Whole-Lake Primary Production Calculator

A thesis submitted in partial fulfilment of the
requirements for the degree of
Master of Science

By

Colin Leong

B.S., Wright State University, 2013

2016

Wright State University

Wright State University
GRADUATE SCHOOL

January 14, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Colin LEONG ENTITLED Whole-Lake Primary Production Calculator BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Travis Doom, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on
Final Examination

Travis Doom, Ph.D.

Yvonne Vadeboncoeur, Ph.D.

Michael Raymer, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and
Dean of the Graduate School

ABSTRACT

Colin Leong. M.S., Department of Computer Science and Engineering, Wright State University, 2015. *Whole-Lake Primary Production Calculator*.

This work describes an implementation of a model for estimation of both benthic and phytoplanktonic primary production in lakes.

The web application makes use of interpolation techniques to allow estimates of primary production using values for photosynthesis/irradiance parameters at only 5 depths. These estimates compare favorably in accuracy with estimates using values listed at over one hundred depths. Validation of the implementation was done by comparison with primary production results from the Northern Temperate Lakes Long Term Ecological Research database.

Contents

1 Introduction **1**

1.1 Introduction 1

1.1.1 Why is primary production (PPR) important? 1

1.1.2 How is PPR determined? 3

1.2 Contribution 4

1.2.1 Packages and software 4

2 Background and literature review **6**

2.1 Lindeman and the study of lakes as ecological systems 6

2.2 Fee, Platt, and Jassby, a proliferation of equations for primary production 9

2.3 Carpenter and estimating the shape of the lake basin 16

2.4 Vadeboncoeuer and the importance of the benthic region 17

2.5 Devlin and validation of the Vadeboncoeur Model: actual bathymetry is ideal 19

3 Benthic primary production **20**

3.1 Calculating benthic primary production 20

3.1.1 Daily BPPR equations 22

3.1.2 Proportion of surface light at depth z 23

3.1.3 Depth of specific light proportion 24

3.1.4 Depth of one percent light. 24

3.1.5 Total littoral area 25

3.1.6 Littoral area at depth z 26

3.1.7 Daily average BPPR at depth z 27

3.1.8 BPPR at depth z, time t 30

3.1.9 Light at depth z, time t 30

3.1.10 Light at surface at time t 31

3.1.11 Fractional littoral area at depth z 31

3.2 Testing and results 31

3.3 Conclusions 33

4 Phytoplanktonic primary production **34**

4.1 Calculating phytoplanktonic primary production 34

4.1.1 Model parameters 34

4.1.2 Thermal layers 35

4.1.3 PPPR at depth z, time t 36

4.1.4 Daily PPPR at depth z 36

4.1.5 Volume at depth z 38

4.2 Testing 38

4.2.1 Initial validation . . . 39
4.2.2 Subsequent validation using NTL LTER data . . . 40
4.2.2.1 Test data assembly and collation . . . 40
4.2.2.2 Validation against NTL LTER data . . . 42
4.3 Discussion . . . 42

5 Making the Vadeboncoeur Model publicly accessible 44
5.1 Implementing the Vadeboncoeur Model in Python . . . 44
5.1.1 Objects and program structure . . . 44
5.1.2 Interpolation with SciPy . . . 46
5.1.3 Reading in lake data With xlrd . . . 47
5.1.4 Open-source development using GitHub . . . 48
5.2 Building the web application . . . 48
5.2.1 Using the Flask Microframework . . . 48
5.2.1.1 Receiving uploaded files from users . . . 49
5.2.1.2 Passing data between views using jsonpickle . . . 50
5.2.1.3 Exporting data for download . . . 51
5.2.2 Prototyping with PythonAnywhere . . . 52
5.2.3 VMware cluster-based execution . . . 52
5.2.4 Deployment using NGINX and Gunicorn . . . 52
5.3 Discussion . . . 52
5.3.1 Lessons learned . . . 52
5.3.1.1 Coupling, and difficulties in adding or removing data fields . . . 53
5.3.1.2 Including web consideration in original design: consider making objects JSON-serializable . . . 53
5.3.2 Future improvements . . . 53
5.3.2.1 Improved user control and user feedback . . . 54
5.3.2.2 Graphing and visualization . . . 54
5.3.2.3 Online data entry . . . 54

6 Conclusions and Future Work 55
6.1 Conclusions . . . 55
6.1.1 Results consistent with expectations; further validation may be desirable . . . 55
6.2 Future work . . . 56

A Project implementation details and source code 58
A.1 Project implementation details . . . 58
A.1.1 Libraries . . . 58
A.2 Program Architecture . . . 59
A.3 Model Source Code (Python) . . . 59
A.3.1 pond.py . . . 60

A.3.2 photosynthesis_measurement.py 97
A.3.3 benthic_photosynthesis_measurement.py 99
A.3.4 phytoplankton_photosynthesis_measurement.py 101
A.3.5 pond_shape.py 107
A.3.6 bathymetric_pond_shape.py 110
A.4 View Code (Python) 122
A.4.1 flask_app.py 123
A.5 Controller Code (Python) 137
A.5.1 data_reader.py 137

B Program validation process **150**
B.1 Data sources 150
B.2 Validating benthic primary production 150
B.2.1 Data used for testing BPPR. 151
B.2.2 Selecting parameter values for interpolation testing 173
B.2.3 Devlin output values 174
B.3 Testing phytoplanktonic primary production 175
B.3.1 Data sources 175
B.3.1.1 Phyte_90s.csv 175
B.3.1.2 PI2000sPhoto.csv 179
B.3.2 Phytoplanktonic primary production test data assembly process 183
B.3.3 Checking for problem values 183
B.3.4 Match with NTL light extinction data 189
B.3.5 Calculate day length, noon light 192
B.3.6 Layer depths 195

C Template user input file **196**
C.1 Specification 197
C.2 Sheet 1: pond_data 207
C.3 Sheet 2: benthic_photo_data 207
C.4 Sheet 3: phytoplankton_photo_data 208
C.5 Sheet 4: shape_data 209

D Parameter addition checklist **210**

E Program requirements **212**

Bibliography **215**

List of Figures

1.1 Regions of photosynthesis . . . 2

2.1 Energy flow in food cycle . . . 8
2.2 Ecological pyramid . . . 8
2.3 Fee P/I curves . . . 10
2.4 P/I curves . . . 13
2.5 Depth ratio cross sections . . . 16
2.6 Vadeboncoeuer 2008 Equations . . . 18
2.7 Vadeboncoeuer 2008 Parameters . . . 18

3.1 Calculating littoral area . . . 26
3.2 Littoral area approximation . . . 26
3.3 BPPRZ . . . 28
3.4 BPPRZ, Fractional-Hour Intervals . . . 28
3.5 BPPRZ, Full-Hour Intervals . . . 29

4.1 Thermal Layers . . . 35
4.2 Volume Calculation . . . 38
4.3 PPPR at z and t . . . 43

5.1 Comparison of Spline/Linear Interpolation with Ik . . . 46
5.2 Comparison of Spline/Linear Interpolation with Pmax . . . 47

B.1 Recorded light values example . . . 192

List of Tables

3.1 BPPR Equations . . . 22
3.2 BPPR test results . . . 32
3.3 BPPR with interpolation, 11 points . . . 32
3.4 BPPR with interpolation, 5 points . . . 32

B.1 pprinputs_Colin.xlsx . . . 151
B.2 Example data for interpolation testing . . . 173
B.3 pproutputs_Colin.xlsx . . . 174
B.4 Phyte_90s.csv . . . 176
B.5 PI2000sPhoto.csv . . . 179
B.6 Assembling data: initial problem check . . . 184
B.7 Assembling data: matching with NTL Light Extinction database . . 190
B.8 The result of finding peak PAR (noon light), length of day, and stratum interval . . . 194
B.9 Stratum/layer depths for each lake and day. . . . 195

C.1 Input file specification . . . 197
C.2 "pond_data" sheet, empty . . . 207
C.3 "pond_data" sheet, filled . . . 207
C.4 "benthic_photo_data" Template, empty . . . 207
C.5 "benthic_photo_data" Template, empty . . . 207
C.6 "phyto_photo_data" sheet, empty . . . 208
C.7 "phyto_photo_data" sheet, filled . . . 208
C.8 "shape_data" sheet, empty . . . 209
C.9 "shape_data" sheet, filled . . . 209

Abbreviations

PAR	**P**hotosynthetically **A**vailable **R**adiation
PP	**P**rimary **P**roduction
PPR	**P**rimary **P**roduction **R**ate
BPPR	**B**enthic **P**rimary **P**roduction **R**ate
PPPR	**P**hytoplankton **P**rimary **Pr**oduction **R**ate

Acknowledgements

In no particular order...

Dr. Doom, my advisor, without whose patient encouragement and wise advice this could not have been done, and who made time over holidays and weekends to help revise this thesis. Thank you for your kind guidance.

Dr. Vadeboncoeur, upon whose work this project is based, for giving me this opportunity. Your longsuffering explanations (and re-explanations) of the science made this possible.

Dr. Raymer, who included me in the Bioinformatics Research Group and mentored me in presentation and writing, and took me to the GL-Bio conference, opening my eyes to the Wonder of Bioinformatics. You made learning fun!

Mrs. Christine Leong, my sister-in-law and friend, who used her graphic design skills to create figures much more professional than mine.

And of course, my family and friends, for their support. To list the things they have done would take far too long, and I have but one page to acknowledge them, so let us just say that I especially appreciated the food.

Thank you all.

Introduction

1.1 Introduction

In the context of Limnology, "primary production" (PP) is the rate that "primary producers" (organisms that use photosynthesis such as plants or algae) "fix" carbon dioxide (convert it to carbohydrates). "Net Primary Production" is the amount of growth or increase in mass of these primary producers. The rate at which primary production occurs, the PPR, is parameterized in terms of milligrams of carbon per square meter per day [Vadeboncoeur et al., 2008].

This project provides two contributions: (1) an online, software-based tool for use by Limnologists in estimating whole-lake primary production, and (2) testing of improvements to previous methods that allow greater accuracy with fewer data points. The theoretical basis for this project is the work of Vadeboncoeur et al. [2008], which provides the underlying mathematical model, incorporating also refinements suggested by Devlin *et al.* [Devlin, 2015].

1.1.1 Why is primary production (PPR) important?

Since plants and other primary producers such as phytoplankton form the base of the entire food web, estimating PPR can be useful for various applications. For example, the United States Geological Survey (USGS) states that "Primary Production controls fisheries yield", and that "The capacity of aquatic systems like estuaries to sustain animal populations is set, in part, by the rate phytoplankton produce new plant matter" [USGS, 1997]. In other words, more primary production may mean there will be more fish to catch! How, then, is this calculation done?

PP in a water body occurs in two primary zones: the benthic zone is the region of the lake at the bottom or lowest level (including near the shoreline). Primary production in the benthic region is known as benthic primary production (BPP), which mainly comes from bottom-dwelling algae known as periphyton. The habitat in which these algae can live is known as the **littoral zone**. The littoral zone is defined by how deep light reaches. In the model, the littoral zone is the area of lake-bottom sediment that receives 1% or more of the light that the surface receives. Of course, the deeper light can penetrate, the larger the littoral zone. If the littoral zone is larger, BPP will generally play a larger role in the Primary Production of the lake in comparison to pelagic primary production.

The pelagic zone consists of the region of water not close to the bottom or shore. Also known as the "water column", this is the zone of water that extends from the surface of the water body downwards. In shallow lakes this zone extends all the way down to the benthic zone. In deep lakes, the bottom limit is the depth of one percent light. Primary production in this zone, known as pelagic primary production or PPP, is generated not by periphyton, but phytoplankton. Phytoplankton occupy the water above the periphyton, and as their mass increases, they cause increased light attenuation, decreasing light availability for the periphyton below. Most research before the Vadeboncoeur model focused on pelagic primary production.

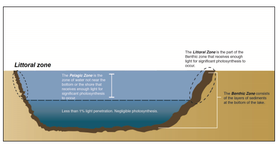

FIGURE 1.1: Regions of photosynthesis in a typical lake.

1.1.2 How is PPR determined?

There are three main determining factors for the PPR of a lake: morphometry, nutrient availability, and light availability.

Morphometry describes the shape of the lake basin, the terrain that holds the water of the lake. In deep lakes with steep sides (e.g. Lake Tanganyika), most of the lake bottom is too deep for light to penetrate, making the littoral zone a narrow band around the edges of the lake. In shallow lakes, or lakes with gentle slopes, the littoral zone is much larger, and so the proportion of PPR supplied by periphyton is larger.

Nutrient availability is classified into several trophic states. Trophic states are simply classifications of the quantities of nutrients available. They range from Oligotrophic, with the lowest level of nutrients, all the way up through Hypereutrophic, with the highest level. Nutrient levels can be a limiting factor on BPP, but increasing them in a water body will also increase production in the upper levels of the lake, which will increase light attenuation.

Light attenuation is the reduction in available light and therefore energy for biological processes, generally as a function of depth. If the water is clearer, light is less attenuated and penetrates deeper, increasing the size of the littoral zone. If the water is murkier, light cannot penetrate as deeply, thus the littoral zone is decreased, thus the total BPP is decreased. Light attenuation can be caused by various things, but an important source of light attenuation is the growth of microscopic organisms known as phytoplankton in the water. These factors are all accounted for using the light attenuation coefficient, which includes both the light attenuation from phytoplankton, and from other sources.

In the past, BPP was thought to be a relatively unimportant component of PP compared to PPP. As a result, research has focused primarily on PPP, with few estimates of BPP. There has been a reported "paucity of estimates of BPP across different lake size gradients" [Vadeboncoeur et al., 2008]. Vadeboncoeur's work demonstrated that BPP can dominate in certain conditions, particularly when lakes are shallow.

1.2 Contribution

Currently, there are no publicly available online tools for calculation of benthic and pelagic primary production in inland water bodies. This thesis describes a convenient online tool to make the Vadeboncoeur model available for public use. Researchers will be able to use this tool for estimates of benthic, pelagic, and combined whole-lake primary production.

There are also some issues with the original model that this project attempts to address. In the original Vadeboncoeur model, photosynthesis and irradiance parameter values were listed at every 0.1 meter interval. For even a relatively shallow lake, this would require the collection of measurements at over a hundred depths. This project investigates to what extent interpolation algorithms can compensate for a reduction of required photosynthesis/irradiance measurements.

1.2.1 Packages and software

To accomplish this goal of creating an online tool to calculate primary production, the Vadeboncoeur model was implemented in Python using the following packages:

- the Flask web framework.
- the werkzeug library for uploading and downloading files.
- the Python Excel library for working with CSV and MS Excel files.

- The SciPy library for scientific and graphing functions.

Prototyping was accomplished using www.pythonanywhere.com.

The project is open source, provided under a standard MIT license. Source code can be found on GitHub at the Project GitHub Repository.

Background and literature review

2.1 Lindeman and the study of lakes as ecological systems

Early systems for study of the ecology of lakes did not consider the lake as a whole system, mostly concerning themselves with distributions of species. The non-living "environment" of the lake was studied separately, and communities of creatures were considered to be "co-acting" with one another, but "reacting" to the non-living environment around them. These approaches were limited, only studying specific aspects of the complex systems involved.

In his seminal work "The Trophic-Dynamic Aspect of Ecology", Lindeman discusses a new way of thinking about ecosystems which he calls the "Trophic-Dynamic Viewpoint." [Lindeman, 1942] Rather than studying components of the system in isolation (e.g. individual species), Lindeman argues that the lake should be thought of as a "primary ecological unit", and studied as a whole.

Furthermore, rather than considering living organisms separately from their non-living environment, Lindeman argues that "analysis of food-cycle relationships indicate that a biotic community cannot be clearly differentiated from its abiotic environment, the ecosystem is hence regarded as the more fundamental ecological unit".

Lindeman gives the example of "a slowly dying pondweed covered with periphytes [microorganisms], some of which are also continually dying". This example shows the difficulty in drawing a line between "living" and "non-living" parts of the lake - should a dead leaf be included as part of the living community? Does the community of periphytes that feed on the leaf, but also on the biomass of their dead comrades, co-act or react with one another? Any such line must be arbitrary in nature.

The Trophic-Dynamic viewpoint, therefore, abandons the idea of living communities in a nonliving environment and instead studies the lake in terms of *energy flow*. Energy from the sun enters the lake in the form of light, and is absorbed by "primary producers" using photosynthesis. The primary producers use this energy to increase their mass (generally measured in terms of *milligrams of carbon*), but are eaten by other organisms. These organisms themselves are eaten by predators, which may have creatures that prey upon them in turn. At each level the organisms absorb some energy from lower-level creatures.

Of course, the direction of energy flow is not exclusively in one direction. For example, some is lost during the digestive process, and higher-level predators eventually die and return their biomass to the system. Thus, energy flows in an ecosystem form a cycle known as the *food cycle* or *food web*.

In Figure 2.1, Lindeman shows a generalized picture of food-cycle relationships. Solar radiation feeds phytoplankton, which feed zooplankton, and so on. The energy levels at each level n are denoted by Λ_n. Though not included in this figure, in the paper Λ_0 is the energy available from solar radiation.

The "levels" of this food cycle are more formally known as "trophic levels", with plants generally forming the first trophic level, herbivores forming the second trophic level, carnivores preying upon them forming the third and subsequent levels, and so on. The numbers of organisms in each level typically decrease rapidly as the level increases, forming a shape known as an Ecological pyramid.

Using this energy-flow paradigm, Lindeman investigates the relationships between trophic levels, describing efficiency, loss, growth, and so on. This work shows that

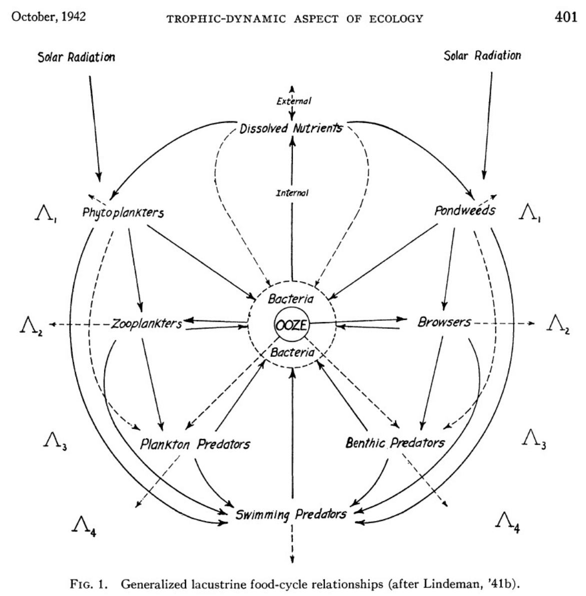

FIGURE 2.1: The complicated energy-flow relationships in a typical lake Lindeman [1942]

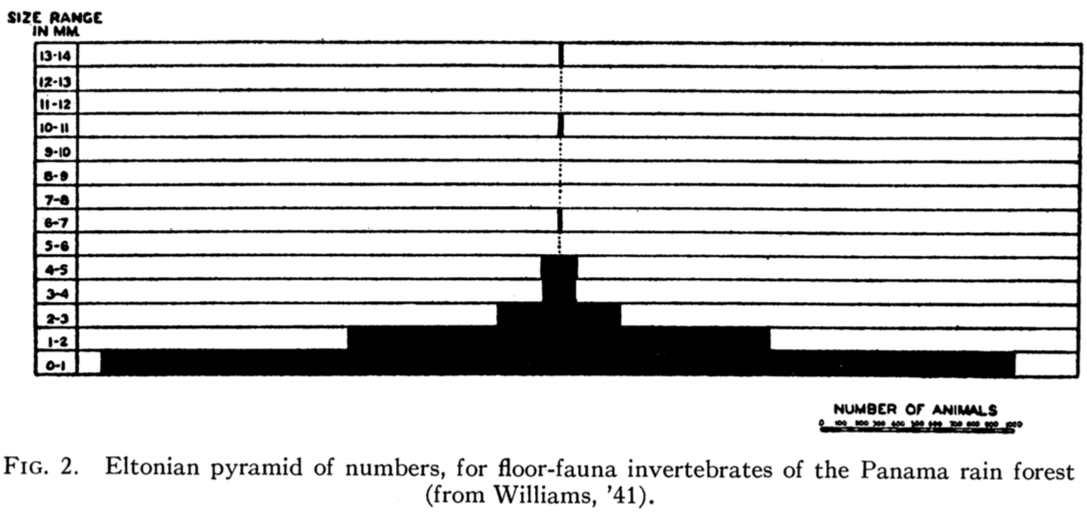

FIGURE 2.2: Representative Ecological pyramid (referred to here as an "Eltonian" pyramid) Lindeman [1942]

λ_1, the rate of primary production by photosynthetic organisms, determines the productivity of higher trophic levels - a critical insight in the field of limnology!

While the concept that primary production drives higher levels of the food cycle may seem intuitively obvious (e.g. more algae means more food for fish, therefore more fish survive), this was the first time it had been confirmed by the scientific method. In fact, this is one of the founding works in the field of Limnology, with over 2500 citations listed on Google Scholar, and laid the foundation for studying lakes as *systems*. According to later authors, Lindeman "explicitly formulated the theoretical basis of dynamical ecology"[Fee, 1971].

Lindeman died during the process of publishing his influential paper, and it was released posthumously. The American Society of Limnology and Oceanography gives an annual award in his honor to this day.

2.2 Fee, Platt, and Jassby, a proliferation of equations for primary production

Lindeman established the importance of primary production, but there was still much to be done. Methods for calculating this critical figure were mostly "empirical and time-consuming" [Fee, 1969]. In some methods, measuring the primary production of a large water body required a ship to go out and place many light and dark bottles in specific locations. Worse, these measurements could only be done during specific times of day.

To provide an alternative to these impractical methods, scientist Everett Fee published a paper entitled "A numerical model for the estimation of photosynthetic production, integrated over time and depth, in natural waters". In this paper, Fee incorporates previous research and sets forth a general model for estimating primary production *mathematically.* The paper describes a process for integrating primary production in the entire water column of a lake (from the surface to the bottom), using a *P/I curve.* That is, calculating primary production **P** as a function of irradiance (light) **I**.

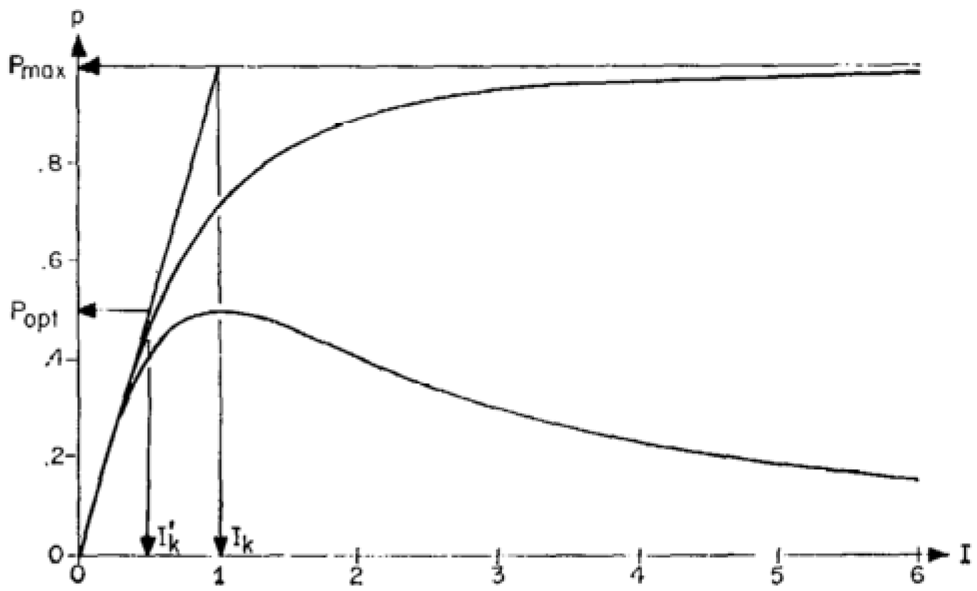

FIGURE 2.3: Two P-I curves, using different parameters for the rate of photosynthesis. Original caption cropped. Fee [1969]

In a later paper, Fee implements a version of his model as a computer program written in FORTRAN (with specifications for punch-card format) [Fee, 1971]. This may be the first computer program ever published for the purpose of estimating primary production. Fee summarizes his algorithm as follows:

"By measuring the transparency of a given water body, it is possible to use [this equation] to compute the photosynthetic rate at any depth if the absolute irradiance at the surface is known. This photosynthesis/depth curve may be integrated in various ways; numerical integration is much the most convenient if a digital computer is available. This gives the integral rate at an instant in time. By measuring surface irradiance at a number of times during the day and repeating the depth integration, a collection of depth integrals is computed. The daily rate is obtained by integrating the depth integrals over time; again, any convenient integration procedure may be used."

$$p = \frac{i}{[(1+i^2)(1+(ai)^2)^n]^{\frac{1}{2}}} \tag{2.1}$$

Equation (2.1) is the Fee model equation, which calculates the primary production rate at a specific light intensity. This is then used in Equation (2.2) to calculate total primary production in the whole water column for the entire day.

$$\sum\sum P = \frac{P_{opt}\lambda}{\epsilon}\left[\frac{\sigma}{2}\int_{-1}^{1}\int_{0}^{I_0(x)/\sigma I'_k}\frac{1}{[(1+y^2)\,(1+a^2y^2)]^{1/2}}dydx\right] \tag{2.2}$$

In both Equation (2.1) and Equation (2.2), the parameters are...

- P = the rate of carbon uptake per unit volume and time. [Note:In subsequent research, this is referred to as the *primary production rate*]
- λ = the daylength, taken with zero at midday.
- P_{opt} = the "experimentally measurable optimum rate of photosynthesis per unit volume of water." It describes the "leveling-off" point of the curve. Note: later formulations of the P/I curve equation do not use P_{opt}, instead using P_{max} directly [Fee, 1969]
- P_{max} = the maximum rate of photosynthesis or carbon uptake per unit volume and time.
- σ = an "auxiliary function" of a and n, used to calculate P_max and I_k from experimentally-measurable parameters [Fee, 1969].
- ϵ = light extinction coefficient of the water body. Note: used to calculate how much light penetrates to a specified depth.

- $I_0(x)$ = the absolute surface irradiance at normalized time x = 2t/λ.

- I_k'a = light saturation parameter (see Fig. 1).

- z =depth.

- t = time.

- x, y = "dummy variables" of the Fee model.

- a, n =parameters of the Fee model [Fee, 1969].

These parameters are used to describe the shape of the Production/Irradiance curve for the lake, then said curve is used to find PPR at different depths. The Vadeboncoeuer model on which our project is based uses much the same approach - Fee's work remains influential to this day.

Models and equations to describe and use the P/I curve proliferated over the years. In Jassby and Platt [1976], eight different equations are reformulated to use just two parameters - one describing the initial slope of the curve, and one that, like the Fee model's "P_opt", describes the highest rate of photosynthesis (i.e. the point where the curve "levels off" (Figure 2.4).

From Jassby and Platt's tests, the most useful equation is found to be based on a hyperbolic tangent function (Equation (2.3)).

$$P^B = P_m^B * tanh(\alpha * I/P_m^B) \tag{2.3}$$

where...

- P_m^B is the optimal (highest) primary productivity in terms of chlorophyll biomass (milligrams of Carbon / milligrams of chlorophyll / hour)

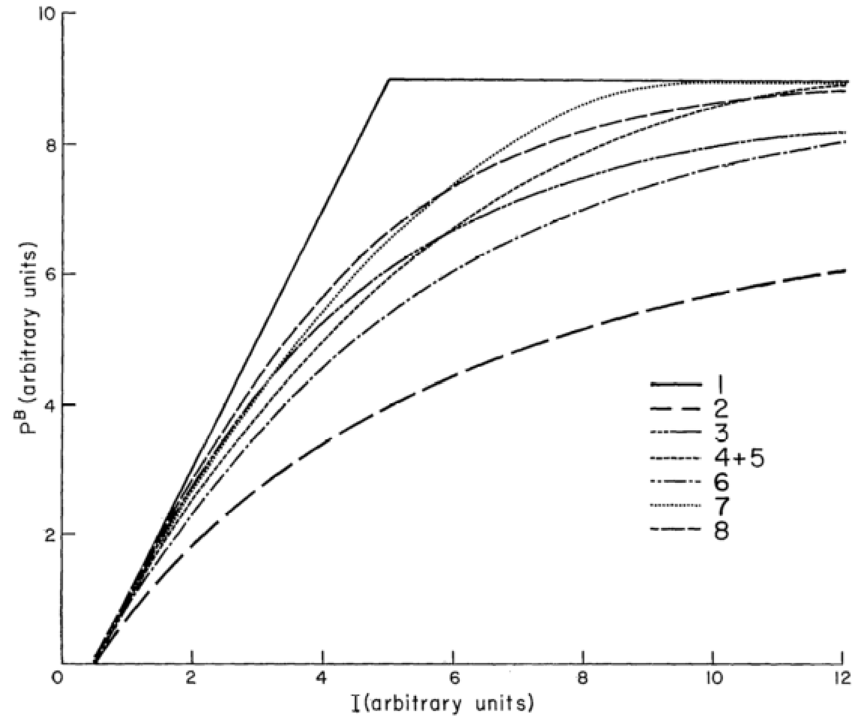

Fig. 1. The eight equations displayed for comparison. The same parameter values are used in each case: $P_m{}^B = 10$; $\alpha = 2$; $R^B = 1$.

