12-2014

A Model of the Diurnal Variation in Lake Surface Temperature

Jonathan L. Hodges
Clemson University

A model of the diurnal variation in lake surface temperature

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science
Mechanical Engineering

---

by
Jonathan L. Hodges
December 2014

Accepted by:
Dr. John R. Saylor, Committee Chair
Dr. Nigel B. Kaye
Dr. Richard Miller

Abstract

Satellite measurements of water surface temperature can benefit several environmental applications such as predictions of lake evaporation, meteorological forecasts, and predictions of lake overturning events, among others. However, limitations on the temporal resolution of satellite measurements restrict these improvements. A model of the diurnal variation in lake surface temperature could potentially increase the effective temporal resolution of satellite measurements of surface temperature, thereby enhancing the utility of these measurements in the above applications. Herein, a one-dimensional transient thermal model of a lake is used in combination with surface temperature measurements from the Moderate Resolution Imaging Spectroradiometer (MODIS) instrument aboard the Aqua and Terra satellites, along with ambient atmospheric conditions from local weather stations, and bulk temperature measurements to calculate the diurnal surface temperature variation for the five major lakes in the Savannah River Basin in South Carolina: Lakes Jocassee, Keowee, Hartwell, Russell, and Thurmond. The calculated solutions are used to obtain a functional form for the diurnal surface temperature variation of these lakes. Differences in diurnal variation in surface temperature between each of these lakes are identified and potential explanations for these differences are presented.

Dedication

I dedicate my master's work to my family. A special gratitude to my parents, Igou and Linda Hodges who have always supported and encouraged me to pursue excellence. My brother Daniel and sister Elizabeth who have always expressed love and interest in me and my work. I also dedicate this master's thesis to my loving wife who has supported me through this process. I will always appreciate the time and energy she has allowed me to use in pursuit of this degree. I appreciate her listening ear and intelligent ideas towards this work.

Acknowledgments

I would like to express my sincere gratitude to my advisers Dr. John R. Saylor and Dr. Nigel B. Kaye for their useful comments, remarks, and engagement through the learning process of this master's thesis. Additionally, I would like to thank the other member of my committee Dr. Richard Miller for sharing his expertise and taking time to review this thesis. I would also like to thank my fellow graduate students for their ideas and contributions towards this work. I would like to especially thank Ryne Philips for his prior work with the ASOS and MODIS data.

Furthermore, financial support from the USGS through the South Carolina Water Resources Center (SCWRC) is gratefully acknowledged as is access to temperature measurements of Lake Hartwell by the U. S. Army Corps of Engineers. Surface temperature and cloud cover data were obtained from the MODIS Adaptive Processing System, part of the NASA Earth-Sun System Division. Ambient atmospheric measurement data were acquired from the National Oceanic and Atmospheric Administration (NOAA).

Table of Contents

List of Tables

List of Figures

Nomenclature

α	volumetric thermal expansion coefficient of water
Δt	simulation time step
ϵ	thermal infrared emissivity of water
η	net efficiency of the introduction of kinetic energy at the surface
λ	Reed correction factor
ϕ	relative humidity
Φ_0	solar constant
Φ_E	energy flux due to entrainment
Φ_{lf}	latent heat flux at the surface
Φ_N	net heat flux at the surface
Φ_{ri}	long wave radiation flux at the surface
Φ_{sf}	sensible heat flux at the surface
Φ_s	incident short wave radiation flux at the surface
ρ	density
σ	Stefan-Boltzmann constant
A	Albedo of water
b_1	solar calibration parameter 1
b_2	solar calibration parameter 2
C	cloud cover index
C_D	drag coefficient
c_E	coefficient of turbulent exchange
c_H	coefficient of turbulent exchange
C_k^f	internal losses coefficient
c_p	specific heat capacity
C_T	kinetic energy coefficient

D_r	Dendritic ratio
e_a	partial pressure of water vapor
E_m	mass evaporation rate
e_{sat}	saturated vapor pressure
f^*	dimensionless frequency
g	acceleration due to gravity
H	depth of the lake
h_{fg}	latent heat of vaporization
h_m	mass transfer coefficient
L	mixed layer depth
N	Total number of satellite measurements
P	pressure
q	vapor concentration
q_*	combination velocity scale
T	temperature
T_{err}	simulation temperature error at a satellite point.
T_{rms}	Simulation rms error in K
T_{sat}	Satellite measured surface temperature in K
T_s	Simulation surface temperature in K
u_*	shearing wind velocity
u_{10}	wind speed 10 meters above the lake surface
w_*	buoyant velocity scale
	Subscripts
a	ambient air
k	harmonic
max	maximum value of an individual day
min	minimum value of an individual day
$rise$	sunrise of an individual day
s	surface
set	sunset of an individual day
w	water
b	bulk
	Superscripts
$*$	denotes non dimensional variable

Chapter 1

Introduction

The air/water interface of lakes and reservoirs is the location where numerous environmentally relevant processes are mediated. These include the transfer of dissolved gases such as oxygen, carbon dioxide and methane, the transfer of heat to and from the atmosphere, and the evaporation and condensation of water at the surface. All of these processes depend critically on the water surface temperature, T_s, which directly or indirectly controls the driving force for all of the transport processes listed above. In addition, meteorological predictions of the global climate and predictions of lake overturning events depend critically on T_s.

1.1 Surface temperature measurement

Measuring T_s is generally more difficult than measuring other ambient parameters relevant to atmospheric processes. Measurements in the literature have been performed using a thermocouple or thermistor located just below the water surface or a radiometer located above the water surface. One advantage of these methods is the capability to obtain a continuous time trace of T_s. However, each of these methods also has its own problems.

When using a thermocouple or thermistor it is difficult to obtain a true T_s measurement since T_s has been shown to vary within the first few mm of water.[2] Thus, waves and variations in lake level can cause the sensor to move above the surface or too deep to measure the true surface temperature. Additionally, although the measurement device itself has a low cost, each measurement location on each lake will require some form of buoy system with a power supply and data acquisition

capability, all of which make it a challenge to deploy enough sensors to ascertain spatial variation of T_s.

Radiometer measurements have the advantage of measuring T_s very close to the surface. However, the accuracy of these measurements can be affected by changes in the spectral properties of the water surface due to waves or large scale mixing. Additionally, radiometers are expensive and will also require some form of buoy system for each measurement location. Thus, until recently obtaining T_s measurements over the surface of a body of water having any significant horizontal extent has been difficult.

Recent advancements in satellite remote sensing allow for measurements of T_s over large areas and with reasonable spatial resolution. When dealing with satellite measurements, there is always some trade-off between spatial and temporal resolution. For example, USGS LANDSAT images of the visible and infrared spectrum are obtained with a spatial resolution of 30 meters, but with a temporal resolution of approximately once every 16 days.[3] This spatial resolution is excellent; however, if knowledge of diurnal variation is desired, the temporal resolution is insufficient. The Moderate Resolution Imaging Spectroradiometer (MODIS) satellites on the other hand have a spatial resolution of 1000 meters obtained twice daily. The two MODIS satellites, Aqua and Terra, follow a similar orbit but have a temporal offset of approximately 3 hours. By compiling data from these two satellites a maximum of 4 measurements per day is obtained.[4]

While a temporal resolution of 4 satellite measurements per day may be satisfactory in some cases, for many applications this resolution will be inadequate since it will make difficult obtaining even a daily maximum and minimum T_s, for example. Having a functional form for the diurnal variation in T_s would effectively enable an increase in the temporal resolution of satellite-obtained measurements of T_s. If such a function existed, the 4 satellite measurements currently available could be used to obtain the unknown constants in such a function, thereby providing an equation for T_s, continuous in time, for each day.

1.2 Motivation

The utility of such a function can be illustrated using lake evaporation measurement as an example. The most common method for measuring lake evaporation is the evaporation pan, where a pan is located on the lake shore. Evaporation from the pan is measured and related to that of

a lake through an empirically determined pan coefficient.[5] However specific setup, maintenance, and environmental and operational conditions significantly affect pan measurements, which makes it difficult to achieve consistent evaporation measurements from such pans.[5] Moreover, several aspects of pan evaporation measurements cause significant inaccuracies which may not be completely accounted for by the pan coefficient. For example, because the thermal inertia of a pan and a lake are so different, the temperature of the pan and the lake water are likely to differ, resulting in errors. Also, the air temperature, humidity, and wind speed above the surface of a shore-based pan will, in general, differ from that above a lake, causing further complications.

A method that has the potential to obtain evaporation rates, E_m, that are more accurate than the pan method is to use a mass transfer equation of the form:

$$E_m = h_m(q_s - q_a) \tag{1.1}$$

where E_m is the water evaporation rate, h_m is a mass transfer coefficient, typically parameterized as a function of wind speed, u, q_s is the vapor concentration at the water surface, which is the saturation value at the water surface temperature, T_s, and q_a is the water vapor concentration of the bulk air above the water surface, which is the saturation value at the air temperature, T_a, multiplied by the relative humidity, ϕ. The parameterization of h_m in terms of wind speed u typically uses a wind speed measured at a ten meter height, u_{10}. This method for obtaining E_m is restricted by the accuracy of h_m in Eq. (1.1). However, the strength of this approach is that it enables estimates of E_m over the surface of the lake, unlike for the shore-based pan method. Using this approach, E_m is a function of T_s, T_a, u_{10}, and ϕ[6,5,7] (all other variables being essentially constant for typical conditions). Hence, limitations on the spatial and temporal measurements of E_m over a lake are restricted only by the resolutions of (T_s, T_a, u_{10}, ϕ). Of these four parameters, (T_a, u_{10}, ϕ) are easily obtained from meteorological stations such as the National Weather Service (NWS) Automated Surface Observing System (ASOS).[8] However, T_s is not provided by ASOS or other meteorological resources (e.g. the Meteorological Terminal Aviation Routine Weather Report [METAR]) due to the challenges associated with measuring T_s, described earlier in this section. Accordingly, it would be very useful to know the general functional form of the diurnal variation in T_s so that the temporal resolution on the MODIS measurements could be improved to match that

of the ASOS system.

A diurnal function of lake surface temperature would be useful to better understand gas exchange between the air-water interface. This mass transfer occurs via multiple physical processes, such as penetrative convection, wind shearing, and bubbling.[9] The literature has focused much of its attention in this field on variations due to the turbulent mixing by wind.[9,10] However it is not uncommon for buoyancy to drive mixing in inland bodies of water. One example would be during the evening when the wind is mild over a lake. In cases such as this, the evaporative and sensible heat fluxes can result in large variation in T_s which drive buoyant mixing of the surface layer with the quiescent lower layers. Researchers interested in understanding diurnal habits of aquatic life, or performing water quality studies could benefit from a better understanding of these diurnal mixing events.

The goal of this work is to determine the functional form of the diurnal variation of T_s which, to the authors' knowledge, has not been developed to date. Herein a one-dimensional model of a lake is used in combination with T_s measurements from Aqua and Terra, and measurements of (T_a, u_{10}, ϕ) from ASOS to calculate the diurnal surface temperature variation on the five major lakes in the Savannah River Basin, Lakes Jocasse, Keowee, Hartwell, Russell, and Thurmond. Using data obtained from these simulations, the general functional form for the diurnal variation in T_s is developed, and lake to lake variation in the average diurnal cycle of T_s is examined.

1.3 Area Description

The Savannah River Basin (SRB) is an inter-connected series of lakes and reservoirs located in the United States in the north-western corner of South Carolina. The Savannah River begins as it flows out of Lake Hartwell. The primary tributaries of the Savannah River are the Tugaloo river from Georgia, and the Seneca River from South Carolina. The Savannah River generally flows from northwest to southeast along the border between South Carolina and Georgia. There are eight lakes and reservoirs typically included within the SRB: Lake Jocassee, Lake Keowee, Lake Hartwell, Lake Russell, Lake Thurmond, Lake Toxaway, Bad Creek Reservoir, and Lake Issaqueena. Estimated volumes of these lakes based on average depth and surface area information from the United States Army Corps of Engineers (USACE) are 3.2 km^3, 1.4 km^3, 1.6 km^3, 1.3 km^3, 3.2 km^3, 0.01 km^3, 0.01 km^3, and 0.002 km^3. Thus, this work focuses on the five major lakes in the SRB, Lakes Jocassee,

Keowee, Hartwell, Rusell, and Thurmond. A map of the SRB presenting these five lakes is shown in Fig. 1.1.

These five reservoirs are used for recreation, power generation, and consumption by many residents of upstate South Carolina. From upstate SC to Hilton Head, SC, the SRB and its rivers provide drinking water to more than 1.5 million people each year. Additionally, industrial facilities consume water from this basin. The demands on the fresh water content in the SRB can be exceedingly high, especially in times of drought. Three of the reservoirs, Lakes Hartwell, Russell, and Thrumond, are maintained by USACE and used for hydroelectric power generation. Lakes Keowee, Jocassee, and Bad Creek Reservoir contain hydroelectric stations operated by Duke Energy. Additionally, Lake Keowee provides cooling water to the Oconee Nuclear Station (ONS).

The five lakes considered in this work are all warm monomictic lakes based on mixing classification. Generally, this means that the lakes get cold enough in the winter to mix thoroughly; however, they do not get cold enough to freeze over. This type of lake is common throughout many areas of the United States. Thus, the diurnal function developed in this work has the potential to be applicable to many more lakes than those considered herein.

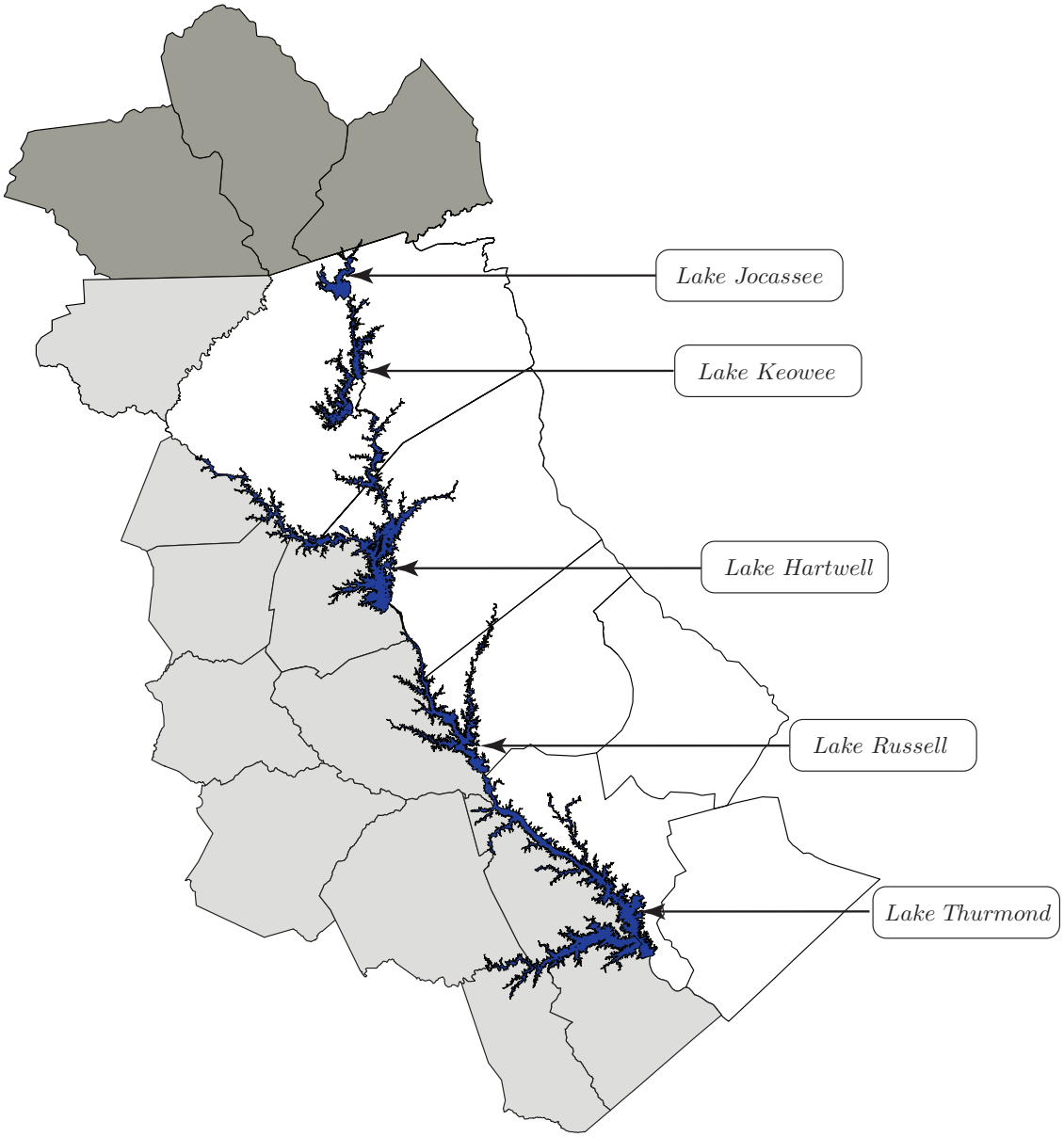

Figure 1.1: Map of the five major lakes considered in this work in the Savannah River Basin. South Carolina counties are shown in white, North Carolina in a darker gray, and Georgia in a lighter gray. Information provided by the South Carolina Department of Natural Resources.

Chapter 2

Methods

The one-dimensional transient model of the lake used here was developed by applying conservation of energy at the water surface, and within the mixed layer, and applying a turbulent kinetic energy balance within the mixed layer. These equations were solved to obtain T_s in the the time intervals between the Aqua and Terra measurements of T_s, herein referred to as T_{sat}. The resulting simulation T_s was then averaged over all the days investigated to obtain an average functional form for the diurnal variation for each lake. The measurements used in this analysis, the equations describing these energy balances, and the simulation algorithm, are presented below.

2.1 ASOS measurements of ambient parameters

Ambient atmospheric measurements of relative humidity, ϕ, air temperature, T_a, and wind speed, u, were available every hour from multiple ASOS stations in the SRB.[8] A map of the SRB with the available ASOS stations is shown in Fig. 2.1. The station closest to the center of each lake was used for the simulation of that lake. Thus, the Oconoee County Regional Airport (KCEU) was used for Lakes Keowee and Jocassee, the Anderson Regional Airport (KAND) was used for Lakes Hartwell and Russell, and the Augusta Regional Airport (KAGS) was used for Lake Thurmond. Ideally the ambient parameters would be measured on the lake; however historical measurements were not available over the desired time interval.

ASOS weather stations use a fully automated hygro-thermometer which uses a resistive temperature device (RTD) to measure T_a. The reported working range of the device is -62 to 54

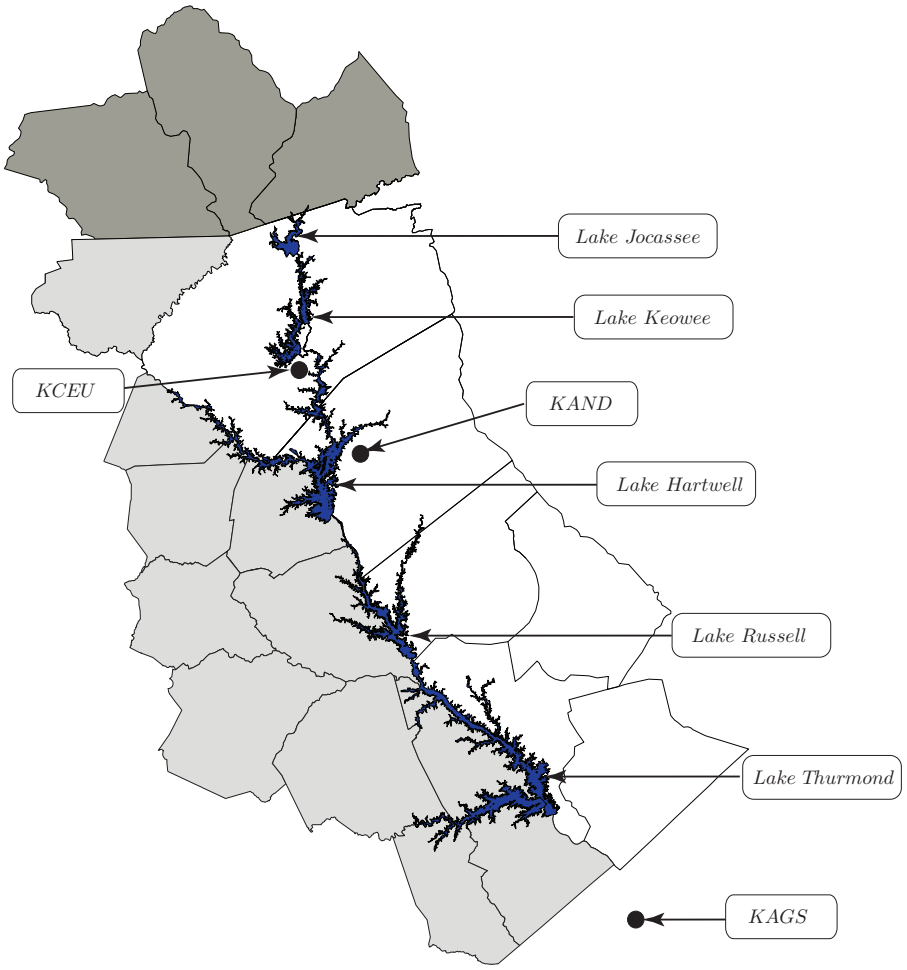

Figure 2.1: Map of the five major lakes and three weather stations considered in this work in the Savannah River Basin. South Carolina counties are shown in white, North Carolina in a darker gray, and Georgia in a lighter gray. Information provided by the South Carolina Department of Natural Resources.

°C with a maximum error of $\pm 1 - 2$ °C depending on where in the range the measurement lies. The resolution is 0.06 °C.[8] The dew point temperature is measured using this device in combination with a heat pump, a small LED light and a mirror. The mirror is cooled until the reflection of the LED light is impaired by condensate on the mirror. The measurement of the temperature of the mirror is then taken to be the dew point temperature. The relative humidity is calculated using the wet and dry bulb temperatures.[8]

Wind measurements are made using a rotating cup anemometer and a simple wind vane in the ASOS system. Wind measurements are generally taken at 10 m above ground, but some variation is allowed based on site specific restrictions. The reported working range of the device is 0 to 64 m/s, with a maximum error of ±1 m/s, and a resolution of 0.5 m/s.[8]

2.2 MODIS measurements of surface temperature

The Moderate Resolution Imaging Spectroradiometer (MODIS) device on the two satellites, Aqua and Terra, is used to calculate surface temperature all over the world. The satellites image 36 spectral bands ranging from 0.4 μ m to 14.4 μ m with a maximum spatial resolution of 1 km. The surface temperature measurements utilized in this work come from a level 3 product created via post-processing of these spectral images.[4] The maximum error under typical conditions has been shown to be ±0.5 K; however in the presence of heavy aerosol loading the maximum error can move up to ±1 K.[11]

To try and minimize error in MODIS measurements, only pixels which contained only water were included in this work. This means any pixels where some of the lake shore or any islands were present were rejected. Pixels were accepted or rejected visually using GIS data from USACE. There were 12 pixels available on Lake Hartwell, 6 pixels available on Lake Jocassee, 1 pixel available on Lake Keowee, 3 pixels available on Lake Russell, and 19 pixels available on Lake Thurmond.

2.3 MODIS measurements of cloud cover

In addition to T_{sat} measurements, MODIS provides cloud cover measurements. The level 2 cloud fraction product was used in this work for cloud cover index.[4] All measurements within a box bounded by 4 GPS coordinates were averaged to give the measurement at each satellite overpass

time. The bounding box coordinates for each lake are shown in Table 2.1. Cloud cover was linearly interpolated between available measurements in this work.

Lake	Max Longitude	Min Longitude	Max Latitude	Min Latitude
Jocassee	-82.90	-83.00	35.07	34.95
Keowee	-82.85	-83.00	34.95	34.68
Hartwell	-82.69	-83.10	34.30	34.70
Russell	-82.40	-82.80	34.30	33.85
Thurmond	-82.15	-82.60	33.85	33.65

Table 2.1: GPS bounding boxes for each lake used in cloud cover measurements.

2.4 USACE measurements of bulk temperature

Measurements of the bulk temperature of the lake, T_b, were available for Lake Hartwell from the United States Army Corps of Engineers (USACE) 6-12 times a year. To facilitate the simulations, a continuous function for T_b was needed. Accordingly, each year of T_b data was fit by a third order polynomial and these were concatenated. To ensure the polynomial fits were continuous, the initial point of each year was forced to match the final point of the polynomial curve for the previous year. For five years (2004, 2006, 2007, 2013, and 2014), the temporal resolution of the measurements was insufficient to create a good fit. For these years, the average yearly trend from all of the T_b measurements was used, with a vertical offset based on the final temperature from the previous year. The developed curve fit was used herein for the simulation. The curve fit of T_b and the USACE data used in developing this curve are shown in Fig. 2.2. Herein the T_b function defined for Lake Hartwell was used for the other four lakes since historical measurements of T_b were unavailable for them. The consequences of this approach will be discussed in Chapter 4.

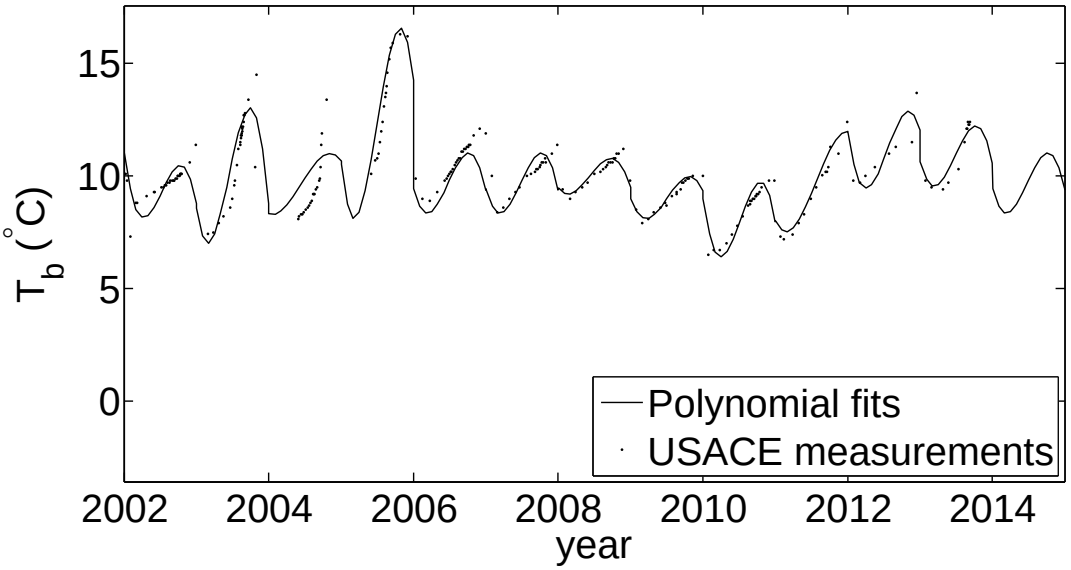

Figure 2.2: Lake Hartwell bulk temperature measurements, T_b versus year. The data from USACE is denoted by the points, and the solid line is the conctenated polynomial curve created from the data.

2.5 Conservation of energy at the surface

The surface energy balance was calculated following the method presented by Alcantara *et al.*[7] The primary energy fluxes which contribute to the net heat flux at the surface, Φ_N, were the incident short wave radiation, Φ_s, the long wave radiation, Φ_{ri}, the sensible heat flux, Φ_{sf}, and the latent heat flux, E_e.[7,12] Thus, neglecting the effects of precipitation, chemical and biological reactions, and kinetic energy, the net energy flux at the lake surface was[12,7]

$$\Phi_N = \Phi_s(1 - A) - (\Phi_{ri} + \Phi_{sf} + E_e) \tag{2.1}$$

where A was the albedo of water, and E_e was the energy flux due to evaporation or, rarely, condensation. When Φ_N was positive, there was a net flux of energy into the lake.

The incident short wave radiation was:

$$\Phi_s = b_1\Phi_0(\sin d)^{b_2}(1 - 0.65C^2) \tag{2.2}$$

where the two calibration parameters b_1 and b_2 were determined from radiometer data to be 0.79 and 1.15 respectively,[7] Φ_0 was the solar constant, 1390 Wm^{-2}, d was the solar elevation angle, and C was the cloud cover index which was obtained from MODIS L2 data.[12,7,4] The solar elevation angle was calculated using the method presented by Reda *et al.*.[13]

The longwave radiation flux was:

$$\Phi_{ri} = \epsilon\sigma T_s^4(0.39 - 0.05e_a^{1/2})(1 - \lambda C) + 4\epsilon\sigma T_s^3(T_s - T_a) \tag{2.3}$$

which was positive when there was a loss of energy from the lake, and where $\epsilon = 0.97$ was the thermal infrared emissivity of water,[7] σ was the Stefan-Boltzmann constant, λ was the Reed correction factor,[14,7,15] and e_a was the partial pressure of water vapor,

$$e_a = \phi e_{sat}(T_a) \tag{2.4}$$

where e_{sat} was the saturated vapor pressure in mb using the equation due to Lowe:[16]

$$\begin{aligned} e_{sat}(T) = {} & 6984.505294 - 188.9039310 \times T + 2.133357675 \times T^2 \\ & - 1.288580973 \times 10^{-2} \times T^3 + 4.393587233 \times 10^{-5} \times T^4 \\ & - 8.023923082 \times 10^{-8} \times T^5 + 6.136820929 \times 10^{-11} \times T^6 \end{aligned} \tag{2.5}$$

where T was temperature in K.

The sensible heat flux was calculated using the equation

$$\Phi_{sf} = \rho_a c_{p_a} c_H u_{10}(T_s - T_a) \tag{2.6}$$

where ρ_a was the air density, c_{p_a} was the specific heat capacity of air, u_{10} was the wind velocity ten meters above the surface, and c_H was a coefficient of turbulent exchange.[7,6]

The energy flux due to evaporation was:

$$E_e = \rho_a c_E h_{fg} u_{10}(e_{sat}(T_s) - \phi e_{sat}(T_a))\frac{0.622}{P_a} \tag{2.7}$$

where h_{fg} was the latent heat of vaporization for water, P_a was the atmospheric air pressure, and c_E was another coefficient of turbulent exchange.[7, 6]

The following assumptions were made in the development of Eqs. (2.1) - (2.7). First, the electromagnetic spectrum was lumped into two separate bands (short wave and long wave radiation), which assumed that at some critical wavelength the spectral response of water experiences a step change. This approach is commonly used in limnology.[7] Next, the latent heat flux and sensible heat flux were assumed to be functions of (T_s, T_a, u_{10}, ϕ), with the remaining complexity being summed up in the turbulent exchange coefficients, C_H and C_E (Eqs. [2.6] and [2.7]). Finally, the short wave radiation was only included during the day since it has been shown that its effects were negligible at night;[7] the other terms in Eq. (2.1) were included at all times in the day and night.

2.6 Conservation of energy of the mixed layer

Most lakes exhibit some degree of thermal stratification, and the temperature distribution in a stratified lake is typically described by three distinct layers where lateral temperature variations are ignored: the mixed layer, the metalimnion, and the hypolimnion. The mixed layer, also called the epilimnion, is the region closest to the surface in which buoyant forces and/or convective forces mix the layer, yielding a layer of finite thickness where the temperature is essentially uniform. Hence, in the simulations presented here, the temperature of the mixed layer and the surface temperature are made equal and are both referred to as T_s. The metalimnion, or thermocline, is the region of sharp temperature change in the lake. The hypolimnion is the quiescent region of the lake which changes temperature slowly from season to season. The temperature of this layer is referred to as the bulk lake temperature, T_b.

