mountain peak to the beach. The inflow, outflow and top face surface are at an ambient air temperature of 298 K while the bottom face (sea level) is 288 K. The desired geometry was reduced by a factor of 10^3m due to computational restraints. The remaining of the study will analyze the LES of the geometry model to see if it is possible to determine C_n^2 values to character atmospheric turbulence effects on laser propagation.

To be able to determine C_n^2, we need to keep in mind that the variation in the refractive index is proportional to the variation in potential temperature and velocity. This allows us to relate the refractive-index constant, C_n^2, to the temperature structure constant C_T^2.

$$C_n^2 = \left(\frac{79 \times 10^{-8}\, P}{T^2}\right)^2 C_T^2 \tag{6.33}$$

where *P* is atmospheric pressure, *T* is air temperature. Values for temperature and pressure depend on the mean atmospheric temperature and pressure along the propagation path. C_T^2 is the structure constant parameter for temperature and can be calculated as

$$C_T^2 = a^2 \left(\frac{K_H}{K_M}\right) L_o^{\frac{4}{3}} \left(\frac{\partial \theta}{\partial x}\right)^2 . \tag{6.34}$$

In this relation a^2 is an empirical constant, K_H and K_m are the exchange coefficients for heat and momentum, $L_o$is the outer length scale of turbulence, and $\frac{\partial \theta}{\partial x}$ is the gradient of potential temperature. D. L. Walters and D. K. Miller [51], [52] determined that the empirical constant to be 2.8. Masciadri, and E.Jabouille deduced that for a thermal and dynamic stability of the atmosphere, $\frac{K_H}{K_M}$ value can be assumed to be 0.7 [53]. The potential temperature, θ, is the temperature at an air parcel would have if it were expanded or compressed adiabatically to a pressure of 100 kPa.

$$\theta = T \left(\frac{10^5}{P}\right)^{0.286} \tag{6.35}$$

T is the static temperature and P is the absolute pressure in Pa over the propagation length. Deardorff [54] calculated the outer scale length in thermally stable conditions to be

$$L_o = 0.76\ \frac{\sqrt{k}}{N} \tag{6.36}$$

where N is the Brunt-Vaisala frequency and k is the turbulent kinetic energy. The Brunt-Vaisala frequency is also known as the buoyancy frequency. This frequency is associated with the oscillation caused by small density differences in a statistically stable environment.

$$N = \sqrt{-\frac{g}{\rho}\frac{\partial \rho}{\partial z}} \tag{6.37}$$

In atmospheric dynamics, the density difference is related to the potential temperature and hence the Brunt-Vaisala frequency can be given as

$$N = \sqrt{\frac{g}{\theta}\left|\frac{\partial \theta}{\partial z}\right|} \tag{6.38}$$