FIGURE 2.4: From [Jassby and Platt, 1976], eight equations form eight P/I curves.

- I is light intensity (Watts per square meter)
- α is the initial slope of the P/I curve ((milligrams of Carbon / milligrams of chlorophyll / hour)/((Watts per square meter))
- P_B is the calculated rate of primary productivity (mg C/ mg Chl /hour, like P_m^B)

Using Equation (2.3), it is possible to easily estimate primary production from only two relatively-easily-calculated parameters (α and P_B) describing the P/I curve, and a value for light intensity. This equation can also be adapted slightly to use different parameters, as follows:

$$P = P_{max} * tanh(\alpha * I/P_{max}) \tag{2.4}$$

Where...

- P_{max} is the highest rate of primary productivity in terms of volume (mg C / cubic meter / hour)
- I is the light intensity (micromoles of photons per square meter per second)
- α is the initial slope of the line, [(mg C / cubic meter / hour)/(micromoles of photons per square meter per second)]
- P is the calculated rate of primary production (mg C / cubic meter / hour)

Note that sometimes instead of α, an alternate parameter is used: *light intensity at onset of saturation* (I_k), or the light value at which the P/I curve levels off. I_k is simply another mathematical descriptor of a P/I curve, and can be calculated from P_{max} and α as shown in Equation (2.5).

$$I_k = \frac{P_{max}}{\alpha} \tag{2.5}$$

Given any two of I_k, P_{max} and α, Equation (2.5) can be solved to find the third. Therefore, any two of the three is sufficient to fully describe a P/I curve. In our project, I_k and P_{max} are used.

A further refinement to this methodology is to account for *photoinhibition*. The models thus far describe primary production as generally increasing with light until a maximum point P_{max} is reached, at which point it levels off, as in figure 2.4. However, in reality, there is a level of light intensity at which there is *too much* light, and photosynthesis is *inhibited*.

In Platt et al. [1980], to properly describe a P/I curve that takes this factor into account, another parameter is added: I_b. I_b is the light intensity where photoinhibition occurs. Similarly to I_k, it is common to use an alternate parameter known as β, which is related to I_b using the Equation (2.6):

$$I_b = \frac{P_{max}}{\beta} \tag{2.6}$$

However, even with all this research, there is much to be done. Estimates of photosynthesis parameters such as α and P_{max} can vary between models, affecting estimates of primary production dramatically. Frenette and Demers [1993] tested various methods for estimating these parameters, finding variations up to *133%* between different models, citing various sources of possible experimental error, and the problem of subjectivity in parameter estimation.

Based on all the preceding research, here is the default form of the P/I curve equation as used in this project for pelagic primary production:

$$P = P_{max} * (1 - exp(\frac{-\alpha * I}{P_{max}})) * exp(\frac{-\beta * I}{P_{max}}) \tag{2.7}$$

This specific form of the equation is adapted from [Rueter, 2008], an online source, which adapts Platt's equation for use in computer code. It includes a photoinhibition parameter.

However, users may choose to override this default equation by setting β to zero. In this case, the program will use the following equation instead:

$$P = P_{max} * tanh(\alpha * I/P_{max}) \tag{2.8}$$

2.3 Carpenter and estimating the shape of the lake basin

Besides light and characteristics of the production curve, another important factor in lake PPR calculations is the *morphometery*, or shape of the basin in which the water sits. Both Fee [1969] and Carpenter [1983] addressed the issue of lake shape. They use the *Depth Ratio*, the ratio of a lake's average depth to its maximum depth, as a simple metric for describing the shape of the lake basin. Figure 2.5 shows some of the different lake basin shapes possible.

LAKE GEOMETRY AND PRODUCTIVITY 275

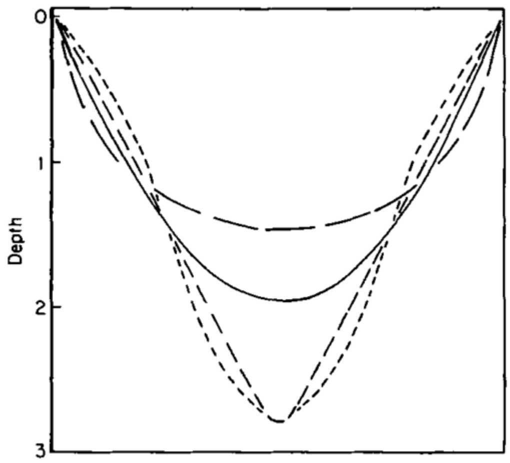

FIG. 1. Cross sections through the deepest points of models of lakes with identical surface areas, mean depths, and volumes. Vertical scale is in multiples of mean depth. ——, paraboloid (depth ratio = 0·5); — — —, ellipsoid (depth ratio = 0·66), - - -, hyperboloid (depth ratio = 0·35); and -----, sinusoid (depth ratio = 0·35).

FIGURE 2.5: Cross sections of lakes, with depth ratios [Carpenter, 1983].

2.4 Vadeboncoeuer and the importance of the benthic region

The Vadeboncoeur model, which addresses benthic primary production, is the basis for this project. Where previous research focused on primary production in the water itself, discounting primary production in the benthic (lake bottom) region as insignificant, Vadeboncoeur et al. [2001] investigated the complex relationship between benthic and planktonic primary production, with algae in upper parts of the water column shading and reducing photosynthesis in lower parts. Vadeboncoeur and Jeppesen [2003] then showed empirically that benthic primary production can be a very significant portion of a lake's primary production - the percent contribution from BPP ranging from only 5%, to as high as 80% in some cases, depending on morphometry, and composition of the littoral zone habitat.

Vadeboncoeur et al. [2008] took this knowledge and used it to create a general model for calculating benthic primary production across various lake sizes, taking into account morphometry (lake shape), nutrients, and light. The equations of the Vadeboncoeur 2008 model are included in Figure 2.6, while the parameters are in Figure 2.7.

TABLE 1. Equations for the model.

Equation number	Model output	Equation
1	lake surface area, A, at depth z	$A_z = A_0[1 - (z/z_{\max})]^{\gamma}$
2	lake volume, V, above depth z	$V_z = \gamma z/(\gamma + 1)$
3	phytoplankton chlorophyll a	$\mathrm{Chl} = 0.41\mathrm{TP}^{0.87}$
4	phytoplankton productivity, PP (mg $\mathrm{C \cdot m^{-3} \cdot h^{-1}}$)	$\mathrm{PP_{max}} = 2.2\mathrm{Chl}$
5	thermocline depth	$Z_{\mathrm{therm}} = 6.95A_0^{0.185}$
6	light-attenuation coefficient ($\mathrm{m^{-1}}$)	$K_{\mathrm{d}} = K_{\mathrm{b}} + 0.015[\mathrm{Chl}]$
7	light at depth z, time t ($\mu\mathrm{mol \cdot m^{-2} \cdot s^{-1}}$)	$I_{zt} = I_{0t}e^{-K_{\mathrm{d}}z}$
8	surface light at time t ($\mu\mathrm{mol \cdot m^{-2} \cdot s^{-1}}$)	$I_{0t} = I_{0\max}\sin\left(\pi\frac{t}{\mathrm{daylen}}\right)$
9	daily phytoplankton primary production at depth z (mg C)	$\mathrm{PP}_z = \Delta t\sum_{\mathrm{sunrise}}^{\mathrm{sunset}}\mathrm{PP_{max}}\tanh(I_{zt}/I_k)(V_z - V_{z-\Delta z})$
10	daily whole-lake phytoplankton production, TPP (mg $\mathrm{C/m^2}$)	$\mathrm{TPP} = \frac{\sum_{z=0}^{z_{\mathrm{epi}}}\mathrm{PP}_z}{A_0}$
11	daily benthic primary production, BP, at depth z (mg C)	$\mathrm{BP}_z = \Delta t\sum_{\mathrm{sunrise}}^{\mathrm{sunset}}\mathrm{BP_{max}}\tanh(I_{zt}/I_k)(A_{z-\Delta z} - A_z)$
12	daily whole-lake periphyton production, TBP (mg $\mathrm{C/m^2}$)	$\mathrm{TBP} = \frac{\sum_{z=0}^{z_{\mathrm{epi}}}\mathrm{BP}_z}{A_0}$

Note: See Table 2 for symbol definitions.

† Sources of equations: Eqs. 1–2, this paper; Eq. 3, Prairie et al. (1989); eq. 4, Guildford et al. (1994); eq. 5, Hanna (1990); Eq. 6, Krause-Jensen and Sand-Jensen (1998); Eqs. 7–10 McBride (1992); Eqs. 11–12, Vadeboncoeur et al. (2001).

FIGURE 2.6: Vadeboncoeur model equations [Vadeboncoeur et al., 2008].

TABLE 2. Definition of model parameters together with units and with input values.

Parameter	Definition	Input values
DR	depth ratio†	0.3, 0.5, 0.7
$\bar{z}$	mean depth (m)	2, 5, 10, 25, 100
$z_{\max}$	maximum depth (m)	$\bar{z}$/DR
Δz	depth interval (m)	0.1
γ	shape factor	DR/(1 − DR)
TP	total phosphorus ($\mathrm{mg/m^3}$)	3, 10, 50, 100, 500, 1000
K_{b}	non-chlorophyll (i.e., background) light attenuation	0.05, 0.2, 0.8, 2.0
t	hours after sunrise	0.25,0.5, 0.75, . . . , 15
daylen	hours of daylight (day length)	15
$I_{0\max}$	surface light at solar noon ($\mu\mathrm{mol \cdot m^{-2} \cdot s^{-1}}$)	1500
Δt	time increment (h)	0.25
$\mathrm{BP_{max}}$	maximum benthic primary production (mg $\mathrm{C \cdot m^{-2} \cdot h^{-1}}$)	5, 50, 120, $28.1\mathrm{TP}^{0.24}$
I_k	light intensity at onset of saturation ($\mu\mathrm{mol \cdot m^{-2} \cdot s^{-1}}$)	$180_{\mathrm{phytoplankton}}$, $300_{\mathrm{periphyton}}$

† $\mathrm{DR} = \bar{z}/z_{\max}$.

FIGURE 2.7: Vadeboncoeur model parameters [Vadeboncoeur et al., 2008].

2.5 Devlin and validation of the Vadeboncoeur Model: actual bathymetry is ideal

Since the Vadeboncoeur model was published, it has undergone testing and validation [Devlin, 2015]. Devlin's work, currently in preparation for submission to *Limnology and Oceanography Methods*, shows that the Vadeboncoeur model's method for calculating lake shape introduced significant error, recommending instead either the methods of Carpenter [1983], or ideally, actual bathymetry (i.e. water surface area at each depth).

In accordance with the assessment by Devlin et al., this project makes use of actual bathymetric figures to determine the morphometry of the lake.

Benthic primary production

As discussed in Chapter 2, the calculation of primary production for an entire lake or water body requires the calculation of primary production in both the bottom sediments (benthic primary production, or BPPR) and in the phytoplankton living in the water column ("pelagic" or "phytoplanktonic" primary production, PPPR).

3.1 Calculating benthic primary production

The algorithm for average daily *benthic* primary production used in the project follows. Source code implementing the algorithm can be found in Appendix A.

1. Pick a depth interval size for calculation. This is the "resolution" at which calculations will occur. We use 0.1 meters as the default, as in the Vadeboncoeur model [Vadeboncoeur et al., 2008]. Devlin's tests also support 0.1 meters as a reasonable "small" depth increment [Devlin, 2015].

2. Pick a time interval size for calculations. We use quarter-hours, again based on the Vadeboncoeur model.

3. Calculate the area of lake-bottom that receives enough light for photosynthesis to occur. This is the Total Littoral Area (or sometimes, the "Littoral Zone") (Equation (3.5)).

4. For each depth interval, starting at the surface of the lake and proceeding down to the depth of 1% light, in increments equivalent to our chosen depth interval size:

(a) Calculate the littoral area that is in this depth interval (Equation (3.11)).

(b) Divide this area by the Total Littoral Area to get the fraction of the littoral zone in this interval (Equation (3.11)). This fraction is used later to weight the result.

(c) For each time of day starting at first light and ending at sundown, proceeding in time increments equal to the chosen time interval size:

    i. Calculate the primary production rate $BPPR_{zt}$ at that time, for the current depth.

(d) Sum the primary production rates for each time to get a rate for this depth, $BPPR_z$ (Equation (3.7)).

(e) Multiply the primary production rate for this depth by the fractional area to get the weighted benthic primary production rate for this depth (Equation (3.8)).

5. Sum the weighted benthic primary production rates for each depth, to calculate a weighted average for benthic primary production for the whole lake (Equation (3.1)), for that day.

In summary, we use a P/I curve equation and parameters for light and photosynthesis to calculate primary production for every depth and every time over the course of the day. These values are summed and weighted by the proportion of the lake sediment occurring at each depth, giving us the average benthic primary production, per meter squared of littoral area, per day.

The list of equations used in calculating BPPR is presented in Table 3.1.

3.1.1 Daily BPPR equations

In this section, we describe in more detail the equations used in the model for calculating benthic primary production.

TABLE 3.1: List of equations for benthic primary production calculations

Equation number	Purpose of Equation
Equation (3.1)	Calculate daily BPPR
Equation (3.2)	Proportion of surface light at depth
Equation (3.3)	Depth of specific light proportion
Equation (3.4)	Depth of light proportion 0.01 (1% light)
Equation (3.5)	Total littoral area
Equation (3.6)	LA_z: Littoral area at depth z
Equation (3.7)	$BPPR_z$: Daily Average BPPR at depth z
Equation (3.8)	$BPPR_{zt}$: BPPR at depth z, time t
Equation (3.9)	Light at depth z, time t
Equation (3.10)	Light at surface at time t
Equation (3.11)	Fractional littoral area at depth z

The equation described above in Section 3.1 to calculate benthic primary production per day, per m^2 of littoral area:

$$BPPR = \frac{\Delta z \sum_{z=0}^{z1\%} BPPR_z}{A_{littoral}} \tag{3.1}$$

[Vadeboncoeur et al. [2008], Equation 12.]

Parameters:

- $BPPR_z$ = daily average benthic primary production at depth z, in milligrams of carbon per square meter per day ($mgC * m^{-2} * d^{-1}$) (Equation (3.7)).
- z = depth in meters.
- Δz = the chosen depth interval, in meters.

- $A_{littoral}$ is the *total littoral area* of the lake, in square meters (m^2) (Equation (3.5)).
- $z_{1\%}$ is the *depth of 1% light*, the depth at which light levels are reduced to 1% of surface light levels, in meters (Equation(3.4)). Below this depth, photosynthesis rates are assumed to drop to zero.

Note: If BPPR per m^2 of *surface area* is needed, A_0, the total surface area of the lake can be substituted in place of $A_{littoral}$

3.1.2 Proportion of surface light at depth z

To use the summation in Equation (3.1), it is necessary to calculate the *depth of 1% light*, the depth at which light levels are reduced to 1% of surface light levels. As photosynthesis levels below this depth are negligible, this forms the upper bound for the summation.

The equation for the proportion of surface light at a specific depth [Baird, 2001] is:

$$I_z/I_0 = e^{-k_d * z} \tag{3.2}$$

Where:

- I_z is the light level at depth z, arbitrary units. In μmoles per square meter per second (μmol*m^{-2}*s^{-1}).
- I_0 is the light level at the surface. Same units as I_z.
- e: the base of the natural logarithm.
- k_d is the *light attenuation coefficient*, one of the model parameters, in inverse meters (m^{-1}).

- z depth, in meters.

3.1.3 Depth of specific light proportion

Solving Equation (3.2), we can calculate the depth to which a specific proportion of surface penetrates $z_{proportion}$ Equation (3.3). This brings us closer to finding the depth of 1% light, allowing us to calculate the depth of *any* light proportion.

$$z_{proportion} = \frac{ln(proportion)}{-k_d} \tag{3.3}$$

Parameters:

- $proportion$ is the proportion of surface light desired.
- $z_{proportion}$ is the depth at which light levels are $proportion$ of surface light. That is, (light levels at surface)/(light levels at $z_{proportion}$) $= proportion$.
- k_d is the *light attenuation coefficient*, one of the model parameters, in inverse meters (m^{-1}).

3.1.4 Depth of one percent light.

Now that we have an equation for calculating the depth to which a specific proportion of light will penetrate, we can find the depth of 1% light, below which very little photosynthesis occurs. Substituting in a proportion of 0.01 (i.e. one percent) to Equation (3.3), we find that the depth of 1% light is approximately:

$$z_{1\%} = \frac{ln(0.01)}{-k_d} = \frac{4.6}{-k_d} \tag{3.4}$$

(source: based on Equation (3.3))

Parameters:

- $z_{1\%}$ is the depth to at which light levels are 1% of surface light levels.
- 4.6 is the natural log of 0.01.
- k_d is the *light attenuation coefficient*, one of the model parameters, in inverse meters (m^{-1}).

3.1.5 Total littoral area

The littoral area is that part of the benthic zone which has sufficient light for photosynthesis. Using $z_{1\%}$, the depth of one percent light, as the lower bound, we calculate the the littoral area by summing the littoral areas at each depth increment.

$$LA_{total} = \Delta z \sum_{z=0}^{z1\%} LA_z \tag{3.5}$$

Parameters:

- LA_{total} is the total littoral area of the lake, in square meters (m^2).
- Δz is the chosen depth interval for calculations, in meters.
- $z_{1\%}$ is the *depth of 1% light*, the depth at which light levels are reduced to 1% of surface light levels, in meters (Equation(3.4)).
- LA_z is the littoral area at depth z (Equation (3.6).

3.1.6 Littoral area at depth z

Since benthic areas at specific depths are often difficult to measure directly, they are rarely included in data sets.

Instead, to approximate the littoral area at a specific depth z, we simply subtract the area of the lake at z from the area at depth z-Δz (figure 3.1). While this does not give an exact result, most lakes are shallow enough that this is a close approximation.

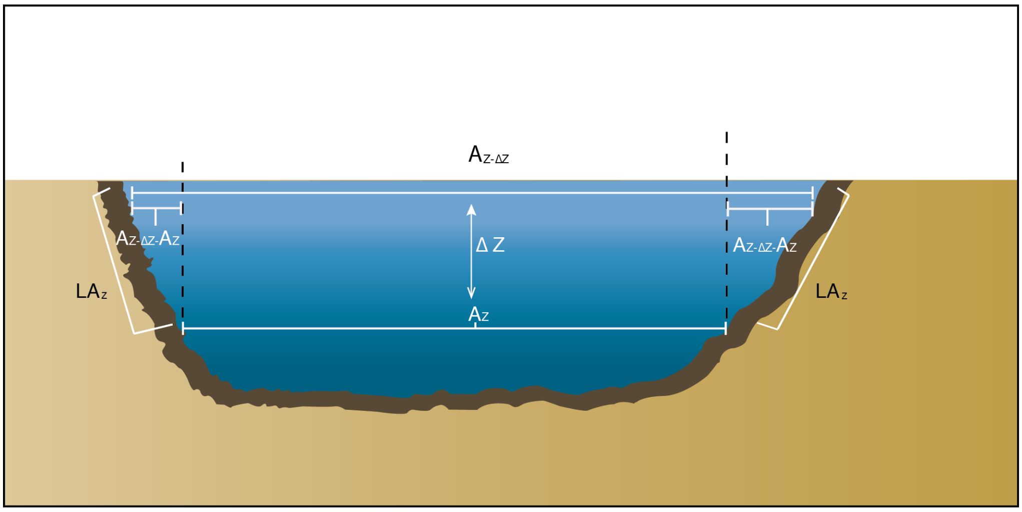

FIGURE 3.1: Calculating littoral area by subtracting area at upper and lower bounds

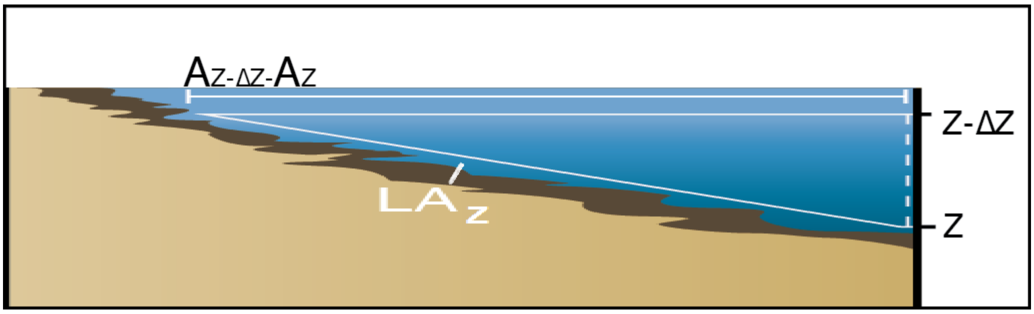

FIGURE 3.2: The difference in areas at z and z-Δz is a close approximation of LA_z

To approximate LA_z, we therefore use the following equation:

$$LA_z = A_{z-\Delta z} - A_z \tag{3.6}$$

[Vadeboncoeur et al. [2008]]

Parameters:

- LA_z is the littoral area at depth z.
- Δz is the depth interval for calculations, in meters.
- A_z is the water area at depth z
- $A_{z-\Delta z}$ is the water area at depth $z - \Delta z$

3.1.7 Daily average BPPR at depth z

To calculate the daily average BPPR at a certain depth z, we first sum up the BPPR at depth z for every time t from sunrise to sunset, using the time increments provided in the model parameters.

In some cases, the chosen time interval is not equal to one hour, yet the equations produce an hourly rate. This can lead to problems. For example, if a quarter-hour time interval is chosen, this will result in hourly rates being calculated four times per hour, essentially quadrupling the sum.

To account for fractional-hour intervals, we then divide by $(1/\Delta t)$. For example, if given a time interval of half, this method would give us two hourly rates for every hour. Summing these values gives a value for $BPPR_z$ 2 times too large. By dividing by $1/0.5$ (equivalent to dividing by 2), we account for this problem [[Vadeboncoeur et al., 2008]].

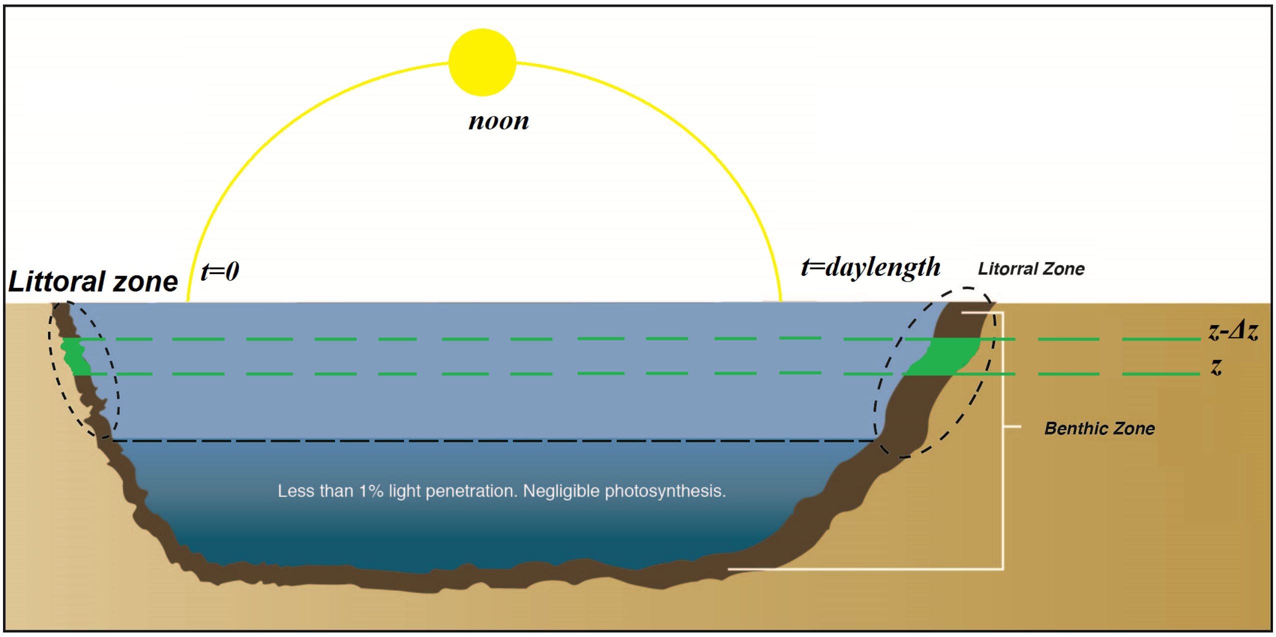

FIGURE 3.3: Benthic primary production for an interval (green) is calculated for every time interval (BPPRZT). The summation of these calculations gives BPPR in the interval, BPPRZ

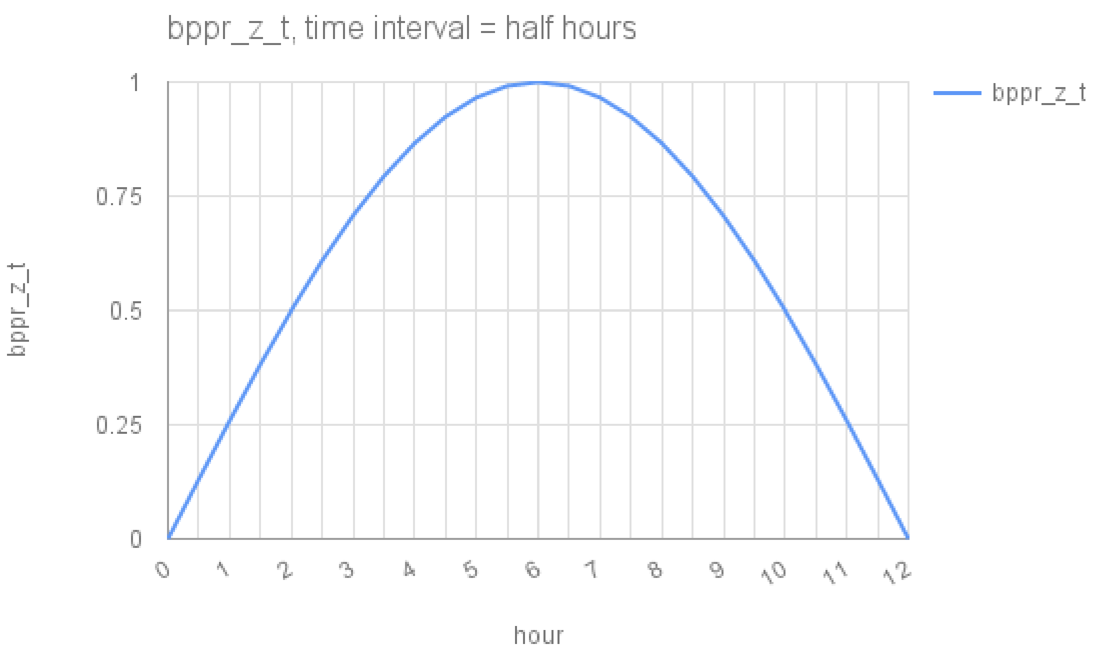

FIGURE 3.4: Primary production at a specific depth over the course of an entire day. Note: calculations are done at half-hour intervals, but each calculation gives an hourly rate. Sum = 15.25705169

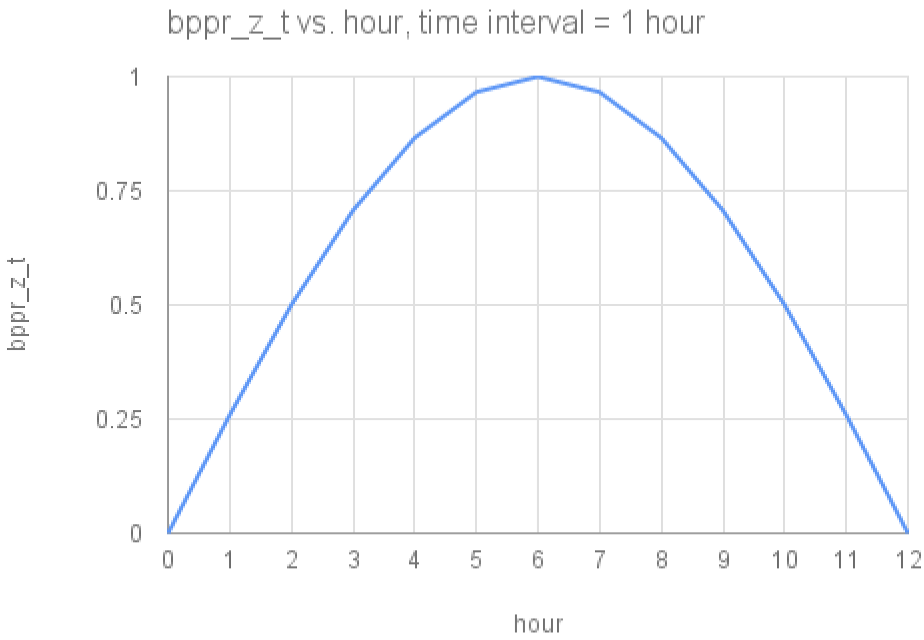

FIGURE 3.5: Primary production at a specific depth over the course of an entire day. Note: calculations are done at one-hour intervals, each calculation gives an hourly rate. Sum = 7.595754113

$$BPPR_z = \frac{(\Delta t \sum_{sunrise}^{sunset} BPPR_{zt})}{1/\Delta t} * LA_{fractional\_z} \tag{3.7}$$

Parameters:

- $BPPR_z$ is the daily average benthic primary production at depth z.
- Δt is the time increment used in calculations, one of the model parameters. We used quarter-hours as our default.
- $BPPR_{zt}$ is the BPPR at a specific depth z and time t. (Equation (3.8)).
- $LA_{fractional\_z}$ is littoral area at depth z, as a fraction of total or littoral area, in square meters (m^2) (Equation (3.11)).