Surface temperature on Lakes Jocasse, Keowee, Hartwell, Russell, and Thurmond located in northwestern South Carolina were simulated. These lakes are warm monomictic lakes, having a single mixing season which lasts through the winter.[17] As shown in Fig. 2.3, a one-dimensional transient mixed layer model was used to simulate these lakes where each lake was divided into two uniform temperature regions: the mixed layer at temperature T_s and the hypolimnion at a temperature T_b. Data on the change in temperature with depth in the thermocline is often used in lake models to increase the simulation accuracy. However such data was not available and herein the

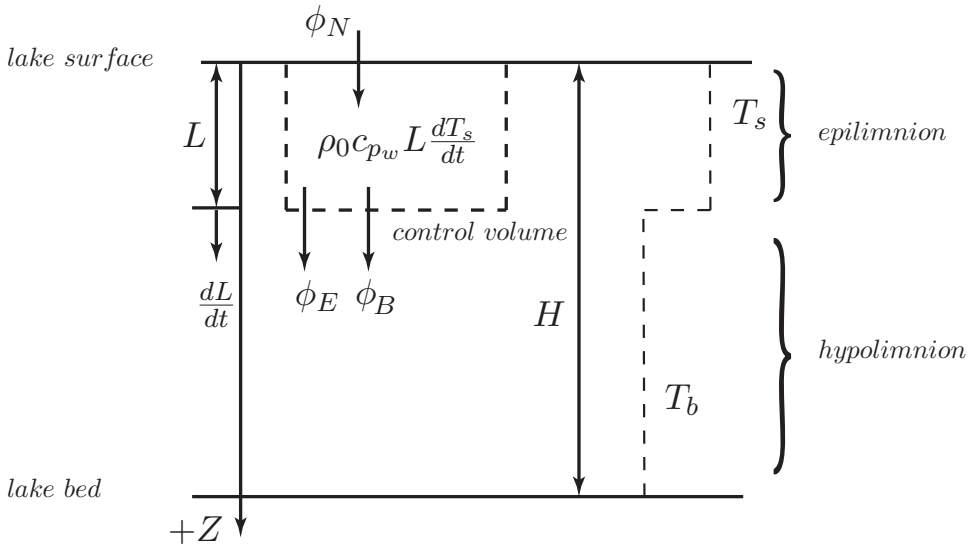

Figure 2.3: Control volume of the mixed layer where L was the effective mixed layer depth, H was the effective lake depth, ρ_0 was the reference water density, c_{p_w} was the specific heat capacit of water, T_s was the mixed layer temperature, T_b was the bulk lake temperature, Φ_N was the net surface flux, and Φ_E was the energy flux due to entrainment.

thermocline was modeled as having a step change in temperature. With this assumption in mind, L from the simulation should be thought of as an effective mixed layer depth for the whole lake rather than a precise measure of mixed layer depth for any individual point of the lake. The control volume used for this model is shown in Fig. 2.3. The general equation for conservation of energy of the control volume was:

$$\rho_0 c_{p_w} L \frac{dT_s}{dt} = \Phi_N - \Phi_E - \Phi_B \tag{2.8}$$

where ρ_0 was the reference water density, c_{p_w} was the specific heat capacity of water, L was the mixed layer depth and the energy flux due to entrainment, Φ_E was calculated using the equation:

$$\Phi_E = \rho_0 c_{p_w} (T_s - T_b) \frac{dL}{dt} \tag{2.9}$$

The energy flux due to heat transfer to the hypolimnion, Φ_B was calculated using the equation:

$$\Phi_B = -\rho_0 c_{p_w} (H - L) \frac{dT_b}{dt} \tag{2.10}$$

where H was the lake depth. Equation (2.9) corresponded to the energy required to change the temperature of the entrained fluid to match T_s, and Eq. (2.10) corresponded to the energy required to change T_b. Combining Eqs. (2.8), (2.9), and (2.10) and rearranging terms, yielded an equation for the time rate of change of T_s:

$$\frac{dT_s}{dt} = \frac{\Phi_N}{c_p \rho_0 L} - \frac{(T_s - T_b)}{L} \frac{dL}{dt} - \frac{(H - L)}{L} \frac{dT_b}{dt} \tag{2.11}$$

2.7 Turbulent kinetic energy budget

Since Eqs. (2.1) and (2.11) had three unknowns ($\frac{dT_s}{dt}$, L, and $\frac{dL}{dt}$) these two equations were not a closed system. To close the system, the turbulent kinetic energy budget was used. The mixed layer depth, L, increased due to wind and buoyant mixing, and these effects were modeled in the turbulent kinetic energy budget as a change in potential and kinetic energy of the entrained water from the hypolimnion. As water was entrained, the control volume increased in size, changing the center of gravity of the control volume, and the velocity of the entrained fluid was accelerated to the turbulent state of the mixed layer.[18]

The turbulent kinetic energy budget was calculated following the method presented by Fischer *et al.*[18] The equation for the time rate of change of the mixed layer depth was

$$\frac{dL}{dt} = \frac{C_k^f q_*^3}{C_T q_*^2 + \alpha (T_s - T_b) g L} \tag{2.12}$$

where C_k^f was the internal losses coefficient, α was the volumetric thermal expansion coefficient of water, g was the acceleration due to gravity, C_T was the kinetic energy coefficient, and q_* was the combined velocity scale:[18]

$$q_*^3 = (w_*^3 + \eta^3 u_*^3) \tag{2.13}$$

where η was the net efficiency of introduction of kinetic energy at the surface, u_* was the shear velocity, modeled as:

$$u_* = \sqrt{\frac{\rho_a C_D u_{10}^2}{\rho_0}} \tag{2.14}$$

C_D was the drag coefficient, modeled as:[7]

$$C_D = 0.00052 u_{10}^{0.44}, \tag{2.15}$$

and w_* was the buoyant velocity scale,[18,19]

$$w_* = \left(\frac{\alpha g \Phi_N L}{C_{p_w} \rho_0} \right)^{1/3}. \tag{2.16}$$

The constants C_T, and η were set to 0.5, and 1.75 respectively as recommended by Fischer *et al.*[18]

Preliminary simulations showed that the solution was most sensitive to the value of the internal loses coefficient, C_k^f, which determines how quickly the mixed layer responds to a change in ambient parameters. The default value of $C_k^f = 10$ was used in the simulation; however, in certain instances the simulations were iterated over C_k^f to decrease the errors in the simulations (simulation error is defined below). The method used to set values for C_k^f is described in Section 2.9.

2.8 Winter Algorithm

The five lakes considered in this study are warm climate monomictic lakes experiencing overturn and complete mixing during the winter.[20] This corresponds to the seasonal mixed layer depth extending to the lake bottom, which the simulations presented herein predict. Since the lake is no longer stratified under such conditions, the assumptions used in the model described above are not valid. Specifically, during overturn T_s would remain essentially constant for the entire season since there is not enough energy on a diurnal time scale to significantly change the temperature of the entire bulk of the lake in a single day. However, from satellite measurements it is known that T_{sat} varies significantly during the course of a day in the winter and that T_{sat} deviates from bulk temperature measurements. Hence, a different simulation algorithm was used for the winter.

Other one dimensional transient models were examined such as utilizing a conduction in

stagnant water approach as proposed by Snider *et al.*[21] and Girgis *et al.*[22] However, these methods assume that the mixing effects of wind are negligible, which is not the case for lakes in the Savannah River Basin in the winter. The eddy coefficient hypothesis presented by Niiler *et al.*[23] was considered; however, this method depends greatly on empirically determined coefficients which would likely not be constant for the duration of the simulation. The mixed layer model presented by Spigel *et al.*[24] was considered; however, it required more knowledge of the development of the diurnal thermocline than was available for this study, such as thermocline thickness, inclination, and the existence of many thermoclines from previous history. The momentum balance method proposed by Imberger *et al.*[19] was considered as well; however, poor agreement was found between simulation results and the satellite measurements during overturn.

Here, the same method described in Sections 2.5 - 2.7 was used but with a constant effective mixed layer depth for the winter. When the simulation predicted overturn, L was set to a constant value which minimized the residual error between simulation results and satellite measurements. Herein a default value of 1.1 m was used for this constant; however, similar to C_k^f mentioned in the previous section, L was varied between satellite measurements to reduce error. This approach will be described more fully in Section 2.9. The winter start and end dates were chosen so as to minimize the simulation error at satellite measurements. These dates were November 15 and March 31 respectively, though the simulations were not overly sensitive to these dates.

2.9 Simulation Algorithm

Simulations were conducted from the summer of 2002 which was the earliest time at which two daily satellite measurements were available from both Aqua and Terra, and continued through the beginning of the summer of 2014. An assumed value for L based on the seasonal thermocline was used as an initial condition. The inputs consisted of four daily T_s measurements from Aqua and Terra, hourly ambient atmospheric conditions (T_a, and ϕ) from the nearest ASOS weather station to the lake being simulated.

Using u_{10} from weather stations yielded poor agreement with satellite measurements of T_{sat}. Accordingly, the simulations were iterated over u_{10} to minimize the root mean square (rms) deviation of T_{sat} from satellite measurements. It has been shown that u_{10} can vary significantly both temporally and spatially over bodies of water compared to land measurements.[25] The consequences

of this approach are presented in Chapter 4.

The details of the solution algorithm are presented below and are graphically illustrated in Fig. 2.4. An example of how the solution converged to a value for T_s between two satellite points is presented in Fig. 2.5. In the following description, t corresponds to the time since the first satellite measurement, t^1_{sat}, and is incremented in time steps of $\Delta t = 60$ sec.

For each pair of satellite measurements, the following process was used. First, the net flux at the surface was calculated using Eqs. (2.1) - (2.5). Next, $\frac{dL}{dt}$ was calculated using Eqs. (2.12) - (2.16). Then $\frac{dT_s}{dt}$ was calculated using Eq. (2.11). New values for T_s were then obtained using the equations:

$$T_s(t + \Delta t) = T_s(t) + \frac{dT_s}{dt}\Delta t \tag{2.17}$$

and

$$L(t + \Delta t) = L(t) + \frac{dL}{dt}\Delta t \tag{2.18}$$

The above process was repeated until t was equal to the time of the next satellite measurement, t^{n+1}_{sat}. As noted above, this process was repeated over a range of u_{10} to give a solution with the least deviation of the simulation from the satellite measurements. The approach for doing this was to run a simulation for u_{10} equal to 0 m/s and 20 m/s. An example of this is shown in Fig. 2.6. As this figure shows, both values of u_{10} yielded values of T_s at t^{n+1}_{sat} unequal to the satellite measurement; however the satellite measurement was between the two solutions. Thus, the next step was a straightforward iteration over u_{10} to find the converged solution, i.e. the solution where T_s at the second satellite measurement time was as close as possible to that second satellite measurement, i.e. where the error was minimized. Here, error was defined as:

$$T_{err} = \left| T_s(t^{n+1}_{sat}) - T^{n+1}_{sat} \right| \tag{2.19}$$

where $T_s(t^{n+1}_{sat})$ was the simulated temperature, and T^{n+1}_{sat} was the satellite measurement temperature. The next value of u_{10} was calculated using linear interpolation from the previous u_{10} and the

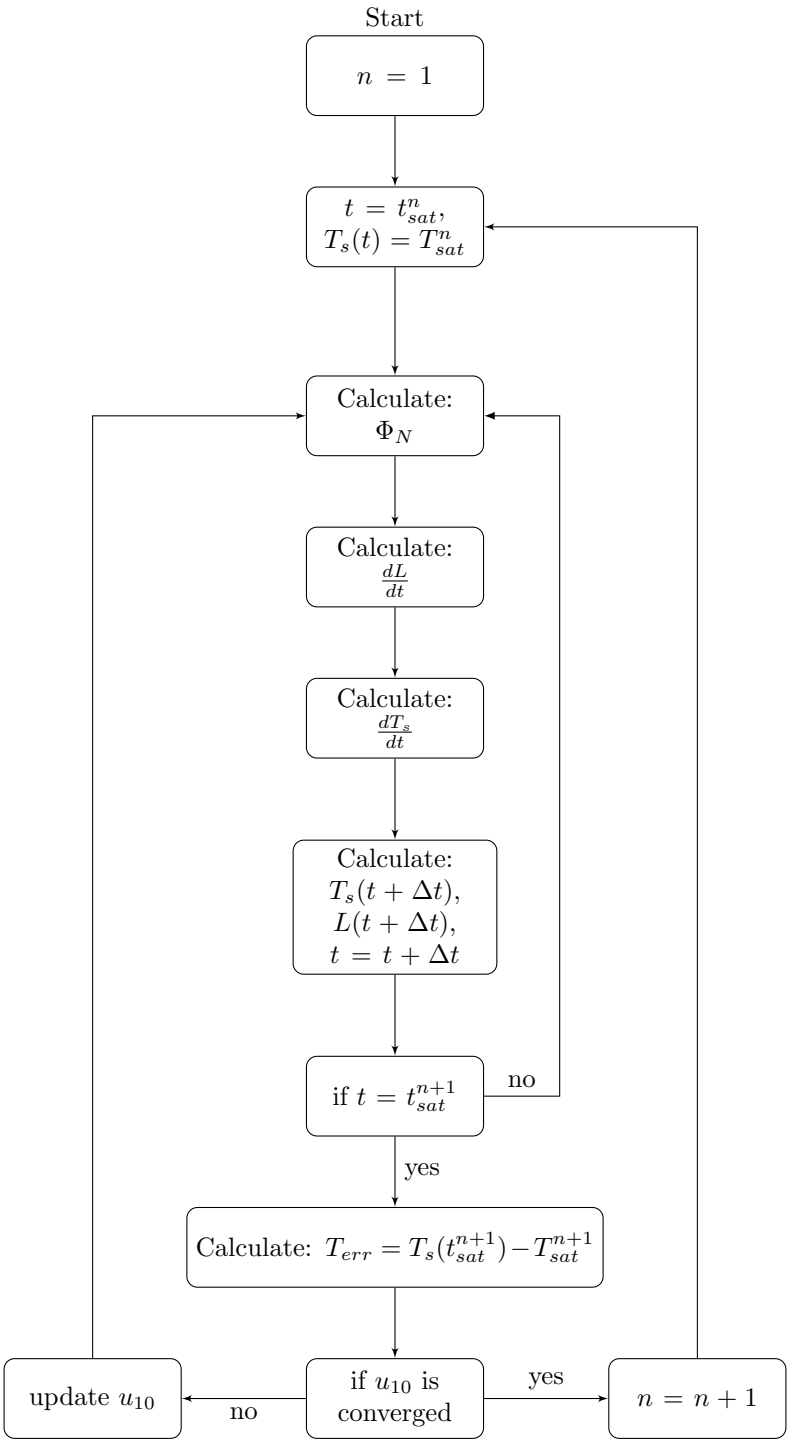

Figure 2.4: Simulation algorithm flow chart. Process starts at with the first pair of satellite measurements and continues until the simulation is complete.

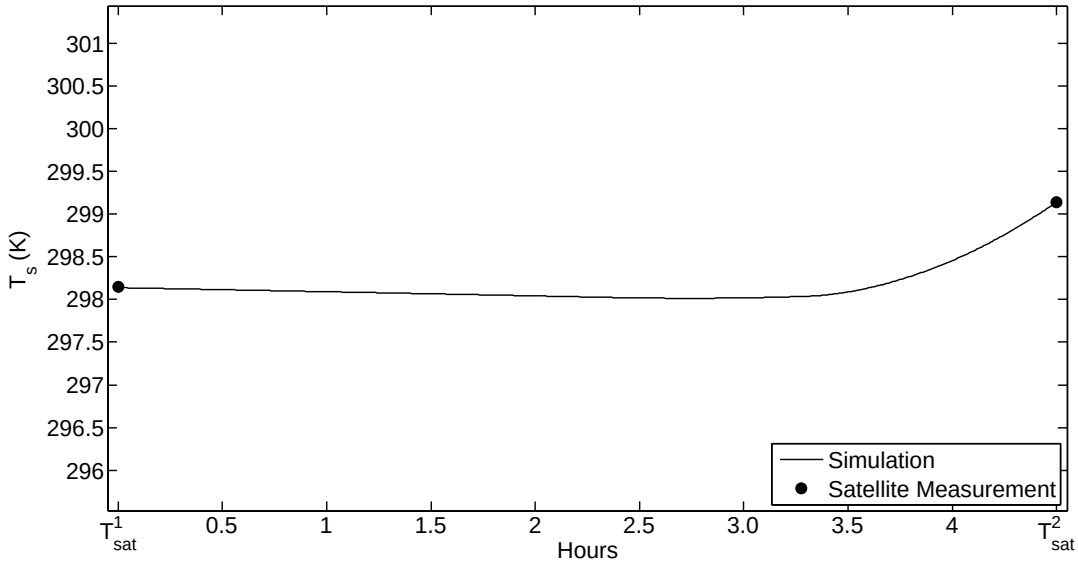

Figure 2.5: Example of converged solution for T_s versus time in hours between two satellite measurements.

u_{10} at which T_{err} was a minimum. The process was repeated until u_{10} was converged within 1/1000 m/s. Typical values of T_{err} ranged from 0 K to 2 K, with an average at less than 2 K.

In certain cases, the two initial values for u_{10} used ($u_{10} = 0$ m/s and $u_{10} = 20$ m/s) gave two values of T_s at t_{sat}^{n+1} that did not span the value of T_{sat}^{n+1}, as was the case in Fig. 2.6. An example of such a situation is shown in Fig. 2.7 where both of the simulations give values of T_s at t_{sat}^{n+1} that are less than the satellite measurement. Since T_s is typically monotonic in u_{10} (when all other parameters are held constant) a second parameter must be varied in these cases to adjust the range of possible solutions until the satellite measurement falls between the two simulated values (this assumes that for 0 m/s $< u_{10} <$ 20 m/s, which is a safe assumption for the lakes investigated here). The result of this is shown in Fig. 2.8 where C_k^f was varied to force upward the two T_s solutions shown in Fig. 2.7, thereby spanning the satellite measurement. In the spring, summer, and fall, C_k^f was used as the second parameter, forcing upward (or downward) the solutions for the initial guesses of $u_{10} = 0, 20$ m/s if necessary. However, as noted in Section 2.8, varying C_k^f in the winter does not significantly affect the solution and so L was used as the second parameter in the winter simulations, when necessary.

Once the satellite measured T_{sat} fell within the possible solutions for T_s, u_{10} was varied to force the T_s solution to hit the satellite point as discussed earlier. The solution with the minimum

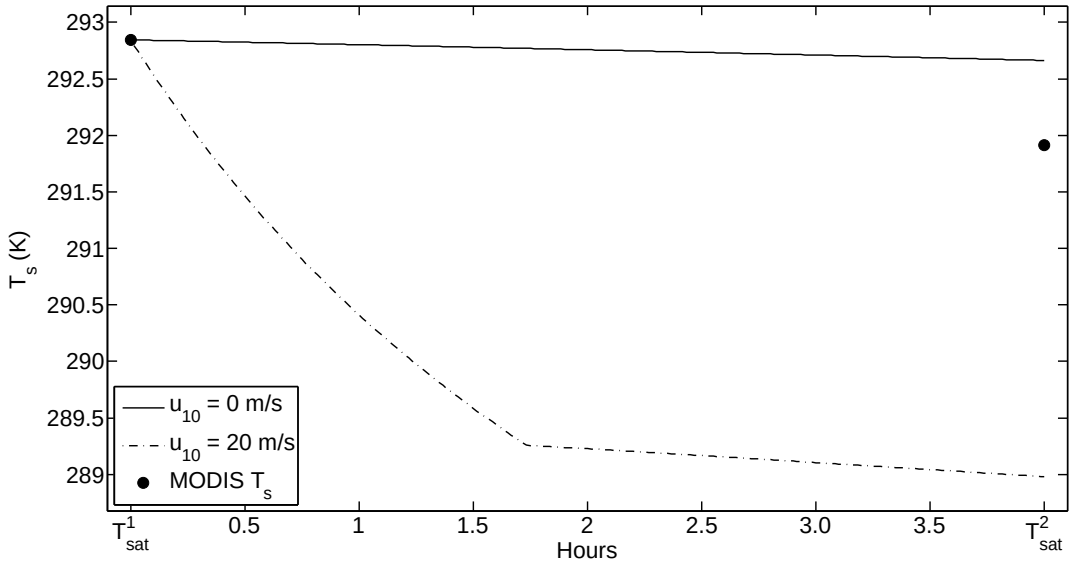

Figure 2.6: Simulation for T_s using $u_{10} = 0$ m/s and $u_{10} = 20$ m/s, 1st example. Note that one simulation gives a final value of T_s greater than T_{sat}^2 and the other gives a final value of T_s less than T_{sat}^2. This enables a straightforward iteration over u_{10} to find the converged solution.

residual error calculated using Eq. (2.19) was selected and the simulation then proceeded to the next satellite point and set of ambient parameters. This algorithm was performed for the five lakes considered in this work: Lakes Jocassee, Keowee, Hartwell, Russell, and Thurmond.

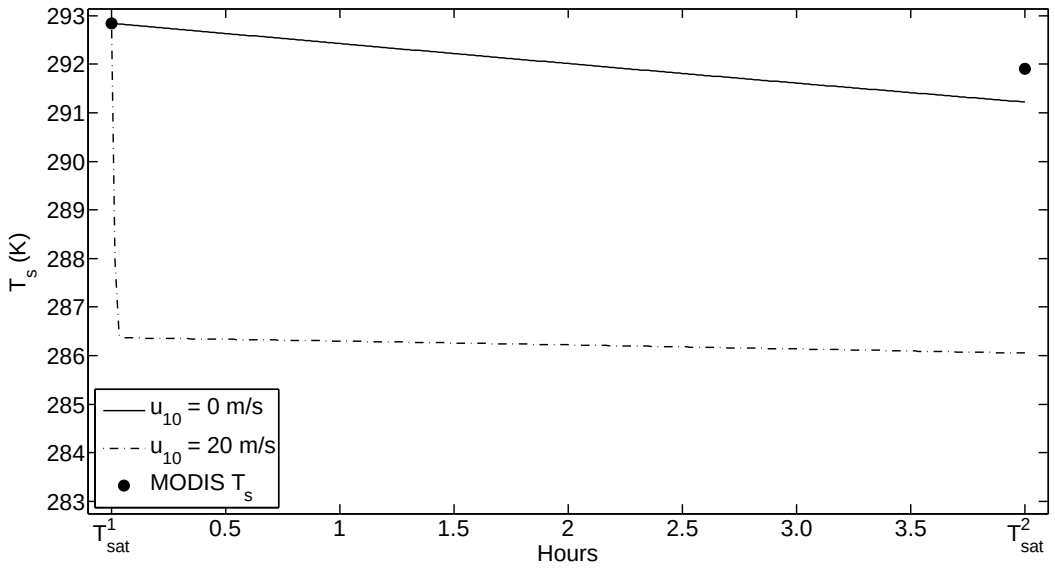

Figure 2.7: Simulation for T_s using $u_{10} = 0$ m/s and $u_{10} = 20$ m/s, 2nd example. Note that both simulations give values of T_s less than T^2_{sat}. Thus T^2_{sat} does not reside in the range of possible solutions for these two wind speeds.

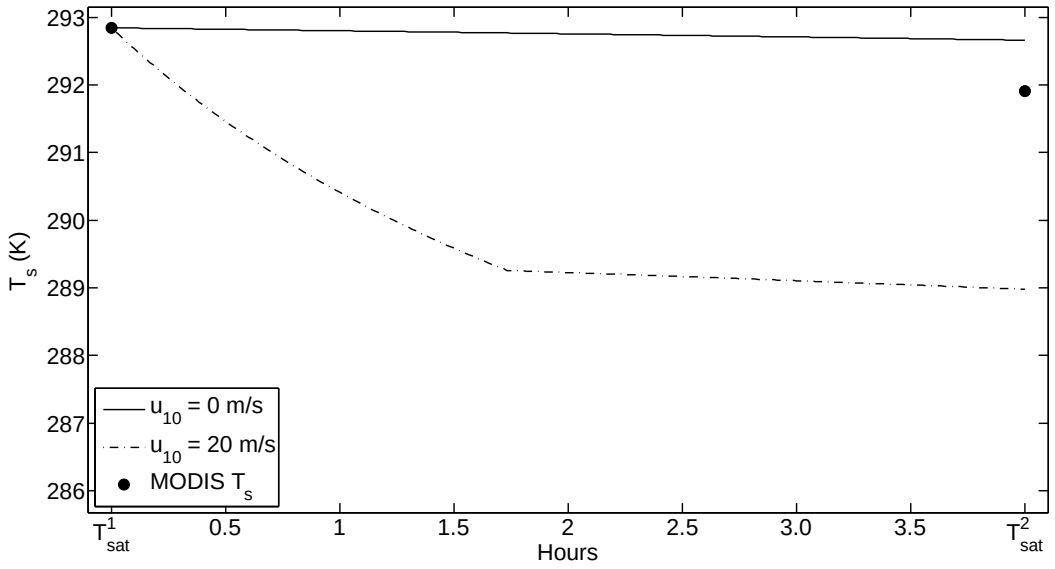

Figure 2.8: Simulation for T_s using $u_{10} = 0$ m/s and $u_{10} = 20$ m/s after iterating over C^f_k. Note that one simulation gives $T_s > T^2_{sat}$ and the other gives $T_s < T^2_{sat}$. This enables a straightforward iteration over u_{10} to find the converged solution.

Chapter 3

Results

Surface temperature was calculated between MODIS measurements of T_{sat} for the five major lakes in the Savannah River Basin. Surface temperatures were simulated from July 2002 (the first time where all four daily T_{sat} measurements from MODIS were available) to July 2014 for Lakes Hartwell, Keowee, and Russell. Lakes Jocassee and Thurmond were simulated from 2006-2014 due to limited availability of KCEU measurements (T_a and ϕ) for earlier years. The simulations of T_s for Lakes Hartwell, Jocassee, Keowee, Russell, and Thurmond are presented in Fig. 3.1, revealing the annual variation in T_s.

Figure 3.1 shows some instances where the simulations deviate significantly from any of the measured values. These events were rare; those instances where T_s deviated from the entire max/min for the satellite data set occured less than 0.1% of the time. The cause of this is described in Chapter 4. These points were omitted and therefore had no impact on the results presented here.

To quantify how well the simulation results matched the satellite measurements, the root mean square deviation of the simulations from the satellite measurement was computed:

$$T_{rms} = \left[\frac{1}{N} \sum_{n=1}^{N} (T_s^n - T_{sat}^n)^2 \right]^{1/2} \tag{3.1}$$

where n was the satellite measurement number, T_{sat}^n is n^{th} the satellite surface temperature measurement, T_s^n was the simulation surface temperature at the time of the n^{th} satellite measurement, and N was the total number of satellite measurements. For Lakes Hartwell, Jocassee, Keowee, Russell,

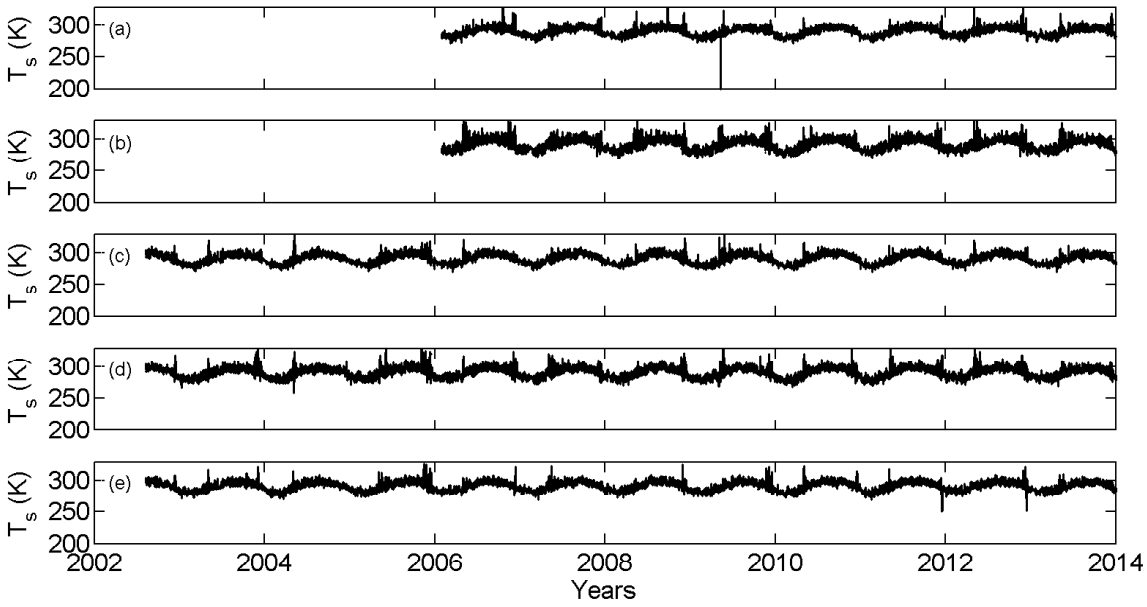

Figure 3.1: Lake surface temperature, T_s in K versus time, t in years from simulation results for Lakes (a) Jocassee, (b) Keowee, (c) Hartwell, (d) Russell, and (e) Thurmond.

and Thurmond, T_{rms} was 1.4 K, 1.5 K, 2.7 K, 2.0 K, and 1.5 K, respectively.

To show the behavior of T_s in greater detail the results for Lake Hartwell are now presented. Figure 3.2(b) presents the simulations of T_s for the entire 12 year time period considered for Lake Hartwell, along with the satellite measurements of T_{sat}. Because of the density of the data, the satellite measurements are difficult to see in Fig. 3.2(b); these data are presented alone in Fig. 3.2(a).

As Fig. 3.2 shows, the trend from year to year was periodic. To show the simulations of T_s more clearly, the simulations from a sample year, 2011, are presented in Fig. 3.3, revealing the seasonal variation. Starting on January 1, 2011 (Day 0 in Fig. 3.3), T_s dropped until it reached a minimum around the middle of February, then steadily increased until it reaches a maximum in the middle of August, and finally began to decrease until the end of the year.

To focus on the diurnal variation, the simulations for a sample week are shown in Fig. 3.4 which shows that the largest T_s was generally found in the early afternoon, and the coolest slightly before sunrise. To obtain the diurnal variation in the surface temperature using the entire data set, a non-dimensional temperature T^* was developed so as to prevent seasonal trends from obscuring

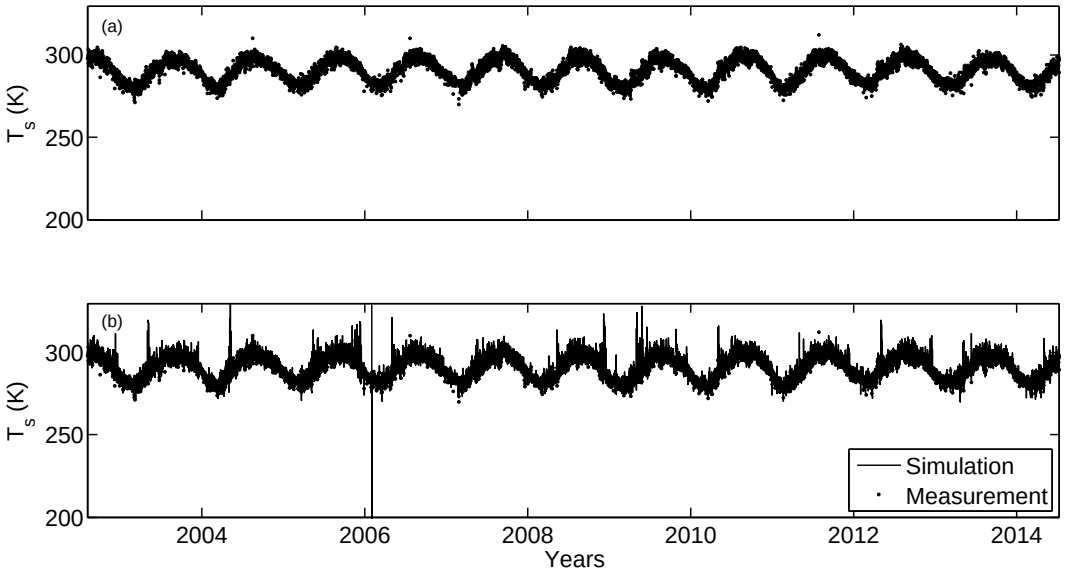

Figure 3.2: Lake Hartwell T_s in K versus date in years. (a) Satellite measurements only. (b) Satellite measurements and simulation results.