C_n^2 can ultimately be simplified as

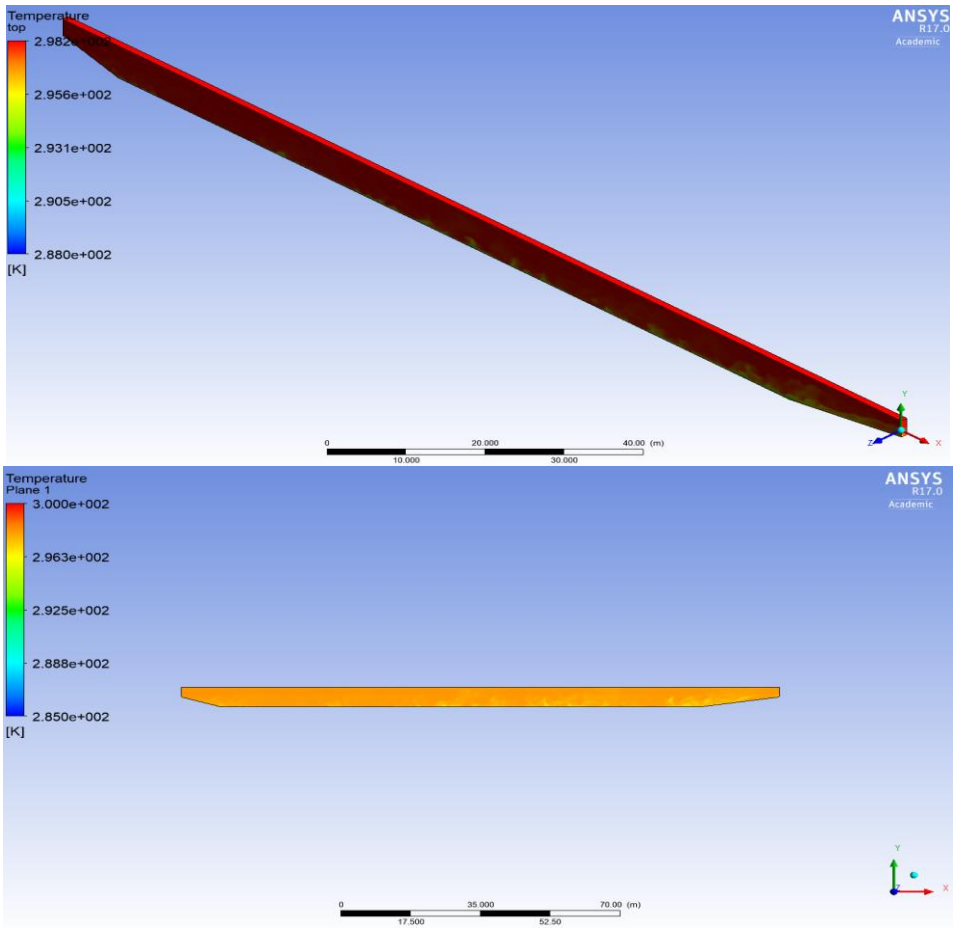
$$C_n^2 = 3.35 \times 10^{-6} P^{0.857}\ \theta^{-\frac{10}{3}} \left(\frac{\partial \theta}{\partial x}\right)^{\frac{4}{3}} (k)^{\frac{2}{3}} \tag{6.39}$$

In this case k angular spatial frequency. The derivation states that the optical refractive index fluctuations are cause by variations in the temperature in the atmosphere. It should be noted that C_n^2 is a variable that describes the strength of the refractive index turbulence. If we can model the

geometry that a laser will propagate and apply the appropriate boundary conditions, we can determine the structure function for laser propagation. This will then in return allow us to determine other optical parameters that help characterize atmospheric turbulence such as the Rytov Index.

6.5 Calculating for Structure Function Constant

Once we get our results of our 3D model, we only need to attain the temperature and pressure profile at the line of sight.



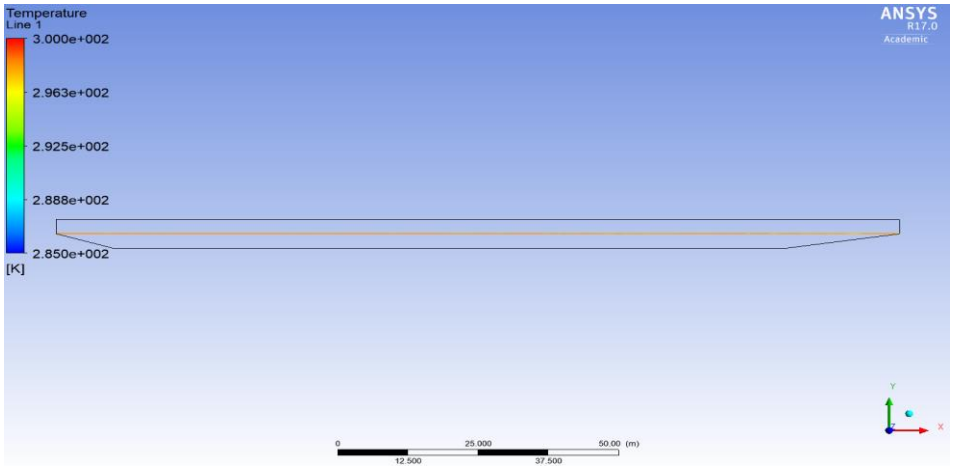


Figure 6.3: Temperature profile of our simplified 3D model

We then use equation (6.39) so solve the structure function constant as a function of atmospheric pressure and air temperature at time, $t = 200$, $t = 400$, and $t = 600$.