3.1.8 BPPR at depth z, time t

To calculate BPPR at a specific depth z, time t:

$$BPPR_{zt} = BP_{max\_z} * \tanh \frac{I_{zt}}{BI_{kz}} \tag{3.8}$$

Parameters:

- BP_{max_z} is the P/I curve parameter P_{max} for benthic primary production for depth z (mg C * m^{-2}*$h^{-}1$).
- BI_{kz} is the P/I curve parameter I_k for benthic primary production for depth z.
- I_{zt} is the light value at depth z, time t. (Equation (3.9)).

3.1.9 Light at depth z, time t

Light at a specific depth z and time t is calculated using [Vadeboncoeur et al. [2008], Equation 7)]

$$I_{zt} = I_{0t} * e^{-K_d * z} \tag{3.9}$$

Parameters:

- I_{zt} is the light value at depth z, time t, in μ mol *m^{-2}*s^{-1}.
- I_{0t} is the light value at depth 0 (the lake surface), time t in μ mol *m^{-2}*s^{-1} (Equation (3.10)).

3.1.10 Light at surface at time t

$$I_{0t} = I_{0max} * \sin(\pi * \frac{t}{daylen}) \tag{3.10}$$

[Vadeboncoeur et al. [2008], Equation 8]

- I_{0t} is the light value at depth 0 (the lake surface), time t, in μ mol $*m^{-2}*s^{-1}$.
- I_{0max} is the model parameter *surface light at solar noon*, in μ mol $*m^{-2}*s^{-1}$.
- π is the ratio of a circle's circumference to its diameter.
- *daylen* is the model parameter *day length*, in hours.

3.1.11 Fractional littoral area at depth z

$$LA_{fractional\_z} = \frac{LA_z}{LA_{total}} \tag{3.11}$$

- LA_{total} is the total littoral area of the lake, in square meters (m^2). (Equation (3.5)).
- LA_z is the littoral area at depth z (Equation (3.6)).

3.2 Testing and results

The model described in the previous section was implemented in Python (Appendix A) Validation shows BPPR values within ten percent of the results of the original Vadeboncoeur model (Appendix B).

To test the efficacy of interpolating from fewer data points, test data is constructed by deletion of rows, leaving data inputs for photosynthesis parameters benthic P_{max} and I_k at only the desired light penetration levels.

TABLE 3.2: BPPR test results using data inputs for P_{max} and at 10-cm intervals. Bppr values from Vadeboncoeur model courtesy of Dr. Shawn Devlin [Devlin, 2015]

lake_ID	DOY	Bppr (Vadeboncoeur model.)	Bppr (test results)	%difference
US_SPARK	160	254.6715329	270.11949	6.07
US_CRYST	164	174.271404	156.0813414	-10.44
US_TROUT	164	188.1237123	195.134938	3.73
US_BIGMU	171	286.8234852	290.4706821	1.27

First, we test the program using data inputs corresponding approximately to light-penetration levels approximately ten percent apart, i.e. values for P_max and I_k at the depth to which 100% of light penetrates, and the depth to which 90% penetrates, etc.

Testing gives BPPR results that closely match the Vadeboncoeur model (maximum difference 9.98%, average difference about 5.2%), almost the same as with the original data set.

TABLE 3.3: Test results with 11 data points, at light proportions: [1.0,0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1,0.01]

lake_ID	day of year	Bppr (devlin)	BPPR results	%difference
US_SPARK	160	254.6715329	267.3550481	4.98
US_CRYST	164	174.271404	156.8736542	-9.98
US_TROUT	164	188.1237123	195.9716558	4.17
US_BIGMU	171	286.8234852	291.4488895	1.61

Finally, we test the program using only 5 data points for each lake. Even at this level of detail, results match the Vadeboncoeur model closely (max difference 12%, average difference about 6.4%).

TABLE 3.4: Test results with 5 data points, at light proportions:[1.00,0.5,0.25,0.1,0.01]

lake_ID	day of year	Bppr (devlin)	Proportions:[1.00,0.5,0.25,0.1,0.01]	%difference
US_SPARK	160	254.6715329	279.8353579	9.88
US_CRYST	164	174.271404	154.2946539	-11.46
US_TROUT	164	188.1237123	193.803522	3.02
US_BIGMU	171	286.8234852	290.9204828	1.43

3.3 Conclusions

The program is able to replicate the results of the Vadeboncoeur model to within 12%. The program output matches the Vadeboncoeur model closely (max difference 12%, average difference about 6.4%) even when the number of data points is decreased from over 100, to only 5.

Given the high variance in primary production techniques, this means that this program is effectively as useful as the original Vadeboncoeur model.

Further testing is needed to determine the point or points where the model degrades beyond usefulness. It may be useful to test both against the Vadeboncoeur model and other models.

Phytoplanktonic primary production

4.1 Calculating phytoplanktonic primary production

As discussed in Chapter 2, the calculation of primary production for an entire lake or water body, requires the calculation of primary production in both the bottom sediments (benthic primary production, or BPPR), and in the phytoplankton living in the water column ("pelagic" or "phytoplanktonic" primary production, PPPR).

The algorithm for average daily *Phytoplanktonic* Primary Production used in the project follows. Source code implementing the algorithm can be found in Appendix A.

4.1.1 Model parameters

Model parameters specific to Phytoplankton Primary Production Rate (PPPR):

- P_{max} is the maximum rate of photosynthesis, in mg C per cubic meter per hour (mgC*m^-3*$hr - 1$)
- α is the the initial slope of the P/I curve (mgC*m^-3*$hr-1$)/(μmol photons*m^-2*s^-1)
- β is the the initial slope of the curve at onset of photoinhibition (mgC*m^-3*$hr-1$)/(μmol photons*m^-2*s^-1). Note that photoinhibition is not likely to be a factor in the lower layers of the lake.
- *thermal_z* is the depth of bottom edge of each thermal layer.

4.1.2 Thermal layers

Lakes can be thermally stratified, with up to three layers (figure 4.1):

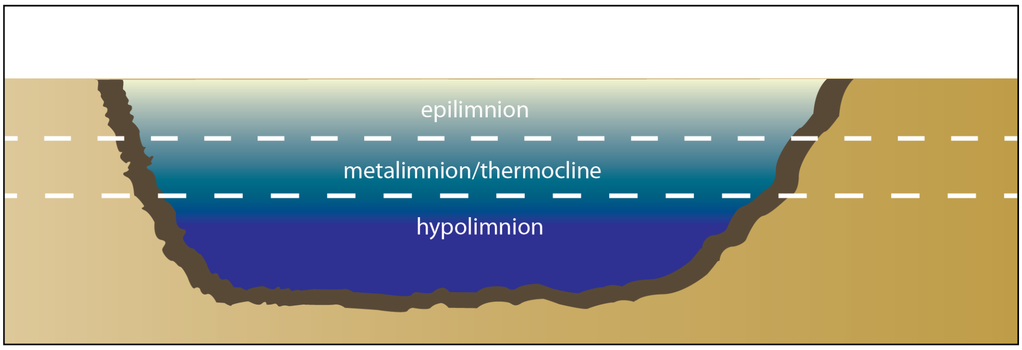

FIGURE 4.1: Thermal layers of a thermally-stratified lake.

The layers are:

1. The epilimnion, the top layer, with the warmest water and the most light.
2. The metalimnion, the middle layer.
3. The hypolimnion, the bottom layer, with the coldest water and the least light.

For the purposes of this project, photosynthesis parameters P_{max} and I_k are assumed to be constant throughout each layer, but differ from layer to layer.

Light attenuation coefficient k_d, which is used for calculating the light at depth z, time t (I_{zt}), is assumed to be constant throughout the entire lake.

4.1.3 PPPR at depth z, time t

Using the values for each model parameter in each thermal layer, we can calculate the phytoplankton primary production rate (PPPR), using the equation developed by Jassby and Platt [1976] (Equation (2.7)).

We use this equation to calculate PPPR at a specific time and depth ($PPPR_{zt}$) (Equation (4.1)).

$$PPPR_{zt} = P_{max\_layer} * (1 - exp(\frac{-\alpha_{layer} * I_{zt}}{P_{max\_layer}})) * exp(\frac{-\beta_{layer} * I_{zt}}{P_{max\_layer}}) \quad (4.1)$$

Parameters:

- PPRzt (mg/m^3/hr)
- $P_{max_layer}$is the photosynthesis parameter P_{max} for this thermal layer in (mgC/m^3/hr).
- α_{layer} is the photosynthesis parameter α for this thermal layer in (mgC/m^3/hr)/(μmol/m^2/s
- β_{layer} is the photoinhibition parameter β for this thermal layer in (mgC/m^3/hr)/(μmol/m^2/
- I_{zt} is the light at depth z, time t in (μmol/m^2/s^1).

4.1.4 Daily PPPR at depth z

Using Equation (4.1), we sum up the values of PPPR over the course of a day, giving us the primary production rate for the entire day, at a given depth, $PPPR_z$. To compensate for fractional time increments, we then divide by $1/\Delta t$. For example, if using a time increment of 0.25 hours, this equation would give us four values of $PPPR_{zt}$ every hour. In such a case, simply adding all the values up would give a $PPPR_z$ value for the that hour that would be 4 times too large. Dividing by $1/0.25 = 4$, we correct for this (Equation (4.2)).

$$PPPR_z = (\frac{\Delta t \sum_{sunrise}^{sunset} PPPR_{zt}}{1/\Delta t}) \tag{4.2}$$

We then multiply this by the volume in our given depth interval to calculate the total Phytoplanktonic primary production in this depth interval, PPP_z (Equation (4.3).

$$PPP_z = PPPR_z * V_z \tag{4.3}$$

Note: V_z, the volume at depth z, is calculated using Equation (4.6)

We sum these values for PPP_z across all depth intervals to calculate total Phytoplankton Primary Production for the entire day, for the entire lake, PPP in mg of C per day (Equation (4.4)).

$$PPP = \Delta z \sum_{0}^{z1\%} PPP_z \tag{4.4}$$

Finally, we divide this by the surface area to calculate the Phytoplanktonic Primary Production Rate, in mg C per square meter of surface area, per day (Equation (4.5)).

$$PPPR = \frac{(\Delta z \sum_0^{z1\%} PPPR_z)}{A_0} \tag{4.5}$$

Parameters:

- Δz depth increment, in hours.

- A_0 Surface area, in meters squared (m^2)

4.1.5 Volume at depth z

To estimate the volume in a specific depth interval, whose lower bound is at depth Z and upper bound bound is at $z - \Delta z$, we use Equation (4.6)

$$V_z = \frac{(\Delta z * A_{z-\Delta z}) + (\Delta z * A_z)}{2} \tag{4.6}$$

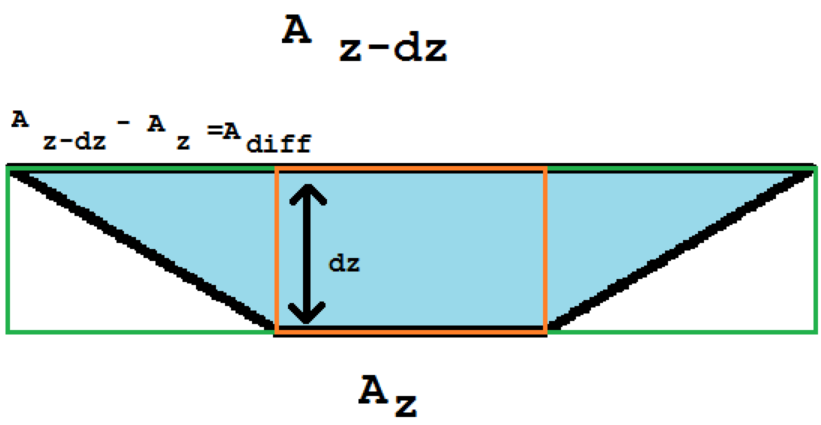

FIGURE 4.2: Volume calculation method. dz=Δzr

If we were to simply use $A_z * \Delta z$ to estimate V_z, (the orange rectangle in figure 4.2), we would underestimate the volume by a factor of $A_{diff} * \Delta z$. If we instead used $A_{z-\Delta z} * \Delta z$, we would *overestimate* by $A_{diff} * \Delta z$.

By taking the average of the two, we can estimate the actual volume.

4.2 Testing

Before making the model publicly available, we validate our implementation to ensure it has been properly coded, and is consistent with expected results.

4.2.1 Initial validation

Initial validation of the implementation is limited by data availability. Several datasets were available. Each of these provides some, but not all, of the required model parameters for PPPR. The datasets all concern the same set of lakes, but are from different dates, and contain different information. Furthermore, they lack corresponding correct PPPR values for these inputs, further limiting validation and testing.

The data set we use to test the implementation, therefore, is constructed by combining the available data sets. While none of the available data sets provides all required parameters, this approach allows the creation of test data where each parameter value is at least *plausible*.

For example, to construct one test data point:

- values for P_{max}, α, and β for each thermal layer are taken from a set of measurements from day 163, year 1996 (“Phyte_90s.csv”).
- the depths of each layer, on the other hand, are taken from a different set of measurements (“PILTER.csv”), dating from 2006.
- parameters for the shape of the lake, light attenuation coefficient, length of day, and surface light at solar noon, and other general model parameters not specifically relating to PPPR are taken from the dataset used for BPPR verification (“pprinputs_Colin.csv”).

Using these “plausible” data points, PPPR values are calculated for several lakes. These PPPR values are then validated against recorded PPPR values from the scientific foundation that manages research on these same lakes, the Northern Temperate Lakes Long Term Ecological Research project (NTL LTER) [NTL, 2008]. We find that our tests produce values for PPPR no more than an order of magnitude different than the recorded values from NTL research.

Furthermore, while lacking a complete set of input/output pairs for validation, we did verify that the implementation responds as expected to changes. We varied the parameters and found the implementation responded as expected. For example, increasing P_{max} resulted in an increase in the calculated value of PPPR, while decreasing it led to a corresponding decrease in the calculated PPPR value.

4.2.2 Subsequent validation using NTL LTER data

After this initial round of testing provided reasonable results, we tested the program more extensively using the North Temperate Lakes Long Term Ecological Research database administered by the Center for Limnology at the University of Wisconsin-Madison [NTL, 2008].

4.2.2.1 Test data assembly and collation

Data was collated together from multiple sources.

- Source 1: Calculations by Dr. Vadeboncoeur. Photosynthesis values P_{max}, α, and β were calculated by Dr. Vadeboncoeur, by a process of curve fitting, using data from the NTL LTER database (PI2000s.csv, and Phyte_90s.csv, Appendix B).
  - P_{max}
  - α.
  - β.

- Source 2: Database entitled "North Temperate Lakes LTER: Light Extinction - Trout Lake Area 1981 - current", found here. Henceforth referred to as "NTL Light Extinction database".
  - Light attenuation coefficient k_d.
- Source 3: Database entitled "North Temperate Lakes LTER: Primary Production - Trout Lake Area 1986 - 2007" (henceforth the "NTL PPR database"), found here. Henceforth referred to as the "NTL PPR database".
  - Noon surface light levels, I_0. When there was no corresponding light extinction coefficient listed for days in Vadeboncoeur data, light extinction coefficients within 7 days were used.
  - Length of day, *daylen*. Calculated by inspection of primary production levels in the database, and light levels.
  - Thermal layer lower bounds. This was not included, directly, in the NTL PPR database. However, inspection of the data revealed that it included PPPR for each thermal layer in both "per meter squared" and "per meter cubed" units. We deduced that this unit conversion had been made by multiplying one value for ppr by the depth of the thermal layer in question. This insight allowed us to back-calculate the distance from the top to the bottom of each thermal layer. From this, we calculated the lower bound of each layer by addition of each distance to the lower bound of the layer above.
- Source 4: Database entitled "North Temperate Lakes LTER Morphometry and Hypsometry data for core study lakes", found here.
  - Area at z, A_z.

We combined data from all these sources. After discarding obviously-erroneous data points (e.g. pmax values 5+ orders of magnitude greater than normal, or with values equal to the default values of the curve-fitting parameter estimation program), we were left with measurements from 5 different days:

- Crystal lake, day 130, 2007
- Trout lake, day 233, 2006
- Sparkling lake, day 151, 1995
- Sparkling lake, day 178, 1995
- Sparkling lake, day 165, 1995

4.2.2.2 Validation against NTL LTER data

Both Equation 2.7 (PPPR with photoinhibition) and Equation 2.8 (PPPR without photoinhibition) were tested.

The NTL PPR database does not include whole-day, whole-lake phytoplanktonic primary production results. Instead, PPPR is listed by thermal layer, for each half-hour period. To validate against this, we plotted program results against NTL PPR values.

4.3 Discussion

Initial testing showed calculated values for PPPR using this implementation to be within an order of magnitude of known correct results from the NTL PPR database, and responded as expected to changes in input.

The second round of testing of our implementations of the photoinhibition equation (Equation (2.7)) and non-photoinhibition equation (Equation (2.8)) showed results that tracked qualitatively with NTL PPR database results (having curves with

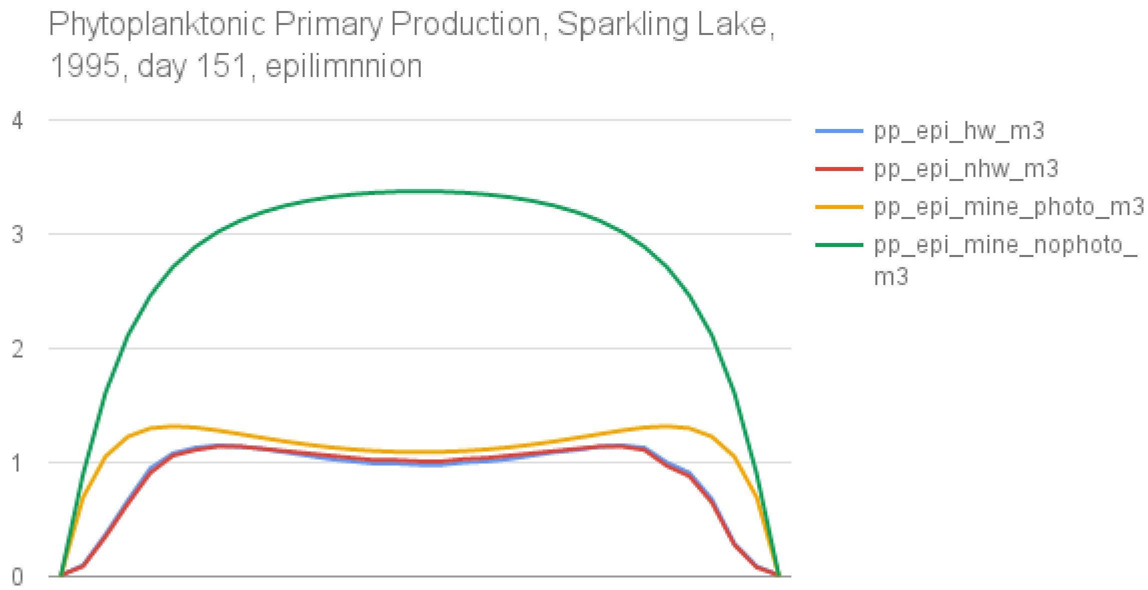

FIGURE 4.3: Example PPPR plot. Each line is phytoplanktonic primary production in $mgC * m^{-3} * hr^{-1}$ over the course of an entire day (sunrise to sunser). In this instance, the results of the photoinhibition equation (orange) seem much closer to the NTL PPR results (red and blue) than the non-photoinhibition equation (green).

similar shapes). However, neither equation was found to be consistently closer to the NTL PPR results. As a result, we have added the option for users to choose which equation they wish to use. If a value of of zero is is given for *beta* thermal layer in the input data file, the program will automatically use the non-photoinhibition equation for calculating phytoplanktonic primary production in that layer.

We were unable to confirm the exact original parameters or equations used by the NTL PPR database for their own estimates of PPR. Parameters cannot be simply estimated, as there are significant discrepancies in common parameter estimation techniques Frenette and Demers [1993]. Therefore, it is not possible to further validate our implementation of the Vadeboncoeur model until a set of parameters matched to known correct outputs *for that model* can be created, or found.

Making the Vadeboncoeur Model publicly accessible

In order to make the Vadeboncoeur model available to scientists we implement an online, web-accessible version. The programming language chosen is Python 2.7, due to ease of development and access to pre-existing scientific libraries such as SciPy.

5.1 Implementing the Vadeboncoeur Model in Python

Python is used to implement the equations detailed in Chapter 3, and Chapter 4.

5.1.1 Objects and program structure

The program is structured using an object-oriented approach. The objects are:

- Pond: contains information and methods necessary to calculate both benthic and phytoplanktonic primary production for a lake, for a single day. This includes lists of PhotosynthesisMeasurement objects for both benthic and phytoplanktonic primary production parameters, as well as a PondShape object for lake shape data.
- PondShape: Generic class. Objects of this class contain information and methods relating to the shape of a lake.

  - SimpleMetricsPondShape: subclass of PondShape, based on the usage of simple shape metrics such as the *depth ratio* (Maximum depth of the lake / mean depth of the lake). Not used in final version of the project, as testing indicates that this simplified morphometric model leads to "substantial over-or-underestimates" of benthic primary production. Removed in final version.
  - BathymetricPondShape: Subclass of PondShape, based on detailed bathymetry instead of simplified metrics. Stores data as a dictionary of depth/area pairs. If area data is needed at a depth not in the dictionary, the SciPy.interpolate library is used to estimate instead.

- PhotosynthesisMeasurement: Generic class. Objects of this class contain P/I curve (photosynthesis) parameters for a specific part of a lake, at a specific time.
  - BenthicPhotosynthesisMeasurement: Subclass of PhotosynthesisMeasurement, contains P/I curve parameters specifically related to benthic primary production.
  - PhytoPlanktonPhotosynthesisMeasurement: Subclass of PhotosynthesisMeasurement, contains P/I curve parameters specifically related to phytoplanktonic primary production.

- DataReader: This class uses the xlrd library to read in lake data from .csv, .xls, or .xlsx files, storing it in Pond objects.

- FlaskApp: Runs the web app itself, interfacing with DataReader and HTML templates to create the website.

For further details, see the source code in Appendix A, or on GitHub at the Project GitHub Repository

5.1.2 Interpolation with SciPy

This project investigates to what extent interpolation algorithms can compensate for a decreased number of data points. Interpolation is done using the SciPy interpolate package.

In the original Vadeboncoeur model, PPR estimation was done with 100-200 P/I curve parameter values (one every 0.1 m, from the top of the lake to the bottom of the photic zone). Given P/I curve parameter values at depths with light levels equal to 100%, 50%, 25%, 10%, and 1% of surface light (5 in total), we tested both spline and linear interpolation.

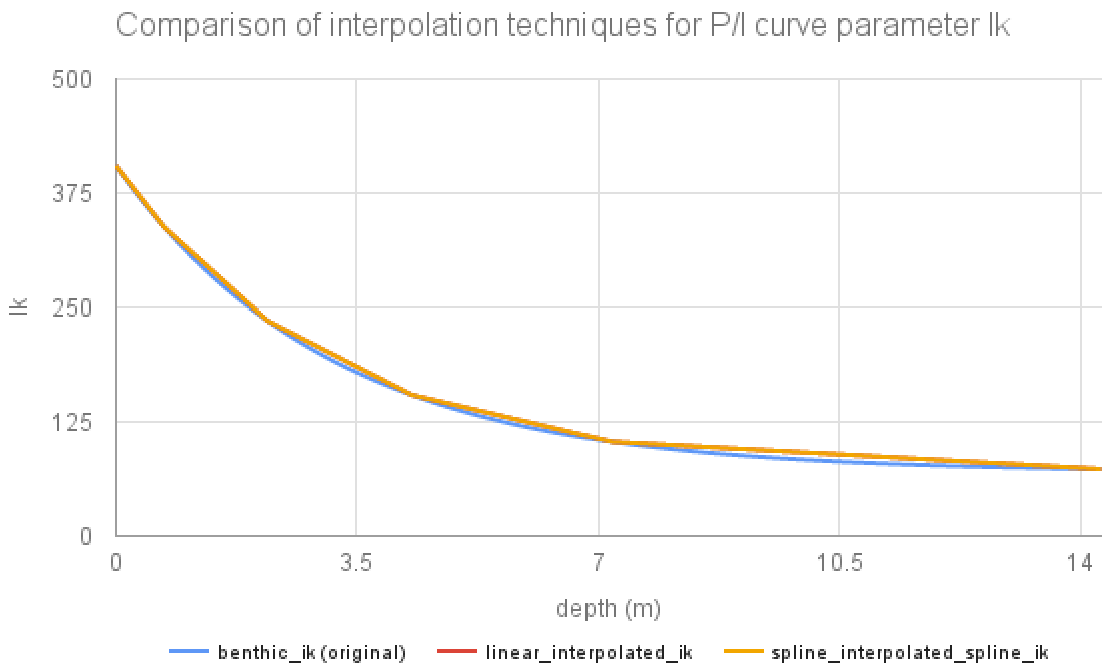

FIGURE 5.1: In this example, values calculated using spline (orange) and linear interpolation (red) correlate closely to original Vadeboncoeur model values (blue) for P/I curve parameter I_k. Interpolation is done from data points with light levels proportional to 1.0, 0.8, 0.5, 0.25, 0.1, and 0.01 of surface light (5 data points)

The maximum measured difference between interpolated and original values using spline interpolation is equal to 16% of the original value. With linear interpolation

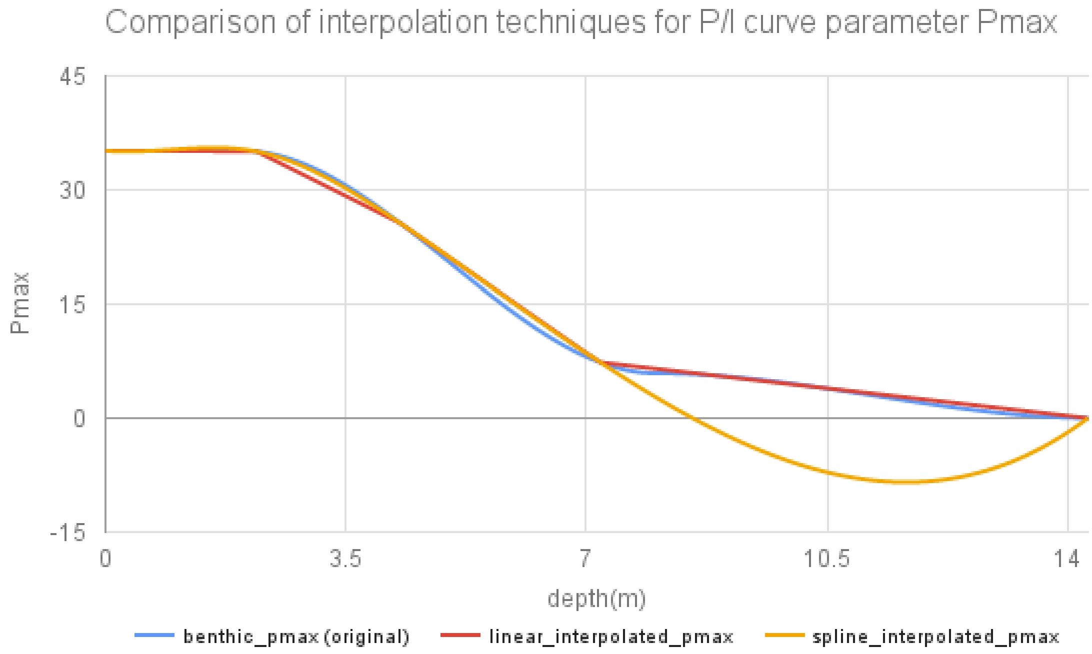

FIGURE 5.2: In this example, values calculated using linear interpolation (red) correlate closely to original Vadeboncoeur model values for P/I curve parameter P_{max}. Spline interpolation (orange), on the other hand, deviates at approximately the depth where light/surface light proportion equals 0.25. Interpolation is done from data points with light levels proportional to 1.0, 0.8, 0.5, 0.25, 0.1, and 0.01 of surface light (5 data points)

this drops further, with the maximum difference being only 4%. Both of these results were achieved with only 5 data points. We conclude that linear interpolation with 5 data points is sufficient for our primary production estimation.