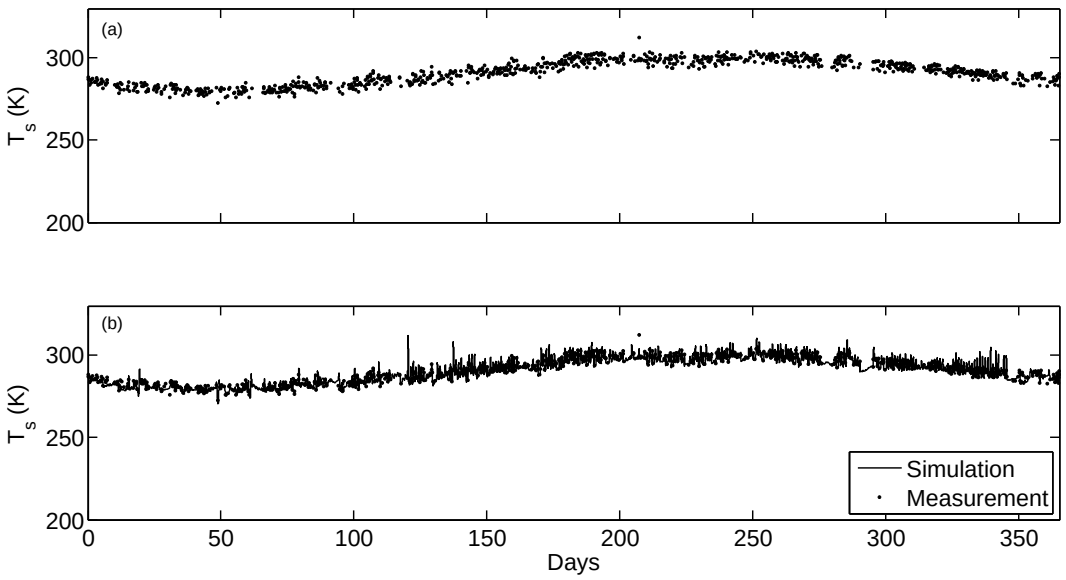

Figure 3.3: Lake Hartwell T_s, in K versus day number from simulation results for 2011. Day 0 corresponds to January 1, 2011, and Day 365 corresponds to December 31, 2011. (a) Satellite measurements only. (b) Satellite measurements and simulation results.

the diurnal trend:

$$T^* = \frac{T - T_{min}}{T_{max} - T_{min}} \tag{3.2}$$

Here the subscripts min and max corresponded to the minimum and maximum values of each individual day. A time trace of T_s^* for a sample week is presented in Fig. 3.5. To further prevent obscuration of the diurnal trend by the seasonal trend, a non-dimensional time scale t^*, was used to define time based on local sunrise and sunset time:

$$t^* = \begin{cases} \frac{24+t-t_{set}}{24-t_{set}+t_{rise}} + 1 & 0 \leq t < t_{rise} \\ \frac{t-t_{rise}}{t_{set}-t_{rise}} & t_{rise} \leq t \leq t_{set} \\ \frac{t-t_{set}}{24-t_{set}+t_{rise}} + 1 & t_{rise} \leq t \leq t_{set} \end{cases} \tag{3.3}$$

In Eq. (3.3) t_{rise} and t_{set} were sunrise and sunset in hours since midnight local time. Hence $t^* = 0$ at sunrise on the current day; $t^* = 1$ at sunset; and $t^* = 2$, its maximum value, at sunrise the following day.

This scaling has a few key advantages over simply using local time. The growth of a new thermocline begins at sunrise when the surface layer begins to absorb solar energy. Using this scaling ensured that this growth began at the same t^* every day, which was useful for averaging purposes across multiple days. Additionally, since solar position and length of day were key parameters in modeling the diurnal variation of T_s, averaging the results from different parts of the year using t instead of t^* may have concealed diurnal trends that were common for the whole year, a further advantage of using t^*. A plot of T^* versus t^* obtained by averaging over every day of the 12 year simulation period is shown in Fig. 3.6 for Lake Jocassee, Fig. 3.7 for Lake Keowee, Fig. 3.8 for Lake Hartwell, Fig 3.9 for Lake Russell, and Fig. 3.10 for Lake Thurmond. The average diurnal cycles shown in Figs. 3.6- 3.10 show some noise. This noise is a result of averaging over t^* which resulted in some bins having less data than others due to the changes in sunrise and sunset times.

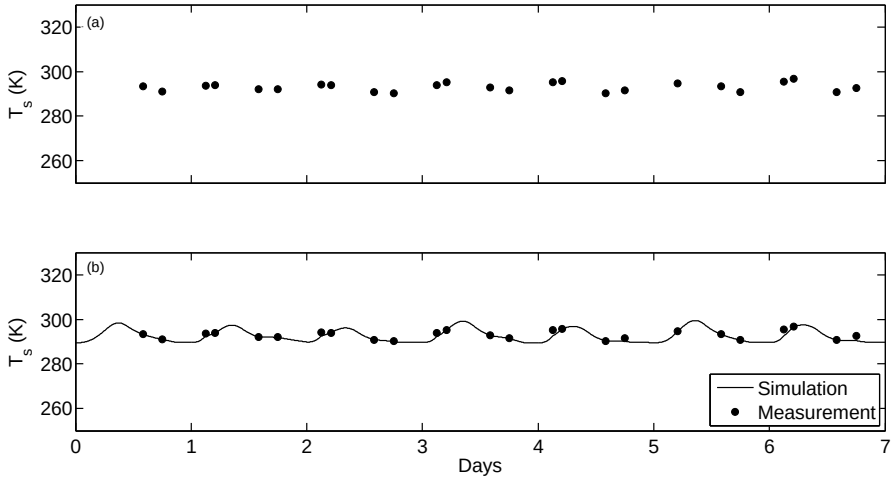

Figure 3.4: Surface temperature, T_s, in K versus day from simulation results for a typical week. (a) Satellite measurements only. (b) Satellite measurements and simulation results.

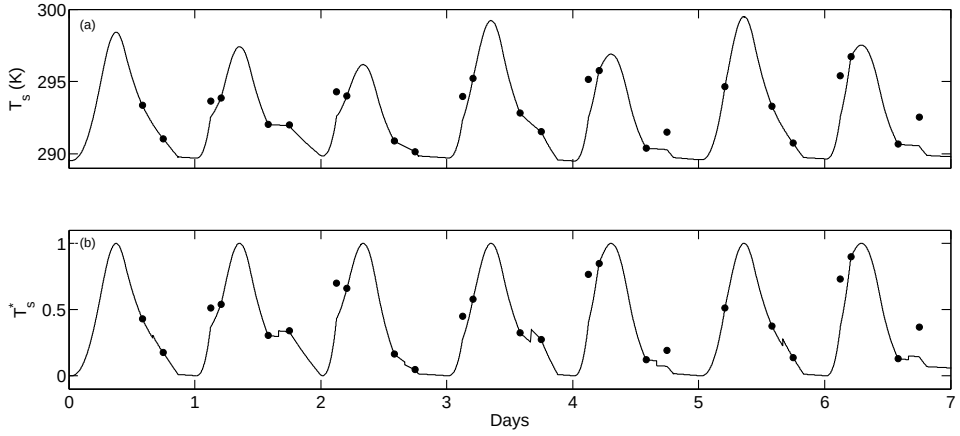

Figure 3.5: Non dimensional temperature scaling applied to a sample week. (a) T_s in K versus time in days from the simulation results for a sample week. (b) Surface temperature transform, T_s^*, from Eq. (3.2) of the sample week in part (a).

The abrupt change in temperature observed in Figs. 3.6- 3.10 between $t^* = 0.4$ and $t^* = 0.5$ is an artifact of the simulation algorithm. The algorithm iterates over wind speed between satellite measurements of T_{sat}. There are times (primarily in the middle of the afternoon) where two satellite measurements are very different. When disagreement with T_{sat} is observed, the simulation will iterate over a second parameter which causes L to change rapidly, which results in T_s changing quickly. An example of this result is shown in Fig. 3.11. The second order discontinuity in T_s predicted by the simulation in day 6 of Fig. 3.11 likely does not predict the real variation in T_s. This difference could come from any number of factors which are not considered in this work. Examples include movement of a front into the region, a sudden change from clear skies to very overcast conditions and precipitation. Using precipitation as an example, a summer storm coming in after the first daytime T_{sat} measurement would cause the second daytime T_{sat} measurement to drop significantly. This adds uncertainty to the simulation which, even after averaging over many days, still appears in the diurnal average. A similar step is observed in the additional averaging methods discussed in Chapter 4. Additional discussion of how the simulation predicts Fig. 3.11 is provided in Chapter 4.

The next step in this work was to develop a functional description of the diurnal variation of lake surface temperature. Moreover, the desire was to develop a function with four fitting parameters so that the known surface temperatures obtained from the four daily satellite measurements which can be obtained from Aqua and Terra could be used to develop an individual equation for any given day. To do this, the Fourier transform of the data presented in Figs. 3.6- 3.10 were obtained. The result of this transform on Lake Hartwell is presented in Fig. 3.12 which shows that the primary dimensionless frequency, f_1^*, using the t^* scaling is 0.5, which was expected since t^* had a fixed period of 2. Additionally, the second, third, and fourth harmonics were at $f^* = 1, 1.5$ and 2, where k is the harmonic. Accordingly, the average diurnal cycles shown in Fig. 3.8- Fig. 3.10 may be fit to:

$$T^*(t^*) = \sum_{k=1}^{4} \left[B_k sin\left(2\pi f_k^* t^* - \psi_k\right)\right] - D \tag{3.4}$$

B_k is the amplitude of each Fourier component, ψ_k is the phase shift for each Fourier component, and D is a DC offset. Of course Eq. (3.4) actually has nine unknown constants, not four. Iterative

solution was used to obtain the optimal values of (B_k, ψ_k, D) for the average diurnal cycle (t^*, T^*) data shown in Figs. 3.8-Figs. 3.10 for each individual lake. These values are summarized in Table 3.1. The peak temperature occurs at $t^* = 0.80$, and the minimum temperature occurs at $t^* = 0.01$. Thus, the peak occurs a few hours before sunset and the minimum shortly after sunrise. This reconstruction is shown along with the original average (t^*, T^*) results for Lake Hartwell in Fig. 3.13.

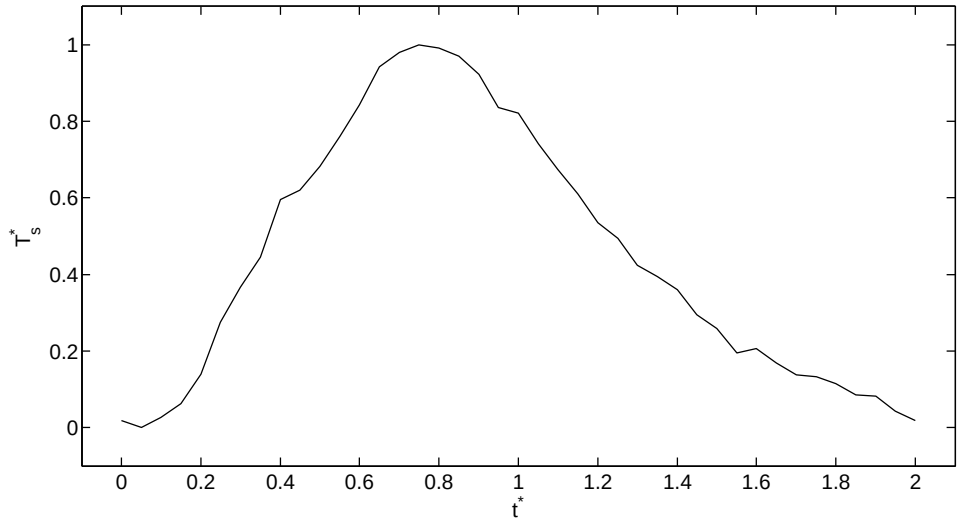

Figure 3.6: Lake Jocassee average plot of T_s^* versus t^* for the entire simulation period (2006-2014).

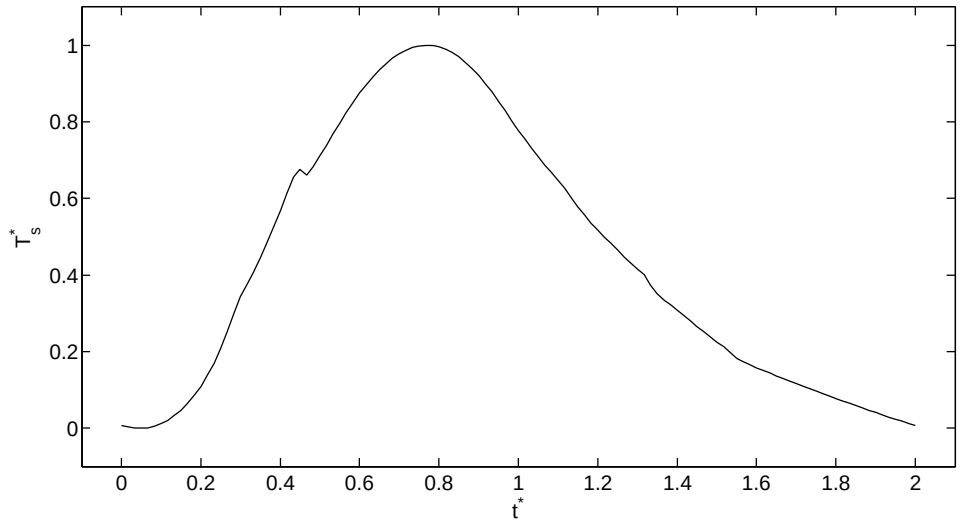

Figure 3.7: Lake Keowee average plot of T_s^* versus t^* for the entire simulation period (2006-2014).

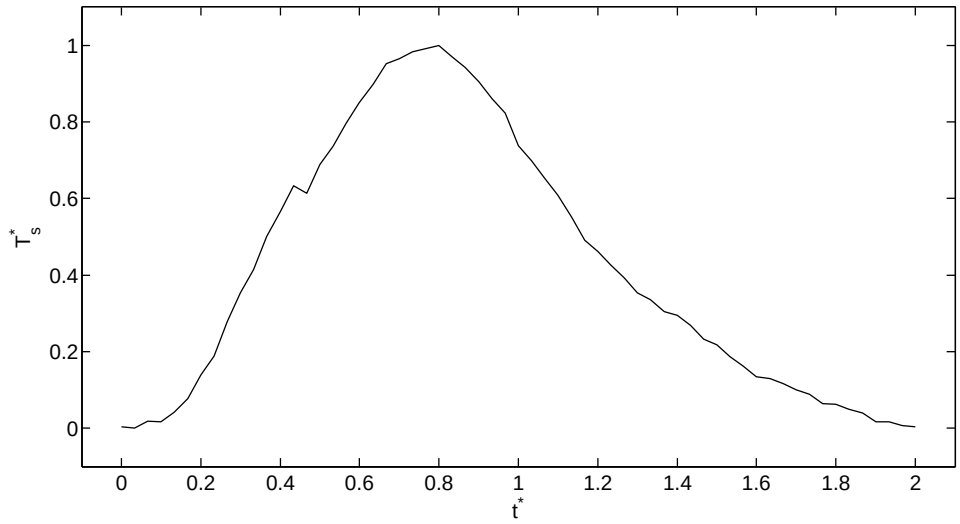

Figure 3.8: Lake Hartwell average plot of T_s^* versus t^* for the entire simulation period (2002-2014).

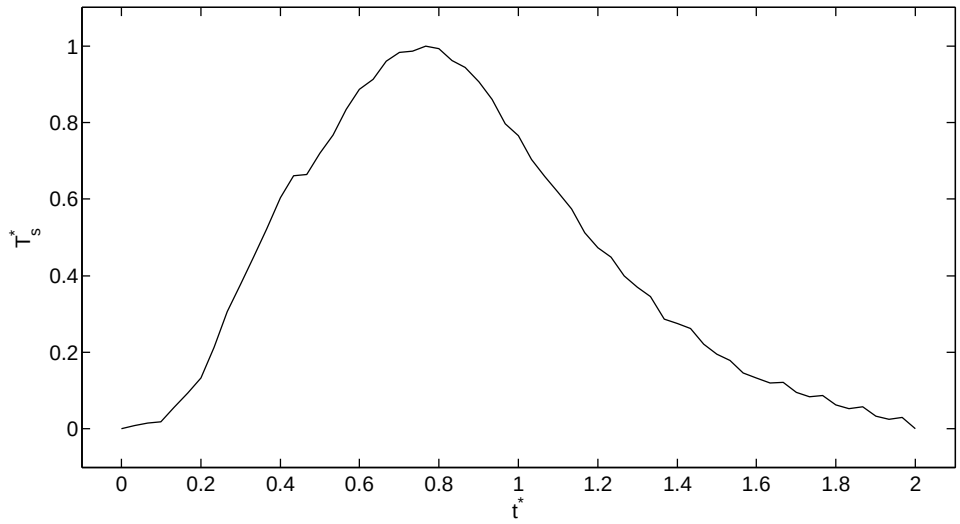

Figure 3.9: Lake Russell average plot of T_s^* versus t^* for the entire simulation period (2002-2014).

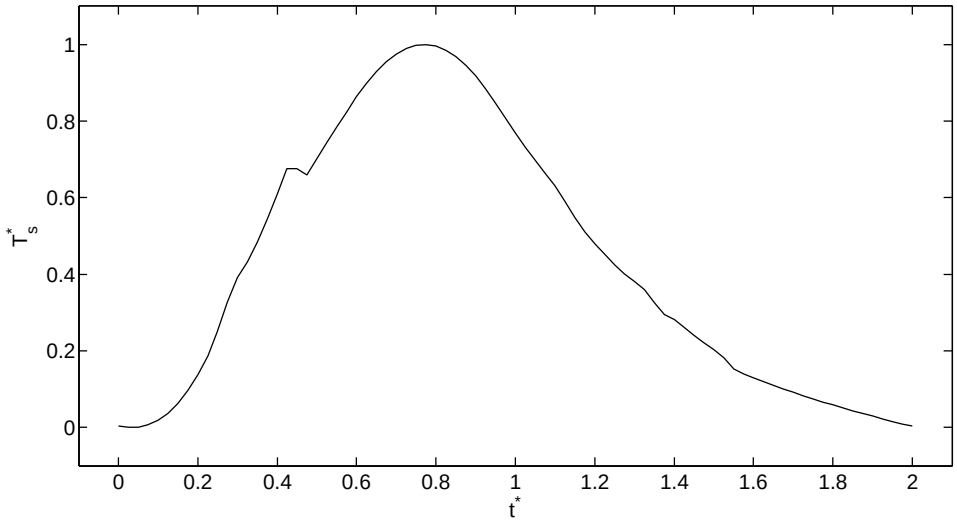

Figure 3.10: Lake Thurmond average plot of T_s^* versus t^* for the entire simulation period (2002-2014).

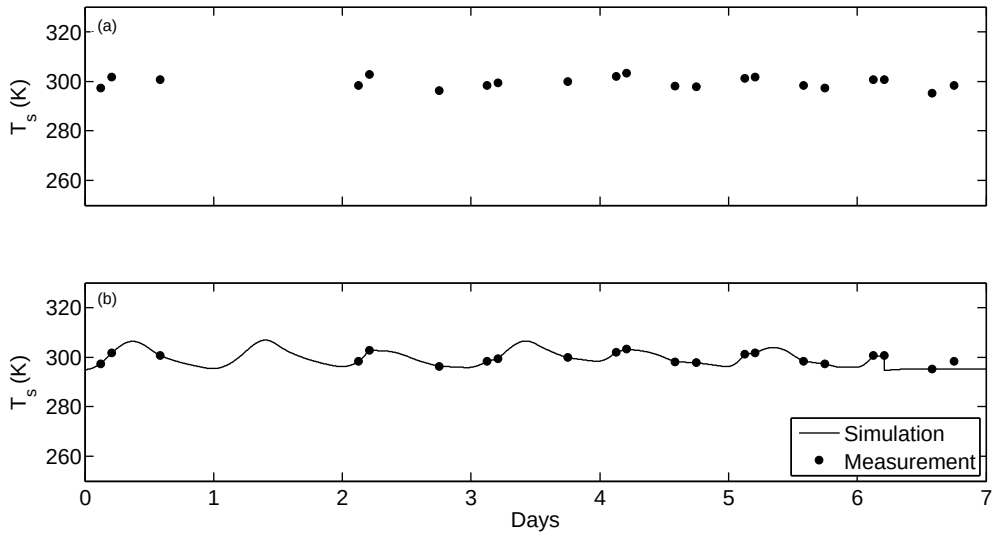

Figure 3.11: Surface temperature, T_s, in K versus day from simulation results for a typical week where both u_{10} and C_k^f are large. (a) Satellite measurements only. (b) Satellite measurements and simulation results.

This same non-dimensional scaling, averaging, and fitting using a Fourier transform was applied to the simulation results for Lakes Jocassee, Keowee, Russell, and Thurmond. The resulting average diurnal cycles of T_s for each lake are presented in Fig. 3.14. The constant values for Eq. (3.4) for each lake are presented in Table 3.1.

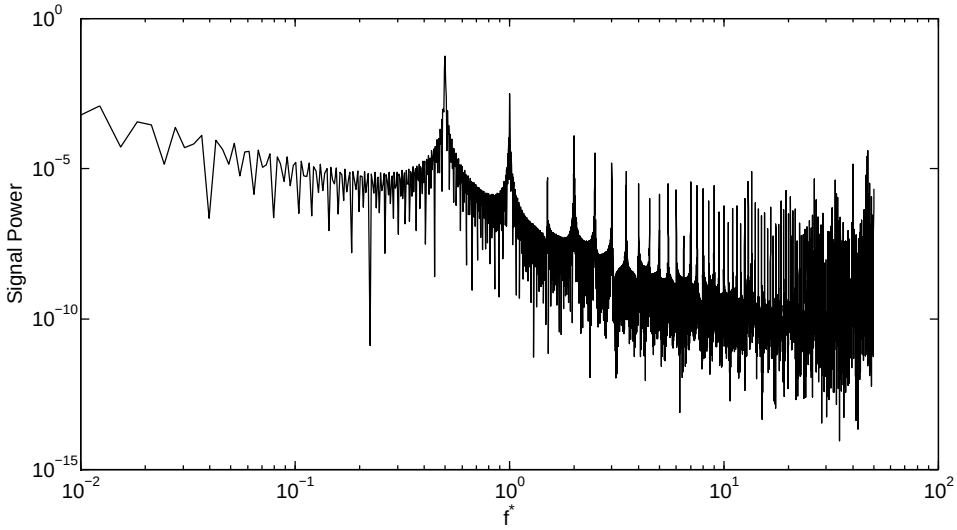

Figure 3.12: Fourier transform of data presented in Fig. 3.8.

Lake	B_1	B_2	B_3	B_4	ψ_1	ψ_2	ψ_3	ψ_4	D
Jocassee	0.4145	0.1036	0.0166	0.0456	1.14	2.66	7.83	8.64	-0.4807
Keowee	0.4665	0.1120	0.0140	0.0187	1.03	2.82	3.43	2.94	-0.4407
Hartwell	0.4442	0.1110	0.0178	0.0311	2.88	6.63	8.94	8.94	-0.4212
Russell	0.4592	0.1148	0.0046	0.0230	0.99	2.82	3.93	2.74	-0.4285
Thurmond	0.4677	0.1029	0.0047	0.0281	0.99	2.80	2.93	2.94	-0.4348

Table 3.1: Constant values for Eq. (3.4) for lakes in the Savannah River Basin.

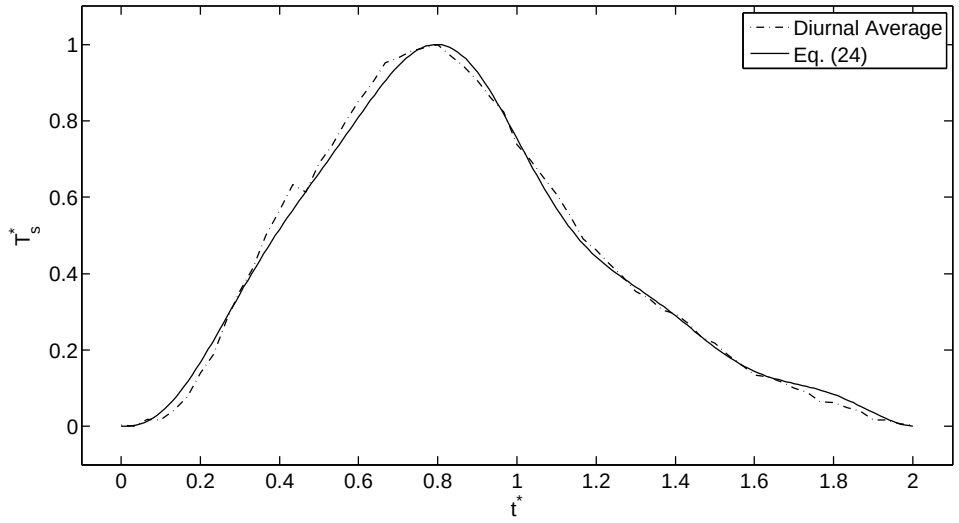

Figure 3.13: Average T_s^* versus t^*, from simulation results from 2002-2014 with sinusoidal function, Eq. (3.4).

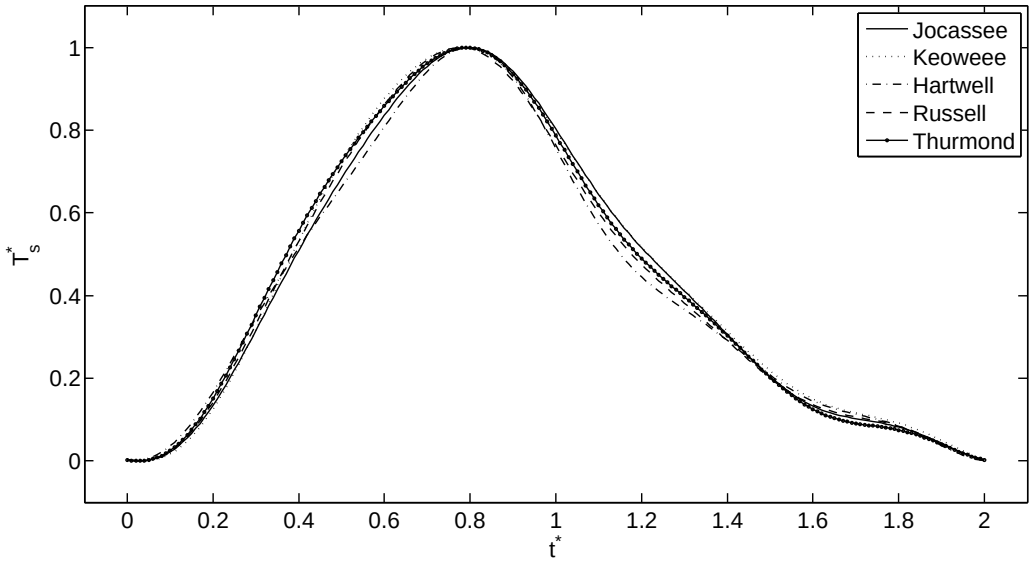

Figure 3.14: Plots of T_s^* versus t^* obtained from Eq. (3.4) for all 5 lakes.

Chapter 4

Discussion

To the author's knowledge, Eq. (3.4) is the first functional description of the diurnal variation in T_s for a lake, which complicates comparison with other results in the literature. Similar work has been presented by Jin *et al.* for the land surface skin temperature diurnal cycle (LSTD).[1] Similar to what is presented here, the LSTD model uses a min/maxed temperature in terms of local sunrise and sunset times. The LSTD authors propose a sinusoidal fit between sunrise and sunset; however, for the period from sunset to midnight they use a power law fit, and a linear fit from midnight to sunrise.

The LSTD authors did not present their average diurnal cycle in t^*. Thus, to fit their data to the scaling defined in this work, first the T^* values of the LSTD as a function of local time were calculated using the equations presented by Jin *et al.* for sunrise and sunset times matching their data. The constants in the LSTD equations were solved iteratively by modifying them one by one and visually checking the resultant plot with the one presented in the LSTD paper. Once the plots matched, the t^* scaling was applied to the LSTD authors' data.

The LSTD function is presented in Fig. 4.1, along with that obtained for lakes showing the similarities and differences between the two models. The minimum occurs at approximately the same time; however the peak time is later in the day on the lake than on land. On land, the heat transfer from the surface layer to the layers below is primarily through conduction, which is a fast mode of heat transfer. Consequently, the temperature as a function of depth should follow the general conduction solution for a radiated surface with a sinusoidal source. Thus, land temperature should monotonically decrease with depth. However as discussed in Section 2.7, on water this is not

the case. Turbulent mixing due to buoyancy and wind shear cause the water to mix to a constant temperature throughout the mixed layer. Therefore there is a much larger volume of water that must gain or lose energy to experience a change in temperature than the volume of land. Thus, it is expected that the land surface temperature would respond more rapidly to radiative forcing than lake surface temperature.

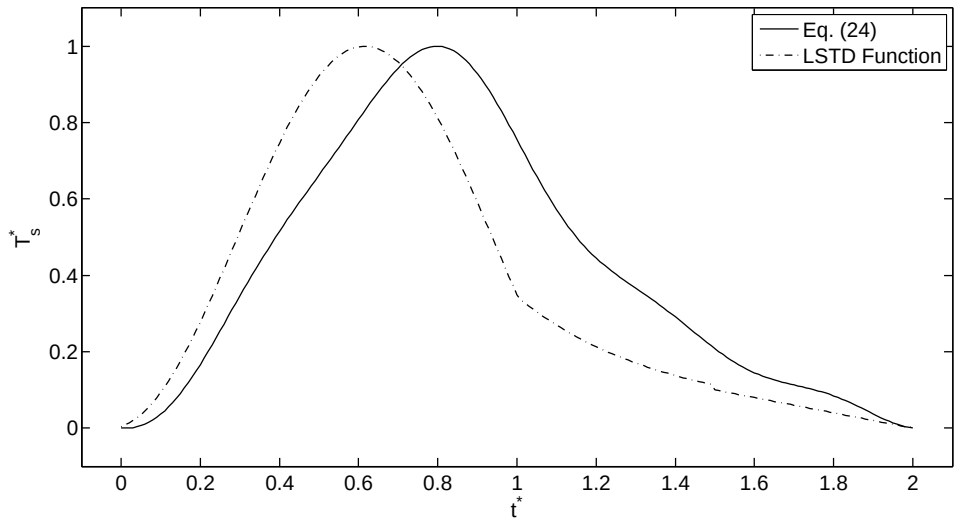

Figure 4.1: Plot of T_s^*, versus t^* for the results developed herein and that of the LSTD model due to Jin *et al.*[1]

Lakes Jocassee, Keowee, Hartwell, Russell, and Thurmond are geographically very close to each other and therefore experience, essentially, the same insolation and weather conditions. However, several aspects of these lakes differ. This is shown in Table 4.1 which reveals significant differences in the depth, area, and coast length of these lakes. In spite of these differences, as Fig. 3.14 shows, there is almost no difference in the averaged diurnal variation in T_s when presented in dimensionless form according to Eqs. (3.2) and (3.3). This suggests a certain robustness in the diurnal variation of lake surface temperature when considered in the dimensionless form developed here, although whether this robustness holds up for lakes experiencing different meteorological conditions would require further research.

Of course by making the T_s versus t data dimensionless, significant variations are purposely masked and such variations may provide useful information. To further develop an understanding of how these lakes are similar and different, the results presented in Fig. 3.14 were reprocessed in two additional ways, each using the same t^* as used in Fig. 3.14, but scaling T_s differently. First,

Lake	*Lat*	*Long*	*Elev*	D_{avg}	D_{max}	C	A_s
Jocassee[2]	34.96°N	82.92°W	338 m	48 m	107 m	121 km	30 km^2
Keowee[2]	34.80°N	82.89°W	240 m	16 m	91 m	623 km	75 km^2
Hartwell[1]	34.47°N	82.85°W	201 m	14 m	56 m	1548 km	230 km^2
Russell[1]	34.09°N	82.63°W	145 m	12 m	45 m	869 km	108 km^2
Thurmond[1]	33.66°N	82.20°W	100 m	11 m	42 m	1930 km	288 km^2

Table 4.1: Physical characteristics of lakes in the Savannah River Basin where D_{avg} is the average lake depth, D_{max} is the max lake depth, C is the coast length and A_s is the lake surface area. Data supplied by USACE.[1] Data supplied by DNR.[2]

the daily time traces of T_s versus t^* were averaged over the entire period of record for each lake. The resulting diurnal cycle is the average day for the entire data set, in Kelvins. The results are presented in Fig. 4.2. This method has the advantage of showing vertical offsets in yearly average temperatures between the lakes.

In the second method the daily mean is subtracted from each daily T_s versus t^* time trace and then all of the days in the period of record are averaged together for each lake. This yields a time trace of the deviation from the daily mean T_s for the simulation. The results of this approach are presented in Fig. 4.3. This method has the advantage of showing which lakes experience the greatest range of temperature change on an average day.

