A TRIDENT SCHOLAR PROJECT REPORT

NO. 462

Modeling the Effects of Meteorological Conditions on the Neutron Flux

by

Midshipman 1/C Thomas J. Wilson, USN

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
05-22-17		

4. TITLE AND SUBTITLE
Modeling the Effects of Meteorological Conditions on the Neutron Flux

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Thomas J. Wilson

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Report no. 462 (2017)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The neutron background at sea level is seen to vary by as much as 20% within a 24 hour period. These short term variations are primarily driven by environmental factors. Radiation sensors detect a signal in a noisy background, so the variation of the background must be understood for sensors to be effective. Neutron sensors are used in the search for transported materials that can be used to make a nuclear weapon. The purpose of this project was to develop a statistical model that predicts environmental neutron background as a function of five meteorological variables: inverse barometric pressure, temperature, local humidity, precipitation, and cloud cover. Neutron data were collected using moderated 3He neutron detection systems located in Annapolis, MD. Data collected using a large neutron sensor in Annapolis show the neutron background varying from 13,000 counts per hour to 9,000 counts per hour, a 20% variation. Meteorological data were collected with two commercially available weather stations in Annapolis, MD along with sensors installed at a nearby airport. Synchronization of the weather and neutron data was an important part of this study and dictated the time interval that could be chosen for the modeling process. Linear autoregression was used to estimate the effects of the meteorological variables on neutron flux while accounting for the correlation among errors at previous time intervals. The dominant variable of the model was inverse barometric pressure with a contribution an order of magnitude larger than any other variable's contribution. The resulting model can predict neutron background with errors of a predictable magnitude. When implemented, this approach will support optimal performance of alarm algorithms, improved sensor effectiveness, and higher likelihood of interdicting the illicit trafficking of Special Nuclear Material out of regulatory control.

15. SUBJECT TERMS
Temporal neutron background, neutron background prediction, weather effects on neutron background, atmospheric particle flux, neutron background radiation, modeling the neutron background

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		82	**19b. TELEPHONE NUMBER** *(include area code)*

U.S.N.A. --- Trident Scholar project report; no. 462 (2017)

MODELING THE EFFECTS OF METEOROLOGICAL CONDITIONS ON THE NEUTRON FLUX

by

Midshipman 1/C Thomas J. Wilson
United States Naval Academy
Annapolis, Maryland

Certification of Advisers Approval

Professor Svetlana Avramov-Zamurovic
Weapons and Systems Engineering Department

_____
(signature)

_____
(date)

Assistant Professor Marshall Millett
Mechanical Engineering Department

_____
(signature)

_____
(date)

Professor Martin E. Nelson (Emeritus)
Mechanical Engineering Department

_____
(signature)

_____
(date)

VADM Charles J. Leidig, USN (Ret.)
Corbin A. McNeill Chair in Engineering

_____
(signature)

_____
(date)

Assistant Professor Douglas N. VanDerwerken
Mathematics Department

_____
(signature)

_____
(date)

Lieutenant Kayla J. Barron, USN
Superintendent's Aide

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

_____
(signature)

_____
(date)

USNA-1531-2

Abstract

The neutron background at sea level is seen to vary by as much as 20% within a 24 hour period. These short term variations are primarily driven by environmental factors. Radiation sensors detect a signal in a noisy background, so the variation of the background must be understood for sensors to be effective. Neutron sensors are used in the search for transported materials that can be used to make a nuclear weapon. The purpose of this project was to develop a statistical model that predicts environmental neutron background as a function of five meteorological variables: inverse barometric pressure, temperature, local humidity, precipitation, and cloud cover. Neutron data were collected using moderated 3He neutron detection systems located in Annapolis, MD. Data collected using a large neutron sensor in Annapolis show the neutron background varying from $13,000$ counts per hour to $9,000$ counts per hour, a 20% variation, over five months of data collection with large variation between days. Meteorological data were collected with two commercially available weather stations in Annapolis, MD along with sensors installed at a nearby airport. Synchronization of the weather and neutron data was an important part of this study and dictated the time interval that could be chosen for the modeling process. Linear autoregression was used to estimate the effects of the meteorological variables on neutron flux while accounting for the correlation among errors at previous time intervals. The dominant variable of the model was inverse barometric pressure with a contribution an order of magnitude larger than any other variable's contribution. The resulting model can predict neutron background with errors of a predictable magnitude. This approach to background correction provides an independent means of normalizing neutron sensor data to a constant background level, whereas traditional means of background correction rely on recent count history - with no assurance that recent changes observed are based on background alone. When implemented, this approach will support optimal performance of alarm algorithms, improved sensor effectiveness, and higher likelihood of interdicting the illicit trafficking of Special Nuclear Material out of regulatory control.

Acknowledgements

As with any large project there are many people to thank and more than I could list here. To start I would like to thank each one of my advisers for their individual contributions to this project. Assistant Professor Millet and Professor Nelson for providing me with the the idea and equipment to perform this project. Professor Avramov-Zamurovic for pushing me even when I did not want to keep working and never giving up. VADM(ret) Leidig, USN for helping me acquire to satellite location giving me a story to tell for the project. Lieutenant Barron, USN, for being my calming voice of reason and keeping expectations realistic during the busy periods of the project. And Assistant Professor VanDerwerken for your countless hours of work and bringing your expertise in statistical modeling to create the final results.

A special thanks to the following people as well for all of their help:

Dan Marron, for all of your help getting the satellite location set up.
To the Naval Support Facility director for letting me use the satellite building.
Nathan Paradis, and his team at SPAWAR for helping me with my internship and contacts to get the verification data.
The Trident Committee for giving me the opportunity to pursue this project.
DTRA at West Point for their technical support and sponsorship of this project.
TSD for their help in providing the necessary equipment for my setup.

KEYWORDS: Temporal neutron background, neutron background prediction, weather effects on neutron background, atmospheric particle flux, neutron background radiation, modeling the neutron background.

Contents

Nomenclature

β	True model coefficients
$\mathbb{E}[X]$	Expectation of x
γ	gamma radiation
$\hat{Q}$	The 'hat' on the variable represents the estimated value of that variable in the paper
$\mathcal{N}$	Normal Distribution
μ	Mean
Ω	Solid angle
ω_i	White noise error term in an AR(n) model
Φ	Measured particle flux at a location
ϕ	Lag coefficient
$\Phi(L)$	Boeing neutron flux
Φ_0	NRL CREME particle flux
$\Phi_{neutron}$	Measured neutron flux
Φ_{NRL}	Particle flux obtained from the NRL CREME model
π	Pi
ρ_o	Humidity reading interpolated between stations by Rosolem et. al.
ρ_{air}	Density of air
ρ_{vo}	Absolute humidity at the sensor
Σ	Variance of an FGLS or AR model
σ^2	Variance
σ_ε^2	AR(2) mobile model error term
Σ_x	Macroscopic cross-section of element x
σ_x	Microscopic cross-section of element x
τ	Substitution variable
ε_i	Random white noise error contained in a model
^{1}H	Hydrogen

^{2}H	Deuterium
^{3}He	Helium
A	Atomic weight
$Corr$	Correlation
cov	Covariance
F_1	Known particle flux
$F_X(x)$	CDF function
$f_X(x)$	PDF function
H	Water vapor scale height
h	Some integer value
I	Identity matrix
i	A single moment in time
idd	Independent and Identically Distributed
L	IBM model factor
n	Number of samples in a distribution
N_{AV}	Avogadro's number
P	Pressure
p	Number of explanatory variables contained in an matrix
$P(X \leq x)$	Probability of event x
P_0	Atmospheric pressure in pascals from a reference city
P_1	Atmospheric pressure in pascals at current location
PI_{Mobile}	Prediction interval of the AR(2) mobile method
$PI_{Stationary}$	Prediction interval of the AR(2) stationary method
R	Ideal gas law constant
r_i	Residual error at time i
T	Temperature
$T_{transmission}$	Percentage transmission of particles
$Var(X)$	Variance of X
X	Continuous random variable
x_{i1}	Input variable for a model
Y_i	True number of neutrons at a given location
z	Height in meters
z_o	Ground level

ALT Altitude

AR(n) Autoregressive model of order n

ASOS Automated Surface weather Observation Station

b 1 barn or 1×10^{-24} cm^2

CDF Cumulative Distribution Function

eV electron volts

FGLS Feasible Generalized Least Squares regression

G Geomagnetic cutoff

GeV 10^9 electron volts

GR Geomagnetic rigidity

IBM International Business Machines

IWV Integral of water vapor

LNS Large Neutron Sensor

MATLAB Matrix laboratory

MData Master Data matrix holding the measured values of neutron counts and all inputs

MeV 10^6 electron volts

NASA National Aeronautics and Space Administration

NOAA National Oceanic and Atmospheric Administration

NRL The Naval Research Laboratory

OLS Ordinary Least Squares regression

PDF Probability Density Function

RIG Geomagnetic cutoff at altitude of 20km

SNM Special Nuclear Material

SPAWAR Space and Naval Warfare Systems Command

USNA United States Naval Academy

List of Figures

List of Tables

Chapter 1

Motivation For Research

Real-time detection of special nuclear material (SNM) requires an accurate assessment of the radiation background given measurable or observable conditions. Determining background radiation levels with detection equipment requires a collection period with controlled survey conditions, which is not always practical or possible in the field due to potential source contamination. Using natural and independent causal factors to model the natural background, however, would eliminate the possibility of source interference without the need for controlled conditions.

A historic example demonstrating the potential value of a reliable, independent neutron background characterization occurred during the Cold War when the United States Navy employed neutron detectors to scan ships for SNM. Anecdotally, the Navy mounted the detectors on vessels and compared readings collected over open ocean to readings near ships. According to their analysis, every ship scanned carried nuclear material, indicating that the detector was producing false positives [1,2].

Follow-on analysis revealed that a natural phenomenon dubbed the "ship effect" caused measurements taken near open water to be lower than those over ships, generating misleading results. Because hydrogen atoms in water molecules absorb, or capture, a large percentage of neutrons, there are not as many low energy neutrons to be detected in the vicinity of large bodies of water. However, as a detector moves closer to a large object made of a higher-atomic number material, like an iron ship, the neutron count increases due to neutron producing reactions. High-energy incident neutrons interact with the iron, producing lower-energy neutrons in all directions. Because there are more low-energy neutrons, the detectable neutron flux will increase when the detector moves from open water toward a ship [1, 2]. This example illustrates that detecting SNM using a mobile neutron detection system requires an accurate estimate of neutron background at a specific location.

The largest effects on neutron background are longitude, latitude, and altitude. The effect of these variables is well understood. However, because they do not account for meteorological conditions, spatial models produce only rough neutron flux estimates. This project will model the effect of meteorological variables on neutron counts, allowing for future integration with spatial models. Combining the models will fine-tune the neutron estimate, reducing uncertainty and making a mobile neutron detection system more effective.

Chapter 2

Basics Of The Neutron Background

2.1 Sources of The Neutron Flux

Cosmic rays incident on the earth's atmosphere from two sources, the sun and the galaxy, create the ambient particle flux at sea level. The number of cosmic rays that reach the earth's atmosphere is affected by the magnetic field produced by the sun. The cosmic rays from the sun and the galaxy, the sun's magnetic field, and their reaction probabilities are discussed in this section.

2.1.1 Solar Cosmic Rays

The sun releases solar particles in the form of protons, neutrons, and heavy ions, but because their kinetic energy is too low, they do not significantly contribute to the neutron flux in the earth's atmosphere. The neutron flux only varies by about 2% between day and night on a given day [2].

In the 1960s, the launch of satellites allowed scientists to measure the sun's cosmic rays outside the earth's atmosphere. Those measurements showed that, at less than one GeV, the sun's cosmic rays do not have enough energy to penetrate the atmosphere. However, because of the magnetic field it produces, the sun has a major effect on the neutron background [2, 3].

The sun's magnetic field interferes with galactic particles travelling toward the earth and, depending on its strength, has the potential to stop a portion of these particles from ever reaching the earth's atmosphere. The sun's ten- to eleven-year solar cycle is driven by sun spot activity. When the sun is very active, its magnetic field encompasses the earth's orbit and acts as a shield. But, when sun activity is low, the magnetic field does not reach the earth and therefore does not shield it from any particles. Because the sun's magnetic field varies over the solar cycle, background neutron counts on the earth are inversely proportional to the sun's activity [2].

The relationship between the sun's magnetic field and neutron flux at sea level is shown in figure 2.1. The solid black line is the percentage the observed neutron flux was below the maximum neutron flux recorded in 1954. The dashed line in figure 2.1 represents the solar cycle, defined by the number of sunspots per year, at the time of the measurements. This graph illustrates the inverse relationship between the sun's magnetic field and neutron flux at sea level [2].

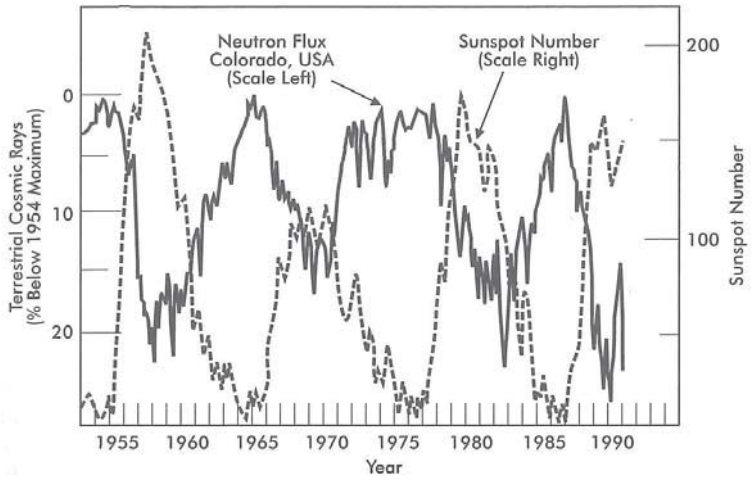


Figure 2.1: The neutron flux from cosmic rays and the activity of the sun [2]

2.1.2 Galactic Cosmic Rays

Galactic cosmic rays are the primary contributor to particles incident on the earth's atmosphere. Although the source of galactic particles is unknown, they are probably created when stars explode or collide. The total number of particles incident upon the earth is considered a constant, and is made up of 85% to 92% protons and 6% to 13% alpha particles. Heavy ions, such as iron nuclei, make up the remaining percentage. Galactic particles do not include neutrons because, when outside the nucleus, a neutron has a half-life of about 10 minutes, which is very short compared to the time required to reach the earth from an unknown point of origin. Once a free neutron is absorbed by a nucleus, it becomes stable and will no longer decay [2].

When galactic rays enter the earth's atmosphere they interact with the nuclei of atmospheric elements, causing spallation. Spallation is the breaking of a nucleus into many fragments, creating sub-atomic and atomic particles, such as pions, muons, neutrons, gammas, positrons, and electrons. A high-energy galactic particle will collide with a nuclei, causing it to give off several lower-energy particles that go on to cause spallation themselves. This is commonly referred to as a particle cascade. The effect, illustrated in figure 2.2, is that just one high-energy galactic particle can create many lower-energy particles at low altitudes, resulting in a high particle flux. The neutron background at the earth's surface is created by this cascade of particles in the atmosphere [2].

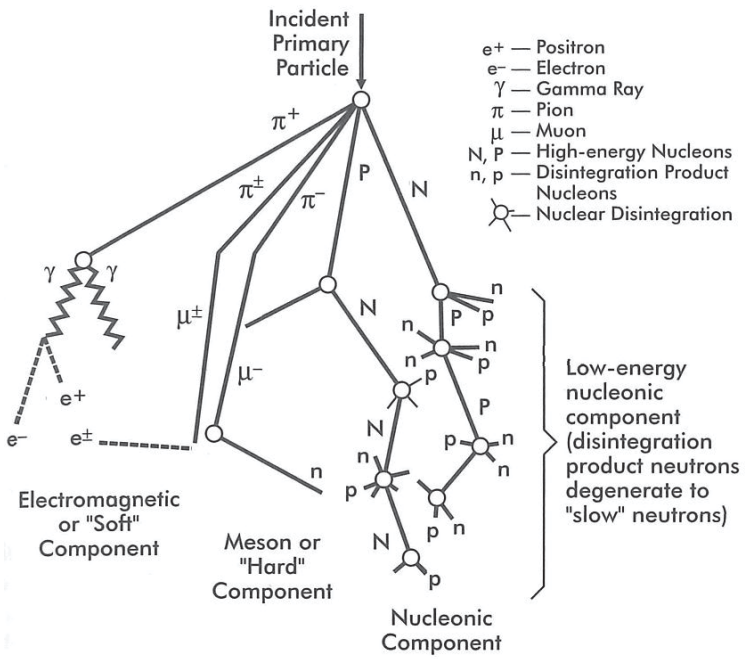


Figure 2.2: The effect of spallation caused by a single high energy cosmic particle [2]

Although no high-energy galactic neutrons enter the atmosphere, due to the particle cascade, neutrons are the second most abundant particle at sea level, surpassed only by muons. (Muons are heavy electrons traveling at relativistic speeds that do not react with other nuclear forces [2].) Neutrons are so abundant because high-energy neutrons can interact with high atomic number materials to cause reactions that produces two or more neutrons, called (n,2n) or (n, 3n) reactions. Figure 2.3 shows the modeled make up of cosmic particles for New York City. Neutrons become a large part of the particle flux inside the atmosphere.

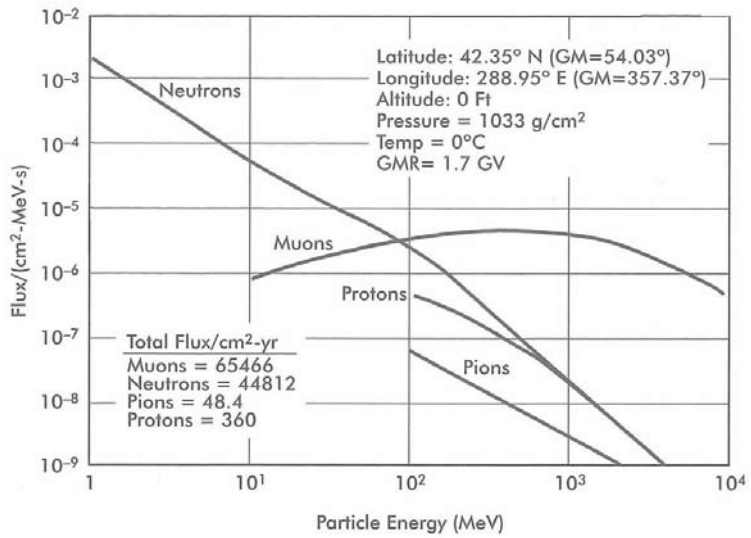


Figure 2.3: The most abundant particles at sea level after spallation has occurred through the atmosphere [2]

2.1.3 Reaction Probabilities

The higher a neutron's kinetic energy, the more likely the it will cause spallation rather than be absorbed. At sea level, most incoming neutrons are in the energy range of $10 - 1000$ MeV, as shown in figure 2.4. Neutrons in this energy range can cause spallation. There is a second peak of neutrons with energy from $0.1 - 5$ MeV. These neutrons are introduced by spallation or neutron-producing reactions, and often have enough energy to cause more reaction events before they are absorbed. This peak results from the "reaction threshold" of many nuclei, or the minimum energy at which the nuclei can have nuclear reactions with incident particles. The third peak is made up of thermal neutrons. Thermal neutrons have lost almost all of their energy and are bouncing around at almost the same energy as sea-level nuclei. The kinetic energy of a neutron is directly proportionally to the square of its velocity. Therefore, neutrons with lower kinetic energies have slower velocities [2].

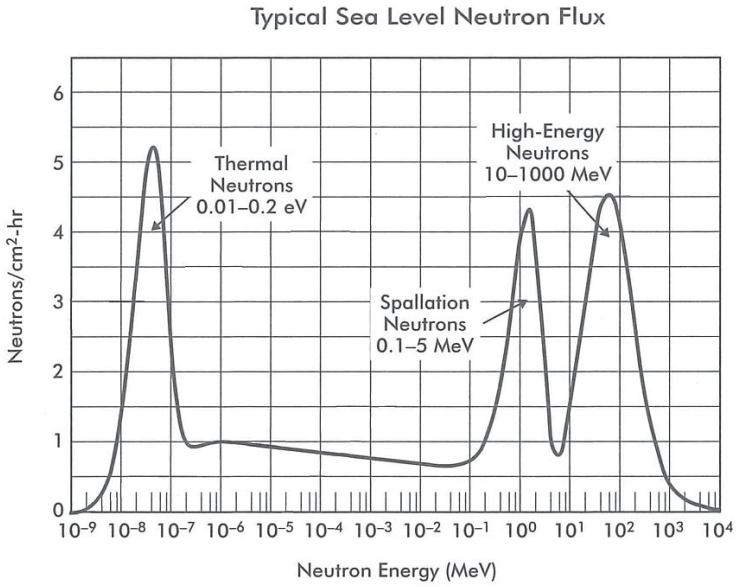
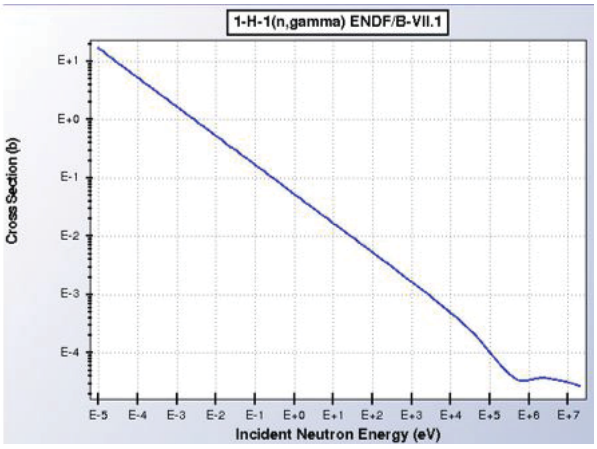


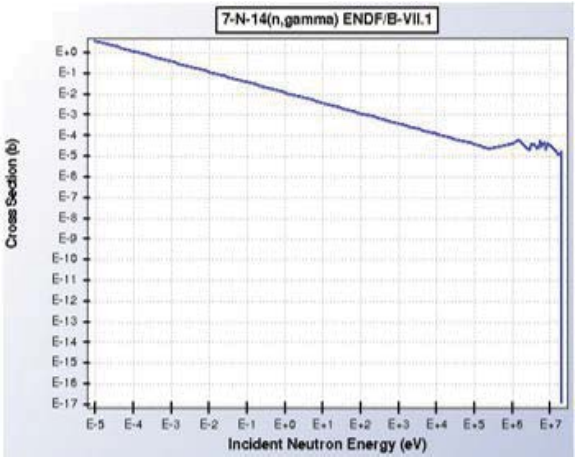
Figure 2.4: Energy of the nuclear particles at sea level [2]

The probability that a neutron will react with an atom per unit distance travelled changes with neutron energy and is described in terms of the atom's microscopic cross section. The cross section is the relative probability that a nuclei will react with a neutron of a specific energy. For most elements, as a neutron's kinetic energy decreases the absorption cross section increases.