While further reduction in the number of data points may be possible, further testing is required to determine the minimum number of data points at which interpolation techniques are no longer useful for estimating P/I curve parameters.

5.1.3 Reading in lake data With xlrd

User input of Pond data for primary production is done by uploading of .csv or .xls files. In order to read this data, the xlrd and xlwt packages are used.

To use the program, users must fill out a template .xls file, putting data in specific columns. The DataReader class specifies the indexes of these as constant values. A possible future improvement would be a more dynamic or flexible method of dealing with this issue, such as allowing user-specification of data/column indexes in an online form, then passing this information to the DataReader class.

See Appendix C for a specification of the format for user data.

5.1.4 Open-source development using GitHub

In order to make the project more accessible to the public, and easier to maintain and develop in the future, GitHub is used for source code management.

The project's source code can be found at the Project GitHub Repository. All code is open to the public, under the MIT license.

5.2 Building the web application

To solve the problem of making the Vadeboncoeur model usable online, we use the Flask Web Framework.

5.2.1 Using the Flask Microframework

The Flask web framework provides a simple way to interface between server-side scripts (such as the DataReader class), and the web. It makes heavy use of Jinja2 templates, as well as "views": pieces of python code that run when certain URLs are accessed.

For example, in this project, users accessing \index page trigger a flask function that renders an HTML template. This HTML template welcomes them to the site, provides both a filled and unfilled template for user data, and contains a form for users to choose and upload files.

5.2.1.1 Receiving uploaded files from users

According to the the Flask Guide to File Uploads, the "problem of file uploads" should be solved thusly:

1. An HTML form is marked with special tags, and the input type is set to a "file".
2. The application accesses the file from the files dictionary on the request object.
3. use the save() method of the file to save the file permanently somewhere on the filesystem.

Using this technique allowed the program to accept user-uploaded data, and pass it between different Flask views. Any view that required access to the data could simply access the file and use the DataReader class to parse it. However, there were some problems with this technique:

- Even with the use of Flask's secure_filename function, user-inputted file names could cause issues with security or reliability.
- Files quickly begin to clutter up the server's upload folder. Errors may result if, for example, two users uploaded their data using the same file name.

The following solutions were attempted:

- Rather than using user-submitted filenames, random ones were assigned using the os.urandom() function.

- Files were saved in the operating system's temporary directory, or saved as "temporary files" generated by the tempfile.NamedTemporaryFile() function.

However, this solution soon proved untenable. Files still accumulated in the temporary directory. With multiple Flask views requiring access to the data, it was not clear when a user-uploaded "temporary file" could be deleted safely.

5.2.1.2 Passing data between views using jsonpickle

The program needed to be able to accept user-uploaded data without saving files to disk.

The critical insight was the possibility of using the session dictionary. In the Flask framework, the Session is a data structure shared between views and templates. If the data parsed from the data file could be saved in the session dictionary, any view or template that required the information could access that, rather than reading from a file.

Unfortunately, initial planning and design of the Pond, PondShape, and other classes had not accounted for the need to serialize these classes into a JSON format. Therefore, they could not be simply saved to the session dictionary.

Instead, the upload method reads the file, creates a list of pond objects from the data, then uses the jsonpickle library to serialize them and put them in the session dictionary for access by other views. This solution was easy to implement, but can result in relatively large serialized objects. To improve memory usage, future versions of the program should investigate the implementation of dedicated save/load methods in the Pond object that serialize only the data.

5.2.1.3 Exporting data for download

Once the output values have been calculated, they can be used in other views, or exported to a .xls file for download. Writing to a .xls file is accomplished via the xlwt package.

Currently, the export function/view outputs a .xls file with two sheets:

1. Daily Statistics. This sheet holds
   - year: 4-digit year. This field, along with *Lake ID* and *day of year*, uniquely identify a Pond on a particular date.
   - Lake ID: the name of the lake.
   - day of year: numerical day of year (e.g. “0” = Jan 1st, “60” = Feb 29, or March 1st)
   - bppr_m2: daily benthic primary production, $mgC * m^{-2}$.
   - pppr_m2: daily phytoplanktonic primary production $mgC * m^{-2}$.
2. Hourly Statistics
   - year: 4-digit year. This field, along with *Lake ID* and *day of year*, uniquely identify a Pond on a particular date.
   - Lake ID: the name of the lake.
   - day: numerical day of year (e.g. “0” = Jan 1st, “60” = Feb 29, or March 1st)
   - layer: the thermal layer of the lake. 1 = epilimnion, 2 = metalimnion, 3 = hypolimnion
   - hour: hours since sunrise.
   - ppr_m3: phytoplanktonic primary production, in $mgC * m^{-3} * h^{-1}$

5.2.2 Prototyping with PythonAnywhere

Initial prototyping was done using the cloud service Python Anywhere. The service comes pre-installed with all relevant packages used by the project (e.g. SciPy, xlrd), and allows us to focus on development of the app's backend objects and classes before concentrating on the web aspect.

5.2.3 VMware cluster-based execution

Hosting of the application is intended to be on an Ubuntu system in the Wright State University cluster administered by the Kno.e.sis group. To ensure compatibility beforehand, an Ubuntu VM is used for testing the application.

5.2.4 Deployment using NGINX and Gunicorn

The Flask microframework development server was sufficient for testing and development, but is not advised to be used in a production system. Deployment was therefore done using a NGINX server, with Gunicorn acting as the WSGI to Flask.

5.3 Discussion

5.3.1 Lessons learned

Over the course of the project's development process certain issues caused problems.

5.3.1.1 Coupling, and difficulties in adding or removing data fields

Some trouble occurred when, during the course of development, it became clear that certain data fields would need to be deleted, and others added, to the program.

The program, however, had already developed a significant amount of coupling between classes. Thus, it became difficult to make changes to the Pond class, or to the data input template, without necessitating many other changes to other parts of the program.

In one case (the addition of the *year* field), it was necessary to make a checklist with approximately 20 items before undertaking the task (Appendix D).

Future work should attempt to mitigate this issue, perhaps through the use of .config files or a project-wide "constants" class.

5.3.1.2 Including web consideration in original design: consider making objects JSON-serializable

When working on a web-related project, it may be important to consider from the beginning whether to include some sort of serialization function in classes, rather than having to include or work around this functionality late in development.

5.3.2 Future improvements

There are several possible improvements to this project:

5.3.2.1 Improved user control and user feedback

Currently, if the user wishes to specify a different file format for uploading data, there is no such capability. Nor is it possible to easily read or convert data files that do not match the template file exactly.

Furthermore, if any error should occur in the data entry or upload, the error information provided to the user may not be sufficient to pinpoint it.

5.3.2.2 Graphing and visualization

Users may wish to have visualized P/I curves or similar features. This feature was partially implemented using the matplotlib package, but cut from the project to focus on reading and outputting data files.

5.3.2.3 Online data entry

Users may wish to have the option to enter data in an online form, rather than downloading, filling out, and uploading a template.

Conclusions and Future Work

6.1 Conclusions

Based on the lessons learned in the course of the project, we draw the following conclusions:

6.1.1 Results consistent with expectations; further validation may be desirable

Validation of benthic primary production output values were within 2-5% of the original Vadeboncoeur model, even when only 5 data points are used for P/I curve parameters. This is opposed to the original data set, which contained over 100 data points (1 for every 0.1 meter depth interval). For further details on the Vadeboncoeur model for estimating benthic primary production, see Vadeboncoeur et al. [2008], and Devlin [2015].

Proper validation of phytoplanktonic primary production results faces significant challenges. As it is difficult to find matched sets of input parameters with output results for primary production, validation consists primarily of piecing together data sets from disparate sources, calculating based on these parameters, and then attempting to compare the results with public databases (see Appendix B for details on the testing and validation process).

However, even when this is accomplished, issues arise:

- Primary production results, such as the NTL LTER database, are typically themselves based on estimates.

- The details of estimation techniques can cause dramatic variations in the end results. For example, differences in parameter selection techniques (the process of estimating P/I curve parameters such as P_{max}) cause dramatic differences in estimates of primary production (Frenette and Demers [1993]). Researchers may even choose to use different equations in different circumstances, i.e. using a photoinhibition-based or non-photoinhibition-based equation depending on surface light conditions. The choice of equation may not always be clearly indicated.

The program's calculated phytoplanktonic primary production values do not exactly match those of the database. Without knowing the exact parameters and equations used in calculating the values in the database, it is difficult to determine the exact causes of these discrepancies.

However, when plotted together on a graph (such as in Figure 4.3), results show qualitative similarity to results from the NTL LTER database, responding as expected to changes in parameters such as water clarity or light levels.

We conclude, therefore, that discrepancies between the program's results and those of the database are likely due to differing choices in parameter estimation or choice of PPR equation, rather than errors in the implementation.

6.2 Future work

Further work should focus on the following areas:

- Improved user experience by the addition of online data-entry forms and clearer error messages when there are problems with the uploaded file.

- Further investigate what are the minimum data points needed for "acceptable" accuracy.
- Investigate public data sources for further validation data, or creation of a new data set for this purpose.
  - It may be possible to record user-uploaded data in a database, to help alleviate the problem of data scarcity. This would likely necessitate the implementation of a user/login system, and integration of the program with a database system.

Project implementation details and source code

This is the source code for the program. It is in several parts, corresponding to a Model-View-Controller architecture. GitHub Source code can be found at Project GitHub Repository

A.1 Project implementation details

In this section I will talk about what I did, and why. Note that Appendix A contains the full source code.

A.1.1 Libraries

Libraries used in this project:

- SciPy: This contains within it many libraries. Used for various mathematical or scientific functions, including interpolation.
  - NumPy: Used for mathematical functions such as tanh and sin.
  - Matplotlib: used for graphs and plots initially, though the revised project scope rendered it unused.

A.2 Program Architecture

The project uses an Object-Oriented approach, using separate classes for data relating to the shape of the pond, and data photosynthesis parameters.

The final form of the code consists of the following basic structure.

- Model
  - Pond A.3
  - Photosynthesis Measurement A.3.1
  - Benthic Photosynthesis Measurement A.3.2
  - Phytoplankton Photosynthesis Measurement A.3.3
  - Pond Shape A.3.4
  - Bathymetric Pond Shape A.3.5
- View
  - Flask app A.4
- Controller
  - Data Reader A.5

A.3 Model Source Code (Python)

In this section is the source code for the lake model. It is separate from the code used to run the server that views the results, or code to control and update the model.

A.3.1 pond.py

This code is for the Pond object, which holds the various equations and metrics relating to the water body as a whole.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 05 09:38:17 2014

@author: cdleong
"""
import traceback
import math as mat
import numpy as np
from pond_shape import PondShape
from benthic_photosynthesis_measurement import BenthicPhotosynthesisMeasurement
from bathymetric_pond_shape import BathymetricPondShape
from scipy.interpolate import interp1d
from phytoplankton_photosynthesis_measurement import
    PhytoPlanktonPhotosynthesisMeasurement
```

```
class Pond(object):

    ###################################
    # CONSTANTS
    ###################################
    MINIMUM_VALID_YEAR = 1 #If you want to do lakes from year 0 or B.C., you code
     it.
    MAXIMUM_VALID_YEAR = 9999 #I'm sorry, people in the year 10000 A.D., and/or
    time travellers.

    MINIMUM_VALID_DAY = 0  # New Year's Day
```

```
MAXIMUM_VALID_DAY = 366  # New Year's Eve in a leap year.

MINIMUM_LENGTH_OF_DAY = 0.0  # north of the arctic circle and south of the antarctic one, this is possible during winter.
MAXIMUM_LENGTH_OF_DAY = 24.0003  # north of the arctic circle and south of the antarctic one, this is possible during summer, if there's a leap second

MINIMUM_NOON_SURFACE_LIGHT = 0.0  # Total darkness. TODO: will this cause divide-by-zero errors?
MAXIMUM_NOON_SURFACE_LIGHT = 1000000.0  # normally it's in the range of ~1000-2000. A 1000x increase, I'm pretty sure, would sterilize the lake. And probably the Earth. According to http://autogrow.com/general-info/light-measurement, the highest it generally gets on Earth is 2000. Don't wanna exclude nuclear weapons going off over lakes, though, you know?


MINIMUM_LIGHT_ATTENUATION_COEFFICIENT = 0.0  # totally, perfectly clear. Units in inverse meters. M^-1 #TODO: will this cause divide-by-zero errors?
MAXIMUM_LIGHT_ATTENUATION_COEFFICIENT = 100  # Close enough to totally opaque to make no practical difference. At this level, light goes from 100% at depth 0 to 0.005% at 10 centimeters. No meaningful photosynthesis is likely to be going on in this lake.

PHOTIC_ZONE_LIGHT_PENETRATION_LEVEL_LOWER_BOUND = 0.01  # 1% light penetration is the definition of the photic zone from various sources, including Vadeboncoeur 2008, and http://limnology.wisc.edu/courses/zoo316/REVIEW%20OF%20A%20FEW%20MAJOR%20CONCEPTS.html

MAXIMUM_LAKE_ID_LENGTH = 140  # value picked arbitrarily during specification process. I chose it because I had Twitter on the mind. Possibly too long. 50 characters would give us enough characters for "Lake Chargoggagoggmanchaoggagoggchaubunaguhgamaugg", the longest lake name in the world...

MAXIMUM_NUMBER_OF_THERMAL_LAYERS = 3 #epilimnion, hypolimnion, metalimnion

DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS = 0.1  # ten centimeters, 0.1 meters. Arbitrary.

DEFAULT_FREEZE_DAY = 349  # December 15 #arbitrary default value, based on median from http://www.epa.gov/climatechange/pdfs/CI-snow-and-ice-2014.pdf
DEFAULT_THAW_DAY = 135  # May 15 #arbitrary default value, based on median from http://www.epa.gov/climatechange/pdfs/CI-snow-and-ice-2014.pdf
```

```
BASE_TIME_UNIT = 1 #hours


###################################
# VARIABLES
###################################

# identifying variables, aka Primary Key
year = 1900
lake_ID = ""  # invalid lake ID I'm assuming. #TODO: check.
day_of_year = 0  # day of year 0-366


# General pond information. Light/photosynthesis
length_of_day = 15  # hours of sunlight
noon_surface_light = 1500  # micromol*m^(-2)*s^(-1 ) (aka microEinsteins?)
light_attenuation_coefficient = 0.05  # aka "kd"

# shape object. Holds information regarding the shape of the lake. Methods
include area at depth, volume at depth, etc.
pond_shape_object = PondShape()

# benthic photosynthesis data list
benthic_photosynthesis_measurements = []

# phytoplankton photosynthesis data list #TODO: everything to do with this
phytoplankton_photosynthesis_measurements = []


# default intervals for calculations is quarter-hours
time_interval = 0.25




############################
# CONSTRUCTOR
###########################
def __init__(self,
             year = 1900,
             lake_ID="",
             day_of_year=0,
```

```
                length_of_day=0.0,
                noon_surface_light=0.0,
                light_attenuation_coefficient=0.0,
                pond_shape_object=PondShape(),
                benthic_photosynthesis_measurements=[],
                phytoplankton_photosynthesis_measurements=[],
                time_interval=0.25):
        '''
        CONSTRUCTOR
        @param lake_ID: string
        @param day_of_year: integer day of year
        @param length_of_day: float number of hours
        @param noon_surface_light: float
        @param light_attenuation_coefficient: float
        @param pond_shape_object: a PondShape object
        @param benthic_photosynthesis_measurements: a list of
BenthicPhotoSynthesisMeasurements
        @param phytoplankton_photosynthesis_measurements:  a list of
PhyttoplanktonPhotoSynthesisMeasurements
        '''
        self.set_year(year)
        self.set_lake_id(lake_ID)
        self.set_day_of_year(day_of_year)
        self.length_of_day = length_of_day
        self.noon_surface_light = noon_surface_light
        self.light_attenuation_coefficient = light_attenuation_coefficient
        self.set_pond_shape(pond_shape_object)
        self.set_benthic_photosynthesis_measurements(
benthic_photosynthesis_measurements)
        self.set_phytoplankton_photosynthesis_measurements(
phytoplankton_photosynthesis_measurements)
        self.set_time_interval(time_interval)





###################
# VALIDATORS
###################

def validate_numerical_value(self, value, max_value, min_value):
        '''
```

```
    Generic numerical validator.
    Checks if value is >max_value or <min_value.
    If it's outside the valid range it'll be set to the closest valid value.
    @param value: numerical value of some sort to be checked.
    @param max_value: numerical value. Max valid value.
    @param min_value: numerical value. Min valid value.
    @return: a valid value in the range (max_value,min_value), inclusive
    @rtype: numerical value
    '''
    validated_value = 0
    if(value < min_value):
        validated_value = min_value
    elif(value > max_value):
        validated_value = max_value
    else:
        validated_value = value
    return validated_value

def validate_year(self, year):
    '''
    Checks if year is set to reasonable value. If not, returns minimum or
maximum possible reasonable value (whichever is closest)
    @param year:
    @return: A valid value in the range (Pond.MAXIMUM_VALID_YEAR, Pond.
MINIMUM_VALID_YEAR), inclusive
    @rtype: int
    '''
    return self.validate_numerical_value(year, Pond.MAXIMUM_VALID_YEAR, Pond.
MINIMUM_VALID_YEAR)

def validate_day_of_year(self, day_of_year=0):
    '''


    @param day_of_year: the day of year the measurement was made.
    @return: a valid value in the range (Pond.MAXMUM_VALID_DAY,Pond.
MINIMUM_VALID_DAY), inclusive
    @rtype: int
    '''
    return self.validate_numerical_value(day_of_year, Pond.MAXIMUM_VALID_DAY,
 Pond.MINIMUM_VALID_DAY)

def validate_length_of_day(self, length_of_day=0.0):
    '''
```

```
        Checks if length_of_day is set to reasonable value. If not, returns
minimum or maximum possible reasonable value (whichever is closest)
        @param length_of_day:
        @return: a valid value in the range (Pond.MAXIMUM_LENGTH_OF_DAY,Pond.
MAXIMUM_LENGTH_OF_DAY), inclusive
        @rtype:  float
        '''
        return self.validate_numerical_value(length_of_day, Pond.
MAXIMUM_LENGTH_OF_DAY, Pond.MINIMUM_LENGTH_OF_DAY)

    def validate_proportional_value(self, proportional_value):
        '''
        Checks if a proportional value is actually within a reasonable range. If
not, returns minimum or maximum possible reasonable value (whichever is
closest)
        @param proportional_value:
        @return: a value in the range (0.0, 1.0) inclusive
        @rtype: float
        '''
        validated_proportional_value = self.validate_numerical_value(
proportional_value, 1.0, 0.0)
        return validated_proportional_value

    def validate_depth(self, depth=0.0):
        '''
        Checks if depth is set to reasonable value. If not, returns minimum or
maximum possible reasonable value (whichever is closest)
        @param depth:
        @return: a depth in the range (0.0 (the surface), and the max depth of
the lake) inclusive
        @rtype: float
        '''
        pond_shape_object = self.get_pond_shape()
        validated_depth = pond_shape_object.validate_depth(depth)
        return validated_depth

    def validate_noon_surface_light(self, noon_surface_light=0.0):
        '''
        Checks if noon_surface_light is set to plausible value. If not, returns
minimum or maximum possible reasonable value (whichever is closest)
        @param noon_surface_light:
        @return: a value in the range (Pond.MAXIMUM_NOON_SURFACE_LIGHT, Pond.
MINIMUM_NOON_SURFACE_LIGHT) inclusive
```

```
    @rtype: float
    '''
    validated_noon_surface_light = self.validate_numerical_value(
noon_surface_light, Pond.MAXIMUM_NOON_SURFACE_LIGHT, Pond.
MINIMUM_NOON_SURFACE_LIGHT)
    return validated_noon_surface_light

def validate_light_attenuation_coefficient(self,
light_attenuation_coefficient):
    '''
    Checks if light attenuation is set to reasonable value. If not, returns
minimum or maximum possible reasonable value (whichever is closest)
    @param light_attenuation_coefficient:
    @return: a value in the range (Pond.MAXIMUM_LIGHT_ATTENUATION_COEFFICIENT
, Pond.MINIMUM_LIGHT_ATTENUATION_COEFFICIENT) inclusive
    @rtype: float
    '''
    validated_light_attenuation_coefficient = self.validate_numerical_value(
light_attenuation_coefficient, Pond.MAXIMUM_LIGHT_ATTENUATION_COEFFICIENT,
Pond.MINIMUM_LIGHT_ATTENUATION_COEFFICIENT)
    return validated_light_attenuation_coefficient

def validate_types_of_all_items_in_list(self, items=[], desired_type=object):
    '''
    Checks if every item in a list is of the correct type of object.
    @param items:
    @param desired_type:
    @return: True if all items in the list are the given type. False
otherwise.
    @rtype: boolean
    '''
    all_valid = False
    if(all(isinstance(item, desired_type) for item in items)):
        all_valid = True
    else:
        all_valid = False
    return all_valid



def validate_time(self, time):
    '''
    Checks if time is set to reasonable value. If not, returns minimum or
maximum possible reasonable value (whichever is closest)
```

```
    Differs from validate_length_of_day in that it checks the value of time
against the length of day of this lake.
    @param light_attenuation_coefficient:
    @return: a value in the range (self.get_length_of_day(), Pond.Pond.
MINIMUM_LENGTH_OF_DAY) inclusive
    @rtype: float
    '''
    length_of_day = self.get_length_of_day()
    validated_time = self.validate_numerical_value(time, length_of_day, Pond.
MINIMUM_LENGTH_OF_DAY)
    return validated_time







#######################
# GETTERS
#######################
def get_key(self):
    '''
    Get Key
    Used for convenient identification of ponds. So long as none of them has
the same year, ID, and day, it works.
    @return: year + lake_id + day
    @rtype: string
    '''
    string1 = str(self.get_year())
    string2 = str(self.get_lake_id())
    string3 = str(self.get_day_of_year())
    return str(string1+string2+string3) #TODO:thisisridiculous

def get_year(self):
    '''
    Get Year
    @return: year
    @rtype: int
    '''
    return int(self.__year)


def get_lake_id(self):
```

```
        '''
        Get Lake ID
        @return: the ID of the lake.
        @rtype: string
        '''
        return self.__lake_ID


    def get_day_of_year(self):
        '''
        Get Day of Year
        @return: the day of on which measurements occurred.
        @rtype: float
        '''
        #TODO: fix this. only made it float so I could fix a temporary issue with
     the test data.
        return int(self.__day_of_year)


    def get_length_of_day(self):
        '''
        Get Length Of Day
        @return: the length of day, in hours, on the day of measurements.
        @rtype: float
        '''
        return self.__length_of_day


    def get_noon_surface_light(self):
        '''
        Get Noon Surface Light
        @return: The surface light intensite at solar noon, in micromoles per
    square meter per second(umol*m^-2*s^-1)
        @rtype: float
        '''
        return self.__noon_surface_light


    def get_light_attenuation_coefficient(self):
        '''
        Get Light Attenuation Coefficient.
        AKA background light attenuation, kd.
        @return:light attenuation coefficient.
```

```python
        @rtype: float
        '''
        return self.__light_attenuation_coefficient

    def get_pond_shape(self):
        '''
        Get Pond Shape
        @return: a PondShape object, holding all the information describing the
shape of the lake.
        @rtype: PondShape
        '''
        return self.pond_shape_object



    def get_benthic_photosynthesis_measurements(self):
        '''
        Get Benthic Photosynthesis Measurements
        @return: the list containing all the Benthic Photosynthesis Measurement
objects, that hold the information regarding benthic photosynthesis.
        @rtype: list containing BenthicPhotosynthesisMeasurement objects
        '''
        return self.__benthic_photosynthesis_measurements



    def get_phytoplankton_photosynthesis_measurements(self):
        '''
        Get Phytoplankton Photosynthesis Measurements
        @return: the list containing all the Phytoplankton Photosynthesis
Measurement objects, that hold the information regarding benthic
photosynthesis.
        @rtype: list containing PhytoplanktonPhotoSynthesisMeasurement objects
        '''
        return self.__phytoplankton_photosynthesis_measurements








    def get_max_depth(self):
        '''
        Get Max Depth
```

```
    Calls get max depth method in PondShape instance.
    @return: maximum depth of lake
    '''
    return self.get_pond_shape().get_max_depth()



def get_time_interval(self):
    '''
    Get Time Interval
    Get the time interval used for calculations.
    @rtype: float
    '''
    return self.__time_interval

def get_list_of_times(self):
    '''
    Gets a list of the times of day used for calculations.

