12-2005

A Dynamic Ampacity Model for the Testing of Advanced Conductors

Matthew Benjamin Sooter

To the Graduate Council:

I am submitting herewith a thesis written by Matthew Benjamin Sooter entitled "A Dynamic Ampacity Model for the Testing of Advanced Conductors." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Leon Tolbert, Major Professor

We have read this thesis and recommend its acceptance:

Jack Lawler, Fangxing Li

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Matthew Benjamin Sooter entitled "A Dynamic Ampacity Model for the Testing of Advanced Conductors." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Leon Tolbert
Major Professor

We have read this thesis
and recommend its acceptance:

Jack Lawler

Fangxing Li

Accepted for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

A Dynamic Ampacity Model for the Testing of Advanced Conductors

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Matthew Benjamin Sooter

December 2005

Dedication

To My Family

and

A Small Dog with a Long Journey

Acknowledgements

There are a number of people who I would like to thank for supporting me in the completion of this thesis. First, I would like to thank my advisor, Dr. Leon Tolbert for his advice and guidance both towards this thesis and towards my personal growth as an engineer.

I would like to thank my committee members, Dr. Jack Lawler and Dr. Fangxing Li, for their helpful suggestions.

I would like to provide a special thanks to John Stovall for not only financially supporting this thesis, but providing me the opportunity to work at Oak Ridge National Laboratory. Without his guidance, wisdom, and friendship this thesis would not have been possible.

I would like to thank all of my peers at The University of Tennessee for not only helping me getting here, but also providing ample distraction along the way.

Abstract

Advanced Conductors are an intriguing new technology that promises to help the utility industry increase the transmission capacity available in the United States. Manufacturers of advanced conductors promise double the current carrying capacity of normal Aluminum Clad Steel Reinforced conductor through a combination of increased thermal limits and decreased thermal elongation.

This thesis will outline a control developed for the Power Line Conductor Accelerated Test Facility at Oak Ridge National Laboratory. This custom facility is designed for the testing and rapid aging of advanced overhead conductors as part of a U.S. Department of Energy initiative to assist the development and deployment of new transmission technologies. The control outlined will make use of an ampacity model capable of accurately describing the current – temperature relationship of bare overhead conductor at an outdoor facility. This model will be used to maintain test conductors at specific elevated temperatures for extended periods of time. It is hoped that these tests will verify the thermal limits of the advanced conductors and encourage utilities to begin deploying them in the field.

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1

INTRODUCTION

Ever since the first high power line in 1891, transmission lines have changed the face of the power industry. For the first time, loads could now be geographically distant from their generators. This ability created a power industry that could effectively leverage economies of scale to produce cheap, reliable electricity for all corners of society. Over the years transmission lines have become increasingly more important as generators attempt to push electricity over even greater distances to satisfy the appetite for a newly deregulated power market.

Unfortunately, investments in transmission have not kept pace with the growing demands for electricity. This lack of investment has begun to lead to bottlenecks in the grid that not only diminish America's ability to trade electricity across the country, costing consumers billions, but also leads to reliability problems and potentially devastating targets for national security. In light of the problems with the present transmission system, a number of ideas have been proposed to solve the issue.

One of the most promising solutions is the use of advanced conductors for both new lines and reconductoring old lines. Advanced conductors may provide a cost effective solution capable of dramatically repairing the grid. These conductors are so new, however, the ever-conservative utility industry is still regarding them with some degree of uncertainty. Because of this, the Oak Ridge National Laboratory (ORNL) in conjunction with the Department of Energy (DOE), has constructed a Powerline Conductor Accelerated Test (PCAT) facility capable of thoroughly testing and rapidly

aging these advanced conductors. It is hoped that performing well-documented tests on these conductors will encourage the utility industry to begin taking advantage of this technology and begin deploying advanced conductors in the field themselves.

1.1 National Transmission Grid

The National Transmission Grid (NTG) consists of a complex network of high-voltage power lines that facilitate the transport of electricity from generators to loads. This transmission system is the backbone of the electric infrastructure and must always be available to carry the electricity that the U.S. needs.

The transmission system originally consisted of only a number of vertically integrated utilities regulated by the government. These utilities had weak interconnections that provided reliability for their systems and allowed them to ship out excess generation. Over the past decade, however, the United States has chosen to deregulate many of its utilities, splitting them into generation, transmission, and distribution companies. The idea behind deregulation was to allow competition in the electricity marketplace to, hopefully, make the NTG more efficient, more reliable, and cheaper to the consumer. Doing this, however, encourages the electric market to "ship" cheap electricity long distances over transmission lines to the load centers. While this concept has merit, it places a huge strain on the transmission system because it asks it to do something it was never designed to do. The original reason for transmission lines was to take clean electric power from remote coal fired power plants to their local urban load centers. These lines would typically average some 18 miles in length and would rarely be

involved in interconnecting other operators. Now, in a competitive marketplace, these same lines are asked to facilitate the regional movement of electricity, dramatically increasing the strain on the system.

This strain in the transmission system has been exacerbated by the fact that investments in new transmission capacity have not kept pace with growth in demand and investments in new generation. The past decade has seen a steady trend of growth in demand outpacing the growth in transmission capacity, as indicated in Figure 1.1 [1]. Transmission problems have also been created by the incomplete transformation to a competitive wholesale electricity market. Because the original NTG was not designed with the new electricity model in mind, the system has become congested by bottlenecks

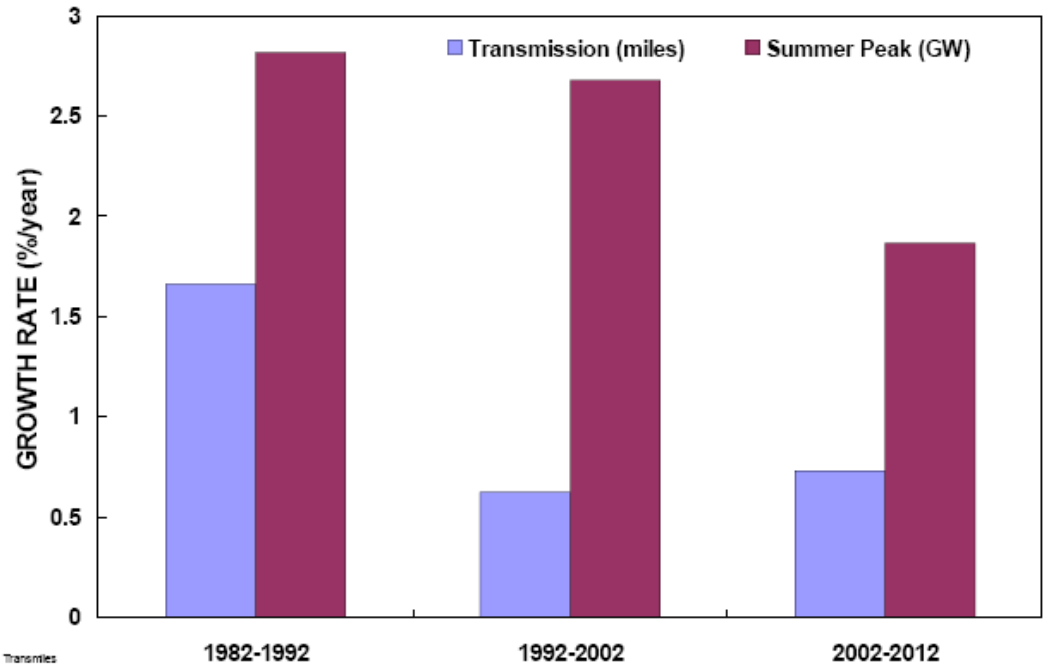

Figure 1.1. Annual average growth rates in U.S. transmission capacity and peak demand [1].

that are increasing the cost of electricity to the consumer and increasing the risk of blackouts [2].

Lower investments in transmission have also begun to raise substantial reliability issues. Because load has been outpacing transmission growth for some time, the transmission lines that are in service have become more important. Lines that were intended to provide backup must now be operated, and many lines are being operated much closer to their electric and thermal limits, decreasing the utility's ability to handle emergency overload. Obviously the Northeast blackout of August 2003 is a prime example of some of the effects of a fragile transmission system. The energy crisis in the western United States before that was also brought on by deficiencies in the transmission grid. These two failures of the national grid serve to highlight the fact that real reliability problems do exist and need to be addressed [2].

Numerous reports have been released in the last five years detailing many of the failures that exist in the current transmission system. Almost universally, however, these reports differ on how these problems can be solved, and how much fixing them will cost. One of the primary problems people have had solving the transmission crisis is the fact that new lines are not only extremely expensive, but require an extraordinary amount of time to get approved. In light of this, an ideal solution would be to take the existing right of ways and with minimal capital expenditure upgrade them to handle present and future loads. Fortunately, a solution that is capable of that already exists. Before this new technological solution can be discussed, however, the requirements and attributes of a transmission line must be understood.

1.2 Transmission Line Requirements

While the job of a transmission line may sound easy, they perform no easy task. At the most basic level, a transmission line is simply a large conductor. It is a conductor that is designed to move large amounts of power from a generator to a load, typically at voltages in excess of 100kV. Because transmission lines carry electricity at such a high potential they require substantial ground clearances for the general safety of the public, and to prevent flashover to terrestrial objects such as trees. These ground clearances require that they be strung in high towers and at very high tensions. Transmission lines are also normally bare, ie. they are not insulated. This exposes the lines to the damaging effects of nature on a daily basis. With all of these requirements, a number of attributes have to come together to make an effective line.

First, the line has to have a low resistance to minimize the losses associated with transmitting the electricity. Second, the line has to be strong enough to handle being pulled into the transmission towers and strung under high tension. Third, because the lines must stay above a minimum ground clearance they cannot sag, due to heat or icing, below that minimum level. Fourth, the line must be able to withstand the stresses of weather and the corrosive nature of rain and pollution. Fifth, the line must not be prohibitively expensive.

Over the years the utility industry has developed a conductor that performs fairly well in all these categories. The Aluminum Clad Steel Reinforced (ACSR) conductor is

Figure 1.2. ACSR

the standard for almost all transmission lines today. This conductor consists of aluminum strands wrapped around steel strands as shown in Figure 1.2. The aluminum strands act as the primary conductor, while the steel strands are used for mechanical strength. ACSR is so widely used because it is cheap, reliable, and can accomplish all the jobs noted above. Typically ACSR is thermally limited to approximately 100°C. This is due to both the negative effects of annealing the aluminum and the large sag generated by the expanded steel. Because the primary source of heat in a conductor is generated by the current flowing through it, the thermal limit of the conductor effectively limits the current that the conductor is able to safely carry.

1.3 Advanced Conductors

Advanced conductors promise the ability to raise the thermal limits of traditional ACSR and allow a line to safely conduct larger current levels. In order to accomplish this, scientists have attacked the two previous conditions. They have reduced the negative effects of heating up the aluminum, causing annealing, and found ways to reduce the thermal elongation of the core, so as to reduce sag.

Standard Aluminum Clad Steel Reinforced cables are made up of simple circular strands of twisted steel cable surrounded by strands of aluminum cable. Advancements of this simple conductor have been going on for some time. In 1974 a steel supported aluminum clad conductor (ACSS) was first introduced. This conductor made use of fully annealed aluminum wires that would not be adversely affected by high temperatures. Because they were fully annealed, the entire mechanical load of the conductor was maintained by the steel core. This core was eventually upgraded from galvanized steel rods to mishmetal steel rods, a combination of steel and mixed rare earth elements, that could withstand higher temperatures and would sag less [3].

New technology in wire forming created the Aluminum Clad Steel Reinforced Trap Wire (ACSR/TW) and Aluminum Clad Steel Supported Trap Wires (ACSS/TW). These trap wires, shown in Figure 1.3, are characterized by the fact that the aluminum conductors instead of being circular are trapezoidal shaped to take up previously empty space inside of a conductor.

Today advanced conductors have culminated in the composite conductor, or ACCR / ACCC conductors. These conductors have composite cores made up of

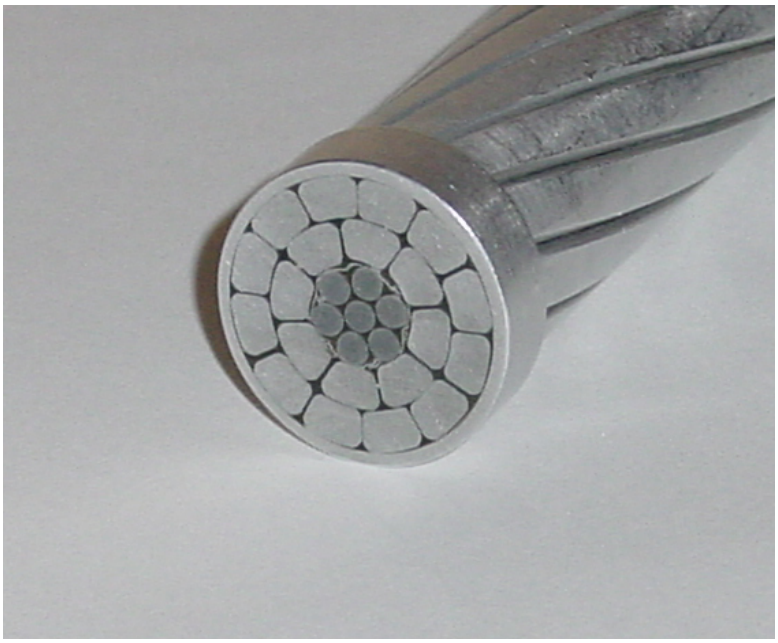

Figure 1.3. ACSR/TW

everything from metal alloys to fiberglass. The two primary corporations currently manufacturing such cables are 3M with its Aluminum Clad Composite Reinforced (ACCR) conductor and CTC with its Aluminum Clad Composite Core (ACCC) conductor. These conductors are shown in Figures 1.4 and 1.5 respectively. Both of these cables use fully annealed aluminum wires on the outside that are supported by these new composite core wires on the inside. What makes the composite cores special is their rate of thermal expansion which is several times less than that of steel. Decreasing thermal expansion is critical because on the majority of transmission lines the thermal limit for that line is first reached, not when the temperature is great enough to cause the steel to mechanically fail, but when the temperature is great enough to cause the steel to sag below regulated clearances.

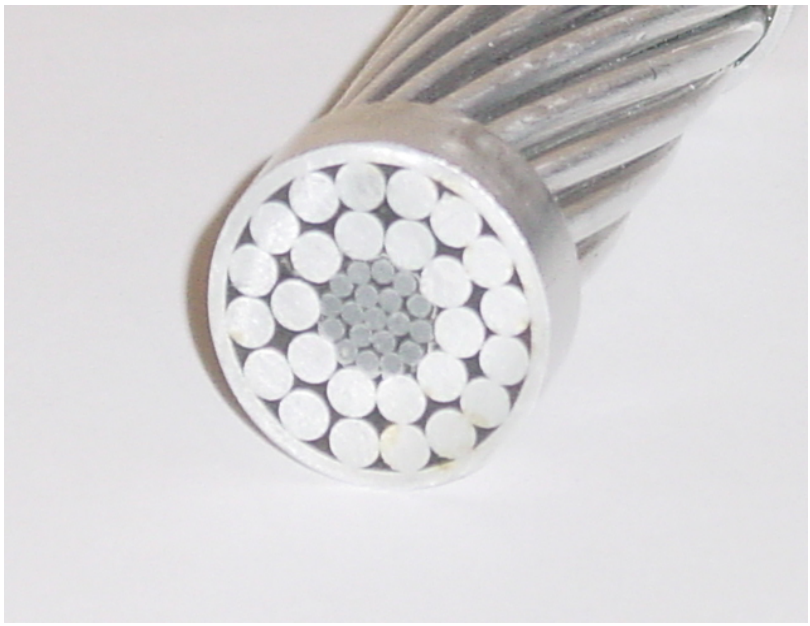

Figure 1.4. 3M Aluminum Clad Composite Reinforced

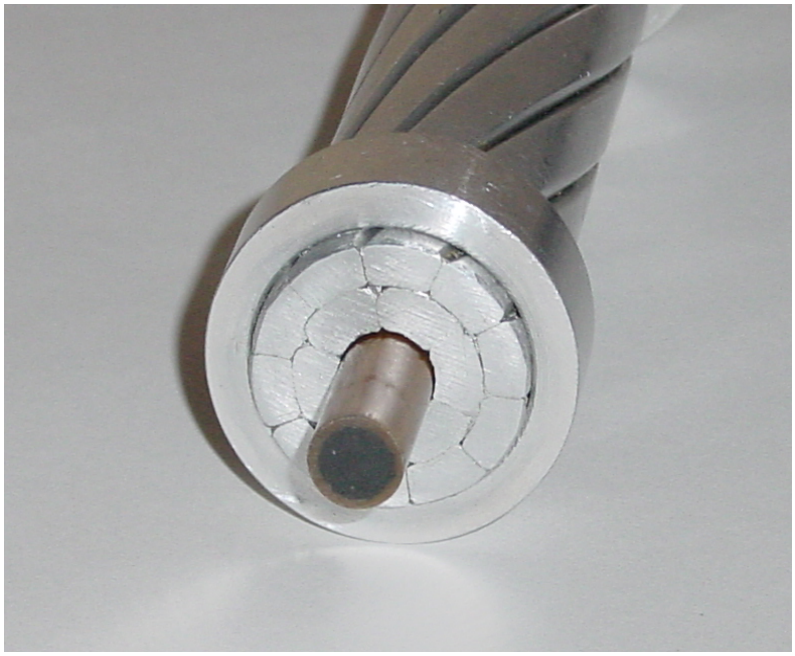

Figure 1.5. CTC Aluminum Clad Composite Core

One of the most exciting things about these advanced conductors is the fact that identical diameter composite conductors can be hung in place of standard ACSR conductors. The composite conductors already in production require nearly the same amount of tension as their ACSR equivalent and could therefore be strung on towers designed for a certain size ACSR conductor. This has the potential to allow utilities to be able to merely reconductor their existing transmission lines and greatly increase their transmission capacity.

Composite conductors also allow for smaller sized conductors to achieve the same amount of ampacity as a larger ACSR conductor. Because the composite conductors are more heat resistant and therefore can conduct more current than a normal ACSR conductor, if a utility wishes to build a new line with a specific ampacity a smaller composite conductor may be used. This is significant because smaller conductors require less tension when they are strung, and this reduced tension translates into reduced tower size. Lowering the tower size not only reduces the capital spent on transmission towers, but potentially allows for smaller right of ways, which could greatly reduce the cost of a new transmission line to a utility.

1.4 Thesis Outline

The purpose of this thesis is to create a control method capable of running automated tests on advanced conductors. This control method will require calculating the real time ampacity for a bare overhead conductor. The information obtained from this

will then be used to control the current level running through the test lines so that they can be maintained at specific temperatures for long periods of time.

1.5 Chapter Outlines

Chapter 2 is summary of capabilities of the PCAT test facility and some of the design considerations that went into its construction.

Chapter 3 is a detailed discussion of the ampacity model developed to describe the real time behavior of the test lines.

Chapter 4 discusses PCATamp, the program written to implement the ampacity model and control testing at the ORNL PCAT facility.

Chapter5 discusses the results of using this control method to maintain a conductor at a specific temperature.

Chapter 6 provides conclusions and recommendations for future work.

Chapter 2

POWERLINE CONDUCTOR ACCELERATED TEST FACILITY

The control developed later on in this thesis was designed for use at Oak Ridge National Laboratory's Powerline Conductor Accelerated Test (PCAT) Facility. The purpose of the PCAT facility is to allow the rapid testing and certification of new and advanced bare overhead transmission conductors. It is hoped that by performing these tests, utilities will be encouraged to begin investing in some of these new technologies that promise to help relieve some of the strain on the national transmission grid. The PCAT facility was one of the first parts of the Department of Energy's (DOE) National Transmission Technology Research (NTTRC) initiative meant to promote research and development in transmission technology.

This chapter will begin by giving an overview of the PCAT facility and how it typically operates. After that some of the design considerations that went into its construction will be discussed. The chapter will then move on to a complete description of the many components that make up the facility and will wind up with a discussion of the numerous measurement systems utilized.

2.1 Overview

The PCAT facility, pictured in Figure 2.1, allows the testing of a 2400ft section of bare overhead conductor. This 2400ft section of conductor consists of two 1200ft sections of wire. These sections are strung on both sides of three high-voltage

Figure 2.1. PCAT Site Photo

transmission towers according to the IEEE Standard 524-2003 for installing overhead transmission line conductor. The facility is visualized in Figures 2.2 and 2.3. Conductors strung at the test facility will then typically undergo a number of rapid aging tests where the temperature of the conductor will be increased to 300°C. This rise in temperature is accomplished by running large currents through the line by connecting it to the positive and negative poles of a 2MW DC converter. The specifics of each test are determined on a case by case basis with each of the customers utilizing the test center. An array of measuring devices are used throughout PCAT, from a complete weather station, to thermocouples measuring actual conductor temperature, to readings of the voltage and current present on the line. All of this data is then collected back at a measurement building, on site, for later analysis by ORNL and its customers.

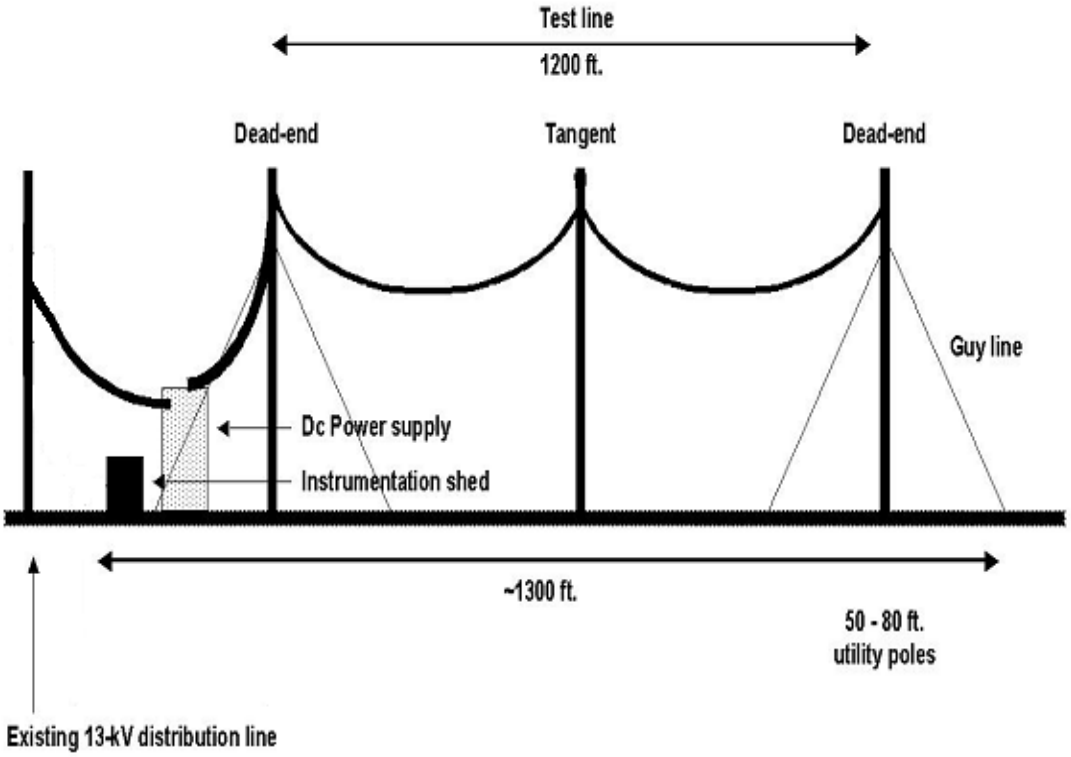

Figure 2.2. PCAT Side view Diagram

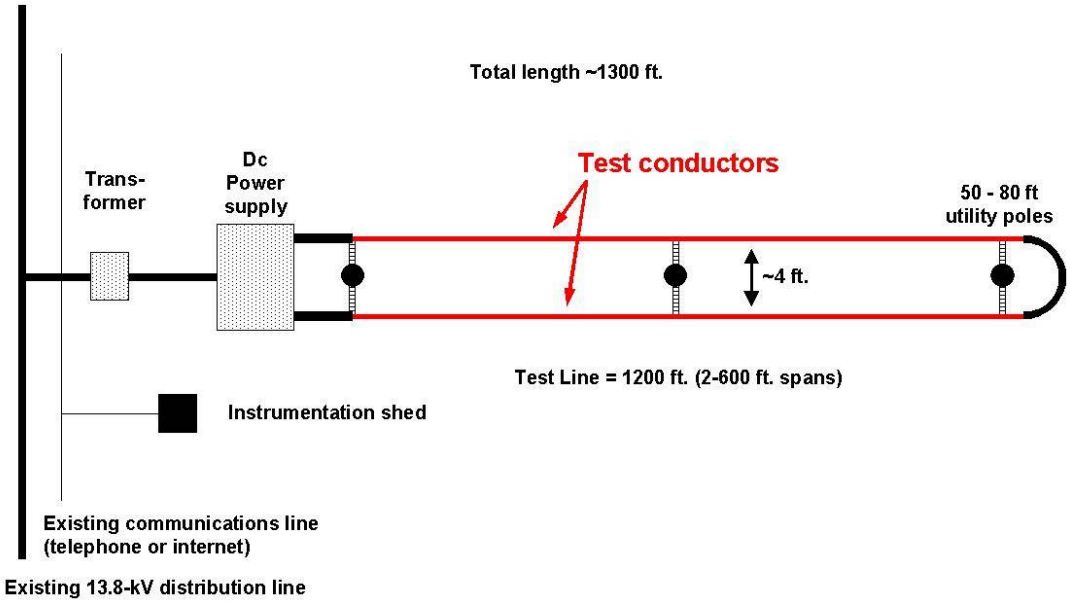

Figure 2.3. PCAT Overhead Diagram

2.2 Design

Basic design elements were taken from a similar facility built at Georgia Power to validate ampacity models. The apparatus used by Georgia Power and Georgia Tech consisted of a 700ft section of 336 kcmil ACSR that could be energized with the use of a 480V transformer [4]. Their setup also included a weather station, the use of ungrounded thermocouples to measure actual conductor temperature, and the use of desktop computers to facilitate the data collection from all of their measurement devices [4]. Many of these elements would be carried over or taken into consideration during the construction of PCAT.