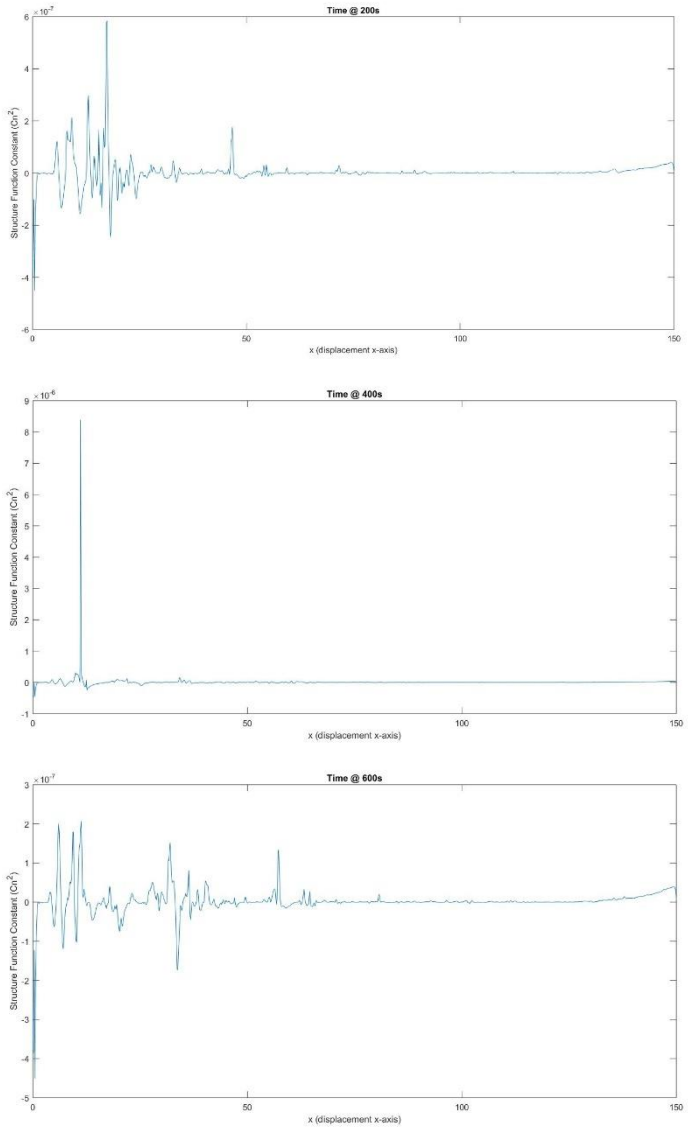


Figure 6.4: Calculations of the Structure Function Constant at, $t = 200$, $t = 400$, and $t = 600$

The average C_n^2 at the line of sight at $t = 200$ is 3.3335e-09, at $t = 400$ is 1.1023e-08, and at $t = 600$ is 2.6862e-09. These values are extremely high for these conditions and there must be some error in our current modeling. Since we mainly use the Rytov index to represent atmospheric turbulence we can calculate C_n^2, from a range of $0.1 \leq \sigma_\chi^2 \leq 1.0$. With the initial conditions, we can use equation (3.1) and determine that

within this range the structure constant should be within $1.8248 * 10^{-13} \leq C_n^2 \leq 1.8248 * 10^{-12}$. We seem to be off significantly, and further revision needs to be made to the model and the boundary conditions given.

If we are then able to model the actual propagation path between Mauna Loa NOAA observatory to the AFRL AEOS telescope on Haleakala we can then compare with experimental results. The are various publications that we can use to compare our simulated results with [50], [55]. However, due to the decrease in geometry, the Large Eddy Simulation is being compared with the current numerical simulation and theoretical approximation. Future work would include monitoring our model alongside previous and current experimental results of the propagational path.

Chapter 7: Putting it All Together

In this study, we analyzed atmospheric effects for a laser propagating a long distance through our developed phase screen simulator. We found that metrics based on the variance of intensity and number of zero intensity value saturate at high turbulence, but phase variance does not. Since the phase variance is dependent mainly on the largest eddies, we can make estimates of the outer scale length by determining the phase variance. However, we see that our values of phase variance fluctuate for $\sigma_\chi^2 \geq 0.5$. This may be due to the 2D Goldstein algorithm breaking down at high levels of atmospheric turbulence. To be more accurate, further investigation is needed at higher grid size and numbers of trials.