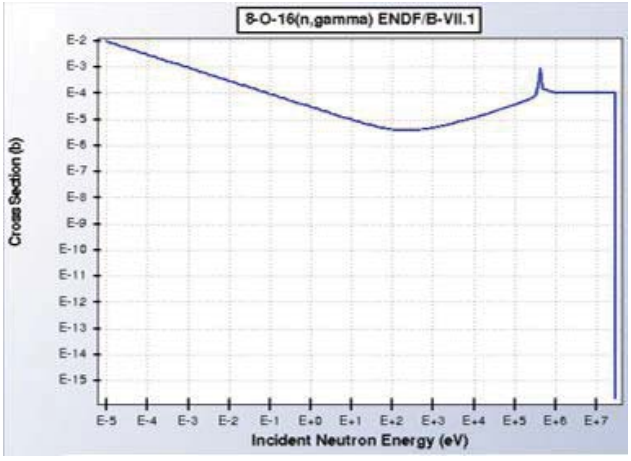
Figures 2.5a through 2.5c show the microscopic absorption cross sections of common atmospheric elements. The x-axis shows neutron energy in electron volts (eV) and the y-axis shows microscopic absorption cross section in barns (b). A barn is a unit of reaction probability defined as 1×10^{-24} cm^2. Figure 2.5a shows the cross section for the neutron-gamma (n, γ) reaction with hydrogen, in which a hydrogen atom absorbs a neutron and releases excitation energy in the form of a gamma ray. Figures 2.5b and 2.5c show the same reaction for nitrogen and oxygen. Hydrogen's (n, γ) cross section is the largest, exceeding ten barns for neutron energies below 10^{-5} eV.



(a) Reaction probability of hydrogen



(b) Reaction probability of nitrogen



(c) Reaction probability of oxygen

Figure 2.5: 2.5a is a log-log plot with the x-axis representing the energy from 10^{-5} eV to 10^{7} eV. The y-axis represents the cross section from 10^{-4} b to 10 b. 2.5b is the reaction probability of nitrogen with the x-axis representing the energy from 10^{-5} eV to 10^{7} eV. The y-axis represents the cross section from 10^{-17} b to 1 b. 2.5c is the reaction probability of oxygen with the x-axis representing the energy from 10^{-5} eV to 10^{7} eV. The y-axis represents the cross section from 10^{-15} b to 10^{-2} b. [4]

The neutron absorption cross section of nuclei is important because as humidity increases, it adds hydrogen, with a very large cross section, to the atmosphere. The more hydrogen added to atmosphere the more probable it is of a neutron being absorbed. Similarly, as air pressure is a measure of the weight of a column of air above the measurement point, pressure is proportional to the interaction probability for high energy neutrons streaming into the atmosphere from high altitude spallation events. The cross section is also important in the development of a neutron detector and deciding what nuclei to use in the detector.

2.2 Previous Models

There are four primary models that predict neutron flux at a given location. However, the models are based solely on the physical position on the earth and do not account for local weather conditions. Additionally, because of their intended application, the models predict neutron flux at specific altitudes above sea level. The four models were developed by IBM, The Naval Research Laboratory (NRL), Boeing, and Boeing-NASA.

2.2.1 IBM

The IBM model was built to model the particle flux over major cities from sea level to about 30,000 feet. It accounts for the latitude, longitude, and altitude near large cities, and in addition to neutron flux, it also predicts the electron, proton, pion, and muon flux at the location [5].

The IBM model uses known flux quantities taken from a location with approximately the same geomagnetic cutoff, explained in section 2.3.1, to interpolate the data for an unknown location. With the data from the known location, the model accounts for the change of altitude at the point of interest using equation 2.1.

$$\Phi = \left(0.0142 \frac{nucleons}{cm^2 s}\right) F_1 e^{\frac{P_0 - P_1}{L}} \tag{2.1}$$

Where Φ is the flux at the location in particles per cm^2 seconds, F_1 is the particle flux of the known city, the 0.0142 is the particle flux at New York City in particles per cm^2 seconds, P_0 is the atmospheric pressure in pascals at the altitude of the reference city, P_1 is the pressure in pascals at the location of interest, and L is a factor that will vary from 100 to 160 grams per cm^2 depending on the particle.[5, 6]

IBM can model the flux at a given location within 20% if the altitude of the location is less than $30,000$ feet. However, if the location is above $30,000$ feet, the model error increases to about four orders of magnitude above the correct measurements. The error is probably caused because the model is based upon a linear attenuation of particles from sea level up to $30,000$ feet and does not account for the change in meteorological conditions [5, 6].

2.2.2 NRL CREME

The CREME model built by the Naval Research Laboratory (NRL) calculates the neutron flux an aircraft would experience at altitudes over $50,000$ feet and is more accurate than the IBM model at high altitudes. NRL used a polynomial fit to the data they collected to create a rough estimate of the expected neutron flux at a given location. Equation 2.2 gives the flux of particles (Φ_0) in terms of neutrons per steradian-cm^2 s.

$$\Phi_0 = 4010 + 1590G + 267G^2 - 22.3G^3 + 0.902G^4 - 0.0141G^5 \tag{2.2}$$

G is the geomagnetic cutoff (see section 2.3.1,) as a function of RIG. RIG is a variable that NRL created to function like a geomagnetic cutoff value, but at 20km above sea level it has no other meaning. RIG is a necessary correction factor because the location is so far above the earth that the earth does not block as many incident particles as it would closer to the surface. G is calculated using the altitude (ALT) in meters and tabulated data for RIG as shown in equation 2.3.

$$G = RIG \left(\frac{2.10(10)^7}{ALT + 2.09(10)^7}\right)^2 \tag{2.3}$$

Equation 2.2 is the fundamental equation for the CREME model that is then made more accurate by adding the following terms to this equation.

The second term that NRL used to refine the model is to account for the conic section that the earth will block as a function of the altitude. As the location increases in altitude the conic section the earth is blocking will decrease, exposing the location to more incident particles. This conic section for a given altitude that the earth is blocking is given, in steradians, by:

$$\Omega = 4\pi \left(.5 + .5 \frac{\left(\left(0.209x10^8 + ALT\right)^2 - 0.438 \times 10^1 5\right)^{.5}}{0.209 \times 10^8 + ALT}\right) \tag{2.4}$$

The final correction is for the amount of particles ($T_{transmission}$) that actually make it to the location as a function of pressure (P) given in equation 2.5. Pressure is given in units of grams per cm^2.

$$T_{transmission} = e^{-0.11P + 0.0245} \tag{2.5}$$

The CREME model is calculated by combining equation 2.3, 2.4, and 2.5 into a single term Φ_{NRL} in terms of nucleons per cm^2 per second [7, 8, 9, 10].

$$\Phi_{NRL} = G\Omega T_{transmission} \tag{2.6}$$

2.2.3 Boeing

Boeing built a model of the low energy neutron flux at cruising altitude for an aircraft. Published in 1993, their model is built off of data collected at 35,000 feet above sea level. This altitude allows less time for spallation events to happen compared to lower altitudes. Boeing considered neutrons with kinetic energies between 1 and 10 MeV because neutrons in that range of energy cause noise in aviation electronics [3].

Boeing estimated the neutron flux at 35,000 feet using equation 2.7, where N is the neutron flux in nucleons per cm^2, and L is the latitude in meters.

$$\Phi(L) = 0.6252^{\left(-0.461(cos(2L))^2\right)-0.94cos(2L)+0.252} \tag{2.7}$$

2.2.4 Boeing-NASA

Boeing developed a second model that used data collected mostly by NASA-Ames to estimate the neutron flux at 50,000 feet above sea level and for all neutron energies. The data were collected using satellites and weather balloons. The model was built to predict how much radiation shielding was needed for new satellites. Because the model predicts neutron flux high in the atmosphere, Boeing did not create a mathematical estimate at different locations, but rather made a global estimate shown in figure 2.6. The neutron values below 30,000 feet are calculated using Boeing's first model and normalized for this model. Above 30,000 feet the neutron flux is an estimate created by taking readings around the world at different heights and latitudes [11].

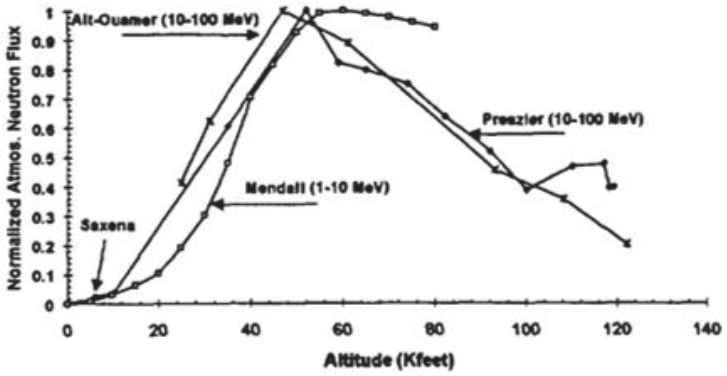


Figure 2.6: Boeings estimation of neutron flux [11]

2.3 Variables that Affect The Neutron Flux

There are several factors that affect the neutron background such as atmospheric pressure, temperature, and humidity. This following section discusses the physics and previous modeling that has been done with each variable.

2.3.1 Magnetic Fields

The magnetic field strength from the sun affects the neutron flux, but the earth's magnetic field also contributes to the formation of the neutron flux. To account for the earth's magnetic field, scientists have done extensive three-dimensional analysis to model possible incident paths and kinetic energies for the incoming cosmic particles. The model shows that incoming particles must be above a certain kinetic energy in order to penetrate to sea level. When discussing the required energy in terms of the longitude and latitude the phrase "geomagnetic cutoff" is used. Or the term "geomagnetic rigidity" (GR) may be used, in reference to the minimum kinetic energy, to describe the same affect as geomagnetic cutoff. Figure 2.7 shows contour lines that represent the particle rigidity required to penetrate the earth's atmosphere in GeV [2].

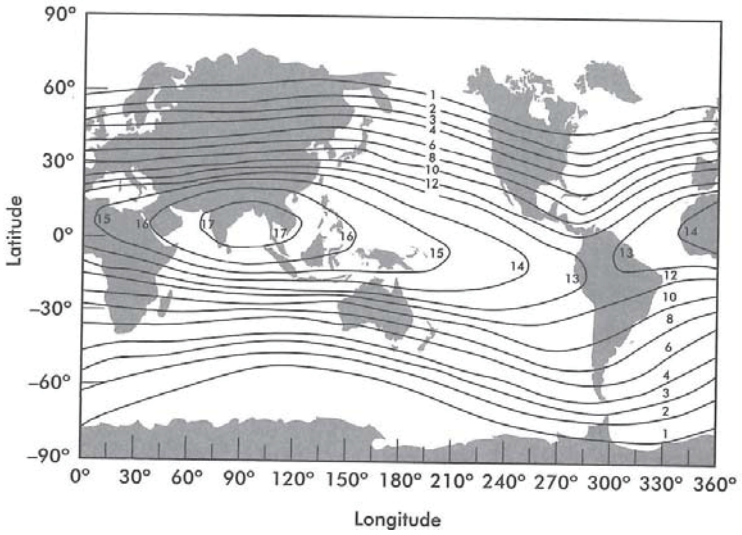


Figure 2.7: Geomagnetic Rigidity lines [2]

Because the earth's magnetic field lines are the weakest at the poles, fewer particles will be deflected and more will reach sea level. Interestingly, the highest GR region centered on the Indian Ocean, where there is about half as much cosmic radiation as most of the United States [2].

2.3.2 Pressure

Atmospheric pressure is another variable condition that affects the neutron flux. Equation 2.8 relates neutron flux to atmospheric pressure.

$$N\left(P_o + \delta P\right) = N\left(P_o\right)^{\alpha \delta P} \tag{2.8}$$

$N(P)$ is the intensity of the neutron flux measured at a pressure and α is a constant called the pressure coefficient. In equation 2.8, P_o is the mean pressure at the given location and δP is the difference of the current reading to the mean pressure readings, both readings are in millibars, equal to 9.869×10^{-4} standard atmospheric pressure [12].

The exact value of the pressure coefficient (α) remains unknown. S. Lindgren made the most significant contribution to determining the pressure coefficient in the late 1960s by comparing two different locations using this relationship.

$$ln\left(\frac{N_1}{N_2}\right) = ln\left(\frac{N_1^*}{N_2^*}\right) + \alpha\left(P_1 - \delta P_2\right) \tag{2.9}$$

Where $N_1 = N(P_o + \delta P)$ and $N_1^* = N(P_o)$ are measured at the daily mean pressure, in millibars, and neutron flux intensities. N_2 and N_2^* represents a second location. Using equation 2.9, Lindgren determined the value and variance of the pressure coefficient, which has units of %/mb, as displayed in figure 2.8. The x-axis is the trial number, and the y-axis is the value of the coefficient [12].

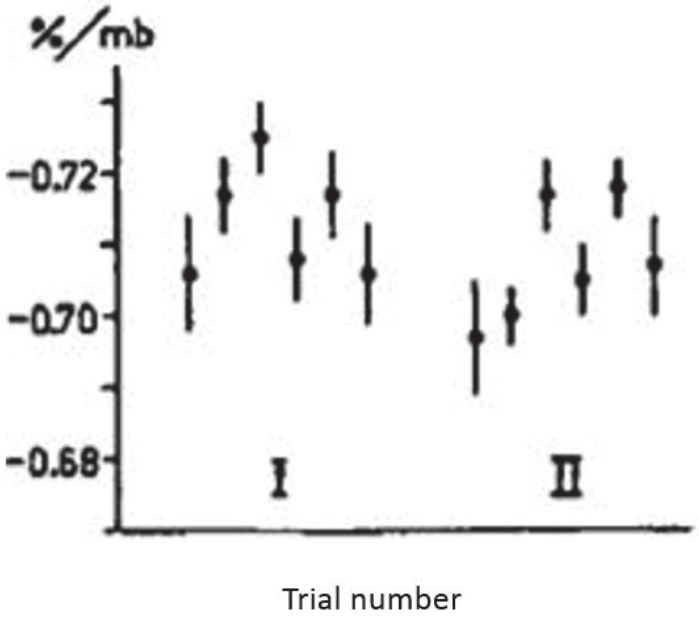


Figure 2.8: Pressure coefficients. X-axis is the experiment number. Y-axis is the value of the coefficient. [12]

Figure 2.8 shows the results of the experiments performed at two different locations, indicated by the roman numerals on the graph. The dot in each line is the mean result of six experiments and the lines indicate the variance. The variance for the mass absorption coefficient between runs is 0.006 percent per millibar. This small amount of variance demonstrates that Lindgren found acceptable values for the pressure coefficient. [12]

2.3.3 Humidity

Humidity affects the neutron flux because water vapor contains hydrogen nuclei that interact with neutrons, removing them as they travel through the atmosphere [13, 14]. The probability that a hydrogen atom will react with a neutron increases as the neutron loses kinetic energy and slows down. A slow neutron takes longer to travel through the nuclear force field that surrounds a nuclei, increasing the probability of reaction. Hydrogen (^1H), nucleus consists of a single proton and has a total cross section many times larger than nitrogen and oxygen, the two major atoms in the atmosphere, as shown in figure 2.9.

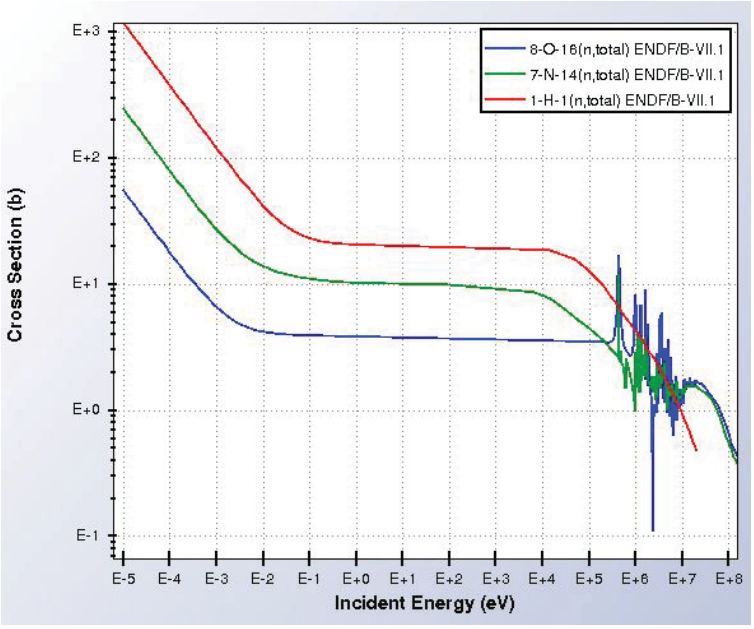


Figure 2.9: Cross section of hydrogen particles in the atmosphere compared to nitrogen and oxygen. The top red line is the total cross section of hydrogen with the total cross section of nitrogen and oxygen respectively below. [21]

The purpose of the research done by Rosolem et. al. [13] was to demonstrate that the humidity in the atmosphere above a detector would effect the neutron measurements. To accomplish this research they relied on simulations of the neutron interactions with hydrogen in the water vapor. Rosolem et. al. hypothesized that only water vapor in the first few hundred meters would effect neutron flux measurements. Atmospheric data was recorded at three different heights on two 447-m tall towers. The humidity readings at every meter between the stations was then interpolated using the equation

$$\rho_o(z) = \rho_{vo}^{\frac{-(z-z_o)}{H}} \tag{2.10}$$

where ρ_{vo} is the absolute humidity ($kg\ m^{-3}$) at the sensor, z is the height in meters of the sensor above the ground, z_o is defined as the ground, and the constant $H = 2.3km$ is the water vapor scale height determined by Reitan [13]. Assuming z_o is zero simplifies the integral in equation 2.11, which is used to calculate the integral of water vapor (IWV) from height z to z_0, the ground, from equation 2.11 as shown below:

$$IWV_{0-z} = \int_{z=0}^{z} \rho_o(z)dz = \rho_{vo}H\left(1 - e^{\frac{-z}{H}}\right) \tag{2.11}$$

This gives IWV in kilograms per meter squared corresponding to the equivalent liquid water in millimeters if all the water was condensed on the ground. [13]

In order to simplify the model, Rosolem et. al. assumed that the humidity above a certain height would not have a major effect on the neutron flux any more. To validate this assumption they performed two simulations. The first simulation started with no humidity in the atmosphere and then from the ground up to one kilometer, humidity was introduced to the atmosphere. A ten meter layer of humidity was added to the air directly above the ground first and then subsequent layers on top of that. After each layer was added the simulation calculated the normalized amount of neutrons expected at ground level.

The second simulation was performed in a similar manner as the first simulation by adding one layer at a time. But for this simulation the atmosphere started with humidity in it and the layers removed the humidity ten meters at a time. The results of these simulations are shown in figure 2.10.

To simplify the model, Rosolem et. al. calculated at that the height that carries the most influence on the neutron flux is equal to the height at 86% of the total system response. For the wet layers added to the

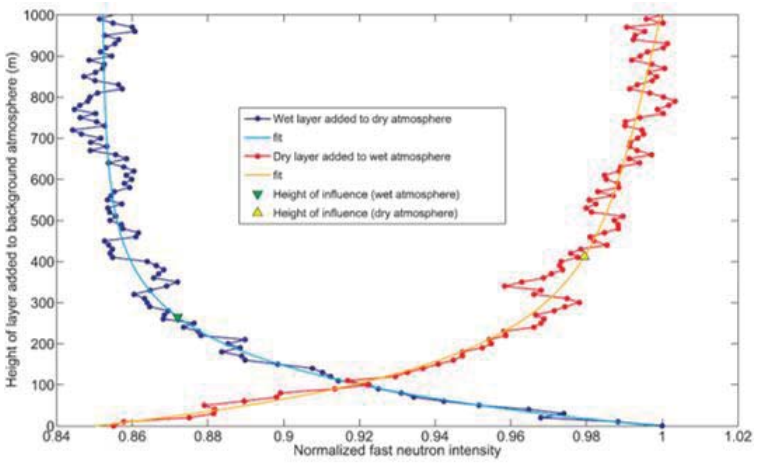


Figure 2.10: Data by Rosolem et. al. to determine the height of the water column [13]

dry atmosphere the height was 265 meters, but for the dry layers added to the wet atmosphere the height was 412 meters. In order for the model to be as accurate as possible, Rosolem et. al. chose the larger of the two heights for their model.

The results of Rosolem et. al. research was that they could demonstrate that humidity is a meteorological factor that effects the neutron flux at sea level [13].

2.3.4 Temperature

To understand how temperature effects neutron flux, one needs to look at the definition of the neutron flux. The definition of the neutron flux is neutron density times neutron velocity and has units of neutrons per centimeter squared per second $\frac{N}{cm^2 s}$. Physically, this represents the total neutron path length, per unit volume, per unit time. The macroscopic cross section is the product of the target's atom density and the microscopic cross section. The microscopic cross section represents the effective target area a nucleus presents to an incident neutron for a particular nuclear reaction, expressed in barns (b). The macroscopic cross section has units of cm^{-1}. It has the physical interpretation of neutron interactions per cm of neutron travel. For example, a macroscopic cross section of 1.0 cm^{-1} would imply the neutron would undergo an average interaction rate of 1.0 interaction per cm of travel. The neutron flux when multiplied by the macroscopic cross section gives the volumetric interaction rate (interactions, per unit volume, per unit time). Mathematically, the neutron volumetric interaction rate for nuclear reaction "x" can be expressed by:

$$interaction\ rate\ for\ reaction\ x = \Sigma_x \Phi_{neutron} \tag{2.12}$$

where Σ_x is the macroscopic cross-section, in barns, of that medium for reaction x, and $\Phi_{neutron}$ is the measured neutron flux in neutrons per cm^2 per second. The air's atom density $\frac{kg}{cm^3}$ can be found by using the following two equations

$$N_{air} = \frac{\rho_{air}}{A} N_{AV} \tag{2.13}$$

$$\rho_{air} = \frac{P}{RT}\,(Ideal\ gas\ law) \tag{2.14}$$

where ρ_{air} is the air mass density in $\frac{grams}{cm^3}$, N_{AV} is Avogadro's number $\left(6.02 \times 10^{23} \frac{atoms}{mole}\right)$, and A is the atomic weight of the isotope undergoing reaction x (grams per mole). The density of the air is determined by the ideal gas law given in equation 2.14, where R is the gas constant of air in kilojoules per kilogram per Kelvin, P is the pressure of the air in kilopascals, and T is the air temperature in Kelvin. Therefore, interaction rate of the neutron with element x is related to the density of air, and by extension temperature. The relationship of air density to neutron interactions is given in equations 2.15 and 2.16.

$$interactions\ rate = \left(\left(\frac{P}{RT}\right)\frac{N_{AV}}{A}\right)\sigma_x \Phi_{neutron} \tag{2.15}$$

$$interactions\ rate = \left(\rho_{air}\frac{N_{AV}}{A}\right)\sigma_x\Phi_{neutron} \tag{2.16}$$

Thus the interaction rate is proportional to the atmospheric pressure and and humidity, but inversely proportional to atmospheric temperature.

Chapter 3

SPAWAR Internship

During the summer of 2016, an internship was performed at SPAWAR, in San Diego, CA. At this time, there were two problems in this project that needed to be addressed. The first problem was acquiring weather data that was collected close to the detectors and over the same time interval of the neutron background data. The second issue was taking the SPAWAR neutron data, which had been collected in the Washington D.C. area from early 2000s through the begining of 2016, into the same format as the data collected at the Naval Academy.

The issue with the weather data was that it needed to be accurate, have frequent measurements, and have data for the complete time period. There are a few organizations with weather data posted on-line that had a few of the specifications, but were missing important factors such as not allowing for large amounts of data to be downloaded at once. However, the National Oceanic and Atmospheric Administration (NOAA) has made all the measurements taken by Automated Surface Weather Observation Stations (ASOS) at all continental airports public record. The ASOS systems are standardized stations that measure pressure, temperate, dew point, condensation, and several other meteorological measurements every minute. The data was taken from Reagan International Airport ASOS system and downloaded in month blocks from the NOAA website.

A MATLAB code was developed that would search the neutron or weather data file for any desired date and time. The code also checked the weather data for continuity, because some periods the sensors where not operational. This program ensured that the trends and correlation between the weather measurements and the neutron data would be consistent.