The differences that exist between the two facilities are primarily due to their different goals. Georgia Tech was interested in validating ampacity models. They needed to be able to load the line and to regulate it, but they did not need to control the current on the line in real time [4]. PCAT was designed to run tests on conductors that required fine control of the currents placed online to either maintain a specified current or temperature. Utilizing auto-transformers at ORNL as they did in Georgia simply could not satisfy this need. Therefore the researchers at ORNL decided to use a DC power source instead of an AC power source. The other primary difference between the facility at Georgia Power and at ORNL is an issue of scale. Georgia Power's apparatus used poles designed for distribution level systems to hang their lines [4]. Distribution poles were not tall enough, however, for the PCAT facility. Because the facility at ORNL was designed for high temperatures, it had to be able to maintain safe clearances for the sags

produced by these loads. Researchers at ORNL also wanted to be able to string a much larger section of conductor than was done in Georgia. The combined need to support the increased strain of a larger conductor and maintain safe clearances from conductor sag, prompted ORNL to consult with the Tennessee Valley Authority (TVA) on the best way to proceed. They recommended the use of heavily guyed transmission grade towers. Specifically, three towers were used 600ft apart to allow two 1200ft sections of conductor to be strung on them.

The measurement requirements of both facilities were very similar and therefore their setups for measurement are both similar. The two test centers were both interested in the conductor temperature, and the best way to acquire that temperature was to have thermocouples on the line. PCAT chose to use ungrounded thermocouples connected to a fiber optic network to ensure electric isolation between the line and the data collection computers. Both test centers are also outdoor facilities which created a need for weather stations.

2.3 PCAT Facility Components

A number of structures, equipment, and devices are necessary to operate the Powerline Conductor Accelerated Test facility efficiently and safely. This section will attempt to detail many of the components used in this system, from the structures utilized, to the equipment required for each of the many measurements that are taken.

2.4 Structures

Six major structures make up the test site. There are two dead-end towers that consist of two 85ft steel poles and a midspan tower composed of a single 85ft steel pole. Then there is a tractor trailor used to house the DC power supply and circuit breakers. Next to that is a mobile office building that houses the PCs and other equipment necessary to control the power supply and access all the measurement equipment. Finally there is a fiberglass tower used to string a wire over the two buildings for lighting protection. These elements are identified in two pictures of the PCAT facility shown in Figures 2.4 and 2.5.

Figure 2.4. Overhead view of PCAT

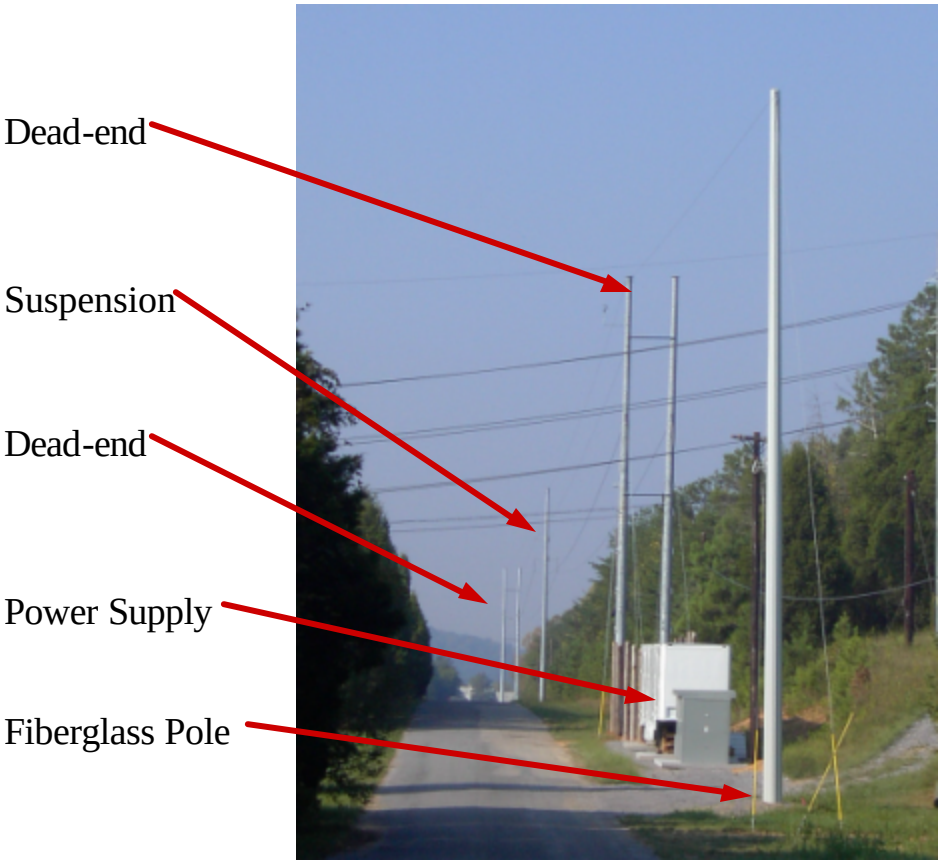

Figure 2.5. View of PCAT

2.5 Power Supply

The heart of the PCAT power supply is a Transrex ISR 2294, a 2MW SCR-based DC converter. This unit allows accurate control of the voltage and current applied to any of the test lines. Power is supplied to this converter through a 4.16kV air-cooled transformer that is connected to a 13.8kV line that runs near the facility. Both circuit breakers and overload circuit protection are included for safety. Power is delivered from the DC converter to the test line through four paralleled sections of very large ACC conductor. This was done to ensure that the wire needed to put power on the test line would not in any way affect or bottleneck the test system. The ACC conductors are connected to large bus plates located on the dead-end tower closest to the power supply. Jumper cables composed of the conductor currently being tested are used to connect the bus plates to the 2400ft section of line.

2.6 Instrumentation

The PCAT facility uses several different types of instrumentation to record what is occurring on a test line at any point in time. Measurements are recorded once every minute and stored onto PCs for later data analysis. This section will go through many of the measurements taken at the PCAT facility and the equipment necessary to take those measurements.

Conductor temperature is measured by 40-80 thermocouples placed on or inside the line being tested. These ungrounded thermocouples are placed along the length of the entire line and its dead-ends and splices. Thermocouples are either attached using special

high temperature conductive tape made by 3M or by being placed inside of the conductor by bird caging it and then allowing friction to hold it in place.

Sag in the line, created by elevating its temperature, is measured directly by a laser centered between the mid-span tower and the dead-end tower nearest to the power supply. Sag may also be measured indirectly by two load cells connected to both sides of the conductor, these units also make it possible to constantly monitor the tension that the line is under.

A number of devices are used to also measure the weather conditions present at the test facility. The weather attributes recorded at the PCAT facility are listed below.

- Ambient temperature (°C)
- Solar radiation (watts per meters squared)
- Wind speed (feet per second)
- Wind direction (degrees)
- Wind elevation (degrees)

The majority of the measurement apparatuses used are connected to the measurement trailer by an RS-485 network. The measurements are read by PC and logged to dated files every minute. These files are then stored indefinitely so the data may be analyzed later as necessary. A single PC controls the power supply and measurement network. A second PC copies the online data files and runs a web server that allows for easy remote monitoring of the system.

2.7 Summary

The Power Line Conductor Accelerated Test facility is uniquely setup for the testing of advanced overhead conductors. Its ability to precisely control very high current levels, while accurately measuring multiple line characteristics, makes it a one of a kind research facility. At the same time, the use of standard stringing equipment and transmission grade towers places test conductors in a setting similar to what they would experience in the field. It is hoped that by providing these capabilities PCAT will encourage utilities to adopt the conductors tested here earlier, providing them with a way to cost effectively increase their transmission capacity.

Chapter 3

AMPACITY

Calculating the current-temperature relationship of overhead conductors is not a new area of study. Over the years transient models have been developed that are capable of calculating the temperature of an overhead conductor in real time. In 1993 IEEE released a revised version of their standard, 738, for calculating the current-temperature relationship of bare overhead conductors. That standard outlines the basic heat balance equation, discussed in detail later, first proposed by House and Tuttle [5]. The equation:

$$q_c + q_r + mC_p \frac{dT_c}{dt} = q_s + I^2 \cdot R(T_c) \tag{3.1}$$

broke up the energy balance into the heat created by conductive resistance, $I^2 \cdot R$, and the heat absorbed from the sun, q_s, with the heat lost to space and air through convection, q_c, and radiation, q_r. The transient equation also accounted for the heat storage of the line represented by $mC_p \frac{dT_c}{dt}$. Over the years more and more accurate methods have been developed to calculate all these different parts of the heat balance equation. Much of the advancement in these individual models has been made possible by the ability to solve their more complex mathematical forms with cheap computer power. Indeed, later in this chapter, a modified method is introduced for calculating the heat absorbed from the sun that is computationally taxing by even modern PC hardware.

The first development of an ampacity model for bare overhead conductors was done by House and Tuttle in 1959. This was the first model to take into account the

various forces of weather that would affect the conductor. House and Tuttle also did the first dissection of the heat balance equation into its various related parts, ie. heat loss through convection, and radiation and heat gain through solar radiation absorption and line losses because of impedance [6]. The field was largely left alone until the late 1970's when M. W. Davis wrote several articles on the subject of thermal rating overhead lines.

In 1982 Dr. W.Z. Black of Georgia Tech and many of his students began developing an ampacity model that would take advantage of new computing equipment. Dr. Black's student, W.R. Byrd, wrote this first ampacity program for his Master's Thesis [7]. This work was expanded by several more of Dr. Black's students to take advantage of newer computers and better modeling techniques that maintained the model's accuracy under high temperatures. PCATamp, the ampacity model that was developed for PCAT, was based largely on the work of Dr. W.Z. Black, W.R. Byrd, R.L. Rehberg, and S.L. Chen and could not have been done without their previous work [7,8,9].

IEEE also created a standard in 1986, revised in 1993, for calculating the current-temperature relationship of bare overhead conductors. IEEE STD 738-1993 details a number of equations and algorithms that can be used to create an ampacity model. IEEE's standard was also heavily referenced while creating PCATamp [5].

3.1 Ampacity Model

In order to accurately control the PCAT facility, it was quickly realized that the forces at work on the conductor needed to be understood. Given the interest in

controlling the temperature of the line by adjusting the current running through it, it made sense to develop a model of the current-temperature relationship, or ampacity, of the conductors at the PCAT facility. Modern ampacity models account for not only the heat generated by conduction, but also the heat lost and gained due to the weather. Because of the extensive measurement capabilities present at PCAT for everything from conductor temperature to solar radiation, it appeared that PCAT was in a prime position to utilize the body of work written on the ampacity of bare overhead conductors. The hundreds of thousands of recorded data points at PCAT provided the information necessary to build an accurate model that would most closely resemble what conductors experienced on a day-to-day basis.

3.1.1 Heat Exchange

The basic heat exchange equation for a bare overhead conductor is as follows [5,7]:

$$q_c + q_r + mC_p \frac{dT_c}{dt} = q_s + I^2 \cdot R(T_c) \tag{3.2}$$

where:

q_c = heat loss through convection (watts per linear foot of conductor)

q_r = heat loss through radiation (watts per linear foot of conductor)

mC_p = heat capacity of conductor (watt minutes per foot degrees centigrade)

T_c = conductor temperature (degrees centigrade)

t = time (minutes)

q_s = heat gain from the sun (watts per linear foot of conductor)

I = conductor current (dc amps)

R = resistance per linear foot of conductor at T_c (ohms per foot)

Rearranging this and solving the differential equation:

$$\frac{dT_c}{dt} = \frac{I^2 \cdot R(T_c) + q_s - q_c - q_r}{mC_p} \tag{3.3}$$

makes it possible to calculate the conductor temperature at a specific time based on the knowledge of each of the above conditions. An explanation of how this transient equation is solved will be provided later in this chapter.

3.1.2 Q_{gen}

Q_{gen} refers to the rate of heat generation in the line per unit length due to electrical current. As part of the heat exchange equation, Q_{gen} refers to $I^2 \cdot R(T_c)$. A great deal of work has been done developing polynomial equations that accurately describe the relationship between temperature and electrical impedance of a metal. However, at the PCAT facility, solving a complex series of equations to describe the conductors overall impedance is unnecessary because the voltage and current on the line can be measured directly. This is desirable for a couple of reasons. Firstly, it reduces the amount of calculation that needs to be done when the model is used in a real-time environment, and secondly it removes the dependence on accurately knowing the temperature-impedance relationship of a material. For typical conductor metals such as aluminum and steel, this

information is well documented. However, part of PCAT's design purpose is for the testing of advanced overhead conductors, many of which are made out of new materials whose temperature-impedance relationship is either unknown or undocumented in the public sector.

For PCATamp, Q_{gen} is solved fairly directly. The resistance R is found by:

$$R = \frac{V/I}{2400} \tag{3.4}$$

where:

V = measured voltage (dc volts)

I = measured conductor current (dc amps)

Because R is in ohms per linear foot of conductor V/I is divided by the length of the conductor, 2400 feet.

After the resistance is found, Q_{gen} is calculated exactly like in the heat exchange equation. There is no need to worry about the temperature-resistance relationship because the resistance is being measured online.

It is worth noting that this method of calculating the heat produced by power losses is different from the bulk of the work performed in ampacity modeling. This is because most ampacity models are set up to describe the power losses of an AC system, whereas PCAT uses a DC based power supply. The primary difference involves the fact that in an AC system skin-effect will increase the overall impedance of the line to some degree. In ampacity models designed for AC systems, this is accounted for in the temperature-resistance curves that the system uses to track the overall resistivity of the

line. PCATamp does not account for things like skin-effect because it is a DC system and the facility is capable of directly measuring voltage and current online. Because PCAT is a DC system, readers should not assume current-temperature relationships for a specific line would apply in a utility setting under AC conditions. While these relationships are used internally to control the behavior of the test facility, they are not meant to describe the conductor under AC conditions. Cable manufacturers should be consulted for such information.

3.1.3 Mass Specific Heat Capacity

The mass specific heat capacity, mC_p, is an index of how well a material retains heat. While temperature dependent polynomials exist to describe the mass specific heat of normal conductors [7], 3M only provides singular average values for the materials in their advanced composite conductors. This is relevant, as all of the testing of PCATamp has been done on 3M's novel ACCR conductor. 3M states that the mass specific heat for the aluminum used in their conductor is 520 *J/ft°C* and 28 *J/ft°C* for their aluminum composite material. Because independent values are given for the mass specific heat, the equation for mC_p is simply:

$$mC_p = \frac{mC_{pAl} + mC_{pCo}}{60} \tag{3.5}$$

Division by 60 is necessary to convert *J/ft°C* to *Watt Minutes/ft°C*.

3.1.4 Q_{con}

The heat lost from a bare overhead conductor due to convection is built on a well developed method of determining convection heat losses by deriving the Grashof, Nusselt, and Reynolds numbers associated with the system [7,8]. Measured values of wind speed and wind direction are used along with knowledge of the ambient temperature and conductor temperature to realize the rate at which heat will leave the bare metal of the conductor for the air. This method of calculating the heat losses associated with convection was derived by Byrd and Rehberg in their thesis on ampacity [7,8]. The algorithm used to determine Q_{con} will be briefly discussed. For a more thorough explanation it is recommended that the reader consult those two theses.

Variables:

w_{Dir} = measured wind direction (radians)

w_{Speed} = measured wind speed (meters per second)

Averages of these two measurements are used due to the fairly noisy nature of wind measurement.

D = diameter of the conductor (meters)

Z_l = line orientation of the conductor (radians)

T_C = conductor temperature (°C)

T_A = measured ambient temperature (°C)

P_r = Prandtl Number (0.71)

v = kinematic viscosity of air (m^2/s)

?$_{wind}$ = wind direction with respect to line orientation (radians)

First the film temperature around the conductor will be found. This is the temperature of the junction where the metal of the conductor meets the air. The conductor temperature is an average of 10 thermocouples placed on and inside the test conductor. The ambient temperature is measured by thermocouples hanging in the air off of the tower closest to the power supply.

$$T_{film} = \frac{T_C + T_A}{2} \tag{3.6}$$

Next determine the angle of the wind direction with respect to the line orientation by performing a cross product.

$$q_{Wind} = \text{ArcSin}\left((\text{Cos}(Z_l) \cdot \text{Sin}(w_{Dir})) - (\text{Cos}(w_{Dir}) \cdot \text{Sin}(Z_l))\right) \tag{3.7}$$

$$w = \left| \frac{p}{2} - q_{wind} \right| \tag{3.8}$$

Next determine the air viscosity.

$$v_f = (1.0167E-10 \cdot T_{film}{}^2 + 8.7305E-8 \cdot T_{film} + 1.3888E-5 \tag{3.9}$$

Next determine the thermal conductivity of the air.

$$k_f = -2.7628E-8 \cdot T_{film}{}^2 + 7.2316E-5 \cdot T_{film} + 0.023681 \tag{3.10}$$

Next calculate the product of the gravitational constant (g) and the coefficient of thermal expansion for air (B) divided by the kinematic viscosity of air squared.

$$\frac{g \cdot B}{v^2} = 1.024E4 \cdot T_{film}{}^2 - 2.394E6 \cdot T_{film} \_ 1.85E8 \tag{3.11}$$

Next determine the Grashof number.

$$G_r = \frac{g \cdot B}{v^2} \cdot D^3 \cdot (T_C - T_A) \tag{3.12}$$

Next determine the Nusselt number for natural convection.

$$x = \log_{10}(G_r \cdot P_r) \tag{3.13}$$

$$ltNuNa = 0.12724 + (0.02238 \cdot x) + (0.04203 \cdot x^2) - (0.0025973 \cdot x^3) \tag{3.14}$$

$$NuNa = 10^{ltNuNa} \tag{3.15}$$

Next find the Reynolds number.

$$R_e = \frac{w_{Speed} \cdot D}{v_f} \tag{3.16}$$

Next find the Nusselt number for forced convection.

$$ltNuUc = -0.070431 + 0.31526 \cdot \log_{10}(R_e) + 0.035527 \cdot \log_{10}(R_e)^2 \tag{3.17}$$

$$NuUc = 10^{ltNuUc} \tag{3.18}$$

$$NuFc = NuUc \cdot (1.194 - \mathrm{Sin}(w) - (0.194 \cdot \mathrm{Cos}(2w)) + (0.368 \cdot \mathrm{Sin}(2w))) \tag{3.19}$$

Next the convective heat transfer coefficients for natural and forced convection will be found. A third coefficient corresponding to the average of the two will also be found to soften a potential discontinuity when solving the transient equation, discussed further on the next page.

Natural:

$$h_{Na} = \frac{NuNa \cdot k_f}{D} \tag{3.20}$$

Forced:

$$h_{Fc} = \frac{NuFc \cdot k_f}{D} \tag{3.21}$$

Average:

$$h_{Avg} = \frac{\frac{NuNa + NuFc}{2} \cdot k_f}{D} \tag{3.22}$$

Next the actual convective heat loss for each type of convection will be found. Division by 3.2808399 is performed to convert watts per meter to watts per foot.

Natural:

$$Q_{conNa} = \frac{h_{Na} \cdot p \cdot D \cdot (T_C - T_A)}{3.2808399} \tag{3.23}$$

Forced:

$$Q_{conFc} = \frac{h_{Fc} \cdot p \cdot D \cdot (T_C - T_A)}{3.2808399} \tag{3.24}$$

Average:

$$Q_{conAvg} = \frac{h_{Avg} \cdot p \cdot D \cdot (T_C - T_A)}{3.2808399} \tag{3.25}$$

After the different values of convective heat loss have been calculated, the one that is chosen and used to determine the ampacity of the conductor must be determined. For wind speeds that are equal to zero, or too small to measure, the value for natural convection will be used. When there is a large amount of wind indicated by (NuFc > NuNa), the Q value for forced convection will be used. For low speed winds where the Nusselt Number for natural convection is higher than the Nusselt number for forced convection (NuNa > NuFc), the average of the convection values is used. This is done to prevent the transient equation from facing a steep discontinuity when wind speed travels to or from zero. While this aspect is important for a model to be able to run through a section of historic data, it does not apply to the control developed for the PCAT facility. This is because the control is interested in the future temperature of the conductor based on present conditions, therefore only the present wind condition is provided to the model eliminating the need to protect against changing wind patterns.

3.1.5 Q_{rad}

Determining the net radiation transfer between the environment and the conductor is much simpler than determining the heat loss due to convection. It is assumed that the area around the cable can be simplified as a black isothermal environment. The heat loss

due to radiation can then be found as follows, as with the calculation of Q_{con} the derivation of this algorithm can be found in Byrd's thesis [7].

Variables:

D = diameter of the conductor (meters)

T_C = conductor temperature (°C)

T_A = measured ambient temperature (°C)

s = Stefan-Boltzman constant (5.67E-8)

e = emmissivity (0.35)

$$Q_{rad} = \frac{e \cdot D \cdot p \cdot s \cdot (T_C + 273.15)^4 - (T_A + 273.15)^4}{3.2808399} \tag{3.26}$$

3.1.6 Q_{sun}

Calculating the heat absorbed from the sun turns out to be one of the most rigorous calculations in an ampacity model. The reason for this is the need to calculate the azimuth and elevation for the sun at the precise time of interest. Traditionally these calculations have been reduced to simple polynomials that mimic the basic course of the sun throughout a day. This was done because accurate calculations of these values involve a huge number of trigonometric equations that must be calculated to accurately model the movement of planets. Alone these calculations pose no problem for modern computers; but, constantly running these calculations in a real time application can tax even today's fastest processors. Because it is possible to run these calculations on current

equipment, however, an accurate planetary movement model has been incorporated into this model. A section of visual basic code written by Gregg Pelletier based on an algorithm created by the National Oceanic and Atmospheric Administration (NOAA) was used to find the azimuth and elevation of the sun; this algorithm can be seen in Appendix IV. The calculations performed by this section of code were confirmed to match both the web-based azimuth and elevation calculators used by the NOAA, and another solar position algorithm written for the National Renewable Energy Laboratory (NREL). The development of the solar position algorithm is far too rigorous to discuss here and it is recommended that the reader consult NREL's technical report on "Solar Position Algorithm for Solar Radiation Applications" by I. Reda and A. Andreas if he needs more information [10].

The second part of calculating the radiation absorption from the sun is knowing how much solar radiation is present on the line. In previous ampacity models this was often reduced to a table that gave an approximate value of the solar radiation based on no obstructions between the surface and the sun, and the time of year. Luckily, at PCAT there is a solar radiation sensor that can measure the solar radiation present at the facility. This is very nice because traditionally there has been no way to account for clouds, etc. in a model. Once the azimuth of the sun, the elevation of the sun, and the amount of solar radiation is known, the heat absorbed by the sun can be calculated as follows. These equations were taken from the IEEE's Standard 738-1993 and Byrd's thesis [5,7].

Variables:

D = diameter of the conductor (meters)

Z_l = line orientation of the conductor (radians)

Z_c = azimuth of the sun (degrees)

H_c = elevation of the sun (degrees)

a = solar absorptivity (0.35)

l_{Sun} = measured solar radation (watts per meter squared)

$$q = \text{ArcCos}\left[\text{Cos}\left(\frac{p \cdot H_c}{180}\right) \cdot \text{Cos}\left(\frac{p \cdot (Z_c - Z_l)}{180}\right)\right] \tag{3.27}$$

$$Q_{sun} = a \cdot \frac{l_{Sun}}{10.7639104} \cdot \text{Sin}(q) \cdot \frac{D}{12} \tag{3.28}$$

3.1.7 Solution

Solving the heat balance equation requires quite a bit of computational effort. So, as was originally recommended by the authors of DynAmp, a runge-kutta routine was used to perform this calculation [7]. Specifically, the Visual Basic version of PCATamp used a static step-size Runge-Kutta-Fehlberg 4-5 (RKF45) routine, available in Appendix I. This routine was written in Visual Basic, with great difficulty, based on algorithms written in Fortran and C by Watts and Shampine [11]. The Visual Basic routine was checked against solutions to some known differential equations as well as the Matlab version of PCATamp that utilized a built in RKF45 routine with step size control to verify its correct functioning.

3.2 Transient vs Steady State

The heat balance equation can be solved in two forms. It can be simplified to steady state form and solved easily for the current used to achieve a specific temperature; or it can solved in the transient form for the temperature that would be associated with running at a specific current through the line. Using PCAT's historic data, both solutions of the heat balance equation were compared. It was found that the steady state solution, which was significantly less costly computationally, did not provide the accuracy necessary to use it as a control for the PCAT system. It makes sense why it did not work because an outdoor test facility is anything but steady state. The transient method was also tested, and after being run against 247,680 historic data points was found to predict the actual temperature of the line quite accurately. The average difference between the modeled temperature and the actual temperature was 5.2115ºC. Considering that testing is typically on the order of 240ºC, a 5ºC differential was considered more than acceptable.

3.3 Summary

Chapter three outlined PCATamp, an ampacity model developed for use at the PCAT test facility. Utilizing the large body of previous research in calculating the ampacity of an overhead conductor a model was created that could calculate the heat exchange parameters affecting the line. Specifically, models were created to handle the heat generated by conduction, the heat absorbed by solar radiation, the heat lost to

convection, and the heat lost to radiation. After the ampacity model was developed it was tested and found to perform well over a wide range of historic data.

Chapter 4

PCATAMP

The purpose of PCATamp was to provide a better control method for the PCAT facility for regulating the temperature of a test conductor. Managing the effects of the weather on an outdoor facility has proven to be quite difficult. Previous attempts to control the temperature of a conductor consisted of attempts to maintain a specific impedance on the line. Wind and other factors, however, resulted in major fluctuations in the temperature on the line while using this control scheme. The goal of PCATamp was to account for these exogeneoss conditions in the hope of creating a tighter control. In order to accomplish this task, the first item that was undertaken was the development of an ampacity model that could accurately describe the temperature on the line based on the real time measurement of its voltage, current, and multiple weather conditions.

4.1 Matlab Test Version

The first part of PCATamp was the development of the mathematical ampacity model which was built in Matlab. Utilizing the body of work on ampacity models discussed in the previous chapter, a program was written to solve for the temperature of a test conductor. This program was then systematically run through some 247,680 archived data points calculating the modeled temperature of one of the lines every minute for a four month period. The final result of this test version of the code had an average error of +/- 5° C from the actual recorded line temperature.