We also attempted to accelerate the numerical simulation by implementing a conditional generative adversarial network (cGAN) that applies phase unwrapping to a wrapped phase screen. The cGan framework establishes competition between two distinct players in the model, the discriminator and the generator. The network is trained on 512x512 wrapped and unwrapped images and learns the mapping to conduct phase unwrapping. We achieved reasonable results of generated images that differ slightly from the target images. However, we do run into complications when we tried to conduct phase unwrapping on the float matrices themselves. The images do not seem to be far off, but slight checkerboard artifacts are apparent in the images. We even attempted to improve our current architecture by using a different up-sampling method but did not generate better results. More in-depth understanding of neural networks and adjusting certain parameters could result in a working network. This will allow us to not only

conduct faster simulations but use larger grid size without much computational cost. This can in return allow us to study phase fluctuations at $\sigma_\chi^2 \geq 0.5$.

We would need to implement high-performance computing to run our phase screen simulator. This will allow us to simulate a spatial grid size that is significantly larger than the outer scale length so the sample frequency spacings can accurately represent outer scale effects. It can also help shorten the time of each simulation and include a greater number of trials to reduce any discrepancies. We have also generated a simple 3D model of the propagation path from the Mauna Loa NOAA observatory to the AFRL AEOS telescope on Haleakala. We do have licensing limitations and had to factor our model by 10^3m. With the given boundary conditions and our derived formulation of the structure-function constant our simulation values over exceed the theoretical. More work is required for improving the simplified model and changing some of the boundary conditions. If improved, we can begin to integrate our CFD simulation of the structure-function constant in our phase screen simulator.

The whole point of this study is to be able to code a numerical simulation of wave propagation for long-range optical systems. We will be able to analytically understand the effects without having to conduct the actual test themselves. This is beneficial since actual testing is not only economically expensive but requires a lot of time and manpower. We want to include our CFD simulation to our current numerical simulation to get an actual representation of C_n^2 across the propagation length instead of a constant C_n^2. More importantly, we want to overcome the fluctuations cause by the 2D Goldstein Algorithm. We can improve on our current neural networks, but we can also increase the grid size by using high-performance computing. Once the

numerical simulation model works as needed, we can begin to characterize the effects of deep atmospheric turbulence on laser propagation over a long path.

References

[1] A. Wheelon, *Electromagnetic Scintillation I. Geometrical Optics*, 1st ed. New York: Cambridge University Press, 2001.

[2] J. M. Mcdonough, "LECTURES IN ELEMENTARY FLUID DYNAMICS : Physics , Mathematics and Applications," *Analysis*, p. 163, 2009.

[3] A. Kolmogorov, *In Turbulence, Classic Papers on Statistical Theory*. New York: Interscience, 1961.

[4] G. Weiss, "Wave Propagation in a Turbulent Medium. V. I. Tatarski. Translated by R. A. Silverman. McGraw-Hill, New York, 1961. 285 pp. Illus. $9.75," *Science (80-.).*, vol. 134, no. 3475, pp. 324–325, Aug. 1961.

[5] V. I. Tatarskii, *The Effects of the Turbulent Atmosphere on Wave Propagation*. Springfield: National Technical Information Service U.S. Dept. of Commerce, 1971.

[6] J. W. Strohbehn, "Line-of-sight wave propagation through the turbulent atmosphere," *Proc. IEEE*, vol. 56, no. 8, pp. 1301–1318, 1968.

[7] A. Quirrenbach, "The Effects of Atmospheric Turbulence on Astronomical Observations," *Exp. Astron.*, 2009.

[8] L. C. Andrews and R. L. Phillips, *Laser Beam Propagation through Random Media*. 1000 20th Street, Bellingham, WA 98227-0010 USA: SPIE, 2005.

[9] L. A. Chernov, *Wave propagation in a random medium. Translated from the Russian by R. A. Silverman*. New York: McGraw-Hill, 1960.

[10] R. L. Phillips and L. C. Andrews, "Measured statistics of laser-light scattering in atmospheric turbulence," *J. Opt. Soc. Am.*, vol. 71, no. 12, p. 1440, 1981.

[11] A. Consortini, F. Cochetti, J. H. Churnside, and R. J. Hill, "Inner-scale effect on irradiance variance measured for weak-to-strong atmospheric scintillation," *J. Opt. Soc. Am. A*, vol. 10, no. 11, p. 2354, 1993.