Next the neutron data files had to have readings in one second intervals with a date and time stamp. However, the data files from the neutron sensor at USNA had the neutron counts summed over a ten minute period. To get both the SPAWAR and USNA data into the same time frame, for comparison purposes, a MATLAB code was written that checked the time stamp for every reading and summed the counts into ten minute intervals.

With the code completed, the first comparison was made between the two different neutron sensors, one located in D.C. and the other in Annapolis, MD. While there is a slight delay between the two sets of data that maybe due to the slight time delay necessary for a weather system to move between locations, both sensors showed the same variations of background neutrons with time.

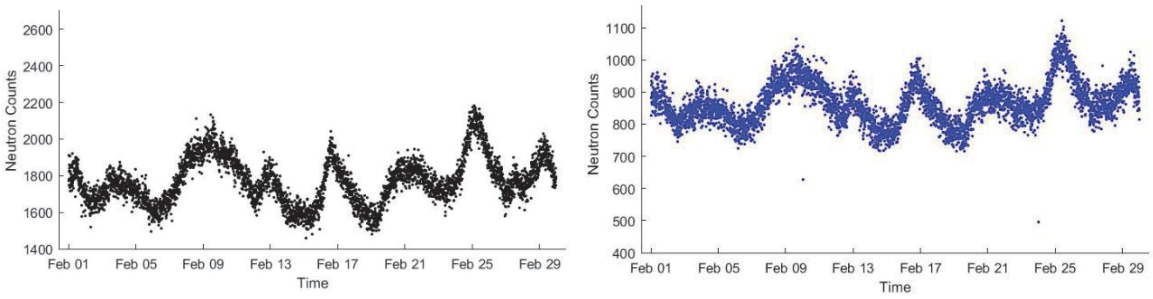


Figure 3.1: Neutron counts from Annapolis and D.C. summed over ten minute intervals. The data collection period is the whole month of February, 2016. The variation of height between the two sensors is based on the sensitivity of each sensor.

To demonstrate the effect of pressure, the inverse pressure and neutron counts where plotted as shown in figure 3.2. The data is normalized, in figure 3.2, so that the visual variations of the inverse pressure and neutron counts can be compared. The smooth line is the inverse pressure in one over inches of mercury and the black line, with more variation, is the neutron counts in ten minute intervals over the month of February, 2016.

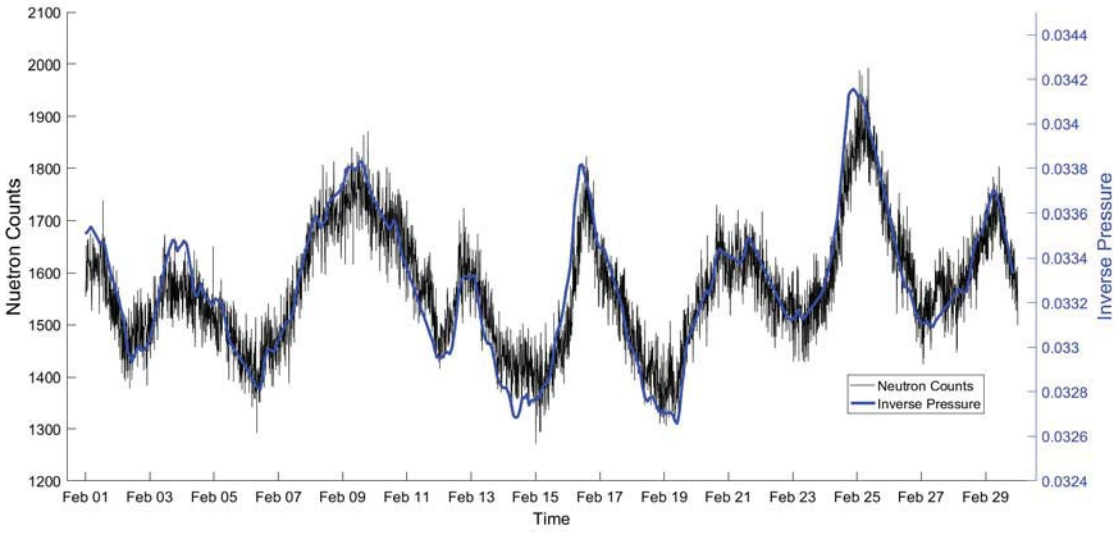


Figure 3.2: Neutron counts from the LNS and inverse pressure from the whole month of February, 2016.

The neutron detector used by SPAWAR was a He-3 detector, the mechanics of a of He-3 detector will be further discussed in the next section. Both the SPAWAR and LNS, employ the same fill gas in the detector. This detector was made of several small neutron tubes approximately 3 feet long and 2 inches in diameter, which add more sensitivity to the system than a single tube could give.

Chapter 4

LNS Detector and Preliminary Experiments

4.1 Detector Characteristics

The neutron detector to be used in this research project originates from the Space and Naval Warfare Systems Command (SPAWAR). This detector is referred to as the large neutron sensor (LNS) in the remainder of the report. The detector contains four helium-3 (^3He) tubes at just over one atmosphere of pressure. Each tube has an internal length of 203 cm, a diameter of 15.24 cm and about 56 liters of helium-3 (a helium nucleus that consists of 1 neutron and 2 protons). ^{3}He, now a substance controlled by the United States Government, is produced by the decay of tritium, an isotope used in nuclear weapons. As the size of the nuclear stockpile shrinks, so does the availability of ^{3}He. Because of the scarcity of helium-3, the detector owned by the U.S. Naval Academy is one of the largest and most sensitive neutron detectors available. Figure 4.1 shows the detector and one of the four tubes outside. [16]

Figure 4.1: USNA neutron detector [16]

Helium-3 is used in neutron detectors because helium is highly reactive with thermal neutrons. Figures 2.5 show the reaction probability cross sections of helium-3 and hydrogen respectively. Helium-3's cross section is several orders of magnitude larger than hydrogen, making the detector very sensitive to incident thermals neutron since the probability that an incident neutron will react with the helium-3 greatly exceeds

any other nuclei in the air.

An incoming cosmic particle will cause a particle shower creating both high energy and low energy neutrons in the lower atmosphere. The tubes of helium-3 are surrounded with ethylene (C_2H_6), which moderates high energy neutrons, slowing them to thermal energies where they have a high probability of reaction with the helium-3. When a neutron reacts with the helium-3 it creates a tritium ion and a proton causing ionization in the tube [16].

Each of the four tubes contains a single wire, kept at a specific voltage, running the length of the tube. The tritium ion and protons are attracted to the charged wire and cause a small change in detector voltage that is amplified and measured. The change in voltage is used to create a transistor-transistor logic (TTL) signal to the controller signifying a neutron reaction. Each tube's controller records the number of neutron reactions, or counts, and gives each count a timestamp down to the milli second. Figure 4.3 is a sample readout of the electrical pulses that the LNS sends to the recording device. [16]

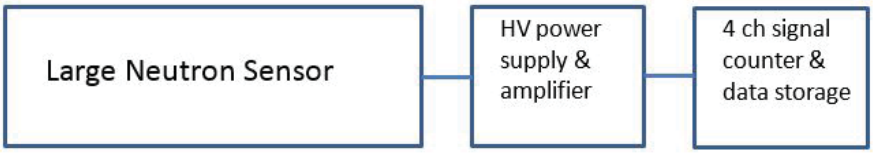


Figure 4.2: Block diagram of USNA detector

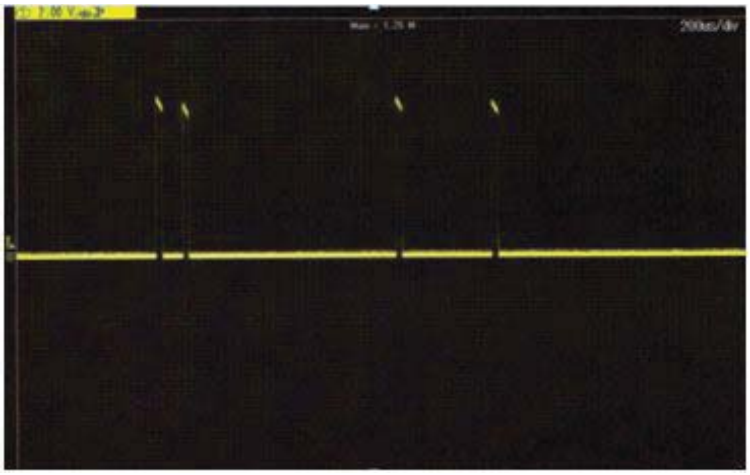


Figure 4.3: Sample read out of the detector [16]

4.2 Preliminary Experiments

To demonstrate the shape of the neutron counts data, the detector was used to collect data from 01JAN16 through 13JAN16. The detector was placed just inside a roll up door in Rickover Hall on the USNA grounds to collected data. The detector counted for ten minutes at a time and then recorded the number of counts for each tube onto a memory chip. The data should be a true representation of the natural neutron background because with little traffic around the area of the detector there should not have been any neutron sources introduced into the environment. The only causes of variations to the neutron counts in figure 4.4 should be the meteorological variables to be studied in this project.

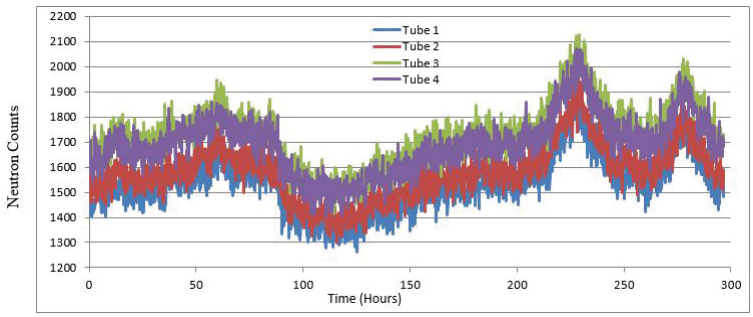


Figure 4.4: Neutron counts from the detector in hour intervals from 03JAN16 to 13JAN16

Figure 4.4 shows measurements from the four different helium tubes contained within the detector. The varying of the four tubes at the same rate demonstrates that all four tubes are working properly and detecting the neutrons. If one of the tubes had a large difference in counts compared to the rest that would suggest that one tube is faulty and the data unusable for the modeling process. The average was about 1600 neutron counts in the ten minute intervals in which the data was collected. This is a large collection window, for the modeling process the data will be collect in smaller windows and the time windows can be added together if it is necessary in order to create a better model.

The second set of data needed was the weather data for that same time that the neutron data was collected. There is a weather station located on Hospital Point on the USNA grounds that uploads the temperature, humidity, and pressure to a national weather station on an hourly basis. The data cannot be exported from the website but it can be viewed and manually copied into a usable form. The weather data from this 300 hour period was retrieved from that weather service and used to make figure 4.5.

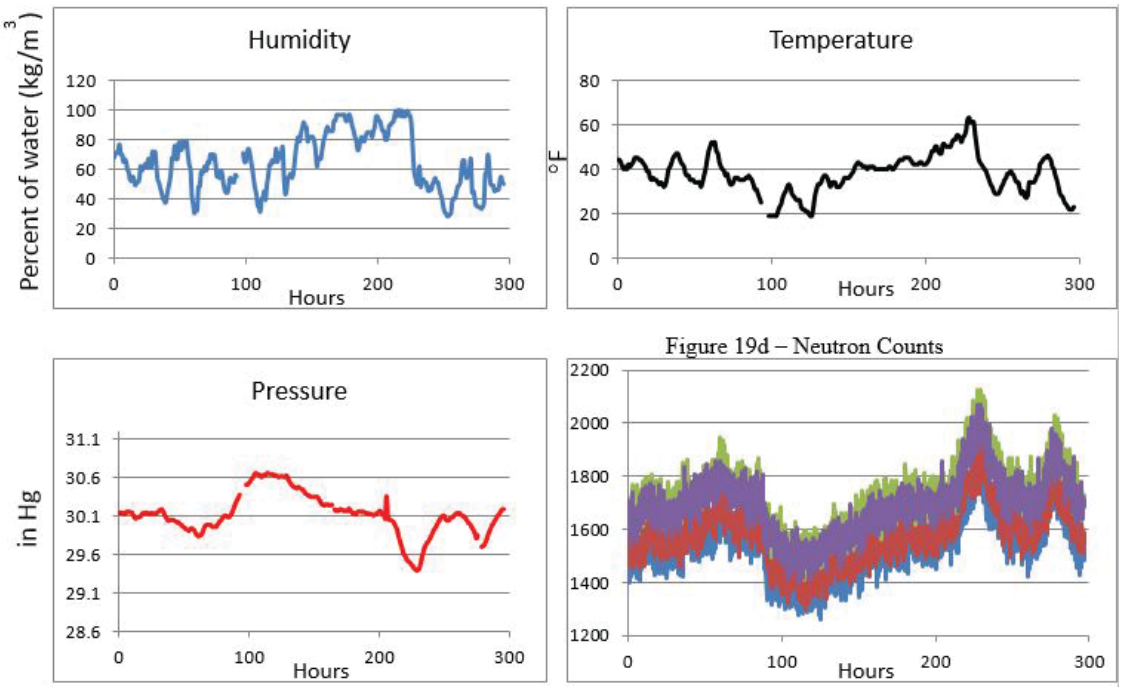


Figure 4.5: Weather and neutron count graphs

These graphs, of data over 300 hours, show a clear correlation with temperature and pressure to the neutron counts. From equation 2.15, the relationship between the neutron counts and temperature was expected to be a proportional, while the relationship between pressure and humidity was expected to be an inverse. These experimental graphs clearly demonstrate that there are strong relationships as expected.

This project collected data of this type almost continuously over the period of several months. The data was taken continuously for so long to make the modeling process able to account for as many meteorological conditions as possible. And the weather station purchased allows for smaller variations of the weather to be captured and provided the data in an easily used format.

Chapter 5

Model Development

This project built a model based on statistical distributions and it is important to understand several parameters about the final distributions used. This chapter will step through the basics of distributions and then explain the final parameters used in the model.

5.1 Types of Distributions

Let X be a continuous random variable with a cumulative distribution function (CDF) given by:

$$F_X(x) = P\left(X \leq x\right). \tag{5.1}$$

The probability density function (PDF) of the same variable, X, is the derivative of the CDF given as:

$$f_X(x) = \frac{d}{dx} F_X(x). \tag{5.2}$$

Therefore by the Fundamental Theorem of Calculus the CDF can also be described this way:

$$P\left(X \in (a,b)\right) = \int_a^b f_X(x)dx = F_X(b) - F_X(a). \tag{5.3}$$

An example CDF defined for all real numbers is given as:

$$F_X(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x \in (0,1) \\ 1 & \text{if } x \geq 1 \end{cases} \quad . \tag{5.4}$$

The PDF is found by taking the derivative of each piece of the CDF. If the PDF is graphed the area under the curve is equal to one, because it encompasses all possible values of the random variable X. The PDF of equation (5.4) is:

$$f_X(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x \in (0,1) \\ 0 & \text{if } x \geq 1 \end{cases} \quad . \tag{5.5}$$

If the CDF is known, then summaries about the distribution of the variable can be defined as well. The expectation ($\mathbb{E}\left[X\right]$) of a continuous random variable X having a PDF is defined as:

$$\mathbb{E}\left[X\right] = \int_{-\infty}^{\infty} x f_X(x)dx = \int_S x f_X(x)dx \text{ for } S = \{x : f_X(x) > 0\}. \tag{5.6}$$

For a normal distribution, represented as $\mathcal{N}$, the expectation is the mean value of the distribution. Where for a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with a mean (μ) and variance (σ^2):

$$f_X(x) = \frac{1}{\sqrt{\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{5.7}$$

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx = \mu. \tag{5.8}$$

The experimental way to approximate the expectation of a random variable is to take a large sample of n numbers. The second step is to take the mean of the sample group:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{5.9}$$

And as n approaches infinity then the mean will approach the expectation of X.

Another summary of a distribution is its variance. Variance is the average squared distance of a random variable from its expected value. Variance is given as:

$$Var(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2. \tag{5.10}$$

The variance is useful when there is only one variable that is being examined. But if there are two or more variables then covariance must be used to account for the multiple dimensions of the variance. The covariance is given as:

$$cov(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]. \tag{5.11}$$

Variance is the scalar of covariance when the covariance between X and itself is taken, as shown:

$$cov(X, X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]. \tag{5.12}$$

Equation 5.12 equals the definition of variance as given in equation 5.10.

Correlation is a scaled version of covariance that represents the strength of the linear relationship between two different random variables. The correlation is given as

$$corr(X, Y) = \frac{cov(X, Y)}{\sqrt{Var(X) Var(Y)}}. \tag{5.13}$$

Correlation is always between -1 and 1. The sign represents whether the linear relationship is downward- or upward-sloping. The magnitude represents the strength of the relationship. Correlations close to zero indicate that the two random variables are not linearly related, whereas correlations close to one in absolute value indicate near perfect linearity.

5.2 OLS and FGLS Models

Ordinary Least Squares regression (OLS) is a method of regression analysis that creates an output given several independent inputs. OLS in its general form is given by

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i \tag{5.14}$$

where Y_i is the response, $\beta_i$are coefficients, x_i are the explanatory variables, i represents a moment in time, and ε_i is an independently and identically distributed error given by

$$\varepsilon_i \backsim \mathcal{N}\left(0, \sigma^2\right) \tag{5.15}$$

where $\mathcal{N}$ stands for a normal distribution. Because ε_i is random, Y_i is random as well.

It is often convenient to write equation 5.14 simultaneously for all i in vector form as follows:

$$Y = X\beta + \varepsilon \tag{5.16}$$

where Y is the (random) response vector, X is a $n \times p$ matrix of the inputs at each sample time, where n is the total number of responses and p is the number of explanatory variables, β is a $p \times 1$, unknown vector, and ε is an $n \times 1$ vector with distribution

$$\varepsilon \backsim \mathcal{N}\left(0, \sigma^2 I\right) \tag{5.17}$$

where I is the identity matrix. The output Y has a distribution of

$$Y \backsim \mathcal{N}\left(X\beta, \sigma I\right). \tag{5.18}$$

A practical application of OLS is to have a set of sample responses ($y = (y_i, ..., y_n)$) and explanatory variables and then solve for a set coefficients. Here, y_i is considered a realization of the random variable Y_i To do that, equation (5.14) has to be algebraically manipulated to solve for the betas and not the output Y_i.

We approximate β given X and y by finding the vector $\hat{\beta}$ that minimizes the sum of the squared residual errors. The residual error is the difference between the estimated response and the experimental response given by

$$r_i = y_i - \hat{y}_i, \tag{5.19}$$

where $\hat{y}_i$ is the predicted response $X_i\beta$. The mean of the squared residual errors approximates the variance of the distribution:

$$\hat{\sigma^2} = \frac{\sum_{i=1}^{n} r_i^2}{n}. \tag{5.20}$$

The least squares solution is given by $\hat{\beta} = (X^T X)^{-1} X^T Y$. The distribution of $\hat{\beta}$ is

$$\hat{\beta} \backsim \mathcal{N}\left(\beta, \sigma^2 \left(X^T X\right)^{-1}\right). \tag{5.21}$$

OLS assumes that the error in each measurement is independent of every other. If the experimental data violates this assumption then the confidence intervals, and prediction intervals, of the OLS estimations of the coefficients are not as accurate.

A way to avoid this issue is to use another type of regression analysis called feasible generalized least squares regression (FGLS). The difference between the FGLS and OLS is that the FGLS errors need not be independent. This can be seen mathematically because the FGLS variance matrix is no longer a diagonal matrix. The general FGLS equation is the same as equation 5.14, but the distribution of the response is now changed to

$$Y \backsim \mathcal{N}\left(X\beta, \Sigma\right) \tag{5.22}$$

where Σ is the variance. If the variance is known, then the least squares solution $\hat{\beta}$ is given as

$$\hat{\beta} = \left(X^T \Sigma^{-1} X\right)^{-1} X^T \Sigma^{-1} Y \tag{5.23}$$

with distribution of

$$\hat{\beta} \backsim \mathcal{N}\left(\beta, \left(X^T \Sigma^{-1} X\right)^{-1}\right). \tag{5.24}$$

But in practice, when Σ is unknown it has to be estimated when calculating $\hat{\beta}$ and then iterated to find the best possible solution. Because FGLS accounts for the relationship between each error it allows for better predictions of future events.

MATLAB's default implementation of FGLS assumes an autoregressive (AR) correlation structure of order zero or a AR(0) model, which is in essence the same as OLS. In other words, each error is assumed to be independent of the previous error. However, if an experiment shows that there is a strong correlation with errors before the previous error then the AR can be modified. Changing the order of the AR is denoted by AR(n), where n is the amount of errors back the AR is using in the computations. MATLAB's default setting is AR(0) to save computing power and time.

OLS and AR were applied to the sample data that was collected in January, 2016 and the results can be compared. First, some of the data points from the weather station were missing from the readings online so for the purpose of the model estimation the four missing values were imputed. Then all the data in MATLAB was placed into a matrix that contained the values for temperature, inverse pressure, and humidity in three columns. One of the first iterations of the code produced the following outputs

Coefficients	OLS Model	AR(1) Model	AR(2) Model
$\hat{\beta}_0$	-8626.7613	-8403.4698	-8623.3
$\hat{\beta}_{Pressure}$	307600	300820	307601
$\hat{\beta}_{Temp}$	0.8538	0.9793	0.8717
$\hat{\beta}_{Humidity}$	0.1356	0.0746	0.0714

Table 5.1: Output of the 'fgls' code giving all the OLS, AR(1), and AR(2) coefficients.

The first steps were done with just the OLS model, but upon further inspection of the errors, it was determined that a AR(2) model was needed as shown in figure 5.1. These estimates were created using the first 200 sample data points as the training data allowing for the last 96 sample points to be used as validation of the estimates.

To validate the estimates, the estimated coefficients and weather inputs were used to create an estimated value and then compared the OLS model and the AR(2) model as shown in figure 5.1.

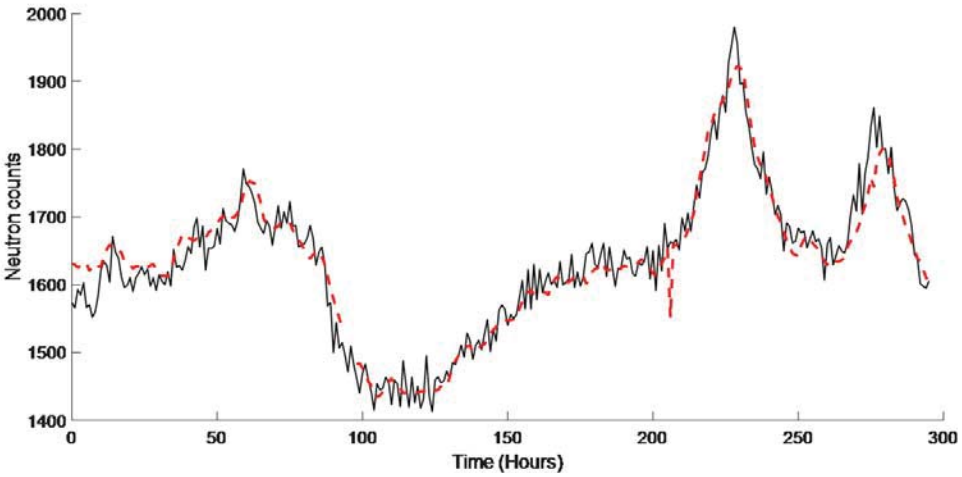


Figure 5.1: The FGLS prediction with the neutron counts from Febuary, 2016. The dashed line is the model AR(2) prediction and the solid line is the OLS prediction.

In figure 5.1 it is hard to distinguish between each estimate to compare which did a better job predicting the output. It may appear that the difference between the OLS and FGLS is inconsequential, but the next section highlights why the FGLS model is preferred.