Of note in the Matlab version of PCATamp are some of the decisions for averaging that were made early on. First, the PCAT test facility is not only outdoor, but quite long. These attributes lead to differences in the measured temperatures along the line. In fact differences between thermocouples measuring line temperature have varied up to 15ºC due to varying wind conditions along the line. In light of these variations it was decided that an average of 10 thermocouples evenly spaced along the line would be recorded. It was determined that this average did the best job of describing the temperature of the line overall and would be the most logical temperature to attempt to match with the ampacity model. Second, measuring the speed and direction of the wind tends to be a particularly noisy business. In order to smooth out the wind data, averages were taken. The length of time averaged was reduced as small as possible to attempt to ensure that while noise was reduced, wind gusts were not smoothed out. This is important because gusts of wind can affect the temperature of the line by several degrees.

4.2 Visual Basic Implementation

After the ampacity model was built and verified in Matlab, it was decided that it was worth attempting to integrate the model into a control for the PCAT facility. All of the existing control and monitoring software had been written in Microsoft Visual Basic 6.0, so it was decided that the easiest way to implement the model would be to simply rewrite it in Visual Basic as well.

4.2.1 Implementing an ODE

The first problem that was identified with writing a Visual Basic implementation was the loss of Matlab's Ordinary Differential Equation solvers, specifically ODE45. ODE45 was used to calculate the transient heat balance equation in the Matlab version. Searching the internet quickly proved that not many people had previously considered using Visual Basic to solve a complex differential equation. In fact no existing Visual Basic code could be found that was capable of solving the ampacity model's transient equation. Previous authors and indeed Matlab's own ODE solvers used a Runge-Kutta-Fehlberg algorithm, so it was decided to simply attempt to implement such an algorithm in Visual Basic. Several versions of the Runge-Kutta-Fehlberg method exist written in Fortran and C. It was decided that the best version to implement was probably the very robust and straight forward algorithm RKF45 initially developed by Watts and Shampine [11]. This algorithm uses a static stepsize Runge-Kutta-Fehlberg 4-5th order method. While RKF45 is slightly less sophisticated than Matlab's dynamic step size version, it could be implemented in Visual Basic and proved capable of solving the transient equation within several decimal places of the solutions calculated by Matlab. The Visual Basic version of RKF45 can be referenced in Appendix I.

4.2.2 Implementing the Ampacity Model

After the problem of solving the transient equation had been solved, the rest of conversion of the ampacity model was fairly straightforward. The ampacity model implemented in Visual Basic is a nearly direct translation of the model described in Chapter 3 of this thesis, and can be seen in Appendix II. The only general issues that

were encountered while writing the Visual Basic version of the ampacity model was the lack of many advanced math functions in Visual Basic's IDE. A module of necessary math functions had to be written and is included in this thesis in Appendix III.

Some new code had to be created for calculating the azimuth and elevation of the sun. The Matlab version of PCATamp previously used a section of code that implemented I. Reda and A. Andreas's algorithm developed for the National Renewable Energy Laboratory (NREL) [10]. Instead of converting this large section of code over to Visual Basic, a Visual Basic version of a similar algorithm developed for the National Oceanic and Atmospheric Administration (NOAA) and translated to Visual Basic by G. Pelletier, was used. Both of these algorithms were written to calculate the position of the sun with extremely high accuracy and produce identical results. During debugging and optimization of PCATamp, the author realized how heavy a load calculating the location of the sun, in near real time, actually is. In fact some 90% of execution time would normally be spent inside of the sun module. In order to speed up PCATamp some modifications of the original sun code were made, at the expense of readability, in an effort to minimize process overhead. The code used for calculating the azimuth and elevation of the sun is located in Appendix IV.

The only safety nets placed inside the ampacity model part of the PCATamp program is a program stop if the conductor temperature is below the ambient temperature. This condition will cause illegal mathematical operations to occur and must therefore be prevented. No other program stops have been coded due to the general resistance of the algorithm to corruption, and its ability to handle very poor data. PCATamp was run continuously for several days with random input data to insure that odd data patterns

would not cause the program to fail. Insuring that PCATamp would not crash erratically or get caught in situations of non-convergence was important because of the potential for serious equipment damage.

Even though PCATamp is quite resilient to crashing, however, it can not be run under extreme weather conditions. The ampacity model used is not capable of accounting for the cooling effects of rain, or ice on the line. Because the effects of these weather events can not be accounted for the ampacity model can not be counted on to provide a reliable answer. Therefore operators must be aware of the prevailing weather conditions to know if suitable conditions exist for running PCATamp.

4.3 Control Scheme

The control created with PCATamp begins by taking data from the online instrumentation, including current, voltage, and weather conditions, and passing it to function PredictTempC, as seen in Appendix II. This function returns the expected temperature of the line at some time in the future, based on the present weather conditions and maintaining the present current value. The expected temperature is then compared against a desired temperature for the line. If the expected temperature varies more than two degrees from the desired temperature, the current value in memory will be raised or lowered two amps and PredictTempC will be rerun. This process will continue until the expected and desired temperatures match. At that point, if the current value in memory has changed from the current value being commanded to the line, the new

current value will be commanded to the system. PCATamp will then wait one minute and start over. This process is illustrated in Figure 4.1 on the next page.

For safety, PCATamp is not allowed to command large changes in the current. The control system will only allow the current change to be less than 300 amps per minute. This was done to prevent large steps in current that could cause damage to the equipment on site. If PCATamp recommends a change of more than 300 amps the system will change a maximum of 300 amps per minute, until the recommended change has been met. A check is also done to ensure that the actual temperature is not too far from the desired temperature, and if it is, the program shuts down. This could potentially happen if some un-modeled weather event is present on the line, such as rain. In this case, the line would be cooling far more than the ampacity model would be aware of causing a disparity in temperatures.

4.4 Summary

Chapter four developed the Visual Basic control in use at PCAT for maintaining test conductors at a specific temperature. A number of challenges were discussed in converting the ampacity model from Matlab to Visual Basic. These challenges primarily involved the loss of the ordinary differential equation solvers built into Matlab. Next, the control scheme used to maintain a test conductor at a specific temperature is discussed. The method used makes use of the ampacity model's ability to predict the future temperature of the line and then adjust the current level to drive the conductor temperature towards its target.

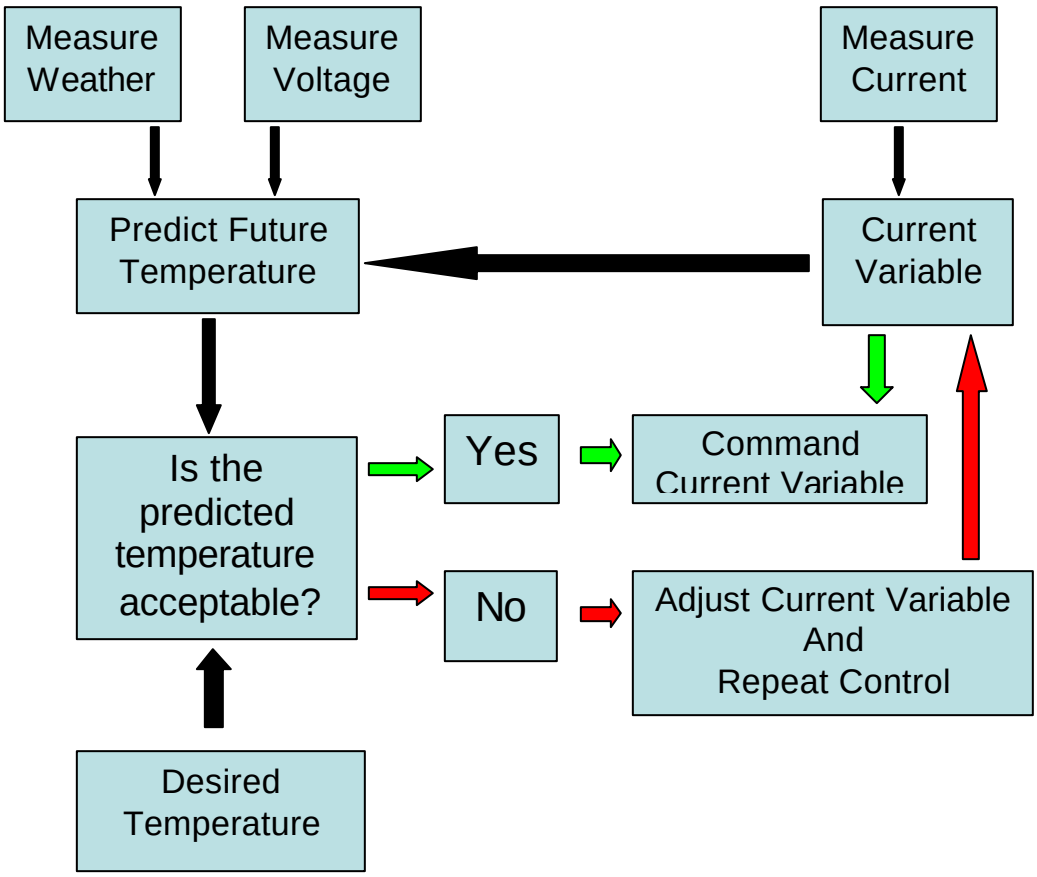

Figure 4.1. PCATamp Control Algorithm

Chapter 5

EXPERIMENTAL RESULTS

In the previous chapters, an ampacity model and a subsequent control based on that model were introduced. This chapter will take the developed control and discuss its performance online at the test facility. A number of factors were found to have significant effects on the performance of the control while tuning it online. Eventually it was discovered that variable sensitivity could be greatly altered by modifying the time constant that the control was using. This realization would result in a significant improvement in the performance of the control.

5.1 Apparatus Setup

The conductor being tested is 795 kcmil 3M ACCR. This composite cable is capable of temperatures up to 240°C. The conductor has the following specifications that apply to the control.

- Number of Aluminum Strands = 26
- Number of Composite Strands = 19
- Diameter of Aluminum Wire = 0.175 in
- Diameter of Composite Wire = 0.082 in
- Diameter of Whole Conductor = 1.11 in
- Heat Capacity of Aluminum = 324 J/ft°C
- Heat Capacity of Composite = 22 J/ft°C

It should be noted that this is not the conductor used to build and tune the initial ampacity model. The cable used for creating the model was a 1272 kcmil 3M ACCR conductor. The primary difference was that the previous conductor was significantly larger, with increased surface area, and increased heat capacities. However, the original model was setup with variables in place that could be modified accordingly per the conductor that was being used. Setting the control up for online use with a different conductor posed some uncertainty because while the original model was setup to be capable of potentially handling any conductor, the effects of changing the conductor were at the time unknown. In the end, however, it was seen that the model behaved as it was designed.

Measuring the conductor temperature is accomplished by using the average of 10 thermocouples along the line. These thermocouples are evenly spaced approximately 150 feet apart from each other along both sides of the conductor loop. The thermocouples used are a combination of ones attached to the surface of the conductor and some that are imbedded next to the core of the conductor. All thermocouple readings are performed by ICP-DAS I-7018 thermocouples, using type T thermocouples. These units were all calibrated by hand using a high accuracy thermocouple calibration device to within +/- 0.1°C. A recent lightening strike that damaged several of the thermocouples, resulting in their replacement, has insured that the thermocouples now in place should be operating to the highest possible degree of accuracy.

Wind speed and direction is measured by multiple devices in both two-dimensions and three-dimensions. For use with the control the data passed by the two dimensional measurement apparatus is used.

Ambient temperature is measured by the same type of thermocouples that are used to measure conductor temperature. Ambient temperature is measured by thermocouples high on the tower placed out in the air.

Solar radiation is measured by a solar radiation device capable of measuring watts per meter squared of solar radiation. Because this device is used in calculating the effect of solar radiation on the line the effects of cloud shading are taken into account as they will be read as reduced exposure to solar radiation.

5.2 Program Setup

Program setup was relatively easy. The main modification made to the original PCAT control software was the addition of a box, Figure 5.1, containing the available variables that could be modified for tuning the control. These variables included the following.

Time Stop: Time Stop was the variable used to control how far into the future the ampacity model would look. The variable effectively modified how many minutes PCATamp would model. For example, a Time Stop value of five indicated that PCATamp should output the expected temperature, based on present weather conditions, five minutes into the future. Typical values: 1 – 20 (integer)

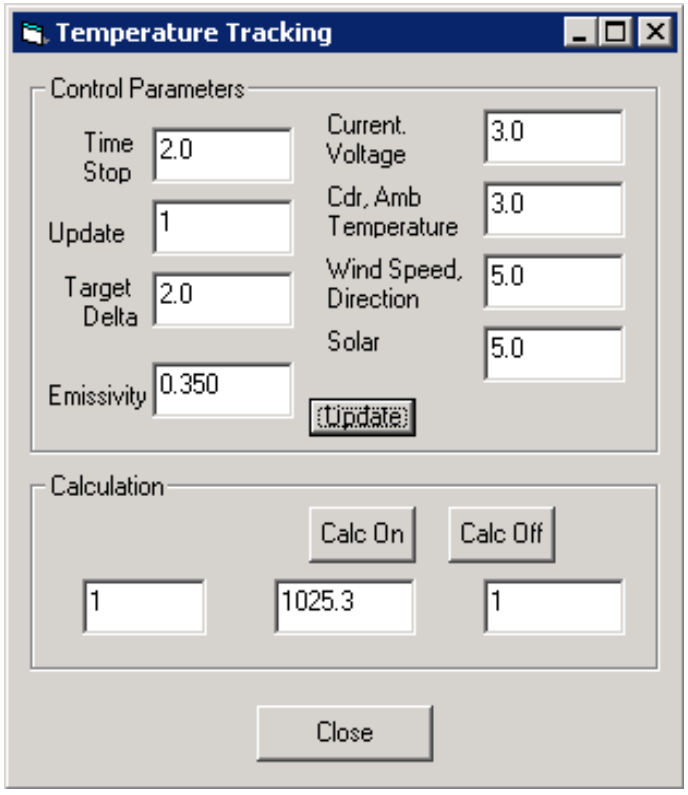

Figure 5.1. Temperature Tracking Box

Update: This was the variable used to control how often a new current value could be commanded to the system. The original software for PCAT operates on a watchdog clock of one minute that precisely dictates when to collect data from the measurement devices and when to command the system to a different current value. The variable Update makes it possible to change how often the ampacity model could update the current on the system. For example, an Update value of one indicated that every minute the latest output of the control would be commanded to the system. An Update value of five would indicate that the control software could only change the commanded current once every five minutes. This variable was introduced because of some initial concern that allowing the control to command the current every minute might lead to excessive switching for the power supply. Typical values: 1 – 5 (integer)

Target Delta: This variable controlled how close to the target temperature the control had to be. The PCATamp control returns a temperature based on present weather conditions and the current on the line. Modifying the target delta changes how near to that target temperature the control has to be to return an acceptable value for commanding the current. For example, a target delta of five indicates that PCATamp has to provide a current value back to the control that results in an expected temperature that is +/-5°C of the temperature the control is attempting to hold the line at. Typical values: 1 - 5 (float)

Emissivity: Emissivity generally describes the condition of the air next to the conductor. This value is modified based on attributes such as, is the line being tested on

a mountain with clean air, or is it running through a major city with significant levels of pollution in the air. The value for emmisivity was tuned during the development of the model and should generally not require modification. However, should the model drift off of the target temperatures during seasonal changes this variable can be used to account for such movement. Typical values: 0.23 - 0.91 (float)

Current / Voltage: This variable is used to control how many current and voltage measurements are averaged together for use with the control. A value of one here would indicate that the control is using real-time current and voltage data from line measurements. A value of five here would indicate that a moving average of five minutes of current values and voltage values would be passed to the control. The variable is present to account for the power supply's inability to immediately change to specific commanded currents. For example, if a current of 1000 Amps is present online and a current of 1200 Amps is commanded to the line the power supply might first change to 1290 Amps and then 1170 Amps and then settle very close to the desired current value, all within a few seconds. This average allows this behavior to be smoothed out for the control. Typical values: 1 – 3 (integer)

Conductor / Ambient Temperature: This variable is used to control how many conductor temperature and ambient temperature values are averaged together for use with the control. A value of one here would indicate that the control is using real-time temperature data off the line. A value of five here would indicate that a moving average of five minutes of temperature data would be passed to the control. This variable is

present because the reading from the thermocouples are never constant and always float around based on the conditions present. The average is in place to smooth out these values for a more consistent input to the control. Typical values: 1 – 3 (integer)

Wind Speed / Wind Direction: This variable is used to control how many wind speed and wind direction measurements are averaged together for use with the control. This variable is quite important because of the very noisy nature of both the wind speed and wind direction measurements. Because wind speed and wind direction are so noisy, an average of their values has to be performed in order to have any indication of what the overall wind pattern present is. Although the wind speed and wind direction variable is noisy, it is the most significant weather variable affecting the ampacity model. This makes finding the average value that provides enough real-time data but does not provide erratic data both difficult and very important. For example, a value of 45 here would indicate that a moving average of 45 minutes of wind speed and wind direction data would be passed to the control. Typical values: 5 – 45 (integer)

Solar: This variable controls how many solar radiation values are averaged together before being provided to the control. It operates very similarly to the variables mentioned above. For example, a value for five here would indicate that a five minute moving average of the solar radiation data is provided to the control. This variable is present to smooth any noise that may be present in the measurement. Typical values: 1 – 5 (integer)

Target Temperature: This is the target variable of the control. This variable sets what temperature the control should attempt to keep the temperature at. For example, a value of 190 would indicate that PCATamp should attempt to elevate and maintain the temperature of the conductor at 190°C. Typical values: 100 - 240 (integer)

5.3 Control Tuning

Tuning the control turned out somewhat unexpectedly. One of the initial configurations tested was as follows.

- Time Stop = 10
- Update = 5
- Target Delta = 5
- Emissivity = 0.35
- Current / Voltage = 3
- Conductor / Ambient Temperature = 3
- Wind Speed / Direction = 45
- Solar = 5
- Target Temp = 100

Figures 5.2 through 5.8 display graphs associated with this test. This test was run from 6:57pm on October 26, 2005 to 3:10pm on October 27, 2005. As can be seen from Figure 5.2, the conductor temperature remained slightly below the target of 100°C during the night, but during the day the control kept the temperature some 20°C above the target. It was discovered that these variations were caused by a number of reasons.

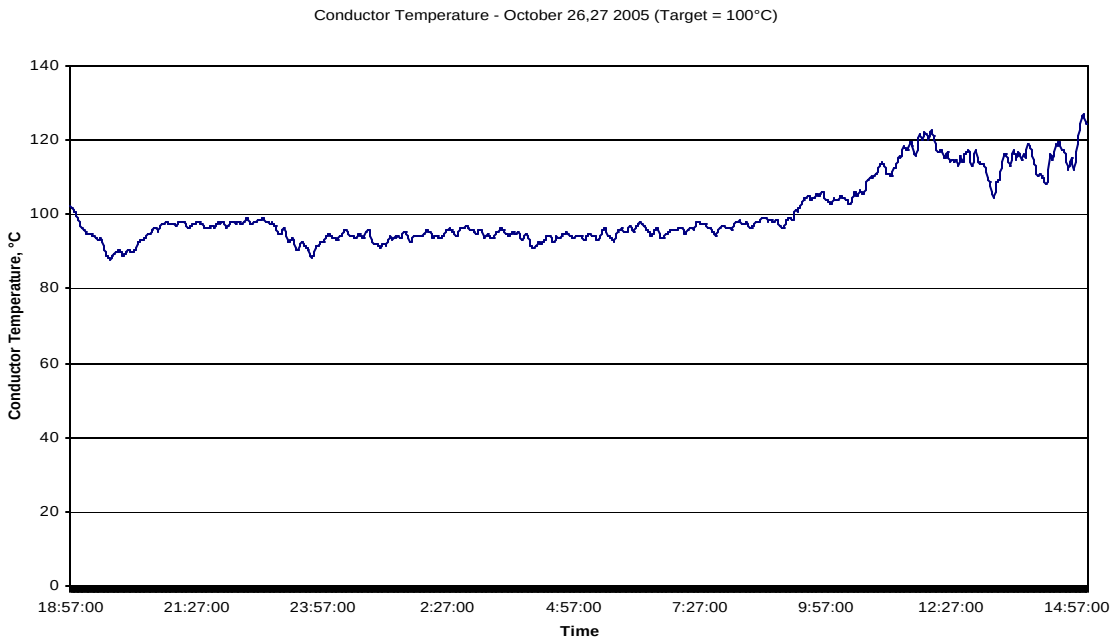

Figure 5.2. Conductor Temperature – 10/26/2005 (Constant Temperature Test)

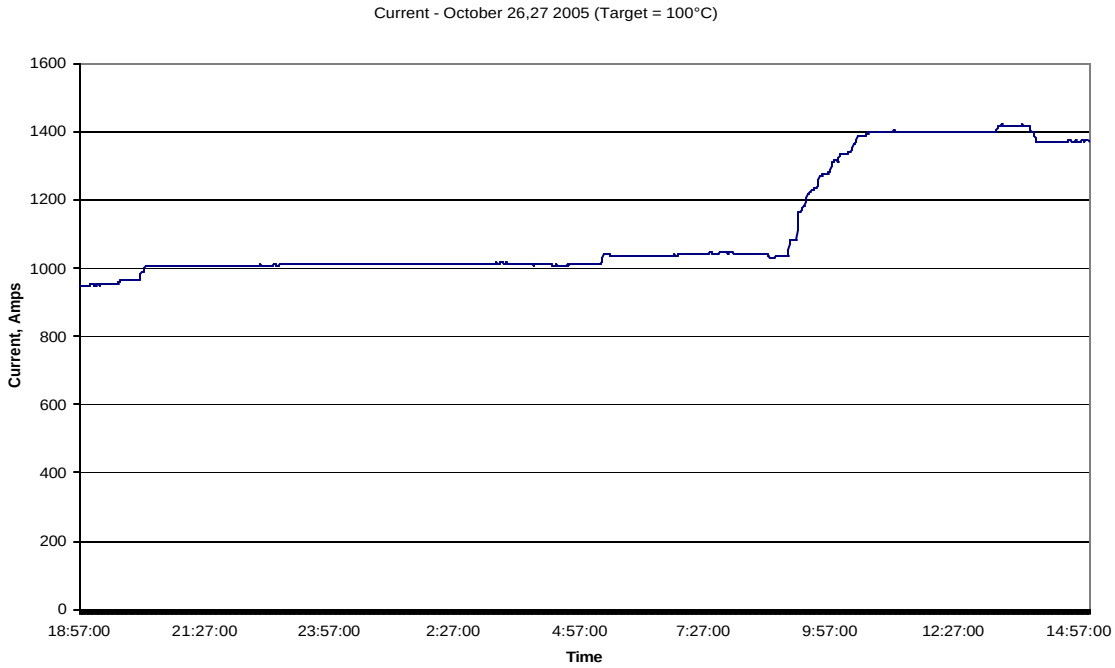

Figure 5.3. Current – 10/26/2005 (Constant Temperature Test)

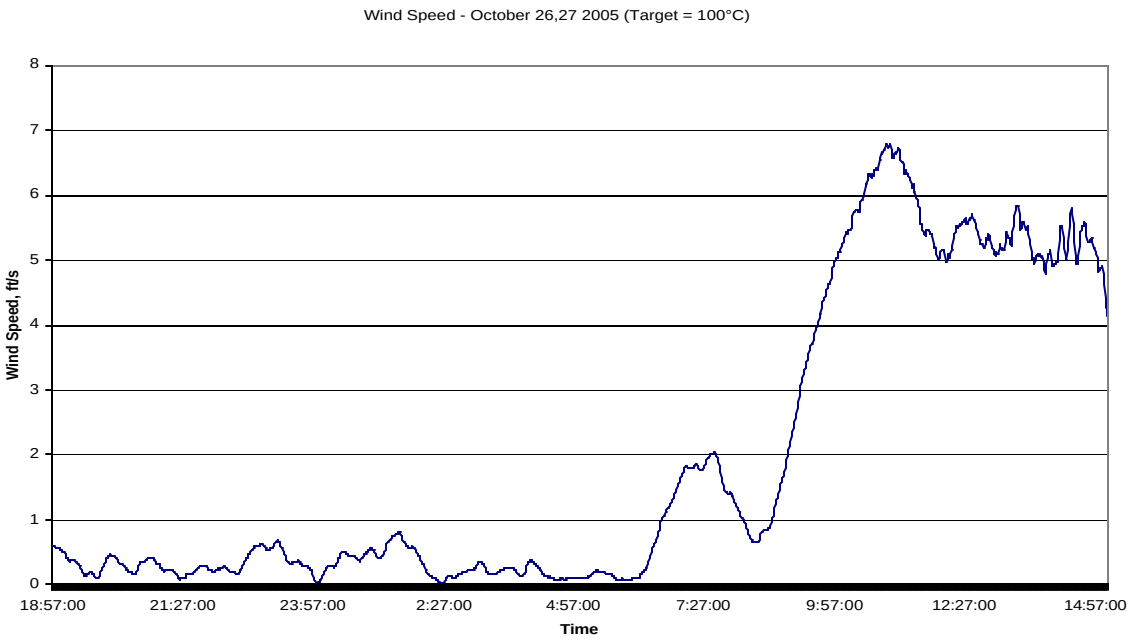

Figure 5.4. Wind Speed – 10/26/2005 (Constant Temperature Test)