    Example: if the day length was 2 hours, and the time interval was 0.25 (
quarter-hours), this would return
    [0.0,0.25,0.5,0.75,1.0,1.25,1.5,1.75,2.0]
    @rtype: list
    '''
    start_time = 0.0
    end_time = self.get_length_of_day()
    time_interval = self.get_time_interval()
    time_list = []
    time =start_time
    while time<=end_time:
        time_list.append(time)
        time+=time_interval


    return time_list




######################
# SETTERS
######################

def set_year(self, year):
```

```
        '''
        Set year
        Validates it first using validate_year
        '''
        self.__year = self.validate_year(year)

    def set_lake_id(self, lake_id):
        '''
        Set Lake ID
        @param lake_id:
        '''
        self.__lake_ID = lake_id



    def set_day_of_year(self, day_of_year):
        '''
        Set Day Of Year
        Validates the value
        @param day_of_year:
        '''
        validated_day_of_year = self.validate_day_of_year(day_of_year)
        self.__day_of_year = validated_day_of_year



    def set_length_of_day(self, length_of_day):
        '''
        Set Length Of Day
        Validates the value
        @param length_of_day:
        '''
        validated_length_of_day = self.validate_length_of_day(length_of_day)
        self.__length_of_day = validated_length_of_day



    def set_noon_surface_light(self, noon_surface_light):
        '''
        Set Noon Surface Light
        Validates the value
        @param noon_surface_light:
        '''
        validated_light = self.validate_noon_surface_light(noon_surface_light)
        self.__noon_surface_light = validated_light
```

```
def set_light_attenuation_coefficient(self, light_attenuation_coefficient):
    '''
    Set Light Attenuation Coefficient
    Validates the value
    @param light_attenuation_coefficient: Also known as light extinction
coefficient, or just kd. Units in inverse meters.
    '''
    validated_light_attenuation_coefficient = self.
validate_light_attenuation_coefficient(light_attenuation_coefficient)
    self.__light_attenuation_coefficient =
validated_light_attenuation_coefficient



def set_time_interval(self, time_interval):
    '''
    Set Time Interval
    @param time_interval: fractional hours. For example, 0.5 = half hours,
0.25 = 15 minutes.
    '''
    self.__time_interval = time_interval

def set_pond_shape(self, pond_shape_object):
    '''
    Set Time Interval
    Validates the value
    @param pond_shape_object: a PondShape object of some type. So long as it
extends PondShape, it should work.
    '''
    if(isinstance(pond_shape_object, PondShape)):
        self.pond_shape_object = pond_shape_object
    else:
        raise Exception("cannot set pond shape. Invalid type")



def set_benthic_photosynthesis_measurements(self, values=[]):
    '''
    Set Benthic Photosynthesis Measurements
    Given a list of BenthicPhotosynthesisMeasurement objects, replaces the
current list with values.
    Validates the list using validate_types_of_all_items_in_list()
```

```
    '''
    all_valid = self.validate_types_of_all_items_in_list(values,
BenthicPhotosynthesisMeasurement)
    if(all_valid):
        self.__benthic_photosynthesis_measurements = values
    else:
        raise Exception("ERROR: all values in
benthic_photosynthesis_measurements must be of type
BenthicPhotosynthesisMeasurement")


def set_phytoplankton_photosynthesis_measurements(self, values=[]):
    '''
    Set Phytoplankton Photosynthesis Measurements
    Given a list of PhytoPlanktonPhotosynthesisMeasurement objects, replaces
the current list with values.
    Validates the list using validate_types_of_all_items_in_list()
    Also makes sure that there are less than or equal to
MAXIMUM_NUMBER_OF_THERMAL_LAYERS measurements.
    '''
    # TODO: use a dict to ensure 3 unique layers.
    all_valid = self.validate_types_of_all_items_in_list(values,
PhytoPlanktonPhotosynthesisMeasurement)
    length_valid = len(values) <= self.MAXIMUM_NUMBER_OF_THERMAL_LAYERS

    if(not all_valid):
        raise Exception("ERROR: all values in
phytoplankton_photosynthesis_measurements must be of type
PhytoPlanktonPhotosynthesisMeasurement")
    elif(not length_valid):
        raise Exception("ERROR: there must be 0 to 3 thermal layers")
    else:
        self.__phytoplankton_photosynthesis_measurements = values




#############################
# DELETERS
#############################

def del_year(self):
```

```
        del self.__year

    def del_lake_id(self):
        del self.__lake_ID


    def del_day_of_year(self):
        del self.__day_of_year


    def del_length_of_day(self):
        del self.__length_of_day


    def del_noon_surface_light(self):
        del self.__noon_surface_light


    def del_light_attenuation_coefficient(self):
        del self.__light_attenuation_coefficient


    def del_benthic_photosynthesis_measurements(self):
        del self.__benthic_photosynthesis_measurements


    def del_phytoplankton_photosynthesis_measurements(self):
        del self.__phytoplankton_photosynthesis_measurements


    def del_time_interval(self):
        del self.__time_interval


    ########################################
    # Properties
    ########################################
    #TODO: write decent docstrings
    year = property(get_year, set_year, del_year, "year's docstring")
    lake_ID = property(get_lake_id, set_lake_id, del_lake_id, "lake_ID's
    docstring")
    day_of_year = property(get_day_of_year, set_day_of_year, del_day_of_year, "
    day_of_year's docstring")
```

```
length_of_day = property(get_length_of_day, set_length_of_day,
del_length_of_day, "length_of_day's docstring")
noon_surface_light = property(get_noon_surface_light, set_noon_surface_light,
 del_noon_surface_light, "noon_surface_light's docstring")
light_attenuation_coefficient = property(get_light_attenuation_coefficient,
set_light_attenuation_coefficient, del_light_attenuation_coefficient, "
light_attenuation_coefficient's docstring")
benthic_photosynthesis_measurements = property(
get_benthic_photosynthesis_measurements,
set_benthic_photosynthesis_measurements,
del_benthic_photosynthesis_measurements, "benthic_photosynthesis_measurements
's docstring")
phytoplankton_photosynthesis_measurements = property(
get_phytoplankton_photosynthesis_measurements,
set_phytoplankton_photosynthesis_measurements,
del_phytoplankton_photosynthesis_measurements, "
phytoplankton_photosynthesis_measurements's docstring")
time_interval = property(get_time_interval, set_time_interval,
del_time_interval, "time_interval's docstring")



########################################
# Appenders/mutators
########################################
def add_benthic_measurement(self, measurement=
BenthicPhotosynthesisMeasurement):
    if(isinstance(measurement, BenthicPhotosynthesisMeasurement)):
        self.benthic_photosynthesis_measurements.append(measurement)
    else:
        raise Exception("ERROR: cannot add measurement to benthic
measurements list - measurement must be of type
BenthicPhotosynthesisMeasurement")

def add_benthic_measurement_if_photic(self, measurement):
    z1Percent = self.calculate_depth_of_specific_light_percentage(self.
PHOTIC_ZONE_LIGHT_PENETRATION_LEVEL_LOWER_BOUND)
    if(measurement.get_depth() <= z1Percent):
        self.add_benthic_measurement(measurement)
    else:
        raise Exception("measurement not within photic zone")
```

```
def add_phytoplankton_measurement(self, measurement=
PhytoPlanktonPhotosynthesisMeasurement):
    if(isinstance(measurement, PhytoPlanktonPhotosynthesisMeasurement)):
        if(len(self.phytoplankton_photosynthesis_measurements) > 0):
            existing_measurement = next((i for i in self.
phytoplankton_photosynthesis_measurements if (i.get_thermal_layer() ==
measurement.get_thermal_layer())), None)  # source: http://stackoverflow.com/
questions/7125467/find-object-in-list-that-has-attribute-equal-to-some-value-
that-meets-any-condi
            if(existing_measurement is not None):
                index = measurement.get_thermal_layer() - 1
                self.phytoplankton_photosynthesis_measurements.remove(
existing_measurement)
                try:
                    self.phytoplankton_photosynthesis_measurements.insert(
index, measurement)
                except TypeError:
                    error = "TypeError: index is ", index, " and measurement
is ", measurement, " for pond ", self.get_lake_id(), " day ", self.
get_day_of_year()
                    raise Exception(error)



        self.phytoplankton_photosynthesis_measurements.append(measurement)
    else:
        raise Exception("ERROR: cannot add measurement to benthic
measurements list - measurement must be of type
PhytoPlanktonPhotosynthesisMeasurement")



def remove_benthic_measurement(self, measurement=
BenthicPhotosynthesisMeasurement):
    self.benthic_photosynthesis_measurements.remove(measurement)

def update_shape(self, other_pond_shape):
    our_shape = self.get_pond_shape()
    if(isinstance(other_pond_shape, BathymetricPondShape)):
        our_shape.update_shape(other_pond_shape)
        self.pond_shape_object = our_shape

############################################
############################################
```

```
# # SCIENCE FUNCTIONS
# # This section is where the science occurs.
############################################
############################################


###########################################################
# BENTHIC PHOTO METHODS
###########################################################

##############################
# BENTHIC PRIMARY PRODUCTIVITY
##############################
def calculate_daily_whole_lake_benthic_primary_production_m2(self,
depth_interval=DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS, use_littoral_area=
True):
    '''

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    Almost everything else in this entire project works to make this method
work.
    #TODO: (someday) allow specification of littoral or surface area
    #TODO: (someday) user-specified depth interval.
    @return: Benthic Primary Production, mg C per meter squared, per day.
    @rtype: float

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    '''

    time_interval = self.get_time_interval()
    length_of_day = self.get_length_of_day()  # TODO: Fee normalized this
around zero. Doesn't seem necessary, but might affect the periodic function.


    benthic_primary_production_answer = 0.0  # mg C per day
    current_depth_interval = 0.0
    previous_depth = 0.0
    current_depth = 0.0
    total_littoral_area=0.0
    total_littoral_area = self.calculate_total_littoral_area()
```

```
    total_surface_area = self.get_pond_shape().
get_water_surface_area_at_depth(0.0)

    # for each depth interval #TODO: integration over whole lake?
    while current_depth < self.calculate_photic_zone_lower_bound():
        bpprz = 0.0  # mg C* m^-2 *day

        # depth interval calculation
        previous_depth = current_depth
        current_depth += depth_interval
        current_depth_interval = current_depth - previous_depth




        area = self.get_pond_shape().get_sediment_area_at_depth(current_depth
, current_depth_interval)

        try:
            ik_z = self.get_benthic_ik_at_depth(current_depth)
            benthic_pmax_z = self.get_benthic_pmax_at_depth(current_depth)
        except:
            raise

        if(True == use_littoral_area):
            f_area = area / total_littoral_area  # TODO: these add up to 1.0,
 right?
        else:
            f_area = area / total_surface_area



        # for every time interval
        t = 0.0  # start of day
        while t < length_of_day:
            bpprzt = 0.0
            izt = self.calculate_light_at_depth_and_time(current_depth, t)
            bpprzt = self.calculate_benthic_primary_production_z_t(izt,
benthic_pmax_z, ik_z)

            bpprz += bpprzt

            t += time_interval
```

```
        bpprz = bpprz / (self.BASE_TIME_UNIT / time_interval)  # account for
the fractional time interval. e.g. dividing by 1/0.25 is equiv to dividing by
 4
        weighted_bpprz = bpprz * f_area  # normalizing




        benthic_primary_production_answer += weighted_bpprz

    return benthic_primary_production_answer






def get_benthic_pmax_at_depth(self, depth=0.0):
    '''
    Get Benthic Pmax At Depth
    Uses interpolation to get the pmax value at the specified depth, if not
known.
    Validates depth first.
    @return: value of pmax at specified depth.
    @rtype: float
    '''
    # if depth is lower than the depth of 1% light, pmax approaches zero.
    if(self.check_if_depth_in_photic_zone(depth) == False):
        return 0


    validated_depth = self.validate_depth(depth)
    pmax_values_list = []
    depths_list = []
    for measurement_value in self.get_benthic_photosynthesis_measurements():
        pmax_value = measurement_value.get_pmax()
        depth_value = measurement_value.get_depth()
        pmax_values_list.append(pmax_value)
        depths_list.append(depth_value)
    bpmax_at_depth = self.interpolate_values_at_depth(validated_depth,
depths_list, pmax_values_list)
    return bpmax_at_depth
```

```
def get_benthic_ik_at_depth(self, depth=0.0):
    '''
    Get Benthic Ik At Depth
    Uses interpolation to get the Ik value at the specified depth, if not
known.
    Validates depth first.
    @return: value of pmax at specified depth.
    @rtype: float
    '''
    validated_depth = self.validate_depth(depth)



    values_list = []
    depths_list = []
    for measurement_value in self.get_benthic_photosynthesis_measurements():
        ik_value = measurement_value.get_ik()
        depth_value = measurement_value.get_depth()
        values_list.append(ik_value)
        depths_list.append(depth_value)

    try:
        ik_at_depth = self.interpolate_values_at_depth(validated_depth,
depths_list, values_list)
    except:
        raise

    return ik_at_depth

def calculate_benthic_primary_production_z_t(self, light_at_time_and_depth,
benthic_pmax_z_t, benthic_ik_z_t):
    '''
    Benthic primary production rate at a specific depth and time
    @return:
    @rtype: float
    '''
    bpprzt = benthic_pmax_z_t * np.tanh(light_at_time_and_depth /
benthic_ik_z_t)
    return bpprzt

def get_benthic_measurements_sorted_by_depth(self):
    '''
    Sorted BenthicPhotosynthesisMeasurement list, by depth.
    @return: sorted benthic measurements
```

```
    @rtype: list of BenthicPhotosynthesisMeasurement objects.
    '''
    # http://stackoverflow.com/questions/403421/how-to-sort-a-list-of-objects
-in-python-based-on-an-attribute-of-the-objects
    unsorted_measurements = self.get_benthic_photosynthesis_measurements()
    sorted_measurements = sorted(unsorted_measurements, key=lambda x: x.
get_depth(), reverse=False)
    return sorted_measurements

##########################################################
# PHYTO PHOTO METHODS
##########################################################


def calculate_daily_whole_lake_phytoplankton_primary_production_m2(self,

depth_interval=DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS,

use_photoinhibition=None):
    '''

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    Calculate Daily Whole-lake Phytoplankton Primary Production
    Almost everything else in this entire project works to make this method
work.
    #TODO: (someday) allow specification of littoral or surface area
    #TODO: (someday) user-specified depth interval.
    @return: Benthic Primary Production, mg C per meter squared, per day.
    @rtype: float

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    '''




    layer_depths = self.get_thermal_layer_depths()
    layer_upper_bound = 0.0
    layer_lower_bound = 0.0
    layer_pp_list = []

    for layer_depth in layer_depths:
```

```
        layer_lower_bound = layer_depth



        layer_pp_daily_m2 = self.
calculate_phytoplankton_primary_production_rate_in_interval(layer_upper_bound
, layer_lower_bound, depth_interval, use_photoinhibition)
        layer_pp_list.append(layer_pp_daily_m2)
        layer_upper_bound = layer_lower_bound #set the new upper bound for
the next round of calculations using the current lower bound.


    pp_lake_daily_total_m2 = sum(layer_pp_list)
    return pp_lake_daily_total_m2  # mgC/m^2/day

def
calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_thermal_layer
(self,

                    layer = 0,

                    depth_interval=DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS,

                    use_photoinhibition=None,

                    convert_to_m2 = False):
    '''
    Used for graphing hourly rates over the course of a day, for a layer.
    @param layer: thermal layer of pond. 0=epilimnion, 1 = metalimnion, 2 =
hypolimnion
    @param depth_interval: the depth interval for calculations
    @param use_photoinhibition: whether or not to use the photoinhibition
equation.
    @param convert_to_m2: whether or not to convert the resulting values to (
per meter squared) instead of (per meter cubed)
    @return: list of hourly rates, over the course of a day. Length should be
 (length of day)/(time_interval)
    '''
    MAX_INDEX = self.MAXIMUM_NUMBER_OF_THERMAL_LAYERS-1
    MIN_INDEX  = 0

    validated_layer=self.validate_numerical_value(layer, MAX_INDEX, MIN_INDEX
) #0=epilimnion, 1=metalimnion, 2 = hypolimnion
```

```
    layer_depths = self.get_thermal_layer_depths()


    #say layer_depths contains [5.0,7.0,16.0].
    #if validate_layer = 0, We would want layer_upper_bound = 0.0,
layer_lower_bound = 5.0 (layer_depths[0])
    #if validate_layer = 1, We would want layer_upper_bound = 5.0 (
layer_depths[0]), layer_lower_bound = 7.0 (layer_depths[1])
    #if validate_layer = 2, We would want layer_upper_bound = 7.0 (
layer_depths[1]), layer_lower_bound = 16.0 (layer_depths[2])

    layer_upper_bound = 0.0
    if(validated_layer>0): #I just feel there's a more elegant way to do this
.
        layer_upper_bound=layer_depths[validated_layer-1]
    layer_lower_bound = layer_depths[validated_layer]
    pp_list = self.
calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_interval
(layer_upper_bound, layer_lower_bound, depth_interval, use_photoinhibition,
convert_to_m2)
    return pp_list




def
calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_interval
(self,

interval_upper_bound,

interval_lower_bound,

depth_interval=DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS,
```

```
use_photoinhibition=None,
                                                              convert_to_m2
 = False):
    '''
    Used for graphing hourly rates over the course of a day, but over a depth
 interval instead of a thermal layer.
    @param interval_upper_bound: depth in meters
    @param interval_lower_bound: depth in meters
    @param depth_interval: the depth interval for calculations
    @param use_photoinhibition: whether or not to use the photoinhibition
equation.
    @param convert_to_m2: whether or not to convert the resulting values to (
per meter squared) instead of (per meter cubed)
    @return: list of hourly rates, over the course of a day. Length should be
 (length of day)/(time_interval)
    '''
    if (use_photoinhibition is None):
        beta_parameter =self.get_phyto_beta_at_depth(interval_lower_bound)
        if(0==beta_parameter):
            use_photoinhibition = False
        else:
            use_photoinhibition = True



    # TODO: validate interval
    time_interval = self.get_time_interval()  # hours
    length_of_day = self.get_length_of_day()

    max_depth = self.get_pond_shape().get_max_depth()
    total_volume = self.get_pond_shape().get_volume_above_depth(max_depth,
depth_interval)
    layer_depth_interval = interval_lower_bound - interval_upper_bound  # "
deeper" is bigger magnitude, so instead of upper-lower we do lower - upper

    hourly_pp_list = []
    current_time = 0.0  # start of day
#         pp_layer_daily_total_hw_m3 = 0.0

    while current_time <= length_of_day:



        depth_m = interval_upper_bound #meters from surface.
```

```
        pp_total_in_thermal_layer_at_time_t_hw_m3 = 0.0 #primary production
in layer, mg C/ m^3 / hour, or mgC*m^-3*hr-1

        while depth_m <= interval_lower_bound:

            light_at_depth_z_time_t = self.calculate_light_at_depth_and_time(
depth_m, current_time)  # umol*m^-2*s^-1
            interval_volume_m3 = self.get_pond_shape().get_volume_at_depth(
depth_m, depth_interval)  # m^3
            fractional_volume = interval_volume_m3 / total_volume
            pp_rate_at_depth_z_time_t_m3 = self.
calculate_phytoplankton_primary_productivity(light_at_depth_z_time_t, depth_m
, use_photoinhibition)  # mgC*m^-3*hr^-1, or mgC per meter cubed per hour
            pp_total_at_depth_z_time_t_m3_in_one_time_unit =
pp_rate_at_depth_z_time_t_m3 * self.BASE_TIME_UNIT  # mgC*m^-3*hr^-1 * 1 hour
 = mgC*m^-3. This line usually multiplies by 1, changing nothing.
            pp_total_at_depth_z_time_t_hw_m3 =
pp_total_at_depth_z_time_t_m3_in_one_time_unit * fractional_volume  # mgC*m
^-3, hypsometrically weighted
            pp_total_in_thermal_layer_at_time_t_hw_m3 +=
pp_total_at_depth_z_time_t_hw_m3 #mgC*m^-3 #THIS IS WHAT I CHECKED TO TEST
AGAINST NTL LTER DATABASE
            depth_m += depth_interval




        if(current_time%time_interval==0):
            hourly_pp_list.append(pp_total_in_thermal_layer_at_time_t_hw_m3)
#             pp_layer_daily_total_hw_m3 +=
pp_total_in_thermal_layer_at_time_t_hw_m3   #Units are still mgC*m^-3



        current_time += time_interval




#
    if(convert_to_m2):
        hourly_pp_list = [value*layer_depth_interval for value in
hourly_pp_list] #multiply by the depth interval of the layer to convert to m2
```

```
    return hourly_pp_list  # mgC/m^2/day



def calculate_phytoplankton_primary_production_rate_in_interval(self,

interval_upper_bound,

interval_lower_bound,

depth_interval=DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS,

use_photoinhibition=None):
    '''
    Allows calculation of daily primary production in any valid depth
interval
    @param interval_upper_bound: depth in meters
    @param interval_lower_bound: depth in meters
    @param depth_interval: the depth interval for calculations
    @param use_photoinhibition: whether or not to use the photoinhibition
equation.
    '''

    if (use_photoinhibition is None):
        beta_parameter =self.get_phyto_beta_at_depth(interval_lower_bound)
        if(0==beta_parameter):
            use_photoinhibition = False
        else:
            use_photoinhibition = True


    # TODO: validate interval


    layer_depth_interval = interval_lower_bound - interval_upper_bound  # "
deeper" is bigger magnitude, so instead of upper-lower we do lower - upper




    #We have "per hour" calculated multiple times per hour. We must correct
for this in our summation.
    time_interval = self.get_time_interval()  # hours
```

```
    time_interval_correction_factor = (self.BASE_TIME_UNIT / time_interval)
# (hr/hr) Account for the fractional time interval. e.g. dividing by 1/0.25
is equiv to dividing by 4
    ppr_over_time_in_interval_list = self.
calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_interval
(interval_upper_bound, interval_lower_bound, depth_interval,
use_photoinhibition)
    pp_layer_daily_total_hw_m3 = sum(ppr_over_time_in_interval_list)


    pp_layer_daily_total_hw_time_corrected_m3 = pp_layer_daily_total_hw_m3/
time_interval_correction_factor




    #convert to m^-2
    #multiplying by the vertical distance from the top to the bottom of the
thermal layer is what was, apparently,
    #done to convert to mgC/m^-2 in the NTL LTER database.
    pp_layer_daily_total_hw_time_corrected_m2 =
pp_layer_daily_total_hw_time_corrected_m3 * layer_depth_interval
    return pp_layer_daily_total_hw_time_corrected_m2  # mgC/m^2/day





def get_phyto_pmax_at_depth(self, depth):
    '''
    @param depth: meters from surface
    @return:
    @rtype:
    '''
    validated_depth = self.validate_depth(depth)
    pmax = 0.0
    measurement = self.get_phytoplankton_photosynthesis_measurement_at_depth(
validated_depth)
    if(measurement is not None):
        pmax = measurement.get_pmax()
    return pmax

def get_phyto_alpha_at_depth(self, depth):
    '''
```

```
    @param depth: meters from surface
    @return:
    @rtype:
    '''
    validated_depth = self.validate_depth(depth)
    phyto_alpha = 0.0  # TODO: safer value?
    measurement = self.get_phytoplankton_photosynthesis_measurement_at_depth(
validated_depth)
    if(measurement is not None):
        phyto_alpha = measurement.get_phyto_alpha()
    return phyto_alpha

def get_phyto_beta_at_depth(self, depth):
    '''
    @param depth: meters from surface
    @return:
    @rtype:
    '''
    validated_depth = self.validate_depth(depth)

    phyto_beta = 0.0  # TODO: safer value?
    measurement = self.get_phytoplankton_photosynthesis_measurement_at_depth(
validated_depth)
    if(measurement is not None):
        phyto_beta = measurement.get_phyto_beta()



    return phyto_beta




def calculate_phytoplankton_primary_productivity(self, izt, depth,
use_photoinhibition=True):
    '''
    Calculate Phytoplankton Primary Productivity
    @param izt: light at depth z, time t (umol*m^-2*s^-1)
    @param depth: depth (m)
    @return ppr_z (mg*m^-3*hr^-1)
    '''
    ppr_z = 0.0
```

```
    phyto_photo_measurement_z = self.
get_phytoplankton_photosynthesis_measurement_at_depth(depth)
    if(phyto_photo_measurement_z is not None):


        phyto_pmax = self.get_phyto_pmax_at_depth(depth)  # mg C per m^3 per
hour (mg*m^-3*hr^-1)
        phyto_alpha = self.get_phyto_alpha_at_depth(depth)  # (mg*m^-3*hr^-1)
/(umol*m^-2*s^-1)
        phyto_beta = self.get_phyto_beta_at_depth(depth)  # (mg*m^-3*hr^-1)/(
umol*m^-2*s^-1)

        if(use_photoinhibition):
            # P-I CURVE EQUATION WITH PHOTOINHIBITION  P = Pmax*(1-exp(-alpha
*I/Pmax))*exp(-beta*I/Pmax)
            # P-I curve equation derived from Jassby/Platt, and specifically
from http://web.pdx.edu/~rueterj/courses/esr473/notes/pvsi.htm
            # that website, of course, got it from "Photoinhibition of
photosynthesis in natural assemblages of marine phytoplankton" By Platt, T.,
C.L. Gallegos and W.G. Harrison 1979.
            interim_value = 1 - mat.exp(-phyto_alpha * izt / phyto_pmax)  #
[(mg*m^-3*hr^-1)/(umol*m^-2*s^-1) * (umol*m^-2*s^-1) = (mg*m^-3*hr^-1)
            other_interim_value = mat.exp(-phyto_beta * izt / phyto_pmax)  #
(mg*m^-3*hr^-1), same as above
            ppr_z = phyto_pmax * interim_value * other_interim_value  # (mg*m
^-3*hr^-1)


        else:
            # P-I CURVE EQUATION WITH NO PHOTOINHIBITION. P = Pmax* tanh(
alpha*I/Pmax)
            ppr_z = phyto_pmax * mat.tanh(phyto_alpha * izt / phyto_pmax)


    return ppr_z  # (mg*m^-3*hr^-1)



def get_phytoplankton_photosynthesis_measurement_at_depth(self, depth):
    '''
    @param depth:
    @return: shallowest layer measurement deeper than specified depth, or
None if not found.
```

```
    @rtype:
    '''
    # find the shallowest layer_measurement that's deeper than this depth.
    # example: layers are at 5, 10, 15. Depth given is 5.5, then use
measurement for second layer.
    measurement = None
    reverse_sorted_measurements = self.get_phyto_measurements_sorted_by_depth
(True)  # sort reversed by depth.
    for layer_measurement in reverse_sorted_measurements:
        if (layer_measurement.get_depth() >= depth):
            measurement = layer_measurement
    return measurement



def get_thermal_layer_depths(self):
    '''
    Sorts and returns the depths of the thermal layers.
    @return: sorted list of thermal layer depths, from shallowest to deepest.
    @rtype: [] list
    '''
    layer_depth_list = []
    reverse_sorted_measurements = self.get_phyto_measurements_sorted_by_depth
(False)  # sort reversed by depth.
    for layer_measurement in reverse_sorted_measurements:
        layer_depth_list.append(layer_measurement.get_depth())
    return layer_depth_list



def get_phyto_measurements_sorted_by_depth(self, reverse=False):
    '''
    Sort
    @return: sorted benthic measurements
    @rtype: list of BenthicPhotosynthesisMeasurement objects.
    '''
    # http://stackoverflow.com/questions/403421/how-to-sort-a-list-of-objects
-in-python-based-on-an-attribute-of-the-objects
    thing = reverse
    unsorted_measurements = self.
get_phytoplankton_photosynthesis_measurements()
    sorted_measurements = sorted(unsorted_measurements, key=lambda x: x.
get_depth(), reverse=thing)
    return sorted_measurements
```

```
############################################################
# OTHER SCIENCE METHODS
############################################################
def check_if_depth_in_photic_zone(self, depth):
    '''
    Check If Depth In Photic Zone
    Used when adding photosynthesis measurements, calculating things, etc.
    @param depth: depth to check, in meters.
    @return: True if in photic zone, False if not.
    '''
    in_zone = True
    photic_zone_lower_bound = self.calculate_photic_zone_lower_bound()
    if(depth < 0 or depth > photic_zone_lower_bound):
        in_zone = False
    else:
        in_zone = True
    return in_zone


def calculate_photic_zone_lower_bound(self):
    '''
    Calculate Photic Zone Lower Bound
    This is actually redundant. It can be accomplished just by calling
calculate_depth_of_specific_light_percentage with
PHOTIC_ZONE_LIGHT_PENETRATION_LEVEL_LOWER_BOUND.
    That said, I might one day wish to decouple photic zone lower bound from
calculate_depth_of_specific_light_percentage, so I'm leaving this.
    NOTE: returns max depth if lower bound is deeper than max depth.
    @return: lower bound of the photic zone, in meters.
    @rtype: float
    '''
    lower_bound = self.calculate_depth_of_specific_light_percentage(self.
PHOTIC_ZONE_LIGHT_PENETRATION_LEVEL_LOWER_BOUND)
    max_depth = self.get_max_depth()
    if(lower_bound > max_depth):
        lower_bound = max_depth
    return lower_bound



def calculate_depth_of_specific_light_percentage(self,
desired_light_proportion=1.0):
```

```
    '''
    Calculate Depth Of Specific Light Proportion

    Calculates the depth of, say, 1% light.
    Uses: light attenuation coefficient kd.
    This is how "optical depth" works.

    Given a proportion, say 0.01 for 1%,
    calculates the depth of the pond at which that much light will reach.

    Equation on which this is based: Iz/I0=e^-kd*z
    Given a desired proportion for Iz/I0, and solved for z, this simplifies
to
    z= kd/ln(desired proportion)



    @param desired_light_proportion:a float value from 0 to 1.0
    @return: the depth, in meters, where that proportion of light penetrates.
    @rtype: float
    '''
    validated_desired_light_proportion = self.validate_proportional_value(
desired_light_proportion)
    depthOfSpecifiedLightProportion = 0.0  # the surface of the pond makes a
good, safe default depth
    backgroundLightAttenuation = self.get_light_attenuation_coefficient()




    if(validated_desired_light_proportion < 1.0 and
validated_desired_light_proportion > 0.0):
        naturalLogOfProportion = mat.log(validated_desired_light_proportion)

        depthOfSpecifiedLightProportion = naturalLogOfProportion / -
backgroundLightAttenuation  # TODO: check if zero.

    return depthOfSpecifiedLightProportion

def calculate_light_proportion_at_depth(self, depth=0.0):
    '''
    Calculate Light Proportion at Depth
```

```
    The inverse operation of "Calculate Depth Of Specific Light Proportion".
Given the depth, calculates what proportion of light
    will be visible at that depth.