5.3 Expectation and Variance of Errors in FGLS

In standard OLS regression (equation 5.14) the errors are assumed to be independent and identically distributed normal random variables. By contrast, in an AR model the errors are identically distributed normal random variables, but they are not independent. In particular, for the AR model of order 2, the error at time t is defined as:

$$\varepsilon_i = \phi_1 \varepsilon_{i-1} + \phi_2 \varepsilon_{i-2} + \omega_i, \tag{5.25}$$

where ϕ_1 and ϕ_2 are the AR(2) coefficients, and $\omega_i \overset{iid}{\sim} \mathcal{N}(0, \sigma_\omega^2)$ is a white noise error term. A standard reference is *Time Series Analysis and Applications* by Robert H. Shumway and David S. Stoffer [20].

Using equation 5.10, the variance of the errors is defined as

$$\sigma_\varepsilon^2 = \mathbb{E}\left[\varepsilon_i^2\right] - \mathbb{E}\left[\varepsilon_i\right]^2 \tag{5.26}$$

but if the system is stationary, meaning that it has the same distribution at any time, then it can be simplified to

$$\sigma_\varepsilon^2 = \mathbb{E}\left[\varepsilon_i^2\right]. \tag{5.27}$$

Because $\mathbb{E}[\varepsilon_i] = 0$. To see this, notice that

$$\begin{aligned}
\mathbb{E}[\varepsilon_i] &= \mathbb{E}[\phi_1\varepsilon_{i-1}] + \mathbb{E}[\phi_2\varepsilon_{i-2}] + \mathbb{E}[\omega_i] && (5.28)\\
&= \phi_1\mathbb{E}[\varepsilon_{i-1}] + \phi_2\mathbb{E}[\varepsilon_{i-2}] + 0. && (5.29)
\end{aligned}$$

By stationarity, $\mathbb{E}[\varepsilon_i]$ is the same for any time i, so

$$\begin{aligned}
\mathbb{E}[\varepsilon_i] &= \phi_1\mathbb{E}[\varepsilon_i] + \phi_2\mathbb{E}[\varepsilon_i] + 0 && (5.30)\\
&= (\phi_1 + \phi_2)\mathbb{E}[\varepsilon_i] && (5.31)\\
&= \frac{0}{1-\phi_1-\phi_2} = 0. && (5.32)
\end{aligned}$$

Knowing that $\mathbb{E}[\varepsilon_i]^2$ is equal to zero makes solving equation 5.27 much simpler.

$$\begin{aligned}
\sigma_\varepsilon^2 &= \mathbb{E}\left[\varepsilon_i^2\right] && (5.33)\\
&= \mathbb{E}[\phi_1\varepsilon_{i-1}\varepsilon_i + \phi_2\varepsilon_{i-2}\varepsilon_i + \omega_i\epsilon_i] && (5.34)\\
&= \mathbb{E}[\phi_1\varepsilon_{i-1}\varepsilon_i + \phi_2\varepsilon_{i-2}\varepsilon_i + \omega_i(\phi_1\varepsilon_{i-1} + \phi_2\varepsilon_{i-2} + \omega_i)]. && (5.35)
\end{aligned}$$

This can be simplified to

$$\sigma_\varepsilon^2 = \phi_1\gamma(1) + \phi_2\gamma(2) + \sigma_\omega^2, \tag{5.36}$$

with τ defined as

$$\tau(h) = \mathbb{E}[\varepsilon_i\varepsilon_{i-h}] \tag{5.37}$$

where h is some integer value. Under the assumption that the process is stationary, $\tau(h)$ does not depend on i. We now find $\tau(1)$ and $\tau(2)$ and use these to solve for σ_ε^2.

$$\begin{aligned}
\tau(1) &= \mathbb{E}[\varepsilon_i\varepsilon_{i-1}] = \mathbb{E}[\phi_1\varepsilon_{i-1}\varepsilon_{i-1} + \phi_2\varepsilon_{i-2}\varepsilon_{i-1} + \omega_i\varepsilon_{i-1}] && (5.38)\\
&= \phi_1\sigma_\varepsilon^2 + \phi_2\tau(1) + 0 && (5.39)\\
&= \frac{\phi_1\sigma_\varepsilon^2}{1-\phi_2}, \quad \text{and} && (5.40)\\
\tau(2) &= \mathbb{E}[\varepsilon_i\varepsilon_{i-2}] = \mathbb{E}[\phi_1\varepsilon_{i-1}\varepsilon_{i-2} + \phi_2\varepsilon_{i-2}\varepsilon_{i-2} + \omega_t\varepsilon_{i-2}] && (5.41)\\
&= \phi_1\tau(1) + \phi_2\sigma_\varepsilon^2 + 0 && (5.42)\\
&= \frac{\phi_1^2\sigma_\varepsilon^2}{1-\phi_2} + \phi_2\sigma_\varepsilon^2. && (5.43)
\end{aligned}$$

The equations for $\tau(1)$ and $\tau(2)$ can be substituted into equation 5.36 to solve for σ_ε^2 as follows:

$$\begin{aligned}
\sigma_\varepsilon^2 &= \phi_1\left[\frac{\phi_1\sigma_\varepsilon^2}{1-\phi_2}\right] + \phi_2\left[\frac{\phi_1^2\sigma_\varepsilon^2}{1-\phi_2} + \phi_2\sigma_\varepsilon^2\right] + \sigma_\omega^2 && (5.44)\\
\sigma_\varepsilon^2 &= \frac{\phi_1^2\sigma_\varepsilon^2}{1-\phi_2} + \frac{\phi_1^2\sigma_\varepsilon^2\phi_2}{1-\phi_2} + \phi_2^2\sigma_\varepsilon^2 + \sigma_\omega^2 && (5.45)\\
\sigma_\varepsilon^2 &= \frac{\sigma_\omega^2}{1-\phi_1^2[\frac{1+\phi_2}{1-\phi_2}] - \phi_2^2}. && (5.46)
\end{aligned}$$

It can be shown that the normality of the underlying white noise term ω_i carries through to ϵ_i. Thus, the marginal distribution of ε_i is $\mathcal{N}(0, \sigma_\varepsilon^2)$. When the previous two error terms are available, however, the autoregressive nature of the errors allows for a reduction of the variance. In particular, the conditional distribution of ε_t, given ε_{i-1} and ε_{i-2}, is $\mathcal{N}(\phi_1\varepsilon_{i-1} + \phi_2\varepsilon_{i-2}, \sigma_\omega^2)$. It is worth mentioning that this reduction in variance is not possible when using standard OLS regression, because OLS assumes the error terms are

independent and hence convey no information about each other. In practice, we will not know the true values of either the AR coefficients, or the variances, or the error terms, but estimates for these are available from computer output, and these can be used to create prediction intervals.

The prediction intervals, or error bars, can be calculated two different ways with the AR(2) model. The first way assumes previous errors are not available. This method uses σ_ε^2 (5.46) as the variance because it does not use the errors. It is known that the approximate point estimation is given by

$$\hat{y_i} = X_i\hat{\beta} + \varepsilon_i \tag{5.47}$$

with

$$\hat{\beta} \backsim \mathcal{N}(\beta, Var(\hat{\beta})) \tag{5.48}$$
$$\varepsilon_i \backsim \mathcal{N}(0, \sigma_\varepsilon^2). \tag{5.49}$$

The true distribution of the random variable $\hat{y}$ is given as

$$\hat{y_i} \backsim \mathcal{N}(X_i\beta, X_iVar(\hat{\beta})X_i^T + \sigma_\varepsilon^2). \tag{5.50}$$

Knowing the distribution of the true measurement, it can be assumed that the approximate distribution is

$$\hat{y_i} \dot{\backsim} \mathcal{N}(X_i\hat{\beta}, X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma_\varepsilon^2}). \tag{5.51}$$

Since the distribution is known and the expected value and the variance are known it is possible to calculate the prediction intervals (PI) by

$$PI_{Mobile} = X_i\hat{\beta} \pm 2\sqrt{X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma_\varepsilon^2}}. \tag{5.52}$$

However, if the model has access to the previous errors then the lag coefficients can be used and the distribution of the errors becomes $\mathcal{N}(\phi_1\varepsilon_{i-1} + \phi_2\varepsilon_{i-2}, \sigma_\omega^2)$ with $\sigma_\omega^2 \ll \sigma_\varepsilon^2$. Now the distribution of $\hat{y}_i$ is given as

$$\hat{y_i} \mid \varepsilon_{i-1}, \varepsilon_{i-2} \backsim \mathcal{N}(X_i\beta + \phi_1\varepsilon_{i-1} + \phi_2\varepsilon_{i-2}, X_iVar(\hat{\beta})X_i^T + \sigma_\omega^2). \tag{5.53}$$

The same assumption can be made to move from the true value of y to the estimated value in this case with one slight change. ε_t is not known because the true value of y is not known. So instead of ε_t the estimate is called residual error and denoted as r_t. The residual error is the difference between the estimated value and the measured value at time t. Using the residual errors the approximate estimation is given by

$$\hat{y_i} \mid r_{i-1}, r_{i-2} \dot{\backsim} \mathcal{N}(X_i\hat{\beta} + \hat{\phi_1}r_{i-1} + \hat{\phi_2}r_{i-2}, X_i^T\hat{Var}(\hat{\beta})X_i + \hat{\sigma_\omega^2}). \tag{5.54}$$

The prediction interval is found by adjusting equation 5.52 with the new mean and variance of the approximate distribution.

$$PI_{Stationary} = X_i\hat{\beta} + \hat{\phi_1}r_{i-1} + \hat{\phi_2}r_{i-2} \pm 2\sqrt{X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma_\omega^2}}. \tag{5.55}$$

Considering the residual errors allows for the model to account for the predictability of the related errors and adjust the prediction based off that relationship. OLS assumes that each error is completely independent and therefore does not have the ability to reduce the predication interval. In practice the $\sqrt{X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma^2}}$ terms are dominated by the magnitude of $\hat{\sigma}$. And we find that $\hat{\sigma}_\omega^2$ is an order of magnitude smaller than $\hat{\sigma}_\varepsilon^2$, resulting in tremendous gains of the AR approach versus OLS.

The danger of this second method is that as the detector moves through an area if SNM is introduced into the detection range slowly enough the model would hide the source in the residual error. As long as the SNM never increased the detector response above the alarm threshold in a single time period, the model will incorporate the signal into the background. However, this method would be very powerful for a stationary detector in a fixed location with constant moving traffic and good measurements of the residual errors. A mobile system the model should utilize the AR(2) mobile that would not be influenced by the previous residual errors.

Chapter 6

USNA Experimental Setup

6.1 Satellite Location

The satellite location used was located at 238 Bullard Boulevard, Naval Support Activity, Annapolis, Maryland. This location was chosen because it limited variables that would could influence the neutron background. The satellite location is a small building, shown in figure 6.1 below, that is mostly isolated from high "z" materials and traffic. High "z" materials are materials such as uranium, plutonium, and shielding materials such as lead, steel, or aluminum.[19] that will will change the amount of neutrons present. Large structures that contain large amounts of metals will create the ship effect around them and create more neutrons than would normally be present. Larges amounts of vehicle traffic can also introduce enough metal to create the ship effect on the LNS readings. But, due to the location, the satellite building is removed from any large structures and exposed to low amounts of traffic.

Figure 6.1: The satellite building from the outside.

The building was full of old equipment that had not been used in years, but need to be moved out. There was a second small wood shed behind the building being used for this project that was mostly empty. All of the old equipment took a few days to move into the second building and then clean the first building for use. Once the building was cleaned, the LNS detector was moved into the building and connected to a power source. The LNS was placed as close as possible along one wall, as shown in figure 6.2 to allow for as little shielding as possible from the building. Since then the LNS has been operational and collecting data.

Figure 6.2: A look at the setup inside the satellite building. The LNS along the back wall, the computer conected to weather station 2, and the memory unit for weather station 2.

The second sensor to be installed was weather station 1 shown in figure 6.3. The system arrived with the parts to be mounted on a 1.5 inch pipe, but the existing pipe mounted on the building was a larger pipe and the factory brackets did not work. It took several days to acquire the necessary parts to complete the installation of weather station 1. Once the station was in place it could be turned on and calibrated. The external sensors are powered by a battery that is recharged during the day by a solar panel and transmits the data wirelessly to a receiving unit inside the building. If the solar panel is not pointing in the correct direction, then during the day the battery is not completely recharged, and the sensors would run out of power in the night. Once that issue was identified the solar panel position was changed to optimized the amount of sun light it would receive and the issue went away. The receiving unit inside was powered by a wall socket with a battery backup and stores up to twenty-two days of data.

The second weather station was the last sensor installed, also shown in 6.3, after the first two sensors were installed. The second weather station was much harder to install because of the way it was manufactured to be mounted. This weather station was meant to be seated on the top of a one inch pipe But there was no such place on the building and one had to be added. The plans where drawn and given to the Naval Academy's machine shop to be manufactured. By 21SEP17, the brackets where finished and ready to be mounted the station. But due to having insufficient tools at the satellite location, the initial measurements of the existing pipe were incorrect. The pipe was estimated to be a $1\frac{3}{4}$ inch pipe and the shop made it to those dimensions. When the brackets where attempted to be placed on the pipe they would not fit because the pipe was really a $1\frac{7}{8}$ inch pipe. So the brackets where returned to the machine shop to be corrected for the new dimensions. On 28SEP17 the weather station was finally mounted and functioning on the building.

Now that the second weather station was installed it had to be connected to a memory source and a power source. There is a single cord running from the bottom of the unit that has a serial connection and power wires in-place for the station. The wire runs through a hold drilled into the attic of the building and then down a wall to connect to the data storage and power outlet. The data storage is a computer left on to receive serial communications from the station and record them.

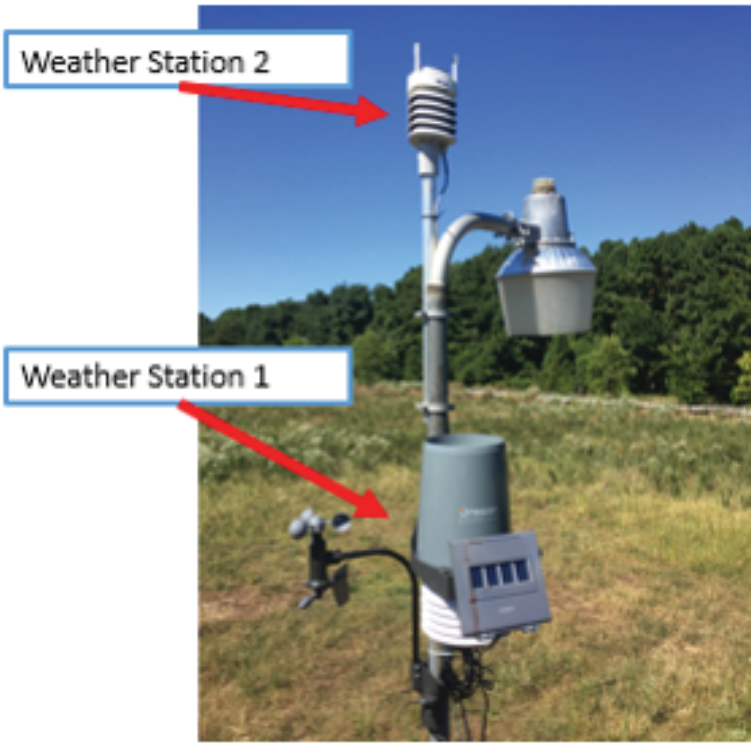


Figure 6.3: Weather station placement on the satellite building

Once all the data was received from the sensors, it was imported into MATLAB to begin the modeling process. In MATLAB each sensor's time stamps was checked to make sure that the sensors did not give two readings with the same time stamp or skip any readings. MATLAB found a few errors in the time series that were dealt with individually. When the data was being used for modeling, the times when one of the sensors was not functioning, was excluded from the modeling process. All the sensors were installed and collecting data by 01SEP16.

6.2 Data Collection

After two months of data collection there was enough data collected to start inspecting it. The first data looked at was all the neutron data as shown in figure 6.4. Upon inspection the data collected looked as it was expected. It was noted that among the expected variations there appeared to be two large drops in the neutron background on 19SEP17 and 28SEP17. The drops where larger than previously seen drops, but the data collection had just started and it was unknown if those were common.

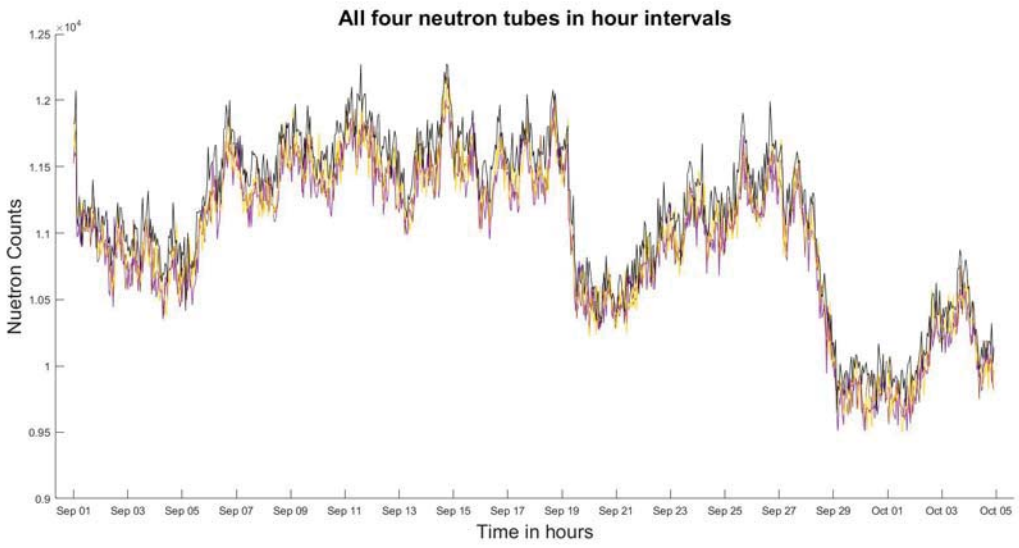


Figure 6.4: The neutron background measurements from 01SEP16 through 20OCT16

The second data set inspected was the pressure readings. The black line, with finer resolution, in figure 6.5 represents the data from weather station 1, while the yellow more square data is weather station 2. Weather station 1 can measure pressure down to a hundredth of an inch of mercury while station two is limited to within a tenth of an inch of mercury. For the purpose of modeling that means that the pressure output from weather station 1 is more useful mathematically because it is more precise and accurate.

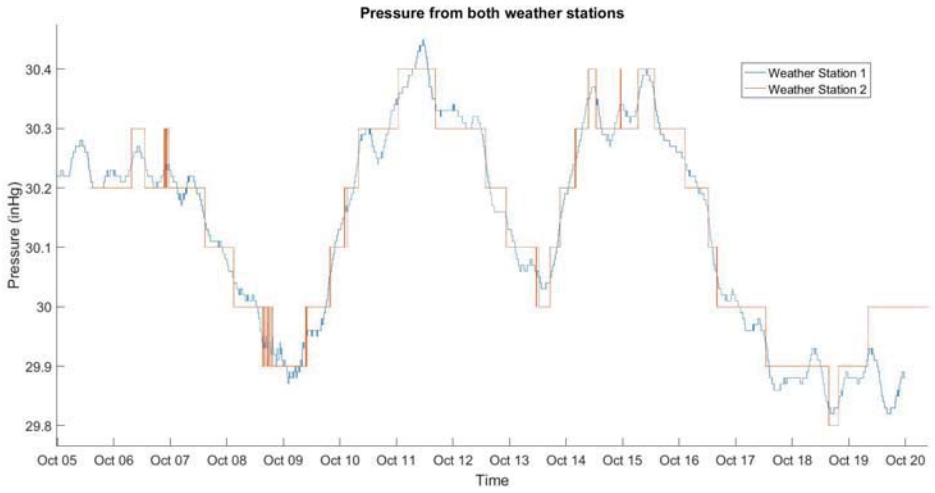


Figure 6.5: Pressure readings from two weather stations. The line with more resolution is from weather station 1 and reads down to a hundredth. The second line is from weather station 2 that only measures down to a tenth.

Humidity was the last variable inspected, this variable is not the absolute humidity, but rather the relative humidity. Relative humidity is the measurement of how much water the air is holding as a percent of the total water it could hold at that temperature. It is not a good measurement of the total amount of water in the air. However, given either the relative humidity or the dew point and temperature it is possible to calculate the absolute humidity of the air. In figure 6.6 the humidity readings from weather station 1 appeared to saturate at 100% at some points, but by using the dew point that can be mitigated. Weather

station 2 appears to be reading some constant below weather station 1, but both stations were tracking the same trends of the humidity.

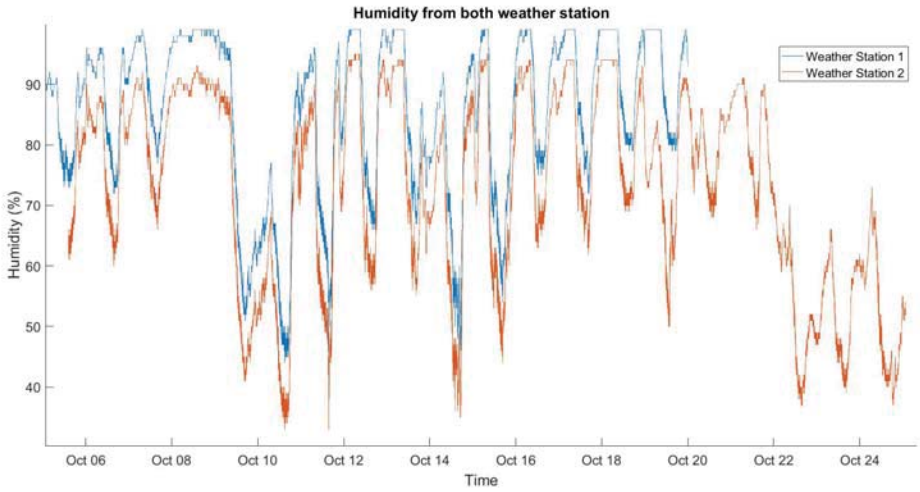


Figure 6.6: Humidity readings from two weather stations. The top line is from weather station 1 and staturates some times. The lower line is from weather station 2.

Weather station 2 was used to confirm weather readings throughout the modeling process, but was not used for the modeling. Weather station 1 gave more resolution in the measurements, more variables, and better data format and was used as the primary source of data for this project.

Chapter 7

Database Development and Model Variables

7.1 Database Development

The first step of modeling was organizing the data into the format that MATLAB needed for the FGLS code to work correctly. When the data from each different sensor was imported into MATLAB it was placed into different cell arrays that allowed for the timestamp and information from each sensor to be kept intact and not mixed with any of the other data. But the original data had imperfections in it such as missing sections of time or two readings from the same minute in the data. This created an issue when the code was trying to match the numbers because the data array dimensions, corresponding to the imperfections in the time series, did not match between the sensors. For the FLGS code to run, a database with all the information from each sensor and paired with the other data with the correlating timestamp was developed.

7.1.1 Time Series Development

To take each sensor data from compartmentalized cell arrays and merge it all into one large matrix took some code development. The basic idea was to create a master time series independent of any sensor to ensure it contained every minute for the collection period and did not have any duplicate times. Then a script was written that would check the timestamp of the collected data against the master time series to align all the data. The simplest way to do that was to have the program search the master time series for the correct time and then match the data. However one of the slowest functions in MATLAB is the function accessing a time variable to verify that the date and time is right, when the basic script was written it initially took about twelve hours to sort two weeks, or $20,160$ minutes, of data from each sensor. To reduce the processing time of the program the code was modified to contain a 'for' loop that remember the index of the last timestamp it matched with, added one, and started there when searching for the next correct timestamp. This modification was not hard to write, but it did not work because each sensor had different issues with the time series that had to be identified and accounted for. This issue will further discussed in the following paragraphs.