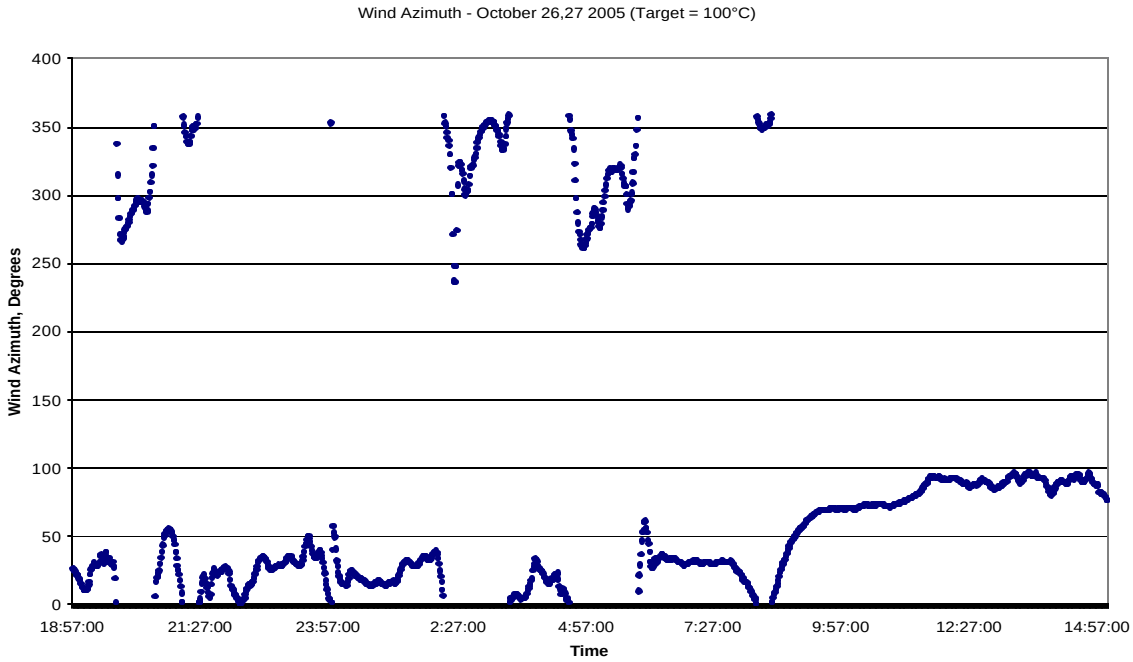

Figure 5.5. Wind Azimuth – 10/26/2005 (Constant Temperature Test)

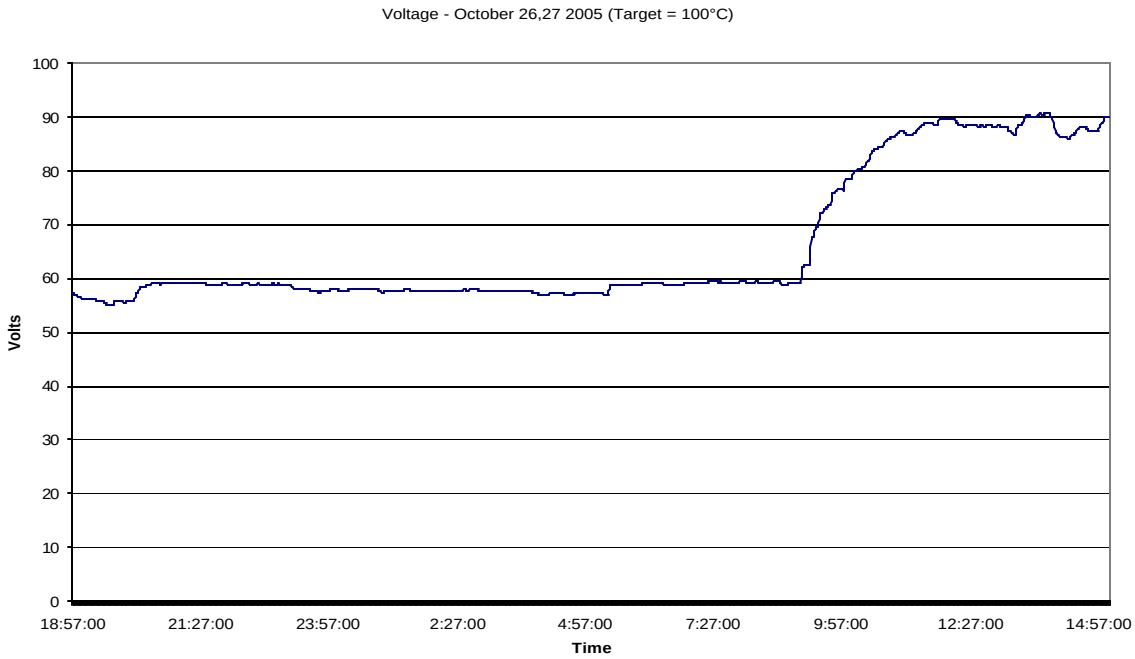

Figure 5.6. Voltage – 10/26/2005 (Constant Temperature Test)

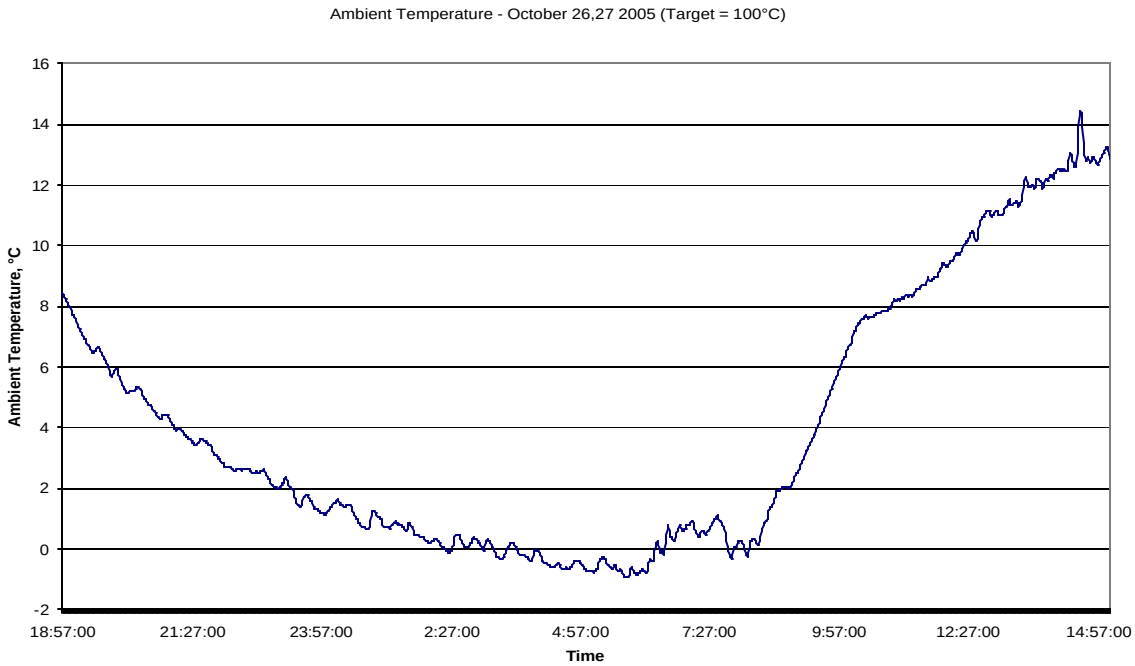

Figure 5.7. Ambient Temperature – 10/26/2005 (Constant Temperature Test)

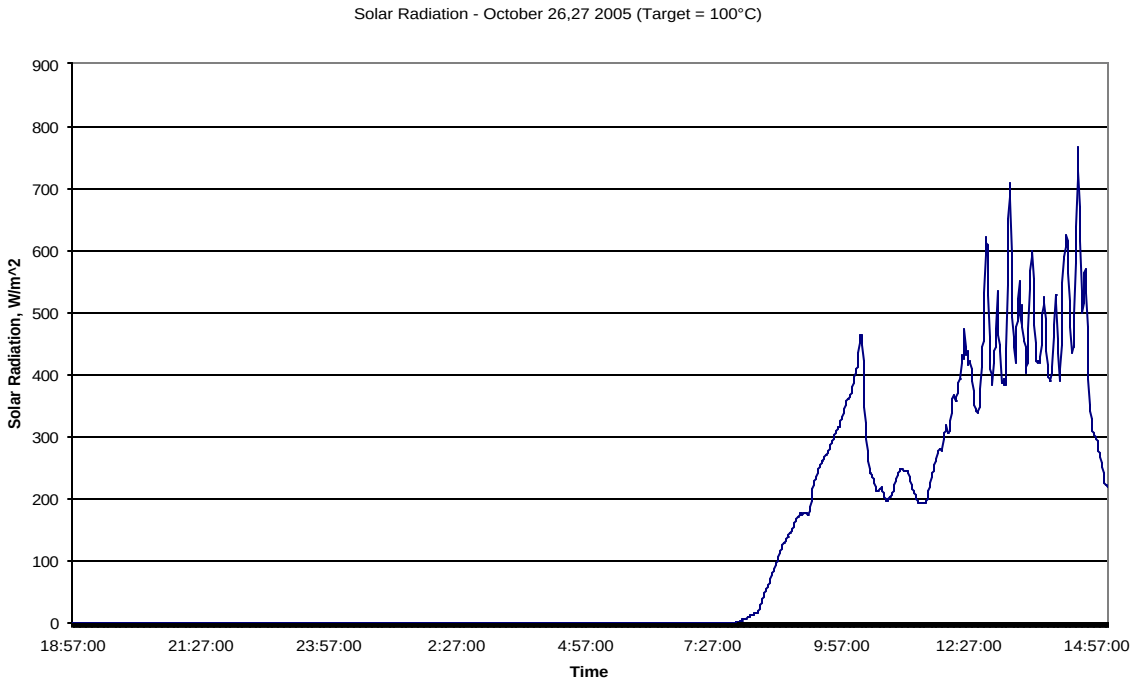

Figure 5.8. Solar Radiation – 10/26/2005 (Constant Temperature Test)

The temperature is consistently 5°C below the target during the night because the delta was set equal to five. The result of this was that the model was consistently providing currents associated with 95°C instead of 100°C.

Another problem was that the current was only allowed to be updated every five minutes, Update = 5. The result of this was that if the control made a bad decision based on noise in the weather measurements, mainly the wind measurements, it had to live with that decision for five minutes.

The other realization was that the control was consistently overcompensating for the cooling effects of the wind. This was the reason behind the elevated temperatures during the day. Two methods of thinking went into how to address this issue. The first

idea was to further increase the amount of wind speed averaged, effectively lowering the wind speed values. This idea was discounted, however, because 45 minutes of data was already being averaged, and it was not realistic to believe further increasing the average could have significant effect. The next idea was that the amount of time the model looked into the future was affecting how it compensated for the wind. This idea prompted a closer look at the model's behavior for various time constants, discussed further in the next section. The result of this investigation, however, was the conclusion that lengthening the time constant of the control not only reduced the overall gain, but increased the current necessary to hold a temperature, leading to overcompensation during the day.

5.5 Time Constant

The IEEE Std 738-1993 includes a method of calculating the time constant associated with a step change in conductor temperature or current [5]. During the investigation behind the reason for current overcompensation during the day, it was realized that the Time Stop variable was effectively modifying the time constant of the ampacity model. Time Stop controls the look ahead of the ampacity model, but also controls the length of time the model will use to complete a step. As an example consider the following table of data. Based on these conditions.

- Wind Speed = 4.1 feet per second
- Wind Direction = 95 degrees
- Ambient Temperature = 14 °C

- Current = 1050 amps
- Voltage = 90 volts
- Initial Conductor Temperature = 100 °C
- Solar Radiation = 700 watts per meters squared
- Desired Temperature = 110 °C

Table 5.1 illustrates how when the model is put in a position where it needs to elevate the temperature 10°C it make significantly different decisions based on how much time it has. If it has to correct the temperature in one minute the control recommends commanding 3595 amps, but if the control has 20 minutes to reach that temperature it recommends using less than half that current, or 1527 amps. To the ampacity model, Time Stop is actually imposing a time constant.

Going back to the simple program provided by the IEEE standard provided an easy way of seeing the model's reaction to different time constants. The results of a

Table 5.1. Current Sensitivity to Time Stop

Time Stop, minutes	Current, amps
1	3595
5	1844
10	1629
20	1527

temperature drop from some temperature back to 100°C is given in Figure 5.9. The curves clearly indicate that there is a significant change in gain associated with the system by moving from a time constant of one to a time constant of 30. In order to drop the temperature from 120°C to 100°C, a time constant of one would drop the current to almost zero, but a time constant of 20 would drop the current only 9 amps. What this means to the control is that a large time stop value will likely never reach the target temperature value. The reason it would not work is that it simply is not reasonable to update the system once ever 20 or 30 minutes. Weather conditions are constantly changing throughout the day and can change significantly over the course of one minute.

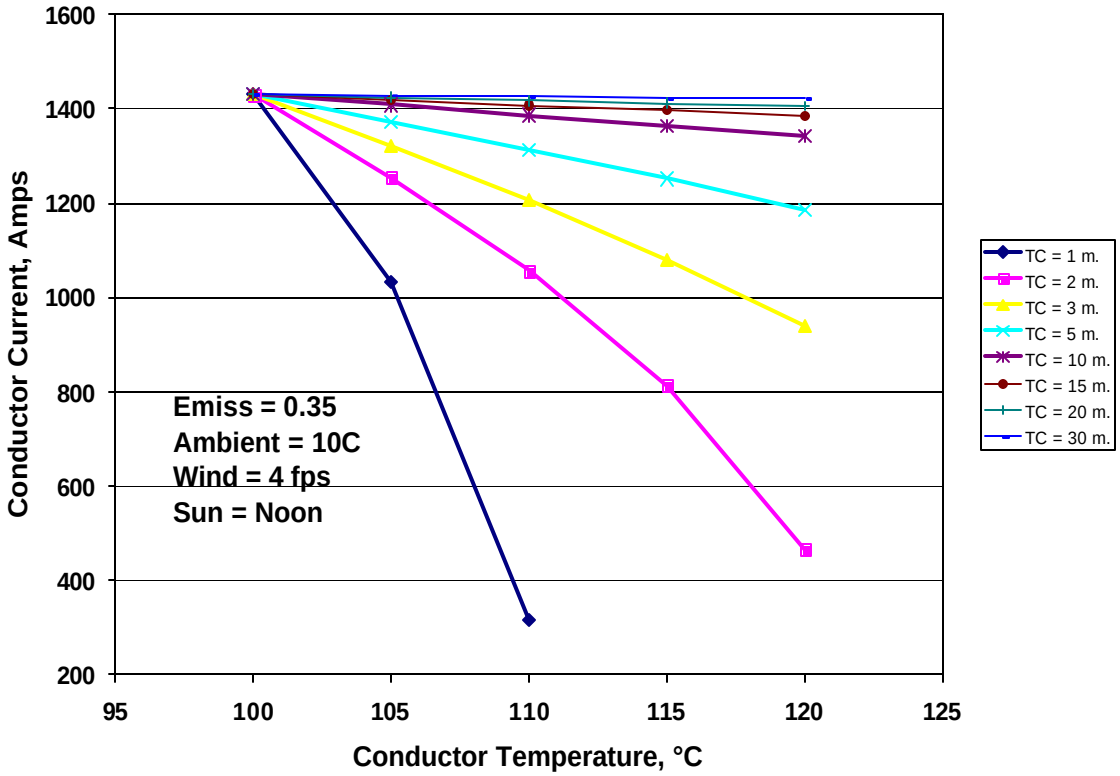

Figure 5.9. Time Constant

Expecting weather conditions to remain constant over half an hour is simply unrealistic. Attempting to update the system more often than the time constant will not work either, because the system has such low gain at that point that it will never have the opportunity to reach the target temperature.

The conclusion drawn was that it was better to minimize the predictive nature of the model and set Time Stop as low as possible. For PCATamp it was decided that Time Stop should be set to two. Based on time constant curves and tests run with an off line version of the control, it was determined that a Time Stop of one had so much gain that it could prove unstable or attempt to command current values that the PCAT power supply is not capable of reaching.

5.6 Constant Current

In order to demonstrate why such a complex method of controlling the temperature of the line is necessary, the curves related to a constant current test are shown below. These curves represent the changes in conductor temperature associated with changing weather conditions throughout the day. In these tests the current was set to a constant 1025 amps for 24 hours, from 9pm on October 30, 2005 to 9pm on October 31, 2005. Figures 5.10 to 5.16 show the conductor temperature, current, voltage and the related weather conditions for that day.

Figure 5.10 shows how the conductor temperature remains fairly constant during the calm conditions of the night when there is very little change in solar, wind, and ambient temperature. However, when the wind begins to pick up late in the morning the

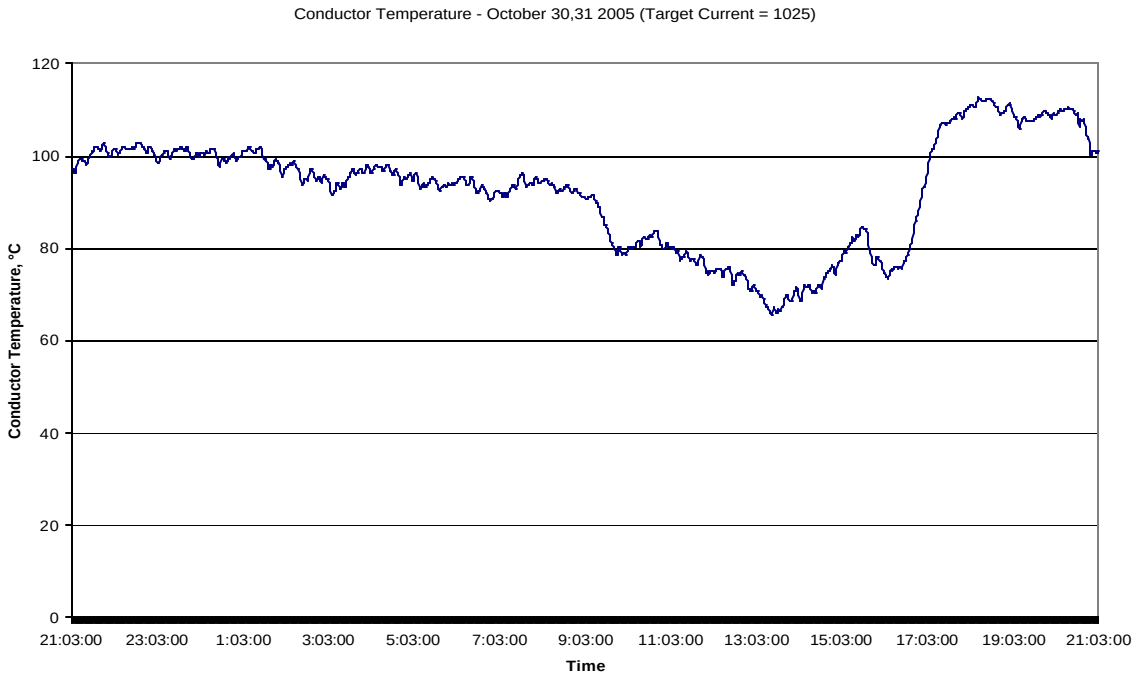

Figure 5.10. Conductor Temperature – 10/30/2005 (Constant Current Test)

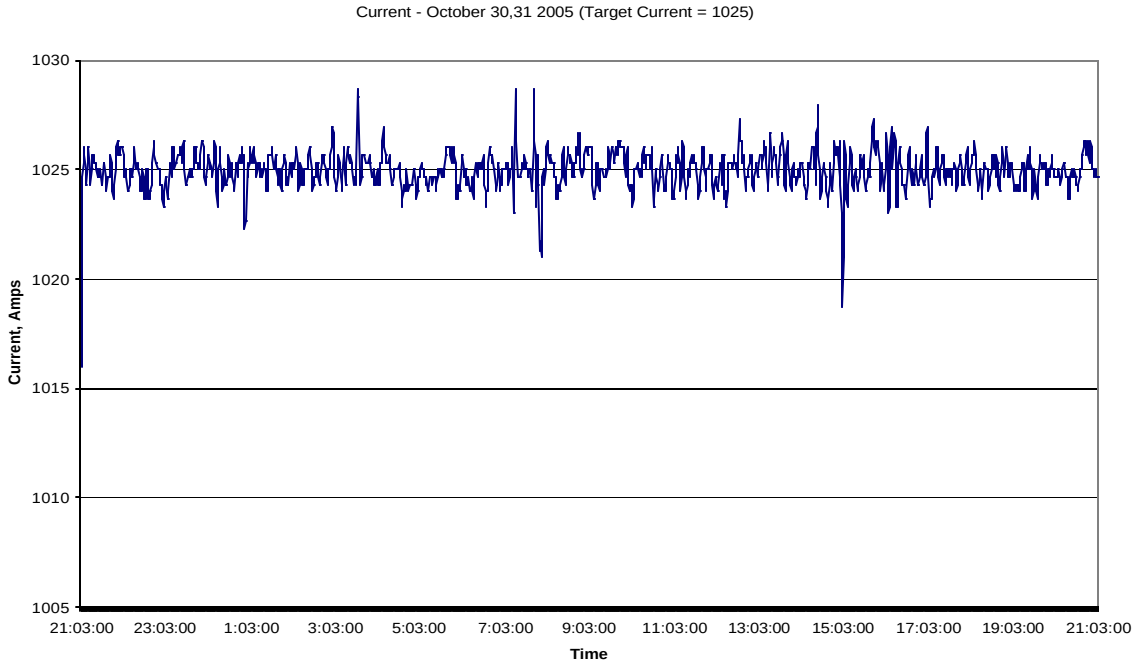

Figure 5.11. Current – 10/30/2005 (Constant Current Test)

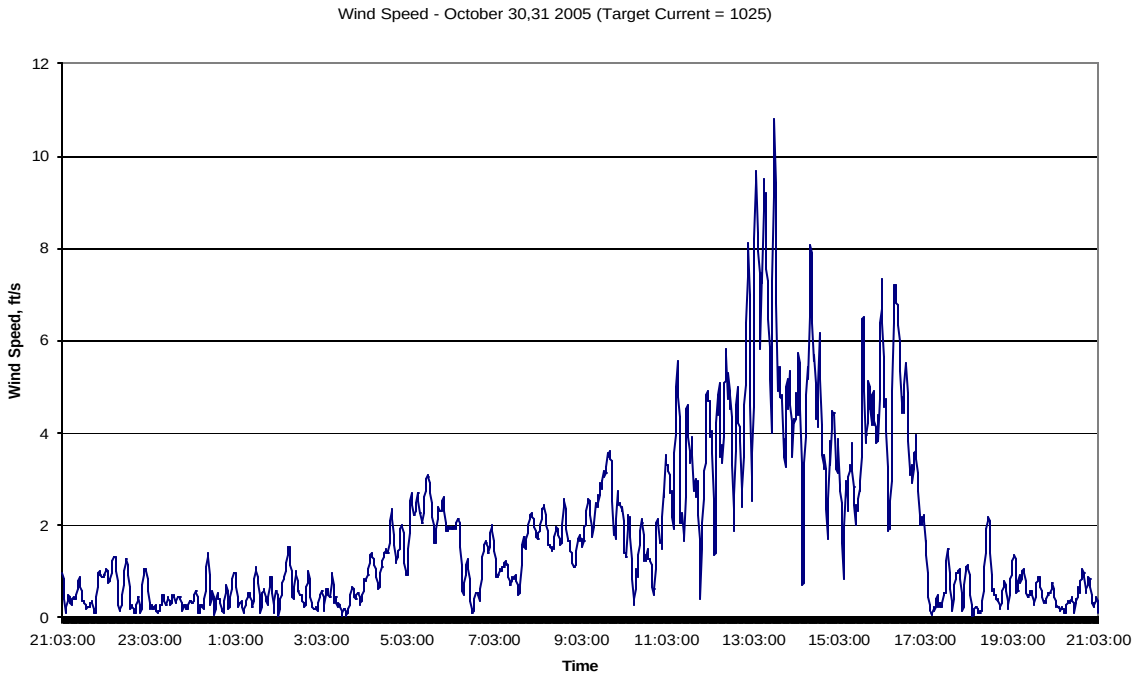

Figure 5.12. Wind Speed – 10/30/2005 (Constant Current Test)

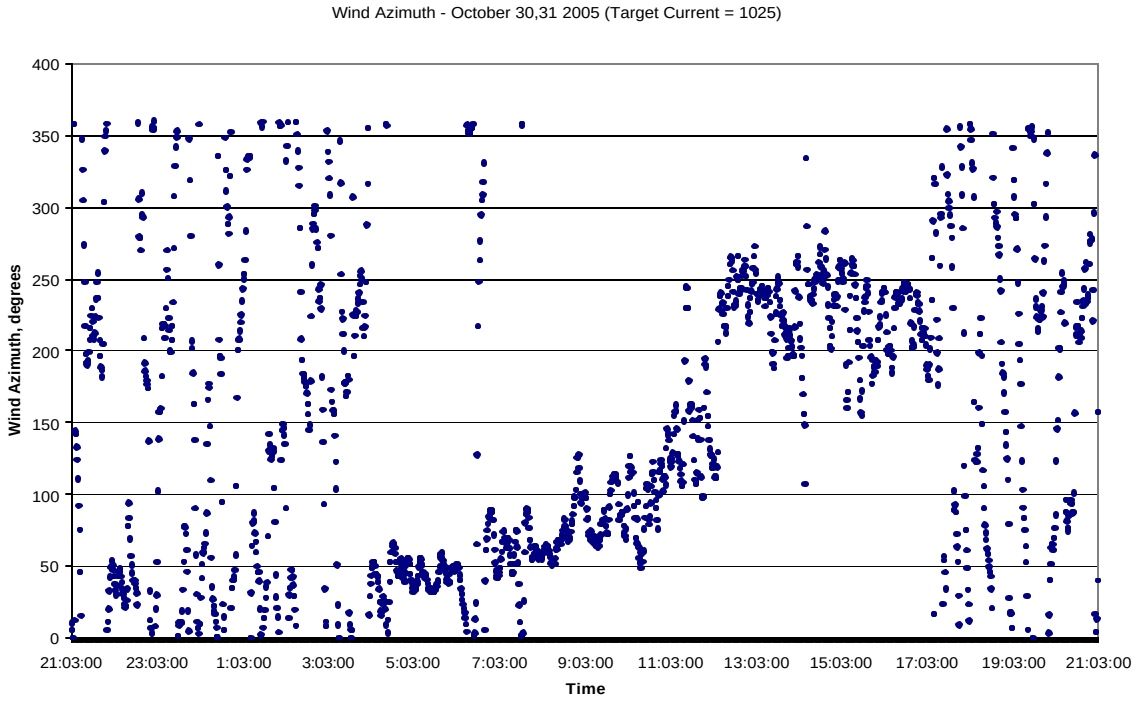

Figure 5.13. Wind Azimuth – 10/30/2005 (Constant Current Test)