[12] D. Dayton, B. Pierson, B. Spielbusch, and J. Gonglewski, "Atmospheric structure function measurements with a Shack-Hartmann wave-front sensor," *Opt. Lett.*, vol. 17, no. 24, p. 1737, 1992.

[13] R. Kerr, "Experiments on Turbulence Characteristics and Multiwavelength Scintillation Phenomena," *J. Opt. Soc. Am.*, vol. 62, no. 9, pp. 1040–1049, 1972.

[14] R. S. Lawrence, G. R. Ochs, and S. F. Clifford, "Measurements of atmopsheric turbulence relevant to optical propagation," *J. Opt. Soc. Am.*, vol. 60, no. 6, pp. 826–830, 1970.

[15] E. Velar, J. Haddon, and J. Haddon, "Measurement and Modeling of Scintillation Intensity

to Estimate Turbulence Parameters in an Earth-Space Path," *IEEE Trans. Antennas Propag.*, vol. 32, no. 4, pp. 340–346, 1984.

[16] A. Zilberman, E. Golbraikh, and N. S. Kopeika, "Propagation of electromagnetic waves in Kolmogorov and non-Kolmogorov atmospheric turbulence: three-layer altitude model.," *Appl. Opt.*, vol. 47, no. 34, pp. 6385–91, 2008.

[17] J. P. Bos, M. C. Roggemann, and V. S. Rao Gudimetla, "Anisotropic non-Kolmogorov turbulence phase screens with variable orientation," *Appl. Opt.*, vol. 54, no. 8, p. 2039, 2015.

[18] B. E. Stribling, "Optical propagation in non-Kolmogorov atmospheric turbulence," in *Proceedings of SPIE*, 1995, vol. 2471, pp. 181–196.

[19] J. D. Schmidt, *Numerical Simulation of Optical Wave Propagation with Examples in MATLAB*. 2010.

[20] C. Smith, "GoldsteinUnwrap2D_r1." MATLAB 7.11 (R2010b), 2010.

[21] V. S. R. Gudimetla, R. B. Holmes, and J. F. Riker, "Analytical expressions for the log-amplitude correlation function for plane wave propagation in anisotropic non-Kolmogorov refractive turbulence," *J. Opt. Soc. Am. A*, vol. 29, no. 12, p. 2622, 2012.

[22] V. S. Gudimetla, "Estimating Cn2 (Refractive index structure constant) in Deep Turbulence using Unwrapped Phase," in *Imaging and Applied Optics*, 2013, p. PTu2F.1.

[23] R. S. Michalski, I. Bratko, and A. Bratko, Eds., *Machine Learning and Data Mining; Methods and Applications*. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[24] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "GeePS : Scalable deep learning on distributed GPUs with a GPU-specialized parameter server."

[25] T. Chen *et al.*, "MXNet : A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," pp. 1–6.

[26] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems."

[27] M. Isard, A. Birrell, and D. Fetterly, "Dryad : Distributed Data-Parallel Programs from Sequential Building Blocks," 2007.

[28] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," pp. 1–13.

[29] J. Dean *et al.*, "Large Scale Distributed Deep Networks," pp. 1–11.

[30] C. Bishop and N. Nasrabadi, "Pattern recognition and machine learning," *Pattern Recognit.*, vol. 4, no. 4, p. 738, 2006.

[31] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Miccai*, 2015.

[32] H. Dong, G. Yang, F. Liu, Y. Mo, and Y. Guo, “Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks,” in *Medical Image Understanding and Analysis: 21st Annual Conference, MIUA 2017, Edinburgh, UK, July 11–13, 2017, Proceedings*, 2017.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Alexnet,” *Adv. Neural Inf. Process. Syst.*, 2012.

[34] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[35] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[36] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context Encoders: Feature Learning by Inpainting,” 2016.

[37] I. Goodfellow *et al.*, “Generative Adversarial Nets,” *Adv. Neural Inf. Process. Syst. 27*, 2014.

[38] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[39] M. Drozdzal, E. Vorontsov, G. Chartrand, and C. V Sep, “The Importance of Skip Connections in Biomedical Image Segmentation.”

[40] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved Techniques for Training GANs,” pp. 1–10, 2016.

[41] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and Checkerboard Artifacts,” *Distill*, vol. 1, no. 10, Oct. 2016.