Weather station one had an issue where it occasionally recorded two measurements in the same minute. Since the master time was made for a single data point within one minute, when the 'for' loop added one to the index it moved the code past the minute it needed. A side effect of a code written with a 'for' loop like this is that it will simply add one to the index when it does not find the correct timestamp on the first try. Now instead of the index being off by one, the index has moved two minutes past the correct timestamp. As the loop runs over and over adding to the index it will run to completion never finding the correct timestamp. To fix this issue the code must identify when this error is occurring, know the index when it started, and then adjust. This issue was fixed by adding an aging variable inside the 'for' loop that remembered the first time a timestamp did not match. With the correct set of conditional statements it can identify this error and fix it before the program terminates.

The LNS had two issues the first was just like the weather station with repeated measurements, but the second was an completely new issue. The LNS would occasionally skip a minute and have no timestamp to match with the master series at all. In effect this was the exact opposite issue from weather station one. Now the code had to be adjusted to recognize when it needed to just skip a minute and check the timestamp in the master series. To do that, an error term was added to the index of the master time series variable that could account for that issue without messing up the index variable of the sensor data. This was again solved by adding a layer of conditional statements and aging variables within the 'for' loop.

The second weather station only issue was that sometimes the recording program on the computer would crash and lose time periods of data. This was easy to identify when shorting times using the methods talked about above. However, since weather station 2 was not used in the modeling process and did not reacquire all of the data to be exactly the same as the other data that went into the MData matrix, discussed in section 7.1.2, and was only used to verify weather station one data. Therefore, weather station two did not require a very complicated code for the purpose of this project.

7.1.2 Master Matrix

The initial master data matrix, containing all the data in a single place, through the end of December was a $175,680$ by 8 column matrix containing data on a minute basis. The columns contained the following variables: constant column of ones, two columns of regime variables (talked about in Section 8.1) average neutron counts, temperature, inverse humidity, dew point, and inverse pressure. As different ideas and variable where introduced into the project this master array grew to over twenty variable or different relationships of variables. One method that was tested was trying the inverse of all the variables to see how that influenced the model, but it only changed the coefficients without improving the models performance. Shown below are several lines from this master matrix, called MData for Model Data.

Constant	Regime 2	Regime 3	Count	Temp	Humidity^{-1}	Dew point	Pressure^{-1}
1	0	1	10205	62.8	0.0108	60.8	0.0331
1	0	1	10128	62.4	0.0105	60.8	0.0331
1	0	1	10148	64.6	0.0111	60.8	0.0331
1	0	1	10171	63.5	0.0110	60.8	0.0331
1	0	1	10274	63.0	0.0110	60.8	0.0331

Table 7.1: Sample rows from MData matrix at hourly time intervals.

The MData matrix is first created on the minute reading basis and then used to group the data into other time frames. For the modeling purposes, the neutron counts where summed over the period of an hour and the weather data was taken from readings at the top of each hour. This matrix was added to as more data was collected and at the end of February 2017 was 4343 by 10. The extra two columns held variables of precipitation and cloud cover, but as neither were included in the final model (see section 7.2) they are not included in the tables above.

7.1.3 Imputing Weather Data

For MATLAB to correctly run the autoregression code it could not contain any missing values within the whole MData structure. There were a few points within the data where the weather station was disabled and was missing readings for a period of time. Each of those points were identified and data was imputed using linear approximation. One such time was at MData index 2238 the weather information was not included, but both the hour before and after contained measured values. The weather data for 2238 was estimated by taking the difference between the two known values, dividing by two, and then adding to the first value. The result is shown in table 7.2 with index 2238 being the middle value.

Constant	Regime 2	Regime 3	Count	Temp	Humidity^{-1}	Dew point	Pressure^{-1}
1	0	1	10929	43.3	0.0154	32.0	0.0333
1	0	1	10968	43.1	0.0154	32.0	0.0333
1	0	1	10942	43.0	0.0154	32.0	0.0332

Table 7.2: MData matrix with imputed weather values. The top and bottom rows are true weather measurements and the middle row is the linear approximation of the missing weather values.

Within the whole matrix of 4343 hours of data the weather data was imputed at 25 places. While the imputed values are not the true values, the variation of the weather over that period is not great enough to create large discrepancies between the imputed value and true. Twenty five values spread out is not enough values to have any substantial impact on the final estimate of the model coefficients.

7.1.4 Imputing Neutron Data

The same issue of missing weather data in MData was also present in the neutron data, however a very different method for imputing data was used for the neutron counts. Due to the amount of natural variation between each hour of neutron counts just using a linear approximation was not the correct approach. The OLS model needs every value within the matrix to calculate the lag constant ϕ but not to estimate the beta coefficients. So a model was created with FGLS and then, using the weather variables, a value for the neutron counts was made and imputed into the matrix. Just like in the weather imputed data, there where not many places this method was used and it was not often enough to affect the outcome of the coefficients. The reason was to allow for the model to have every value it needed to estimate the lag coefficients ϕ.

7.2 Model Variables Added

The initial model relied on just pressure, temperature, and relative humidity but as the modeling process matured it became apparent that there was something else that was influencing the neutron counts. The model was good based off the current inputs, but could become better if the other factors are identified and used in the final model. This project looked at dew point, cloud cover, and total precipitation as being another factors in the model. The dew point was used because the weather station was measuring it and initially it was not difficult to include it in the model. By the final model however, dew point had replaced the humidity measurements. The dew point and temperature can be used to calculate the relative humidity and it became apparent that dew point was providing the same information that would be included from the humidity readings. Relative humidity was dropped because when the AR model was created the standard error range of the beta coefficient included zero. Statistically, when the coefficient includes zero within its confidence interval it is not relevant to the model, or at least does not have much contribution to the model.

Cloud cover and precipitation information was attained from the National Oceanic and Atmospheric Administration (NOAA). The data was collected at the NOAA weather station located on Hospital Point at the U.S. Naval Academy. The station recorded about twenty-five different weather conditions every hour and all the data is accessible through the NOAA website. The cloud cover is described with a number, between zero and eight (okta) with zero being no clouds in the sky and eight being completely overcast conditions, and an altitude that the cloud coverage starts at. The sensor has the ability to split the conidions into three separate layers giving amout of cloud cover and altitude it starts at. A sample of the data is shown in table 7.3.

Station	Sky Conditions	Precipitation $\frac{in}{hour}$
WBAN:13752	FEW:02 20 BKN:07 28 OVC:08 49	0.28
WBAN:13752	SCT:04 19 BKN:07 28 OVC:08 49	0.28
WBAN:13752	BKN:07 36	0.00

Table 7.3: Sample output of the NOAA sky conditions in the data file.

The first column is the ID of the weather station at the Naval Academy, the third column is total amount of rainfall, in inches per hour, within the past hour. The second column contains the useful cloud cover information. In row one and two it shows how the data is split into three different layers in the sky. The first reading of "$FEW : 02$" is a description of the clouds, the "02" is the zero through eight descriptor is the okta talked about previously. A complete table with the possible values is shown below. This is the value that the model actually used for the autoregression. In the second set of numbers, "20" is the altitude in hundreds of feet at which the cloud cover is located. In this case there are a "few" clouds starting at 2000 feet. The second layer is "broken" clouds starting at 2800 feet and so forth of the readings. It was decided that the clouds starting the highest in the atmosphere would be the best value to keep for the model. So a MATLAB code went through and found the number description of the clouds $(0 - 8)$ farthest to the right in the array and used that value.

Coverage	Coverage Descriptor	Oktas Numbers Possible
Clear sky	CLR	00
Few	FEW	01-02
Scattered clouds	SCT	03-04
Broken clouds	BKN	05-06
Overcast	OVC	07-08

Table 7.4: Possible cloud descriptions from NOAA weather data.

The model was trained from 06OCT16 through 19JAN17 with temperature, inverse pressure, dew point, cloud cover, and precipitation as the inputs. The in-sample results were examined and the following plot shows the contribution of cloud cover to the overall model. The solid line was the estimated neutron counts with the left y-axis showing the magnitude. The dashed line was the total counts add or subtracted due to just the cloud variable. In other words, the solid line is the product of all the inputs, $X_i\beta$ and the dashed line is just a single variable contained within the larger model.

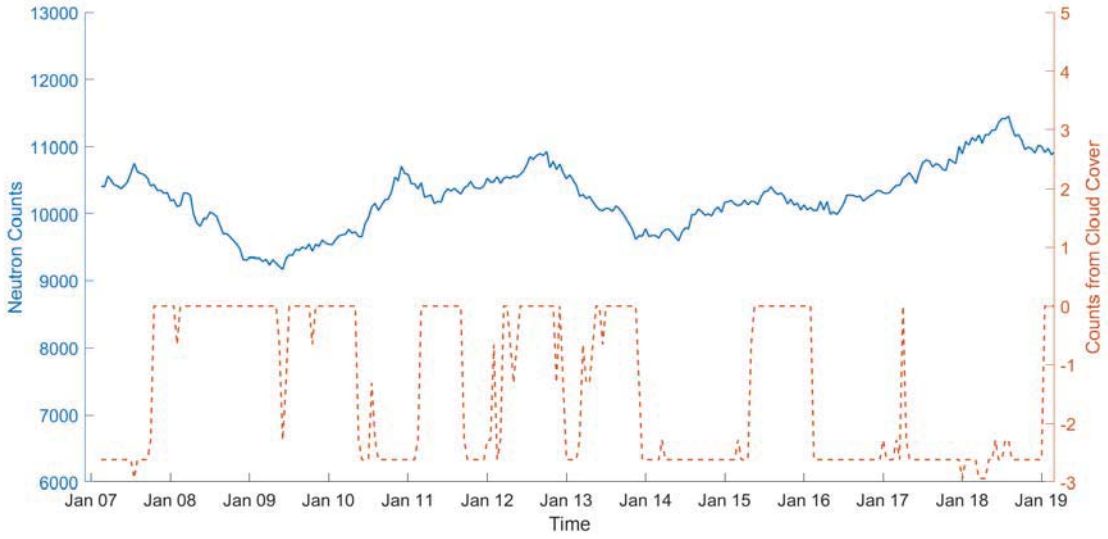


Figure 7.1: The AR model in-sample model from 01JAN17 through 19JAN17 showing total estimate and the contribution from just cloud cover. The solid line is the total estimate and the dashed line is the contribution from the cloud cover variable.

The magnitude of the neutron estimation is about 10500 counts per hour and cloud cover would only change that number at most by three counts, as shown if figure 7.1. That means that cloud cover amounts to less than 0.0286% of the counts in this model and is statistically so insignificant that there is no reason

to include it in the model.

The same method was used to see the contribution from precipitation over the same time period.

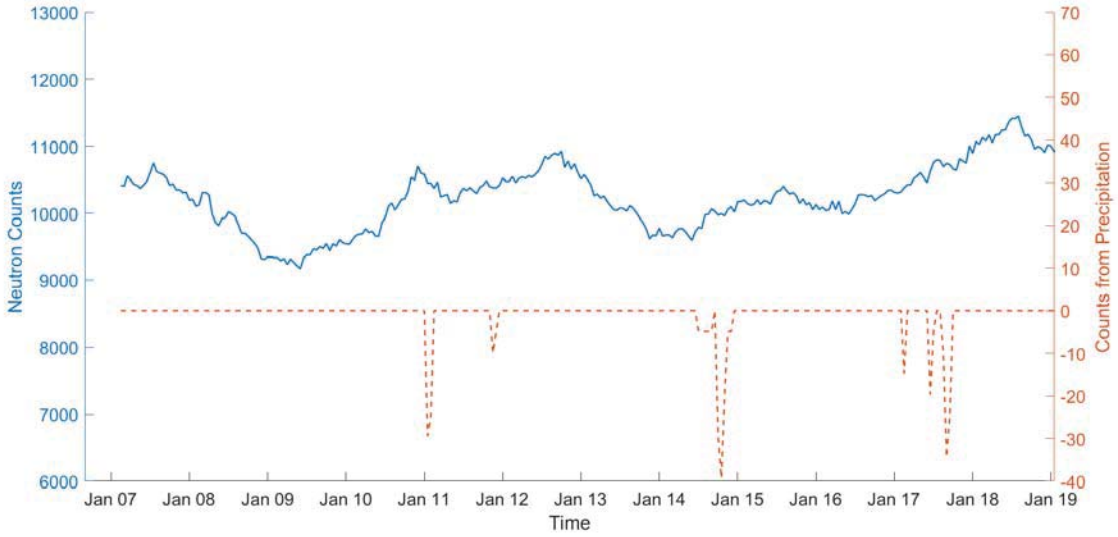


Figure 7.2: The AR model in-sample model from 01JAN17 through 19JAN17 showing total estimate and the contribution from just precipitation. The solid line is the total estimate and the dashed line is the contribution from the precipitation variable.

Precipitation changes the total neutron counts by a max of forty counts an hour over these nineteen days in January, 2017. This is slightly more than cloud cover added to the overall model, but still so small that it was determined to leave precipitation and cloud cover out of the final model.

7.3 MATLAB Code

MATLAB has different functions that can calculate an AR(2) model, the first function used was the 'fgls' function. This function calculates the both the OLS and the FGLS coefficients that is very useful in the beginning stages of the project. At the start, it was unknown how related the residual errors were and how many lags needed to be included. For comparison between the OLS and FGLS, it was very convenient to have both outputs for each iteration. The actual line of code used in MATLAB is:

fgls(MData,Y,'innovMdl','AR','arLags',2,'display','final','numIter',20).

This is the main line of code for this model development where MData is a $n \times p$ matrix containing a constant row and all the desired weather variables and Y is a $m \times 1$ matrix with the measured neutron counts. The next important parameter is the " 'arLags',2" piece of code. The 'arLags' controls the amount of lags, or ϕ coefficients calculated. The last parameter, 'numIter', defines the max amount of iterations the code will do to minimize the residual errors. This was capped at 20 to limit the amount time it would take the code to run.

The 'fgls' function was useful for a time, but eventually a different MATLAB function was used called 'regARIMA' function. This function was used because it made more parameters about the distribution available. The 'fgls' function did not output the variance of the the coefficients or provide the σ_ω^2 that are needed to calculate the prediction intervals in equation 5.55.

The two main lines to the 'argARIMA' code are:

Model1 = regARIMA(2,0 ,0)
[EstMdl,EstParamCov,logL,info] = estimate(Model1, Y ,'X',MData)

were ‘regARIMA(2,0,0)’ tells MATLAB to make a AR(2) model. Then using the MATLAB ‘estimate’ function with the inputs of neutron counts (‘Y’) then the explanatory variables (‘MData’). This code output much more information about the distribution and therefore was the main code used to create the final model as seen in the ‘Final code’ section of the appendix. The complete MATLAB code for this project is included as an appendix.

Chapter 8

Modeling Results

Before the results of this project are discussed, the four different types of modeling need to be clearly defined. A naive model is created by collecting data over a long period of time and then without any inputs predict that the next reading will be the mean of all the previous data collected. This is the most basic model possible and is considered the baseline, that needs to be improved upon, in this project. The first model created during this project was an OLS model that requires inputs and was discussed in Chapter 5. The final model type of model is based on an AR(2) foundation with two variations, the difference between the mobile and stationary model was discussed in section 5.3.

All of the models talked about here were based on an hour interval scale. The neutron counts were collected on a minute basis and then summed over an hour. All the weather data were the values from the reading at the top of the hour. Therefore, all of the results and model coefficients give an output assuming that same time interval. For different time intervals this process can be redone and different values for the coefficients obtained, but this project did not deal with that.

This chapter discuses the results from several different types of modeling to include in-sample modeling, naive modeling, and OLS modeling. But the final product of this project is the AR(2) mobile and stationary models that are discussed last.

8.1 In-Sample Results

There are two different types of prediction that are commonly used to measure the effectiveness of a model, in-sample modeling and out-of-sample predictions. In-sample modeling is when the model is trained on one set of data and then used to predict values within the same set of data. Using this method the model receives the largest amount of data possible to estimate the coefficients. The output is checked against the original data to see how close the predicted values are to the true values, in essence it is checking how well the model fits the data. Out-of-sample modeling is when a subset, almost the whole set, of data is used to estimate the coefficients. If the total data sample set has a hundred readings in it, out-of-sample modeling may only use seventy of the readings and then predict over the last thirty the model was not trained on.

As the model is being developed, in-sample modeling is very useful to identify issues within the model before the final product of out-of-sample prediction is attempted. The purpose is to potentially identify times that a variation is measured but not captured in the input variables or to confirm that the model can predict given the current inputs. Figure 8.1 shows a time period of in-sample modeling where the model is shown along with the measured values. The model was trained on data from 06OCT16 through 31JAN17 and then plotted from 21NOV16 through 31DEC16 to show the in-sample performance.

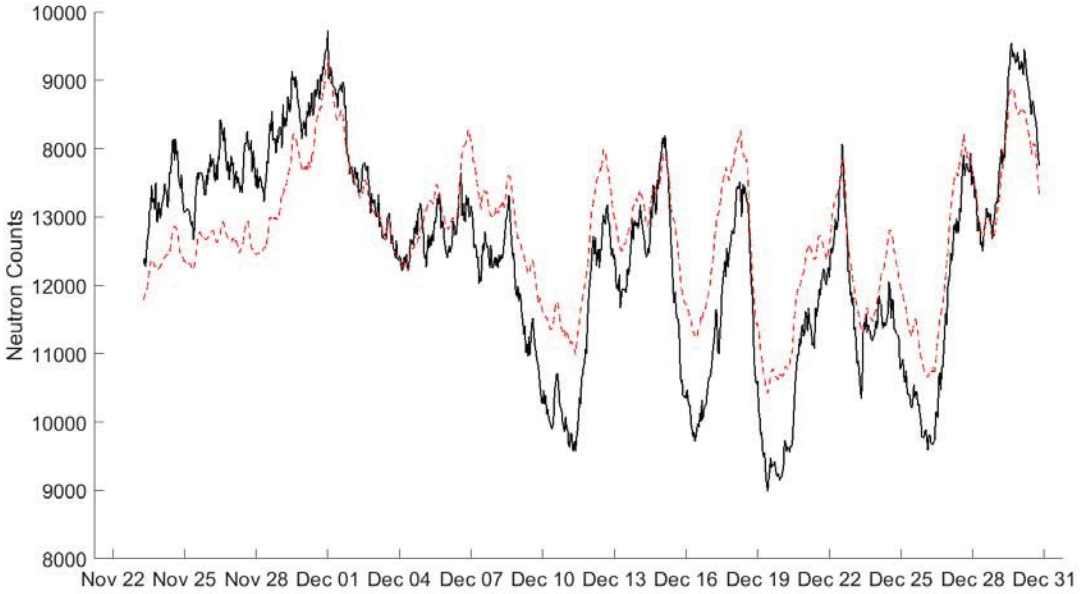


Figure 8.1: This is an observation of the in-sample estimation with the measured value of neutron counts from 22NOV16 through 31DEC16. The solid line is the measured counts and the dashed line is the estimated values from the OLS model.

Some time periods did better than this period in matching the model to the data, but some also showed great amounts of deviation from the expected model. In-sample modeling showed two times that the measured neutron counts dropped several hundred counts and the model had no indication that such a change should happen. Figure 8.2 shows the model's performance from 07SEP16 through 12OCT16. The top plot shows the measured values and the estimation while the bottom plot shows the error between the measured and estimated values in the top plot.

Figure 8.2: In-sample estimation from 07SEP16 through 12OCT16. This shows the two large drops of neutron counts noted in section 6.2 and how OLS predicts over the same time period.

On the 19SEP16 and 28SEP16 there are two large discrepancies between the model and measured counts. These sudden changes appear to be some short of 'regime' change in the neutron background that needed to be investigated. The first result of this observation was to investigate other possible influences that could result in this change that was not already included in the model. But as time passed no other instances of these sudden changes happened without the modeling adjusting for the change.

The second result was to search for a second independent source of neutron counts to verify that the LNS was operating correctly. A second source of neutron counts was located within forty miles of the first detection station with data available for most of the time period in question. The following figure, 8.3, is the normalized data from the two independent sources, both helium-3 detectors, over the same time period. The data was normalized by dividing the total counts by the average number of counts from each detector respectively. This method will be used for several plots in this section to better understand the percent variations of the data and how well the each model preforms. The second source had data from 01OCT16 through 07JAN17 and the data showed 0.82 correlation between the two sets of data.

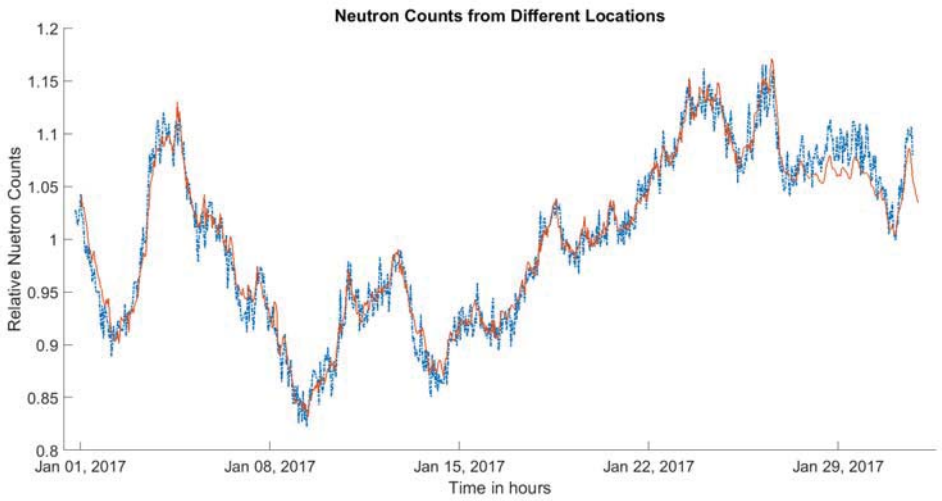


Figure 8.3: Neutron counts from LNS and a second detector. The dashed line is the LNS counts and the solid line is the second detector counts. The total neutron counts was divided by the mean of each detector to normalize the data for all of January, 2017.

After trying several modifications to the model to account for the regime changes it was decided, to indicate to the model what regime it was in and see how the results looked from that. So two columns of 'regime' numbers were added to the MData matrix, seen in table 7.1. From the start of the data collection until the first drop on 19SEP16 both of the regime constants were zero, and after the drop 'regime 1' was changed to a one. This in effect was just changing the β_0 term while allowing the rest of the coefficients to stay the same. After the second regime change 'regime 1' was set back to zero and 'regime 2' was changed to a one. This created two new coefficients in the OLS model that produced the following results.

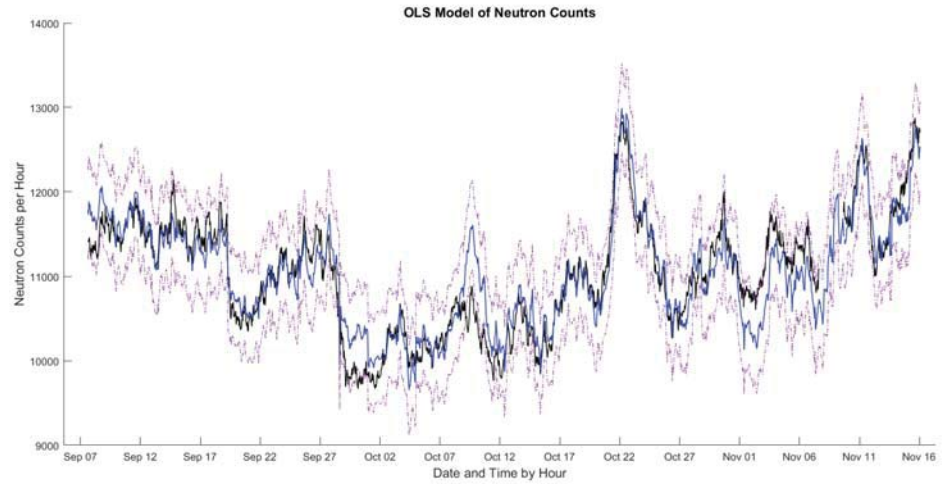


Figure 8.4: This is the in-sample estimation from 07SEP16 through 16OCT16 from the OLS with regime identifiers included. The black line is the measured counts, the blue line is the OLS estimation and the dotted lines are the prediction intervals of the OLS system.