It is noted that in Figs. 4.2 and 4.3, the actual T_s simulations are presented, not the Fourier fit which was shown in Fig. 3.14. This is why Figs. 4.2 and 4.3 are somewhat noisier. The reason for this noise was discussed briefly in Chapter 3, and will be discussed more later in this section.

The next step is to determine if the lake-to-lake differences shown in Figs. 4.2 and 4.3 are related to any of the physical lake characteristics presented in Table 4.1. Observing the trends of T_s versus t^* shown in Fig. 4.2, the ordering of lakes from the highest average T_s to the lowest are: Keowee, Thurmond, Hartwell, Russell, and Jocassee. None of the parameters listed in Table 4.1 follow this same trend. However Lake Keowee is a heat sink for the Duke Energy Oconee Nuclear Station (ONS), and this excess energy may cause Lake Keowee's T_s results in Fig. 4.2 to be an outlier. The likelihood of this is supported by the experimental work of Oliver *et al.* where T_s was observed to increase by 4 K when ONS became operational.[26] Neglecting Lake Keowee, computing the average over the diurnal cycle $\overline{T_s}$ for the data in Fig. 4.2, and plotting this versus A_s and C for the remaining four lakes reveals a monotonically increasing trend in both cases, as shown in Figs. 4.4 and 4.5, respectively.

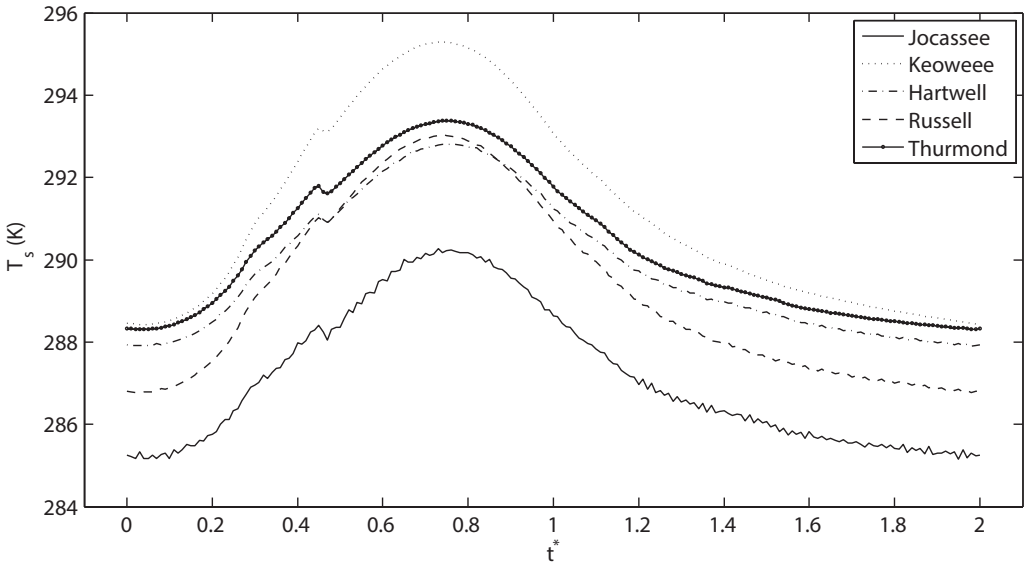

Figure 4.2: T_s versus t^* time trace obtained by averaging all daily time traces for the period of record.

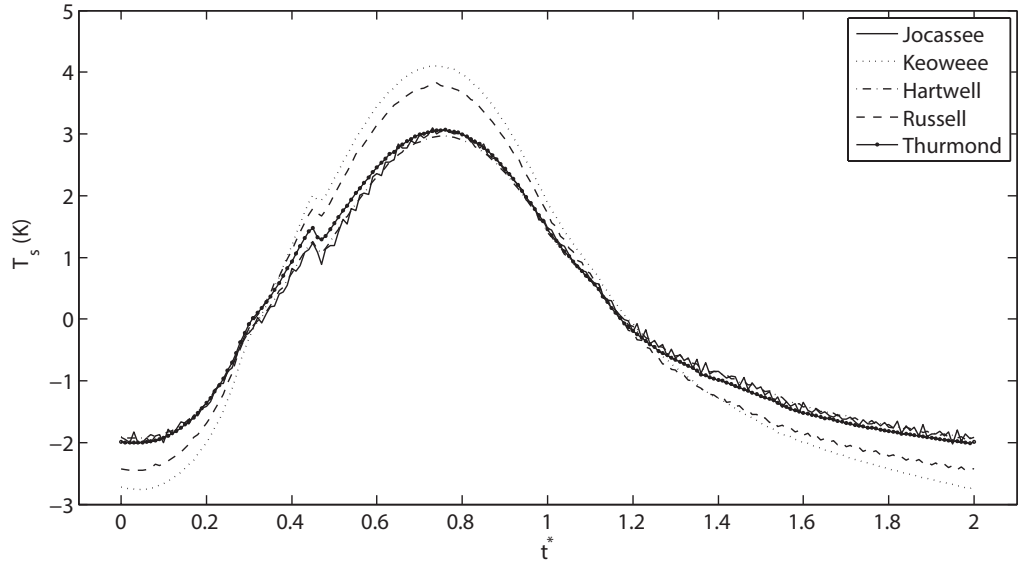

Figure 4.3: T_s versus t^* time trace where the daily mean is subtracted from each day and then all days were averaged over the period of record.

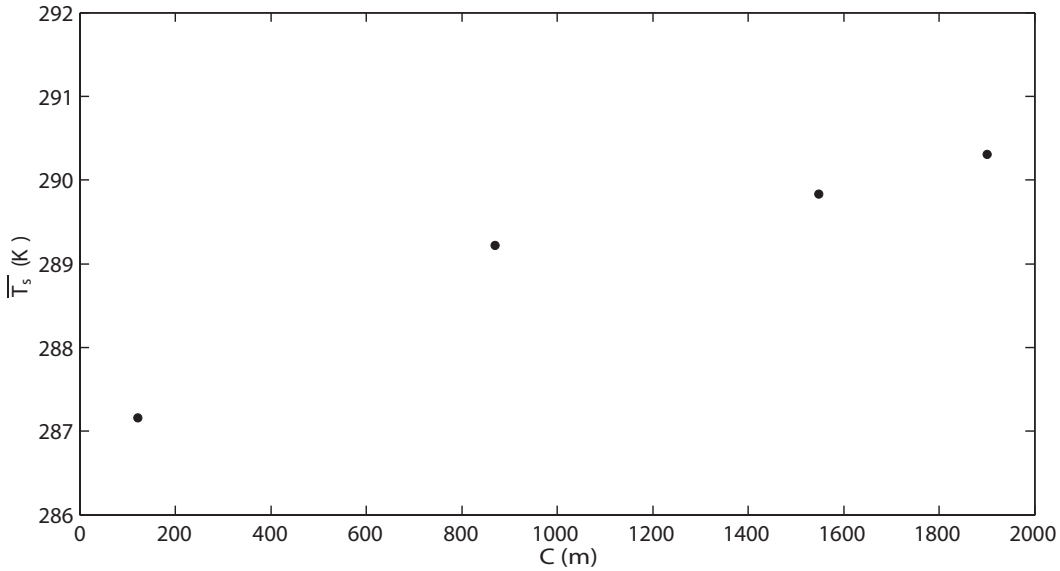

Figure 4.4: Average T_s from each lake diurnal cycle versus coast length for the four lakes in the Savannah River Basin.

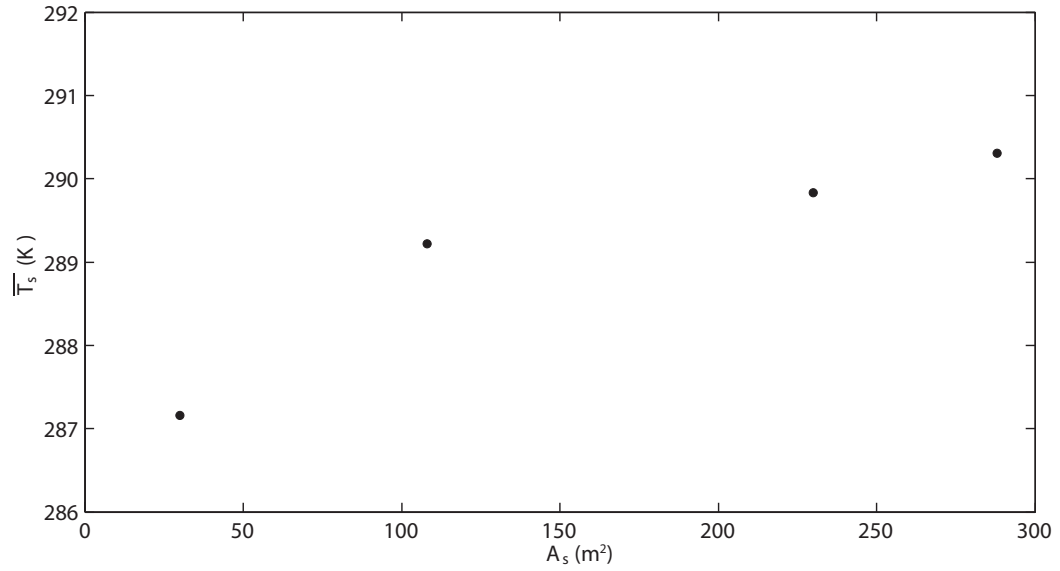

Figure 4.5: Average T_s from each lake diurnal cycle versus surface area for the four lakes in the Savannah River Basin.

Both C and A_s generally increase as the size of the lake increases. However, it is not immediately apparent why a larger lake would have a higher $\overline{T_s}$ than a smaller lake under similar meteorological conditions. One possible explanation for the increase in $\overline{T_s}$ presented in Figs. 4.4 and 4.5 can be explained by the existence and extent of dendrites. Many lakes contain inlets, outlets, bays, and coves which can account for a substantial amount of A_s and C. In the SRB, these dendrites generally have a smaller depth than that of the rest of the lake. In some instances, these dendrites are shallow enough that solar radiation penetrates to the bottom of the lake and creates a buoyantly unstable system which causes the water to fully mix in this area. This can result in dendrites having a higher T_s than the rest of the lake. Wind across the surface and circulation within the lake can spread these higher T_s regions toward the center of the lake. Thus, it would make sense for lakes with a higher proportion of dendrites to have a higher $\overline{T_s}$. To quantify the dendrites in the SRB, the ratio:

$$D_r = \frac{C}{P} \tag{4.1}$$

was used where D_r is the dendritic ratio, C is the coast length of the lake, and P is the perimeter of a circle with a surface area equal to that of the lake. Thus D_r is the ratio of the actual coast length to the minimum possible coast length, which correlates to how prevalent dendrites are. As an example, the outline of Lake Jocassee ($D_r = 6.2$) and Lake Hartwell ($D_r = 28.8$) are shown side by side in Fig. 4.6. Values for D_r for each lake in the SRB are presented in Table 4.2. A plot of $\overline{T_s}$ versus D_r is presented in Fig. 4.7, which shows that $\overline{T_s}$ increases monotonically with D_r which supports the theory that the prevalence of dendrites affects $\overline{T_s}$.

Lake	Jocassee	Keowee	Hartwell	Russell	Thurmond
D_r	6.2	13.5	28.8	23.6	31.6

Table 4.2: Dendritic ratio, D_r, for the lakes in the SRB

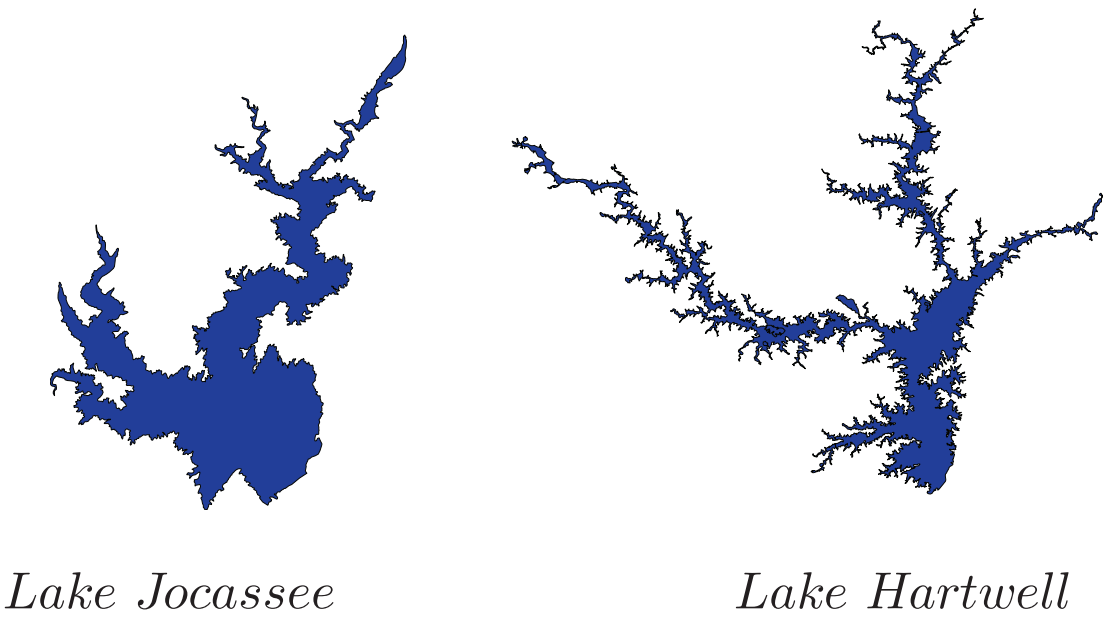

Figure 4.6: Comparison of Lake Jocassee ($D_r = 6.2$) and Lake Hartwell ($D_r = 28.8$). Note that the two lakes have been scaled to appear the same size to better present the dendrites.

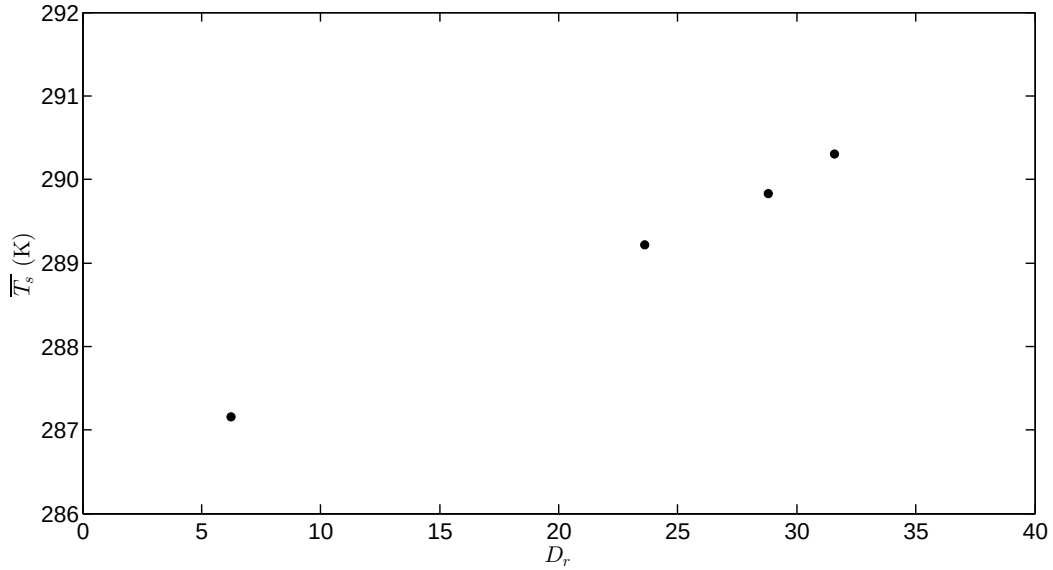

Figure 4.7: Average T_s from each lake diurnal cycle versus dendritic ratio for four of the lakes in the Savannah River Basin.

To better show the relationships between C, A_s and D_r, Eq. 4.1 can be rewritten as the following:

$$D_r = \frac{C}{2\left(\frac{4A_s}{\pi}\right)^{1/2}} \tag{4.2}$$

According to Eq. 4.2, as A_s increases, D_r should decrease, and as C increases, D_r should increase. Figure 4.8 shows that the lakes in this work follow the expected trend of D_r and C being directly correlated. However, Fig. 4.9 shows that A_s also monotonically increases with D_r within the SRB. This increase in the prevalence of dendrites as lake size increases in the SRB is due to C increasing proportionally more than A_s in these lakes. This is likely due to an increase in tributary basins as lake size increases in the SRB. Thus, the trends observed in Fig. 4.4 and 4.5 may not be true of other basins which may have different inlet and outlet conditions. Although $\overline{T_s}$ scales similarly with A_s, C, and D_r for the lakes examined in this work, using D_r provides a physical explanation for why $\overline{T_s}$ would behave in this way. However, additional data from lakes varying D_r with different combinations of large and small A_s and C would be needed to test this hypothesis.

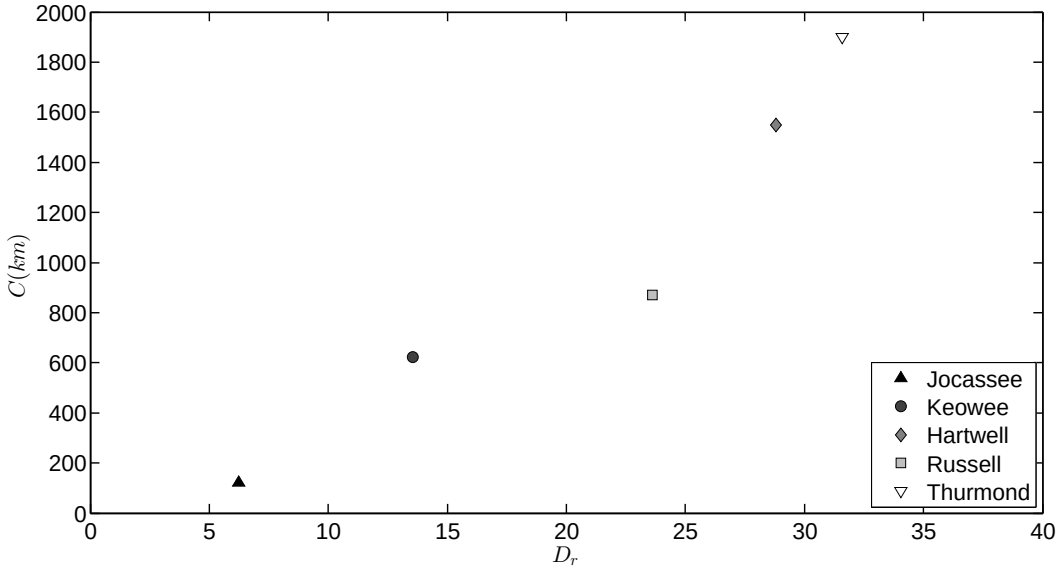

Figure 4.8: Lake D_r versus C for each of the lakes in the SRB

Plots of the deviation from the mean of T_s versus t^* shown in Fig. 4.3 indicates that Lakes Keowee and Russell experience the largest range of temperature change in the average diurnal cycle; whereas the other three lakes (Hartwell, Jocassee, and Thurmond) experience essentially the same

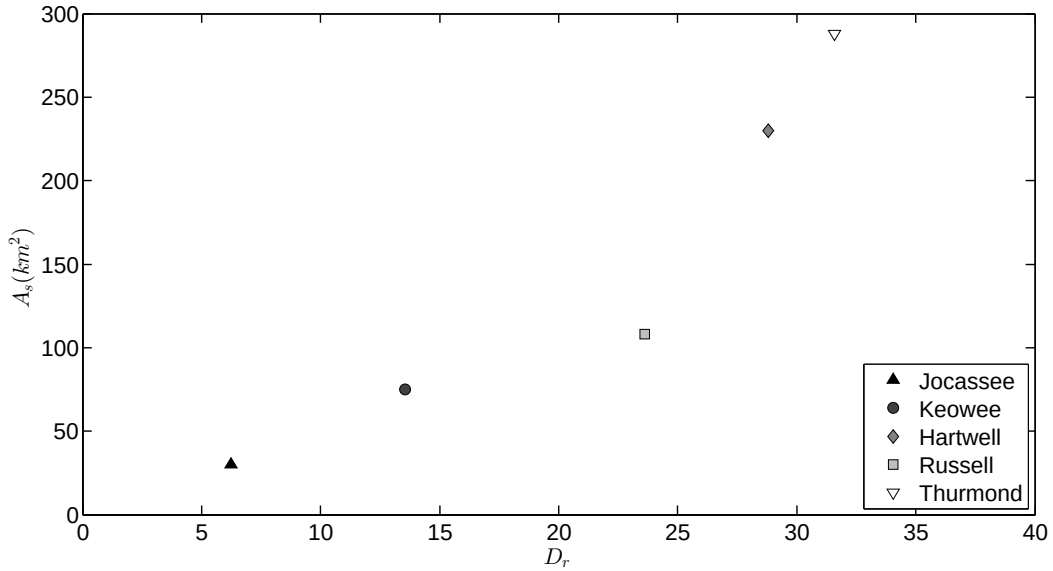

Figure 4.9: Lake D_r versus A_s for each of the lakes in the SRB

trend. This means that on an average day, T_s on Lakes Keowee and Russell will change more than on Lakes Hartwell, Jocassee, and Thurmond. None of the parameters cataloged in Table 4.1 explain this deviation. However, if Lake Keowee is discounted for the same reasons discussed above, then the only outlier is Lake Russell.

Lake Russell is interesting since it utilizes pumpback turbines. This means that periodically the flow of the Savannah River is reversed to pump water from Lake Thurmond to Lake Russell. Pumpback systems are generally designed such that the water entering the upper and lower basins enters the lake with an angle above horizontal. The water coming from the upper reservoir is typically as vertical as possible, and the water coming from the lower reservoir is just slightly above horizontal.[27] The water from the lower reservoir is taken from well below the lake surface to ensure that there will be enough water in the lake for the pumpback systems to operate during drought. Thus, the water being pumped from Lake Thurmond at night is coming from the colder, bulk layer and being pumped into Lake Russell.[27] For a lake the size of Lake Russell, it is possible that this jet of cold water could travel through the bulk of the lake and mix with the epilimnion layer, resulting in a reduction of T_s at night. This drop in T_s at night would explain the larger range of temperatures experienced by Lake Russell in Fig. 4.3. Looking at the trend for Lake Russell shown in Fig. 4.2, the night time average T_s is less than that of Lake Hartwell which has a similar D_r. This observation agrees with the explanation that the pumpback system lowers T_s for Lake Russell at night. Lake

Jocassee also utilizes pumpback systems with Bad Creek Reservoir. However, the pumpback system into Lake Jocassee discharges into the hypolimnion and it is less likely that the plume of water from the pumpback system penetrates the epilimnion and affects T_s due to the depth of Lake Jocassee.[28]

After having eliminated Lakes Russell and Keowee, only Lakes Jocassee, Hartwell, and Thurmond remain. These three lakes exhibit very similar behavior in Fig. 4.3. This is intriguing since Lake Jocassee is so much deeper than the other two. This suggests that variations in the parameters listed in Table 4.1 do not affect the range of temperature change on a daily basis. However, significant differences in latitude and longitude were not considered in this work, which seem to be the parameters most likely to cause deviation in this averaging method by increasing or decreasing the length of day.

In this work T_b for each lake was considered to be the same as Lake Hartwell since T_b measurements were only available from Lake Hartwell throughout the simulation duration. As discussed in Chapter 2.4, the T_b used for Lake Hartwell was a concatenation of third-order polynomial best fits to USACE measurements for individual years. Using T_b measurements from Lake Hartwell for all five lakes could lead to an overestimation of the collapse in the diurnal function. However, in the development of the model on Lake Hartwell, it was found that changing T_b values changed the solution for L in the model but did not significantly affect T_s. This is because the model calculates an effective mixed layer depth which best fits the T_s measurements from the MODIS instruments. Thus, changing T_b would not affect the T_s solution unless it were very different.[29]

To confirm that the trend observed in Fig. 4.2 was not affected by the use of Lake Hartwell bulk measurements for all the lakes, the average T_{sat} at each satellite over-pass time was computed. These measurements were obtained directly from MODIS, and were not affected by any assumptions made in the simulation. These average measurements are presented in Fig. 4.10 along with the simulations presented in Fig. 4.2. The order from minimum to maximum $\overline{T_{sat}}$ follows the same trend as that of the simulation results as shown in Fig. 4.11 which is a plot of $\overline{T_s}$ versus $\overline{T_{sat}}$ for each of the five lakes. Here, $\overline{T_s}$ is computed by averaging T_s for the diurnal cycle for each lake shown in Fig. 4.2. $\overline{T_{sat}}$ is computed by averaging T_{sat} shown in Fig 4.10 for each lake. This further demonstrates that the variation in average surface temperature from lake to lake follows the same trend in the simulation results and in the MODIS measurements. The values of both $\overline{T_s}$ and $\overline{T_{sat}}$ are presented in Table 4.3. The satellite average temperatures were generally higher than the simulation average temperatures at the same t^*. This is likely due to T_{sat} being limited to clear sky days, since

MODIS cannot provide T_{sat} through clouds. However, the simulation predicts T_s for cloudy days even when there are T_{sat} dropouts. Since cloudy days would experience less solar insolation, these days would have a lower average T_s.

Lake	Jocassee	Keowee	Hartwell	Russell	Thurmond
$\overline{T_{sat}}(K)$	289.8	292.4	291.1	290.8	291.2
$\overline{T_s}(K)$	287.2	291.2	289.8	289.2	290.3

Table 4.3: $\overline{T_{sat}}$ for the satellite measurements on on each lake.

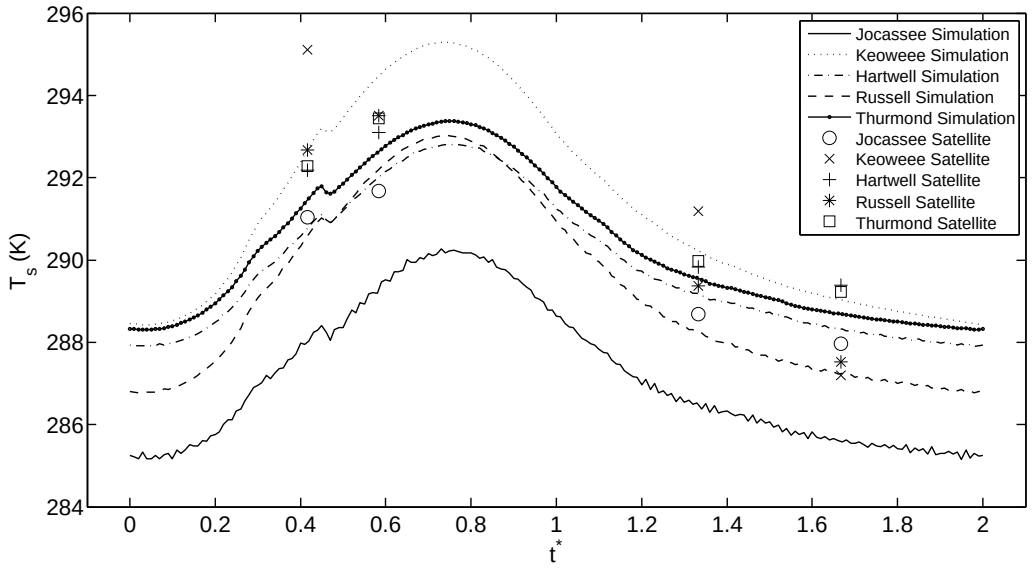

Figure 4.10: T_s versus t^* time trace obtained by averaging all daily time traces for the period of record with average T_{sat}

The purpose of developing a diurnal function for T_s was to enable one to obtain values at times in between satellite overpasses. To determine the utility of Eq. (3.4) in doing this, four daily T_{sat} satellite measurements were fit to Eq. (3.4). Fitting was accomplished by generating a linear set of four equations and then solving for B_1 through B_4 in Eq. (3.4). The values for f_k, ψ_k, and D were obtained from Table 3.1, i.e. the values obtained when fitting to the entire data simulation period presented in Fig. 3.8. A comparison between the satellite data, the simulations and Eq. (3.4) is presented for two sample days in Figs. 4.12 and 4.13. As these figures show, fitting Eq. (3.4) to the satellite data gives very poor performance; the resulting plot agrees with neither the satellite data nor the simulations. The average diurnal trend presented in Fig. 3.8 shows that the average diurnal T_s is driven primarily by the sun. The variation in the ambient parameters (u_{10}, T_a, ϕ) is

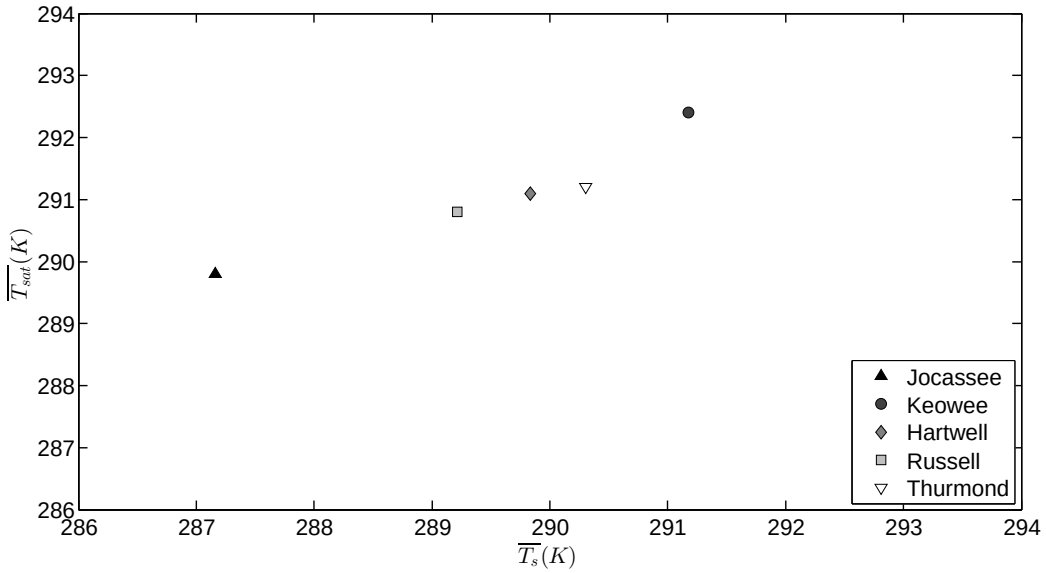

Figure 4.11: $\overline{T_s}$ and $\overline{T_{sat}}$ from each lake diurnal cycle for five of the lakes in the Savannah River Basin.

damped when enough days are averaged. However, for any given individual day, T_s can be greatly affected by variation in these ambient parameters, and this is the main reason for our inability to use Eq. (3.4) to predict T_s in between satellite overpasses. Consequently, a useful, though less than dramatic result of this work is that a simple linear interpolation between satellite measurements is likely to give better results for daily estimates of T_s than use of Eq. (3.4). However, performing the full simulation presented herein may still be useful in providing daily maximum and minimum temperatures.

In this work u_{10} was calculated by minimizing the error between the simulation T_s and MODIS measurements of T_{sat}. To compare the validity of the u_{10} used in the simulations with the ASOS measurements, the correlation coefficient between the ASOS measurements and the simulation u_{10} for Lake Hartwell was computed. Specifically, the u_{10} used in the simulations was compared to those measured at three neighboring weather stations: the Anderson Regional Airport (AND), the Greenville Downtown Airport (GMU), and the Oconee County Regional Airport (CEU). Additionally, the correlation coefficient was calculated using the average of the three stations. For comparison, the correlation coefficients of the wind measurements between each of the three stations were also calculated. These correlation coefficients are presented in Table 4.4. None of the correlation coefficients between individual data sets exceeded 0.04, showing there is very little correlation

between the calculated wind speed values and the available measurements. It is not entirely unexpected that there is poor agreement of simulation predicted wind speeds and ASOS measurements since the calculated wind speed was used to incorporate all the unknown conditions that occurred throughout the day to ensure convergence at satellite measurements of T_{sat}. Ideally there would be greater correlation; however based on the lack of correlation between individual stations, spatial variation in wind is significant, and it would not be reasonable to expect the calculated wind values to be more correlated than the individual stations. Thus, leaving u_{10} as a floating variable in the simulation likely will yield better results than simply choosing a single individual station's measurements. The correlation coefficient of each individual station is higher with the average of the three stations, which is expected since each individual station counts for 1/3 of the average in the calculation.