    Given a depth, say "10" for 10 meters, calculates the proportion of light
 (Iz/I0) that will reach that depth

    Equation on which this is based: Iz/I0=e^-kd*z

    If you want Iz, just do Iz*I0 again. #TODO: just light at depth z

    @param depth: depth in meters.
    @return: proportion of light at depth z as a number in the range (0.0,
1.0), inclusive.
    @rtype:  float
    '''
    validated_depth = self.validate_depth(depth)
    light_attenuation_coefficient = self.get_light_attenuation_coefficient()
    multiplied = light_attenuation_coefficient * validated_depth
    light_proportion_at_depth = mat.exp(-multiplied)
    return light_proportion_at_depth




def calculate_light_at_depth_and_time(self, depth, time):
    '''
    Calculate Light At Depth And Time
    @param depth:
    @param time:
    @return: the light at that depth and time, in micromoles/m^2/sec
    @rtype: float
    '''

    validated_depth = self.validate_depth(depth)
    validated_time = self.validate_time(time)
    noonlight = self.get_noon_surface_light()
    length_of_day = self.get_length_of_day()
    surface_light_at_t = noonlight * np.sin(np.pi * validated_time /
length_of_day)
    light_attenuation_coefficient = self.get_light_attenuation_coefficient()
```

```
    light_at_z_and_t = surface_light_at_t * np.exp(-
light_attenuation_coefficient * validated_depth)
    return light_at_z_and_t



def calculate_total_littoral_area(self):
    '''
    Calculate Total Littoral Area
    Uses kd to calculate the depth of 1% light, then uses the pond_shape to
find sediment area above that.
    @return:
    @rtype:
    '''
    z1percent = self.calculate_photic_zone_lower_bound()
    shape_of_pond = self.get_pond_shape()

    littoral_area = shape_of_pond.get_sediment_area_above_depth(z1percent,
Pond.DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS)
    return littoral_area

def calculate_total_photic_volume(self):
    z1percent = self.calculate_photic_zone_lower_bound()
    shape_of_pond = self.get_pond_shape()

    photic_volume = shape_of_pond.get_volume_above_depth(z1percent, Pond.
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS)
    return photic_volume


def interpolate_values_at_depth(self, depth, depths_list=[], values_list=[]):
    '''
    INTERPOLATE VALUES AT DEPTH
    Essentially, given an array of "x" (validated_depths) and "y" values,
interpolates "y" value at specified validated_depth.

    Based on SciPy interpolation:http://docs.scipy.org/doc/scipy/reference/
tutorial/interpolate.html

    Used for things like, "I have pmax values at 0 and 1, but I need one at
0.5)
```

```
        @param depth: depth to interpolate _at_. In the above example, this would
     be 0.5
        @param depths_list: list of depths where we have data.
        @param values_list: corresponding data that goes with the depths. So
    value[0] is the value at depth[0] for example.
        @return: a single value, the value calculated for the specified depth.
        @rtype: a number. #TODO: what _kind_ of number?
        '''



        # Uses http://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.
    html
        validated_depth = self.validate_depth(depth)

        max_depth_given = max(depths_list)
        min_depth_given = min(depths_list)

        if(validated_depth > max_depth_given):
            validated_depth = max_depth_given
        elif(min_depth_given < min_depth_given):
            validated_depth = min_depth_given



        # get interpolation function #TODO: x, y need to be in order for scipy.
    interp1d to work I think. Check?
        x = depths_list
        y = values_list

        if(len(x)<2):
            traceback.print_exc()
            error_message = 'Cannot interpolate at depth ', depth,', because
    there are not enough data points!'
            error_message = str(error_message)
            print error_message
            raise Exception(error_message)
        f = interp1d(x, y)

        # magic from http://docs.scipy.org/doc/scipy/reference/tutorial/
    interpolate.html
        #SPLINES....!!!
#           tck = interpolate.splrep(x, y, s=0)
#           xnew = [validated_depth]
```

```
#         spline_interpolated = interpolate.splev(xnew, tck, der=0) #0th
    derivative
        linear_interpolated = f(validated_depth)



#         value_at_depth = spline_interpolated[0] #TODO: inefficient to get the
    whole array and return just one.
        value_at_depth = linear_interpolated
        return value_at_depth




    def calculate_depths_of_specific_light_percentages(self,
    light_penetration_depths):
        '''
        CALCULATE DEPTHS OF SPECIFIC LIGHT PERCENTAGES
        Given a list of light penetration depths returns the depth in meters
    needed for each of those light penetration levels.
        I only used this once, for making some testing data.
        @rtype: list
        '''
        depths = []
        for light_penetration_depth in light_penetration_depths:
            depth_m_of_light_penetration = self.
    calculate_depth_of_specific_light_percentage(light_penetration_depth)
            depths.append(depth_m_of_light_penetration)
        return depths




#############################################################################
# TESTING SECTION
#############################################################################
```

```
def main():
    '''
    TESTING TIME!!!

    '''
    print "Hello, world. You should never see this. I deleted all the test code
    that was here anyway."




if __name__ == "__main__":
    main()
```

Listing A.1: Pond Class

A.3.2 photosynthesis_measurement.py

This code is for the generic Photosynthesis Measurement object, which holds information relating to photosynthesis at various depths. Both benthic_photosynthesis_measurement and phytoplankton_photosynthesis_measurement.py are subclasses of this.

```
'''
Created on Jun 17, 2015

@author: cdleong
'''


class PhotosynthesisMeasurement(object):
    '''
    Abstract/generic class.
    double depth-         the depth, in meters, of the measurement
    double PMax         - From the Photosynthesis-Irradiance curve. Pmax = alpha*
    Ik
    '''
    ##########
    # KNOWS
    ##########
    depth = 0.0
```

```
    pmax = 0.0

    def __init__(self, depth=0.0, pmax=0.0):
        self.depth = depth
        self.pmax = pmax

    def get_depth(self):
        return self.__optical_depth

    def get_pmax(self):
        return self.__pmax

    def set_depth(self, value):
        self.__optical_depth = value

    def set_pmax(self, value):
        self.__pmax = value

    def del_depth(self):
        del self.__optical_depth

    def del_pmax(self):
        del self.__pmax

    depth = property(get_depth, set_depth, del_depth, "depth's docstring")
    pmax = property(get_pmax, set_pmax, del_pmax, "pmax's docstri")
```

```
def main():
    print "hello world"







if __name__ == "__main__":
    main()
```

Listing A.2: PhotosynthesisMeasurement Class

A.3.3 benthic_photosynthesis_measurement.py

This code is for the Benthic Photosynthesis Measurement object, which holds information relating to Benthic photosynthesis at various depths.

```
'''
Created on Jun 17, 2015

@author: cdleong
'''
from photosynthesis_measurement import PhotosynthesisMeasurement

class BenthicPhotosynthesisMeasurement(PhotosynthesisMeasurement):
    '''
    classdocs
    '''
    MAX_VALID_IK = 10.0 #Arbitrary value one order of magnitude greater than
    typical. According to Kalff's Limnology, page 333, Ik typically falls between
     0.14 and 0.72
    MIN_VALID_IK = 0.01 #Arbitrary value one order of magnitude less than typical
    . According to Kalff's Limnology, page 333, Ik typically falls between 0.14
    and 0.72
```

```
ik=0.0

def __init__(self, depth, pmax, ik):
    super(BenthicPhotosynthesisMeasurement, self).__init__(depth, pmax)
    self.set_ik(ik)

#GETTERS

def get_depth(self):
    return PhotosynthesisMeasurement.get_depth(self)



def get_pmax(self):
    return PhotosynthesisMeasurement.get_pmax(self)

def get_ik(self):
    return self.__ik

#SETTERS

def set_depth(self, value):
    #TODO: validate
    return PhotosynthesisMeasurement.set_depth(self, value)



def set_pmax(self, value):
    #TODO: validate
    return PhotosynthesisMeasurement.set_pmax(self, value)



def set_ik(self, value):
    #TODO: validate
    self.__ik = value



def del_ik(self):
    del self.__ik



def del_depth(self):
    return PhotosynthesisMeasurement.del_depth(self)
```

```
    def del_pmax(self):
        return PhotosynthesisMeasurement.del_pmax(self)




    #autogenerated by PyDev.
    ik = property(get_ik, set_ik, del_ik, "ik's docstring")






def main():
    print "hello world"





if __name__ == "__main__":
    main()
```

LISTING A.3: BenthicPhotosynthesisMeasurement Class

A.3.4 phytoplankton_photosynthesis_measurement.py

This code is for the Phytoplankton Photosynthesis Measurement object, which holds information relating to Phytoplankton photosynthesis at various depths.

```
'''
Created on Jun 17, 2015

@author: cdleong
'''
from photosynthesis_measurement import PhotosynthesisMeasurement

class PhytoPlanktonPhotosynthesisMeasurement(PhotosynthesisMeasurement):
    '''
    Holds information and methods relating to P/I curve for pytoplanktonic
    primary production, for a specific thermal layer.
```

```
'''
##########
#CONSTANTS
##########
MAX_VALID_THERMAL_LAYER = 3 #assumption: no more than three thermal layers
MIN_VALID_THERMAL_LAYER = 1
MAX_VALID_DEPTH = 2000 #lake Baikal, the deepest lake on earth, is only 1642m
. Adding a bit on that to be safe. Alternately we could go with Challenger
Deep, the deepest part of the ocean, which is about 11000 meters.
MIN_VALID_DEPTH = 0 #If you want to
MAX_VALID_PMAX = 5000 #arbitrary value an order of magnitude greater than any
 I've seen.
MIN_VALID_PMAX = 0
MAX_VALID_ALPHA = 100 #arbitrary. Biggest I've ever seen is less than 1.
MIN_VALID_ALPHA = 0.00001 #arbitrary value greater than zero. Smallest I've
seen is ~0.05
MAX_VALID_BETA = 100 #arbitrary. Biggest I've ever seen is less than 1.
MIN_VALID_BETA = 0.0 #if zero, we use the non-photoinhibition equation

##################
#VARIABLES
##################

thermal_layer = 1 #1 is epilimnion, 2 is metalimnion, 3 is hypolimnion
phyto_alpha = 0.0
phyto_beta = 0.0




def __init__(self, thermal_layer=0, depth=0.0, phyto_pmax_biomass=0.0,
phyto_alpha=0.0, phyto_beta=0.0):
    super(PhytoPlanktonPhotosynthesisMeasurement, self).__init__(depth,
phyto_pmax_biomass)
    self.set_thermal_layer(thermal_layer)
    self.set_phyto_alpha(phyto_alpha)
    self.set_phyto_beta(phyto_beta)




def get_thermal_layer(self):
    '''
```

```
    @return: the thermal layer with which these P/I curve parameters are
associated. 1=epilimnion, 2=metalimnion, 3=hypolimnion
    @rtype: int
    '''
    return self.__thermal_layer


def get_phyto_alpha(self):
    '''
    @return: P/I curve parameter alpha.
    @rtype: double
    '''
    return self.__phyto_alpha


def get_phyto_beta(self):
    '''
    @return: P/I curve parameter beta.
    @rtype: double
    '''
    return self.__phyto_beta


def set_thermal_layer(self, value):
    '''
    Sets thermal layer associated with which these P/I curve parameters are
associated. If given value outside min/max, sets to closest valid value.
    @param value: thermal layer. Valid values: 1=epilimnion, 2=metalimnion,
3=hypolimnion
    '''
    max_value = PhytoPlanktonPhotosynthesisMeasurement.
MAX_VALID_THERMAL_LAYER
    min_value = PhytoPlanktonPhotosynthesisMeasurement.
MIN_VALID_THERMAL_LAYER
    validated_value = self.validate_numerical_value(value, max_value,
min_value)
    if(validated_value == value):
        self.__thermal_layer = value
    else:
        raise Exception("PhytoPlanktonPhotosynthesisMeasurement thermal layer
 cannot be set to value outside of reasonable range: ", value,". Must be
within range ",min_value,":",  max_value, "")
```

```
def set_depth(self, value):
    '''
    Sets lower bound of thermal layer with which these P/I curve parameters
are associated. If given value outside min/max, sets to closest valid value.
    @param value: the lower bound of the thermal layer, in meters from the
surface.
    '''
    max_value = PhytoPlanktonPhotosynthesisMeasurement.MAX_VALID_DEPTH
    min_value = PhytoPlanktonPhotosynthesisMeasurement.MIN_VALID_DEPTH
    validated_value = self.validate_numerical_value(value, max_value,
min_value)
    if(validated_value == value):
        return PhotosynthesisMeasurement.set_depth(self, validated_value)
    else:
        raise Exception("PhytoPlanktonPhotosynthesisMeasurement depth cannot
be set to value outside of reasonable range: ", value,". Must be within range
",min_value,":", max_value, "")



def set_pmax(self, value):
    '''
    Sets P/I curve parameter pmax. If given value outside min/max, sets to
closest valid value.
    @param value: pmax
    '''
    max_value = PhytoPlanktonPhotosynthesisMeasurement.MAX_VALID_PMAX
    min_value = PhytoPlanktonPhotosynthesisMeasurement.MIN_VALID_PMAX
    validated_value = self.validate_numerical_value(value, max_value,
min_value)
    if(validated_value == value):
        return PhotosynthesisMeasurement.set_pmax(self, validated_value)
    else:
        raise Exception("PhytoPlanktonPhotosynthesisMeasurement pmax cannot
be set to value outside of reasonable range: ", value,". Must be within range
",min_value,":", max_value, "")
```

```
def set_phyto_alpha(self, value):
    '''
    Sets P/I curve parameter alpha. If given value outside min/max, sets to
closest valid value.
    @param value: alpha
    '''
    max_value = PhytoPlanktonPhotosynthesisMeasurement.MAX_VALID_ALPHA
    min_value = PhytoPlanktonPhotosynthesisMeasurement.MIN_VALID_ALPHA
    validated_value = self.validate_numerical_value(value, max_value,
min_value)
    if(validated_value == value):
        self.__phyto_alpha = value
    else:
        raise Exception("PhytoPlanktonPhotosynthesisMeasurement alpha cannot
be set to value outside of reasonable range: ", value,". Must be within range
 ",min_value,":",  max_value, "")



def set_phyto_beta(self, value):
    '''
    Sets P/I curve parameter alpha. If given value outside min/max, sets to
closest valid value.
    @param value: beta
    '''
    max_value = PhytoPlanktonPhotosynthesisMeasurement.MAX_VALID_BETA
    min_value = PhytoPlanktonPhotosynthesisMeasurement.MIN_VALID_BETA
    validated_value = self.validate_numerical_value(value, max_value,
min_value)
    if(validated_value == value):
        self.__phyto_beta = value
    else:
        raise Exception("PhytoPlanktonPhotosynthesisMeasurement beta cannot
be set to value outside of reasonable range: ", value,". Must be within range
 ",min_value,":",  max_value, "")




def del_thermal_layer(self):
    del self.__thermal_layer
```

```
def del_phyto_alpha(self):
    del self.__phyto_alpha


def del_phyto_beta(self):
    del self.__phyto_beta


def get_depth(self):
    return PhotosynthesisMeasurement.get_depth(self)


def get_pmax(self):
    return PhotosynthesisMeasurement.get_pmax(self)


def del_depth(self):
    return PhotosynthesisMeasurement.del_depth(self)


def del_pmax(self):
    return PhotosynthesisMeasurement.del_pmax(self)


thermal_layer = property(get_thermal_layer, set_thermal_layer,
del_thermal_layer, "thermal_layer's docstring")
phyto_alpha = property(get_phyto_alpha, set_phyto_alpha, del_phyto_alpha, "
phyto_alpha's docstring")
phyto_beta = property(get_phyto_beta, set_phyto_beta, del_phyto_beta, "
phyto_beta's docstring")
```

```
    #############################################
    #VALIDATORS
    #############################################
    def validate_numerical_value(self, value, max_value, min_value):
        '''
        Generic numerical validator.
        Checks if value is >max_value or <min_value.
        If it's outside the valid range it'll be set to the closest valid value.
        @param value: numerical value of some sort to be checked.
        @param max_value: numerical value. Max valid value.
        @param min_value: numerical value. Min valid value.
        @return: a valid value in the range (min_value,max_value), inclusive
        @rtype: numerical value
        '''
        validated_value = 0
        if(value < min_value):
            validated_value = min_value
        elif(value > max_value):
            validated_value = max_value
        else:
            validated_value = value
        return validated_value



def main():
    print "hello world"



if __name__ == "__main__":
    main()
```

LISTING A.4: PhytoPlanktonPhotosynthesisMeasurement Class

A.3.5 pond_shape.py

This code is for the generic Pond Shape object, which deals with information relating to the shape of the pond. The program is not expected to use this directly, but instead some subclass implementation of this.

Having a generic version allows flexibility in other parts of the program. Methods need not concern themselves, for example, with what form the pond shape data is in, only that the Pond Shape object has some method of calculating volume.

```
'''
Created on Jun 17, 2015

@author: cdleong
'''

class PondShape(object):
    '''
    abstract class. Nothing's really implemented.
    '''
    #######################################################
    #KNOWS... nothing. Depends on implementation.
    #######################################################



    def get_volume(self):
        '''
        @return: The volume of the lake, in m^3
        @rtype: double
        '''
        pass

    def get_max_depth(self):
        '''
        @return: get maximum depth, in meters
        @rtype: double
        '''
        pass

    def get_mean_depth(self):
        '''
        @return: get mean depth, in meters
        @rtype: double
        '''
        pass
```

```
    def get_water_surface_area_at_depth(self, depth =0.0):


        pass

    def get_sediment_area_at_depth(self, depth=0.0, depth_interval=0.1):
        pass

    def get_volume_above_depth(self, depth=0.0, depth_interval=0.1):
        pass

    def get_sediment_area_above_depth(self, depth=0.0):
        pass

    def get_fractional_sediment_area_at_depth(self, depth=0.0,
    total_sediment_area=0.0, depth_interval=0.1):
        pass

    #June 22: depth intervals aren't a thing now.

    def validate_depth(self, depth):
        pass

    def update_shape(self, other_pond_shape):
        pass



def main():
    print "hello world"



if __name__ == "__main__":
    main()
```

Listing A.5: PondShape Class

A.3.6 bathymetric_pond_shape.py

This code is for the Bathymetric Shape object, which deals with information relating to the shape of the pond. This implementation of the generic Pond_Shape object uses Bathymetry data in the form of depth/water surface area pairs.

```
'''
Created on Jun 18, 2015

@author: cdleong
'''
from pond_shape import PondShape
from scipy.interpolate import interp1d
from __builtin__ import str


class BathymetricPondShape(PondShape):
    '''
    This class stores information about the shape of a pond based on bathymetric
    measurements.
    In essence, this means that the area of the pond is given at various depths.
    Example: "At depth of 10 meters, water surface area is 6,000 square meters."

    For depths without a specified area, the class will interpolate using scipy.
    interpolate.interp1d,
    documented at http://docs.scipy.org/doc/scipy/reference/generated/scipy.
    interpolate.interp1d.html#scipy.interpolate.interp1d

    It will not work well if not given at least the area at the surface and at
    the bottom of the lake.
    '''

    #CONSTANTS
    DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS = 0.1  # ten centimeters, 0.1 meters.
     Arbitrary.


    # dict tutorial here: http://www.tutorialspoint.com/python/python_dictionary.
    htm
    water_surface_areas = {}  #Dictionary. Keys are depth values in meters,
    values are the surface area at that depth.
    #June 22 edit: depth intervals are now independent for every calculcation.
```

```
    def __init__(self, areas={}):
        '''
        Constructor
        @param areas: a python dict containing depth/area pairs.
        '''
        #make sure all the darn keys are FLOATS, NOT STRINGS
#         fixed_dict = self.convert_dict_keys_to_floats(areas)
        self.water_surface_areas = areas




    def get_dict(self):
        '''
        Getter method.
        @return: the dictionary of depth/area pairs.
        @rtype: python dict.
        '''
        return self.water_surface_areas

    def get_volume(self):
        '''
        Get Total Lake Volume
        @return:  the total volume of the entire lake, in cubic meters
        @rtype: float
        '''
        return self.get_volume_above_depth(self.get_max_depth())

    def addBathymetryLayer(self, depth_value=0.0, area_value=0.0):
        '''
        Adds a depth/area pair to the dictionary.
        @param depth_value: float value, depth_value in meters of area_value
    measurement.
        @param area_value:  float value, area_value in meters squared at
    depth_value.
        '''
        depth_value = float(depth_value)
        if (depth_value < 0.0 or area_value <= 0.0):
```

```
            raise Exception("invalid depth_value or area_value. Depth value must
NOT be less than zero. Depth Value given: ",depth_value, " Area must NOT be
less than, or equal to, zero. area_value given:",area_value)
        else:
            self.water_surface_areas[depth_value]=area_value



    def update_shape(self, other_pond_shape):
        '''
        Adds all depth/area pairs from other_pond_shape to this one.

        Does not check or validate anything. Currently expects another perfectly-
constructed BathymetricPondShape.
        @param  other_pond_shape:    another pond_shape object.
        '''
        #TODO: validate other data.
        #TODO: handle exceptions/problems.
        otherdict = other_pond_shape.water_surface_areas

        self.water_surface_areas.update(otherdict)

    def get_max_depth(self):
        '''
        Get Maximum Depth
        finds the maximum depth in the dict of areas.

        Throws exception if dictionary is empty.

        @return: in meters, the maximum depth of the lake.
        @rtype: float
        '''
        max_depth =0.0
        keys = self.water_surface_areas.keys()



        has_areas = bool(self.water_surface_areas) #evaluates to false if empty.

        if(False == has_areas):
            raise Exception("No shape data exists. max depth is 0")
        else:
            max_depth = max(keys)
```

```
    return float(max_depth)


def get_mean_depth(self, depth_interval=
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS):
    '''
    Get Mean Depth

    Calculates the mean depth.
    @param depth_interval: depth interval for calculations, in meters.
    @return: average depth, in meters, for the whole pond.
    @rtype: float
    '''
    validated_depth_interval = self.validate_depth_interval(depth_interval)
    max_depth = self.get_max_depth()
    total_area = self.get_sediment_area_above_depth(max_depth)
    if(0==total_area):
        #only possible if the sides are literally vertical.
        return max_depth


    current_depth = validated_depth_interval  # no point starting at 0, since
 that's just gonna be zero anyway.
    weighted_total = 0.0 #initialize to float
    while current_depth <= max_depth:
        area_at_depth = self.get_sediment_area_at_depth(current_depth) #
there are this many square meters at this depth
        weighted_total+=area_at_depth*current_depth
        current_depth += validated_depth_interval

    mean_depth  = weighted_total/total_area
    return mean_depth




def get_water_surface_area_at_depth(self, depth=0.0):
    '''
    Get Water Surface Area at Specified Depth
    Figures out the surface area at depth
```

```
    Uses http://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html
    For area/depth combinations not given.
    @param depth: depth in meters to calculate at. depth should between 0 and
 max_depth. It'll be set to one of those if not so.
    @return: the surface area of the water at  the specified depth, in m^2.
    @rtype: float
    '''
    #TODO: check and see if this still gives errors outside proper range

    validated_depth = self.validate_depth(depth)




    if(validated_depth in self.water_surface_areas):
        return self.water_surface_areas[validated_depth]







    # get interpolation function
    x = self.water_surface_areas.keys()
    y = self.water_surface_areas.values()
    if(len(x)<2):
        error_message = "Cannot interpolate to determine water surface area
at depth ", depth,", because there are not enough depth/area pairs."
        print error_message
        raise Exception(str(error_message))


    #make sure they are ordered by depth. interpolation requires it.
    xy = zip(x, y)
    xy.sort()

    x_sorted = []
    y_sorted = []
    for x_value, y_value in xy:
        x_sorted.append(x_value)
        y_sorted.append(y_value)
```

```
        f = interp1d(x_sorted, y_sorted)

        #interpolate

        water_surface_area_at_depth = f(validated_depth)

        return water_surface_area_at_depth




    def get_sediment_area_at_depth(self, depth=0.0, depth_interval=None):
        '''
        Get Sediment Area at Specified Depth
        Essentially, returns an estimate of the area of the section of lake
    bottom,
        whose bottom edge is at depth and top edge is at (depth-
    validated_depth_interval)
        On a perfectly conical lake this would form an inverted funnel shape.
        If given depth = max_depth and validated_depth_interval also = max_depth,
     should estimate sediment for the whole lake.
        ...which should add up to water surface area at depth 0 by the way

        @param depth: depth in meters to calculate at. depth should between 0 and
     max_depth. It'll be set to one of those if not so.
        @param depth_interval: depth interval for calculations, in meters.
        @return: sediment area: the area of the lake sediment in the interval
    between depth = depth and depth = depth-validated_depth_interval
        @rtype: float
        '''

        if(depth_interval is None):
            depth_interval = 1 #1 meter by default

        validated_depth = self.validate_depth(depth)
        validated_depth_interval = self.validate_depth_interval(depth_interval)
#           validated_depth_interval = self.get_depth_interval_meters()
```

```
    lower_edge_depth = validated_depth
    upper_edge_depth = self.validate_depth(validated_depth -
validated_depth_interval)


    #validate the depths of the two
    if(lower_edge_depth > upper_edge_depth):  # upper edge should be a
smaller value of depth
        # all is well. Do nothing.
        pass
    elif (lower_edge_depth < upper_edge_depth):
        # validated_depth_interval was negative?
        # switch them.
        lower_edge_depth, upper_edge_depth = upper_edge_depth,
lower_edge_depth
    else:  # they are the same
        #lower and upper bounds of sediment region are the same. Area is 0
        return 0


    upper_water_area = self.get_water_surface_area_at_depth(upper_edge_depth)

    lower_water_area = self.get_water_surface_area_at_depth(lower_edge_depth)



    # The theory is, we get basically the top side of a right cone/donut
thing.
    #
    # ASCII picture of a lake cross-section:
    #
    #          "sediment_area"
    #          |
    #  ________|_________
    #  |                 |
    #  \/                \/
    # _______________________      <--- upper_water_area
    # \   |              |   /
    #  \  |depth_interval|  /
    # h \ |              | /
    #    \|______________|/        <--- lower_water_area
    #
```

```
    # What we ACTUALLY want is h, but in practice the slope is generally
shallow enough in the littoral zone
    # (the zone we are interested in) that this is a good approximation.
Source: Dr. Vadeboncoeur
    sediment_area = upper_water_area - lower_water_area

    # it's _possible_, technically, that the lake gets *wider* as it goes
down. Is it?
    sediment_area = abs(sediment_area)


    return sediment_area




def get_volume_above_depth(self, depth=0.0, depth_interval=
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS):
    '''
    Get Volume Above Depth

    Calculates water volume above specified depth, in meters cubed.

    O(number of depth intervals*O(get_volume_at_depth()))
    @param depth: depth in meters to calculate at. depth should between 0 and
 max_depth. It'll be set to one of those if not so.
    @param depth_interval: depth interval for calculations, in meters.
    @return: volume above specified depth, in m^3
    @rtype: float

    '''

    validated_depth = self.validate_depth(depth)
    validated_depth_interval = self.validate_depth_interval(depth_interval)

    # just find the volume at each interval and add them all up.
    current_depth = 0.0  # no point starting at 0, since that's just gonna be
 zero anyway.
    total_volume = 0.0
    while current_depth <= validated_depth:
        current_volume = self.get_volume_at_depth(current_depth,
validated_depth_interval)
        total_volume += current_volume
```

```
        current_depth += validated_depth_interval
    return total_volume



def get_volume_at_depth(self, depth=0.0, depth_interval =
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS):
    '''
    Given a depth, gives the volume of the shape with a lower surface at area
 and upper surface at area-validated_depth_interval
    @param depth: depth in meters to calculate at. depth should between 0 and
 max_depth. It'll be set to one of those if not so.
    @return: volume, in m^3, between depth=depth, and depth = depth-
depth_interval.
    @rtype: float
    '''
    ###################################################
    # ASCII picture of an example lake cross-section:
    #
    # ________________________     <-areas[z0]
    # .\    |              |    /.
    # . \   |interval      |   / .
    # .  \ |               | /   .
    # ....\|______________|/....    <-areas[z1]
    #                         ^
    #                         |
    #                         |
    #                         x = 1/2 (areas[z0]-areas[z1])
    #
    # error is just 2x*interval, or just (areas[z0]-areas[z1])*interval
    # Volume from z0 to z1 can be approximated using
    # areas[z0]*interval, which overestimates by error    :result is
correctAnswer+error
    # areas[z1]*interval, which underestimates by error   :result is
correctAnswer-error
    # correct answer should be areas[z0]*interval -error or areas[z1]*
interval +error
    #
    ###################################################

    validated_depth = self.validate_depth(depth)
    validated_depth_interval = self.validate_depth_interval(depth_interval)
    lower_edge_depth = validated_depth
```

```
    upper_edge_depth = self.validate_depth(validated_depth -
validated_depth_interval)


    #validate the depths of the two
    if(lower_edge_depth > upper_edge_depth):  # upper edge should be a
smaller value of depth
        # all is well. Do nothing.
        pass
    elif (lower_edge_depth < upper_edge_depth):
        # validated_depth_interval was negative?
        lower_edge_depth, upper_edge_depth = upper_edge_depth,
lower_edge_depth   # switch them.
    else:
        # they are the same
        return 0.0

    upper_water_area = self.get_water_surface_area_at_depth(upper_edge_depth)
    lower_water_area = self.get_water_surface_area_at_depth(lower_edge_depth)

    upper_calculated_volume = upper_water_area*validated_depth_interval #
equivalent to correct answer + error
    lower_calculated_volume = lower_water_area*validated_depth_interval #
equivalent to correct answer - error


    volume_at_depth = (upper_calculated_volume+lower_calculated_volume)/2 #
equivalent to (correct answer)
    return volume_at_depth

def get_sediment_area_above_depth(self, depth=0.0, depth_interval=
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS):
    '''
    Get Sediment Area above Depth.