This model shows that besides those two regime shifts the model does very well predicting the variations of the background. Because it was never again observed that these regime changes occurred, it was impossible to verify with a second detector that the shifts were a result of some unknown factor. That led to the decision that these shifts were either abnormal and until further instances are observed impossible to model or there

was an error with the LNS at those times. Since there was not enough data to support either of those ideas, we decided to not model those time periods and all the further modeling for this project started on 06OCT16, a few days after the last regime change.

8.2 Out-Of-Sample Results

8.2.1 OLS Model Results

The point of a predictive model is to predict the background and reduce the error fluctuation to, as close as possible, zero. A naive model is to predict the average value of counts for the next time. If that is the case then during the month of February, 2017 the natural fluctuation is from a low of 10201 to a high of 13627 counts. So in essence the best this basic model can do is $\pm 2.02\%$ by taking two times the standard deviation of the errors. The goal of the more advanced models is to drive that fluctuation to zero and know as precise as possible the counts at a given time. The first model developed was based on OLS or an AR(0) model. The model was trained on data from 06OCT16 through 31JAN17 and then used to predict, out-of-sample, during the month of February, 2017. The top plot shows the measured counts and the estimated counts by the OLS model. The second plot shows the difference between the two lines of the first plot, this is the residual errors that the model is trying to make zero. Table 8.1 contains the OLS coefficients and the average number of measured counts for the month of February, 2017.

Parameter	Value	Standard Error
$\hat{\beta}_0$	-66065	1201.59
$\hat{\beta}_{Pressure}$	2302600	36520.78
$\hat{\beta}_{Temp}$	32.4	1.64
$\hat{\beta}_{DP}$	-27.2	1.38
Average counts	1161	N/A

Table 8.1: The OLS coefficients used for the model.

These coefficients represent the relations that we expected to see from the discussions in chapter 2.3. As expected the inverse pressure variable is the largest factor of the background with both dew point and temperature being important, but not as large of factors. These coefficients are the values for equation 5.14. The following plots have been normalized to highlight the variation and the ability of the model to predict values.

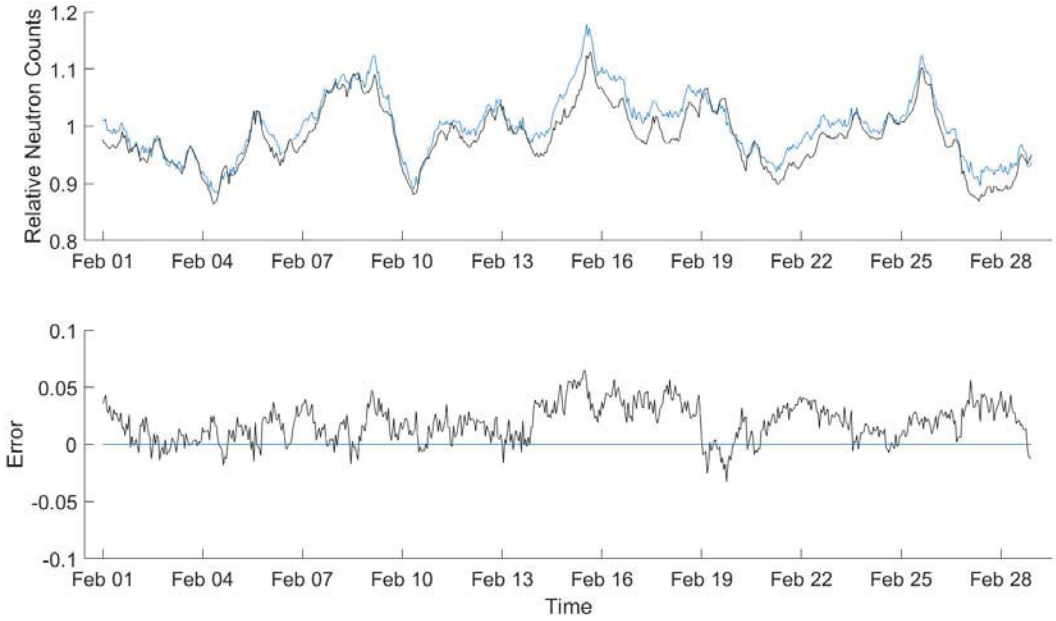


Figure 8.5: This is the out-of-sample OLS model with the measured values during February, 2017 and the residual errors. In the top plot, the solid blue line is the measured counts by the LNS and the dotted black line is the prediction of the OLS model. The bottom plot is the residual errors over the same time period

For comparison, the following plot is the error from both the naive and the OLS models. The naive models residual errors fluctuates ±13.25% from the average value, but the OLS estimate only fluctuates ±7.26%.

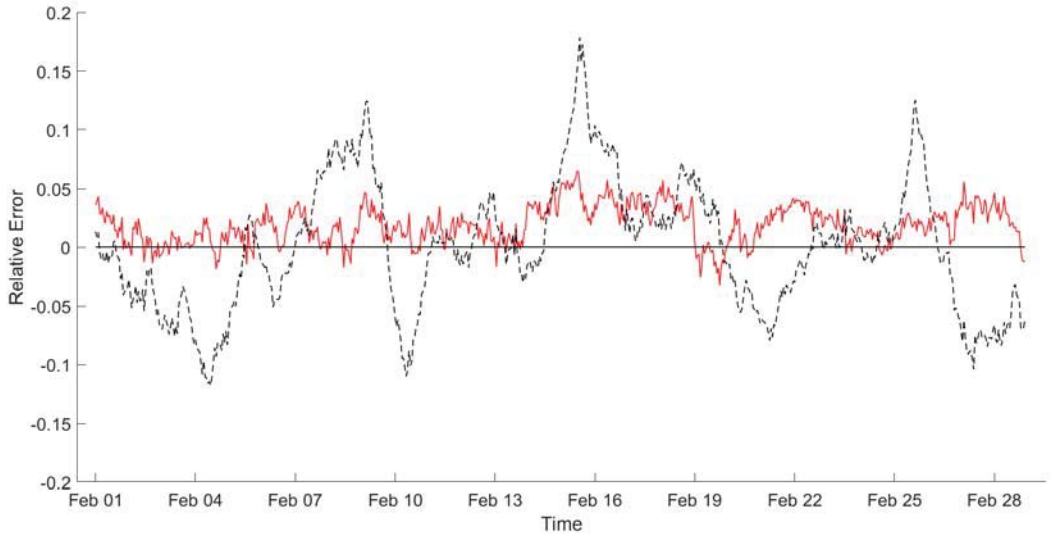


Figure 8.6: This figure compares the results of the naive model with the OLS model be comparing the ressidual error of the two models. The solid line is the OLS residual errors and the dashed line is the naive residual errors.

The OLS model reduces the residual errors when compared to the naive model, but OLS operates with

several assumptions that are violated by this model. One such assumption is that the mean of the errors has a mean of zero as shown in equation 5.15. However, figure 8.5 shows that the errors do not have a mean of zero. Rather the errors appear to be related and always positive over a long period of time, for example from 14FEB17 through 19FEB17. Due to violation of assumptions, the OLS is not the best method for modeling the neutron background.

8.2.2 AR(2) Mobile Results

The AR(2) model is the modified OLS model that has been discussed in section 5.2. The '2' means that it is accounting for a relationship of the last two residual errors with the current estimation. If it was an AR(n) model, then it would be accounting for the last n residual errors being related. The computer modeling output is changed to include coefficients that describe the relationship of the past errors called the lag coefficients. If the lag coefficients are not used then the AR(2) model almost has the same performace as the OLS model. But it will be shown, using the model explained in section 5.3, how the AR(2) model is the correct and much more powerful way to model for this data on an hour time interval. If a different time interval is chosen, then some investigation will be need to correctly modify the AR(n) choice. It does not impede the model to include more lag coefficients, it just takes more computing processing power to calculate and implement the coefficients. For the hour time interval the relationship between the residual errors becomes insignificant after two hours and are no longer relevant to the model.

The AR(2) model was trained on the same data as the OLS model from 06OCT16 to 31JAN17 and then used to predict over the month of FEB17. The coefficients are shown below as well as the graph showing the models performance.

Parameter	Value	Standard Error
$\hat{\beta}_0$	-49793.9	2091.92
$\hat{\phi_1}$	0.6694	0.01257
$\hat{\phi_2}$	0.3130	0.0123
$\hat{\beta}_{Pressure}$	1822371.7	63311.6
$\hat{\beta}_{Temp}$	9.0432	0.88654
$\hat{\beta}_{DP}$	-6.6029	0.97687

Table 8.2: The coefficients for the AR(2) Model

The standard error of the coefficients are used in MATLAB to create the $p \times p$ matrix $\hat{Var}(\hat{\beta})$ used in equation 5.52 and 5.55 to calculate the prediction intervals.

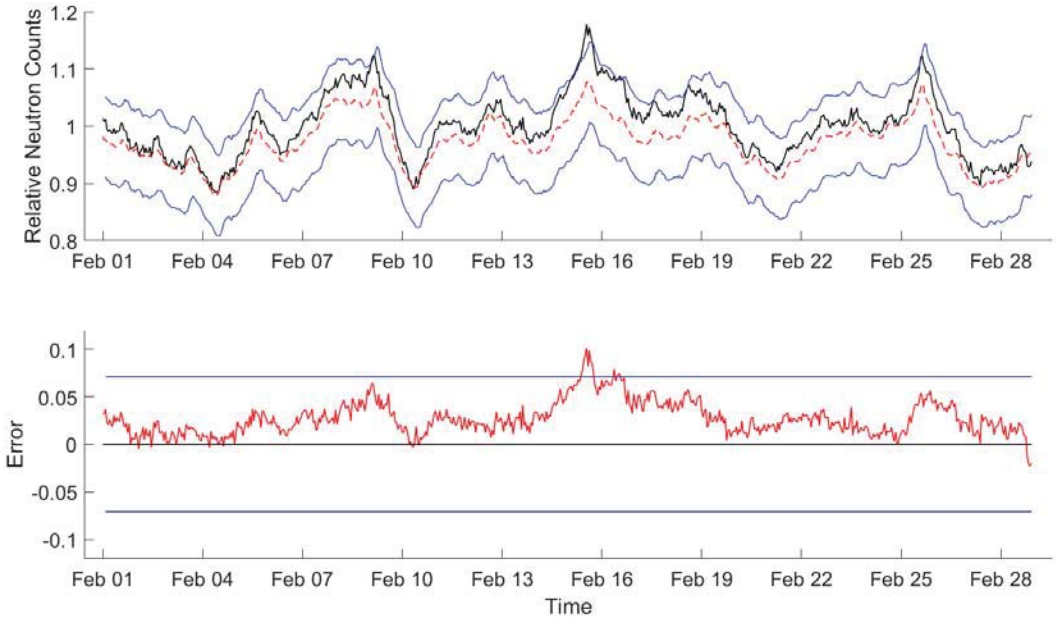


Figure 8.7: This is the performace of the mobile AR(2) model. The solid black line is the measured counts, the dashed line is the model estimate. The top and bottom lines are the prediction interval for the model based of the known parameters. The bottom plot is the residual errors of the top plot.

Initially the OLS model visually appears to out preform the AR(2) model, but the extra parameters contained in the AR(2) model, that the OLS does not contain, is where the AR(2) performance improves. Mathematically, the AR(2) model does better because the standard error of the AR(2) model is $\pm 7.21\%$ and the standard error for the OLS model was $\pm 7.26\%$. There are two extra lines, the uppermost and bottom lines, in figure 8.7 are the prediction interval for the AR(2) model that does not contain any information about its previous errors. The lines are calculated using equation 5.52 and represent the 95% prediction interval of the model. The values of the residual errors is larger, but all of the residuals are with in the statistical prediction errors of the model. It is expected that 5% of the residual errors would lie outside these lines. So the few hours around 15FEB17, that are outside the prediction interval, are normal and can be expected with this model.

The AR(2) model still greatly increases the ability to predict the neutron background from the naive method. A basic alarm algorithm is to alarm when the measured counts exceeds, two sigma, or this 95% prediction interval. For the naive method, due to the large fluctuations, it is very hard to accurately set that alarm threshold. However, with the AR(2) system there is a well defined 95% prediction interval that is much smaller than the naive method. This allows for the system to become more sensitive and able to identify smaller sources.

The equation used to predict using the AR(2) mobile uses the same inputs as the OLS model, but the difference between this and the OLS is how the coefficients are calculated. The AR(2) model knows that the residual errors are related and accounts for that in its standard errors and by extension in the prediction intervals. The OLS does not know about the residual errors relationship and therefore has a falsely low standard error and prediction interval.

The results in figure 8.7 are from a model that does not use any information about previous residual errors to estimate the current prediction. The positive side of this system is that it would not be contaminated by the presence of a source. This model, theoretically (see section 9.2), could be moved into any location and just by the input variable predict the number of neutrons expected. But if the system relied on the residual errors and it was moved into the presence of a source it could mistake the source as just an residual error and never identify the anomaly. Because this method, called the "mobile" model, cannot be contaminated by a neutron source it should be used for a mobile detector.

8.2.3 AR(2) Stationary Results

The second way a AR(2) model can be used is by incorporating the previous errors into the current estimation. This type of prediction, a "stationary" detector uses equation 5.55 to calculate the prediction value and the 95% prediction interval. This type of prediction, shown in figure 8.8, uses the same coefficients as the mobile model, the difference is that it includes the lag coefficients in the calculations.

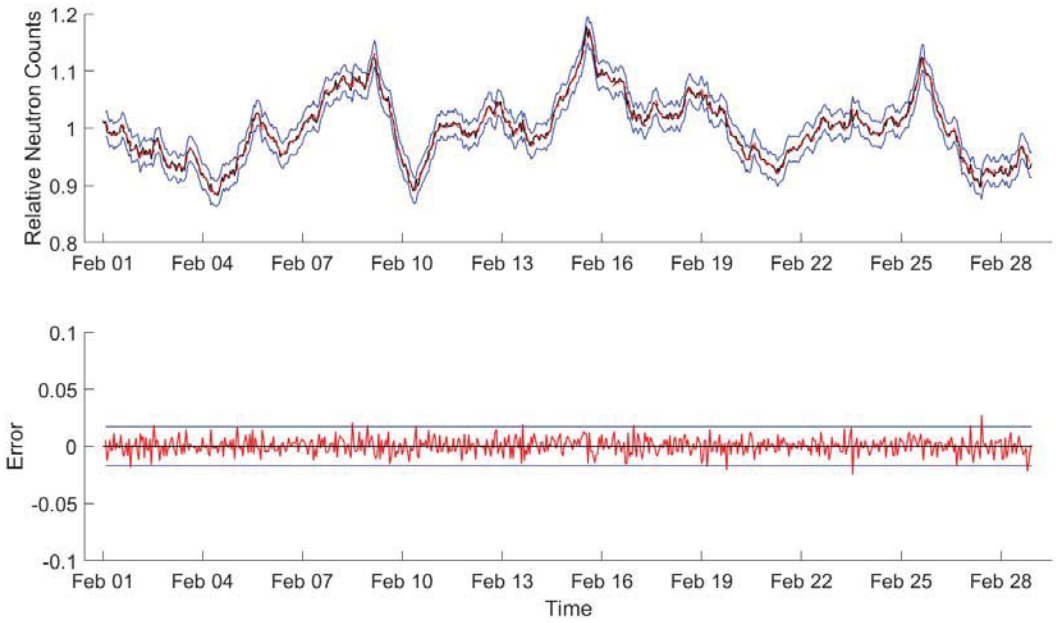


Figure 8.8: The results of the AR(2) stationary model for the out-of-sample prediction over the month of February, 2017. The top plot has a solid line for the measured counts, dashed for the predicted counts, and two prediction interval line. The bottom is the residual errors with the confidence interval.

This figure includes the same four lines plotted in figure 8.7 the measured counts, predicted counts, and the 95% prediction interval. It is difficult to see the separation between those lines in the first plot, but that make sense when looking at the residual errors plot below it. To better understand the residual errors of the AR(2) stationary system, figure 8.9 is a zoomed in plot of the residual errors.

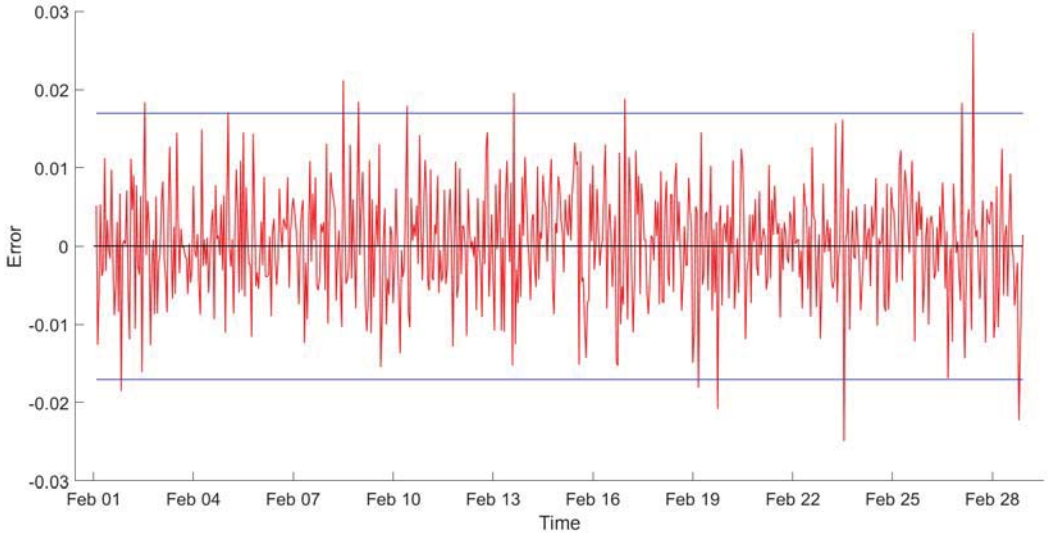


Figure 8.9: This is the residual errors from the stationary AR(2) model shown in figure 8.8. The upper and lower prediction intervals are the two lines just

The prediction interval dropped from $\pm 7.21\%$ in the mobile model to just under $\pm 1.54\%$ in the stationary model. The stationary model is extremely sensitive to neutron sources now. It could be possible for a source to be lost in the noise in the mobile detection system because of the larger prediction interval. But the prediction interval of the stationary model has become so small that it would be almost impossible for a neutron source not to emit enough neutrons to be detected and alarm a detection algorithm.

Chapter 9

Conclusions and Recommendations for Future Work

9.1 Conclusion

There are several conclusions from this project that could have huge implications for future work in this field of research. The three conclusions most clearly seen is the dominate role of pressure in determining the background, the extra factors studied that do not have an impact on the neutron background, and finding the correlation of the residual errors. Pressure, the model input is inverse pressure but it will just be referred to as 'pressure' for this section, has been studied before, but it was not known that it was the most dominate factor of weather variables. The previous models made, section 2.2, relied on altitude and latitude as the inputs, but pressure is how altitude is measure by most systems. Now, rather than needing an altitude input for a global model, pressure could be substituted in and potentially the only other variable that would need to be added to this model for it to work around the world is the latitude.

We looked into using absolute humidity rather than relative humidity that the weather stations read, amount of rainfall over an hour, and the cloud cover. As was discussed in section 7.2 none of these variables added significant improvements to the overall model. An additional advantage of not including rainfall and cloud cover in model is that it is still a self contained unit. It is very easy to carry a weather station that reads the current variables needed for the model. But to include cloud cover the system would also need access to a weather RADAR station for that information and past rain history based of the sensors position. This would make the prediction method much harder and impractical for field applications. Due to our investigation, the effect of these variables has been seen and future work can focus on other inputs.

The largest conclusion from this project is identifying the correct modeling process and applying it. Note that almost all of the residual errors from the mobile AR(2) model, as shown in figure 8.7 are above zero. Normally in a model the error term is expected to be independent and have a white noise variance with a mean of zero as seen in figure 8.8. However, this residual error plot clearly violates that assumption and proves that the errors are correlated. That is why the AR(2) lag coefficients are the correct way to model the neutron background. Based on these results alone we know that it is an AR(n) model.

The following table is a summary of the four different model types and their performances from the worst case naive model to the most advanced AR(2) stationary model.

Model name	Equation	Relative standard errors
Naive	$y_i = y_{avg} + \varepsilon_i$	$\pm 13.25\%$
OLS	$y_i = X_i\beta + \varepsilon_i$	$\pm 7.26\%$
AR(2) mobile	$y_i = X_i\hat{\beta} \pm 2\sqrt{X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma_\varepsilon^2}}$	$\pm 7.21\%$
AR(2) stationary	$y_i = X_i\hat{\beta} + \hat{\phi_1}r_{i-1} + \hat{\phi_2}r_{i-2} \pm 2\sqrt{X_i\hat{Var}(\hat{\beta})X_i^T + \hat{\sigma_\omega^2}}$	$\pm 1.54\%$

Table 9.1: Summary of all four models used in this project with their equations and prediction errors in relative percentage.

The results of the AR(2) stationary model is a reduction of the unknown variance of the neutron background from $\pm 13.25\%$ down to $\pm 1.54\%$ when compared to the naive model.

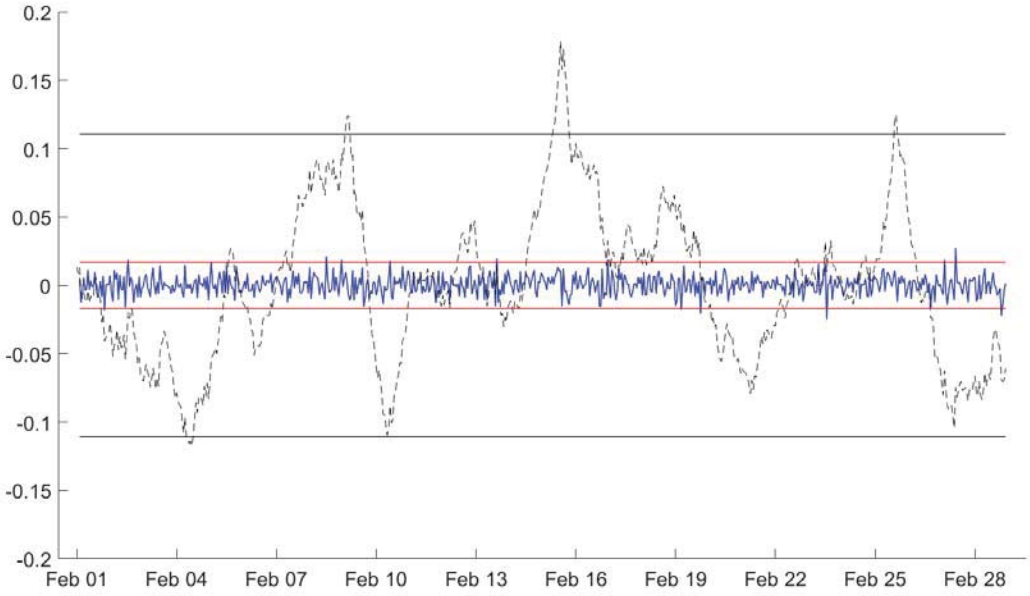


Figure 9.1: Results of the AR(2) stationary model compared to the naive model with 95% prediction intervals. The solid blue line is the AR(2) residual errors between the prediction intervals from the AR(2) model. The dashed line is the naive residual errors with its prediction intervals.