[42] T. Chacón Rebollo and R. Lewandowski, *Mathematical and Numerical Foundations of Turbulence Models and Applications*. New York, NY: Springer New York, 2014.

[43] F. H. Harlow and J. E. Welch, “Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface,” *Phys. Fluids*, vol. 8, no. 12, p. 2182, 1965.

[44] H. K. Versteeg and W. Malaskekera, *An Introduction to Computational Fluid Dynamics*, vol. M. 2007.

[45] S. V. Patankar and D. B. Spalding, “A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows,” *Int. J. Heat Mass Transf.*, vol. 15, no. 10, pp. 1787–1806, 1972.

[46] Y. Zhiyin, “Large-eddy simulation: Past, present and the future,” *Chinese J. Aeronaut.*, vol. 28, no. 1, pp. 11–24, 2015.

[47] Ansys, “Ansys fluent 12.0 Theory Guide,” *ANSYS Inc.*, no. April, pp. 49–53, 2009.

[48] F. G. Schmitt, “About Boussinesq’s turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity,” *Comptes Rendus - Mecanique*, vol. 335, no. 9–10. pp. 617–627, 2007.

[49] F. Ducros, F. Nicoud, and T. Poinsot, “Wall-adapting local eddy-viscosity models for simulations in complex geometries,” *Conf. Numer. Methods Fluid Dyn.*, pp. 1–7, 1998.

[50] M. Vorontsov *et al.*, “Characterization of Atmospheric Turbulence Effects Over 149 km Propagation Path Using Multi-Wavelength Laser Beacons,” Proc. AMOS Conference, 2010.

[51] D. L. Walters and D. K. Miller, “Evolution of an upper-tropospheric turbulence event–Comparison of observations to numerical simulations,” in *Preprints, 13th Symposium on Boundary Layer Turbulence*, 1999, pp. 157–160.

[52] S. Bermon and C. K. So, “Determination of the kondo scattering amplitude and coherence distance from tunneling in doped-electrode junctions,” *Phys. Rev. Lett.*, vol. 40, no. 1, pp. 53–56, 1978.

[53] E. Masciadri and P. Jabouille, “Improvements in the optical turbulence parameterization for 3D simulations in a region around a telescope,” *Astron. Astrophys.*, vol. 376, no. 2, pp. 727–734, 2001.

[54] J. W. Deardorff, “Stratocumulus-capped mixed layers derived from a three-dimensional model,” *Boundary-Layer Meteorol.*, vol. 18, no. 4, pp. 495–527, 1980.

[55] M. Vorontsov *et al.*, “Comparison of turbulence-induced scintillations for multi-wavelength laser beacons over tactical (7 km) and long (149 km) atmospheric propagation paths.” Proc. AMOS Conference, 2011

Vita

I am Diego Alberto Lozano Jimenez and I am pursuing a master's in mechanical engineering at the University of Texas at El Paso. Under my faculty advisor Dr. Vinod Kumar, I interned as a graduate research assistant for the AFRL Summer Faculty Fellowship Program. During this time, I had to understand the effect of atmospheric turbulence in the field of Directed Energy and laser systems. With the limited time I was given I was able to simulate long distance laser propagation under atmospheric turbulence through the Kolmogorov and non-Kolmogorov models. With this current numerical simulation our goal is to characterize the effects of atmospheric turbulence on long range laser propagation. My task was to determine the variance, covariance, auto-correlation and structure function of the phase fluctuations.

The following year I continued my research by implementing Machine Learning to our current laser simulation. To accelerate simulation, we implement a conditional generative adversarial network (cGAN) that applies phase unwrapping to a wrapped phase screen. Additional work is required to conduct phase unwrapping on 4096x4096 wrapped images with our currently trained model.

I am finishing my research by studying the impact of strong atmospheric turbulence in spatial, temporal, and related spectral domains will be examined using Large Eddy Simulation (LES) turbulence modeling. LES turbulence modeling allows us to study strong fluid turbulence and can predict advection-dissipation at various energy spectrums. I have had the opportunity to also submit two publication under AMOS and SPIE conferences. Hopefully I will continue working in research and development for the Department of Defense or the Department of Energy.

This thesis/dissertation was typed by Diego Alberto Lozano Jimenez.