    Similar to get_sediment_area_at_depth, except that it calculates the area
 of the lake bottom sediment from the specified depth, all the way to the
surface.
    @param depth: depth in meters to calculate at. depth should between 0 and
 max_depth. It'll be set to one of those if not so.
    @return: the area of the sediment above a specific depth, in m^2.
    @rtype: float value
    '''
```

```
    validated_depth = self.validate_depth(depth)
    validated_depth_interval = self.validate_depth_interval(depth_interval)

    # add up the sediment area at every interval.
    total_area = 0.0

    current_depth = 0.0
    while current_depth <= validated_depth:
        current_area = self.get_sediment_area_at_depth(current_depth,
validated_depth_interval)
        total_area += current_area
        current_depth += validated_depth_interval


    return total_area

def get_fractional_sediment_area_at_depth(self, depth=0.0,
total_sediment_area=None, depth_interval =
DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS):
    '''
    Sediment area at depth, as a fraction of total_sediment_area.
    If total area isn't supplied, we go with total littoral area.

    For this to work with littoral area, pond object needs to supply it with
the proper littoral area, which must be calculated using the light
attenuation coefficient.
    @param depth:
    @param total_sediment area:
    @param depth_interval:
    @return:
    @rtype:
    '''
    if(total_sediment_area is None):
        total_sediment_area = self.get_sediment_area_above_depth(self.
get_max_depth())



    validated_depth = self.validate_depth(depth)
    sediment_area_at_depth = self.get_sediment_area_at_depth(validated_depth,
 depth_interval)
```

```
        fractional_sediment_area = sediment_area_at_depth/total_sediment_area
        return fractional_sediment_area



    def validate_depth(self, depth):
        '''
        Given a depth, checks to see if it is between 0 and max_depth.
        If outside that range, sets it to the closest one.
        @param depth: the value to be validated.
        @return: a float value between 0 and max depth of pond
        '''

        validated_depth = 0.0
        if(depth < 0):
            validated_depth=0.0
        elif(depth > self.get_max_depth()):
            validated_depth = self.get_max_depth()
        else:
            validated_depth=depth

        return validated_depth


    def validate_depth_interval(self, depth_interval):
        '''
        Checks to make sure that the depth_interval is between 0 and self.
get_max_depth() meters. If value is less than 0, it sets it to 1% of maximum
depth
        @param depth_interval:depth to validate, in meters.
        @return: a value between 0 and the maximum depth of the lake.
        @rtype: float
        '''
        max_depth= self.get_max_depth()
        validated_depth_interval = self.DEFAULT_DEPTH_INTERVAL_FOR_CALCULATIONS #
default value, chosen based on
        if(depth_interval<=0):
            validated_depth_interval=max_depth/100 #set to 1% of max depth. Seems
 safe enough.
        elif(depth_interval>max_depth):
            validated_depth_interval=max_depth
        else:
            validated_depth_interval = depth_interval
        return validated_depth_interval
```

```
    def add_bathymetry(self, otherObject):
        '''
        Given another BathymetricPondShape object, copies all entries in the
    other's dictionary of depth/area pairs into the dictionary of this one.
        @param otherObject:
        '''
        if(isinstance(otherObject, BathymetricPondShape)):
            thisDict = self.water_surface_areas
            otherdict = otherObject.water_surface_areas
            thisDict.update(otherdict)
            self.water_surface_areas=thisDict

def main():
    '''
    Used for testing!
    '''

    print "hello world"




if __name__ == "__main__":
    main()
```

Listing A.6: BathymetricPondShape Class

A.4 View Code (Python)

This section holds the source code for the View. Code in this section controls what the user sees. It is separate from the Model, which deals with the problem representation and equations, and the Controller, which commands and controls the model.

A.4.1 flask_app.py

This is the code that deals with the Flask framework, allowing the program to run as a web app.

```
import os


import traceback
from flask import Flask, request, url_for, render_template, redirect, Response,
    session, make_response
import StringIO
from data_reader import DataReader
import xlwt #excel writing. used for the excel output.
import sys
import mimetypes
from werkzeug.datastructures import Headers #used for exporting files
import jsonpickle #lets us transfer Pond object between views.

#for graphing
#we need to import matplotlib and set which renderer to use before we use pyplot.
     This allows it to work without a GUI installed on the OS.
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
import numpy as np

##############################################################
#IMPORTANT VARIABLES
#
##############################################################

#How to work with file uploads http://flask.pocoo.org/docs/0.10/patterns/
    fileuploads/
# This is the path to the upload directory



ALLOWED_EXTENSIONS = set(['xls', 'xlsx', 'csv'])
TEMPLATE_FILE = 'template.xls'
TEMPLATE_FILE_ROUTE = '/'+TEMPLATE_FILE
EXAMPLE_FILE = 'example_data.xls'
EXAMPLE_FILE_ROUTE = '/'+EXAMPLE_FILE
```

```python
INTERNAL_SERVER_ERROR_TEMPLATE_FILE = "500.html"
INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE = '/'+INTERNAL_SERVER_ERROR_TEMPLATE_FILE

FIRST_DATA_ROW_FOR_EXPORT = 1


#SESSION KEYS
PICKLED_POND_LIST_KEY = 'pickled_pond_list'


# Initialize the Flask application
app = Flask(__name__)

random_number = os.urandom(24)
app.secret_key = random_number

# These are the extension that we are accepting to be uploaded
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 #arbitrary 16 megabyte upload
     limit






def getPondList():
    #SALAMANDER
    pond_list = unpickle_pond_list()
    return pond_list







#used for making it possible to get numbers from python, and put them in HTML
#Got this from http://blog.bouni.de/blog/2013/04/24/call-functions-out-of-jinjs2-
    templates/
@app.context_processor
def my_utility_processor():

    def ponds():
```

```
        print "running ponds method"
        pond_list = getPondList()
        print "length of pond list: ", len(pond_list)
        return pond_list


    return dict(ponds=ponds)




@app.route('/', methods=['GET', 'POST'])
@app.route('/index', methods=['GET', 'POST'])
def indexView():
    '''
    Renders the template for the index.
    '''
#      if 'pond_pic_visible' not in session:
#          session['pond_pic_visible']='visible'



    #http://runnable.com/UiPcaBXaxGNYAAAL/how-to-upload-a-uploaded_file-to-the-
    server-in-flask-for-python
    if request.method == 'POST': #true if the button "upload" is clicked
        # Get the name of the uploaded uploaded_file
        uploaded_file = request.files['uploaded_file']




        # Check if the uploaded_file is one of the allowed types/extensions
        if uploaded_file and allowed_file(uploaded_file.filename):



            pond_file = request.files['uploaded_file']

            try:
```

```
            reader = DataReader("") #I don't plan on using this filename,
thanks
            pond_list = reader.readFile(pond_file.read()) #read method is
http://werkzeug.pocoo.org/docs/0.10/datastructures/#werkzeug.datastructures.
FileStorage,
        except Exception as e:
            print "error in getPondList"
            print str(e)
            return render_template(INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE,
error = str(e))







        ##################################################################
        #let's try something. AARDVARK <--easy to search for this
        #(this might be more work than making Pond objects serializable)
        ##################################################################
        ##trying http://jsonpickle.github.io/
        pickle_pond_list(pond_list)





        return redirect(url_for("primary_production"))

    else:
        error_message = "Apologies, that file extension is not allowed.
Please try one of the allowed extensions."
        return render_template('home_with_error.html', template_file_route =
TEMPLATE_FILE_ROUTE, example_file_route = EXAMPLE_FILE_ROUTE,error_message=
error_message)

return render_template('home.html', template_file_route = TEMPLATE_FILE_ROUTE
, example_file_route = EXAMPLE_FILE_ROUTE)
```

```
@app.route(TEMPLATE_FILE_ROUTE, methods=['GET', 'POST'])
def template():
    '''
    Used to offer template data file
    #http://stackoverflow.com/questions/20646822/how-to-serve-static-files-in-
    flask
    '''
    try:
        return app.send_static_file(TEMPLATE_FILE)
    except Exception as e:
        print str(e)
        return render_template(INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE, error = str(
    e))



@app.route(EXAMPLE_FILE_ROUTE, methods=['GET', 'POST'])
def example_file_view():
    '''
    Used to offer example data file
    #http://stackoverflow.com/questions/20646822/how-to-serve-static-files-in-
    flask
    '''
    try:
        return app.send_static_file(EXAMPLE_FILE)
    except Exception as e:
        print str(e)
        return render_template(INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE, error = str(
    e))

#################################################################
#renders the primary_production template.
#################################################################
@app.route('/primary_production', methods=['GET', 'POST'])
@app.route('/primary_production.html', methods=['GET', 'POST'])
def primary_production():
    '''
    Renders the primary_production template, which shows calculated values and a
    button to download them.
    '''
    print "primary_production view"
```

```
    try:
        return render_template("primary_production.html")
    except Exception as e:
        print str(e)
        return render_template(INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE, error = str(
    e))


#c.f. flask quickstart "variable rules"
# @app.route('/graph/<pond_key>/<int:layer>')
# @app.route('/graph')
@app.route('/graph/<pond_key>/<int:layer_index>')
def hourly_ppr_in_layer_graph(pond_key="", layer_index = 0):
    '''
    #TODO: comments
    '''
    #get the correct pond from the list in the session dict

    print "***************"
    print "pond_key is ", pond_key
    print "layer_index is ", layer_index

    try:
        pond = retrieve_pond(pond_key)
        times  = pond.get_list_of_times()
        ppr_values = pond.
    calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_thermal_layer
    (layer_index)
        x_values = times
        y_values = ppr_values
#         print "x values: ", x_values
#         print "x length: ", len(x_values)
#         print "y values: ", y_values
#         print "y length: ", len(y_values)



        x_label = "hour"
        y_label  = "PPPR (mgC*m^-3)"
        graph_title = "PPPR, ", pond.get_lake_id(), " layer ", layer_index+1
        return graph(x_values,y_values,x_label, y_label,graph_title)
    except:
        print "Unexpected error:", sys.exc_info()[0]
```

```
        #return error graphic
        #TODO: an error graphic
        return app.send_static_file('graph_error.png')





@app.route('/export')
def export_view():
    '''
    Code to make an excel file for download.
    Modified from...
    http://snipplr.com/view/69344/create-excel-file-with-xlwt-and-insert-in-flask
    -response-valid-for-jqueryfiledownload/
    '''
    print "export"
    #########################
    # Code for creating Flask
    # response
    #########################
    response = Response()
    response.status_code = 200


    ##################################
    # Code for creating Excel data and
    # inserting into Flask response
    ##################################

    #.... code here for adding worksheets and cells
    #Create a new workbook object
    workbook = xlwt.Workbook()

    #Add a sheet
    daily_worksheet = workbook.add_sheet('Daily Statistics')

    #columns to write to
    year_column = 0
```

```
lake_ID_column = year_column+1
day_of_year_column = lake_ID_column+1
bppr_column = day_of_year_column+1
pppr_column = bppr_column+1

#get data from session, write to daily_worksheet
#PLATYPUS
pond_list  = unpickle_pond_list()



year_list = []
lake_id_list = []
day_of_year_list = []
bpprList =[]
ppprList = []

for pond in pond_list:
    year = pond.get_year()
    lake_id = pond.get_lake_id()
    day_of_year = pond.get_day_of_year()
    bppr = pond.calculate_daily_whole_lake_benthic_primary_production_m2()
    pppr = pond.
calculate_daily_whole_lake_phytoplankton_primary_production_m2()

    year_list.append(year)
    lake_id_list.append(lake_id)
    day_of_year_list.append(day_of_year)
    bpprList.append(bppr)
    ppprList.append(pppr)



write_column_to_worksheet(daily_worksheet, year_column, "year", year_list)
write_column_to_worksheet(daily_worksheet, lake_ID_column, "Lake ID",
lake_id_list)
write_column_to_worksheet(daily_worksheet, day_of_year_column, "day of year",
 day_of_year_list)
write_column_to_worksheet(daily_worksheet, bppr_column, "bppr_m2", bpprList)
write_column_to_worksheet(daily_worksheet, pppr_column, "pppr_m2", ppprList)



#Add another sheet
hourly_worksheet = workbook.add_sheet('Hourly Statistics')
```

```
#columns
year_column = 0
lake_ID_column = year_column+1
day_of_year_column = lake_ID_column+1
layer_column = day_of_year_column+1
hour_column = layer_column+1
hourly_ppr_rates_column = hour_column+1


#lists
year_list = []
lake_id_list = []
day_of_year_list = []
layer_list = []
hour_list = []
hourly_ppr_rates_list = []
counter = 0
for pond in pond_list:
    year = pond.get_year()
    lake_id = pond.get_lake_id()
    day_of_year = pond.get_day_of_year()
    for layer in range (0, len(pond.get_thermal_layer_depths())):
        hourly_ppr_in_this_layer_list = []
        hourly_ppr_in_this_layer_list = pond.
calculate_hourly_phytoplankton_primary_production_rates_list_over_whole_day_in_thermal_layer
(layer)
        hour = 0.0
        time_interval = pond.get_time_interval()
        for hourly_ppr in hourly_ppr_in_this_layer_list:
            year_list.append(year)
            lake_id_list.append(lake_id)
            day_of_year_list.append(day_of_year)
            layer_list.append(layer)
            hour_list.append(hour)
            hourly_ppr_rates_list.append(hourly_ppr)
            hour+=time_interval
            counter+=1
            if(counter>10000):
                raise Exception("too big! The ouput is too big!!!")
                sys.exit()
                exit()
```

```
#write to columns
write_column_to_worksheet(hourly_worksheet, year_column, "year", year_list)
write_column_to_worksheet(hourly_worksheet, lake_ID_column, "lake",
lake_id_list)
write_column_to_worksheet(hourly_worksheet, day_of_year_column, "day",
day_of_year_list)
write_column_to_worksheet(hourly_worksheet, layer_column, "layer", layer_list
)
write_column_to_worksheet(hourly_worksheet, hour_column, "hour", hour_list)
write_column_to_worksheet(hourly_worksheet, hourly_ppr_rates_column, "ppr_m3"
, hourly_ppr_rates_list)


#This is the magic. The workbook is saved into the StringIO object,
#then that is passed to response for Flask to use.
output = StringIO.StringIO()
workbook.save(output)
response.data = output.getvalue()

################################
# Code for setting correct
# headers for jquery.fileDownload
#################################
filename = "export.xls"
mimetype_tuple = mimetypes.guess_type(filename)

#HTTP headers for forcing file download
response_headers = Headers({
        'Pragma': "public",  # required,
        'Expires': '0',
        'Cache-Control': 'must-revalidate, post-check=0, pre-check=0',
        'Cache-Control': 'private',  # required for certain browsers,
        'Content-Type': mimetype_tuple[0],
        'Content-Disposition': 'attachment; filename=\"%s\";' % filename,
        'Content-Transfer-Encoding': 'binary',
        'Content-Length': len(response.data)
    })

if not mimetype_tuple[1] is None:
    response.update({
            'Content-Encoding': mimetype_tuple[1]
        })
```

```python
    response.headers = response_headers

    #as per jquery.fileDownload.js requirements
    response.set_cookie('fileDownload', 'true', path='/')

    ################################
    # Return the response
    #################################
    return response




@app.errorhandler(413)
def request_entity_too_large(error):
    '''
    Error handler view. Should display when files that are too large are uploaded
    .
    '''
    return 'File Too Large'

@app.errorhandler(404)
def pageNotFound(error):

    return "Page not found"

@app.errorhandler(500)
def internalServerError(internal_exception):
    '''
    Prints internal program exceptions so they are visible by the user. Stopgap
    measure for usability.

    '''

    #TODO: more and better errors, so that when specific parts of the data are
    wrong, users can figure it out.
    traceback.print_exc()
    print str(internal_exception)
    return render_template(INTERNAL_SERVER_ERROR_TEMPLATE_ROUTE, error = str(
    internal_exception))
```

```
#HELPER METHODS
def write_column_to_worksheet(worksheet,column_number=0, column_header = "",
    values_list=[]):
    '''
    Prepends a column header and puts the data in values_list into worksheet at
    the specified column
    @param worksheet: An xlrd worksheet to write to.
    @param column_number: Column number to write to.
    @param column_header: Header to put at the top of the column.
    @param values_list: list of values to put in the column.
    '''
    print "writing column to worksheet"
    values_list.insert(0, column_header) #stick the column header at the front.
    numRows = len(values_list)


    for i in range(0, numRows):
        row = i
        column = column_number
        value=values_list[row]
        worksheet.write(row,column,value)



def retrieve_pond(pond_key = ""):

    #pickled pond list from session
    print "retrieve pond", pond_key
    pond_list = unpickle_pond_list()
    try:
        pond = next(pond for pond in pond_list if pond.get_key()==pond_key)
    except:
        raise Exception("Could not find pond")
    print "found pond"
    return pond




def unpickle_pond_list():

    pickled_ponds_list = session[PICKLED_POND_LIST_KEY]
    pond_list = []
```

```
    for pickled_pond in pickled_ponds_list:
        pond = jsonpickle.decode(pickled_pond, keys=True) #BEWARE! THIS TURNS ALL
     THE KEYS IN BATHYMETRIC POND SHAPE TO STRINGS
        pond_list.append(pond)

    return pond_list

def pickle_pond_list(pond_list = []):
    pickled_ponds_list = []
    for pond in pond_list:
        pickled_pond = jsonpickle.encode(pond,keys=True) #make it NOT SET THE
    KEYS TO STRINGS

        pickled_ponds_list.append(pickled_pond)

    session[PICKLED_POND_LIST_KEY] = pickled_ponds_list


def graph(x_vals=[],y_vals=[],x_label = "x label", y_label="y label", graph_title
     = "graph_title", graph_line_width=3):
    print "graphing"

#         #get arguments.
#     graph_type = request.args.get('graph_type')


    #make the figure
    fig = plt.figure()
#     fig = plt.Figure()



    #make the graph.
    f_subplot = fig.add_subplot(1, 1, 1) #http://stackoverflow.com/questions
    /3584805/in-matplotlib-what-does-111-means-in-fig-add-subplot111

     #setup y_vals-f_subplot
    if(len(x_vals)<2):
        x_vals=np.arange(0.0, 8.0, 0.01)

     #Setup x_vals-f_subplot
#     y_vals=[]
    if(len(y_vals)<2):
        y_vals = np.sin(2*np.pi*x_vals)
```

```
    #set labels and graph_title
    #fancy number formatting from http://stackoverflow.com/questions/21226868/
    superscript-in-python-plots
    f_subplot.set_xlabel(x_label)
    f_subplot.set_ylabel(y_label)
    f_subplot.set_title(graph_title)

    #plot
    f_subplot.plot(x_vals, y_vals, linewidth = graph_line_width)



    #package up the image and send it back. All of this replaces the ".show()"
    step.
    #figure to canvas. canvas with StringIO to png. png passed to make_response.
    canvas = FigureCanvas(fig)
    output = StringIO.StringIO()
    canvas.print_png(output)
    response = make_response(output.getvalue())
    response.mimetype = 'image/png'
    return response

# For a given file, return whether it's an allowed type or not
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS










if __name__ == '__main__':

    print "a random number is: ", random_number



    print "secret key is", app.secret_key
    debug_mode = False
    i_am_sure_i_want_to_let_people_execute_arbitrary_code = "no" #"yes" for yes.
```

```
    i_want_an_externally_visible_site = True
    if(debug_mode and "yes"==
    i_am_sure_i_want_to_let_people_execute_arbitrary_code):
        print "running in debug mode"
        app.run(debug=True)
        print "stopped running app"
    elif(i_want_an_externally_visible_site):
        app.run(host='0.0.0.0')
    else:
        app.run()
```

LISTING A.7: Flask app

A.5 Controller Code (Python)

In this section is the Controller section of the project. Code in this section controls or updates the model.

A.5.1 data_reader.py

This code reads in data from an Excel file in the format specified in our data template.

```
'''
Created on Mar 5, 2015

This class reads in data from an excel file, packages it up, and sends it to the
    model for processing.

@author: cdleong
'''
import xlrd, xlwt #reading and writing, respectively.
from pond import Pond
from numpy.distutils.npy_pkg_config import FormatError

from benthic_photosynthesis_measurement import BenthicPhotosynthesisMeasurement
from bathymetric_pond_shape import BathymetricPondShape
```

```
from phytoplankton_photosynthesis_measurement import
    PhytoPlanktonPhotosynthesisMeasurement
import sys



#Useful notes: http://www.youlikeprogramming.com/2012/03/examples-reading-excel-
    xls-documents-using-pythons-xlrd/

class DataReader(object):

    '''
    classdocs
    '''

    ##################################
    # Class variables
    ##################################

    # Data    Method    Source
    # pmax,    copied directly    https://drive.google.com/open?id=1
    jxqTExiqx5Y3rqf8Q3UBus5Rjr5ETVLCUy08Stey4w4
    # alpha,    copied directly    https://drive.google.com/open?id=1
    jxqTExiqx5Y3rqf8Q3UBus5Rjr5ETVLCUy08Stey4w4
    # beta    copied directly    https://drive.google.com/open?id=1
    jxqTExiqx5Y3rqf8Q3UBus5Rjr5ETVLCUy08Stey4w4
    # stratum number    copied directly    https://drive.google.com/open?id=1
    jxqTExiqx5Y3rqf8Q3UBus5Rjr5ETVLCUy08Stey4w4
    # stratum depth    calculated from the daily average of pp_epi_nhw_m2/
    pp_epi_nhw_m3,pp_met_nhw_m2/pp_met_nhw_m3, and pp_hyp_nhw_m2/pp_hyp_nhw_m3
    for the epilimnion, metalimnion, and hypolimnion respectively    https://lter
    .limnology.wisc.edu/dataset/north-temperate-lakes-lter-primary-production-
    trout-lake-area-1986-2007
    # light extinction coefficient    copied directly    https://lter.limnology.
    wisc.edu/dataset/north-temperate-lakes-lter-light-extinction-trout-lake-area
    -1981-current
    # pppr        https://lter.limnology.wisc.edu/dataset/north-temperate-lakes-
    lter-primary-production-trout-lake-area-1986-2007
#    filename = "example_data.xls" #Removed everything but one lake from Oct 16
    _test_data.
```

```
filename = "Sep_17_test_data.xls" #used for testing







#data starts at row 1. Row 0 is column headings
DEFAULT_COLUMN_HEADINGS_ROW = 0
DEFAULT_FIRST_DATA_ROW = 1
DEFAULT_NUMBER_OF_SHEETS = 4 #Pond, benthic, planktonic. Guide optional.






########################
#Worksheet Names/Indices
########################

#TODO: maybe some arrays? Or some more flexible solution anyway

#default indices.
POND_DATA_SHEET_INDEX =0
BENTHIC_PHOTO_DATA_SHEET_INDEX = 1
PHYTOPLANKTON_PHOTO_DATA_SHEET_INDEX = 2
SHAPE_DATA_SHEET_INDEX = 3

#default names
POND_DATA_SHEET_NAME = "pond_data"
BENTHIC_PHOTO_DATA_SHEET_NAME = "benthic_photo_data"
PHYTOPLANKTON_PHOTO_DATA_SHEET_NAME = "phytoplankton_photo_data"
SHAPE_DATA_SHEET_NAME = "shape_data"







#############
#Data Indices
#############
#TODO: a dict?
```

```
#TODO: some way for the user to specify all this on sheet 0, perhaps?

#indices common to all sheets
yearIndex = 0
dayOfYearIndex = yearIndex+1 #"DOY"
lakeIDIndex = dayOfYearIndex+1 #"Lake_ID"

#indices for Pond vars in pond_data worksheet
kd_index = lakeIDIndex+1 #index of light attenuation coefficient kd
noon_surface_light_index = kd_index+1 #"midday.mean.par"
length_of_day_index = noon_surface_light_index+1 #"LOD" in hours


#indices for vars in benthic_photo_data worksheet
benthic_light_penetration_proportion_index = lakeIDIndex+1
benthic_pmax_index = benthic_light_penetration_proportion_index+1 #"pmax.z"
benthic_ik_index = benthic_pmax_index+1 #"ik_z" light intensity at onset of
saturation

#indices for vars in phytoplankton_photo_data worksheet
phyto_thermal_layer_index =lakeIDIndex+1
phyto_depth_index = phyto_thermal_layer_index+1
phyto_pmax_index = phyto_depth_index+1
phyto_alpha_index = phyto_pmax_index+1
phyto_beta_index = phyto_alpha_index+1

#indices for vars in shape_data worksheet
shape_ID_index = 0
shape_depth_index = shape_ID_index+1 #"z" in meters #depth is in several
sheets #TODO: different variable?
shape_area_index = shape_depth_index+1 #"kat_div" in meters squared.


###############
#CONSTANTS
###############
DEFAULT_DEPTH_INTERVAL_PERCENTAGE=1
DEFAULT_TIME_INTERVAL = 0.25


def __init__(self, filename, testFlag=0):
    '''
    Constructor
```

```
        '''
        self.filename = filename









    #TODO: this should return nothing. Bad style. Or rename it.
    def read(self):
        try:
            book = xlrd.open_workbook(self.filename)
        except:
            raise Exception("error in read method. xlrd.open_workbook gave an
Exception with filename: ", self.filename)

        return self.read_pond_list_from_workbook(book)



    #TODO: redundant with read()
    def readFile(self,inputfile):
        '''
        READ FILE
        Given an inputFile object, opens the workbook and calls the function to
read the pond_list.
        '''
        #http://stackoverflow.com/questions/10458388/how-do-you-read-excel-files-
with-xlrd-on-appengine
        try:
            book =  xlrd.open_workbook(file_contents=inputfile)
        except IOError:
            raise Exception ("Error in readFile. xlrd.open_workbook(file_contents
=inputfile) gave exception with inputfile", inputfile)


        return self.read_pond_list_from_workbook(book)


    #reads all the pond data from the excel file.
```

```
def read_pond_list_from_workbook(self,book):
    '''
    READ POND LIST FROM WORKBOOK

    Opens the xlrd workbook and returns a list of Pond objects.
    @param book: an xlrd Workbook
    @return: list of Pond objects, storing the information in the workbook.
    @rtype: list
    '''
    ##############
    #Worksheets
    ##############
    nsheets = book.nsheets


    sheet_names = book.sheet_names()

    pond_data_workSheet = xlrd.book
    benthic_photo_data_workSheet= xlrd.book
    phytoplankton_photo_data_sheet= xlrd.book
    shape_data_sheet =xlrd.book



    if(nsheets<self.DEFAULT_NUMBER_OF_SHEETS): #Pond, benthic, planktonic.
Guide optional.
        raise IOError("file format incorrect. Number of sheets less than
expected")

    if(self.POND_DATA_SHEET_NAME in sheet_names and
       self.BENTHIC_PHOTO_DATA_SHEET_NAME in sheet_names and
       self.PHYTOPLANKTON_PHOTO_DATA_SHEET_NAME in sheet_names and
       self.SHAPE_DATA_SHEET_NAME in sheet_names):
        pond_data_workSheet = book.sheet_by_name(self.POND_DATA_SHEET_NAME)
        benthic_photo_data_workSheet = book.sheet_by_name(self.
BENTHIC_PHOTO_DATA_SHEET_NAME)
```

```
        phytoplankton_photo_data_sheet = book.sheet_by_name(self.
PHYTOPLANKTON_PHOTO_DATA_SHEET_NAME)
        shape_data_sheet = book.sheet_by_name(self.SHAPE_DATA_SHEET_NAME)
    else:
        #Standard sheet names not detected. Attempting to read using sheet
indices.
        pond_data_workSheet = book.sheet_by_index(self.POND_DATA_SHEET_INDEX)
        benthic_photo_data_workSheet = book.sheet_by_index(self.
BENTHIC_PHOTO_DATA_SHEET_INDEX)
        phytoplankton_photo_data_sheet = book.sheet_by_name(self.
PHYTOPLANKTON_PHOTO_DATA_SHEET_INDEX)
        shape_data_sheet = book.sheet_by_name(self.SHAPE_DATA_SHEET_INDEX)

    ##############
    #Rows, Columns
    ##############

    pond_data_workSheet_num_rows = pond_data_workSheet.nrows
    benthic_data_workSheet_num_rows = benthic_photo_data_workSheet.nrows
    phytoplankton_photo_data_sheet_num_rows = phytoplankton_photo_data_sheet.
nrows
    shape_data_sheet_num_rows = shape_data_sheet.nrows




    ###############################################
    #make all the objects!
    ###############################################
    pond_list = [] #list of pond objects. The same water body on a different
day counts as a separate "Pond"






    ##############################################
    #Make Pond objects from pond_data sheet
    ##############################################
    sheet = pond_data_workSheet
    num_rows  = pond_data_workSheet_num_rows #TODO: read until blank space
encountered might be better.
```

```
    curr_row = self.DEFAULT_FIRST_DATA_ROW #start at 1. row 0 is column
headings
    while curr_row<num_rows:
        row = sheet.row(curr_row)

        #values
        try:
            row_year_value = row[self.yearIndex].value
            row_doy_value = row[self.dayOfYearIndex].value
            row_lakeID_value = row[self.lakeIDIndex].value
            row_kd_value = float(row[self.kd_index].value)
            row_noonlight_value = float(row[self.noon_surface_light_index].
value)
            row_lod_value = float(row[self.length_of_day_index].value)
        except Exception as e:
            print str(e)
            print "Error: couldn't read values properly."