Wind Source	AND	GMU	CEU	AVG
SIM	-0.0075	0.0323	0.0042	0.0160
AND	-	0.0322	0.0181	0.6357
GMU	0.0322	-	0.0035	0.5908
CEU	0.0181	0.0035	-	0.5330

Table 4.4: Correlation coefficient, R, between measured u_{10} and simulation output u_{10}, and correlation coefficients of each of the station with each other.

One of the biggest sources of error in the simulation algorithm comes from how u_{10} is handled when the equations predict large values. The maximum allowed u_{10} of 20 m/s for the simulation was chosen based on the maximum ASOS measurement observed in the simulation time frame. However, due to the solution method, u_{10} is set to 20 m/s more often than ASOS measurements predict. However, setting $u_{10} = 20$ m/s generally results in large spikes in T_s. Simulation results for a sample week where T_s experiences such a spike are shown in Fig. 3.11. The sharp change in temperature at day 6 occurs when both u_{10} and C_k^f are changing rapidly, as is shown in the sample week in Figs. 4.14 and 4.15 for u_{10} and C_k^f respectively. This causes a large shift in L, shown for the sample week in Fig. 4.16, which causes the entrained water at T_b to change T_s rapidly. The first order discontinuity in T_s in this situation makes the simulation results less reliable. With more knowledge of u_{10} on the lake surface, this error could be reduced. This error occurs in less than 7% of the simulation.

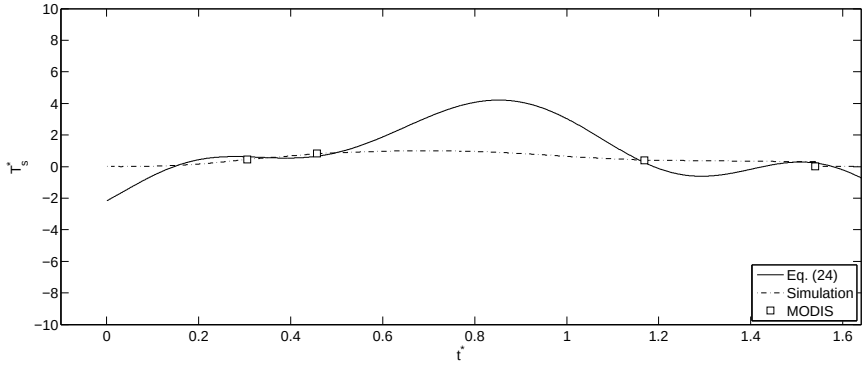

Figure 4.12: Plot of T_s^* versus t^* for the satellite data, the simulations, and Eq. (3.4) for a sample day.

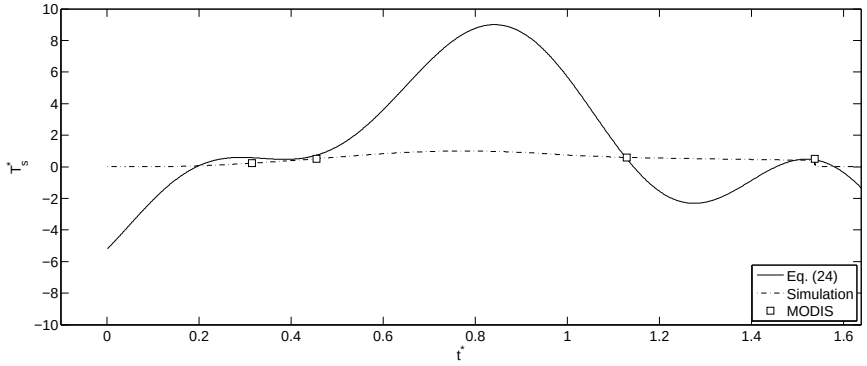

Figure 4.13: Plot of T_s^* versus t^* for the satellite data, the simulations, and Eq. (3.4) for a sample day.

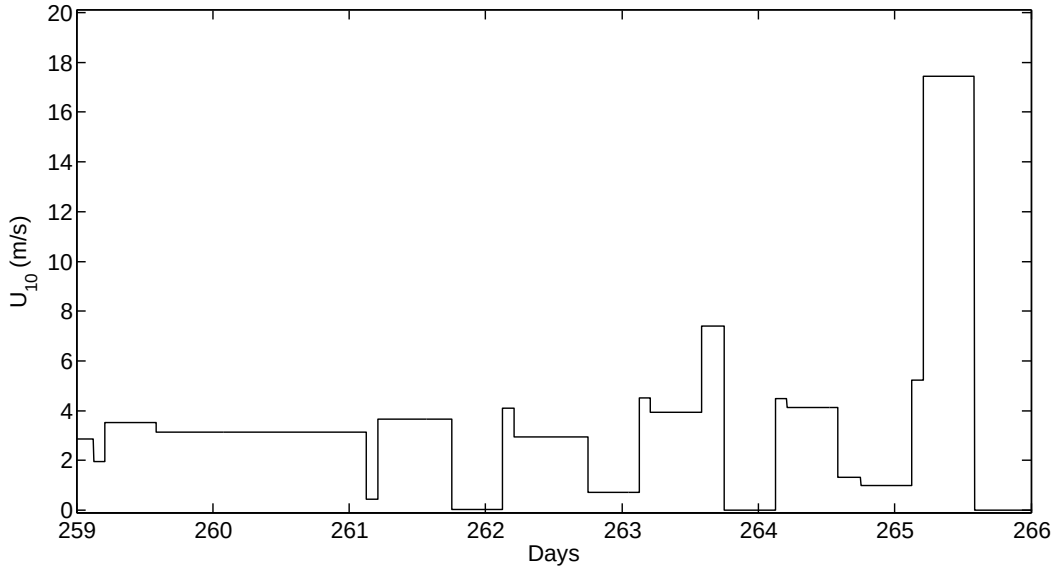

Figure 4.14: Wind speed, u_{10}, in m/s versus day number from simulation results for a typical week.

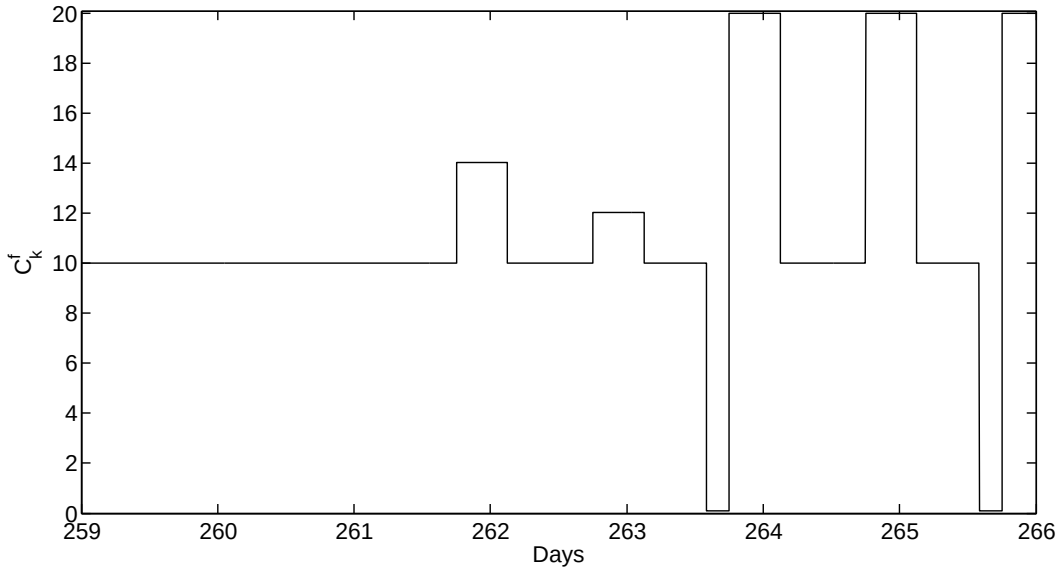

Figure 4.15: Internal losses coefficient, C_k^f versus day number from simulation results for a typical week.

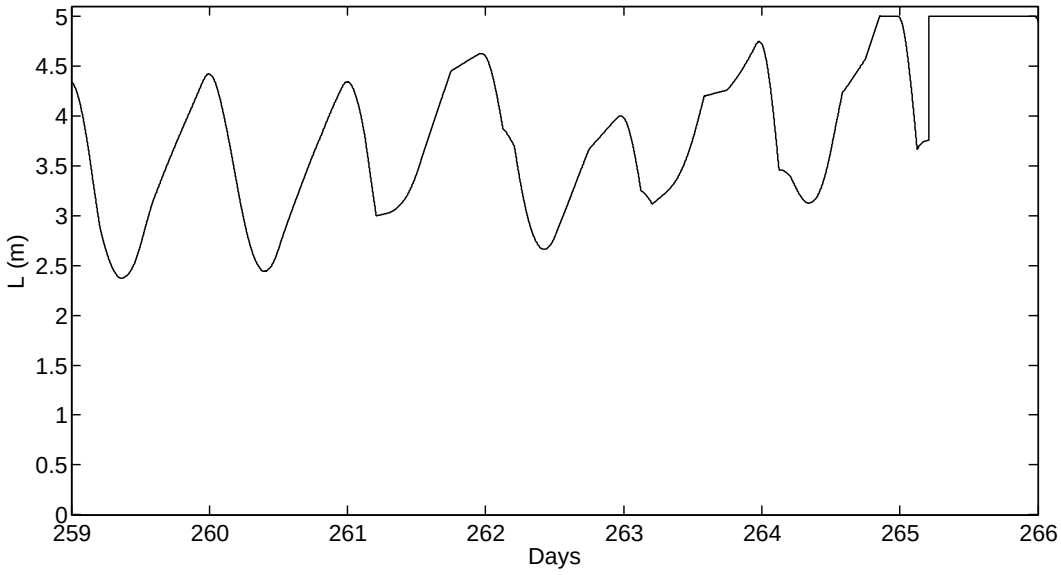

Figure 4.16: Mixed layer depth, L, in m versus day number from simulation results for a typical week.

Chapter 5

Conclusion

Hourly surface temperature, T_s, was simulated for Lakes Jocassee, Keowee, Hartwell, Russell, and Thurmond in the Savannah River Basin using measurements of ambient atmospheric conditions from the Oconee County Regional Airport, the Anderson Regional Airport, and the Augusta Regional Airport along with bulk temperature measurements from USACE and four daily satellite measurements of T_s from the MODIS sensors on NASA's two satellites, Aqua and Terra. The simulation results were collapsed based on daily temperature extrema and daily sunrise and sunset times.

The average diurnal cycle for each of the lakes on the non dimensional scales presented in this work were found to collapse to similar functions, approximated by a summation of four Fourier components. This functional form is an excellent approximation of the average trend and, to the author's knowledge is the first suggested functional form for the diurnal variation of T_s on a lake surface. However, applying this average trend to individual days does not enable good estimation of T_s between satellite overpasses, and a simple linear interpolation between satellite measurements exhibits better performance. However, using linear interpolation between measurements will miss the daily minimum and maximum T_s.

The consistency of the results for each of the lakes implies generality to all warm, monomictic lakes. However, the diurnal cycle of the dimensional temperature versus time does show differences between the lakes, generally scaling with coastal length and surface area of the lake. A dendritic ratio was defined which collapsed the effects of coastal length and surface area on $\overline{T_s}$. Future investigations comparing these results to those of warm, monomictic lakes in other regions of the world would be

illuminating. Additionally, future work investigating polymictic and bimictic lakes using the method presented herein could lead to a greater fundamental understanding of the general diurnal variation on all inland lakes and reservoirs.