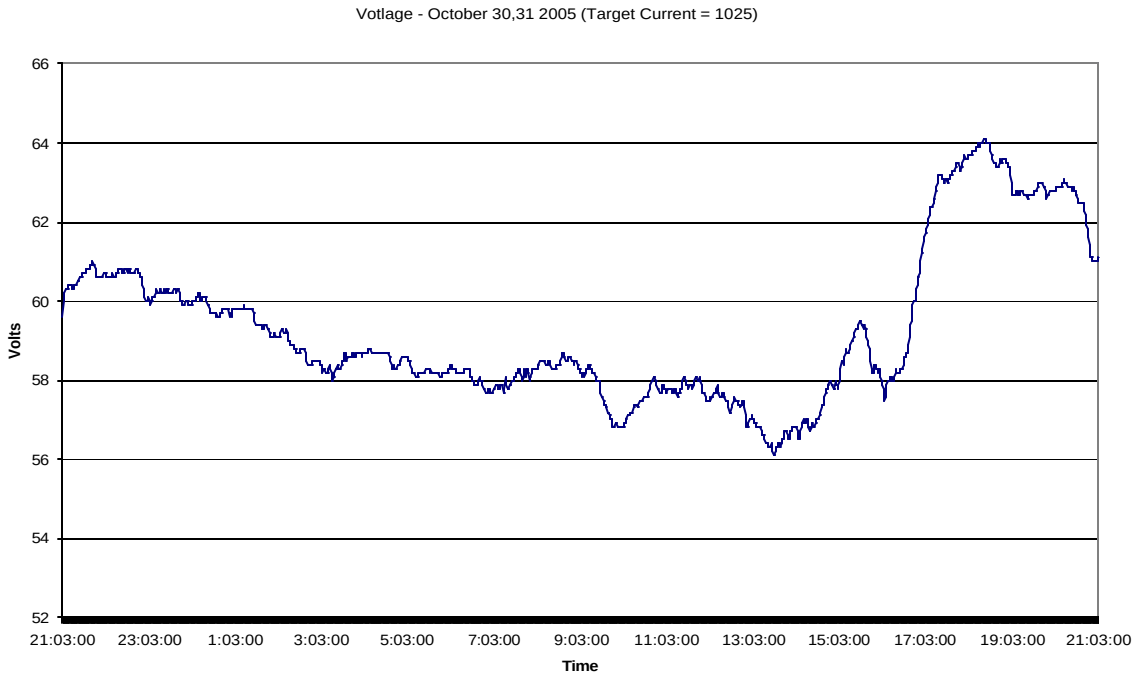

Figure 5.14. Voltage – 10/30/2005 (Constant Current Test)

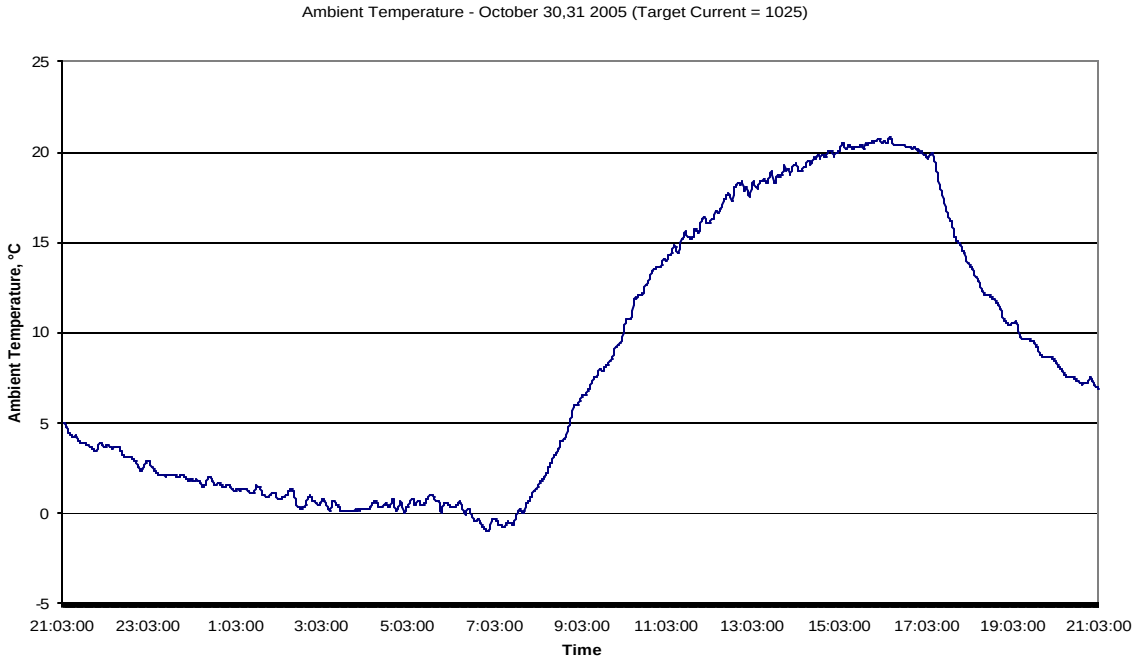

Figure 5.15. Ambient Temperature – 10/30/2005 (Constant Current Test)

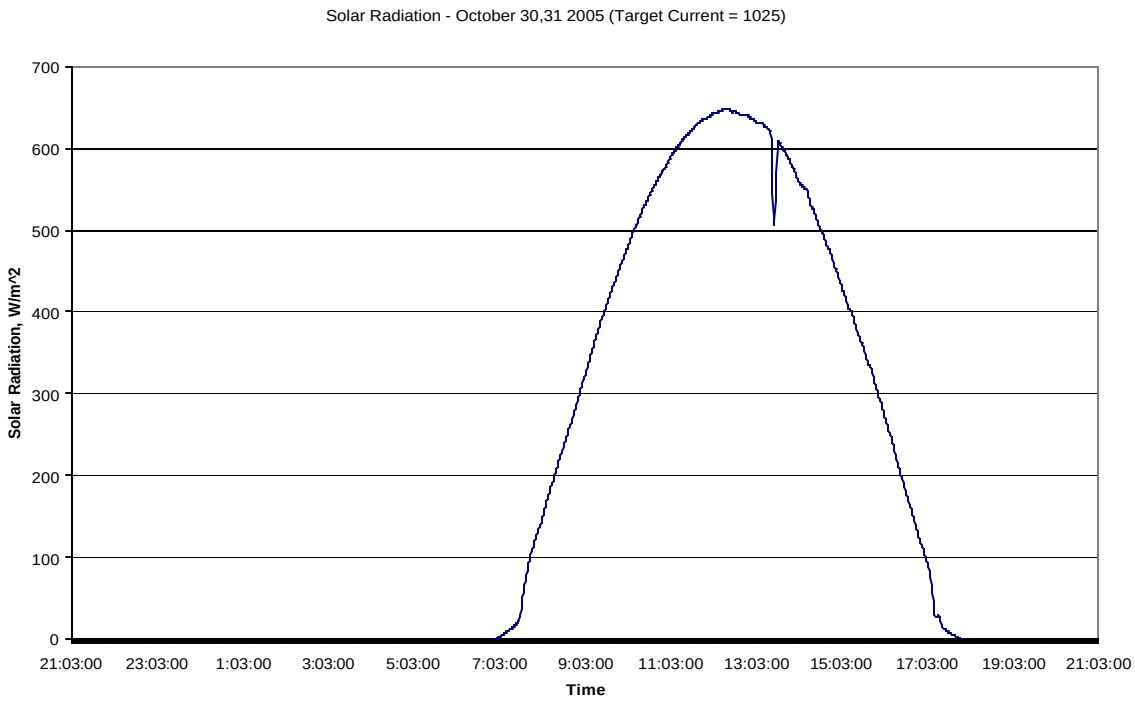

Figure 5.16. Solar Radiation – 10/30/2005 (Constant Current Test)

temperature of the line begins to vary widely, over 40°C throughout the day. Referencing the graph of wind speed, Figure 5.12, shows how changes in wind speed dramatically affect the temperature of the line. It is hoped that with PCATamp a constant temperature can be maintained in this region of the day as well.

The reader may be curious as to why Figure 5.16 looks so much smoother than the Solar Radiation curve from the previous test, Figure 5.8. What Figure 5.8 shows is the result of a cloudy day. Large amounts of cloud cover during that day seriously impeded the amount of solar radiation seen on the ground. The curve shown in Figure 5.16, however, is quite smooth because it was a clear day.

5.7 Constant Temperature

After a good deal of tuning was done on the control, it was decided that the controls performance could begin to be measured. The following test was run on October 29, 2005. Figures 5.17 to 5.23 show the results of this test.

- Time Stop = 2
- Update = 1
- Target Delta = 2
- Emissivity = 0.35
- Current / Voltage = 3
- Conductor / Ambient Temperature = 3
- Wind Speed / Direction = 15
- Solar = 5
- Target Temp = 100

This test performed much better than the previous tests done before and during tuning. As can be seen from Figure 5.16, the temperature was held around the target within +/-10°C. For the purposes of PCAT this level of accuracy is considered acceptable. The ability of the ampacity model to account for changes in wind, solar radiation levels, and ambient temperature in real time is clearly shown by the relatively small effect those changes had on the conductor temperature.

There was some concern that decreasing Time Stop to two might cause the model to become unstable if it got very far from its target temperature. Fortunately, the model

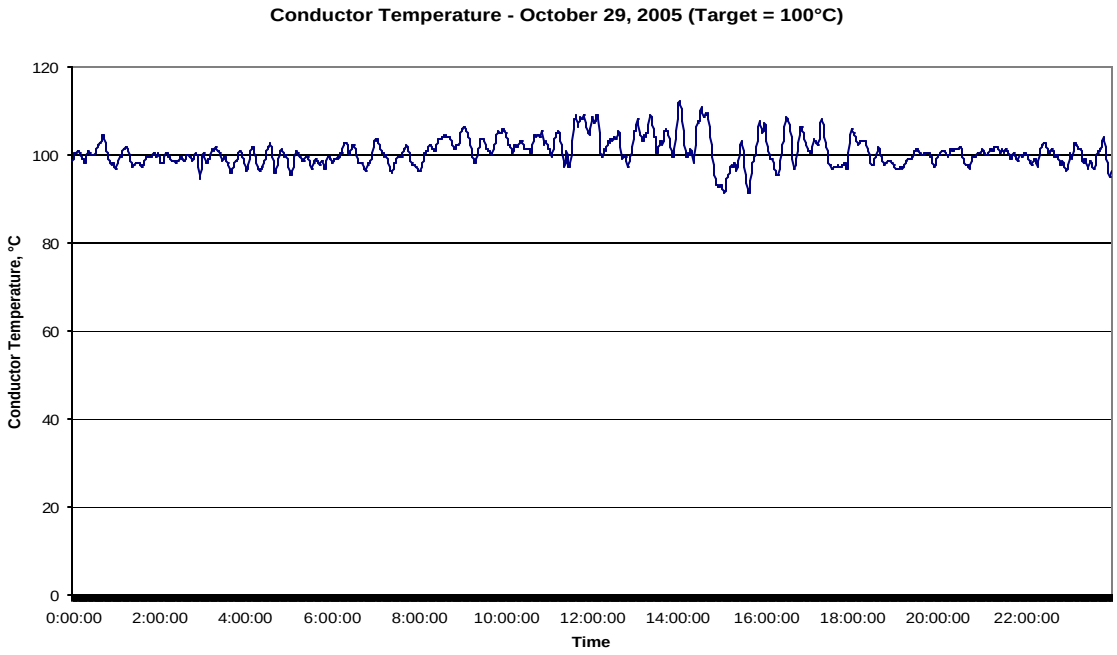

Figure 5.17. Conductor Temperature – 10/29/2005 (Constant Temperature Test)

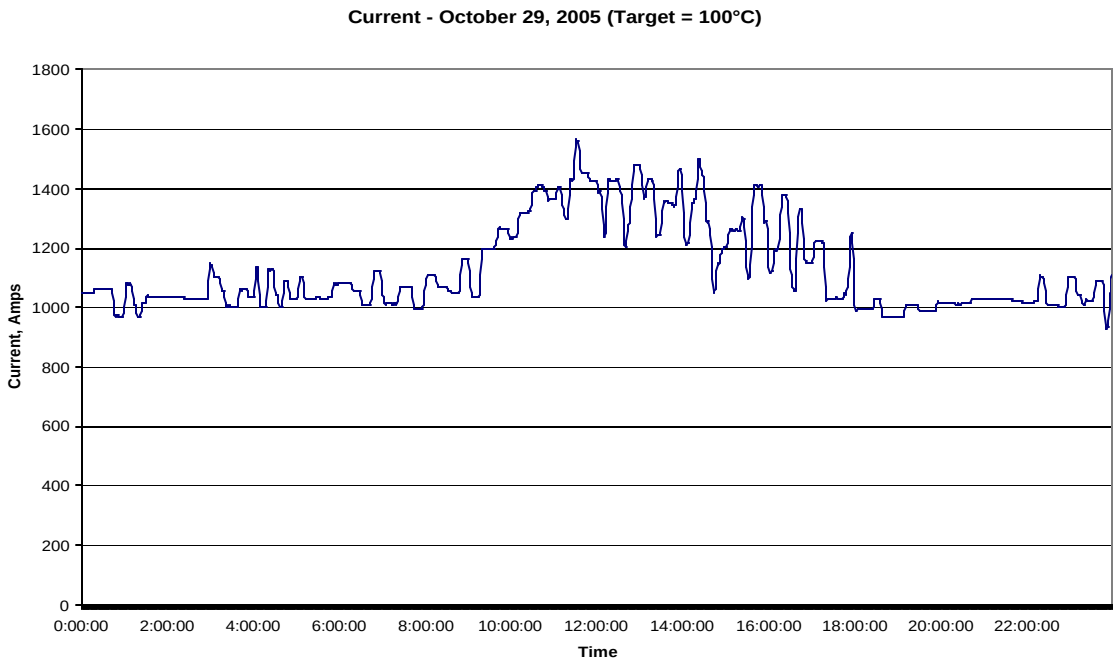

Figure 5.18. Current – 10/29/2005 (Constant Temperature Test)

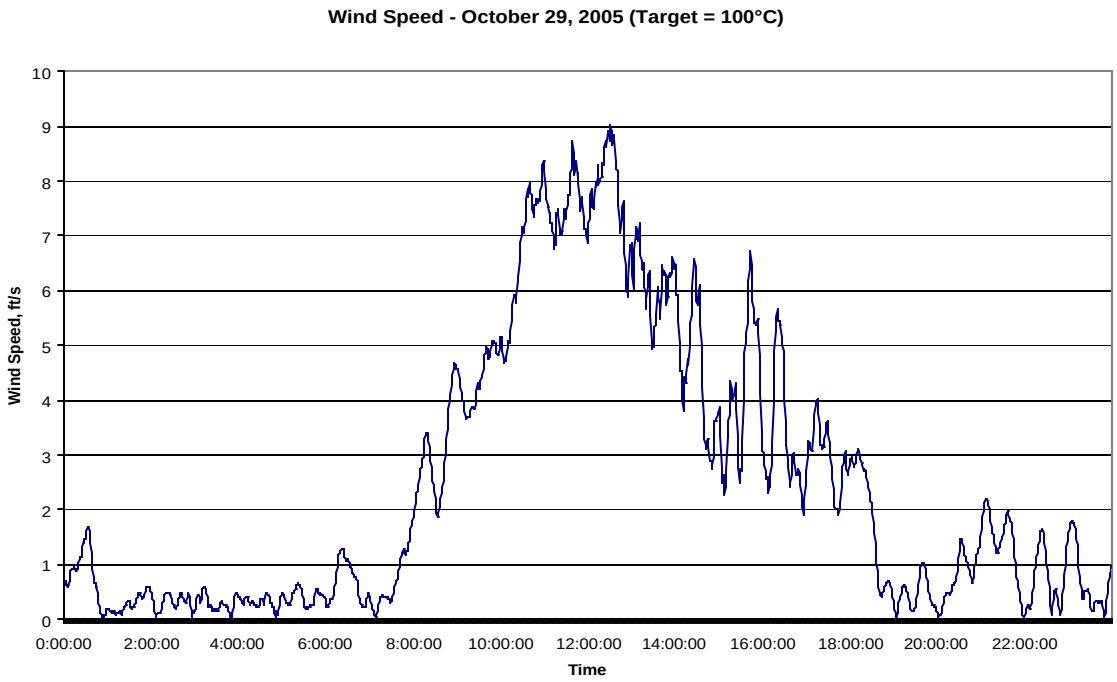

Figure 5.19. Wind Speed – 10/29/2005 (Constant Temperature Test)

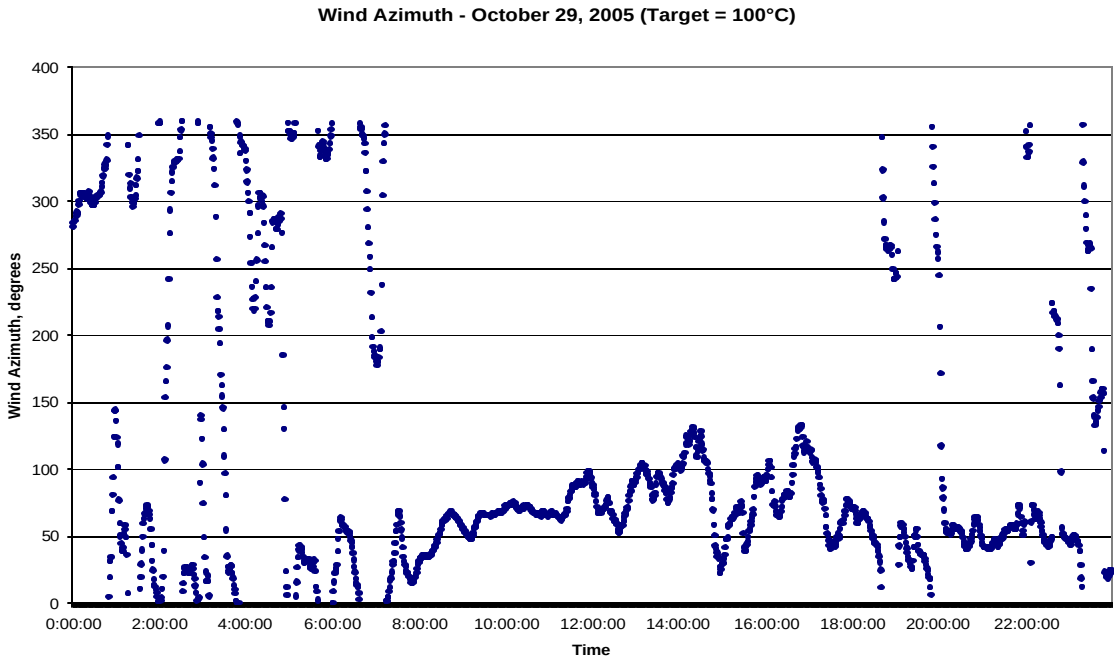

Figure 5.20. Wind Azimuth – 10/29/2005 (Constant Temperature Test)

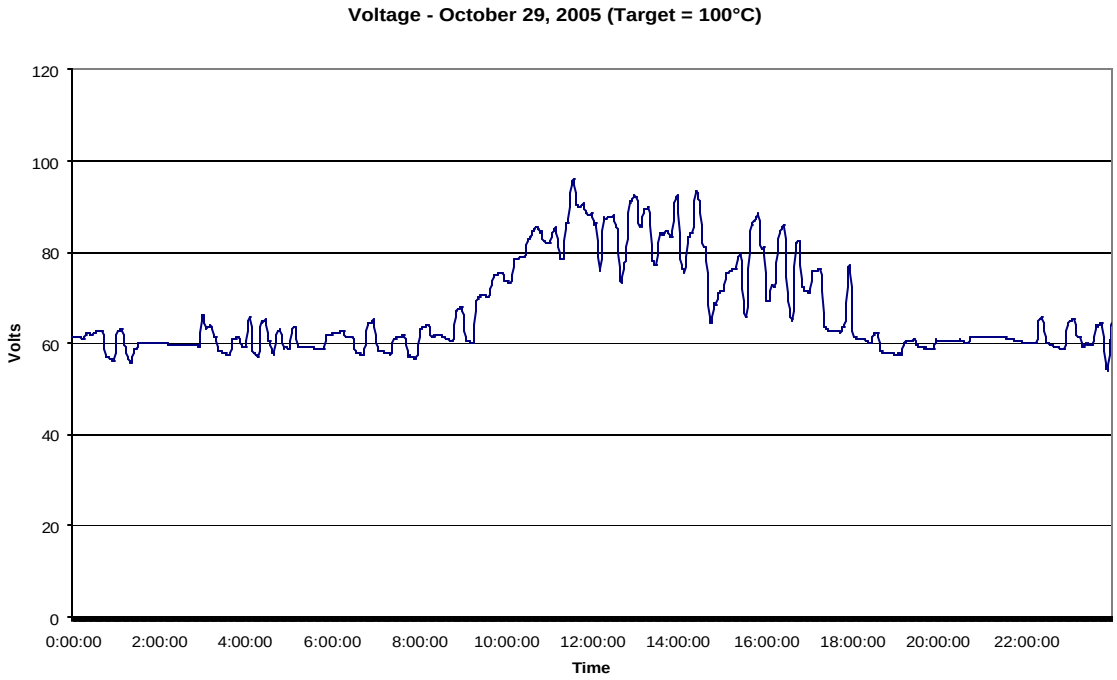

Figure 5.21. Voltage – 10/29/2005 (Constant Temperature Test)

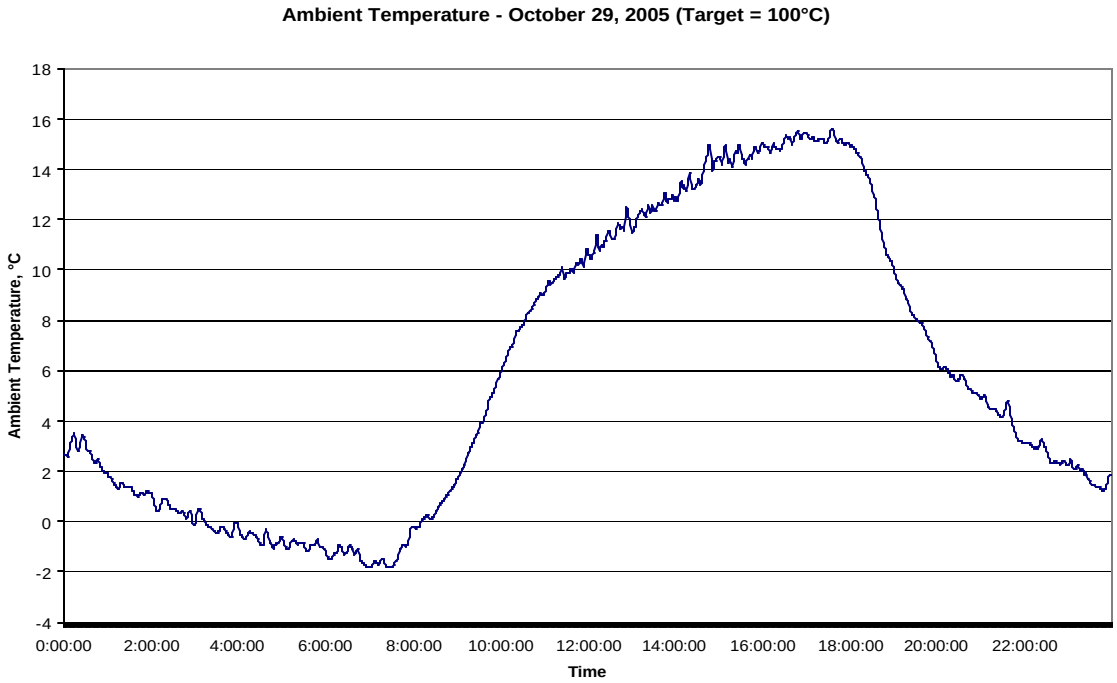

Figure 5.22. Ambient Temperature – 10/29/2005 (Constant Temperature Test)

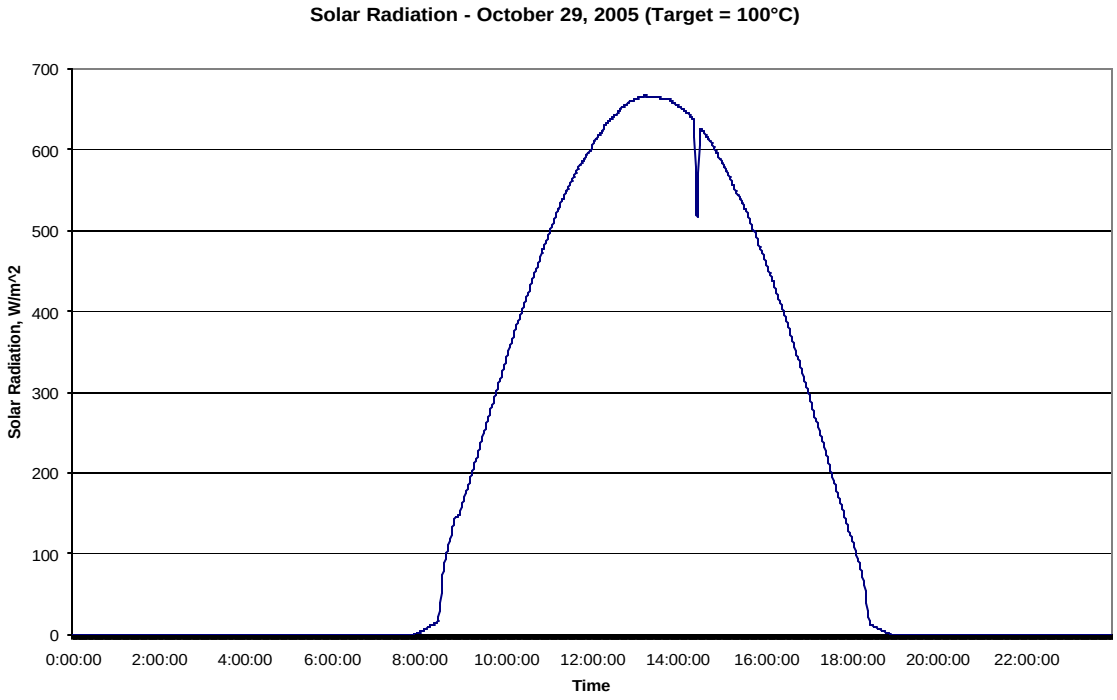

Figure 5.23. Solar Radiation – 10/29/2005 (Constant Temperature Test)

had no trouble correcting temperature differentials up to 10°C and settling back in around the target temperature within a matter of minutes.