The AR(2) model does over an order of magnitude better than any previous model to predict the neutron background. Not only that, but this model only uses pressure, temperature, and dew point to create its prediction. All three of those variables are easy to read real time next to the detector and does not require expensive equipment or network access.

Not only does this project find that it is possible to create a model that does over an order of magnitude better than a naive model, but also created a mobile system that does 2.5 times better than the naive model. Not only does it do 2.5 times better, but it also provides a detection algorithm with ability to create accurate prediction intervals and create lower alarm thresholds. The lower alarm thresholds lead to more sensitive detectors and much more useful applications.

9.2 Recommendations for Future Work

The AR(2) model does much better than any previously seen model, but this project was just the start to field of research that has many obvious potential expansions such as location, inputs, detector size and time

interval, and source identification. This project was limited to a single location with a verification location close by. However, it is well known that geographic location has a large role on the neutron background. There needs to be another project that expands to several locations and determines how this model would to be adjusted for a new area. The largest factor that previous models accounted for was the altitude of the system because it measured the amount of atmosphere above their location. But pressure is also a measure of the density of the atmosphere about a location, so it may be found that altitude is not needed as long as pressure is a factor.

We looked at humidity, rain fall, and cloud cover, and several other variables were discussed but never utilized in this project. The water table in the ground could have an impact after large rain storms, because the hydrogen in the water is close to the surface of the ground, and reduce the counts. There are some labs that measure the strength of the sun's magnetic field on a daily basis and this factor could better adjust for the amount of incident particles on the upper atmosphere. Also there could be an argument made to take pressure readings at different altitudes to better account for the air density above the detector.

The detector size could play a large role in this. This model was built for the LNS, a very large detector, that is able to capture more variation than a smaller detector. This model will need to be normalized and adjusted for the sensitivity of a smaller detector. Once that is done, the model will need to be tested to ensure that it holds for any detector. Along with changing the detector, the time interval for detection can be changed. The current model uses an hour time interval, but most detection systems need to act on a much smaller time scale to detect live time events. But as the time interval is decreased the order of the AR model will need to be determined. And with smaller amounts of counts the variance of the natural background decreases and becomes harder to identify.

The final obvious test for this model is to introduce a neutron source into the detector's environment and see how the model preforms. The size and strength of the source can be varied to determine the limits of the model. For the stationary model, it needs to be tested how strong the source has to be to create the alarm and not be assumed to be noise of the natural background.

More generally, the this same method could also be applied to gamma radiation counts. A gamma detector is more sensitive to physical surroundings and has many things that can influence the gamma ray count rate. For example, when people are in the near vicinity of the detector, the count rate may change based on either shielding effect or a source effect (i.e. medical treatments). With so many things influencing the gamma count rate it would be harder to find times when there are no outside factors present.

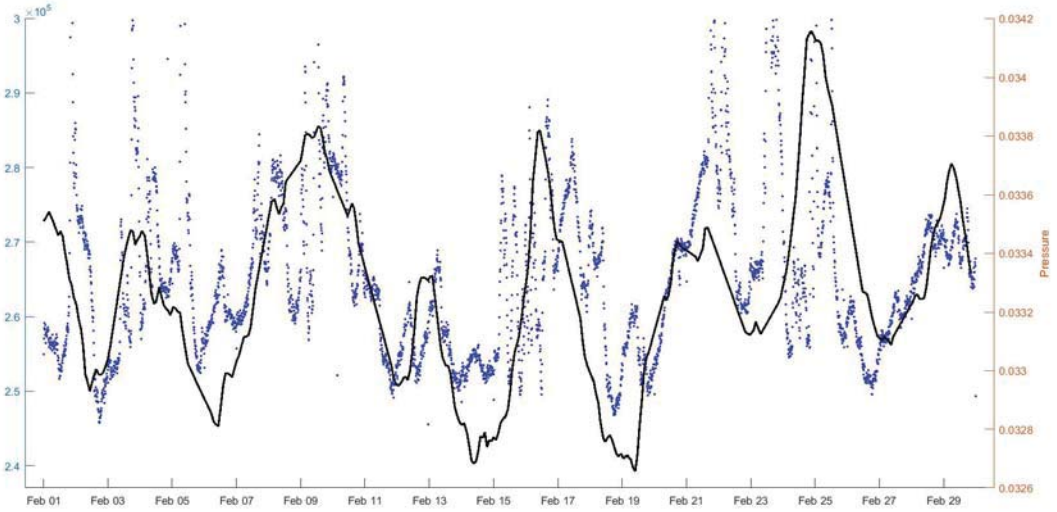


Figure 9.2: The solid black line is inverse pressure and the dotted line is gamma counts during February 2016 in the Washington, D.C. area.

While the overall graph looks does not have large areas that appear correlated, there are small areas

where the data is effected by sources being introduced into its environment. One such area of correlated data is the weekends when there is little to no traffic moving past the detectors. Because the sensors are located around an office type of building the foot and vehicle traffic is large during the week, but low on the weekends leaving correlated data. February 26th through the 29th was a holiday weekend in 2016 and is represented on the right side of the graph. There is a definite relationship there as the inverse pressure rises so does the gamma ray counts. The timing is delayed, but some of the other weekends show stronger correlations. It might be possible that the same meteorological variables that effect the neutron background effect the gamma background as well.

Chapter 10

References

[1] Kouzes, et. all., Cosmic-ray-induced ship-effect neutron measurements and implications for cargo scanning at borders, Nucearl Instruments and Methods in Physics Research, October 3,2007.

[2] James F. Ziegler, "SER – History, Trends and Challenges: A guide for Designing with Memory ICs," Cypress, 2004

[3] E. Normand and T.J. Baker, Altitude and Latitude Variations in Avionics SEU and Atmospheric Neutron Flux, IEEE Transactions on Nuclear Science, Vol. 40, 1484-1487

[4] National Nuclear Data Center, http://www.nndc.bnl.gov/. Web, JAN 13, 2016

[5] James F. Ziegler, Terrestrial Cosmic Ray Intensities, IBM Journal of Research and Development, Vol. 40, No. 1, January 1996.

[6] Ziegler, James F. Terrestrial Cosmic Ray Intensities, IBM Journal of Research and Development, Vol. 42, No. 1, January 1998.

[7] J.H. Adams Jr., R. Silberberg, and C.H. Tsao, Cosmic Ray Effects on Microelectronics: Part I, The Near-Earth Particle Environment, NRL Memorandum Report 4885, August 25, 1981.

[8] J.H. Adams Jr., R. Silberberg, and C.H. Tsao, Cosmic Ray Effects on Microelectronics: Part II, The Geomagnetic Cuttoff Effects, NRL Memorandum Report 5099, May 26, 1983.

[9] J.H. Adams Jr., R. Silberberg, and C.H. Tsao, Cosmic Ray Effects on Microelectronics: Part III, Propagation of Cosmic Rays in the Atmosphere, NRL Memorandum Report 5402, August 9, 1984.

[10] J.H. Adams Jr., R. Silberberg, and C.H. Tsao, Cosmic Ray Effects on Microelectronics: Part IV,, NRL Memorandum Report 5901, December 31, 1984.

[11] Eugene Normand, Single Event Effects in Avionics, Boeing Defense & Space Group Document, Seattle, WA, 1995.

[12] S. Lindgren. On the pressure dependence of the cosmic ray intensity recorded by the standard neutron monitor, University of Uppsala, Physics, August 2, 1961.

[13] R. Rosolem, et all., The Effect of Atmospheric Water Vapor on Neutron Count in the Cosmic-Ray Soil Moisture Observing System, J. Hydrometeor Journal of Hydrometeorology Vol. 14, No. 5, 1659-1671. October, 2013.

[14] K. O'Brien, et. all. Cosmic Ray Induced Neutron Background Sources and Fluxes for Geometries of Air Over Water, Ground, Iron, and Aluminum, Geophysical Research, Vol 83, No A1. January 1, 1978.

[15] Glenn F. Knoll. "Radiation Detection and Measurement" New York: Wiley, 1979.

[16] Richard T. Kouzes, SPAWAR Neutron Detector Information, U.S. Department of Energy PNWD-4429. July 2014.

[17] Benjamin C. Etringer, A Modeling and Data Analysis of Laser Beam Propagation in the Maritime Domain, USNA Trident Scholar Project report; no 433 (2015).

[18] Olga Korotkova, Svetlana Avramov-Zamurovis, Reza Malek-Madani, and Charles Nelson, Probability density function of the intensity of a laser beam propagation in the Maritime environment, OSA, Vol. 19, No. 21. October 3, 2011.

[19] Tatjana Jevremovic. Nuclear Principles in Engineering. Springer, 2005. Page 7.

[20] Robert H. Shumway, David S. Stoffer. *Time Series Analysis and Applications Using the R statistical package*. New York: Springer Science Business Mediar, 2010. Print. Page 96.

[21] www.nndc.bnl.gov; ENDF/B-VII.1 data.

Chapter 11

Appendix

MATLAB Code

Importing Neutron Data

```
% Script for importing data from the following text file:
%
%    E:\LNS Data\0916
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2016/09/21 21:17:11

%% Initialize variables.
Month = 2;
% filename = 'E:\LNS Data\1216_Full.txt';
filename = 'E:\LNS Data\0217_Full.txt';
% 'E:\LNS Data\0916_Full.txt';
delimiter = ',';

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%*s%s%s%s%s%s%[^\n\r]';

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

dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,
'ReturnOnError', false);

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

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
```

```
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[3,4,5,6]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*
        [eEdD]{0,1}[-+]*\d*[i]{0,1})|
        ([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(numbers, thousandsRegExp, 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

dateFormats = {'yyyy/MM/dd', 'HH:mm:ss'};
dateFormatIndex = 1;
blankDates = cell(1,size(raw,2));
anyBlankDates = false(size(raw,1),1);
invalidDates = cell(1,size(raw,2));
anyInvalidDates = false(size(raw,1),1);
for col=[1,2]% Convert the contents of columns with dates to MATLAB datetimes using
date format string.
    try
        dates{col} = datetime(dataArray{col}, 'Format', dateFormats{col==[1,2]},
        'InputFormat',
        dateFormats{col==[1,2]}); %#ok<SAGROW>
    catch
        try
            % Handle dates surrounded by quotes
            dataArray{col} = cellfun(@(x) x(2:end-1), dataArray{col},
            'UniformOutput', false);
            dates{col} = datetime(dataArray{col}, 'Format', dateFormats{col==[1,2]},
```

```
            'InputFormat',dateFormats{col==[1,2]}); %%#ok<SAGROW>
        catch
            dates{col} = repmat(datetime([NaN NaN NaN]), size(dataArray{col}));
        end
    end

    dateFormatIndex = dateFormatIndex + 1;
    blankDates{col} = cellfun(@isempty, dataArray{col});
    anyBlankDates = blankDates{col} | anyBlankDates;
    invalidDates{col} = isnan(dates{col}.Hour) - blankDates{col};
    anyInvalidDates = invalidDates{col} | anyInvalidDates;
end
dates = dates(:,[1,2]);
blankDates = blankDates(:,[1,2]);
invalidDates = invalidDates(:,[1,2]);

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [3,4,5,6]);

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); % Find
non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%%
date = dates{:, 1};
time = dates{:, 2};
date.Format = 'dd.MM.uuuu HH:mm';
time.Format = 'dd.MM.uuuu HH:mm';
dd = date + timeofday(time);
%% Allocate imported array to column variable names
Data{Month}.LNS{1}.Date = dates{:, 1};
Data{Month}.LNS{1}.Time = dates{:, 2};
Data{Month}.LNS{1}.DT = dd;
Data{Month}.LNS{1}.Tube1 = cell2mat(rawNumericColumns(:, 1));
Data{Month}.LNS{1}.Tube2 = cell2mat(rawNumericColumns(:, 2));
Data{Month}.LNS{1}.Tube3 = cell2mat(rawNumericColumns(:, 3));
Data{Month}.LNS{1}.Tube4 = cell2mat(rawNumericColumns(:, 4));

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans raw col numericData
rawData row
regexstr result numbers invalidThousandsSeparator thousandsRegExp me dateFormats
dateFormatIndex
dates blankDates anyBlankDates invalidDates anyInvalidDates rawNumericColumns R dd
date time Month;
```

Importing Weather Station 1 Data

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    E:\WeatherOSPRO\Import_31OCT.csv
```

```
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2016/10/31 22:29:21

%% Initialize variables.
filename = 'E:\WeatherOSPRO\Import_31OCT.csv';
delimiter = ',';
startRow = 5;

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%*s%*s%s%s%s%s%s%[^\n\r]';

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

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'HeaderLines'
,startRow-1, 'ReturnOnError', false);

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

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[2,3,4,5]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}
        [-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))
        (?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
```

```
                if isempty(regexp(numbers, thousandsRegExp, 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

% Convert the contents of columns with dates to MATLAB datetimes using date
% format string.
try
    dates{1} = datetime(dataArray{1}, 'Format', 'MM/dd/yyyy HH:mm', 'InputFormat',
    'MM/dd/yyyy HH:mm');
catch
    try
        % Handle dates surrounded by quotes
        dataArray{1} = cellfun(@(x) x(2:end-1), dataArray{1}, 'UniformOutput',
        false);
        dates{1} = datetime(dataArray{1}, 'Format', 'MM/dd/yyyy HH:mm',
        'InputFormat', 'MM/dd/yyyy HH:mm');
    catch
        dates{1} = repmat(datetime([NaN NaN NaN]), size(dataArray{1}));
    end
end

anyBlankDates = cellfun(@isempty, dataArray{1});
anyInvalidDates = isnan(dates{1}.Hour) - anyBlankDates;
dates = dates(:,1);

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [2,3,4,5]);

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns);
% Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
Data{Month}.Weather{1}.DT = dates{:, 1};
Data{Month}.Weather{1}.Pressure = cell2mat(rawNumericColumns(:, 1));
Data{Month}.Weather{1}.Temp = cell2mat(rawNumericColumns(:, 2));
Data{Month}.Weather{1}.Humidity = cell2mat(rawNumericColumns(:, 3));
Data{Month}.Weather{1}.weatherVIC5 = cell2mat(rawNumericColumns(:, 4));

% For code requiring serial dates (datenum) instead of datetime, uncomment
% the following line(s) below to return the imported dates as datenum(s).
```

```
% DT=datenum(DT);


%% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans raw col
numericData rawData row regexstr result numbers invalidThousandsSeparator
thousandsRegExp me dates blankDates anyBlankDates invalidDates anyInvalidDates
rawNumericColumns R;
```

Importing Weather Station 2 Data

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    E:\Weather_Orion\September.csv
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2016/10/13 22:00:45

%% Initialize variables.
%filename = 'E:\Weather_Orion\September.csv';
% filename = 'E:\Weather_Orion\OCT.csv';
% filename = 'E:\Weather_Orion\NOV16.csv';
% filename = 'E:\Weather_Orion\DEC16.csv';
% filename = 'E:\Weather_Orion\JAN17.csv';
filename = 'E:\Weather_Orion\FEB17.csv';
delimiter = ',';
startRow = 2;
Month = 2;
Year = 17;
%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%
*s%*s%*s%s%s%*s%*s%*s%*s%*s%*s%*s%
s%*s%*s%*s%*s%*s%*s%*s%s%s%*s%s%s%s%s%s%s%s%s%s%[^\n\r]';

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

%% Read columns of data according to format string.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,
'HeaderLines' ,startRow-1,
'ReturnOnError', false);

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

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
```

```
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[3,4,5,6,7,8,9,10,12,13,14,17]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}
        [-+]*\d*[i]{0,1})|
        ([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(numbers, thousandsRegExp, 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

dateFormats = {'MM/dd/yyyy', 'HH:mm', 'MM/dd/yyyy', 'HH:mm'};
dateFormatIndex = 1;
blankDates = cell(1,size(raw,2));
anyBlankDates = false(size(raw,1),1);
invalidDates = cell(1,size(raw,2));
anyInvalidDates = false(size(raw,1),1);
for col=[1,2,15,16]% Convert the contents of columns with
dates to MATLAB datetimes using date
                    %format string.
    try
        dates{col} = datetime(dataArray{col}, 'Format',
        dateFormats{col==[1,2,15,16]},
        'InputFormat', dateFormats{col==[1,2,15,16]}); %#ok<SAGROW>
    catch
        try
```

```
            % Handle dates surrounded by quotes
            dataArray{col} = cellfun(@(x) x(2:end-1), dataArray{col},
            'UniformOutput', false);
            dates{col} = datetime(dataArray{col}, 'Format',
            dateFormats{col==[1,2,15,16]},
            'InputFormat',dateFormats{col==[1,2,15,16]}); %%#ok<SAGROW>
        catch
            dates{col} = repmat(datetime([NaN NaN NaN]), size(dataArray{col}));
        end
    end

    dateFormatIndex = dateFormatIndex + 1;
    blankDates{col} = cellfun(@isempty, dataArray{col});
    anyBlankDates = blankDates{col} | anyBlankDates;
    invalidDates{col} = isnan(dates{col}.Hour) - blankDates{col};
    anyInvalidDates = invalidDates{col} | anyInvalidDates;
end
dates = dates(:,[1,2,15,16]);
blankDates = blankDates(:,[1,2,15,16]);
invalidDates = invalidDates(:,[1,2,15,16]);

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [3,4,5,6,7,8,9,10,12,13,14,17]);
rawCellColumns = raw(:, [11,18]);


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns);
% Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
Data{Month}.Orion{1}.Date = dates{:, 1};
Data{Month}.Orion{1}.Time = dates{:, 2};
Data{Month}.Orion{1}.Temp_in = cell2mat(rawNumericColumns(:, 1));
Data{Month}.Orion{1}.Temp_out = cell2mat(rawNumericColumns(:, 2));
Data{Month}.Orion{1}.Humidity_in = cell2mat(rawNumericColumns(:, 3));
Data{Month}.Orion{1}.Humidity_out = cell2mat(rawNumericColumns(:, 4));
Data{Month}.Orion{1}.DP_out = cell2mat(rawNumericColumns(:, 5));
Data{Month}.Orion{1}.Wind_direction = cell2mat(rawNumericColumns(:, 6));
Data{Month}.Orion{1}.GustSpeedKnots = cell2mat(rawNumericColumns(:, 7));
Data{Month}.Orion{1}.AVGWind = cell2mat(rawNumericColumns(:, 8));
Data{Month}.Orion{1}.WindchillF = rawCellColumns(:, 1);
Data{Month}.Orion{1}.RainRate = cell2mat(rawNumericColumns(:, 9));
Data{Month}.Orion{1}.HourlyRainFall = cell2mat(rawNumericColumns(:, 10));
Data{Month}.Orion{1}.TotalRain = cell2mat(rawNumericColumns(:, 11));
Data{Month}.Orion{1}.AccumStartDate = dates{:, 3};
Data{Month}.Orion{1}.AccumStartTime = dates{:, 4};
Data{Month}.Orion{1}.Pressure = cell2mat(rawNumericColumns(:, 12));
Data{Month}.Orion{1}.Weather = rawCellColumns(:, 2);
date = Data{1,Month}.Orion{1}.Date;
time = Data{1,Month}.Orion{1}.Time;
date.Format = 'dd.MM.uuuu HH:mm';
time.Format = 'dd.MM.uuuu HH:mm';
```

```
Data{Month}.Orion{1}.DT = date + timeofday(time);

%% Clear temporary variables
clearvars Month time i j date filename delimiter startRow formatSpec
fileID dataArray ans raw col
numericData rawData row regexstr result numbers invalidThousandsSeparator
thousandsRegExp me
dateFormats dateFormatIndex dates blankDates anyBlankDates invalidDates
anyInvalidDates
rawNumericColumns rawCellColumns R;
```

NOAA Data code

```
%%

a = {'CLEAR NIGHT'};            %Bad, replace with "g"
b = {'PARTLY CLOUDY AT NIGHT'}; %Bad, replace with "d"
c = {'SUNNY'};                  %Bad, replace with "g"
d = {'PARTLY CLOUDY'};          %Good
e = {'RAINY'};                  %Good
f = {'CLOUDY'};                 %Good
g = {'CLEAR'};                   %Good
startindex = 0;
for Month=9:10
    for i=1:length(Data{1,Month}.Orion{1}.Weather)

        if (cellfun(@strcmp, Data{1,Month}.Orion{1}.Weather(i), b))
            %partly cloudy at night
            MData{1}.Weather(i + startindex) = d;
        elseif (cellfun(@strcmp, Data{1,Month}.Orion{1}.Weather(i), a))
            %clear night
            MData{1}.Weather(i + startindex) = g;
        elseif (cellfun(@strcmp, Data{1,Month}.Orion{1}.Weather(i), c))
            %sunny
            MData{1}.Weather(i + startindex) = g;
        else
            MData{1}.Weather(i + startindex) = Data{1,Month}.Orion{1}.Weather(i);
        end
    end
    startindex = startindex + length(Data{1,Month}.Orion{1}.Weather);
end

clear a b c d e f g i startindex Month

% int = cell(1, length(a));
% int(index) = a(index);
%% Find anytime the rain fall was over 0.10 in per hour
Rain2 = Data{1}.NOAA{1}.Precip >= 0.2;
%%
figure,
plot(Data{1}.NOAA{1}.DT,Rain2)

%% Cloud stuff
```

```
Data{1}.NOAA{1}.Sky = char(Data{1}.NOAA{1}.Clouds);

%%
% leng = length(Data{1}.NOAA{1}.Sky);
j=1;
Data{1}.NOAA{1}.SkyCond(1:leng,2) = 0;
for i=1:leng
    ind = max(find(isspace(Data{1}.NOAA{1}.Sky(i,:))==0));
    if (Data{1}.NOAA{1}.Sky(i,ind-2)==' ')
        Data{1}.NOAA{1}.SkyCond(i) = str2num(Data{1}.NOAA{1}.Sky(i,ind-4:ind-3));
    elseif (Data{1}.NOAA{1}.Sky(i,ind-3)==' ')
        Data{1}.NOAA{1}.SkyCond(i,1) = str2num(Data{1}.NOAA{1}.Sky(i,ind-5:ind-4));
    elseif str2num(Data{1}.NOAA{1}.Sky(i,5:6))==0
        Data{1}.NOAA{1}.SkyCond(i,1) = 0;
    else
        fprintf('Did not work on row %d that is %s\n',i, Data{1}.NOAA{1}.Sky(i,:))
        messed(j,1) = i;
        j = j+1;

    end
end
%%
for i=1:length(messed)
    if (isempty(str2num(Data{1}.NOAA{1}.Sky(messed(i),5:6)))==0)
        Data{1}.NOAA{1}.SkyCond(messed(i),1) =
        str2num(Data{1}.NOAA{1}.Sky(messed(i),5:6));
    end
end
```