Appendices

Appendix A

Simulation

This appendix contains the primary MATLAB code which follows the simulation algorithm presented in the previous chapters.

```
function [Message,results_path,results_name] =
TKE_method(Lake,f_dir,r_dir,varargin)
%%   This program calculates the mixing depth of a lake from the input file
%   Syntax: TKE_method(Lake,f_dir,r_dir,'OptionalName1','OptionalValue1',
%           'OptionalName2','OptionalValue2',...,'OptionalNameN',
%           'OptionalValueN')
%
%   The optional values allow the user to specifiy simulation settings
%   different from the saved defaults.  The available options are the
%   following:
%
%   'fixT' = Binary 0 or 1.  If 1, simulation temperatures will be auto-
%   corrected to match satellite measurements before the next data set is
%   calculated. If 0, this feature is turned off.  Default is 0.
%   NOTE: residual error is calculated before this correction is made.
%
%   'itErr' = Binary 0 or 1.  If 1, the simulation will ensure that the
%   residual error always gets close to 0 by including an error term.
%   Useful for seeing how much energy is actually required to hit the
%   satellite measurement.  If 0, this feature is turned off.  Default is
%   0.
%
%   'TbType' = Binary 0 or 1.  If 1, simulation uses polynomial curve fits
%   for bulk temperature produced from USACE data.  If 0, simulation uses a
%   step change in bulk temperature at each month change.  Default is 1.
%
%   'dTbType' = Binary 0 or 1.  If 0, energy required to change the bulk
%   temperature over time is included in conservation of energy.  If 1, it
%   is not included.  Default is 0.
```

```
%
%   'Utype' = 0, 1 or 2.  If 0, CK is iterated using measurements for wind
%   speed.  If 1, wind speed is brute force iterated.  If 2, wind speed is
%   iterated using a root finding algorithm.  Default is 2.
%
%   'dt' = Simulation time step in seconds.  Default it 60 seconds.
%
%   'N' = Number of days to simulate.  If N = 0, the entire input file is
%   simulated.  Default is 0.
%
%   'CKdef' = Default value of the coefficient used in the Turbulent
%   Kinetic Energy budget, CK. Default is 5.
%
%   'U10min' = Minimum wind speed to consider for iterating in (m/s).
%   Default is 0 m/s.
%
%   'U10max' = Maximum wind speed to consider for iterating in (m/s).
%   Default is 10 m/s.
%
%   'U10int' = Resolution for wind speed brute force iteration. Default is
%   0.2 m/s.
%
%   'maxL' = Maximum mixed layer depth allowed in m.  Default is 15 m.
%
%   'minL' = Minimum mixed layer depth allowed in m.  Default is 0.099 m.
%
%   'constL' = Constant mixed layer depth in m for winter.  Default is 0.25
%   m.
%
%   'initL' = Initial mixed layer depth in m.  Default is 5 m.
%
%   'ProFlag' = Flag whether to output current progress to command window.
%   Useful for debugging, can make running many different test cases
%   annoying. 0 is off, 1 is on.  Default is 0.
%
%   'ItArr' = Flag whether arrays should be allocated to maximum size and
%   set to 0 after each dataset or if they should be cleared and
%   reallocated after each dataset. There is a minor time saving for data
%   sets greater than 5 years.  Binary, 0 is off, 1 is on.  Default is 1.
%
%   Primary references used in the creation of this program:
%
%   [1] National Aeronautics and Space Administration. Land Processes
%   Distributed Active Archive Center (NASA LP DAAC). Products MOD06L2,
%   MYD06L2, MOD11A1, and MYD11A1.  USGS/Earth resources observation and
%   science (EROS) center, South Dakota, 2014.
%   [2] E. H. Alcantara, J. L. Stech, J. A. Lorenzzetti, M. P. Bonnet, X.
%   Casamitjana, A. T. Assireu, and E. M. Novo. Remote sensing of water
%   surface temperature and heat flux over a tropical hydroelectric
%   reseroir.  Remote sensing of environment, 114:2651-2665, 2010.
%   [3] National Oceanic and Atmospheric Administration, Department of
```

```
%   Defense, Federal Aviation Administration, United States Navy.
%   Automated surface observing system (ASOS_ user's guide, 1988.
%   [4] I. Reda and Andreas A. Solar position algorithm for solar radiation
%   applications.  Technical report, National Renewable Energy Laboratory,
%   2008.
%   [5] P.R. Lowe. An approximating polynomial for the computation of
%   saturation vapor pressure.  Journal of Applied Meteorology, 16:100-103,
%   1976.
%   [6] H.B. Fischer, E. J. List, R. C. Y. Koh, J. Imberger, and N. H.
%   Brooks. Mixing in Inland and Coastal Waters.  Academic Press, London,
%   1979.
%   [7] J. Imberger. The diurnal mixed layer.  Limnol. Oceanogr.,
%   30:737-770, 1985.
%   [8] S. S. Girgis and A. C. Smith. On thermal stratification in stagnant
%   lakes.  Int. J. Egng ci., 18:69-79, 1980.
%   [9] R. H. Spigel, J. Imberger, and Rayner K. N. Modelling the diurnal
%   mixed layer.  Limnol. Oceanogr., 31:533-556, 1986.
%   [10] C. O. Justice, and A. Mikhail.  Height variation of wind speed and
%   wind distribution statistics.  Geophysical Research Letters, 3,
%   261-264.
%
%
tic()

%% Read varargin inputs
%  input_poss contains the variable name to be replaced by varargin input
%  input_def contains the default value for the corresponding variable name
%  in input_poss.

input_poss{1,1} = 'fixT';   input_def(1,1) = 0;         % Binary, 0 or 1
input_poss{2,1} = 'itErr';  input_def(2,1) = 0;         % Binary, 0 or 1
input_poss{3,1} = 'TbType'; input_def(3,1) = 1;         % Binary, 0 or 1
input_poss{4,1} = 'dTbType';input_def(4,1) = 0;         % Binary, 0 or 1
input_poss{5,1} = 'UType';  input_def(5,1) = 2;         % 0, 1, or 2
input_poss{6,1} = 'dt';     input_def(6,1) = 60;        % seconds
input_poss{7,1} = 'N';      input_def(7,1) = 0;         % Number of days
                                                        % 0 = entire file
input_poss{8,1} = 'CKdef';  input_def(8,1) = 5;
input_poss{9,1} = 'U10min'; input_def(9,1) = 0;         % m/s
input_poss{10,1} = 'U10max';input_def(10,1) = 10;       % m/s
input_poss{11,1} = 'U10int';input_def(11,1) = 0.2;      % m/s
input_poss{12,1} = 'maxL';  input_def(12,1) = 15;       % m
input_poss{13,1} = 'minL';  input_def(13,1) = 0.099;    % m
input_poss{14,1} = 'constL';input_def(14,1) = 0.25;     % m
input_poss{15,1} = 'initL'; input_def(15,1) = 5;        % m
input_poss{16,1} = 'ProFlag'; input_def(16,1) = 0;      % Binary, 0 or 1
input_poss{17,1} = 'ItArr'; input_def(17,1) = 1;        % Binary, 0 or 1
input_poss{18,1} = 'CKmax'; input_def(18,1) = 10;
input_poss{19,1} = 'CKmin'; input_def(19,1) = 1;

%% Break varargin inputs into names and values
```

```
vargs = varargin;
nargs = length(varargin);       % Store number of non-default parameters
names = vargs(1:2:nargs);       % Store names of non-default parameters
values = vargs(2:2:nargs);      % Store values of non-default parameters

%% Load varargin input values into workspace

for i = 1:1:nargs/2                     % Loop through all varargin values
    for j = 1:1:length(input_def(:,1))  % Loop through possible varargin
                                        % prameters
        if strcmp(names{1,i},...        % Check if varargin name i matches
                input_poss{j,1}) == 1;  % possible name j
            dat_name = genvarname(...   % Generate variable name from
                names{1,i});            % string
            eval(strcat(dat_name,...    % Add varargin value to matched
                '=',num2str(...         % possible input name to the
                values{1,i}),';'));     % workspace.
        end
    end
end

%% Load default values for varargin inputs not included

for j = 1:1:length(input_def(:,1))      % Loop through all varargin values
    if eval(strcat('~exist(''',...      % Check if possible varargin name
            input_poss{j,1},...         % was already added to workspace.
            ''',''var'')')) == 1;
        eval(strcat(...                 % Add default value to the
            input_poss{j,1},'=',...     % workspace.
            num2str(input_def(j,1)),';'));
    end
end

%% Designate input data file name and location
%  Syntax: LakeName_data.txt Rows correspond to different time stamps,
%          Columns correspond to different data measurements. See Data
%          Format below.
%
%  Data Format:
%
%  Column   Description
%  1        latitude
%  2        longitude
%  3        date (yyyymmdd)
%  4        time
%  5        ambient air temperature, T_a
%  6        air velocity, U_z
%  7        relative humidity, r
%  8        Satellite surface temperature
%  9        Solar altitude, degrees
```

```
%  10       Sunrise local time, hours since local midnight
%  11       Sunset local time, hours since local midnight
%  12       Cloud fraction from modis 0 (clear skies) to 1 (very cloudy)
%
%  Workspace Variable Names:
%  C := Cloud cover index
%  d := solar altitude
%  jd := number of days since the start of the data year
%  r := relative humidity
%  T_a := surface air temperature
%  T_b := bulk water temperature
%  T_s := surface water temperature
%  U_10 := wind speed 10 m above the surface (m/s)
%  U_z := wind speed z_a m above the surface (m/s)

input_path = strcat(f_dir,Lake,'_data.txt');

%% Programaticaly generate result name
%  Syntax: Lakename_NonDefaultNameOne_NonDefaultValueOne_NonDefaultNameTwo_
%          NonDefaultValueTwo   etc...
%  Note:   The order of the non default parameters in the file name is
%          based on the order of the variables included in input_poss above

results_name = Lake; % Base results name, just the lake name

for j = 1:1:length(input_def(:,1))
    loc = find(strncmp(...                      % Find loc of varargin name
        input_poss{j,1},names,10));             % matching possible inputs
    if isempty(loc) == 0                        % Check for no match
        results_name = strcat(results_name,...  % Load current name
            '_',input_poss{j,1},...             % Append "_NonDefaultName"
            '_',num2str(values{1,loc}));        % Append "_NonDefaultValue"
        results_name = strrep(...               % Replace any "." from
            results_name,'.','_');              % NonDefaultValues with "_"
    end
end

%% Designate result path
%  Syntax: Base directory for all results.  New directory will be added for
%          an individual lake if it does not currently exist.

results_dir = strcat(r_dir,Lake,'\');
results_path = strcat(results_dir,results_name,'.txt');

if exist(results_dir,'dir') == 0
    mkdir(results_dir);
end

%% Find full length of input data if N = 0

i = 0; p = 0; j = 0; max_dist=0;% Initialize counters
```

```
f_id = fopen(input_path);       % Open the file for reading
while ~feof(f_id);              % Continue reading file until end of
    val = load_data(f_id);      % file is reached.
    if ~isempty(val);           % Check if end of file has been reached
        if val(8,1) ~= 0        % Check if row contains a valid
            i = i+1;            % surface temperature measurement
            j = j+1;
            if i == 1;
                k = 1; j = 1;
            end
            if j == 2
                if k+1 > max_dist
                    max_dist = k+1;
                end
                j = 1;
                k = 1;
            end
        else
            k = k+1;
        end
        p = p+1;            % Increment number of hours in file, p
    end
end

if N == 0                           % Check if N = 0
    N = p/24;                       % Store number of days as p/24 hours
elseif N > p/24
    fprintf('Data file does not contain %0.0f days.\n',N);
    N = p/24;
    fprintf('Simulating full data file, %0.0f days.\n',N);
end
k_max = i;                      % Store number of satellite points
fclose(f_id);                   % Close the file

%% Alcantara Constants
%  These constants were defined in Alcantara 2012.
%  A := albedo of water, 0.07
%  a_1 := calibration parameter for phi_s
%  b_1 := calibration parameter for phi_s
%  B_k := empirical coefficient ~= to von Karman's constant, 0.4
%  c_E := coefficient of turbulent exchange
%  c_H := coefficient of turbulent exchange
%  M := vaporization of latent heat
%  epsilon := thermal infrared emissivity of water
%  lambda := Reed correction factor, 0.8

A = 0.07; a_1 = 0.79; b_1 = 1.15; B_k = 0.4; c_E = 1.1*10^-3;
M = 2.501*10^6; c_H = 1.1*10^-3; epsilon = 0.97; lambda = 0.8;

%% Fischer Constants
%  The constants were defined by Fischer 1979
```

```
%  CN := coefficient describing relative effectiveness of mixing due to
%        wind shear and buoyant plumes
%  CT := coefficient describing relative effectiveness of mixing due to
%        the combination velocity scale and entrainment of the quiescent
%        fluid
%  CK := coefficient describing the conversion of Turbulent Kinetic Energy
%        to Thermal Energy.
%        Note: CK is defined earlier in the varargin section.

CN = 1.75; CT = 0.5;

%% Physical Constants
%  The constants are the physical constants used for the air and water.
%  Most are well established in the literature.
%  c_p_water := specific heat of water (J/kgK)
%  c_p_air := specific heat of air (J/kgK)
%  g := gravitational acceleration (m/s^2)
%  alpha := volumetric coefficient of thermal expansion for water (1/C)
%  phi_0 := solar constant (w/M^2)
%  rho := density of water (kg/m^3)
%  rho_a := density of air (kg/m^3)
%  sigma := Stefan-Boltzmann constant (W/m^2 K^4)

c_p_water = 4186;       % (J/kgK)
c_p_air = 1.005*10^3;   % (J/kgK)
g = 9.81;               % (m/s^2)
alpha = 1.8 * 10^-4;    % (1/C)
phi_0 = 1390;           % (W/m^2) Alcantara 2010
rho = 1000;             % (kg/m^3)
rho_a = 1.2;            % (kg/m^3)
sigma = 5.670373*10^-8; % (W/m^2 K^4)

%% Lake Specific Constants
%  These constants will be different for each lake dependant.
%  Lat := Latitude of the data
%         Note: Latitude is stored in the data file
%  Long := Longitude of the data
%         Note: Longitude is stored in the data file
%  P_a := atmospheric surface pressure (Pascal)
%  z := height at which U_10 is desired (m)
%  z_a := height at which U_z is measured (m)

Lat = 34.4652;          % (Degrees)
P_a = 101325;           % (Pa)
z = 10;                 % (m)
z_a = 3;                % (m)

%% Data set constants
%  These constants do not change from dataset to dataset

inter = floor(3600/dt);
```

```
max_leng = (max_dist-1)*inter+1;

%% Allocate variable spaces
%  Matlab allocates space based on the first time it is called.  By
%  creating each of these arrays with a fixed length independently, the
%  memory addresses are stored automatically.  Only used if ItArr = 1.

if ItArr == 1

    r = zeros(max_leng,1);
    year = zeros(max_leng,1);
    month = zeros(max_leng,1);
    day = zeros(max_leng,1);
    time = zeros(max_leng,1);
    T_a = zeros(max_leng,1);
    U_z = zeros(max_leng,1);
    d = zeros(max_leng,1);
    jd = zeros(max_leng,1);
    T_check = zeros(max_leng,1);
    U_10 = zeros(max_leng,1);
    phi_N = zeros(max_leng,1);
    phi_s = zeros(max_leng,1);
    phi_ri = zeros(max_leng,1);
    phi_sf = zeros(max_leng,1);
    phi_lf = zeros(max_leng,1);
    e_a = zeros(max_leng,1);
    w_star = zeros(max_leng,1);
    q_star = zeros(max_leng,1);
    dLdt = zeros(max_leng,1);
    u_star = zeros(max_leng,1);
    err_1 = zeros(max_leng,1);
    err_2 = zeros(max_leng,1);
    err_3 = zeros(max_leng,1);
    dLdt = zeros(max_leng,1);
    dTdt = zeros(max_leng,1);
    dTbdt = zeros(max_leng,1);
    dEdt = zeros(max_leng,1);
    L = zeros(max_leng,1);
    T_s = zeros(max_leng,1);
    CK = zeros(max_leng,1);
    C = zeros(max_leng,1);
    T_b = zeros(max_leng,1);
    Sunrise = zeros(max_leng,1);
    Sunset = zeros(max_leng,1);

end

%% Initialize counter variables

p = 0; i = 0; k = 0; newpoint = 0; redo = 0; CK_signchange = 1;
daynum = 0; constL_old = constL; constL_in = constL;
```

```
%% Begin Simulation
%  The simulation herein solves the coupled system of equations for
%  surface temperature and mixed layer depth.  The general method is
%  described by Fischer 1979.  The simulation continues reading input data
%  until it has found two satellite measurements of surface temperature.
%  Since the temporal resolution on ASOS measurements of ambient parameters
%  is much higher than that of the MODIS satellites, these ambient
%  parameters are used to predict the surface temperature between satellite
%  measurements.  The section between two satellite measurements in solved
%  by assuming a constant wind speed between measurements and iterating
%  over this value until the simulation predicted final temperature matches
%  the second satellite measurement.  Once this temperature matches, the
%  next set of data is read and the simulation continues until the desired
%  length of simulation (N days) is reached.

f_id = fopen(input_path);       % Open data file for reading

while daynum < N                % Define outer loop for number of processed
                                % days.
    divy = 1;                   % Initialize counter variables
    phi_error = 0;
    ct3 = 0;
%     dbloop
    %% Read new values
    %  The reading process varies slightly for the first dataset. See notes
    %  on right of code for more details.

    if redo == 0                            % Check if "redo" flag is
                                            % tripped (=1).
        k = k+1;                            % Increment k
        if p ~= 0;                          % Read new values for second
                                            % set and onward.
            prev = data(leng1,1:12);        % Store the last row of the old
                                            % data.
            i = 1;                          % Initialize counters
            newpoint = 0;
            clear data;                     % Clear previous data
            data(1,1:12) = prev;            % Set the first row of the new
                                            % data equal to the last row of
                                            % the old data.
            data(1,8) = last_T;             % Set the temperature at the
                                            % first satellite point either
                                            % equal to the satellite
                                            % measurement or to the
                                            % simulation result, based on
                                            % fix_T flag.
        else                                % Read new values for first set
                                            % of data.
            T_s_all = 0;                    % Initialize T_s_all.
            sat_loc(k,1) = 1;               % Store the location in the
```

```
                                        % data array of the first
                                        % satellite point.
    end
    while newpoint == 0                 % Import data line by line
        val = load_data(f_id);
        p = p+1; i = i+1;               % Increment counters
        data(i,1:12) = val;             % Save data in data variable
        if val(8,1) == 0;               % Check if imported line has a
                                        % surface temperature
                                        % measurement
            newpoint = 0;               % If it does not, flag as not a
                                        % new satellite point.
        elseif i ~= 1;
            newpoint = 1;               % If it does, flag as a new
                                        % satellite point.
        end

    end

    CK_new = CKdef;                     % Define new CK value (useful
                                        % if iterating over CK instead
                                        % of wind speed).
end

%% Clear previous values
%  Ready the workspace by removing value calculated in previous data
%  sets.  This is important since the time between satellite
%  measurements is rarely the same from set to set.

if ItArr == 0
    clear year; clear month; clear day; clear time; clear T_a;
    clear U_z; clear r; clear d; clear jd; clear T_check; clear diff;
    clear U_10; clear phi_N; clear phi_s; clear phi_ri; clear phi_sf;
    clear phi_lf; clear e_a; clear w_star; clear q_star; clear dLdt;
    clear u_star; clear err_1; clear err_2; clear dLdt; clear dTdt;
    clear L; clear T_s; clear CK; clear C;



else
    if k ~= 1
        r(1:leng,1) = 0;
        year(1:leng,1) = 0;
        month(1:leng,1) = 0;
        day(1:leng,1) = 0;
        time(1:leng,1) = 0;
        T_a(1:leng,1) = 0;
        U_z(1:leng,1) = 0;
        d(1:leng,1) = 0;
        jd(1:leng,1) = 0;
        T_check(1:leng,1) = 0;
```

```
        U_10(1:leng,1) = 0;
        phi_N(1:leng,1) = 0;
        phi_s(1:leng,1) = 0;
        phi_ri(1:leng,1) = 0;
        phi_sf(1:leng,1) = 0;
        phi_lf(1:leng,1) = 0;
        e_a(1:leng,1) = 0;
        w_star(1:leng,1) = 0;
        q_star(1:leng,1) = 0;
        dLdt(1:leng,1) = 0;
        u_star(1:leng,1) = 0;
        err_1(1:leng,1) = 0;
        err_2(1:leng,1) = 0;
        err_3(1:leng,1) = 0;
        dLdt(1:leng,1) = 0;
        dTdt(1:leng,1) = 0;
        dTbdt(1:leng,1) = 0;
        dEdt(1:leng,1) = 0;
        L(1:leng,1) = 0;
        T_s(1:leng,1) = 0;
        CK(1:leng,1) = 0;
        C(1:leng,1) = 0;
        T_b(1:leng,1) = 0;
        Sunrise(1:leng,1) = 0;
        Sunset(1:leng,1) = 0;
    end
end

clear L_with_U; clear U_old; clear T_with_U; clear U_old;

%% Check for data dropouts
clear data2; data2(1,:) = data(1,:);

for i = 2:1:length(data(:,1))
    d2l = length(data2(:,1))+1;
    if data(i,4) - data(i-1,4) == 2
        data2(d2l,:) = data(i,:);
        data2(d2l,4) = (data(i,4)+data(i-1,4))/2;
        data2(d2l,5) = (data(i,5)+data(i-1,5))/2;
        data2(d2l,6) = (data(i,6)+data(i-1,6))/2;
        data2(d2l,7) = (data(i,7)+data(i-1,7))/2;
        data2(d2l,8) = 0;
        data2(d2l,9) = (data(i,9)+data(i-1,9))/2;
        data2(d2l,12) = (data(i,12)+data(i-1,12))/2;
        data2(d2l+1,:) = data(i,:);
    else
        data2(d2l,:) = data(i,:);
    end
end

clear data; data = data2; clear data2;
```

```
%% Organize new values
%  Define the length of the data set and the number of points between
%  each ASOS measurement that must be interpolated.

leng1 = length(data(:,1));
leng = (leng1-1)*inter+1;

if ItArr == 0
    [year,month,day,time,T_a,U_z,r,...  % This function take the
        d,jd,T_check,diff,Sunrise,...   % curent working dataset and
        Sunset,C]=organ_data2(data,...  % interpolates to the desired
        leng1,inter);                   % length.
else
    [year,month,day,time,T_a,U_z,r,d,jd,T_check,Sunrise,Sunset,C]=...
        organ_data3(data,leng1,leng,inter,...
        year,month,day,time,T_a,U_z,r,d,jd,T_check,Sunrise,Sunset,C);
end
if k > 1                                % Reset inital T_check point to
    T_check(1,1) = last_T;               % the desired output from
end                                     % previous set.

%% Allocate space
%  Re-allocate space for all the variables requried for the simulation
%  for the current dataset if ItArr = 0.

if ItArr == 0
    T_b=zeros(leng,1);phi_N=T_b;phi_s=T_b;phi_ri=T_b;phi_sf=T_b;
    phi_lf=T_b; e_a=T_b;w_star=T_b;q_star=T_b;dLdt=T_b;L=T_b;err_1=T_b;
    err_2=T_b; dTdt=T_b; u_star = T_b; dTbdt = T_b; dEdt = T_b;
end

%% Initial values
%  Define inital values for T_s, L, and jd.  Process is slightly
%  different for the first data set due to initial conditions.  See
%  comments to right of code for details.

if k == 1
    T_s(1:leng)=T_check(1:leng)+273;% Set the initial surface
                                    % temperature equal to the
                                    % satellite measurements
    L(1,1) = initL;                 % Set the initial mixed layer depth
                                    % to the pre-defined initial value.
    jd_all(1,1) = jd(1,1);          % Set initial jd to that of the
                                    % first point.
else
    T_s(1,1) = data(1,8)+273;       % Set the initial surface
                                    % temperature equal to the new
                                    % satellite measurement
    L(1,1) = L_old;                 % Set the initial mixed layer depth
                                    % to the final value from the
```

```
                                % simulation on the previous data
                                % set.
end

%% Define Bulk Temperature
%  The temperature of the hypolimnion, herein referred to as the bulk
%  temperature of the lake was taken from USACE measurements.  There
%  are two different possible bulk temperatures currently included in
%  this code.  First, TbType = 0 corresponds to a hard-coded bulk
%  temperature for each month based on the minimum value observed from
%  the USACE data.  Second (preferred), TbType = 1 corresponds to a
%  polynomial bet fit to the USACE measurements based on the data
%  available for each individual year.

if redo == 0                        % Check if redo is flagged (=1)
    for i = 1:1:leng                % Loop through length of dataset
        if TbType == 0              % Hard code bulk temperature method
            floor(month(i,1));      % Determine month number
            if     and(month(i,1)>= 1,month(i,1)<2)==1;T_b(i,1)= 7.4;
            elseif and(month(i,1)>= 2,month(i,1)<3)==1;T_b(i,1) = 7.4;
            elseif and(month(i,1)>= 3,month(i,1)<4)==1;T_b(i,1)=  7.5;
            elseif and(month(i,1)>= 4,month(i,1)<5)==1;T_b(i,1)=  7.9;
            elseif and(month(i,1)>= 5,month(i,1)<6)==1;T_b(i,1)=  7.9;
            elseif and(month(i,1)>= 6,month(i,1)<7)==1;T_b(i,1)=  8.6;
            elseif and(month(i,1)>= 7,month(i,1)<8)==1;T_b(i,1)=  9.5;
            elseif and(month(i,1)>= 8,month(i,1)<9)==1;T_b(i,1)=  9.7;
            elseif and(month(i,1)>= 9,month(i,1)<10)==1;T_b(i,1)=  9.9;
            elseif and(month(i,1)>= 10,month(i,1)<11)==1;T_b(i,1)=10.1;
            elseif and(month(i,1)>= 11,month(i,1)<12)==1;T_b(i,1)=10.6;
            elseif month(i,1)>= 12;T_b(i,1)=11.4;%14
            end
        elseif TbType == 1                  % Yearly bet fit polynomial
                                            % method
            if month(i,1) == 1 ...          % Define number of days in
                    || month(i,1) == 3 ...  % the current month.
                    || month(i,1) == 5 ...
                    || month(i,1) == 7 ...
                    || month(i,1) == 8 ...
                    || month(i,1) == 10 ...
                    || month(i,1) == 12
               mon_days = 31;
            elseif month(i,1) == 4 ...
                    || month(i,1) == 6 ...
                    || month(i,1) == 9 ...
                    || month(i,1) == 11
                mon_days = 30;
            elseif month(i,1) == 2 ...
                    && year(i,1)/4-floor(year(i,1)/4) ~= 0
                mon_days = 28;
            elseif month(i,1) == 2 ...
                    && year(i,1)/4-floor(year(i,1)/4) == 0
```

```
            mon_days = 29;
        else
            fprintf('Error in month number.\n');
            mon_days = 30;
        end

        mon_temp = month(i,1)...                % Define number
            +(day(i,1)+time(i,1)/24)/mon_days;  % decimal month
                                                % number.
        if year(i,1) == 2002        % Define polynomial for 2002.
            B1 = -0.0212;B2 = 0.4682;  B3 = -2.8666; B4 = 13.50;
        elseif year(i,1) == 2003    % Define polynomial for 2003.
            B1 = -0.0344;B2 = 0.6659;  B3 = -3.0146; B4 = 11.00;
        elseif year(i,1) == 2004    % Define polynomial for 2004.
            B1 = -0.0063;B2 = 0.1211;  B3 = -0.3467; B4 =  8.55;
        elseif year(i,1) == 2005    % Define polynomial for 2005.
            B1 = -0.0375;B2 = 0.7913;  B3 = -3.9094; B4 = 13.75;
        elseif year(i,1) == 2006    % Define polynomial for 2006.
            B1 = -0.01623;B2 = 0.32988;B3 = -1.65474;B4 = 10.80;
        elseif year(i,1) == 2007    % Define polynomial for 2007.
            B1 = -0.01623;B2 = 0.32988;B3 = -1.65474;B4 = 10.80;
        elseif year(i,1) == 2008    % Define polynomial for 2008.
            B1 = -0.00876;B2 = 0.16304;B3 = -0.67704;B4 = 10.00;
        elseif year(i,1) == 2009    % Define polynomial for 2009.
            B1 = -0.01014;B2 = 0.21898;B3 = -1.18352;B4 = 10.00;
        elseif year(i,1) == 2010    % Define polynomial for 20010.
            B1 = -0.02341;B2 = 0.50918;B3 = -2.93464;B4 = 11.50;
        elseif year(i,1) == 2011    % Define polynomial for 2011.
            B1 = -0.00879;B2 = 0.20615;B3 = -0.94572;B4 =  8.75;
        elseif year(i,1) == 2012    % Define polynomial for 2012.
            B1 = -0.01946;B2 = 0.44512;B3 = -2.66665;B4 = 14.25;
        elseif year(i,1) == 2013    % Define polynomial for 2013.
            B1 = -0.01623;B2 = 0.32988;B3 = -1.65474;B4 = 12.00;
        elseif year(i,1) == 2014
            B1 = -0.01623;B2 = 0.32988;B3 = -1.65474;B4 = 10.80;
        else                        % Define general polynomial for
                                    % undefined years
            B1 = -0.01623; B2 = 0.32988; B3 = -1.65474;B4 = 10.8;
        end
        T_b(i,1) = B1*mon_temp^3+B2*mon_temp^2+B3*mon_temp+B4;
    end
end;
T_b = T_b + 273;                    % Convert T_b to Kelvin
end

%% Define CK
%  This section defines the new CK value if CK is being iterated over
%  instead of wind speed.

if redo == 0                        % Check if redo is flagged (=1)
    CK(1:leng,1) = CKdef;           % Reset CK to original value
```

```
else
    CK(1:leng,1) = CK_new;              % Change CK to predicted new
end                                     % value.

%% Update number of days processed
%  Number of days processed since the initial date.

daynum = jd(1,1)-jd_all(1,1);


%% Output current point identification
%  Output current data set limits being simulated to command window

if ProFlag == 1
    fprintf(...
     '%0.4f %% Complete, k = %.0f, jd_init = %.4f, jd_fin = %.4f\n',...
        daynum/N,k,daynum,max(jd(:,1))-jd_all(1,1));
end

if k >= 2652
   hihi = 1;
   % T_s was = 0 from daynum = 908.3750 to next sat point
end

%% Define wind speed
%  Define wind speed for current dataset.  If UType = 0, use ASOS
%  measurements.  If UType = 1, use wind speed iteration process.

if UType == 0
    U_10(1:leng,1) = ...                    % Convert measurement to
       u10convert(U_z(1:leng,1),z_a,z,leng);% U10 (Alcantara 2010).
else
    U_10(1:leng,1) = U10min;
end

%% Initialize Counters for inner loop

U_step = 0; wind_flag = 0; U_ct = 1; redo = 0; flag_redo = 0;