5.8 Summary

Chapter five discussed the results of running PCATamp online. A number of insights were made into how the ampacity model functions in a control setting during the tuning phase. The insights ultimately lead to a control capable of holding the conductor temperature at +/-10ºC from a target temperature. This result was considered a success as changes in the weather are capable of pushing the conductor over 40ºC throughout a day.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The chief motivation behind the work performed in this thesis was to assist the utility industry by rapidly certifying new conductor technologies. Because of the increasing strain on the national transmission grid, it is important that the utility industry be provided with as many solutions as possible. In order to facilitate this rapid certification, an algorithm was developed that was capable of holding a section of bare overhead conductor at an elevated temperature for extended periods of time. Doing this is necessary in order to rapidly age a conductor. Rapid aging is important because in a matter of months the PCAT test facility can place the same strain on a conductor as it would experience with years of normal usage.

In order to develop this algorithm, a detailed ampacity model had to be developed for conductors tested at the Powerline Conductor Accelerated Test facility. PCATamp was created by utilizing several of the most precise heat transfer calculation methods available. Tuned against over 200,000 historic data points recorded at the PCAT test center, PCATamp proved that it could predict the temperature of bare overhead conductors remarkably well.

After the ampacity model was developed, it was ported over to Visual Basic so it could easily be used by the existing measurement and control software that had already been developed. Once ported over, an algorithm was developed to use PCATamp's

ability to predict the future temperature of the tested conductors to ensure that the driving current was always at the right level. By constantly adjusting the current flowing through the line, PCATamp could stabilize the temperature at any specified point.

Overall PCATamp was capable of controlling the temperature of the test conductors. After the optimal time constant was determined, PCATamp could be depended on to autonomously keep a conductor within +/-10ºC of the target temperature despite changes in the weather. Indeed, one of PCATamps more impressive attributes has been its resilience to crashing or leading to conditions of non-convergence.

6.2 Future Work

A number of opportunities exist not only for the further development of PCATamp but also for its use in other applications. First, one of the issues with PCATamp, and indeed ampacity research in general, is their inability to handle rain. PCATamp can not be run during rainy weather because it does not have any ability to model its cooling effects. A number of measurement devices exist today that are capable of determining the frequency and amount of rainfall, and it would be a welcome addition to be able to model the effects of this rain on the conductor.

Second, in its current state PCATamp is capable of only running in almost real time. The time needed to process a solution can take upwards of a full minute to calculate. The majority of the code has been written with readability in mind as opposed to performance optimization. More work could be done optimize the code, and it may also be helpful to use a more efficient compiler.

Third, the transient solution to the heat transfer equation is solved using a static step size Runge-Kutta-Fehlberg 45 algorithm. While this algorithm produces nearly the exact same answer as the more advanced ODE45 algorithm used by Matlab, its inability to do step-size control could mean that the PCATamp routine is doing more iterations than is necessary. Implementing a more advanced solver could help reduce computational effort.

Fourth, while PCATamp was designed to maintain conductors at test temperatures. It could be adapted to maintain a conductor at a maximum temperature. This may be useful for utilities wishing to allow as much current as their thermal limits will allow to be conducted over their lines.

Fifth, further research could be done in adapting the ampacity model for use in a control setting. Some insight into the control may be possible by linearizing the associated transient equation for use with more traditional controls research.

List of References

[1] Hirst, E., "U.S. Transmission Capacity: Present Status and Future Prospects," Edison Electric Institute and the Office of Electric Transmission and Distribution, U.S. Department of Energy, Aug. 2004

[2] Abraham, S., "National Transmission Grid Study", U.S. Department of Energy Study, May 2002.

[3] Baker, G.C., "Reonductoring Power Lines: An Example Exercise in Conductor Selection," *IEEE Rural Electric Power Conference, 2001.* pp.D1,1-D1,6, May 2001.

[4] Black, W.Z., Bush, R.A., Byrd, W.R., and Champion, T.C., "Experimental Verification of a Real-Time Program for the Determination of Temperature and Sag of Overhead Lines," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-102, pp. 2284-2288, Jul. 1983.

[5] IEEE Std 738-1993, "IEEE Standard for Calculating the Current-Temperature Relationship of Bare Overhead Conductors," Nov. 1993.

[6] House, H.E. and Tuttle, P.D., "Current carrying capacity of ACSR," *IEEE Transactions on Power Apparatus and Systems*, pp. 1169-1178, Feb. 1959.

[7] Byrd, W.R., "Transient Thermal Model of Overhead Lines," Master's Thesis, School of Mechanical Engineering, Georgia Institute of Technology, 1982.

[8] Rehberg, R.L., "High Temperature Ampacity and Sag Model for ACSR Conductors," Master's Thesis, School of Mechanical Engineering, Georgia Institute of Technology, 1983.

[9] Chen, S.L., "Programs for Calculating High Temperature Ampacity and Sag for Overhead Electrical Conductors," Master's Thesis, School of Mechanical Engineering, Georgia Institute of Technology, 1999.

[10] Reda, I. and Andreas, A., "Solar Position Algorithm for Solar Radiation Applications," *NREL Technical Report*, Jun. 2004.

[11] Shampine, L.F. and Watts, H.A., "Solving Non-stiff Ordinary Differential Equations – The State of the Art," *SIAM Review*, vol. 18, pp. 376-411, 1976.

Appendices

Appendix I

RKF45 ROUTINES

```
Option Explicit
Option Base 0


'**************************************************************************
'
Public Sub fehl_d(NEQN As Integer, y() As Double, t As Double, h As Double, yp() As Double, f1() As
Double, f2() As Double, f3() As Double, f4() As Double, f5() As Double, s() As Double)
'
'**************************************************************************
'
'  Purpose:
'
'    FEHL_D takes one Fehlberg fourth-fifth order step (double precision).
'
'  Discussion:
'
'    This routine integrates a system of NEQN first order ordinary differential
'    equations of the form
'      dY(i)/dT = F(T,Y(1:NEQN))
'    where the initial values Y and the initial derivatives
'    YP are specified at the starting point T.
'
'    The routine advances the solution over the fixed step H and returns
'    the fifth order (sixth order accurate locally) solution
'    approximation at T+H in array S.
'
'    The formulas have been grouped to control loss of significance.
'    The routine should be called with an H not smaller than 13 units of
'    roundoff in T so that the various independent arguments can be
'    distinguished.
'
'  Modified:
'
'    10 May 2005
'
'  Author:
'
'    H A Watts and L F Shampine,
'    Sandia Laboratories,
'    Albuquerque, New Mexico.
'
'  Reference:
'
'    E. Fehlberg,
'    Low-order Classical Runge-Kutta Formulas with Stepsize Control,
'    NASA Technical Report R-315.
'
'    L F Shampine, H A Watts, S Davenport,
'    Solving Non-stiff Ordinary Differential Equations - The State of the Art,
```

```
'    SIAM Review,
'    Volume 18, pages 376-411, 1976.
'
'  Parameters:
'
'    Input, external F, a user-supplied subroutine to evaluate the
'    derivatives Y'(T), of the form:
'
'      void f ( double t, double y[], double yp[] )
'
'    Input, int NEQN, the number of equations to be integrated.
'
'    Input, double Y[NEQN], the current value of the dependent variable.
'
'    Input, double T, the current value of the independent variable.
'
'    Input, double H, the step size to take.
'
'    Input, double YP[NEQN], the current value of the derivative of the
'    dependent variable.
'
'    Output, double F1[NEQN], F2[NEQN], F3[NEQN], F4[NEQN], F5[NEQN], derivative
'    values needed for the computation.
'
'    Output, double S[NEQN], the estimate of the solution at T+H.
'
'{
Dim ch As Double
Dim i As Integer

ch = h / 4#

For i = 0 To NEQN - 1

   f5(i) = y(i) + ch * yp(i)

Next

Call f(t + ch, f5, f1)

ch = 3# * h / 32#

For i = 0 To NEQN - 1

   f5(i) = y(i) + ch * (yp(i) + 3# * f1(i))

Next

Call f(t + 3# * h / 8#, f5, f2)

ch = h / 2197#

For i = 0 To NEQN - 1
```

```
   f5(i) = y(i) + ch * (1932# * yp(i) + (7296# * f2(i) - 7200 * f1(i)))

Next

Call f(t + 12# * h / 13#, f5, f3)

ch = h / 4104#

For i = 0 To NEQN - 1

   f5(i) = y(i) + ch * ((8341# * yp(i) - 845# * f3(i)) + (29440# * f2(i) - 32832# * f1(i)))

Next

Call f(t + h, f5, f4)

ch = h / 20520#

For i = 0 To NEQN - 1

   f1(i) = y(i) + ch * ((-6080# * yp(i) + (9295# * f3(i) - 5643# * f4(i))) + (41040# * f1(i) - 28352# * f2(i)))

Next

Call f(t + h / 2#, f1, f5)

ch = h / 7618050#

For i = 0 To NEQN - 1#
   s(i) = y(i) + ch * ((902880# * yp(i) + (3855735# * f3(i) - 1371249# * f4(i))) + (3953664# * f2(i) +
277020# * f5(i)))

Next

End Sub

'*************************************************************************
'
Public Function rkf45_d(NEQN As Integer, y() As Double, yp() As Double, t As Double, tout As Double,
relerr As Double, abserr As Double, flag As Integer) As Integer
'
'*************************************************************************
'
'  Purpose:
'
'    RKF45_D carries out the Runge-Kutta-Fehlberg method (double precision).
'
'  Discussion:
'
'    This routine is primarily designed to solve non-stiff and mildly stiff
'    differential equations when derivative evaluations are inexpensive.
'    It should generally not be used when the user is demanding
'    high accuracy.
'
```

```
'   This routine integrates a system of NEQN first-order ordinary differential
'   equations of the form:
'
'     dY(i)/dT = F(T,Y(1),Y(2),...,Y(NEQN))
'
'   where the Y(1:NEQN) are given at T.
'
'   Typically the subroutine is used to integrate from T to TOUT but it
'   can be used as a one-step integrator to advance the solution a
'   single step in the direction of TOUT.  On return, the parameters in
'   the call list are set for continuing the integration.  The user has
'   only to call again (and perhaps define a new value for TOUT).
'
'   Before the first call, the user must
'
'   * supply the subroutine F(T,Y,YP) to evaluate the right hand side;
'     and declare F in an EXTERNAL statement;
'
'   * initialize the parameters:
'     NEQN, Y(1:NEQN), T, TOUT, RELERR, ABSERR, FLAG.
'     In particular, T should initially be the starting point for integration,
'     Y should be the value of the initial conditions, and FLAG should
'     normally be +1.
'
'   Normally, the user only sets the value of FLAG before the first call, and
'   thereafter, the program manages the value.  On the first call, FLAG should
'   normally be +1 (or -1 for single step mode.)  On normal return, FLAG will
'   have been reset by the program to the value of 2 (or -2 in single
'   step mode), and the user can continue to call the routine with that
'   value of FLAG.
'
'   (When the input magnitude of FLAG is 1, this indicates to the program
'   that it is necessary to do some initialization work.  An input magnitude
'   of 2 lets the program know that that initialization can be skipped,
'   and that useful information was computed earlier.)
'
'   The routine returns with all the information needed to continue
'   the integration.  If the integration reached TOUT, the user need only
'   define a new TOUT and call again.  In the one-step integrator
'   mode, returning with FLAG = -2, the user must keep in mind that
'   each step taken is in the direction of the current TOUT.  Upon
'   reaching TOUT, indicated by the output value of FLAG switching to 2,
'   the user must define a new TOUT and reset FLAG to -2 to continue
'   in the one-step integrator mode.
'
'   In some cases, an error or difficulty occurs during a call.  In that case,
'   the output value of FLAG is used to indicate that there is a problem
'   that the user must address.  These values include:
'
'   * 3, integration was not completed because the input value of RELERR, the
'     relative error tolerance, was too small.  RELERR has been increased
'     appropriately for continuing.  If the user accepts the output value of
'     RELERR, then simply reset FLAG to 2 and continue.
'
```

```
'   * 4, integration was not completed because more than MAXNFE derivative
'     evaluations were needed.  This is approximately (MAXNFE/6) steps.
'     The user may continue by simply calling again.  The function counter
'     will be reset to 0, and another MAXNFE function evaluations are allowed.
'
'   * 5, integration was not completed because the solution vanished,
'     making a pure relative error test impossible.  The user must use
'     a non-zero ABSERR to continue.  Using the one-step integration mode
'     for one step is a good way to proceed.
'
'   * 6, integration was not completed because the requested accuracy
'     could not be achieved, even using the smallest allowable stepsize.
'     The user must increase the error tolerances ABSERR or RELERR before
'     continuing.  It is also necessary to reset FLAG to 2 (or -2 when
'     the one-step integration mode is being used).  The occurrence of
'     FLAG = 6 indicates a trouble spot.  The solution is changing
'     rapidly, or a singularity may be present.  It often is inadvisable
'     to continue.
'
'   * 7, it is likely that this routine is inefficient for solving
'     this problem.  Too much output is restricting the natural stepsize
'     choice.  The user should use the one-step integration mode with
'     the stepsize determined by the code.  If the user insists upon
'     continuing the integration, reset FLAG to 2 before calling
'     again.  Otherwise, execution will be terminated.
'
'   * 8, invalid input parameters, indicates one of the following:
'     NEQN <= 0;
'     T = TOUT and |FLAG| /= 1;
'     RELERR < 0 or ABSERR < 0;
'     FLAG == 0  or FLAG < -2 or 8 < FLAG.
'
' Modified:
'
'   21 April 2005
'
' Author:
'
'   H A Watts and L F Shampine,
'   Sandia Laboratories,
'   Albuquerque, New Mexico.
'
' Reference:
'
'   E. Fehlberg,
'   Low-order Classical Runge-Kutta Formulas with Stepsize Control,
'   NASA Technical Report R-315.
'
'   L F Shampine, H A Watts, S Davenport,
'   Solving Non-stiff Ordinary Differential Equations - The State of the Art,
'   SIAM Review,
'   Volume 18, pages 376-411, 1976.
'
' Parameters:
```

```
'
'   Input, external F, a user-supplied subroutine to evaluate the
'   derivatives Y'(T), of the form:
'
'     void f ( double t, double y[], double yp[] )
'
'   Input, int NEQN, the number of equations to be integrated.
'
'   Input/output, double Y[NEQN], the current solution vector at T.
'
'   Input/output, double YP[NEQN], the derivative of the current solution
'   vector at T.  The user should not set or alter this information!
'
'   Input/output, double *T, the current value of the independent variable.
'
'   Input, double TOUT, the output point at which solution is desired.
'   TOUT = T is allowed on the first call only, in which case the routine
'   returns with FLAG = 2 if continuation is possible.
'
'   Input, double *RELERR, ABSERR, the relative and absolute error tolerances
'   for the local error test.  At each step the code requires:
'     abs ( local error ) <= RELERR * abs ( Y ) + ABSERR
'   for each component of the local error and the solution vector Y.
'   RELERR cannot be "too small".  If the routine believes RELERR has been
'   set too small, it will reset RELERR to an acceptable value and return
'   immediately for user action.
'
'   Input, int FLAG, indicator for status of integration. On the first call,
'   set FLAG to +1 for normal use, or to -1 for single step mode.  On
'   subsequent continuation steps, FLAG should be +2, or -2 for single
'   step mode.
'
'   Output, int RKF45_D, indicator for status of integration.  A value of 2
'   or -2 indicates normal progress, while any other value indicates a
'   problem that should be addressed.
'

Const MAXNFE = 3000#

Dim ae As Double
Dim dt As Double
Dim ee As Double
Dim eeoet As Double
Dim eps As Double
Dim esttol As Double
Dim et As Double
Dim f1(1) As Double
Dim f2(1) As Double
Dim f3(1) As Double
Dim f4(1) As Double
Dim f5(1) As Double
Dim hfaild As Boolean
Dim hmin As Double
Dim i As Integer
```

```
Dim k As Integer
Dim mflag As Integer
Dim output As Boolean
Dim relerr_min As Double
Dim s As Double
Dim scale1 As Double
Dim tol As Double
Dim toln As Double
Dim ypk As Double

Static flag_save As Double
Static h As Double
Static init As Integer
Static kflag As Integer
Static kop As Integer
Static nfe As Integer
Static relerr_save As Double
Static remin As Double
Static abserr_save As Double

' Initialize static variables
If flag = 1 Then
   flag_save = -1000#
   h = 1#
   init = -1000#
   kflag = -1000#
   kop = -1#
   nfe = -1#
   relerr_save = -1#
   remin = 0.000000000001
   abserr_save = -1#
End If

'
' Check the input parameters.
'
eps = d_epsilon()

If (NEQN < 1) Then
   rkf45_d = 8#
   Exit Function
End If

If (relerr < 0) Then
   rkf45_d = 8#
   Exit Function
End If

If (abserr < 0) Then
   rkf45_d = 8#
   Exit Function
End If

If (flag = 0) Or (8 < flag) Or (flag < -2) Then
```

```
   rkf45_d = 8#
   Exit Function
End If

mflag = Abs(flag)

'
'  Is this a continuation call?
'
If (mflag <> 1) Then

   If (t = tout) And (kflag <> 3) Then
      rkf45_d = 8#
      Exit Function
   End If

'
'  FLAG = -2 or +2:
'
   If (mflag = 2) Then
      If (kflag = 3) Then
         flag = flag_save
         mflag = Abs(flag)
      ElseIf (init = 0) Then
         flag = flag_save
      ElseIf (kflag = 4) Then
         nfe = 0
      ElseIf (kflag = 5) And (abserr = 0) Then
         Exit Function
      ElseIf (kflag = 6) And (relerr <= relerr_save) And (abserr <= abserr_save) Then
         Exit Function
      End If

'
'  FLAG = 3, 4, 5, 6, 7 or 8.
'
   Else
      If (flag = 3) Then
         flag = flag_save
         If (kflag = 3) Then
            mflag = Abs(flag)
         End If
      ElseIf (flag = 4) Then
         nfe = 0
         flag = flag_save
         If (kflag = 3) Then
            mflag = Abs(flag)
         End If
      ElseIf (flag = 5) And (0 < abserr) Then
         flag = flag_save
         If (kflag = 3) Then
            mflag = Abs(flag)
         End If
```

```
'
' Integration cannot be continued because the user did not respond to
' the instructions pertaining to FLAG = 5, 6, 7 or 8.
'
      Else
         Exit Function
      End If
   End If
End If

'
' Save the input value of FLAG.
' Set the continuation flag KFLAG for subsequent input checking.
'
flag_save = flag
kflag = 0

'
' Save RELERR and ABSERR for checking input on subsequent calls.
'
relerr_save = relerr
abserr_save = abserr

'
' Restrict the relative error tolerance to be at least
'
'   2*EPS+REMIN
'
' to avoid limiting precision difficulties arising from impossible
' accuracy requests.
'
relerr_min = 2 * d_epsilon() + remin

'
' Is the relative error tolerance too small?
'
If (relerr < relerr_min) Then
   relerr = relerr_min
   kflag = 3#
   rkf45_d = 3#
   Exit Function
End If

dt = tout - t

'
' Initialization:
'
' Set the initialization completion indicator, INIT;
' set the indicator for too many output points, KOP;
' evaluate the initial derivatives
' set the counter for function evaluations, NFE;
' estimate the starting stepsize.
'
```

```
'  VB6 Conversion Note:
'
'  NEQN is always 1 for me, instead of the dynamically sized array used in the original C routine, I've
hardcoded
'  the array sizes for f1, f2, f3, f4, and f5 above.  If NEQN is not 1 the array size must be modified
appropriatly.
'  Below is the original C code for reference.
'
'  f1 = new double[neqn];
'  f2 = new double[neqn];
'  f3 = new double[neqn];
'  f4 = new double[neqn];
'  f5 = new double[neqn];

If (mflag = 1) Then
    init = 0#
    kop = 0#
    Call f(t, y, yp)
    nfe = 1#
    If (t = tout) Then
        rkf45_d = 2#
        Exit Function
    End If
End If

If (init = 0) Then
    init = 1
    h = Abs(dt)

    toln = 0

    For k = 0 To NEQN - 1

        tol = relerr * Abs(y(k)) + abserr

        If (0 < tol) Then
            toln = tol
            ypk = Abs(yp(k))

            If (tol < ypk * h ^ 5) Then
                h = (tol / ypk) ^ 0.2

            End If
        End If
    Next

    If (toln <= 0) Then
        h = 0#

    End If

    h = d_max(h, 26 * eps * d_max(Abs(t), Abs(dt)))

    If (flag < 0) Then
```

```
      flag_save = -2#
   Else
      flag_save = 2#
   End If
End If

'
'  Set stepsize for integration in the direction from T to TOUT.
'
h = d_sign(dt) * Abs(h)

'
'  Test to see if too may output points are being requested.
'
If (2 * Abs(dt) <= Abs(h)) Then
   kop = kop + 1
End If

'
'  Unnecessary frequency of output.
'
If (kop = 100) Then
   kop = 0#
   rkf45_d = 7#
    f1(0) = 0
    f2(0) = 0
    f3(0) = 0
    f4(0) = 0
    f5(0) = 0
   Exit Function
End If

'
'  If we are too close to the output point, then simply extrapolate and return.
'
If (Abs(dt) <= 26 * eps * Abs(t)) Then
   t = tout
    For i = 0 To NEQN - 1
      y(i) = y(i) + dt * yp(i)
   Next
   Call f(t, y, yp)
   nfe = nfe + 1#
   rkf45_d = 2#
   f1(0) = 0
    f2(0) = 0
    f3(0) = 0
    f4(0) = 0
    f5(0) = 0
   Exit Function
End If

'
'  Initialize the output point indicator.
'
```

```
output = False

'
'  To avoid premature underflow in the error tolerance function,
'  scale the error tolerances.
'
scale1 = 2 / relerr
ae = scale1 * abserr

'
'  Step by step integration.
'

Do While True

   hfaild = False

'
'  Set the smallest allowable stepsize.
'
   hmin = 26 * eps * Abs(t)

'
'  Adjust the stepsize if necessary to hit the output point.
'
'  Look ahead two steps to avoid drastic changes in the stepsize and
'  thus lessen the impact of output points on the code.
'

   dt = tout - t

'
'  Will the next successful step complete the integration to the output point?
'

   If (2# * Abs(h)) <= Abs(dt) Then

   Else

      If (Abs(dt) <= Abs(h)) Then
         output = True
         h = dt

      Else
         h = 0.5 * dt

      End If

   End If

'
'  Here begins the core integrator for taking a single step.
'
'  The tolerances have been scaled to avoid premature underflow in
```

```
' computing the error tolerance function ET.
' To avoid problems with zero crossings, relative error is measured
' using the average of the magnitudes of the solution at the
' beginning and end of a step.
' The error estimate formula has been grouped to control loss of
' significance.
'
' To distinguish the various arguments, H is not permitted
' to become smaller than 26 units of roundoff in T.
' Practical limits on the change in the stepsize are enforced to
' smooth the stepsize selection process and to avoid excessive
' chattering on problems having discontinuities.
' To prevent unnecessary failures, the code uses 9/10 the stepsize
' it estimates will succeed.
'
' After a step failure, the stepsize is not allowed to increase for
' the next attempted step.  This makes the code more efficient on
' problems having discontinuities and more effective in general
' since local extrapolation is being used and extra caution seems
' warranted.
'
' Test the number of derivative function evaluations.
' If okay, try to advance the integration from T to T+H.
'
   Do While True

'
' Have we done too much work?
'
' VB6 Conversion Note:
'
' If your output indicates only Flag = 4  and therefore nfe > MAXNFE it is a good indication that some
variables
' have blown up and the program can't converge on a solution.  This is a good indication that you've made a
' typo and should go back and check your code.

      If (MAXNFE < nfe) Then
         kflag = 4#
         rkf45_d = 4#
         f1(0) = 0
         f2(0) = 0
         f3(0) = 0
         f4(0) = 0
         f5(0) = 0
         Exit Function
      End If


'
' Advance an approximate solution over one step of length H.
'

      Call fehl_d(NEQN, y, t, h, yp, f1, f2, f3, f4, f5, f1)
```

```
        nfe = nfe + 5#

'
' Compute and test allowable tolerances versus local error estimates
' and remove scaling of tolerances.  The relative error is
' measured with respect to the average of the magnitudes of the
' solution at the beginning and end of the step.
'
        eeoet = 0#

        For k = 0 To NEQN - 1
           et = Abs(y(k)) + Abs(f1(k)) + ae
           If (et < 0) Then
              rkf45_d = 5#
              f1(0) = 0
              f2(0) = 0
              f3(0) = 0
              f4(0) = 0
              f5(0) = 0
              Exit Function
           End If

           ee = Abs((-2090# * yp(k) + (21970# * f3(k) - 15048# * f4(k))) + (22528# * f2(k) - 27360# * f5(k)))
           eeoet = d_max(eeoet, ee / et)
        Next

        esttol = Abs(h) * eeoet * scale1 / 752400#

        If (esttol < 1) Then
           Exit Do
        End If

'
' Unsuccessful step.  Reduce the stepsize, try again.
' The decrease is limited to a factor of 1/10.
'
        hfaild = True
        output = False

        If (esttol < 59049) Then
           s = 0.9 / (esttol ^ 0.2)
        Else
           s = 0.1
        End If

        h = s * h

        If (Abs(h) < hmin) Then
           kflag = 6#
           rkf45_d = 6#
           f1(0) = 0
           f2(0) = 0
           f3(0) = 0
           f4(0) = 0
```

```
            f5(0) = 0
            Exit Function
        End If

    Loop

'
' We exited the loop because we took a successful step.
' Store the solution for T+H, and evaluate the derivative there.
'

    t = t + h
    For i = 0 To NEQN - 1
        y(i) = f1(i)
    Next
    Call f(t, y, yp)
    nfe = nfe + 1

'
' Choose the next stepsize.  The increase is limited to a factor of 5.
' If the step failed, the next stepsize is not allowed to increase.
'

    If (0.0001889568 < esttol) Then
        s = 0.9 / (esttol ^ 0.2)
    Else
        s = 5#
    End If

    If hfaild Then
        s = d_min(s, 1#)
    End If

    h = d_sign(h) * d_max(s * Abs(h), hmin)

'
' End of core integrator
'
' Should we take another step?
'
    If (output) Then
        t = tout
        rkf45_d = 2#
        f1(0) = 0
        f2(0) = 0
        f3(0) = 0
        f4(0) = 0
        f5(0) = 0
        Exit Function
    End If

    If (flag <= 0) Then
        rkf45_d = -2#
        f1(0) = 0
```

```
        f2(0) = 0
        f3(0) = 0
        f4(0) = 0
        f5(0) = 0
         Exit Function
    End If

Loop

End Function
```