Absolute Humidity code

```
%% Define constants
a1 = -7.85951783;   a2 = 1.84408259;    a3 = -11.7866497;
a4 = 22.6807411;    a5 = -15.9618719;   a6 = 1.80122502;
Tc = 647.096;   Pc = 22.064;    Rw = 461.5;

%%
% Temp = 90;
Temp = MData{1}.Hour_Data(:,7);
T = convtemp(Temp,'F','K');
C = 1-T/Tc;
% RH = 90;
RH = MData{1}.Hour_Data(:,8);
%%
for i=1:1176
    Pws(i) = Pc*exp((Tc/T(i))*(a1*C(i) + a2*C(i)^1.5 + a3*C(i)^3 + a4*C(i)^3.5 +
    a5*C(i)^4 + a6*C(i)^7.5));
    Pw(i) = (RH(i)*Pws(i))/100;
    AH(i) = Pw(i)/(Rw*T(i))*1000000;
    MData{1}.Hour_Data(i,19) = AH(i);
%     fprintf('Absolute Humidity is: %f\n kilogram per cubic meter\n',AH(i))
end
```

```
%%
clear Temp T C RH a1 a2 a3 a4 a5 a6 Tc Pc Rw Pws Pw AH
```

Creating a Time Vector

```
totmin = 44640; %for 2 months 87840; %add 44640 for 31 days, and 43200 for 30 days
% MData{1}.DT(1:totmin,1) = Data{9}.LNS{1}.DT(1); %= 'dd.MM.uuuu HH:mm';
i = 1;
oldtime = 0;
tic
for year = 2016:2016
    for month = 11:12
        if (mod(month,2) == 0)
            endday = 31;
        else
            endday = 30;
        end
        for day = 1:endday
            for hour=0:23
                for min=0:59
                    MData{1}.DT(i,1).Year = year;
                    MData{1}.DT.Month(i) = month;
                    MData{1}.DT.Day(i) = day;
                    MData{1}.DT.Hour(i) = hour;
                    MData{1}.DT.Minute(i) = min;
                    %             MData{1}.DT.Second(i) = sec;

                    i = i+1;
                end
            end
            fprintf('Day %d done in %f seconds.\n Total Time is:
            %f\n',(day),(toc-oldtime),toc)
            oldtime = toc;
        end
    end
end
clear year endday month i hour min day
```

Matching Times for MData

```
% Data{Month}.LNS{1}.DT.Second(:) = 0;
% Data{Month}.Orion{1}.DT.Second(:) = 0;

howLong = Data{Month}.LNS{1}.DT(1:40000)==Data{Month}.Orion{1}.DT(1:40000);
%Compaire all the timestamps

index = find(~howLong,1)    %Find the first nonzero index

fprintf('LNS time is:')
Data{Month}.LNS{1}.DT(index)
fprintf('Orion time is:')
Data{Month}.Orion{1}.DT(index)
```

```
%% Add the new times in
Month = 10;
startt = length(MData{1}.DT) + 1
IndEnd = startt + length(Data{Month}.LNS{1}.DT)
%%
j = 1;
for i=startt:IndEnd
MData{1}.DT(i,1) = Data{Month}.LNS{1}.DT(j);
j = j + 1;
end



% Weather data matching
e = 0;
endd = 0;
Month = 2;
% startt =  1;      %For the start of September
% startt = 43201; %For the start of OCT
% startt = 87841; %For NOV
% startt = 131041; %For DEC
% startt = 175681; %For JAN17
startt = 220321; %For FEB17

IndEnd = startt + length(Data{Month}.LNS{1}.DT);
Data{Month}.Orion{1}.DT(:).Second = 0;
%* You will probably get an error about "exceeds matrix dimensions" don't
%worry about that.  Just verify manually that it ran through the end of the
%month and you will be fine.  (Its because I did not tell it to check that
%adding in the error terms was not exceeding matrix)
for i=startt:IndEnd
    if(Data{Month}.Orion{1}.DT(i-startt+1) == MData{1}.DT(i+e))
        MData{1}.Data(i+e,6) = Data{Month}.Orion{1}.Temp_out(i-startt+1);
        MData{1}.Data(i+e,7) = Data{Month}.Orion{1}.Humidity_out(i-startt+1);
        MData{1}.Data(i+e,8) = Data{Month}.Orion{1}.DP_out(i-startt+1);
        MData{1}.Data(i+e,9) = Data{Month}.Orion{1}.Pressure(i-startt+1);
        MData{1}.Data(i+e,10) = Data{Month}.Orion{1}.RainRate(i-startt+1);
        MData{1}.Data(i+e,11) = Data{Month}.Orion{1}.AVGWind(i-startt+1);
    else
        e=e+1;
        while(endd == 0)
            if(Data{Month}.Orion{1}.DT(i-startt+1) == MData{1}.DT(i+e))
                MData{1}.Data(i+e,6) = Data{Month}.Orion{1}.Temp_out(i-startt+1);
                MData{1}.Data(i+e,7) =
                Data{Month}.Orion{1}.Humidity_out(i-startt+1);
                MData{1}.Data(i+e,8) = Data{Month}.Orion{1}.DP_out(i-startt+1);
                MData{1}.Data(i+e,9) = Data{Month}.Orion{1}.Pressure(i-startt+1);
                MData{1}.Data(i+e,10) = Data{Month}.Orion{1}.RainRate(i-startt+1);
                MData{1}.Data(i+e,11) = Data{Month}.Orion{1}.AVGWind(i-startt+1);
                endd = 1;
            else
                e = e + 1;
                if e>100
                    i
```

```
                    fprintf('e is above 100\n')
                end
            end
        end
        endd = 0;
    end
end
MData{1}.Data(MData{1}.Data == 0) = NaN


%% Neutron data matching
e = 0;
fwd = 0;
endd = 0;
Month = 2;
% startt =  1;      %For the start of September
% startt = 43201; %For the start of OCT
% startt = 87841; %For NOV
% startt = 131041; %For DEC
% startt = 175680; %For JAN17
startt = 220321; %For FEB17

IndEnd = startt + length(Data{Month}.LNS{1}.DT);
Data{Month}.LNS{1}.DT(:).Second = 0;

for i=startt:IndEnd
    %     e = oldE
    if(Data{Month}.LNS{1}.DT(i-startt+1) == MData{1}.DT(i+e-fwd))
        MData{1}.Data(i+e-fwd,1) = Data{Month}.LNS{1}.Avg(i-startt+1);
        MData{1}.Data(i+e-fwd,2) = Data{Month}.LNS{1}.Tube1(i-startt+1);
        MData{1}.Data(i+e-fwd,3) = Data{Month}.LNS{1}.Tube2(i-startt+1);
        MData{1}.Data(i+e-fwd,4) = Data{Month}.LNS{1}.Tube3(i-startt+1);
        MData{1}.Data(i+e-fwd,5) = Data{Month}.LNS{1}.Tube4(i-startt+1);
    else
        if (Data{Month}.LNS{1}.DT(i-startt+1).Day == 30)
            i
        end
        Data{Month}.LNS{1}.DT(i-startt+1)
        e=e+1;

        oldE = e
        while(endd == 0)
            if(Data{Month}.LNS{1}.DT(i-startt+1) == MData{1}.DT(i+e-fwd))
                MData{1}.Data(i+e-fwd,1) = Data{Month}.LNS{1}.Avg(i-startt+1);
                MData{1}.Data(i+e-fwd,2) = Data{Month}.LNS{1}.Tube1(i-startt+1);
                MData{1}.Data(i+e-fwd,3) = Data{Month}.LNS{1}.Tube2(i-startt+1);
                MData{1}.Data(i+e-fwd,4) = Data{Month}.LNS{1}.Tube3(i-startt+1);
                MData{1}.Data(i+e-fwd,5) = Data{Month}.LNS{1}.Tube4(i-startt+1);
                endd = 1;
            else
                e = e + 1;
                if e>100
                    fprintf('e is above 100\n')
                    %                     i
```

```
                    %This will tell me at what index the error
                    %                     happens
                    endd = 1        %Kicks it out of the "while" loop
                    e = oldE - 1;   %
                    fwd = fwd+1;
                    %I'm assuming the sensor has doubbled on a reading.  So I
                    %set the error term back to where it was before the bad
                    %reading.  The "fwd" allows for the index of the sensor to
                    %move forward while the overall time does not change.
                    %This makes it so that the overall spread sheet still gets
                    %a reading every minute.
                end
            end
        end
        endd = 0;
    end
end
MData{1}.Data(MData{1}.Data == 0) = NaN

%% NOAA Data

endd = length(Data{1}.NOAA{1}.DT);
j = 1;
for i=1:endd
    if zer(i)==1
        if(Data{Month}.NOAA{1}.DT(i) == MData{1}.Hour_DataT(j))
            MData{1}.Hour_DataA(j+1,9) = Data{Month}.NOAA{1}.SkyCond(i);
            MData{1}.Hour_DataA(j+1,10) = Data{Month}.NOAA{1}.Precip(i);
        else
            fprintf('Did not match i = %d and j = %d\n',i,j)
        end
        j = j+1;
        if i==3509
            j = j+1;
        end
    end
end
```

Weekly Task Code

```
%% Making the Avg number of counts for the 4 tubes
for Month=2
    for i=1:length(Data{Month}.LNS{1}.Tube4)
        Data{Month}.LNS{1}.Avg(i,1) = sum((Data{Month}.LNS{1}.Tube1(i) +
        Data{Month}.LNS{1}.Tube2(i + Data{Month}.LNS{1}.Tube3(i) +
        Data{Month}.LNS{1}.Tube4(i))/4);
    end
end
%% Making 60 minute intervals of neutron data
for Month=2
    j = 1;
    for i=1:60:length(Data{Month}.LNS{1}.Tube1)
        if i+60>length(Data{Month}.LNS{1}.Tube1)
```

```
            break
        else
            Data{Month}.LNS{1}.Tube1_60(j) = sum(Data{Month}.LNS{1}.Tube1(i:i+60));
            Data{Month}.LNS{1}.Tube2_60(j) = sum(Data{Month}.LNS{1}.Tube2(i:i+60));
            Data{Month}.LNS{1}.Tube3_60(j) = sum(Data{Month}.LNS{1}.Tube3(i:i+60));
            Data{Month}.LNS{1}.Tube4_60(j) = sum(Data{Month}.LNS{1}.Tube4(i:i+60));
            Data{Month}.LNS{1}.Avg_60(j) = sum(Data{Month}.LNS{1}.Avg(i:i+60));
            j = j+1;
        end
    end
end


%% Plot Neutron data
figure, hold on
for Month = 2%9:12
    plot(Data{Month}.LNS{1}.DT(1:60:end-60),Data{Month}.LNS{1}.Tube1_60,'Color',
    '[0.5451 0.27 0.0745]')
    plot(Data{Month}.LNS{1}.DT(1:60:end-60),Data{Month}.LNS{1}.Tube2_60,'Color',
    '[0.5804 0 0.8275]')
    plot(Data{Month}.LNS{1}.DT(1:60:end-60),Data{Month}.LNS{1}.Tube3_60,'Color',
    '[1 0.8431 0]')
    plot(Data{Month}.LNS{1}.DT(1:60:end-60),Data{Month}.LNS{1}.Tube4_60,'k')
end
title('All four neutron tubes in hour intervals')
% gca.YAxis.TickLabelFormat = '%,.1f';
% set(gca,'YTickLabel',num2str(get(gca,'YTick')))
%% Plot Neutron counts and Pressure
figure, hold on
for Month = 9:10
    yyaxis left
    plot(Data{Month}.LNS{1}.DT(1:60:end-60),Data{Month}.LNS{1}.Avg_60,'k')
    ylabel('Neutrons')
    %     ylim([400 1200])
    yyaxis right
    plot(Data{Month}.Orion{1}.DT,1./Data{Month}.Orion{1}.Pressure,'b','LineWidth',2)
    ylabel('Pressure')
    %     ylim([0.03138 0.03459])
    legend('Neutron Counts','Inverse Pressure','location','best')
end
%%
figure, hold on
plot(MData{1}.DT(1:70564),MData{1}.Data(:,6))
plot(Data{10}.Weather{1}.DT,Data{10}.Weather{1}.Temp)

%%
figure, hold on
plot(MData{1}.DT(1:70564),MData{1}.Data(:,7))
plot(Data{10}.Weather{1}.DT,Data{10}.Weather{1}.Humidity)
legend('Weather Station 1','Weather Station 2','location','best')
%%
figure, hold on
plot(MData{1}.DT(1:70564),MData{1}.Data(:,9))
plot(Data{10}.Weather{1}.DT,Data{10}.Weather{1}.Pressure)
```

```
legend('Weather Station 1','Weather Station 2','location','best')
```

Final Modeling and Plots

```
[formatcom=\sffamily]


%% OLS
Model = fgls(MData{1}.Hour_DataA(861:3076,[5 7 8]),MData{1}.Hour_DataA(861:3076,4),
'innovMdl','AR','arLags',2,'display','final','numIter',20);
OLS_b = [-66064.6210, 32.3930, -27.1703, 2302600];
for i=163:4343
    Y_ols(i) = OLS_b*MData{1}.Hour_DataA(i,[1 5 7 8])';
end
%% Plot OLS
% plotstart = 163;    plotend = 4343;%length(MData{1}.Hour_DataA);  %all data
% plotstart = 721;   plotend = 1464;    %OCT
% plotstart = 1464;   plotend = 2184;    %NOV
% plotstart = 3076;   plotend = 3672;    %JAN
plotstart = 3673;   plotend = 4343;    %FEB

figure
subplot(2,1,1)
hold on
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,4)./Yavg)
plot(MData{1}.Hour_DataT(plotstart:plotend),Y_ols(plotstart:plotend)'./Yavg,'k')
% plot(MData{1}.Hour_DataT(plotstart:plotend),Y2(plotstart:plotend)','r')
ylabel('Neutron Counts')
subplot(2,1,2)
hold on
plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA(plotstart:plotend,
4)-Y_ols(plotstart:plotend)')./Yavg,'k')
% plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Y2(plotstart:plotend)'),'r')
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,1)-1)
ylabel('Error');    xlabel('Time')


%% Plot y-ybar and model
figure, hold on
% plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Y_ols(plotstart:plotend)'),'k')
% plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Y2(plotstart:plotend)'),'r','LineWidth',1,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),(MData{1}.Hour_DataA
(plotstart+2:plotend,4)-Y2_new(plotstart:plotend-2)'),'k','LineWidth',1,
'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Yavg),'r--')
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
```

```
(plotstart:plotend,1)-1,'Color','[0.5020 0 0]')



%% AR2 2/21/17
%This code makes the final model I'm going to use
Model1 = regARIMA(2,0 ,0)
% [EstMdl,EstParamCov,logL,info] =
estimate(Model1,MData{1}.Hour_DataA(160:end,4),'X',MData{1}.Hour_DataA
(160:end,[2:3 5:10]))
% [EstMdl,EstParamCov,logL,info] =
estimate(Model1,MData{1}.Hour_DataA(861:3076,4),'X',MData{1}.Hour_DataA
(861:3076,5:10))
[EstMdl,EstParamCov,logL,info] =
estimate(Model1,MData{1}.Hour_DataA(861:3672,4),'X',MData{1}.Hour_DataA
(861:3672,[5 7 8]))
Var = EstMdl.Variance;
Beta2 = EstMdl.Beta;
phi = cell2mat(EstMdl.AR);
% VarBeta = EstParamCov([1 4:9],[1 4:9]);
VarBeta = EstParamCov([1 4 5 6],[1 4 5 6]);
% sigma_eps = Var / (1 - phi(1)*((1+phi(2))/(1-phi(2))) - phi(2)^2);

for i=3673:length(MData{1}.Hour_DataA)
    sigma_eps(i-3672) = 2*sqrt(MData{1}.Hour_DataA(i,
    [1 5 7 8])*VarBeta*MData{1}.Hour_DataA(i,[1 5 7 8])' +
    (Var / (1 - phi(1)^2*((1+phi(2))/(1-phi(2))) - phi(2)^2)));
    sigma_omega(i-3672) = 2*sqrt(MData{1}.Hour_DataA(i,
    [1 5 7 8])*VarBeta*MData{1}.Hour_DataA(i,[1 5 7 8])' + Var);
end
%% Standard Error Will all variables
for i=3673:length(MData{1}.Hour_DataA)
    sigma_omega(i-3672) =
    sqrt(MData{1}.Hour_DataA(i,[1 5:10])*VarBeta*MData{1}.Hour_DataA
    (i,[1 5:10])' + Var);
end
% endd = length(MData{1}.Hour_DataA);
Y2 = EstMdl.Intercept*1 + Beta2*MData{1}.Hour_DataA(1:end,5:10)';
r = MData{1}.Hour_DataA(:,4) - Y2';
Y2_new = EstMdl.Intercept*1 + Beta2*MData{1}.Hour_DataA(3:end,5:10)'
+ phi(1)*r(2:end-1)' + phi(2)*r(1:end-2)';
var(Y2_new(161:end)'-MData{1}.Hour_DataA(163:end,4))
errors = MData{1}.Hour_DataA(163:end,4)-Y2_new(161:end)';

%% Standard Error with three variables
% for i=3673:length(MData{1}.Hour_DataA)
%     sigma_omega(i-3672) = 2*sqrt(MData{1}.Hour_DataA(i,
[1 5 7 8])*VarBeta*MData{1}.Hour_DataA(i,[1 5 7 8])' + Var);
% end
% endd = length(MData{1}.Hour_DataA);
Y2 = EstMdl.Intercept*1 + Beta2*MData{1}.Hour_DataA(1:end,[5 7 8])';
r = MData{1}.Hour_DataA(:,4) - Y2';
Y2_new = EstMdl.Intercept*1 + Beta2*MData{1}.Hour_DataA(3:end,[5 7 8])'
+ phi(1)*r(2:end-1)' + phi(2)*r(1:end-2)';
```

```
var(Y2_new(161:end)'-MData{1}.Hour_DataA(163:end,4))
errors = MData{1}.Hour_DataA(163:end,4)-Y2_new(161:end)';

%
% sigma_eps = Var / (1 - phi(1)*((1+phi(2))/(1-phi(2))) - phi(2)^2);

%%
% plotstart = 163;    plotend = 4343;%length(MData{1}.Hour_DataA);  %all data
% plotstart = 721;   plotend = 1464;    %OCT
% plotstart = 1464;   plotend = 2184;    %NOV
% plotstart = 3076;   plotend = 3672;    %JAN
plotstart = 3673;   plotend = 4343;    %FEB

Yavg = mean(MData{1}.Hour_DataA(plotstart:plotend,4));
Yavg_ars = mean(Y2_new(plotstart:plotend-2));
Yavg_arm = mean(Y2(plotstart:plotend));
%% Plot the prediction in top plot with the error in
the bottom plot for a stationary location
% plotstart = 163;    plotend = length(MData{1}.Hour_DataA);  %all data
% plotstart = 721;   plotend = 1464;    %OCT
% plotstart = 1464;   plotend = 2184;    %NOV
% plotstart = 3076;   plotend = 3672;    %JAN
plotstart = 3673;   plotend = 4343;    %FEB
maxx = length(MData{1}.Hour_DataT(plotstart+2:plotend));
maxx_sig(1:maxx) = 196.6889; %max(sigma_omega);
figure
% subplot(2,1,1)
hold on
% plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,4)./Yavg,'k','LineWidth',1,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),Y2_new(plotstart:plotend-2)'
./Yavg,'r','LineWidth',1,'LineStyle','--')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),(Y2_new(plotstart:plotend-2)'
- sigma_omega(1:end-2)')./Yavg,'b','LineWidth',0.8,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),(Y2_new(plotstart:plotend-2)'
+  sigma_omega(1:end-2)')./Yavg,'b','LineWidth',0.8,'LineStyle','-')
% ylabel('Neutron Counts')
% subplot(2,1,2)
hold on
plot(MData{1}.Hour_DataT(plotstart+2:plotend),((MData{1}.Hour_DataA
(plotstart+2:plotend,4)-Y2_new(plotstart:plotend-2)'))./Yavg,'r','LineWidth',
1,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,1)-1),'k','LineWidth',1,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),maxx_sig./Yavg,'b','LineWidth',
0.8,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),(-1)*maxx_sig./Yavg,'b',
'LineWidth',0.8,'LineStyle','-')
ylabel('Error');    xlabel('Time')
%% Plot the prediction in top plot with the error in the bottom plot for a
mobile location
% plotstart = 163;    plotend = length(MData{1}.Hour_DataA);  %all data\
% plotstart = 2000;   plotend = 2900;
% plotstart = 721;   plotend = 1464;    %OCT
```

```
% plotstart = 1464;  plotend = 2184;    %NOV
% plotstart = 3076;  plotend = 3672;    %JAN
plotstart = 3673;  plotend = 4343;    %FEB
maxx_m = length(MData{1}.Hour_DataT(plotstart+2:plotend));
maxx_sig_m(1:maxx_m) = max(sigma_eps);

figure
subplot(2,1,1)
hold on
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,4)./Yavg,'k','LineWidth',1,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart:plotend),Y2(plotstart:plotend)'./Yavg,'r',
'LineWidth',1,'LineStyle','--')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),(Y2(plotstart:plotend-2)'
- sigma_eps(1:end-2)')./Yavg,'b','LineWidth',0.8,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),(Y2(plotstart:plotend-2)'
+  sigma_eps(1:end-2)')./Yavg,'b','LineWidth',0.8,'LineStyle','-')
ylabel('Neutron Counts')
subplot(2,1,2)
hold on
plot(MData{1}.Hour_DataT(plotstart:plotend),((MData{1}.Hour_DataA
(plotstart:plotend,4)-Y2(plotstart:plotend)'))./Yavg,'r','LineWidth',1,
'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,1)-1,'k','LineWidth',1,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),maxx_sig_m./Yavg,'b',
'LineWidth',0.8,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart+2:plotend),(-1)*maxx_sig_m./Yavg,'b',
'LineWidth',0.8,'LineStyle','-')
ylabel('Error');    xlabel('Time')

%%
maxx = length(MData{1}.Hour_DataT(plotstart+2:plotend));
maxx_avg(1:maxx) = 1280.7;
figure, hold on
% plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Y2(plotstart:plotend)'),'r','LineWidth',1,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Y_ols(plotstart:plotend)')./Yavg,'r')     %OLS
plot(MData{1}.Hour_DataT(plotstart+2:plotend),((MData{1}.Hour_DataA
(plotstart+2:plotend,4)-Y2_new(plotstart:plotend-2)'))./Yavg,'r','LineWidth',
1,'LineStyle','-')  %AR Stationary
% plot(MData{1}.Hour_DataT(plotstart:plotend),((MData{1}.Hour_DataA
(plotstart:plotend,4)-Y2(plotstart:plotend)'))./Yavg,'r','LineWidth',1,
'LineStyle','-')    % AR Mobile
plot(MData{1}.Hour_DataT(plotstart:plotend),(MData{1}.Hour_DataA
(plotstart:plotend,4)-Yavg)./Yavg,'k--')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),maxx_avg./Yavg,'k',
'LineWidth',0.8,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),(-1)*maxx_avg./Yavg,
'k','LineWidth',0.8,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),maxx_sig./Yavg,'r',
'LineWidth',0.8,'LineStyle','-')
% plot(MData{1}.Hour_DataT(plotstart+2:plotend),(-1)*maxx_sig./Yavg,
```

```
'r','LineWidth',0.8,'LineStyle','-')
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA
(plotstart:plotend,1)-1,'k','LineWidth',1,'LineStyle','-')
%% The contribution of each variable
variable = 6;
Y2 = Beta2(variable)*MData{1}.Hour_DataA(1:end,variable+4)';
plotstart = 3076;   plotend = length(MData{1}.Hour_DataA);    %JAN
% plotstart = 163;    plotend = length(MData{1}.Hour_DataA);      %all data

figure; hold on
yyaxis left
plot(MData{1}.Hour_DataT(plotstart:plotend),MData{1}.Hour_DataA(plotstart:plotend,4)
,'LineWidth',1.5)
ylabel('Neutron Counts');   xlabel('Time')
yyaxis right
plot(MData{1}.Hour_DataT(plotstart:plotend),Y2(plotstart:plotend)','LineWidth',1.5,
'LineStyle','--')
```