Terrors_2 = 5000; CK_ct = 0; CK_flag = 0;

%% Loop over wind speed
%  This inner loop corresponds to attempts to minimize residual error
%  between simulation results and satellite measurements.  The inner
%  loop either iterates over wind speed or CK.

while wind_flag < 1
    %% Reset relevant terms to 0

        phi_N(1:leng,1) = 0;
        phi_s(1:leng,1) = 0;
        phi_ri(1:leng,1) = 0;
```

```
    phi_sf(1:leng,1) = 0;
    phi_lf(1:leng,1) = 0;
    e_a(1:leng,1) = 0;
    w_star(1:leng,1) = 0;
    q_star(1:leng,1) = 0;
    dLdt(1:leng,1) = 0;
    u_star(1:leng,1) = 0;
    err_1(1:leng,1) = 0;
    err_2(1:leng,1) = 0;
    err_3(1:leng,1) = 0;
    dLdt(1:leng,1) = 0;
    dTdt(1:leng,1) = 0;
    dTbdt(1:leng,1) = 0;
    dEdt(1:leng,1) = 0;

%% Define Winter
%  Simulating the surface temperature in the winter is especially
%  difficult since the entire lake is mixing, which corresponds to
%  a mixed layer depth of L = max_L.  At this point, there is not
%  enough energy to change the surface temperature significantly at
%  any point in time.  However, since we know from the satellite
%  meaurements that the surface temperature indeed does vary
%  significantly, an alternative method is used in the winter.
%  Herein, a constant "effective" mixed layer depth is used for the
%  winter, defined as const_L.  The value of const_L can be set
%  using 'const_L',value as an input to this function.  The winter
%  dates were arbitrarily set based on the when the average year
%  tended to not match the satellite measurements well.

cond_flag = 0;          % Set winter flag to 0

if month(1,1) < 3       % Define winter region in January-February
    cond_flag = 2;
end
if month(1,1) == 3      % Define winter region in March
    if day(1,1) < 31
        cond_flag = 2;
    end
end
if month(1,1) == 11     % Define winter region in November
    if day(1,1) > 15
        cond_flag = 2;
    end
end
if month(1,1) > 11      % Define winter region in December
    cond_flag = 2;
end

%% Define important simulation parameters
%  These flux terms are used for the surface energy balance
%  following the method presented by Alcantara 2010.
```

```
%  phi_lf := latent heat flux (W/m^2)
%  phi_N := surface heat flux (W/m^2)
%  phi_ri := longwave flux (W/m^2)
%  phi_s := incident short-wave radiation (W/m^2)
%  phi_sf := sensible heat flux (W/m^2)

%% Spring through fall TKE and CoE solver.
%  Herein a Turbulent Kinetic Energy balance and Conservation of
%  Energy within the mixed layer and at the surface of the mixed
%  layer is used to calculate the development of surface
%  temperature and mixed layer depth.  Some important parameters
%  for this method are defined below:
%  C_D := drag coefficient Alcantara 2010
%  u_star := shear velocity of the wind (m/s) Alcantara 2010
%  w_star := buoyant velocity scale (m/s) Fischer 1979
%  tau_s := shear stress at the air/water interface Alcantara 2010

if cond_flag == 0

    for i = 1:1:leng        % This loop moves the simulation from
                            % the initial timestamp to the final
                            % timestamp of the current dataset.
        if flag_redo == 0   % Check if flag_redo is tripped (=1)
            u_star(i,1) = ...                   % Calculate u_star
                ustar2(U_10(i,1),rho_a,rho);
            [phi_N(i,1),phi_s(i,1),...          % Calculate flux
                phi_ri(i,1),phi_sf(i,1),...     % terms for the
                phi_lf(i,1),e_a(i,1)] = ...     % surface energy
                energybalance(d(i,1),a_1,...    % balance.
                phi_0,b_1,C(i,1),r(i,1),...
                T_a(i,1),epsilon,sigma,...
                T_s(i,1),lambda,rho_a,...
                c_p_air,c_H,U_10(i,1),...
                c_E,M,P_a,A);
            [w_star(i,1),q_star(i,1)] = ...     % Calculate buoyant
                qstar(g,alpha,phi_N(i,1),...    % and combination
                c_p_water,rho,L(i,1),CN,...     % velocity scales.
                u_star(i,1));
            dLdt(i,1) = CK(i,1)*...             % Calcualte dLdt
                q_star(i,1)^3/(CT*...
                q_star(i,1)^2+alpha*(T_s(i,1)...
                -(T_b(i,1)))*g*L(i,1));

            if i <= leng-1                      % Calculate new L
                L_new = L(i,1)+dLdt(i,1)*dt;
                if isnan(L_new)==1;             % Check if new L is
                    L_new = L(i,1);             % not a number.  If
                    dLdt(i,1) = 0;              % yes, set new L
                    err_2(i,1) = 1;             % equal to previous
                end                             % set dLdt = 0, and
                                                % flag err_2 (=1).
```

```
if L_new >= maxL;               % Check if new L is
    L_new = maxL;               % above max_L.  If
    if i ~= 1                   % yes, set new L
    dLdt(i,1) = (L(i,1)...      % equal to max_L,
        -L(i-1,1))/dt;          % calculate dLdt
    end                         % from finite
    err_1(i,1) = 1;             % difference, and
end                             % flag err_1 (=1).


if L_new <= minL;               % Check if new L is
    if minL == 0                % less than minL.
        L(i,1) = constL;        % If yes, set L
        L_new = constL;         % equal to minL,
    else                        % calculate dLdt
        L(i,1) = minL;          % from finite
        L_new = minL;           % difference, and
        err_2(i,1) = 2;         % flag err_2 (=2).
    end
    if i == 1
        dLdt(i,1) = 0;
    else
        dLdt(i,1) = (L(i,1)-L(i-1,1))/dt;
    end
end

%% Calculate CoE terms
%  Calculate the simulation surface temperature.
%  Process is slightly different for i = 1

if i == 1
    if dTbType == 0         % Check if dTbdt should
        dTbdt(i,1) = ...    % be included in CoE.
            -(maxL-L(i,1))/L(i,1)...
            *(T_b(i+1,1)-T_b(i,1))/dt;
    else
        dTbdt(i,1) = 0;
    end
    dEdt(i,1) = ...         % Calculate entrainment
        -((T_s(i,1)-(T_b(i,1)))/L(i,1))*dLdt(i,1);
    dTdt(i,1) = ...         % Calculate dTdt
        phi_N(i,1)/(rho*c_p_water*L(i,1))...
        +dEdt(i,1)+dTbdt(i,1)+phi_error;

else
    if dTbType == 0         % Check if dTbdt should
        dTbdt(i,1) = ...    % be included in CoE
            -(maxL-L(i,1))/L(i,1)...
            *(T_b(i,1)-T_b(i-1,1))/dt;
    else
        dTbdt(i,1) = 0;
```

```
                    end
                    dEdt(i,1) = ...          % Calculate entrainment
                        -((T_s(i,1)-(T_b(i,1)))/L(i,1))*dLdt(i,1);
                    dTdt(i,1) = ...          % Calcualte dTdt
                        phi_N(i,1)/(rho*c_p_water*L(i,1))...
                        +dEdt(i,1)+dTbdt(i,1)+phi_error;
                end

                L(i+1,1) = L_new;            % Calculate new L
                T_s(i+1,1) = T_s(i,1)+...    % Calculate new T_s
                    dTdt(i,1)*dt;
            end
        end
    end

end

%% Conduction solution
%  This section contains code to calculate the surface temperature
%  in the case of pure conduction using the method proposed by
%  Girgis.  However, this solution yields poor agreement with
%  satellite measurements and is not used in the simulation.

if cond_flag == 1;

    f = zeros(100,1); x(:,1) = linspace(0,maxL);

    if ~exist('T_1d')
        f(:,1) = f(:,1)+T_s(1,1);
    else
        f(:,1) = T_1d(:,length(T_1d(1,:)));
    end

    T_1d = zeros(length(x(:,1)),leng);
    T_1d(:,1) = f(:,1); T_1d(:,2) = f(:,1);

    beta = 0.4; eta = 1; N_max = 50;
    A_cond = zeros(leng,1); B_cond = A_cond;
    a = zeros(N_max,1); b = a;

    A_cond(1,1) = eta*(1-beta)*...
        (phi_s(1,1)-phi_ri(1,1))/(rho*c_p_water);
    A_cond(2,1) = eta*(1-beta)*...
        (phi_s(2,1)-phi_ri(2,1))/(rho*c_p_water);
    B_cond(1,1) = ((phi_sf(1,1)+phi_lf(1,1))-...
        beta*(phi_s(1,1)-phi_ri(1,1)))/(rho*c_p_water*alpha);
    B_cond(2,1) = ((phi_sf(2,1)+phi_lf(2,1))-...
        beta*(phi_s(2,1)-phi_ri(2,1)))/(rho*c_p_water*alpha);
    B_prime(2,1) = (B_cond(2,1)-B_cond(1,1))/(dt);

    for i = 3:1:leng
```

```
        [phi_N(i,1),phi_s(i,1),phi_ri(i,1),...
            phi_sf(i,1),phi_lf(i,1),e_a(i,1)] = ...
            energybalance(d(i,1),a_1,phi_0,b_1,C(i,1),...
            r(i,1),T_a(i,1),epsilon,sigma,T_1d(1,i-1),...
            lambda,rho_a,c_p_air,c_H,U_10(i,1),c_E,M,P_a,A);
        t1 = dt*(i-1); t2 = dt*i;
        f(:,1) = f(:,1)+T_1d(:,length(T_1d(1,:)));
        A_cond(i,1) = eta*(1-beta)*...
            (phi_s(i,1)-phi_ri(i,1))/(rho*c_p_water);
        B_cond(i,1) = ((phi_sf(i,1)+phi_lf(i,1))-...
            beta*(phi_s(i,1)-phi_ri(i,1)))/(rho*c_p_water*alpha);
        B_prime(i,1) = (B_cond(i,1)-B_cond(i-1,1))/(t2-t1);
        [T_1d(:,i),a,b] = conduction(eta,alpha,maxL,...
            N_max,T_b(1,1),A_cond(i-1:i,1),B_cond(i-1:i,1),...
            B_prime(i-1:i,1),x,t1,t2,a,b,f);
    end
    T_s(:,1) = T_1d(1,:);
    cond_flag = 0;
end

%% Winter solution
%  This section simulates thes surface temperature in the winter.
%  Herein, the effective mixed layer depth is assumed to be a
%  constant.  Thus, the primary driving force for the surface
%  temperature is the solar radiation.

if cond_flag == 2;

    for i = 1:1:leng        % This loop moves the simulation from
                            % the initial timestamp to the final
                            % timestamp of the current dataset.
        u_star(i,1) = ...                   % Calculate u_star
            ustar2(U_10(i,1),rho_a,rho);
        [phi_N(i,1),phi_s(i,1),...          % Calculate flux
            phi_ri(i,1),phi_sf(i,1),...     % terms for the
            phi_lf(i,1),e_a(i,1)] = ...     % surface energy
            energybalance(d(i,1),a_1,...    % balance.
            phi_0,b_1,C(i,1),r(i,1),...
            T_a(i,1),epsilon,sigma,...
            T_s(i,1),lambda,rho_a,...
            c_p_air,c_H,U_10(i,1),...
            c_E,M,P_a,A);

        if i <= leng-1
            if i == 1

                L(i,1) = constL;            % Set L = const
                dLdt(i,1) = 0;              % Set dLdt = 0
                if dTbType == 0             % Check if dTbdt should
                    dTbdt(i,1) = ...        % be included in CoE.
                        -(maxL-L(i,1))/L(i,1)...
```

```
                    *(T_b(i+1,1)-T_b(i,1))/dt;
                else
                    dTbdt(i,1) = 0;
                end
                dEdt(i,1) = 0;              % Set entrainment = 0
                dTdt(i,1) = ...             % Calculate dTdt
                    phi_N(i,1)/(rho*c_p_water*L(i,1))...
                    +dEdt(i,1)+dTbdt(i,1)+phi_error;
            else

                L(i,1) = constL;            % Set L = const
                dLdt(i,1) = 0;              % Set dLdt = 0
                if dTbType == 0             % Check if dTbdt should
                    dTbdt(i,1) = ...        % be included in CoE.
                        -(maxL-L(i,1))/L(i,1)...
                        *(T_b(i,1)-T_b(i-1,1))/dt;
                else
                    dTbdt(i,1) = 0;
                end
                dEdt(i,1) = 0;              % Set entrainment = 0
                dTdt(i,1) = ...             % Calculate dTdt
                    phi_N(i,1)/(rho*c_p_water*L(i,1))...
                    +dEdt(i,1)+dTbdt(i,1)+phi_error;

            end
            L(i+1,1) = L(i,1)+dLdt(i,1)*dt;     % Calculate new L
            T_s(i+1,1) = T_s(i,1)+dTdt(i,1)*dt; % Calculate new T_s

            if isnan(T_s(i+1,1)) == 1
                hihi = 1;
            end

        end
    end
end

%% Iterate_U == 1
%  This section iterates U using a brute force method to find the
%  minimum residual error.  UType = 1.

if UType == 1
    if U_step == 2

        if itErr == 1   % Check if iterating over error
            Terrors = (T_check(leng,1)+273)-T_s(leng,1);
            ct3 = ct3+1;
            if ct3 > 100    % Reduce step size if having difficulty
                ct3 = 0;    % finding a solution
                divy = divy*0.1;

            end
```

```
        if and(...      % Check for error convergence
                or(abs(Terrors) < 10^-3,divy< 0.00001) == 1,...
                isnan(Terrors) == 0) == 1
            % Reset parameters for next datset
            redo = 0; wind_flag = 1; U_step = 2; divy = 100;
            ct3 = 0;
        end
    else
        con_flag = 0;       % Unflag winter
        wind_flag = 1;      % Flag converged
    end

end

%% Find minimum residual
%  This section checks the residual error with each simulation
%  wind speed value and finds the minimum.

if U_step == 1
    clear Terrors; clear T_1d;
    Terrors(1,1:length(T_with_U(1,:))) = ... % Calculate errors
        T_with_U(leng,1:length(T_with_U(1,:)))...
        -T_check(leng,1)-273;

    GoodWindLoc = ...           % Find best solution
        find(abs(Terrors(1,:)) == min(abs(Terrors(1,:))));
    [rw cl] = size(GoodWindLoc);% Find number of times best
                                % solution occurs
    if cl == 0                  % Check if no solution exists
        redo = 0;               % Unset redo to move on
        T_s(:,1) = -999;        % Flag T_s as being erronious
        T_s(leng,1) = ...       % Set Last T_s value to next
          T_check(leng,1)+273;  % satellite measurement
        L(1:leng,1) = L(1,1);   % set L to a constant
        wind_flag = 1;          % Flag converged
        fprintf(strcat('No',... % Print error message
            'valid solution without changing CK, setting',...
            'L to a constant and moving on.\n'));
    else
        % Reset parameters to the best solution and run through
        % simulation one last time
        T_s(1:leng,1) = T_with_U(1:leng,GoodWindLoc(1,1));
        U_10(1:leng,1) = U_old(1:leng,GoodWindLoc(1,1));
        L(1:leng,1) = L_with_U(1:leng,GoodWindLoc(1,1));
        U_10(1:leng,1) = U_old(1:leng,GoodWindLoc(1,1));
        redo = 0; divy = 100; ct3 = 0;

        U_step = 2;     % Flag that a minimum exists and cont.
    end
end
```

```
%% Calculate all the U10 solutions
%  This section increments and store the various solutions from
%  the different constant U10 values for the current dataset.
%  Once all values have been evaluated, this section is flagged
%  complete and the simulation moves on to the next section.

if U_step == 0
    % Store simulation value for current U10
    T_with_U(1:leng,U_ct) = T_s(1:leng,1);
    L_with_U(1:leng,U_ct) = L(1:leng,1);
    U_old(1:leng,U_ct) = U_10(1:leng,1);

    if flag_redo == 1   % Check if flag_redo is flagged (=1)

        flag_redo = 0;
        redo = 1;
        Terrors(1,1:U_ct) = ... % Store error term
            T_with_U(leng,1:U_ct)-T_check(leng,1)-273;

        GoodWindLoc = ...       % Find best solution
            find(abs(Terrors(1,:)) == min(abs(Terrors(1,:))));
        [rw cl] = size(GoodWindLoc);

        if and(GoodWindLoc(1,1) == 1,...    % Increment CK
                length(T_with_U(1,:)) == 1)
            CK_new = CK(1,1)-1*CK_signchange;
        elseif GoodWindLoc(1,1) <= length(T_with_U(1,:))
            % Flag that all the individual solutions are
            % complete.
            redo = 0;
            U_step = 1;
            T_s(1:leng,1) = ...
                T_with_U(1:leng,GoodWindLoc(1,1));
            L(1:leng,1) =...
                L_with_U(1:leng,GoodWindLoc(1,1));
            U_10(1:leng,1) = ...
                U_old(1:leng,GoodWindLoc(1,1));
        end



    else

        if or(isnan(sum(T_s(1:leng,1))),... % Check for not a
        isinf(abs(sum(T_s(1:leng,1))))) == 0% number or
                                            % infinite number
                                            % errors

            if U_10(1,1) >= U10max;     % Check if U10max has
                U_step = 1;             % been reached.  If yes
                U_ct = U_ct+1;          % flag that individual
```

```
                else                        % solutions are
                    U_10(1:leng,1) = ...    % complete.  If no
                        U_10(1,1)+U10int;   % increment U10.
                    U_ct = U_ct+1;
                end
            else
                U_step = 1;     % Flag complete if in error.
            end
        end
    end



    if redo == 1        % Check if redo is flagged (=1)
        flag_redo = 0;
        wind_flag = 0;
        U_step = 0;
        redo = 0;
        CK(1:leng,1) = CK_new;
    end

end

%% Iterate_U == 2
%  This section iterates U using a root finding method to find the
%  minimum residual error.  UType = 2.

if UType == 2
    if U_step == 2

        if k >= 228
            hihi = 1;
        end

        % Store current solution
        T_with_U(1:leng,U_ct) = T_s(1:leng,1);
        L_with_U(1:leng,U_ct) = L(1:leng,1);
        U_old(1:leng,U_ct) = U_10(1:leng,1);
        Terrors(1,1:length(T_with_U(1,:))) = T_with_U(...
            leng,1:length(T_with_U(1,:)))-T_check(leng,1)-273;

        if length(T_with_U(1,:)) == 3   % Check if this is the
                                        % first iteration with
                                        % U_step = 2
            GoodWindLoc = find(...      % Find whether Umin or Umax
                abs(Terrors(1,1:2)) ... % yielded a better solution
                == min(abs(Terrors(1,1:2))));

            pos_loc = find(Terrors(1,1:2)>0);
            neg_loc = find(Terrors(1,1:2)<0);
            clear pos_val; clear neg_val;
```

```
                for ze = 1:1:length(pos_loc);
                    pos_val(ze,1) = Terrors(1,pos_loc(ze));
                end

                for ze = 1:1:length(neg_loc);
                    neg_val(ze,1) = Terrors(1,neg_loc(ze));
                end

                GoodPosLoc = find(Terrors(1,1:2) == min(pos_val(:,1)));
                GoodNegLoc = find(Terrors(1,1:2) == max(neg_val(:,1)));

                % Linearly interpolate between Umin and Umax to find
                % predicted U10 value needed to hit the satellite point
                T2 = T_with_U(leng,GoodPosLoc);%T_with_U(leng,3);
                T1 = T_with_U(leng,GoodNegLoc);%T_with_U(leng,GoodWindLoc);
                U2 = U_old(leng,GoodPosLoc);%U_old(leng,3);
                U1 = U_old(leng,GoodNegLoc);%U_old(leng,GoodWindLoc);
                T3 = T_check(leng,1)+273;
                slop = (T2-T1)/(U2-U1);
                intp = T1-slop*U1;
            else
                % Linearly interpolate between Umin and Umax to find
                % predicted U10 value needed to hit the satellite point
%                 T2 = T_with_U(leng,length(T_with_U(1,:)));
%                 T1 = T_with_U(leng,length(T_with_U(1,:))-1);
%                 T1 = T_with_U(leng,GoodWindLoc);
%                 U2 = U_old(leng,length(U_old(1,:)));
%                 U1 = U_old(leng,GoodWindLoc);
%                 T3 = T_check(leng,1)+273;
%                 slop = (T2-T1)/(U2-U1);
%                 intp = T1-slop*U1;

                GoodWindLoc = find(...      % Find whether Umin or Umax
                    abs(Terrors(1,:)) ... % yielded a better solution
                    == min(abs(Terrors(1,:))));
                pos_loc = find(Terrors(1,:)>0);
                neg_loc = find(Terrors(1,:)<0);

                for ze = 1:1:length(pos_loc);
                    pos_val(ze,1) = Terrors(1,pos_loc(ze));
                end

                for ze = 1:1:length(neg_loc);
                    neg_val(ze,1) = Terrors(1,neg_loc(ze));
                end
                GoodPosLoc = find(Terrors(1,:) == min(pos_val(:,1)));
                GoodNegLoc = find(Terrors(1,:) == max(neg_val(:,1)));

                T2 = T_with_U(leng,GoodPosLoc);%T_with_U(leng,3);
                T1 = T_with_U(leng,GoodNegLoc);%T_with_U(leng,GoodWindLoc);
                U2 = U_old(leng,GoodPosLoc);%U_old(leng,3);
```

```
    U1 = U_old(leng,GoodNegLoc);%U_old(leng,GoodWindLoc);
    T3 = T_check(leng,1)+273;
    slop = (T2-T1)/(U2-U1);
    intp = T1-slop*U1;
end

if isnan(slop) == 1 ... % Check for errors
        || isfinite(slop) == 0 ...
        || isnan(intp) == 1 ...
        || isfinite(intp) == 0
    U_sat = U1;
else
    U_sat = ...                 % Calculate predicted U10
        (T_check(leng,1)+273-intp)/slop;
end

prev_check = find(...       % Check uniqueness of new U10
    floor(10^3*U_old(1,:)) == floor(10^3*U_sat));

if U_sat > max(U_old(1,:))  % Check if new U10 falls within
    U_sat = max(U_old(1,:));% allowed range.
elseif U_sat < min(U_old(1,:));
    U_sat = min(U_old(1,:));
end

err_flag = 0;

if isempty(prev_check) == 0 ... % Check convergence
        || and(U_ct > 10,...
        abs(U_old(1,length(U_old(1,:)))...
        -U_old(1,length(U_old(1,:))-1))...
        <10^-3) == 1



    % Check for errors
    if isnan(sum(T_s(1:leng,1))) == 1
        err_flag = 1;
    end
    if isinf(abs(sum(T_s(1:leng,1)))) == 1
        err_flag = 1;
    end
    if min(T_s(1:leng,1)) < 250
        err_flag = 1;
    end
    if max(T_s(1:leng,1)) > 350
        err_flag = 1;
    end

    if err_flag == 0 || ct5 == 1;
```

```
                    T_s(1:leng,1) = T_with_U(:,GoodWindLoc);
                    L(1:leng,1) = L_with_U(:,GoodWindLoc);
                    U_10(1:leng,1) = U_old(:,GoodWindLoc);
                    if isnan(T_s(leng,1)) == 1
                        T_s(leng,1) = T_check(leng,1)+273;
                    end
                    ct5 = 0;
                    wind_flag = 1;
                    constL = constL_in;
                    U_ct = 0;
                else
                    ct5 = 1;
                    U_10(1:leng,1) = mean(U_old(1,1:2));
                end


            else
%                     U_10(1:leng,1) = ...    % Increment U10
%                         U_10(1:leng,1)+0.1*(U_sat-U_10(1,1));

                if length(T_with_U(1,:)) == 3   %
                    U_10(1:leng,1) = U_old(1,1)*0.8+0.2*U_sat;
                else
                    U_10(1:leng,1) = U_old(1,U_ct)*0.8+0.2*U_sat;
                end
                U_ct = U_ct+1; redo = 0; U_step = 2; ct3 = 0;
            end
            con_flag = 0;

        end
        % Calculate mini
        if U_step == 1


            if k >= 228
                hihi = 1;
            end

            clear Terrors; clear T_1d;
            Terrors(1,1:length(T_with_U(1,:))) = T_with_U(...
                leng,1:length(T_with_U(1,:)))-T_check(leng,1)-273;
            GoodWindLoc = find(...      % Find whether Umin or Umax
                abs(Terrors(1,1:2)) ... % yielded a better solution
                == min(abs(Terrors(1,1:2))));

            if sign(Terrors(1,1)) ~= sign(Terrors(1,2))
                slop = (T_with_U(leng,2)-T_with_U(leng,1))/...
                    (U_old(leng,2)-U_old(leng,1));
                intp = T_with_U(leng,1)-slop*U_old(leng,1);

                U_sat = (T_check(leng,1)+273-intp)/slop;
```

```
    if U_sat > max(U_old(1,:))
        U_sat = max(U_old(1,:));
    elseif U_sat < min(U_old(1,:));
        U_sat = min(U_old(1,:));
    end
    U_10(1:leng,1) = ...    % Increment U10
        U_old(1,GoodWindLoc)+0.1*(U_sat-U_old(1,GoodWindLoc));
    redo = 0;
    U_step = 2;
    ct3 = 0;
else

    if cond_flag ~= 2

        if CK_ct == 0
            CK_new = CKmin;
        elseif CK_ct >= 1
            CK_new = CK(1,1)+0.1*(CKmax-CKmin);
        end

        if min(abs(Terrors_2(1,:))) > min(abs(Terrors(1,:)))
            CK_store = CK(1,1);
            U_10_store = U_old(1,GoodWindLoc);
            Terrors_2 = Terrors;
        end

        redo = 1;
        CK_ct = CK_ct+1;

        if CK_flag == 1
            T_s(1:leng,1) = T_with_U(:,GoodWindLoc);
            L(1:leng,1) = L_with_U(:,GoodWindLoc);
            U_10(1:leng,1) = U_old(:,GoodWindLoc);
            wind_flag = 1;
            U_ct = 1;
            redo = 0;
        elseif CK_ct > 11 || CK_new < 0 || CK_new > 20
            CK(1:leng,1) = CK_store;
            CK_new = CK_store;
            U_10(1:leng,1) = U_10_store;
            wind_flag = 0;
            redo = 1;
            CK_flag = 1;
        end

    else
        sumphi = mean(phi_N(1:leng,1));
        if min(abs(Terrors_2(1,:))) > min(abs(Terrors(1,:)))
            constL_store = constL;
            U_10_store = U_old(1,GoodWindLoc);
```

```
                Terrors_2 = Terrors;
            end

            if Terrors(1,GoodWindLoc) < 0
                if sumphi < 0
                    constL = constL*1.5;
                else
                    constL = constL*0.5;
                end
            else
                if sumphi < 0
                    constL = constL*0.5;
                else
                    constL = constL*1.5;
                end
            end

            if constL > maxL
                constL = maxL;
            elseif constL < minL
                constL = minL;
            end

            redo = 1;
            CK_ct = CK_ct+1;
            if CK_flag == 1
                T_s(1:leng,1) = T_with_U(:,GoodWindLoc);
                L(1:leng,1) = L_with_U(:,GoodWindLoc);
                U_10(1:leng,1) = U_old(:,GoodWindLoc);
                wind_flag = 1;
                constL = constL_in;
                U_ct = 1;
                redo = 0;
            elseif CK_ct > 11 %|| constL > maxL || constL < minL
                constL = constL_store;
                U_10(1:leng,1) = U_10_store;
                wind_flag = 0;
                redo = 1;
                CK_flag = 1;
            end
        end
    end
end
% Calculate minimum and maximum U10 solutions
if U_step == 0

    % Store solutions
    T_with_U(1:leng,U_ct) = T_s(1:leng,1);
    L_with_U(1:leng,U_ct) = L(1:leng,1);
    U_old(1:leng,U_ct) = U_10(1:leng,1);
    err_flag = 0;
```

```
     % Check for errors
    if isnan(sum(T_s(1:leng,1))) == 1
        err_flag = 1;
    end
    if isinf(abs(sum(T_s(1:leng,1)))) == 1
        err_flag = 1;
    end
    if min(T_s(1:leng,1)) < 250
        err_flag = 1;
    end
    if max(T_s(1:leng,1)) > 350
        err_flag = 1;
    end

    if err_flag == 0
        if length(U_old(1,:)) == 2
            U_step = 1;
            U_ct = U_ct+1;
        else
            U_10(1:leng,1) = U10max;
            U_ct = U_ct+1;
        end
    else            % If there is an error, increase minimum or
                    % decrease maximum until error disappears.
        U_step = 0; % Flag that solution must stay in this step
        err_flag = 0;
        if length(U_old(1,:)) == 1
            U_10(1:leng,1) = U_10(1,1)+0.1;
            if U_old(1,1) > 10
                wind_flag = 1;
                T_s(leng,1) = T_check(leng,1);
            end
        else
            U_10(1:leng,1) = U_10(1,1)-0.1;
            if U_old(1,1) < 0
                wind_flag = 1;
                T_s(leng,1) = T_check(leng,1);
            end
        end
    end
end

if redo == 1    % Check if redo is flagged (=1)
    clear U_old; clear Terrors;
    flag_redo = 0;
    wind_flag = 0;
    U_step = 0;
    redo = 0;
    U_ct = 1;
    U_10(1:leng,1) = U10min;
    CK(1:leng,1) = CK_new;
```

```
            end
        end
        %% Iterate_U == 0
        %  This section iterates an error term to hit the satellite points.
        %  Unused in the simulation.  UType = 0.
        if UType == 0
            Terrors = (T_check(leng,1)+273)-T_s(leng,1);
            phi_error = divy*Terrors/(dt*(leng-1))+phi_error;
            ct3 = ct3+1;
            if ct3 > 100
                ct3 = 0;
                divy = divy*0.1;

            end
            if isnan(phi_error) == 1
                phi_error = 0;
            end
            if and(or(abs(Terrors) < 10^-3,divy< 0.00001) == 1,...
                    isnan(Terrors) == 0) == 1
                redo = 0; wind_flag = 1; U_step = 2;
                divy = 100; ct3 = 0;
            end
        end
    end
    %% Print results to file
    %  This section prints the results to a file.

    if redo == 0

%         if k >= 228
        hihi = 1;
%         run exampleplot.m

%         end
        if ~exist('L_old','var')
            fid_out = fopen(results_path,'wt');
            index1 = leng;
            last_T = T_check(leng,1);
            T_s_all(1:index1,1) = T_s(1:leng,1);
            T_check_all(1:index1,1) = T_check(1:leng,1);
            jd_all(1:index1,1) = jd(1:leng,1);
            for i = 1:1:leng
                fprintf(fid_out,...
                    '%0.10f\t%0.4f\t%0.4f\t%0.4f\t%0.4f\t
%0.4f\t%0.4f\t%0.4f\t%0.4f\t%0.4f\t
%0.4f\t%0.4f\t%0.4f\n',...
                    jd(i,1),U_10(i,1),T_s(i,1),
T_check(i,1)+273,T_b(i,1),L(i,1),CK(i,1),
phi_N(i,1),phi_s(i,1),Sunrise(i,1),
Sunset(i,1),r(i,1),T_a(i,1));
            end
```

```
            sat_loc(k+1,1) = leng;
        else
            fid_out = fopen(results_path,'at');
            index1 = length(T_s_all(:,1));
            index2 = length(T_s_all(:,1))+leng-1;
            last_T = T_check(leng,1);
            T_s_all(index1:index2,1) = T_s(1:leng,1);
            T_check_all(index1:index2,1) = T_check(1:leng,1);
            jd_all(index1:index2,1) = jd(1:leng,1);
            for i = 2:1:leng
                fprintf(fid_out,...
                    '%0.10f\t%0.4f\t%0.4f\t%0.4f\t%0.4f\t%0.4f
\t%0.4f\t%0.4f\t%0.4f\t%0.4f\t%0.4f\t
%0.4f\t%0.4f\n',...
                    jd(i,1),U_10(i,1),T_s(i,1),
T_check(i,1)+273,T_b(i,1),L(i,1),CK(i,1),
phi_N(i,1),phi_s(i,1),Sunrise(i,1),
Sunset(i,1),r(i,1),T_a(i,1));
            end
            sat_loc(k+1,1) = sat_loc(k,1)+leng-1;
        end
        fclose(fid_out);
        CK_signchange = 1;
        %% Check whether to correct final T_s to satellite measurement for
        %  the next dataset or not.
        if fixT == 1 || isnan(sum(T_s(:,1))) == 1  ||
isfinite(sum(T_s(leng,1))) == 0    % Store final T
            last_T = T_check(leng,1);
        else
            last_T = T_s(leng,1)-273;
        end
        L_old = L(leng,1);    % Store final L
    end
end

fclose(f_id);

% Clear unnecessary variables from workspace

clear year; clear month; clear day; clear time; clear T_a; clear U_z;
clear r; clear d; clear jd; clear T_check; clear diff; clear U_10;
clear phi_N; clear phi_s; clear phi_ri; clear phi_sf; clear phi_lf;
clear e_a; clear w_star; clear q_star; clear dLdt; clear u_star;
clear err_1; clear err_2; clear dLdt; clear dTdt; clear L; clear T_s;
clear sat_loc; clear dt; clear T_b;

clear Terrors; clear T_with_U; clear U_old;

toc()

Message = sprintf('Simulation complete for Lake %s \n',Lake);
```

```
% Call the plotting function
% T_res = TKE_method_plot(results_dir,results_name,results_path)

end
```