Appendix II

AMPACITY MODEL ROUTINES

```
Option Explicit
Option Base 0

'*************************************************************************
'
Public Function SteadyStateCurrent(wSpeed As Double, wDir As Double, TempC As Double, TempA As
Double) As Double
'
'*************************************************************************
'
' Purpose:
'
'  Calculate what the current should be for the specified weather conditions.
'  This is done by solving the steady state heat balance equation of the ampacity model
'
' Discussion:
'
'   This routine solves the steady-state heat balance equation for an ampacity model
'   that describes the temperature-current relationship of bare overhead conductor.  The routine
'   takes into account a number of conductor specific attributes, described in the 'InitConductor_*'
'   routines, and a number of weather conditions, such as ambient temperature, wind speed, wind
'   direction, and solar radiation.
'
'   The steady state equation being solved is:
'
'    Current = SquareRoot( ( Qcon + Qrad - Qsun ) / Resistance )
'
'   The following primary routines are utilized by running this function:
'
'    TcResistance(TempC As Double) - Calculates the total impdedance of a section of bare
'    overhead conductor based on its temperature.
'
'    Qsun() - Calculates the rate of radiation absorbed by the line from only the sun per unit length.
'    This routine is based on the NOAA Sunrise/Sunset and Solar Position Calculators.
'
'    mCp() - Calculates the average mass-specific heat of the bare overhead conductor
'
'    Qcon(wSpeed As Double, wDir As Double, TempC As Double, TempA As Double) - Rate of heat
transfer
'    from cable due to convection per unit length
'
'    Qrad(TempC As Double, TempA As Double) - Rate of net radiation transfer between environment
'    and cable excluding sun per unit length
'
'    If the expected current is over 1000A then SteadyStateCurrent should return a value:
'    0.85 * Actual Current <= SteadyStateCurrent <= 1.12 * Actual Current
'    On average SteadyStateCurrent should be within 0.3277% of the actual current value
'
'    If the expected current is less than 1000A then SteadyStateCurrent can not be counted on
'    to return a meaningful value.
```

```
'
' Modified:
'
'   26 May 2005
'
' Author:
'
'   Ben Sooter
'   Oak Ridge National Laboratory
'   Oak Ridge, Tennessee.
'
' Reference:
'
'  1.   Real-Time Ampacity Model For Overhead Lines
'       W.Z. Black , W.R. Byrd
'       July, 1983
'
'  2.   High-Temperature Ampacity Model for Overhead Conductors
'       S.L. Chen, W. Z. Black
'       October, 2002
'
'  3.   High Temperature Ampacity and Sag Model For ACSR Conductors
'       R.L. Rehberg
'       Thesis
'       December, 1983
'
'  4.   Transient Thermal Model of Overhead Electric Lines
'       W.R. Byrd
'       Thesis
'       June, 1982
'
'  5.   IEEE Std 738-1993
'       IEEE Standard for Calculating the Current-Temperature
'       Relationship of Bare Overhead Conductors
'
'  6.   Current-Carrying Capacity of ACSR
'       H.E. House, P.D. Tuttle
'       Februrary, 1959
'
' Paramters:
'
'  Input, wSpeed As Double, Wind Speed
'
'  Input, wdir As Double, Wind Direction
'
'  Input, TempC As Double, Temperature of the test conductor
'
'  Input, TempA As Double, Ambient Temperature
'
'  Output, SteadyStateCurrent As Double, the expected current for the provided conditions.

Dim ResistanceVar#, QsunVar#, QconVar#, QradVar#

ResistanceVar = TcResistance(TempC)
```

```
QsunVar = QsunMeasure(gSun)
QconVar = Qcon(wSpeed, wDir, TempC, TempA)
QradVar = Qrad(TempC, TempA)

SteadyStateCurrent = Sqr((QconVar + QradVar - QsunVar) / ResistanceVar)

End Function

'********************************************************************************
'
Public Function PredictTempC(TempC As Double) As Double
'
'********************************************************************************
'
' Purpose:
'
'  Predict the temperature of the test conductor at some time in the future
'  based on present weather conditions.
'
' Discussion:
'
'   This routine solves the non-steady-state heat balance equation for an ampacity model
'   that describes the temperature-current relationship of bare overhead conductor.  The routine
'   takes into account a number of conductor specific attributes, described in the 'InitConductor_*'
'   routines, and a number of weather conditions, such as ambient temperature, wind speed, wind
'   direction, and solar radiation.
'
'   The non-steady-state equation being solved is:
'
'    (dConductorTemperature / dTime) = ( ( Resistance * Current ^ 2 ) + Qsun - Qcon - Rad ) / mCp
'
'   The following primary routines are utilized by running this function:
'
'    CVResistance(TempC As Double) - Calculates the total impdedance of a section of bare
'    overhead conductor based on the measured voltage and current at that time.
'
'    Qsun() - Calculates the rate of radiation absorbed by the line from only the sun per unit length.
'    This routine is based on the NOAA Sunrise/Sunset and Solar Position Calculators.
'
'    mCp() - Calculates the average mass-specific heat of the bare overhead conductor
'
'    Qcon(wSpeed As Double, wDir As Double, TempC As Double, TempA As Double) - Rate of heat
transfer
'    from cable due to convection per unit length
'
'    Qrad(TempC As Double, TempA As Double) - Rate of net radiation transfer between environment
'    and cable, excluding sun, per unit length
'
'    f(t_temp As Double, y() As Double, yp() As Double) - The routine that contains the differential
'    equation, above, that must to solved to calculate the future temperature.
'
'    rkf45_d(NEQN As Integer, y() As Double, yp() As Double, t As Double, tout As Double,
'    relerr As Double, abserr As Double, flag As Integer) - ODE Solver based on the
'    4th-5th order Runge-Kutta-Fehlberg method written by Watts and Shampine.
```

```
'
' Modified:
'
'   20 May 2005
'
' Author:
'
'   Ben Sooter
'   Oak Ridge National Laboratory
'   Oak Ridge, Tennessee.
'
' Reference:
'
'  1.   Real-Time Ampacity Model For Overhead Lines
'       W.Z. Black , W.R. Byrd
'       July, 1983
'
'  2.   High-Temperature Ampacity Model for Overhead Conductors
'       S.L. Chen, W. Z. Black
'       October, 2002
'
'  3.   High Temperature A mpacity and Sag Model For ACSR Conductors
'       R.L. Rehberg
'       Thesis
'       December, 1983
'
'  4.   Transient Thermal Model of Overhead Electric Lines
'       W.R. Byrd
'       Thesis
'       June, 1982
'
'  5.   IEEE Std 738-1993
'       IEEE Standard for Calculating the Current-Temperature
'       Relationship of Bare Overhead Conductors
'
'  6.   Current-Carrying Capacity of ACSR
'       H.E. House, P.D. Tuttle
'       Februrary, 1959
'
' Parameters:
'
'  Input, TempC As Double, the present temperature of the conductor
'
'  Output, PredictTempC As Double, the predicted temperature of the conductor 20 min in the future.
'

Const NEQN = 1
Dim abserr As Double
Dim flag As Integer
Dim i_step As Integer
Dim n_step As Integer
Dim relerr As Double
Dim t_out As Double
Dim t_start As Double
```

```
Dim t_stop As Double



abserr = Sqr(d_epsilon())
relerr = Sqr(d_epsilon())


flag = 1

t_start = 0#
t_stop = 20#

n_step = CInt((t_stop - t_start) * 2)

t = 0#
t_out = 0#

y(0) = TempC#
Call f(t, y, yp)


'Debug.Print ""
'Debug.Print "FLAG                T                          Y                Y'"
'Debug.Print ""
'Debug.Print flag; "              "; t; "                        "; y(0); "                    "; yp(0)

For i_step = 1# To n_step
    t = ((n_step - i_step + 1) * t_start + (i_step - 1) * t_stop) / (n_step)
    t_out = ((n_step - i_step) * t_start + (i_step) * t_stop) / (n_step)
    flag = rkf45_d(NEQN, y, yp, t, t_out, relerr, abserr, flag)
'    Debug.Print flag; "               "; t; "              "; y(0); "          "; yp(0)

Next

PredictTempC = y(0)

End Function

'*******************************************************************************
'
Public Sub f(t_temp As Double, y() As Double, yp() As Double)
'
'*******************************************************************************
'
'  Purpose:
'
'   f is the function containing the differential equation you wish to solve.
'
'  Parameters:
'
'   Input, t_temp As Double, the current value of the independent variable.
'   It should not be defined in this function, only used to reference the dependent variable.
'
```

```
'   Input/Output, y() As Double, the current value of the dependent variable.
'
'   Input/Output, yp() As Double, the current value of the derivative of the
'   dependent variable.

Dim ResistanceVar As Double
Dim QsunVar As Double
Dim mCpVar As Double
Dim QconVar As Double
Dim QradVar As Double

ResistanceVar = CVResistance(gCurrent, gVoltage)
QsunVar = QsunMeasure(gSun)
mCpVar = mCp()
QconVar = Qcon(gwSpeed, gwDir, y(0), gTempA)
QradVar = Qrad(y(0), gTempA)

yp(0) = ((vCurrent ^ 2 * ResistanceVar) + QsunVar - QconVar - QradVar) / (mCpVar)

'dTdt = (Qgen + Qsun - Qrad - Qcon) / mCp;
'yp(0) = 0.25 * y(0) * (1# - y(0) / 20#)

End Sub

'*****************************************************************************
'
Public Function TcResistance(TempC As Double) As Double
'
'*****************************************************************************
'
'  Purpose:
'
'   Calculates the total impedance of the components of bare oveheard conductor. (ohms / ft)
'   Based on temperature/impedance curves for aluminum and steel
'
'  Modified:
'
'    20 May 2005
'
'  Reference:
'
'   1.    High-Temperature Ampacity Model for Overhead Conductors
'         S.L. Chen, W. Z. Black
'         October, 2002
'
'   2.    High Temperature Ampacity and Sag Model For ACSR Conductors
'         R.L. Rehberg
'         Thesis
'         December, 1983
'
'  Parameters:
'
'   Input, TempC As Double, the temperature of the conductor
'
```

```
'   Global, NAl As Double, the number of aluminum strands
'
'   Global, NSt As Double, the number of steel or composite strands
'
'   Global, dAl As Double, the diameter of an aluminum wire, ft
'
'   Global, dSt As Double, the diameter of a steel or composite wire, ft
'
'   Output, Resistance As Double, the calculated impedance of the metal.

Dim rohAl#, rohSt#, RAl#, RSt#, AcAl#, AcSt#

AcAl = pi * NAl * (dAl / 2#) ^ 2#   ' % Area of Aluminum, ft^2
AcSt = pi * NSt * (dSt / 2#) ^ 2#   ' % Area of Steel, ft^2
rohAl = ((4.716 * 10# ^ -8#) * TempC ^ 2# + (1.16852 * 10# ^ -4#) * TempC + 0.0245494) * (3.2808399 *
10# ^ -6#)
rohSt = ((4.05 * 10# ^ -7#) * TempC ^ 2# + (6.9068 * 10# ^ -4#) * TempC + 0.18149) * (3.2808399 * 10#
^ -6#)
RAl = rohAl / AcAl
RSt = rohSt / AcSt
TcResistance = (RAl * RSt) / (RAl + RSt)


End Function

'*************************************************************************
'
Public Function CVResistance(lCurrent As Double, lVoltage As Double) As Double
'
'*************************************************************************
'
'  Purpose:
'
'   Calculates the total impedance of the components of bare oveheard conductor. (ohms / ft)
'   Based on a measured voltage and current.
'
'  Modified:
'
'    20 May 2005
'
'  Parameters:
'
'   Input, lVoltage As Double, measured voltage
'
'   Input, lCurrent As Double, measured current
'
'   Output, Resistance As Double, the calculated impedance of the metal.

CVResistance = (lVoltage / lCurrent) / 2400#
End Function

'*************************************************************************
'
Public Function QsunMeasure(lSun As Double) As Double
```

```
'
'*************************************************************************************
'
'  Purpose:
'
'   Calculate the rate of radiation absorbed by the line from only the sun per unit length (W/ft)
'   (always positive ... zero at night)
'
'  Modified:
'
'    23 May 2005
'
'  Parameters:
'
'   Input, lSun As Double, measured solar radiation W/cm^2   (cm^2 generic area)
'
'   Output, Qsun as double, solar radiation W/ft  (ft of conductor)

'Dim ExposedAreasqft As Double
'Dim ExposedAreasqm As Double

'ExposedAreasqft = (1.5 * (pi * ((D / 12) / 2))) * 2400
'ExposedAreasqm = ExposedAreasqft * 0.09290304
'
' 0.25 is a fudge factor to lower the value
'QsunMeasure = lSun * 0.033564308407225 * 0.25
Dim Zl#, He#, Zc#, Hc#, qsa#, qsc#, qsi#, ecf#, theta#, Qsun1#
Dim lat As Double, lon As Double, timezone As Double
Dim daycount As Double, ndays As Double
Dim timestep As Double
Dim curyear As Double, curmonth As Double, curday As Double, curhh As Double, curmm As Double,
curss As Double
Dim az As Double, el As Double, azimuth As Double, Elevation As Double
Dim currentdatetime As Date

' The following parameters are site specific and need only be changed if this software is used
' at someplace other than PCAT.

Zl = 45#                    ' Line orientation 0=North-South 90=East-West   0 <= Zl <= 180
He = 875#                   ' Conductor elevation above sea level, ft.
lat = 36#                   ' Latitude
lon = -84.269               ' Longitude
timezone = -5#              ' Hours off GMT (-5 = EST)

'ndays = 1#                 ' Number of Days, Used in SunCalc stuff
currentdatetime = Now       ' Get the date and time right now
'daycount = 0#
curyear = Val(DatePart("yyyy", currentdatetime))    ' Year
curmonth = Val(DatePart("m", currentdatetime))      ' Month
curday = Val(DatePart("d", currentdatetime))        ' Day
curhh = Val(DatePart("h", currentdatetime))         ' Hour
curmm = Val(DatePart("n", currentdatetime))         ' Minute
curss = Val(DatePart("s", currentdatetime))         ' Second
```

```
'Zc = solarazimuth(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
'Hc = solarelevation(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
Call SolarAziAndElev(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
Zc = gSolarAzimuth
Hc = gSolarElevation

theta = ArcCos(Cos(pi * Hc / 180#) * Cos(pi * (Zc - Zl) / 180#))

QsunMeasure = absorb * (lSun / 10.7639104) * Sin(theta) * (D / 12#)



End Function


'*************************************************************************************
'
Public Function QsunCalc() As Double
'
'*************************************************************************************
'
' Purpose:
'
'  Calculate the rate of radiation absorbed by the line from only the sun per unit length (W/ft)
'  (always positive ... zero at night)
'
' Modified:
'
'   16 May 2005
'
' Reference:
'
'  1.   IEEE Std 738-1993
'       IEEE Standard for Calculating the Current-Temperature
'       Relationship of Bare Overhead Conductors
'
' Parameters:
'
'  Global, D As Double, diameter of whole conductor, inches
'
'  Global, absorb As Double, solar absorptivity (0.23 to 0.91)
'
'  Global, Atmos As Double, air quality (0 = Clear atmosphere to 1 = Industrial atmosphere)
'
'  Global, dlstime As Integer, Day Light Savings Time (0 for No, 1 for Yes)

Dim Zl#, He#, Zc#, Hc#, qsa#, qsc#, qsi#, ecf#, theta#, Qsun1#
Dim lat As Double, lon As Double, timezone As Double
Dim daycount As Double, ndays As Double
Dim timestep As Double
Dim curyear As Double, curmonth As Double, curday As Double, curhh As Double, curmm As Double,
curss As Double
Dim az As Double, el As Double, azimuth As Double, Elevation As Double
Dim currentdatetime As Date
```

```
' The following parameters are site specific and need only be changed if this software is used
' at someplace other than PCAT.

Zl = 45#                    ' Line orientation 0=North-South 90=East-West   0 <= Zl <= 180
He = 875#                   ' Conductor elevation above sea level, ft.
lat = 36#                   ' Latitude
lon = -84.269               ' Longitude
timezone = -5#              ' Hours off GMT (-5 = EST)

ndays = 1#                  ' Number of Days, Used in SunCalc stuff
currentdatetime = Now       ' Get the date and time right now
daycount = 0#
curyear = Val(DatePart("yyyy", currentdatetime))    ' Year
curmonth = Val(DatePart("m", currentdatetime))      ' Month
curday = Val(DatePart("d", currentdatetime))        ' Day
curhh = Val(DatePart("h", currentdatetime))         ' Hour
curmm = Val(DatePart("n", currentdatetime))         ' Minute
curss = Val(DatePart("s", currentdatetime))         ' Second

'Zc = solarazimuth(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
'Hc = solarelevation(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
Call SolarAziAndElev(lat, lon, curyear, curmonth, curday, curhh, curmm, curss, timezone, dlstime)
Zc = gSolarAzimuth
Hc = gSolarElevation

If Atmos < 0.5 Then
   qsa = (-3.786 * 10# ^ -10#) * Hc ^ 6# + (1.80527 * 10# ^ -7#) * Hc ^ 5# + (-3.3549 * 10# ^ -5#) * Hc ^
4# + (0.003223) * Hc ^ 3# + (-0.17856) * Hc ^ 2# + (5.92762) * Hc - 3.92414
ElseIf Atmos > 0.4 Then
   qsa = (1.22967 * 10# ^ -9#) * Hc ^ 6# + (-4.03627 * 10# ^ -7#) * Hc ^ 5# + (5.07752 * 10# ^ -5#) * Hc ^
4# + (-0.0029411) * Hc ^ 3# + (0.061444) * Hc ^ 2# + (1.320247) * Hc + (4.940779)
ElseIf Atmos = 0.5 Then
   qsc = (-3.786 * 10# ^ -10#) * Hc ^ 6# + (1.80527 * 10# ^ -7#) * Hc ^ 5# + (-3.3549 * 10# ^ -5#) * Hc ^
4# + (0.003223) * Hc ^ 3# + (-0.17856) * Hc ^ 2# + (5.92762) * Hc - 3.92414
   qsi = (1.22967 * 10# ^ -9#) * Hc ^ 6# + (-4.03627 * 10# ^ -7#) * Hc ^ 5# + (5.07752 * 10# ^ -5#) * Hc ^
4# + (-0.0029411) * Hc ^ 3# + (0.061444) * Hc ^ 2# + (1.320247) * Hc + (4.940779)
   qsa = (qsc + qsi) / 2#
End If

ecf = (-1# * 10# ^ -9#) * He ^ 2# + (3.5 * 10# ^ -5#) * He + 1#   ' Elevation Correction Factor

theta = ArcCos(Cos(pi * Hc / 180) * Cos(pi * (Zc - Zl) / 180#))
Qsun1 = absorb * qsa * Sin(theta) * ((D * 1#) / 12#)
QsunCalc = ecf * Qsun1    ' Solar hear gain, watts/ft

' The following conditional statements tweak the effective angles of the sun caused by
' physical obstruction (trees, mountains, etc) and based on observations from site
' specific data analysis.

If Hc < 12 Then
   If Val(DatePart("h", currentdatetime)) < 12 Then
      QsunCalc = 0#
   End If
```

```
End If

If Hc < 4 Then
   If Val(DatePart("h", currentdatetime)) > 12 Then
      QsunCalc = 0#
   End If
End If

If QsunCalc < 0 Then
   QsunCalc = 0#
End If

End Function


'*****************************************************************************
'
Public Function mCp() As Double
'
'*****************************************************************************
'
'  Purpose:
'
'   Calculate the average mass-specific heat of the bare overhead conductor (J/ft*C)
'
'  Modified:
'
'    16 May 2005
'
'  Reference:
'
'   1.    Transient Thermal Model of Overhead Electric Lines
'         W.R. Byrd
'         Thesis
'         June, 1982
'
'  Parameters:
'
'   Output, mCp As Double, the mass-specific heat of the overhead conductor

Dim mCpAl#, mCpSt#

mCpAl = 520#                    ' Heat capacity of Aluminium, J/ftC
mCpSt = 28#                     ' Heat capacity of Steel or Composite, J/ftC

'The following code is a more accurate method of calculating the mass-specific heat for ACSR
'For more information on the values needed for Mal and Mst see the above reference.
'Note that to use this code the function must be provided the conductor temperature, Tc, and
'any modification such as that will require the user edit the the function call in
'Public Sub f(t_temp As Double, y() As Double, yp() As Double) accordingly.
'
'  mCp = (((Mal * 0.45359237) * (0.32236 * Tc + 929.4)) + ((Mst * 0.45359237) * (0.47517 * Tc +
441.2)))/60;
```

```
'The following code should be utilized for 3M Composite cables, and may be used for ACSR
mCp = (mCpAl + mCpSt) / 60

End Function


'*************************************************************************************
'
Public Function Qcon(wSpeed2 As Double, wDir2 As Double, TempC As Double, TempA As Double) As
Double
'
'*************************************************************************************
'
' Purpose:
'
'  Calculate the rate of heat transfer from cable due to convection per unit length (W/ft)
'
' Modified:
'
'   16 May 2005
'
' Reference:
'
'  1.   High Temperature Ampacity and Sag Model For ACSR Conductors
'       R.L. Rehberg
'       Thesis
'       December, 1983
'
'  2.   Transient Thermal Model of Overhead Electric Lines
'       W.R. Byrd
'       Thesis
'       June, 1982
'
' Parameters:
'
'  Input, wSpeed As Double, the wind speed
'
'  Input, wDir As Double, the wind direction
'
'  Input, TempC As Double, the temperature of the conductor
'
'  Input, TempA As Double, the ambient temperature
'
'  Global, D As Double, the diameter of the whole conductor, inches
'
'  Global, Pr As Double, the Prandtl Number
'
'  Output, Qcon As Double, the calculated rate of heat transfer from the cable.

Dim Tfilm#, Zlrad#, thetawind#, omega#, vf#, kf#, gbvsq#, Gr#, Dsi#
Dim x#, log10nuna#, Nuna#, Re#, y#, log10Nuuc#, Nuuc#, Nufc#, h#, hfc#
Dim havg#, qcnA#, qcf#, qcavg#
Dim wDir As Double
Dim wSpeed As Double
```

```
Dsi = D / (3.2808399 * 12)          ' % Diameter of whole conductor, meters

'Convert direction to radians
wDir = 0
wDir = (wDir2 * pi) / 180#

'Convert speed to m/s
wSpeed = 0
wSpeed = wSpeed2 / 3.2808399

Tfilm = (TempC + TempA) / 2

'    Angle between conductor, Zl, and wind, Wdirrad, degrees (0=across, 90=along)
'    Cross Product:  Zl x Wdirrrad
Zlrad = (45# * pi) / 180#           ' % Line orientation in radians
thetawind = ArcSin((Cos(Zlrad) * Sin(wDir)) - (Cos(wDir) * Sin(Zlrad)))
omega = Abs((pi / 2#) - thetawind)

'    Air viscosity, W/(m*C)
vf = (1.0167 * 10# ^ -10#) * Tfilm ^ 2# + (8.7305 * 10# ^ -8#) * Tfilm + (1.3888 * 10# ^ -5#)

'    Thermal conductivity of air, m/sec
kf = (-2.7628 * 10# ^ -8#) * Tfilm ^ 2# + (7.2316 * 10# ^ -5#) * Tfilm + 0.023681

'    g: gravitional constant , B: coeffecient of thermal expansion for air  g*B/vf^2
gbvsq = (1.024 * 10# ^ 4#) * Tfilm ^ 2# + (-2.394 * 10# ^ 6#) * Tfilm + (1.85 * 10# ^ 8#)

'    Grashof number
Gr = gbvsq * (Dsi ^ 3) * (TempC - TempA)

'Log10 will fail if TempC > TempA because of Log10(-neg)
If TempC < TempA Then
   Debug.Print "Conductor Temperature Less Than Ambient Temperature!"
End If

If (Gr * Pr) = 0 Then
   x = 0#
Else
   x = Log10(Gr * Pr)
End If

'    Nusselt Number for Natural Convection
log10nuna = 0.12724 + (0.02238 * x) + (0.04203 * x ^ 2#) - (0.0025973 * x ^ 3#)
Nuna = 10 ^ log10nuna

'    Reynolds Number
Re = (wSpeed * Dsi) / vf

'    Prevent log10(0) from occurring
If Re = 0 Then
   y = 0#
Else
   y = Log10(Re)
```

```
End If


'    Nusselt Number for Forced Convection
log10Nuuc = -0.070431 + 0.31526 * y + 0.035527 * y ^ 2#
Nuuc = 10# ^ log10Nuuc
Nufc = Nuuc *  (1.194 - Sin(omega) - (0.194 * Cos(2# * omega)) + (0.368 * Sin(2# * omega)))


'    Convective heat transfer coefficient
h = (Nuna * kf) / Dsi
hfc = (Nufc * kf) / Dsi
havg = (((Nuna + Nufc) / 2) * kf) / Dsi

'    Convective Heat Loss for Natural Convection
qcnA = h * (pi * Dsi) * (TempC - TempA)

'    Convective Heat Loss for Forced Convection
qcf = hfc * (pi * Dsi) * (TempC - TempA)

'    Convective Heat Loss for Averaged Convection
qcavg = havg * (pi * Dsi) * (TempC - TempA)



If Re = 0# Then
   Qcon = qcnA / 3.2808399     'Division by 3.2808399 for converting W/m to W/ft
Else
   If Nuna > Nufc Then
      Qcon = qcavg / 3.2808399
   Else
      Qcon = qcf / 3.2808399
   End If
End If


End Function


'*************************************************************************************
'
Public Function Qrad(TempC As Double, TempA As Double) As Double
'
'*************************************************************************************
'
'  Purpose:
'
'   Calculate the rate of net radiation transfer between environment and cable excluding
'   sun per unit length (W/ft)
'
'  Modified:
'
'    16 May 2005
'
```

```
' Reference:
'
'  1.   Transient Thermal Model of Overhead Electric Lines
'       W.R. Byrd
'       Thesis
'       June, 1982
'
' Parameters:
'
'  Input, TempC As Double, the temperature of the conductor
'
'  Input, TempA As Double, the ambient temperature
'
'  Global, D As Double, the diameter of the whole conductor, inches
'
'  Global, emiss As Double, emissivity (0.23 to 0.91)
'
'  Output, Qrad As Double, the calculated rate of heat transfer between environment and cable.

Dim Dsi#

Dsi = D / (3.2808399 * 12)              ' Diameter of whole conductor, meters

Qrad = (emiss * Dsi * pi * (5.67 * 10# ^ -8#) * ((TempC + 273.15) ^ 4# - (TempA + 273.15) ^ 4#)) /
3.2808399

End Function
```