Appendix B

Post-Processing

This appendix contains the MATLAB code which performs most of the post processing on the simulation results.

```
function [T_RMS,T_check_stat,RMS] =
TKE_method_plot(r_dir,results_name,results_path,varargin)
%% This program performs post-processing on the input results file and
%  plots the desired results.
%   Syntax: TKE_method_plot(r_dir,results_name,results_path,'OptionalName1'
%           ,'OptionalValue1','OptionalName2','OptionalValue2',...,
%           'OptionalNameN','OptionalValueN'))
%   r_dir = the results directory.
%   results_name = the result filename.
%   results_path = the full result path+filename.
%
%   The optional values allow the user to specifiy which plots and settings
%   different from the saved defaults.  The available options are the
%   following:
%
%   NOTE:  '<value>'_xxxx corresponds to whether or not to plot <value> for
%   duration xxxx, where xxxx can be "full", "year", or "week".  "full"
%   correlates to plotting the full simulation, "year" correlates to
%   plotting a single year, and "week" correlates to plotting a single
%   week.  If '<value>'_xxx is set to 1, it will be plotted, if it is 0 it
%   will not be plotted.
%
%   NOTE:  Which week or year to choose is chosen using the option 'year'
%   and 'week' respectively.  If no 'year' or 'week' is included, the
%   default is the first 365 days and the first 7 days included in the
%   result file respectively.
%
%   EXAMPLE:  To plot Ts for the year 2005, the syntax would be:
%             T_RMS = TKE_method_plot(...
```

```
%                                   r_dir,results_name,results_path,...
%                                   'Ts_year',1,'year',2005)
%
%   'Ts_full' = Binary 0 or 1.  Default is 1.
%
%   'Ts_year' = Binary 0 or 1.  Default is 1.
%
%   'Ts_week' = Binary 0 or 1.  Default is 1.
%
%   'L_full' = Binary 0 or 1.  Default is 1.
%
%   'L_year' = Binary 0 or 1.  Default is 1.
%
%   'L_week' = Binary 0 or 1.  Default is 1.
%
%   'U10_full' = Binary 0 or 1.  Default is 0.
%
%   'U10_year' = Binary 0 or 1.  Default is 1.
%
%   'U10_week' = Binary 0 or 1.  Default is 1.
%
%   'Tb_full' = Binary 0 or 1.  Default is 0.
%
%   'Tb_year' = Binary 0 or 1.  Default is 0.
%
%   'Tb_week' = Binary 0 or 1.  Default is 1.
%
%   'CK_full' = Binary 0 or 1.  Default is 0.
%
%   'CK_year' = Binary 0 or 1.  Default is 1.
%
%   'CK_week' = Binary 0 or 1.  Default is 1.
%
%   'Ter_full' = Binary 0 or 1.  Default is 0.
%
%   'Ter_year' = Binary 0 or 1.  Default is 1.
%
%   'Ter_week' = Binary 0 or 1.  Default is 1.
%
%   'year' = Select which year to plot for all '<value>_year' plots.  If an
%   invalid year is selected, the default year will be plotted instead.
%   Default is 0, which will plot the first 365 days in the result file.
%
%   'week' = Select which week to plot for all '<value>_week' plots.  If an
%   invalid week is selected, the default week will be plotted instead.
%   Default is 0, which will plot the first 7 days in the result file.
%
%   'Ts_di_yr' = Binary 0 or 1.  If 1, yearly diurnal trend for Ts is
%   plotted.  Default is 1.
%
%   'L_di_yr' = Binary 0 or 1.  If 1, yearly diurnal trend for L is
```

```
%   plotted.  Default is 0.
%
%   'U10_di_yr' = Binary 0 or 1.  If 1, yearly diurnal trend for U10 is
%   plotted.  Default is 0.
%
%   'Ts_di_mo' = Binary 0 or 1.  If 1, monthly diurnal trend for Ts is
%   plotted.  Default is 0.  To choose which month is plotted use 'month'.
%
%   'L_di_mo' = Binary 0 or 1.  If 1, monthly diurnal trend for L is
%   plotted.  Default is 0.  To choose which month is plotted use 'month'.
%
%   'U10_di_mo' = Binary 0 or 1.  If 1, monthly diurnal trend for U10 is
%   plotted.  Default is 0.  To choose which month is plotted use 'month'.
%
%   'month' = 0 - 13. If 0, plots monthly diurnal trend for all months on a
%   single plot.  If 1-12, plots monthly diurnal trend for a single month.
%   If 13, plots monthly diurnal trend for all months on individual plots
%   for each month.  Default is 0.
%
%   'full_x_style' = 0, 1, or 2.  If 0, x-axis in '<value>_full' plots will
%   be in years.  If 1, it will be in days.  If 2, it will be in hours.
%   Default is 0.
%
%   'year_x_style' = 0, 1, or 2.  If 0, x-axis in '<value>_year' plots will
%   be in years.  If 1, it will be in days.  If 2, it will be in hours.
%   Default is 1.
%
%   'week_x_style' = 0, 1, or 2.  If 0, x-axis in '<value>_week' plots will
%   be in years.  If 1, it will be in days.  If 2, it will be in hours.
%   Default is 1.
%
%   'full_y_style' = 0, 1, or 2.  If 0, y-axis in 'Ts_full' plot will be in
%   K.  If 1, y-axis will be in non-dimensional units T*.  If 2, individual
%   plots for K and T* y-axises will be created.  Default is 0.
%
%   'year_y_style' = 0, 1, or 2.  If 0, y-axis in 'Ts_year' plot will be in
%   K.  If 1, y-axis will be in non-dimensional units T*.  If 2, individual
%   plots for K and T* y-axises will be created.  Default is 2.
%
%   'week_y_style' = 0, 1, or 2.  If 0, y-axis in 'Ts_week' plot will be in
%   K.  If 1, y-axis will be in non-dimensional units T*.  If 2, individual
%   plots for K and T* y-axises will be created.  Default is 2.
%
%   'IncWinter' = Flags whether or not to include winter in yearly average
%   plots.  If = 1, it is included, if = 0 it is not.  Default is 1.
%
%


%% Read varargin inputs
%  input_poss contains the variable name to be replaced by varargin input
%  input_def contains the default value for the corresponding variable name
```

```
%  in input_poss.

input_poss{1,1} = 'Ts_full';    input_def(1,1) = 1;     % Binary, 0 or 1
input_poss{2,1} = 'Ts_year';    input_def(2,1) = 1;     % Binary, 0 or 1
input_poss{3,1} = 'Ts_week';    input_def(3,1) = 1;     % Binary, 0 or 1
input_poss{4,1} = 'L_full';     input_def(4,1) = 1;     % Binary, 0 or 1
input_poss{5,1} = 'L_year';     input_def(5,1) = 1;     % Binary, 0 or 1
input_poss{6,1} = 'L_week';     input_def(6,1) = 1;     % Binary, 0 or 1
input_poss{7,1} = 'U10_full';   input_def(7,1) = 0;     % Binary, 0 or 1
input_poss{8,1} = 'U10_year';   input_def(8,1) = 1;     % Binary, 0 or 1
input_poss{9,1} = 'U10_week';   input_def(9,1) = 1;     % Binary, 0 or 1
input_poss{10,1} = 'Tb_full';   input_def(10,1) = 0;    % Binary, 0 or 1
input_poss{11,1} = 'Tb_year';   input_def(11,1) = 0;    % Binary, 0 or 1
input_poss{12,1} = 'Tb_week';   input_def(12,1) = 0;    % Binary, 0 or 1
input_poss{13,1} = 'CK_full';   input_def(13,1) = 0;    % Binary, 0 or 1
input_poss{14,1} = 'CK_year';   input_def(14,1) = 1;    % Binary, 0 or 1
input_poss{15,1} = 'CK_week';   input_def(15,1) = 1;    % Binary, 0 or 1
input_poss{16,1} = 'Ter_full';  input_def(16,1) = 0;    % Binary, 0 or 1
input_poss{17,1} = 'Ter_year';  input_def(17,1) = 1;    % Binary, 0 or 1
input_poss{18,1} = 'Ter_week';  input_def(18,1) = 1;    % Binary, 0 or 1
input_poss{19,1} = 'year';      input_def(19,1) = 0;    % 0 or yyyy
input_poss{20,1} = 'week';      input_def(20,1) = 0;    % 0 or week#
input_poss{21,1} = 'Ts_di_yr';  input_def(21,1) = 1;    % Binary, 0 or 1
input_poss{22,1} = 'L_di_yr';   input_def(22,1) = 0;    % Binary, 0 or 1
input_poss{23,1} = 'U10_di_yr'; input_def(23,1) = 0;    % Binary, 0 or 1
input_poss{24,1} = 'Ts_di_mo';  input_def(24,1) = 0;    % Binary, 0 or 1
input_poss{25,1} = 'L_di_mo';   input_def(25,1) = 0;    % Binary, 0 or 1
input_poss{26,1} = 'U10_di_mo'; input_def(26,1) = 0;    % Binary, 0 or 1
input_poss{27,1} = 'month';     input_def(27,1) = 0;    % 0-13
input_poss{28,1} = 'full_x_style'; input_def(28,1) = 0; % 0, 1, or 2
input_poss{29,1} = 'year_x_style'; input_def(29,1) = 1; % 0, 1, or 2
input_poss{30,1} = 'week_x_style'; input_def(30,1) = 1; % 0, 1, or 2
input_poss{31,1} = 'full_y_style'; input_def(31,1) = 0; % 0, 1, or 2
input_poss{32,1} = 'year_y_style'; input_def(32,1) = 2; % 0, 1, or 2
input_poss{33,1} = 'week_y_style'; input_def(33,1) = 2; % 0, 1, or 2
input_poss{34,1} = 'IncWinter'; input_def(34,1) = 1;    % Binary, 0 or 1

close all;


%% Flag Outputs

% whichplot, 0 = off, 1 = on
whichplot = [1;     % wind speed
    1;     % converged mixed depth
    1;     % converged surface temperature
    0;     % bulk temperature
    0;     % monthly diurnal trend, T
    0;     % monthly diurnal trend, L
    0;     % monthly diurnal trend, U_10
    1;     % CK
```

```
    0;     % residual
    0;     % broad solution with U = 0 through U = 10
    1;     % total diurnal trend, T
    0;     % average diurnal solar radiation
    1;     % average diurnal net flux
    0;     % average trend divided by net flux
    0];    % average trend divided by solar radiation

if exist(r_dir,'dir') == 0
    mkdir(r_dir);
end

fprintf('Now plotting:\n%s\n',results_path);

%% Find end of file

f_id = fopen(results_path);
i = 0;
while ~feof(f_id); val = fscanf(f_id,[
'%f','%f','%f','%f','%f','%f','%f','%f','%f',
'%f','%f'],11); if ~isempty(val); i = i+1; end
end
eof_out = i;

fclose(f_id);

%% Load results

f_id = fopen(results_path);

jd = zeros(eof_out,1); U_10 = jd; T_s = jd;
T_check = jd; T_b = jd; L = jd; CK = jd; phi_N = jd;
phi_s = jd; ct = 0; Sunrise = jd; Sunset = jd;
jd_t_star = jd;

for i = 1:1:eof_out
    val = fscanf(f_id,[
'%f','%f','%f','%f','%f','%f','%f','%f',
'%f','%f','%f'],11);
    jd(i,1) = val(1,1); U_10(i,1) = val(2,1);
T_s(i,1) = val(3,1); T_check(i,1) = val(4,1);
T_b(i,1) = val(5,1); L(i,1) = val(6,1);
CK(i,1) = val(7,1); phi_N(i,1) = val(8,1);
phi_s(i,1) = val(9,1); Sunrise(i,1) = val(10,1);
    Sunset(i,1) = val(11,1);

    jd_t_star(i,1) = tstar(jd(i,1),Sunrise(i,1),Sunset(i,1));

    if T_check(i,1) ~= 273
        ct = ct+1;
        sat_loc(ct,1) = i;
```

```
    end
end

fclose(f_id);

%% Calculate dt and i_fin
clear day_number;
day_number(:,1) = jd(:,1)-jd(1,1);
dt = round((jd(2,1)-jd(1,1))*24*3600);
i_fin = floor(length(T_s(:,1))/1);

di_len = floor(24*3600/dt);

%% Calculate diurnal averages

T_di_noavg = zeros(di_len,12); cn = T_di_noavg;

for i = 1:1:i_fin%length(T_s(:,1))

    [y,m,d,h,mi,s] = datevec(jd(i,1));
    ti = 3600*h+60*mi+s; % seconds
    ind = floor(ti/dt+1);
    tempz(i,1) = ti/dt+1;
    if i >= 2
        tempz2(i,1) = (jd(i,1)-jd(i-1,1))*24*3600;
    end

    if isnan(T_s(i,1)) == 0 && and(T_s(i,1) > 250,T_s(i,1) < 350)
        T_di_noavg(ind,m) = T_di_noavg(ind,m) + T_s(i,1);
        cn(ind,m) = cn(ind,m)+1;
    end

end

for i = 1:1:length(T_di_noavg(:,1))
    for j = 1:1:length(T_di_noavg(1,:))
        if cn(i,j) ~= 0
            T_di_noavg(i,j) = T_di_noavg(i,j)/cn(i,j);
        end
    end
end

for i = 1:1:length(T_di_noavg(:,1))

    if include_winter == 1
        T_di_noavg_mn(i,1) = mean(T_di_noavg(i,:));
    else
        T_di_noavg_mn(i,1) = mean(T_di_noavg(i,4:11));
    end

end
```

```
ct = 0; avg_day = 0; T_s2 = T_s; T_s3 = T_s; T_s4 = T_s;
day_min = 300; day_max = 250;

for i = 1:1:i_fin
    t_star(i,1) = tstar(jd(i,1),Sunrise(i,1),Sunset(i,1));
    if i >= 523500
        hihi = 1;
    end
    if T_s2(i,1) < day_min
        day_min = T_s2(i,1);
    end

    if T_s2(i,1) > day_max
        day_max = T_s2(i,1);
    end
    if i == i_fin

        if T_s2(i_fin,1) < day_min
            day_min = T_s2(i_fin,1);
        elseif T_s2(i_fin,1) > day_max
            day_max = T_s2(i_fin,1);
        end

        ct = ct+1;
        avg_day = avg_day+T_s2(i,1);
        T_s2(i-ct+1:i,1) = T_s2(i-ct+1:i,1)-(avg_day/ct);
        T_s4(i-ct+1:i,1) = (T_s(i-ct+1:i,1)-day_min)/(day_max-day_min);

    elseif floor(jd(i,1)) == floor(jd(i+1,1))
        avg_day = avg_day+T_s2(i,1);

        ct = ct+1;
    else



        if ct == 0
            T_s2(i,1) = 0;
        else
            if or(abs(avg_day/ct) > 273+1000,abs(avg_day/ct)<200-1000) == 1
                T_s2(i-ct:i,1) = -999;
                T_s3(i-ct:i,1) = -999;
                T_s4(i-ct:i,1) = -999;
            else


                T_s2(i-ct:i,1) = T_s2(i-ct:i,1)-(avg_day/ct);
                T_s3(i-ct:i,1) = T_s3(i-ct:i,1);
                T_s4(i-ct:i,1) = (T_s(i-ct:i,1)-day_min)/(day_max-day_min);
                t_star(i-ct:i,1) = t_star(i-ct:i,1)-ct*2;
```

```
            end
            day_min = 300;
            day_max = 250;
            avg_day = 0;%T_s2(i,1);
            ct = 0;
        end
    end

end


%% Find where L < 0.1m

L(:,2) = 0;

for i=1:1:length(L(:,1))

    if L(i,1) <= 0.01 || isnan(T_s(i,1)) == 1 ||
T_s(i,1) < 250 || T_s(i,1) > 350
        L(i,2) = 1;
    end
end

fd = find(L(:,2) == 1);

for i = 1:1:length(fd(:,1))

    jd_fd(i,1)  = jd(fd(i,1),1);
    T_fd(i,1) = T_s(fd(i,1),1);
    L_fd(i,1) = L(fd(i,1),1);
    T_s2(fd(i,1),1) = -999;
    T_s3(fd(i,1),1) = -999;

    T_s(fd(i,1),1) = 0;


end

T_di = zeros(di_len,12); phi_s_di = T_di; phi_N_di = T_di; L_di = T_di;
cn = T_di; U_di = T_di; jd_di = T_di; phi_error_di = T_di; T_di_tp2 = T_di;

for i = 1:1:i_fin%length(T_s(:,1))

    [y,m,d,h,mi,s] = datevec(jd(i,1));
    ti = 3600*h+60*mi+s; % seconds
    ind = floor(ti/dt+1);

    if isnan(phi_N(i,1)) == 1
        hihi = 1;
    end
```

```
    if and(or(ind == 1,ind == 2),m == 1) == 1
        hihi = 1;
    end

    if T_s2(i,1) ~= -999
        T_di(ind,m) = T_di(ind,m) + T_s3(i,1);
        T_di_tp2(ind,m) = T_di_tp2(ind,m)+T_s4(i,1);
        phi_s_di(ind,m) = phi_s_di(ind,m) + phi_s(i,1);
        phi_N_di(ind,m) = phi_N_di(ind,m) + phi_N(i,1);
%         phi_error_di(ind,m) = phi_error_di(ind,m) + phi_error_store_all(i,1);
        L_di(ind,m) = L_di(ind,m) + L(i,1);
        U_di(ind,m) = U_di(ind,m) + U_10(i,1);
        cn(ind,m) = cn(ind,m)+1;
    end

    jd_di(ind,1) = ti/(3600*24);

end

for i = 1:1:length(T_di(:,1))
    for j = 1:1:length(T_di(1,:))
        if cn(i,j) ~= 0
            T_di(i,j) = T_di(i,j)/cn(i,j);
            phi_s_di(i,j) = phi_s_di(i,j)/cn(i,j);
            phi_N_di(i,j) = phi_N_di(i,j)/cn(i,j);
%             phi_error_di(i,j) = phi_error_di(i,j)/cn(i,j);
            L_di(i,j) = L_di(i,j)/cn(i,j);
            U_di(i,j) = U_di(i,j)/cn(i,j);
        end

        if T_di(i,j) <= 273
            T_di(i,j) = 273;
        end

    end
end

for j = 1:1:length(T_di(1,:))
    T_di_m(:,j) = T_di(:,j)-mean(T_di(:,j));
    T_di_min(1,j) = min(T_di(:,j));
    T_di_max(1,j) = max(T_di(:,j));

    if T_di_min(1,j) <= 0
        T_di_min(1,j) = 0;
    end

    T_di_tp(1:length(T_di(:,1)),j) =
(T_di(:,j)-T_di_min(1,j))/
(T_di_max(1,j)-T_di_min(1,j));
end
```

```
for i = 1:1:length(T_di(:,1))

    if include_winter == 1
        T_di_mn(i,1) = mean(T_di_m(i,:));
        phi_s_di_mn(i,1) = mean(phi_s_di(i,:));
        phi_N_di_mn(i,1) = mean(phi_N_di(i,:));
    else
        T_di_mn(i,1) = mean(T_di_m(i,4:11));
        phi_s_di_mn(i,1) = mean(phi_s_di(i,4:11));
        phi_N_di_mn(i,1) = mean(phi_N_di(i,4:11));
    end


%     phi_error_di_mn(i,1) = mean(phi_error_di(i,:));


end

%% Create residual error array

clear Terrors; clear ct;
Terrors = zeros(nnz(T_check(:,1)-273),1); jd_errors = Terrors; ct = 0;

for i = 1:1:length(T_check(:,1))

    if T_check(i,1)-273 == 0
    else
        ct = ct+1;
        Terrors(ct,1) = T_s(i,1)-T_check(i,1);
        jd_errors(ct,1) = jd(i,1);
    end

end






%% Plot information

scr = get(0,'ScreenSize');          % Store screen size for figures

%% Output

%% U_10
[yr,mth,dy,mi] = datevec(jd(1,1));
    im1 = figure;

    set(im1,'Position',[0 0 scr(3) scr(4)]);
    hold on; clear x; clear y;
```

```
    x(:,1) = (jd(1:i_fin,1)-jd(1,1))/365.25+yr+(mth+dy/31)/12;
%     x(:,1) = 1:1:i_fin';
    y(:,1) = U_10(1:i_fin,1);
    plot(x,y(:,1),'-k','MarkerSize',10,'LineWidth',1);
    ylabel('U_1_0 (m/s)','fontsize',28)
    xlabel('t (hr)','fontsize',28)
    xlim([min(x(:,1)) max(x(:,1))])
    ylim([min(U_10(:,1))-0.1 max(U_10(:,1))+0.1])
%     axis([0 i_fin 0 11])
    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])

    box on
        set(gca,'LineWidth',4)
    r = 150;
    set(gcf,'PaperUnits','inches','PaperPosition',[0 0 1800 900]/r);


    export_fig('filename',strcat(r_dir,results_name,'_u_10','.png'),im1)
%     saveas(im1,strcat(r_dir,results_name,'_u_10'),'png');
    saveas(im1,strcat(r_dir,results_name,'_u_10'),'fig');

    print(gcf,'-depsc2',sprintf('-r%d',r),strcat(r_dir,'sim_U.eps'));
    if whichplot(1,1) == 1
    else
        close(im1);
    end

%% Converged L


    im2 = figure; set(im2,'Position',[0 0 scr(3) scr(4)]);
    hold on; clear y
    max_L = max(L(:,1));
    y(:,1) = L(1:i_fin,1);
    plot(x,y,'-k','MarkerFaceColor',[1,1,1],'MarkerSize',10,'LineWidth',1);
    ylabel('L (m)','fontsize',28)
    xlabel('t (hr)','fontsize',28)
    xlim([min(x(:,1)) max(x(:,1))])
    ylim([-0.1 max_L+0.1])

    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on
    set(gca,'LineWidth',4)
    r = 150;
    set(gcf,'PaperUnits','inches','PaperPosition',[0 0 1800 900]/r);
    print(gcf,'-depsc2',sprintf('-r%d',r),strcat(r_dir,'sim_L.eps'));

    export_fig('filename',strcat(r_dir,results_name,'_L','.png'),im2)
%     saveas(im2,strcat(r_dir,results_name,'_L'),'png');
```

```
    saveas(im2,strcat(r_dir,results_name,'_L'),'fig');

    if whichplot(2,1) == 1
    else
        close(im2);
    end

%% Converged T_s

    im3 = figure;

    set(im3,'Position',[0 0 scr(3) scr(4)]);
    hold on; clear y; clear x2; clear y2; clear x3; clear y3; clear yT;
    y(:,1) = T_s(1:i_fin,1);
    yT(:,1) = T_check(1:i_fin,1);

    for i = 1:length(sat_loc(:,1))
        if sat_loc(i,1) <= i_fin
            x2(i,1) = x(sat_loc(i,1),1);
            y2(i,1) = yT(sat_loc(i,1),1);
            RMS(i,1) = (yT(sat_loc(i,1),1)- y(sat_loc(i,1),1))^2;
        end
    end

    T_RMS = (sum(RMS(:,1))/length(RMS(:,1)))^(1/2);
    titl = sprintf('RMS Error = %0.5f',T_RMS);

    %% Plot temperatures

    plot(x,y','-k','MarkerFaceColor',[1,1,1],'MarkerSize',10,'LineWidth',1);
    plot(x2,y2,'ok','MarkerFaceColor',[0,0,0],'MarkerSize',2);

    %% Make pretty

    ylabel('T_s (K)','fontsize',28)
    xlabel('Years','fontsize',28)
    title(titl,'fontsize',28)
    xlim([min(x(:,1)) max(x(:,1))])
    ylim([0+260 330])

    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on
        set(gca,'LineWidth',4)
    r = 150;
    set(gcf,'PaperUnits','inches','PaperPosition',[0 0 1800 900]/r);
    print(gcf,'-depsc2',sprintf('-r%d',r),strcat(r_dir,'sim_T.eps'));

    export_fig('filename',strcat(r_dir,results_name,'_T','.png'),im3)
    saveas(im3,strcat(r_dir,results_name,'_T'),'fig');
```

```
    if whichplot(3,1) == 1
    else
        close(im3);
    end

%% Plot CK

    im8 = figure;

    set(im8,'Position',[0 0 scr(3) scr(4)]);
    hold on; clear x; clear y;

%     x(:,1) = (jd(1:i_fin,1)-jd(1,1))*24;
    x(:,1) = (jd(1:i_fin,1)-jd(1,1));

    y(:,1) = CK(1:i_fin,1);
    plot(x,y','-k','MarkerFaceColor',[1,1,1],
'MarkerSize',10,'LineWidth',2);

    ylabel('CK','fontsize',28)
    xlabel('t (dy)','fontsize',28)
    xlim([0 (jd(i_fin,1)-jd(1,1))])
    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on

    export_fig('filename',strcat(r_dir,results_name,'_CK','.png'),im8)
    saveas(im8,strcat(r_dir,results_name,'_CK'),'fig');

    if whichplot(8,1) == 1
    else
        close(im8);
    end

%% Yearly diurnal trend, T

    im11 = figure;

    set(im11,'Position',[0 0 scr(3) scr(4)]);
    hold on; clear x; clear y; clear ymin; clear ymax;

    x(:,1) = jd_di(:,1)*24;
    y(:,1) = (T_di_mn(:,1));
    plot(x,((y(:,1))),'-k','MarkerFaceColor',
[1,1,1],'MarkerSize',10,'LineWidth',2); %'-sg'

    ymin = min(min(y(:,:)));
    ymax = max(max(y(:,:)));

    if isnan(ymin) == 1
        ymin = 0;
```

```
    end
    if isnan(ymax) == 1
        ymax = 3;
    end

    ylabel('T_s (^{\circ}C)','fontsize',28)
    xlabel('t (hr)','fontsize',28)
    title('Average Diurnal Trend','fontsize',28)

    axis([0 24 ymin ymax])
    set(gca,'XTick', 0:2:floor(length(x(:,1))));
    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on

    export_fig('filename',strcat(r_dir,results_name,'_T_yr','.png'),im11)
    saveas(im11,strcat(r_dir,results_name,'_T_yr'),'fig');

    if whichplot(11,1) == 1
    else
        close(im11);
    end

star_bins = 100;

ct = 0; avg_day = 0; T_s2 = T_s; T_s3 = T_s;
T_s4 = T_s; clear freq; clear pow;
day_min = 300; day_max = 250;

which_plot_v2 = [1; %frequencies
                 1]; % diurnals


for i = 1:1:i_fin

    if i >= 523500
        hihi = 1;
    end
    if T_s2(i,1) < day_min
        day_min = T_s2(i,1);
    end

    if T_s2(i,1) > day_max
        day_max = T_s2(i,1);
    end
    if i == i_fin

        if T_s2(i_fin,1) < day_min
            day_min = T_s2(i_fin,1);
        elseif T_s2(i_fin,1) > day_max
            day_max = T_s2(i_fin,1);
```

```
        end

        ct = ct+1;
        avg_day = avg_day+T_s2(i,1);
        T_s2(i-ct+1:i,1) = T_s2(i-ct+1:i,1)-(avg_day/ct);
        T_s4(i-ct+1:i,1) = (T_s(i-ct+1:i,1)-day_min)/(day_max-day_min);

    elseif floor(jd(i,1)) == floor(jd(i+1,1))
        avg_day = avg_day+T_s2(i,1);

        ct = ct+1;
    else



        if ct == 0
            T_s2(i,1) = 0;
        else
            if or(abs(avg_day/ct) > 273+50,abs(avg_day/ct)<273-1) == 1
                T_s2(i-ct:i,1) = -999;
                T_s3(i-ct:i,1) = -999;
                T_s4(i-ct:i,1) = -999;
            else
                tempz = (T_s(i-ct:i,1)-day_min)/(day_max-day_min);

                T_s2(i-ct:i,1) = T_s2(i-ct:i,1)-(avg_day/ct);
                T_s3(i-ct:i,1) = T_s3(i-ct:i,1);
                T_s4(i-ct:i,1) = (T_s(i-ct:i,1)-day_min)/(day_max-day_min);
                hihi=1;
            end
            day_min = 300;
            day_max = 250;
            avg_day = 0;%T_s2(i,1);
            ct = 0;
        end
    end


%     T_di2(i,ct) = T_s(i,1);

end

ct = 1; clear T_s5; clear jd_5;

for i=1:1:length(T_s4(:,1))

    if T_s4(i,1) ~= -999
        T_s5(ct,1) = T_s4(i,1);
        jd_5(ct,1) = jd(i,1);
        ct=ct+1;
    end
```

```
end



T_di = zeros(di_len,12); cn = T_di; T_di_m = T_di; T_di_tp = T_di; T_di_mn = T_di;
T_di_min = zeros(1,12); T_di_max = T_di_min;

T_di_star = zeros(2*star_bins,12); cn_2 = T_di_star; jd_di_star = T_di_star(:,1);


for i = 1:1:i_fin%length(T_s(:,1))

    [y,m,d,h,mi,s] = datevec(jd(i,1));
    ti = 3600*h+60*mi+s; % seconds
    ind = floor(ti/dt+1);

    ind2 = floor(jd_t_star(i,1)*star_bins)+1;



    if T_s2(i,1) ~= -999
        T_di(ind,m) = T_di(ind,m) + T_s3(i,1);
        T_di_star(ind2,m) = T_di_star(ind2,m) + T_s3(i,1);
        cn(ind,m) = cn(ind,m)+1;
        cn_2(ind2,m) = cn_2(ind2,m)+1;
    end

    jd_di(ind,1) = ti/(3600*24);
    jd_di_star(ind2,1) = floor(jd_t_star(i,1)*star_bins)/star_bins;

end

% jd_di_star(2001,1) = 2;



for i = 1:1:length(T_di(:,1))
    for j = 1:1:length(T_di(1,:))
        if cn(i,j) ~= 0
            T_di(i,j) = T_di(i,j)/cn(i,j);
        end
    end
end



for i = 1:1:length(T_di_star(:,1))
    for j = 1:1:length(T_di_star(1,:))
```

```
        if cn_2(i,j) ~= 0
            T_di_star(i,j) = T_di_star(i,j)/cn_2(i,j);
        end
    end
end



for j = 1:1:length(T_di(1,:))
    T_di_m(:,j) = T_di(:,j)-mean(T_di(:,j));
    T_di_min(1,j) = min(T_di(:,j));
    T_di_max(1,j) = max(T_di(:,j));

    if T_di_min(1,j) <= 0
        T_di_min(1,j) = 0;
    end

    T_di_tp(1:length(T_di(:,1)),j) =
(T_di(:,j)-T_di_min(1,j))/
(T_di_max(1,j)-T_di_min(1,j));
end

T_di_star_m = zeros(length(T_di_star(:,1)),12);
T_di_star_min = T_di_star_m;
T_di_star_max = T_di_star_m;
T_di_star_tp = T_di_star_m;

for j = 1:1:length(T_di_star(1,:))
    T_di_star_m(:,j) = T_di_star(:,j)-mean(T_di_star(:,j));
    T_di_star_min(1,j) = min(T_di_star(:,j));
    T_di_star_max(1,j) = max(T_di_star(:,j));

    if T_di_star_min(1,j) <= 0
        T_di_star_min(1,j) = 0;
    end

    T_di_star_tp(1:length(T_di_star(:,1)),j) =
(T_di_star(:,j)-T_di_star_min(1,j))/
(T_di_star_max(1,j)-T_di_star_min(1,j));
end



for i = 1:1:length(T_di(:,1))
    if include_winter == 1
        T_di_mn(i,1) = mean(T_di_m(i,:));
        T_di_tp_mn(i,1) = mean(T_di_tp(i,:));
    else
        T_di_mn(i,1) = mean(T_di_m(i,4:11));
        T_di_tp_mn(i,1) = mean(T_di_tp(i,4:11));
```

```
    end
end

T_di_star_mn = zeros(length(T_di_star(:,1)),1); T_di_star_tp_mn = T_di_star_mn;


for i = 1:1:length(T_di_star(:,1))
    if include_winter == 1
        T_di_star_mn(i,1) = mean(T_di_star_m(i,:));
        T_di_star_tp_mn(i,1) = mean(T_di_star_tp(i,:));
    else
        T_di_star_mn(i,1) = mean(T_di_star_m(i,4:11));
        T_di_star_tp_mn(i,1) = mean(T_di_star_tp(i,4:11));
    end
end




clear val;

for i = 0:1:23

    val(i+1,1) = find(jd_di(:,1)*24 == i);
    T_di_mn_lim(i+1,1) = T_di_noavg_mn(val(i+1,1),1);
    T_di_tp_mn_lim(i+1,1) = T_di_tp_mn(val(i+1,1),1);
    jd_di_lim(i+1,1) = jd_di(val(i+1,1),1);

end



clear x; clear y; clear x2; clear y2; ct = 1;

x = jd_di_lim(:,1);
y = T_di_tp_mn_lim(:,1);


% plot(x,y,'ok','MarkerSize',10)


    sun_mn = mean(Sunrise(:,1));
    set_mn = mean(Sunset(:,1));

for i = 1:1:length(x(:,1))

    if x(i,1)*24 < sun_mn

    elseif x(i,1)*24 > set_mn

    else
```

```
        x2(ct,1) = x(i,1);
        y2(ct,1) = y(i,1);
        ct = ct+1;

    end



end


for  i = 1:1:length(jd_di_lim(:,1))

    jd_di_lim_scale(i,1) =
((jd_di_lim(i,1)-jd_di_lim(1,1))*24-sun_mn)/
(set_mn-sun_mn);

end

    clear n; clear dt; clear v; clear fs;
clear x; clear y; clear v; clear v_x;
clear V2; clear V;

dt = 60;

if T_di_tp_mn(length(T_di_tp_mn(:,1)),1) ~= T_di_tp_mn(1,1)
   T_di_tp_mn(length(T_di_tp_mn(:,1))+1,1) = T_di_tp_mn(1,1);
end

if T_di_star_tp_mn(length(T_di_star_tp_mn(:,1)),1) ~= T_di_star_tp_mn(1,1)
   T_di_star_tp_mn(length(T_di_star_tp_mn(:,1))+1,1) = T_di_star_tp_mn(1,1);
end

if jd_di(length(jd_di(:,1)),1) ~= 1
    jd_di(length(jd_di(:,1))+1,1) = 1;
end

if jd_di_star(length(jd_di_star(:,1)),1) ~= 2
    jd_di_star(length(jd_di_star(:,1))+1,1) = 2;
end


% v = T_di_tp_mn(:,1);
% v = T_s5(:,1);
v = T_di_star_tp_mn(:,1);
% v_x = jd_di(:,1);
% v_x = jd_5(:,1);
v_x = jd_di_star(:,1);

for k=1:1:100
```

```
    v(length(v(:,1)):length(v(:,1))
+ length(T_di_star_tp_mn(:,1))-1)
= T_di_star_tp_mn(:,1);
    v_x(length(v_x(:,1)):length(v_x(:,1))
+ length(T_di_star_tp_mn(:,1))-1)
= jd_di_star(:,1)+k;
end

dt = v_x(2,1)-v_x(1,1);

fs = 1/dt;

n = length(v(:,1));


d = 2^(nextpow2(n));
V = fft(v,d)/n;
V2(:,1) = 2*abs(V(1:(floor(d/2)+1))).^2;
x(:,1) = fs/2*linspace(0,1,(floor(d/2)+1));




if which_plot_v2(1,1) == 1

    im1 = figure;
    set(im1,'Position',[0 0 scr(3) scr(4)]);
    loglog(x,V2,'-k','MarkerFaceColor',[1,1,1],'MarkerSize',10,'LineWidth',2)

    ylabel('Signal Power','fontsize',28)
    xlabel('Frequency (1/s)','fontsize',28)
    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on

    axis([10^-2 10^2 10^-15 10^0])

    set(gca,'LineWidth',4)

    r = 150;
    set(gcf,'PaperUnits','inches','PaperPosition',[0 0 1800 900]/r);
%     set(gcf,'PaperUnits','inches','PaperPosition',[0 0 1500 900]/r);
    print(gcf,'-depsc2',sprintf('-r%d',r),strcat(r_dir,'signal1.eps'));

end

freq(:,1) = fs/2*linspace(0,1,(floor(d/2)+1));


[SRT POS] = sort(abs(V(1:(floor(d/2)+1))));
```

```
% [SRT POS] = sort(abs(V));

sintouse = 120;%length(V(:,1));

% for i = 1:1:length(POS(:,1))-sintouse-1%3
%
%     V(POS(i,1),1) = 0;
%
% end
%

ct2 = 0; clear V3; clear freq3; clear H100; clear H; clear loc; clear Hfreq;
for i =1:1:floor(d/2)+1

    if and(freq(i,1) <= 1*10^-3,freq(i,1) >= 10^-5) == 1
        ct2 = ct2+1;
        V3(ct2,1) = V(i,1);
    else
%             V(i,1) = 0;
    end

    if or(or(or(and(freq(i,1) <= 0.5006,
freq(i,1) >= 0.5004),and(freq(i,1)
<= 1.0011,freq(i,1) >= 1.0009)),and(freq(i,1)
<= 1.5016,freq(i,1) >= 1.5014)),and(freq(i,1)
<= 2.0021,freq(i,1) >= 2.0019)) == 1
    else
            V(i,1) = 0;
    end

end

V((floor(d/2)+1)+1:length(V(:,1)),1) = 0;

pow(:,1) = 2*abs(V(1:(floor(d/2)+1))).^2;

if which_plot_v2(1,1) == 1

    im1 = figure;
    set(im1,'Position',[0 0 scr(3) scr(4)]);
    loglog(freq,(real(V(1:d/2+1,1)).^2
+ imag(V(1:d/2+1,1)).^2).^(1/2),'sk',
'MarkerFaceColor',[1,1,1],'MarkerSize',
10,'LineWidth',2)

    ylabel('Signal Power','fontsize',28)
    xlabel('Frequency (1/t^*)','fontsize',28)
    set(gca,'fontsize',28)
    set(gcf,'Color',[1 1 1 ])
    box on
    axis([10^-2 10^2 10^-15 10^0])
```

```
    set(gca,'LineWidth',4)
    r = 150;
    set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
    print(gcf,'-depsc2',sprintf('-r
%d',r),strcat(r_dir,'signal2_tstar.eps'));

end

r = 150;
val6 = find(real(V(:,1)) ~= 0);
stopzz = 0;
f_change = 0;
shift_change = 0;

clear val; clear jd_di_lim_scale2;
clear val2; clear jd_di_2;

val = find(T_di_tp_mn(:,1)
== max(T_di_tp_mn(:,1)));

val3 = find(T_di_tp_mn(:,1)
== min(T_di_tp_mn(:,1)));

jd_di_2(:,1) = (jd_di(:,1)*24-sun_mn)/
(set_mn-sun_mn);
val5 = find(jd_di_2(:,1) == 0);

min_loc = val3;
peak_loc = val;
rise_loc = val5;

clear val; clear val2; clear val3; clear val4;

sum_pow = abs(V(val6(1,1),1))
+abs(V(val6(2,1),1))
+abs(V(val6(3,1),1))
+abs(V(val6(4,1),1));

p_c = 1.34;
A1 = 1;
A2 = 0.22;
A3 = 0.0031;
A4 = 0.0021;
C1 = -0.1+p_c;
C2 = 0.49+p_c*2;
C3 = 0.6+p_c*3;
C4 = 3.9+p_c*4;

B1 = freq(val6(1,1),1)*2*pi();
```

```
B2 = freq(val6(2,1),1)*2*pi();
B3 = freq(val6(3,1),1)*2*pi();
B4 = freq(val6(4,1),1)*2*pi();

sum_phase = phase(V(val6(1,1),1))
+phase(V(val6(2,1),1))
+phase(V(val6(3,1),1))
+phase(V(val6(4,1),1));

for i = 1:1:length(jd_di(:,1))
    y_recon(i,1) = A1*sin(B1*jd_di(i,1)*24-C1)...
        +A2*sin(B2*jd_di(i,1)*24-C2)...
        +A3*sin(B3*jd_di(i,1)*24-C3)...
        +A4*sin(B4*jd_di(i,1)*24-C4);
end

A5 = 1/(max(y_recon(:,1))-min(y_recon(:,1)));
D = min(y_recon(:,1))*A5;

fprintf('A_1: %0.4f\tA_2: %0.4f\tA_3: %0.4f
\tA_4: %0.4f\nB_1: %0.4f\tB_2: %0.4f\tB_3:
%0.4f\tB_4: %0.4f\nC_1: %0.4f\tC_2: %0.4f
\tC_3: %0.4f\tC_4: %0.4f\nD: %0.4f\nA_5:
%0.4f\n',A1,A2,A3,A4,B1,B2,B3,B4,C1,C2,C3,C4,D,A5);
clear y_recon; y_recon
= zeros(length((T_di_tp_mn(:,1))),1);

for i = 1:1:length(jd_di(:,1))
        y_recon(i,1) = A5*(A1*sin(B1*jd_di(i,1)*24-C1)...
            +A2*sin(B2*jd_di(i,1)*24-C2)...
            +A3*sin(B3*jd_di(i,1)*24-C3)...
            +A4*sin(B4*jd_di(i,1)*24-C4))-D;

end

clear y_recon2
for i =1:1:length(jd_di_star(:,1))

            y_recon2(i,1) = A5*(A1*sin(B1*jd_di_star(i,1)-C1)...
            +A2*sin(B2*jd_di_star(i,1)-C2)...
            +A3*sin(B3*jd_di_star(i,1)-C3)...
            +A4*sin(B4*jd_di_star(i,1)-C4))-D;
end

    q = 1;%amount of zero padding,
%1 = no zero padding,
%2 = double the length, etc.
%     vr = ifft(vertcat(V(1:d/2),zeros((q-1)*d,1),V(d/2:d)),q*d)*q*n;
    vr = real(ifft(V)*n);
    y = vr(1:n);
    x = dt*(0:n-1)/3600;
```

```
    if which_plot_v2(2,1) == 1

        im2 = figure; hold on;
        set(im2,'Position',[0 0 scr(3) scr(4)]);

        plot(jd_di_star(:,1),((v(1:201,1))
-min(v(1:201,1)))/(max(v(1:201,1))
-min(v(1:201,1))),'-k','MarkerFaceColor',
[1,0,0],'MarkerSize',10,'LineWidth',4);
        xlabel('t^*','fontsize',28);
        ylabel('T^*','fontsize',28)
        set(gca,'fontsize',28)
        set(gcf,'Color',[1 1 1 ])
        axis([0 2 0 1])
        set(gca,'XTick', 0:0.2:2);
        box on

        set(gca,'LineWidth',4)
        r = 150;

        set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
        print(gcf,'-depsc2',sprintf('-r
%d',r),strcat(r_dir,'diurnal_recon.eps'));

        im2 = figure; hold on;
        set(im2,'Position',[0 0 scr(3) scr(4)]);

        plot(jd_di_star(:,1),((v(1:201,1))
-min(v(1:201,1)))/(max(v(1:201,1))
-min(v(1:201,1))),'-k','MarkerFaceColor',
[1,0,0],'MarkerSize',10,'LineWidth',4);

        plot(jd_di_star(:,1),(y_recon2(:,1)
-min(y_recon2(:,1)))/(max(y_recon2(:,1))
-min(y_recon2(:,1))),'-.b','MarkerFaceColor',
[1,0,0],'MarkerSize',10,'LineWidth',4);

        xlabel('t^*','fontsize',28);
        ylabel('T^*','fontsize',28)
        set(gca,'fontsize',28)
        set(gcf,'Color',[1 1 1 ])
        axis([0 2 0 1])
        set(gca,'XTick', 0:0.2:2);
        box on

        set(gca,'LineWidth',4)
        r = 150;

        hleg1 = legend('Simulation',strvcat(
```

```
'Diurnal','Function'),'Location','EastOutside');
        set(hleg1,'LineWidth',4);

        set(gcf,'PaperUnits','inches','PaperPosition',
[0 0 1800 900]/r);
        print(gcf,'-depsc2',sprintf('-r
%d',r),strcat(r_dir,'diurnal_recon2.eps'));

    end

no_1 = ((v(1:1441,1))-min(v(1:1441,1)))/
(max(v(1:1441,1))-min(v(1:1441,1)));

no_2 = (y(1:1441,1)-min(y(1:1441,1)))/
(max(y(1:1441,1))-min(y(1:1441,1)));

no_avg = no_1-no_2;
no_avg = no_avg.^2;

no_avg_sum = sum(no_avg(:,1));
SEF = (no_avg_sum/(1441-(4+1)))^(1/2);

fprintf('Number of periods:\t\t%.0f\n
Number of sine waves:\t%.0f\n
Standard Error of Fit:\t%.5f\n',k,sintouse,SEF);

%% Create Typical Week T plot

OFFSET_typ_week = 5;

im3 = figure; set(im3,'Position',[0 0 scr(3) scr(4)]);
hold on; clear x; clear y;
[yr,mth,dy,mi] = datevec(jd(1,1));
x(:,1) = (jd(1:i_fin,1)-jd(1,1))-OFFSET_typ_week;
y(:,1) = T_s(1:i_fin,1);

plot(x,y','-k','MarkerFaceColor',[1,1,1],
'MarkerSize',10,'LineWidth',2);
ylabel('T_s (K)','fontsize',28)
xlabel('Days','fontsize',28)
xlim([0 7]); ylim([0+260 330]);
set(gca,'fontsize',28); set(gcf,'Color',
[1 1 1 ]); box on; set(gca,'LineWidth',4);
r = 150; set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
print(gcf,'-depsc2',sprintf('-r%d',r),
strcat(r_dir,'sim1.eps'));

%% Create Typical Week T* plot

im3 = figure; set(im3,'Position',[0 0 scr(3) scr(4)]);
```

```
hold on; clear y;
[yr,mth,dy,mi] = datevec(jd(1,1));
y(:,1) = T_s4(1:i_fin,1);

plot(x,y','-k','MarkerFaceColor',[1,1,1],
'MarkerSize',10,'LineWidth',2);
ylabel('T_s^*','fontsize',28)
xlabel('Days','fontsize',28)
xlim([0 7]); ylim([0 1]);
set(gca,'fontsize',28); set(gcf,'Color',
[1 1 1 ]); box on; set(gca,'LineWidth',4);
r = 150; set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
print(gcf,'-depsc2',sprintf('-r%d',r),
strcat(r_dir,'sim2.eps'));

%% Create Typical Week L plot

im3 = figure; set(im3,'Position',[0 0 scr(3) scr(4)]);
hold on; clear y;
[yr,mth,dy,mi] = datevec(jd(1,1));
y(:,1) = L(1:i_fin,1);

plot(x,y','-k','MarkerFaceColor',[1,1,1],
'MarkerSize',10,'LineWidth',2);
ylabel('L (m)','fontsize',28)
xlabel('Days','fontsize',28)
xlim([0 7]); ylim([-0.1 max_L+0.1]);
set(gca,'fontsize',28); set(gcf,'Color',
[1 1 1 ]); box on; set(gca,'LineWidth',4);
r = 150; set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
print(gcf,'-depsc2',sprintf('-r%d',r),
strcat(r_dir,'sim3.eps'));

%% Create Typical Week U plot

im3 = figure; set(im3,'Position',[0 0 scr(3) scr(4)]);
hold on; clear y;
[yr,mth,dy,mi] = datevec(jd(1,1));
y(:,1) = U_10(1:i_fin,1);

plot(x,y','-k','MarkerFaceColor',[1,1,1],
'MarkerSize',10,'LineWidth',2);
ylabel('U_1_0 (m/s)','fontsize',28)
xlabel('Days','fontsize',28)
xlim([0 7]); ylim([-0.1 10+0.1]);
set(gca,'fontsize',28); set(gcf,'Color',
[1 1 1 ]); box on; set(gca,'LineWidth',4);
r = 150; set(gcf,'PaperUnits','inches',
'PaperPosition',[0 0 1800 900]/r);
```

```
print(gcf,'-depsc2',sprintf('-r%d',r),
strcat(r_dir,'sim4.eps'));

ct = 0;
sumTcheck = 0;
sumTchecksquare = 0;

for i = 1:1:length(T_check(i,1))
    if T_check(i,1) ~= 273
        sumTcheck = sumTcheck+T_check(i,1);
        ct = ct+1;
    end
end

mnTcheck = sumTcheck/ct;

for i = 1:1:length(T_check(i,1))
    if T_check(i,1) ~= 273
        sumTchecksquare = sumTchecksquare
+(T_check(i,1)-mnTcheck)^2;
    end
end

stdevTcheck = (sumTchecksquare/ct)^(1/2);

T_check_stat(1,1) = mnTcheck;
T_check_stat(1,2) = stdevTcheck;

end
```

Bibliography

[1] M. Jin and R. E. Dickinson. Interpolation of surface radiative temperature measured from polar orbiting satellites to a diurnal cycle. *Journal of Geophysical Research*, 104:2105–2116, 1999.

[2] R. C. Wilson, S. J. Hook, P. Schneider, and Schladow S. G. Skin and bulk temperature difference at lake tahoe: A case study on lake skin effect. *Journal of Geophysical Research: Atmospheres*, 118:10332–10346, 2013.

[3] NASA. *Landsat 7 science data users handbook*. NASA, 2000.

[4] NASA. *NASA Land Processes Distributed Active Archive Center (LP DAAC): MOD06L2, MYD06L2, MOD11A1, and MYD11A1*. NASA/USGS, 2014.

[5] W. Abtew and A. Melesse. *Evaporation and Evapotranspiration Measurements and Estimations*. Springer, New York, 2013.

[6] W. G. Large. Sensitivity to surface forcing and boundary layer mixing in a global ocean model: annual-mean climatology. *Journal of Physical Oceanography*, 27:2418–2447, 1997.

[7] E. H. Alcantara, J. L. Stech, J. A. Lorenzzetti, M. P. Bonnet, X. Casamitjana, A. T. Assireu, and E. M. Novo. Remote sensing of water surface temperature and heat flux over a tropical hydroelectric reservoir. *Remote Sensing of Environment*, 114:2651–2665, 2010.

[8] V. L. Nadolski. *Automated Surface Observing System (ASOS) user's guide*. NOAA, DoD, FAA, U.S. Navy, 1998.

[9] S. MacIntyre, W. Eugster, and G. W. Kling. The critical importance of buoyacy flux for gas flux across the air-water interface. *Geophysical Monograph*, 127:134–139, 2002.

[10] B. Jähne and H. Haußecker. Air-water gas exchange. *Ann. Rev. Fluid Mech.*, 30:443–468, 1998.

[11] J. Wang, T. W. Sammis, and V. B. Gutschick. A remote sensing model estimating lake evaporation. *IEEE International Geoscience and Remote Sensing Symposium*, pages 439–442, 2008.

[12] B. Henderson-Sellers. Calculating the surface energy balance for lake and reservoir modeling: a review. *Rev. Geophys.*, 24:625–649, 1986.

[13] I. Reda and A. Andreas. Solar position algorithm for solar radiation applications. Technical report, National Renewable Energy Laboratory, 2008.

[14] R. K. Reed. On estimating insolation over the ocean. *Journal of Physical Oceanography*, 7:482–485, 1977.

[15] I. Y. Fung. On the variability of the net longwave radiation at the ocean surface. *Reviews of Geophysics and Space Physics*, 22:177–193, 1984.

[16] P. R. Lowe. An approximating polynomial for the computation of saturation vapor pressure. *Journal of Applied Meteorology*, 16:100–103, 1976.

[17] Jr Lewis, W. M. A revised classification of lakes based on mixing. *Can. J. Fish Aquat. Sci.*, 40:1779–1787, 1983.

[18] H. B. Fischer, E. J. List, R. C. Y. Koh, J. Imberger, and N. H. Brooks. *Mixing in Inland and Coastal Waters*. Academic Press, London, 1979.

[19] J. Imberger. The diurnal mixed layer. *Limnol. Oceanogr.*, 30:737–770, 1985.

[20] Sebnem Seker-Elci. *Modeling of hydrodynamic circulation and cohesive sediment transport and prediction of shoreline erosion in Hartwell Lake, SC/GA*. PhD thesis, Georgia Institute of Technology, 2004.

[21] D. M. Snider and R. Viskanta. Combined conduction-radiation energy transfer in stagnant water. *Water Resources Research*, 10:939–946, 1974.

[22] S. S. Girgis and A. C. Smith. On thermal stratification in stagnant lakes. *Int. J. Eng. Sci.*, 18:69–79, 1980.

[23] P. P. Niiler and E. B. Kraus. One-dimensional models of the upper ocean. In *Modelling and prediction of the upper layers of the ocean*, pages 143–172, New York, 1977. Pergamon Press.

[24] R. H. Spigel, J. Imberger, and Rayner K. N. Modelling the diurnal mixed layer. *Limnol. Oceanogr.*, 31:533–556, 1986.

[25] E. T. Crosman and J. D. Horel. Sea and lake breezes: a review of numerical studies. *Boundary-Layer Meteorol.*, 137:1–29, 2010.

[26] J. L. Oliver and P. L. Hudson. Thermal and dissolved oxygen characteristics of a South Carolina cooling reservoir. *Water Resources Bulletin*, 23:258–269, 1987.

[27] Tracy Robillard. Pumpback and its effects on Thurmond Lake, 2013. http://balancingthebasin.armylive.dodlive.mil/2013/02/27/pumpback-2/.

[28] Eddie Duncan. Duke Energy Hydro West. Personal Communication.

[29] J. L. Hodges, J. R. Saylor, and N. B. Kaye. A functional form for the diurnal variation of lake surface temperature. Remote Sensing of Environment. pending publication.