Appendix III

MATH ROUTINES

```
Option Explicit
Option Base 0

'*****************************************************************************
'
Public Function d_epsilon() As Double
'
'*****************************************************************************
'
'  Purpose:
'
'    D_EPSILON returns the round off unit for double precision arithmetic.
'
'  Discussion:
'
'    D_EPSILON is a number R which is a power of 2 with the property that,
'    to the precision of the computer's arithmetic,
'      1 < 1 + R
'    but
'      1 = ( 1 + R / 2 )
'
'  Modified:
'
'    21 April 2005
'
'  Author:
'
'    John Burkardt
'
'  Parameters:
'
'    Output, double D_EPSILON, the floating point round-off unit.
'
'  VB6 Conversion Note:
'
'    D_EPSILON will most likely return a value several factors of 10 larger than
'    then equivelant routine in C or FORTRAN.  I have not run into a problem because
'    of this but felt it worth noting.

Dim R As Double
'Original program starts this at 1, but I've reduced it to 1E-14 to decrease run time.
R = 0.00000000000001
Do While (1# < 1# + R)
   R = R / 2#
Loop
d_epsilon = (2# * R)
End Function

'*****************************************************************************
'
```

```
Public Function d_max(ByVal A As Double, ByVal B As Double) As Double
'
'*****************************************************************
'
' Purpose:
'
'   D_MAX returns the maximum of two double precision values.
'
' Modified:
'
'   21 April 2005
'
' Author:
'
'   John Burkardt
'
' Parameters:
'
'   Input, double a, b, the quantities to compare.
'
'   Output, double D_MAX, the maximum of a and b.
'
If (B < A) Then
   d_max = A
Else
   d_max = B
End If
End Function

'*****************************************************************
'
Public Function d_min(ByVal A As Double, ByVal B As Double) As Double
'
'*****************************************************************
'
' Purpose:
'
'   D_MIN returns the minimum of two double precision values.
'
' Modified:
'
'   21 April 2005
'
' Author:
'
'   John Burkardt
'
' Parameters:
'
'   Input, double a, b, the quantities to compare.
'
'   Output, double D_MIN, the minimum of a and b.
'
If (B < A) Then
```

```
   d_min = B
Else
   d_min = A
End If
End Function

'**********************************************************************
'
Public Function d_sign(ByVal A As Double) As Double
'
'**********************************************************************
'
' Purpose:
'
'   D_SIGN returns the sign of a real number.
'
' Modified:
'
'   21 April 2005
'
' Author:
'
'   John Burkardt
'
' Parameters:
'
'   Input, double a, the number whose sign is desired.
'
'   Output, double D_SIGN, the sign of a.
'
If (A < 0#) Then
   d_sign = -1#
Else
   d_sign = 1#
End If
End Function

'**********************************************************************
'
Public Function ArcCos(A As Double) As Double
'
'**********************************************************************
'
' Purpose:
'
'   ArcCos returns the Arc-Cosine (cos^-1) of a number.
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double a, the number whose ArcCos is desired.
```

```
'
'   Output, double ArcCos, the Arc-Cosine of a.
'
   'Check for A = 1 to prevent div by zero
   If Abs(A) = 1 Then
      ArcCos = 0
   Else
      ArcCos = Atn(-A / Sqr(-A * A + 1)) + 2 * Atn(1)
   End If
End Function

'*****************************************************************************
'
Public Function ArcSin(A As Double) As Double
'
'*****************************************************************************
'
' Purpose:
'
'   ArcSin returns the Arc-Sine (sin^-1) of a number.
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double a, the number whose ArcSin is desired.
'
'   Output, double ArcSin, the Arc-Sine of a.
'
   'Check for A = 1 to prevent div by zero
   If Abs(A) = 1 Then
      ArcSin = Sgn(A) * 1.57079633
   Else
      ArcSin = Atn(A / Sqr(-A * A + 1))
   End If
End Function
'*****************************************************************************
'
Public Function Log10(A As Double) As Double
'
'*****************************************************************************
'
' Purpose:
'
'   Log10 returns the of a number for base 10.
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
```

```
'   Input, double a, the number whose log equivlent is desired.
'
'   Output, double Log10, the log of a.
'
   Log10 = Log(A) / Log(10#)
End Function

'*********************************************************************
'
Public Function radToDeg(angleRad)
'
'*********************************************************************
'
' Purpose:
'
'   Convert a radian angle to degrees
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double angleRad, the number whose degree equivalent is desired.
'
'   Output, double radToDeg, the degree equivalent of angleRad
'
   radToDeg = 57.2957795131 * angleRad

End Function

'*********************************************************************
'
Public Function degToRad(angleDeg)
'
'*********************************************************************
'
' Purpose:
'
'   Convert a degree angle to radians
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double angleDeg, the number whose radian equivalent is desired.
'
'   Output, double degToRad, the radian equivalent of angleDeg
'

   degToRad = 0.01745329252 * angleDeg
```

```
End Function

'*********************************************************************
'
Public Function roundDown(dblValue As Double) As Double
'
'*********************************************************************
'
' Purpose:
'
'   round down a number to whole integers only
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double dBlValue, the number that you want to round down.
'
'   Output, double roundDown, the whole integer value of dblValue
'

   roundDown = Round((dblValue - 0.5), 0)

End Function
'*********************************************************************
'
Public Function Atan2(ByVal y As Double, ByVal x As Double) As Double
'
'*********************************************************************
'
' Purpose:
'
'   The Atan2 function computes the principal value of the arc tangent of y/x,
'   using the signs of both arguments to determine the quadrant of the return value.
'
' Modified:
'
'   21 April 2005
'
' Parameters:
'
'   Input, double x, double y, the numbers whose Arc-Tangent is desired.
'
'   Output, double Atan2, the arc tangent of y/x in the range [-, ] radians.
'
 Dim theta As Double

 If (Abs(x) < 0.0000001) Then
   If (Abs(y) < 0.0000001) Then
     theta = 0#
   ElseIf (y > 0#) Then
     theta = 1.5707963267949
```

```
    Else
      theta = -1.5707963267949
    End If
  Else

    theta = Atn(y / x)

    If (x < 0) Then
      If (y >= 0#) Then
        theta = 3.14159265358979 + theta
      Else
        theta = theta - 3.14159265358979
      End If
    End If
  End If

  Atan2 = theta
End Function
```

Appendix IV

SUN TRAJECTORY ROUTINES

```
Option Explicit
Option Base 0


' Calculation of local times of sunrise, solar noon, and sunset
' based on the calculation procedure by NOAA in the javascript in
' http://www.srrb.noaa.gov/highlights/sunrise/sunrise.html and
' http://www.srrb.noaa.gov/highlights/sunrise/azel.html
'
' Five functions are available for use from Excel worksheets:
'
'   - sunrise(lat, lon, year, month, day, timezone, dlstime)
'   - solarnoon(lat, lon, year, month, day, timezone, dlstime)
'   - sunset(lat, lon, year, month, day, timezone, dlstime)
'   - solarazimuth(lat, lon, year, month, day, hour, minute, second, timezone, dlstime)
'   - solarelevation(lat, lon, year, month, day, hour, minute, second, timezone, dlstime)
'
' The sign convention for inputs to the functions named sunrise, solarnoon,
' sunset, solarazimuth, and solarelevationis:
'
'   - positive latitude decimal degrees for northern hemisphere
'   - negative longitude degrees for western hemisphere
'   - negative time zone hours for western hemisphere
'
' The other functions in the VBA module use the original
' NOAA sign convention of positive longitude in the western hemisphere.
'
' The calculations in the NOAA Sunrise/Sunset and Solar Position
' Calculators are based on equations from Astronomical Algorithms,
' by Jean Meeus. NOAA also included atmospheric refraction effects.
' The sunrise and sunset results were reported by NOAA
' to be accurate to within +/- 1 minute for locations between +/- 72°
' latitude, and within ten minutes outside of those latitudes.
'
' This translation was tested for selected locations
' and found to provide results within +/- 1 minute of the
' original Javascript code.
'
' This translation does not include calculation of prior or next
' susets for locations above the Arctic Circle and below the
' Antarctic Circle, when a sunrise or sunset does not occur.
'
' Translated from NOAA's Javascript to Excel VBA by:
'
' Greg Pelletier
' Olympia, WA
' e-mail: pelican@vei.net

Public Function calcJD(year, month, day)
```

```
'*************************************************************************/
'* Name:    calcJD
'* Type:    Function
'* Purpose: Julian day from calendar day
'* Arguments:
'*   year : 4 digit year
'*   month: January = 1
'*   day  : 1 - 31
'* Return value:
'*   The Julian day corresponding to the date
'* Note:
'*   Number is returned for start of day.  Fractional days should be
'*   added later.
'*************************************************************************/

Dim A As Double, B As Double, JD As Double

      If (month <= 2) Then
       year = year - 1#
       month = month + 12#
      End If

      A = roundDown(year / 100#)
      B = 2# - A + roundDown(A / 4#)

      JD = roundDown(365.25 * (year + 4716#)) + _
          roundDown(30.6001 * (month + 1#)) + day + B - 1524.5
      calcJD = JD

'gp put the year and month back where they belong
      If month = 13 Then
       month = 1#
       year = year + 1#
      End If
      If month = 14 Then
       month = 2#
       year = year + 1#
      End If

End Function

'REPLACED IN REST OF PROGRAM BY CUT AND PASTE OF CODE BELOW
'Public Function calcTimeJulianCent(JD)
'
''*************************************************************************/
''* Name:    calcTimeJulianCent
''* Type:    Function
''* Purpose: convert Julian Day to centuries since J2000.0.
''* Arguments:
''*   jd : the Julian Day to convert
''* Return value:
''*   the T value corresponding to the Julian Day
''*************************************************************************/
'
```

```
'Dim t As Double
'
'       t = (JD - 2451545#) / 36525#
'       calcTimeJulianCent = t
'
'End Function


'Public Function calcJDFromJulianCent(t)
'
''*************************************************************************/
''* Name:    calcJDFromJulianCent
''* Type:    Function
''* Purpose: convert centuries since J2000.0 to Julian Day.
''* Arguments:
''*   t : number of Julian centuries since J2000.0
''* Return value:
''*   the Julian Day corresponding to the t value
''*************************************************************************/
'
'Dim JD As Double
'
'       JD = t * 36525# + 2451545#
'       calcJDFromJulianCent = JD
'
'End Function

'REPLACED IN REST OF PROGRAM BY CUT AND PASTE OF CODE BELOW
'Public Function calcGeomMeanLongSun(t)
'
''*************************************************************************/
''* Name:    calGeomMeanLongSun
''* Type:    Function
''* Purpose: calculate the Geometric Mean Longitude of the Sun
''* Arguments:
''*   t : number of Julian centuries since J2000.0
''* Return value:
''*   the Geometric Mean Longitude of the Sun in degrees
''*************************************************************************/
'
'Dim ten As Double
'
'       ten = 280.46646 + t * (36000.76983 + 0.0003032 * t)
'       ten = ten - 2160
'' WE ARE CONCERNED WITH PRESENT TIME, SO WE DON'T NEED THIS
''       Do While (ten > 360) Or (ten < 0)
''          Debug.Print "WE ENTERED THE LOOP AND WE DIDNT EXPECT TO"
''          If (ten <= 360) And (ten >= 0) Then Exit Do
''          If ten > 360 Then ten = ten - 360#
''          If ten < 0 Then ten = ten + 360#
''       Loop
'
'       calcGeomMeanLongSun = ten
''REPLACED WITH THIS CODE
```

```
"((280.46646 + t * (36000.76983 + 0.0003032 * t))-2160)
'
'End Function

'REPLACED IN REST OF PROGRAM BY CUT AND PASTE OF CODE BELOW
'Public Function calcGeomMeanAnomalySun(t)
'
"*********************************************************************/
"* Name:    calGeomAnomalySun
"* Type:    Function
"* Purpose: calculate the Geometric Mean Anomaly of the Sun
"* Arguments:
"*   t : number of Julian centuries since J2000.0
"* Return value:
"*   the Geometric Mean Anomaly of the Sun in degrees
"*********************************************************************/
'
'Dim m As Double
'
'       m = 357.52911 + t * (35999.05029 - 0.0001537 * t)
'       calcGeomMeanAnomalySun = m
'
'End Function

'REPLACED IN REST OF PROGRAM BY CUT AND PASTE OF CODE BELOW
'Public Function calcEccentricityEarthOrbit(t)
'
"*********************************************************************/
"* Name:    calcEccentricityEarthOrbit
"* Type:    Function
"* Purpose: calculate the eccentricity of earth's orbit
"* Arguments:
"*   t : number of Julian centuries since J2000.0
"* Return value:
"*   the unitless eccentricity
"*********************************************************************/
'
'Dim e As Double
'
'       e = 0.016708634 - t * (0.000042037 + 0.0000001267 * t)
'       calcEccentricityEarthOrbit = e
'
'End Function


Public Function calcSunEqOfCenter(t)

'*********************************************************************/
'* Name:    calcSunEqOfCenter
'* Type:    Function
'* Purpose: calculate the equation of center for the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
```

```
'*   in degrees
'*************************************************************************/

Dim m As Double, mrad As Double, sinm As Double, sin2m As Double, sin3m As Double
Dim c As Double

      m = 357.52911 + t * (35999.05029 - 0.0001537 * t)

      mrad = degToRad(m)
      sinm = Sin(mrad)
      sin2m = Sin(mrad + mrad)
      sin3m = Sin(mrad + mrad + mrad)

      c = sinm * (1.914602 - t * (0.004817 + 0.000014 * t)) + sin2m * (0.019993 - 0.000101 * t) + sin3m *
0.000289

      calcSunEqOfCenter = c

End Function


Public Function calcSunTrueLong(t)

'*************************************************************************/
'* Name:    calcSunTrueLong
'* Type:    Function
'* Purpose: calculate the true longitude of the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun's true longitude in degrees
'*************************************************************************/

Dim ten As Double, c As Double, O As Double

      ten = ((280.46646 + t * (36000.76983 + 0.0003032 * t)) - 2160)
      c = calcSunEqOfCenter(t)

      O = ten + c
      calcSunTrueLong = O

End Function


Public Function calcSunTrueAnomaly(t)

'*************************************************************************/
'* Name:    calcSunTrueAnomaly (not used by sunrise, solarnoon, sunset)
'* Type:    Function
'* Purpose: calculate the true anamoly of the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun's true anamoly in degrees
```

```
'*************************************************************************/

Dim m As Double, c As Double, v As Double

      m = 357.52911 + t * (35999.05029 - 0.0001537 * t)
      c = calcSunEqOfCenter(t)

      v = m + c
      calcSunTrueAnomaly = v

End Function


Public Function calcSunRadVector(t)

'*************************************************************************/
'* Name:    calcSunRadVector (not used by sunrise, solarnoon, sunset)
'* Type:    Function
'* Purpose: calculate the distance to the sun in AU
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun radius vector in AUs
'*************************************************************************/

Dim v As Double, e As Double, R As Double

      v = calcSunTrueAnomaly(t)
      e = 0.016708634 - t * (0.000042037 + 0.0000001267 * t)

      R = (1.000001018 * (1# - e * e)) / (1# + e * Cos(degToRad(v)))
      calcSunRadVector = R

End Function


Public Function calcSunApparentLong(t)

'*************************************************************************/
'* Name:    calcSunApparentLong (not used by sunrise, solarnoon, sunset)
'* Type:    Function
'* Purpose: calculate the apparent longitude of the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun's apparent longitude in degrees
'*************************************************************************/

Dim O As Double, omega As Double, lambda As Double

      O = calcSunTrueLong(t)

      omega = 125.04 - 1934.136 * t
      lambda = O - 0.00569 - 0.00478 * Sin(degToRad(omega))
```

```
    calcSunApparentLong = lambda

End Function


Public Function calcMeanObliquityOfEcliptic(t)

'************************************************************************/
'* Name:    calcMeanObliquityOfEcliptic
'* Type:    Function
'* Purpose: calculate the mean obliquity of the ecliptic
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   mean obliquity in degrees
'************************************************************************/

Dim seconds As Double, e0 As Double

    seconds = 21.448 - t * (46.815 + t * (0.00059 - t * (0.001813)))
    e0 = 23# + (26# + (seconds / 60#)) / 60#
    calcMeanObliquityOfEcliptic = e0

End Function


Public Function calcObliquityCorrection(t)

'************************************************************************/
'* Name:    calcObliquityCorrection
'* Type:    Function
'* Purpose: calculate the corrected obliquity of the ecliptic
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   corrected obliquity in degrees
'************************************************************************/

Dim e0 As Double, omega As Double, e As Double

    e0 = calcMeanObliquityOfEcliptic(t)

    omega = 125.04 - 1934.136 * t
    e = e0 + 0.00256 * Cos(degToRad(omega))
    calcObliquityCorrection = e

End Function


Public Function calcSunRtAscension(t)

'************************************************************************/
'* Name:    calcSunRtAscension (not used by sunrise, solarnoon, sunset)
'* Type:    Function
```

```
'* Purpose: calculate the right ascension of the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun's right ascension in degrees
'**********************************************************************/

Dim e As Double, lambda As Double, tananum As Double, tanadenom As Double
Dim alpha As Double

      e = calcObliquityCorrection(t)
      lambda = calcSunApparentLong(t)

      tananum = (Cos(degToRad(e)) * Sin(degToRad(lamb da)))
      tanadenom = (Cos(degToRad(lambda)))

'original NOAA code using javascript Math.Atan2(y,x) convention:
'       var alpha = radToDeg(Math.atan2(tananum, tanadenom));
'       alpha = radToDeg(Application.WorksheetFunction.Atan2(tananum, tanadenom))

'translated using Excel VBA Application.WorksheetFunction.Atan2(x,y) convention:
      alpha = radToDeg(Atan2(tanadenom, tananum))

      calcSunRtAscension = alpha

End Function


Public Function calcSunDeclination(t)

'**********************************************************************/
'* Name:    calcSunDeclination
'* Type:    Function
'* Purpose: calculate the declination of the sun
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   sun's declination in degrees
'**********************************************************************/

Dim e As Double, lambda As Double, sint As Double, theta As Double

      e = calcObliquityCorrection(t)
      lambda = calcSunApparentLong(t)

      sint = Sin(degToRad(e)) * Sin(degToRad(lambda))
      theta = radToDeg(ArcSin(sint))
      calcSunDeclination = theta

End Function


Public Function calcEquationOfTime(t)
```

```
'*************************************************************************/
'* Name:    calcEquationOfTime
'* Type:    Function
'* Purpose: calculate the difference between true solar time and mean
'*     solar time
'* Arguments:
'*   t : number of Julian centuries since J2000.0
'* Return value:
'*   equation of time in minutes of time
'*************************************************************************/

Dim epsilon As Double, ten As Double, e As Double, m As Double
Dim y As Double, sin2l0 As Double, sinm As Double
Dim cos2l0 As Double, sin4l0 As Double, sin2m As Double, Etime As Double

      epsilon = calcObliquityCorrection(t)
      ten = ((280.46646 + t * (36000.76983 + 0.0003032 * t)) - 2160)
      e = 0.016708634 - t * (0.000042037 + 0.0000001267 * t)
       m = 357.52911 + t * (35999.05029 - 0.0001537 * t)

      y = Tan(degToRad(epsilon) / 2#)
      y = y ^ 2

      sin2l0 = Sin(2# * degToRad(ten))
      sinm = Sin(degToRad(m))
      cos2l0 = Cos(2# * degToRad(ten))
      sin4l0 = Sin(4# * degToRad(ten))
      sin2m = Sin(2# * degToRad(m))

      Etime = y * sin2l0 - 2# * e * sinm + 4# * e * y * sinm * cos2l0 _
            - 0.5 * y * y * sin4l0 - 1.25 * e * e * sin2m

      calcEquationOfTime = radToDeg(Etime) * 4#

End Function


Public Sub SolarAziAndElev(lat, lon, year, month, day, hours, minutes, seconds, timezone, dlstime)

Dim longitude As Double, latitude As Double
Dim zone As Double, daySavings As Double
Dim hh As Double, mm As Double, ss As Double, timenow As Double
Dim JD As Double, t As Double, R As Double
Dim alpha As Double, theta As Double, Etime As Double, eqtime As Double
Dim solarDec As Double, earthRadVec As Double, solarTimeFix As Double
Dim trueSolarTime As Double, hourangle As Double, harad As Double
Dim csz As Double, zenith As Double, azDenom As Double, azRad As Double
Dim azimuth As Double, exoatmElevation As Double
Dim step1 As Double, step2 As Double, step3 As Double
Dim refractionCorrection As Double, te As Double, solarzen As Double

' change sign convention for longitude from negative to positive in western hemisphere
        longitude = lon * -1#
        latitude = lat
```

```
        If (latitude > 89.8) Then latitude = 89.8
        If (latitude < -89.8) Then latitude =  -89.8

'change time zone to ppositive hours in western hemisphere
        zone = timezone * -1#
        daySavings = dlstime * 60#
        hh = hours - (daySavings / 60#)
        mm = minutes
        ss = seconds

'//   timenow is GMT time for calculation in hours since 0Z
        timenow = hh + mm / 60# + ss / 3600# + zone

        JD = calcJD(year, month, day)
        t = ((JD + timenow / 24#) - 2451545#) / 36525#
        R = calcSunRadVector(t)
        alpha = calcSunRtAscension(t)
        theta = calcSunDeclination(t)
        Etime = calcEquationOfTime(t)

        eqtime = Etime
        solarDec = theta '//    in degrees
        earthRadVec = R

        solarTimeFix = eqtime - 4# * longitude + 60# * zone
        trueSolarTime = hh * 60# + mm + ss / 60# + solarTimeFix
        '//    in minutes

        Do While (trueSolarTime > 1440)
          trueSolarTime = trueSolarTime - 1440#
        Loop

        hourangle = trueSolarTime / 4# - 180#
        '//    Thanks to Louis Schwarzmayr for the next line:
        If (hourangle < -180) Then hourangle = hourangle + 360#

        harad = degToRad(hourangle)

        csz = Sin(degToRad(latitude)) * _
            Sin(degToRad(solarDec)) + _
            Cos(degToRad(latitude)) * _
            Cos(degToRad(solarDec)) * Cos(harad)

        If (csz > 1#) Then
          csz = 1#
        ElseIf (csz < -1#) Then
          csz = -1#
        End If

        zenith = radToDeg(ArcCos(csz))

        azDenom = (Cos(degToRad(latitude)) * Sin(degToRad(zenith)))

        If (Abs(azDenom) > 0.001) Then
```

```
            azRad = ((Sin(degToRad(latitude)) * _
               Cos(degToRad(zenith))) - _
               Sin(degToRad(solarDec))) / azDenom
            If (Abs(azRad) > 1#) Then
               If (azRad < 0) Then
                  azRad = -1#
               Else
                  azRad = 1#
               End If
            End If

            azimuth = 180# - radToDeg(ArcCos(azRad))

            If (hourangle > 0#) Then
               azimuth = -azimuth
            End If
         Else
            If (latitude > 0#) Then
               azimuth = 180#
            Else
               azimuth = 0#
            End If
         End If
         If (azimuth < 0#) Then
            azimuth = azimuth + 360#
         End If

         exoatmElevation = 90# - zenith

'beginning of simplified expression
         If (exoatmElevation > 85#) Then
            refractionCorrection = 0#
         Else
            te = Tan(degToRad(exoatmElevation))
            If (exoatmElevation > 5#) Then
               refractionCorrection = 58.1 / te - 0.07 / (te * te * te) + _
                  0.000086 / (te * te * te * te * te)
            ElseIf (exoatmElevation > -0.575) Then
               step1 = (-12.79 + exoatmElevation * 0.711)
               step2 = (103.4 + exoatmElevation * (step1))
               step3 = (-518.2 + exoatmElevation * (step2))
               refractionCorrection = 1735# + exoatmElevation * (step3)
            Else
               refractionCorrection = -20.774 / te
            End If
            refractionCorrection = refractionCorrection / 3600#
         End If
'end of simplified expression

         solarzen = zenith - refractionCorrection

'          If (solarZen < 108#) Then
'           solarazimuth = azimuth
           gSolarElevation = 90# - solarzen
```

```
        gSolarAzimuth = azimuth

End Sub
```

Vita

Matthew B. Sooter has been studying electrical engineering since 1999. He received his Bachelors of Science from the University of Texas at Austin in May 2003. After that Matthew went on to pursue his Masters of Science in Electrical Engineering, specializing in Power Systems and Power Electronics at the University of Tennessee at Knoxville. Matthew received his Masters of Science in December 2005. During his graduate studies he worked as part of the Cooling, Heating, and Power group at Oak Ridge National Laboratory where his work with advanced high temperature conductors lead to this thesis. Matthew is currently pursuing a career in Electrical Engineering.