        #Do we need to make a pond object?
        pond = None
        pond = next((i for i in pond_list if (i.get_lake_id()==
row_lakeID_value and
                                             i.get_day_of_year()==
row_doy_value and
                                             i.get_year() == row_year_value))
,None) #source: http://stackoverflow.com/questions/7125467/find-object-in-
list-that-has-attribute-equal-to-some-value-that-meets-any-condi
        if pond is None: #not in list. Must create Pond object
            emptyShape = BathymetricPondShape({}) #initialize with empty dict
            pond = Pond(row_year_value, row_lakeID_value, row_doy_value,
row_lod_value, row_noonlight_value, row_kd_value, emptyShape, [], [], self.
DEFAULT_TIME_INTERVAL)
            pond_list.append(pond)
        curr_row+=1



    #######################################################
    #we made all the ponds. Time to add all the members
    #######################################################
```

```
        ##################################
        #Shape data from shape_data_sheet
        ##################################

        sheet = shape_data_sheet
        num_rows = shape_data_sheet_num_rows
        curr_row = self.DEFAULT_FIRST_DATA_ROW #start at 1. row 0 is column
    headings
        while curr_row<num_rows:
            row = sheet.row(curr_row)



            #values
            row_lakeID_value = row[self.shape_ID_index].value
            row_depth_value = float(row[self.shape_depth_index].value)
            row_area_value = float(row[self.shape_area_index].value)

            row_dict = {row_depth_value:row_area_value}
            row_shape = BathymetricPondShape(row_dict)

            #find the correct pond
            pond = None
#             pond = next((i for i in pond_list if (i.get_lake_id()==
    row_lakeID_value )),None) #source: http://stackoverflow.com/questions
    /7125467/find-object-in-list-that-has-attribute-equal-to-some-value-that-
    meets-any-condi
            #http://stackoverflow.com/questions/14366511/return-the-first-item-in
    -a-list-matching-a-condition
#             matchingPonds = filter(next((i for i in pond_list if (i.get_lake_id
    ()== row_lakeID_value )),None), pond_list)
            for pond in pond_list:
                if(pond.get_lake_id()==row_lakeID_value):
                    pond.update_shape(row_shape)                #add to Pond

            #increment while loop to next row
            curr_row+=1
```

```
    ###############
    #Benthic data
    ###############

    sheet = benthic_photo_data_workSheet
    num_rows = benthic_data_workSheet_num_rows
    curr_row = self.DEFAULT_FIRST_DATA_ROW #start at 1. row 0 is column
headings
    while curr_row<num_rows:
        row = sheet.row(curr_row)



        #values
        row_year_value = row[self.yearIndex].value
        row_doy_value = row[self.dayOfYearIndex].value
        row_lakeID_value = row[self.lakeIDIndex].value
        row_light_penetration_proportion_value = float(row[self.
benthic_light_penetration_proportion_index].value)
        row_pmax_value = float(row[self.benthic_pmax_index].value)
        row_ik_value = float(row[self.benthic_ik_index].value)

        #find the correct pond
        pond = None
        pond = next((i for i in pond_list if (i.get_lake_id()==
row_lakeID_value and
                                             i.get_day_of_year()==
row_doy_value and
                                             i.get_year() == row_year_value))
,None) #source: http://stackoverflow.com/questions/7125467/find-object-in-
list-that-has-attribute-equal-to-some-value-that-meets-any-condi
        if pond is None: #something is terribly wrong
            raise FormatError("Something went wrong. Benthic Measurement with
 DOY "+str(row_doy_value) + " and Lake ID " + row_lakeID_value + " does not
match to any Pond.")
            #TODO: handle this better.
        else:
            #create PhotoSynthesisMeasurement object using values specific to
 that benthic_measurement/row
```

```
            row_depth_value = pond.
calculate_depth_of_specific_light_percentage(
row_light_penetration_proportion_value) #convert from light proportions to
depth in meters.


            benthic_measurement = BenthicPhotosynthesisMeasurement(
row_depth_value, row_pmax_value, row_ik_value)
            pond.add_benthic_measurement_if_photic(benthic_measurement)
            #add to Pond

        curr_row+=1
    #end of while loop


    ###############
    #Phyto data
    ###############
    sheet = phytoplankton_photo_data_sheet
    num_rows = phytoplankton_photo_data_sheet_num_rows
    curr_row = self.DEFAULT_FIRST_DATA_ROW #start at 1. row 0 is column
headings
    while curr_row<num_rows:
        row = sheet.row(curr_row)


        #values
        row_year_value = row[self.yearIndex].value
        row_doy_value = row[self.dayOfYearIndex].value
        row_lakeID_value = row[self.lakeIDIndex].value
        row_thermal_layer_value = row[self.phyto_thermal_layer_index].value
        row_depth_value = row[self.phyto_depth_index].value
        row_phyto_pmax_value = row[self.phyto_pmax_index].value
        row_alpha_value = row[self.phyto_alpha_index].value
        row_beta_value = row[self.phyto_beta_index].value


        #find the correct pond
        pond = None
        pond = next((i for i in pond_list if (i.get_lake_id()==
row_lakeID_value and
                                              i.get_day_of_year()==
row_doy_value and
```

```
                                        i.get_year() == row_year_value))
,None) #source: http://stackoverflow.com/questions/7125467/find-object-in-
list-that-has-attribute-equal-to-some-value-that-meets-any-condi
        if pond is None: #something is terribly wrong
            raise FormatError("Something went wrong. Benthic Measurement with
 DOY "+str(row_doy_value) + " and Lake ID " + row_lakeID_value + " does not
match to any Pond.")
        else:
            #create PhotoSynthesisMeasurement object using values specific to
 that benthic_measurement/row

            phyto_measurement = PhytoPlanktonPhotosynthesisMeasurement(
row_thermal_layer_value, row_depth_value, row_phyto_pmax_value,
row_alpha_value, row_beta_value)
            pond.add_phytoplankton_measurement(phyto_measurement)
            #add to Pond

        curr_row+=1




    return pond_list

#END OF read_pond_list_from_workbook METHOD




def write(self, filename="output.xls"):
    '''
    Write to file
    @param filename: the name of the output file.
    '''

    #TODO:return whether it was successful.
    #Create a new workbook object
```

```
        workbook = xlwt.Workbook()

        #Add a sheet
        worksheet = workbook.add_sheet('Statistics')

        #Add some values
        for x in range(0, 10):
            for y in range(0,10):
                worksheet.write(x,y,x*y)

        workbook.save(filename)

'''
let us test things
'''
def main():
    print "hello world"
    sys.exit()




if __name__ == "__main__":
    main()
```

Listing A.8: DataReader class

Program validation process

B.1 Data sources

- "pprinputs_Colin.xlsx", received from Dr. Vadeboncoeur: α, P_{max}, and β for each layer, for each pond, for each day.
- NTL LTER Light Extinction database: light extinction coefficient
- North Temperate Lakes LTER: Primary Production - Trout Lake Area 1986 - 2007 database: day length, noon light, thermal layer depths. Also, this database contains the output values against which we compared the program output.

The actual lakes in question are those studied as a part of the *Northern Temperate Lakes* project, whose website can be found here. Specifically, validation was done on data from Sparkling, Trout, Crystal, and Big Muskogee lakes.

Bathymetry was taken from the Northern Temperate Lakes project site, specifically from this collection.

B.2 Validating benthic primary production

Validation of benthic primary production was primarily a matter of comparing the implementation results with those calculated by Drs. Devlin and Vadeboncoeur.

B.2.1 Data used for testing BPPR.

The input data used for validating Benthic Primary Production was received from Dr. Vadeboncoeur (Table B.1).

TABLE B.1: pprinputs_Colin.xlsx

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	0	14.7563703653	404.943
160	US_SPARK	0.1	14.7563703653	393.6482954318
160	US_SPARK	0.2	14.7563703653	382.73440925
160	US_SPARK	0.3	14.7563703653	372.1885015754
160	US_SPARK	0.4	14.7563703653	361.9981654449
160	US_SPARK	0.5	14.7563703653	352.1514122156
160	US_SPARK	0.6	15.3070925213	342.63665746
160	US_SPARK	0.7	16.8567990535	333.4427073373
160	US_SPARK	0.8	19.2518000578	324.5587454245
160	US_SPARK	0.9	22.3384056302	315.9743199911
160	US_SPARK	1	25.9629258666	307.6793317025
160	US_SPARK	1.1	29.9716708631	299.664021739
160	US_SPARK	1.2	34.2109507157	291.9189603145
160	US_SPARK	1.3	38.5270755202	284.4350355825
160	US_SPARK	1.4	42.7663553727	277.2034429168
160	US_SPARK	1.5	46.7751003692	270.2156745524
160	US_SPARK	1.6	50.3996206057	263.463509577
160	US_SPARK	1.7	53.4862261781	256.9390042588
160	US_SPARK	1.8	55.8812271824	250.6344827015
160	US_SPARK	1.9	57.4309337145	244.5425278131
160	US_SPARK	2	57.9816558706	238.6559725805
160	US_SPARK	2.1	57.8275306141	232.9678916375

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	2.2	57.3864135006	227.4715931174
160	US_SPARK	2.3	56.6901925141	222.1606107798
160	US_SPARK	2.4	55.770755639	217.0286964038
160	US_SPARK	2.5	54.6599908591	212.069812437
160	US_SPARK	2.6	53.3897861587	207.2781248921
160	US_SPARK	2.7	51.9920295219	202.647996484
160	US_SPARK	2.8	50.4986089327	198.173979997
160	US_SPARK	2.9	48.9414123753	193.850811877
160	US_SPARK	3	47.3523278338	189.6734060386
160	US_SPARK	3.1	45.7632432923	185.6368478817
160	US_SPARK	3.2	44.2060467349	181.7363885093
160	US_SPARK	3.3	42.7126261457	177.9674391411
160	US_SPARK	3.4	41.3148695089	174.3255657144
160	US_SPARK	3.5	40.0446648085	170.8064836679
160	US_SPARK	3.6	38.9339000287	167.4060529007
160	US_SPARK	3.7	38.0144631535	164.1202729019
160	US_SPARK	3.8	37.3182421671	160.9452780441
160	US_SPARK	3.9	36.8771250535	157.8773330353
160	US_SPARK	4	36.722999797	154.9128285247
160	US_SPARK	4.1	36.6865912661	152.0482768564
160	US_SPARK	4.2	36.5798340485	149.2803079662
160	US_SPARK	4.3	36.4064307065	146.6056654166
160	US_SPARK	4.4	36.1700838027	144.0212025663
160	US_SPARK	4.5	35.8744958995	141.5238788674
160	US_SPARK	4.6	35.5233695593	139.1107562891
160	US_SPARK	4.7	35.1204073447	136.7789958608
160	US_SPARK	4.8	34.6693118181	134.525854332

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	4.9	34.173785542	132.3486809453
160	US_SPARK	5	33.6375310787	130.2449143176
160	US_SPARK	5.1	33.0642509909	128.2120794268
160	US_SPARK	5.2	32.4576478409	126.2477847
160	US_SPARK	5.3	31.8214241911	124.3497191997
160	US_SPARK	5.4	31.1592826042	122.5156499056
160	US_SPARK	5.5	30.4749256425	120.7434190867
160	US_SPARK	5.6	29.7720558685	119.0309417636
160	US_SPARK	5.7	29.0543758446	117.3762032549
160	US_SPARK	5.8	28.3255881333	115.7772568074
160	US_SPARK	5.9	27.5893952971	114.2322213057
160	US_SPARK	6	26.8494998985	112.7392790592
160	US_SPARK	6.1	26.1096044999	111.2966736634
160	US_SPARK	6.2	25.3734116637	109.9027079337
160	US_SPARK	6.3	24.6446239524	108.5557419087
160	US_SPARK	6.4	23.9269439285	107.254190921
160	US_SPARK	6.5	23.2240741545	105.9965237324
160	US_SPARK	6.6	22.5397171928	104.7812607331
160	US_SPARK	6.7	21.8775756059	103.6069722003
160	US_SPARK	6.8	21.2413519562	102.4722766167
160	US_SPARK	6.9	20.6347488061	101.3758390449
160	US_SPARK	7	20.0614687183	100.3163695571
160	US_SPARK	7.1	19.525214255	99.2926217174
160	US_SPARK	7.2	19.0296879789	98.3033911151
160	US_SPARK	7.3	18.5785924523	97.3475139485
160	US_SPARK	7.4	18.1756302377	96.423865655
160	US_SPARK	7.5	17.8245038975	95.5313595885

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	7.6	17.5289159943	94.6689457406
160	US_SPARK	7.7	17.2925690905	93.8356095059
160	US_SPARK	7.8	17.1191657485	93.0303704878
160	US_SPARK	7.9	17.0124085309	92.2522813452
160	US_SPARK	8	16.976	91.5004266783
160	US_SPARK	8.1	16.9589202065	90.7739219511
160	US_SPARK	8.2	16.9085305276	90.0719124515
160	US_SPARK	8.3	16.826105516	89.393572285
160	US_SPARK	8.4	16.7129197243	88.7381034036
160	US_SPARK	8.5	16.5702477051	88.1047346665
160	US_SPARK	8.6	16.399364011	87.4927209334
160	US_SPARK	8.7	16.2015431948	86.9013421873
160	US_SPARK	8.8	15.9780598089	86.3299026878
160	US_SPARK	8.9	15.730188406	85.7777301525
160	US_SPARK	9	15.4592035388	85.244174966
160	US_SPARK	9.1	15.1663797598	84.7286094155
160	US_SPARK	9.2	14.8529916217	84.2304269529
160	US_SPARK	9.3	14.5203136771	83.7490414804
160	US_SPARK	9.4	14.1696204786	83.2838866616
160	US_SPARK	9.5	13.8021865789	82.8344152551
160	US_SPARK	9.6	13.4192865305	82.4000984703
160	US_SPARK	9.7	13.0221948861	81.9804253459
160	US_SPARK	9.8	12.6121861983	81.5749021482
160	US_SPARK	9.9	12.1905350197	81.1830517906
160	US_SPARK	10	11.758515903	80.8044132723
160	US_SPARK	10.1	11.3174034007	80.4385411356
160	US_SPARK	10.2	10.8684720656	80.0850049424

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	10.3	10.4129964501	79.7433887673
160	US_SPARK	10.4	9.952251107	79.4132907084
160	US_SPARK	10.5	9.4875105889	79.0943224148
160	US_SPARK	10.6	9.0200494483	78.7861086292
160	US_SPARK	10.7	8.5511422379	78.4882867468
160	US_SPARK	10.8	8.0820635103	78.2005063885
160	US_SPARK	10.9	7.6140878182	77.922428989
160	US_SPARK	11	7.1484897142	77.6537273979
160	US_SPARK	11.1	6.6865437508	77.3940854954
160	US_SPARK	11.2	6.2295244808	77.1431978201
160	US_SPARK	11.3	5.7787064567	76.9007692098
160	US_SPARK	11.4	5.3353642311	76.666514454
160	US_SPARK	11.5	4.9007723567	76.4401579587
160	US_SPARK	11.6	4.4762053861	76.2214334217
160	US_SPARK	11.7	4.0629378719	76.0100835199
160	US_SPARK	11.8	3.6622443668	75.8058596061
160	US_SPARK	11.9	3.2753994233	75.6085214165
160	US_SPARK	12	2.9036775941	75.4178367884
160	US_SPARK	12.1	2.5483534318	75.2335813866
160	US_SPARK	12.2	2.210701489	75.0555384399
160	US_SPARK	12.3	1.8919963184	74.8834984858
160	US_SPARK	12.4	1.5935124725	74.7172591241
160	US_SPARK	12.5	1.316524504	74.5566247789
160	US_SPARK	12.6	1.0623069656	74.4014064685
160	US_SPARK	12.7	0.8321344097	74.2514215828
160	US_SPARK	12.8	0.6272813892	74.1064936688
160	US_SPARK	12.9	0.4490224565	73.9664522231

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
160	US_SPARK	13	0.2986321643	73.8311324907
160	US_SPARK	13.1	0.1773850652	73.7003752718
160	US_SPARK	13.2	0.0865557119	73.5740267342
160	US_SPARK	13.3	0.0274186569	73.4519382325
160	US_SPARK	13.4	0.0012484529	73.3339661329
164	US_CRYST	0	19.1303546145	404.943
164	US_CRYST	0.1	19.1303546145	394.0860610853
164	US_CRYST	0.2	19.1303546145	383.5809927417
164	US_CRYST	0.3	19.1303546145	373.4163909353
164	US_CRYST	0.4	19.1303546145	363.581221234
164	US_CRYST	0.5	19.1303546145	354.0648068289
164	US_CRYST	0.6	19.1303546145	344.8568169436
164	US_CRYST	0.7	19.1303546145	335.9472556194
164	US_CRYST	0.8	19.1303546145	327.3264508643
164	US_CRYST	0.9	19.1303546145	318.9850441528
164	US_CRYST	1	19.1303546145	310.9139802668
164	US_CRYST	1.1	19.1303546145	303.1044974656
164	US_CRYST	1.2	19.1303546145	295.5481179741
164	US_CRYST	1.3	19.1303546145	288.2366387798
164	US_CRYST	1.4	19.1303546145	281.1621227277
164	US_CRYST	1.5	19.1303546145	274.3168899038
164	US_CRYST	1.6	19.1303546145	267.6935092982
164	US_CRYST	1.7	19.1303546145	261.2847907382
164	US_CRYST	1.8	19.1303546145	255.0837770826
164	US_CRYST	1.9	19.1303546145	249.0837366695
164	US_CRYST	2	19.1303546145	243.2781560081
164	US_CRYST	2.1	19.195871963	237.6607327084

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_CRYST	2.2	19.3833871327	232.2253686392
164	US_CRYST	2.3	19.6793448102	226.9661633081
164	US_CRYST	2.4	20.070189682	221.8774074562
164	US_CRYST	2.5	20.5423664346	216.95357686
164	US_CRYST	2.6	21.0823197546	212.189326335
164	US_CRYST	2.7	21.6764943285	207.5794839323
164	US_CRYST	2.8	22.3113348428	203.1190453249
164	US_CRYST	2.9	22.9732859841	198.8031683745
164	US_CRYST	3	23.6487924388	194.6271678754
164	US_CRYST	3.1	24.3242988935	190.5865104682
164	US_CRYST	3.2	24.9862500348	186.6768097182
164	US_CRYST	3.3	25.6210905491	182.8938213544
164	US_CRYST	3.4	26.215265123	179.2334386611
164	US_CRYST	3.5	26.755218443	175.6916880205
164	US_CRYST	3.6	27.2273951956	172.2647245984
164	US_CRYST	3.7	27.6182400674	168.948828171
164	US_CRYST	3.8	27.9141977449	165.7403990858
164	US_CRYST	3.9	28.1017129146	162.6359543541
164	US_CRYST	4	28.167230263	159.63212387
164	US_CRYST	4.1	28.1305558072	156.725646752
164	US_CRYST	4.2	28.0230188436	153.9133678026
164	US_CRYST	4.3	27.8483489779	151.1922340837
164	US_CRYST	4.4	27.6102758157	148.5592916021
164	US_CRYST	4.5	27.3125289627	146.0116821026
164	US_CRYST	4.6	26.9588380246	143.5466399654
164	US_CRYST	4.7	26.552932607	141.1614892039
164	US_CRYST	4.8	26.0985423157	138.8536405592

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_CRYST	4.9	25.5993967563	136.6205886898
164	US_CRYST	5	25.0592255344	134.4599094516
164	US_CRYST	5.1	24.4817582559	132.3692572665
164	US_CRYST	5.2	23.8707245262	130.3463625758
164	US_CRYST	5.3	23.2298539512	128.3890293768
164	US_CRYST	5.4	22.5628761364	126.4951328385
164	US_CRYST	5.5	21.8735206876	124.6626169952
164	US_CRYST	5.6	21.1655172104	122.8894925146
164	US_CRYST	5.7	20.4425953106	121.173834538
164	US_CRYST	5.8	19.7084845937	119.5137805909
164	US_CRYST	5.9	18.9669146654	117.9075285612
164	US_CRYST	6	18.2216151315	116.3533347426
164	US_CRYST	6.1	17.4763155976	114.8495119422
164	US_CRYST	6.2	16.7347456694	113.3944276482
164	US_CRYST	6.3	16.0006349525	111.9865022584
164	US_CRYST	6.4	15.2777130526	110.6242073648
164	US_CRYST	6.5	14.5697095754	109.306064095
164	US_CRYST	6.6	13.8803541266	108.0306415063
164	US_CRYST	6.7	13.2133763119	106.7965550326
164	US_CRYST	6.8	12.5725057368	105.602464981
164	US_CRYST	6.9	11.9614720072	104.4470750779
164	US_CRYST	7	11.3840047286	103.3291310615
164	US_CRYST	7.1	10.8438335068	102.2474193204
164	US_CRYST	7.2	10.3446879474	101.200765576
164	US_CRYST	7.3	9.890297656	100.1880336076
164	US_CRYST	7.4	9.4843922385	99.2081240194
164	US_CRYST	7.5	9.1307013004	98.2599730465

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_CRYST	7.6	8.8329544474	97.3425514004
164	US_CRYST	7.7	8.5948812852	96.4548631516
164	US_CRYST	7.8	8.4202114194	95.5959446485
164	US_CRYST	7.9	8.3126744558	94.764863471
164	US_CRYST	8	8.276	93.9607174187
164	US_CRYST	8.1	8.2691292569	93.1826335311
164	US_CRYST	8.2	8.2488269903	92.4297671403
164	US_CRYST	8.3	8.2155581444	91.7013009537
164	US_CRYST	8.4	8.1697876633	90.9964441671
164	US_CRYST	8.5	8.111980491	90.3144316059
164	US_CRYST	8.6	8.0426015718	89.6545228947
164	US_CRYST	8.7	7.9621158497	89.0160016535
164	US_CRYST	8.8	7.8709882688	88.3981747199
164	US_CRYST	8.9	7.7696837733	87.8003713968
164	US_CRYST	9	7.6586673073	87.2219427242
164	US_CRYST	9.1	7.5384038149	86.6622607747
164	US_CRYST	9.2	7.4093582401	86.1207179718
164	US_CRYST	9.3	7.2719955273	85.5967264305
164	US_CRYST	9.4	7.1267806203	85.089717319
164	US_CRYST	9.5	6.9741784634	84.5991402412
164	US_CRYST	9.6	6.8146540007	84.1244626391
164	US_CRYST	9.7	6.6486721763	83.6651692148
164	US_CRYST	9.8	6.4766979344	83.2207613711
164	US_CRYST	9.9	6.2991962189	82.7907566702
164	US_CRYST	10	6.1166319741	82.3746883101
164	US_CRYST	10.1	5.9294701441	81.9721046174
164	US_CRYST	10.2	5.738175673	81.5825685577

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_CRYST	10.3	5.5432135048	81.2056572606
164	US_CRYST	10.4	5.3450485838	80.8409615607
164	US_CRYST	10.5	5.144145854	80.4880855537
164	US_CRYST	10.6	4.9409702596	80.1466461665
164	US_CRYST	10.7	4.7359867447	79.8162727412
164	US_CRYST	10.8	4.5296602533	79.4966066328
164	US_CRYST	10.9	4.3224557297	79.18730082
164	US_CRYST	11	4.1148381179	78.8880195282
164	US_CRYST	11.1	3.907272362	78.5984378654
164	US_CRYST	11.2	3.7002234062	78.3182414692
164	US_CRYST	11.3	3.4941561945	78.0471261653
164	US_CRYST	11.4	3.2895356712	77.784797638
164	US_CRYST	11.5	3.0868267803	77.5309711102
164	US_CRYST	11.6	2.8864944659	77.2853710343
164	US_CRYST	11.7	2.6890036721	77.047730793
164	US_CRYST	11.8	2.4948193431	76.8177924103
164	US_CRYST	11.9	2.304406423	76.5953062708
164	US_CRYST	12	2.1182298559	76.3800308494
164	US_CRYST	12.1	1.9367545859	76.1717324484
164	US_CRYST	12.2	1.7604455571	75.9701849446
164	US_CRYST	12.3	1.5897677137	75.775169543
164	US_CRYST	12.4	1.4251859998	75.5864745399
164	US_CRYST	12.5	1.2671653595	75.4038950929
164	US_CRYST	12.6	1.1161707368	75.2272329983
164	US_CRYST	12.7	0.972667076	75.0562964764
164	US_CRYST	12.8	0.8371193211	74.8908999627
164	US_CRYST	12.9	0.7099924163	74.7308639071

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_CRYST	13	0.5917513056	74.5760145785
164	US_CRYST	13.1	0.4828609333	74.4261838765
164	US_CRYST	13.2	0.3837862433	74.2812091486
164	US_CRYST	13.3	0.2949921799	74.140933014
164	US_CRYST	13.4	0.2169436871	74.0052031924
164	US_CRYST	13.5	0.150105709	73.8738723392
164	US_CRYST	13.6	0.0949431899	73.7467978848
164	US_CRYST	13.7	0.0519210737	73.6238418805
164	US_CRYST	13.8	0.0215043047	73.5048708483
164	US_CRYST	13.9	0.0041578269	73.3897556364
164	US_TROUT	0	35.1534649462	404.943
164	US_TROUT	0.1	35.1534649462	394.38320353
164	US_TROUT	0.2	35.1534649462	384.156280583
164	US_TROUT	0.3	35.1534649462	374.2517380797
164	US_TROUT	0.4	35.1534649462	364.6594137112
164	US_TROUT	0.5	35.1534649462	355.3694655122
164	US_TROUT	0.6	35.1534649462	346.3723617628
164	US_TROUT	0.7	35.1534649462	337.6588712091
164	US_TROUT	0.8	35.1534649462	329.2200535913
164	US_TROUT	0.9	35.1534649462	321.0472504709
164	US_TROUT	1	35.1534649462	313.132076347
164	US_TROUT	1.1	35.1534649462	305.4664100525
164	US_TROUT	1.2	35.1534649462	298.0423864216
164	US_TROUT	1.3	35.1534649462	290.8523882199
164	US_TROUT	1.4	35.1534649462	283.889038329
164	US_TROUT	1.5	35.1534649462	277.1451921773
164	US_TROUT	1.6	35.1534649462	270.6139304096

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_TROUT	1.7	35.1534649462	264.2885517875
164	US_TROUT	1.8	35.1534649462	258.162566314
164	US_TROUT	1.9	35.1534649462	252.2296885744
164	US_TROUT	2	35.1534649462	246.4838312874
164	US_TROUT	2.1	35.1293724828	240.9190990593
164	US_TROUT	2.2	35.0581778999	235.5297823354
164	US_TROUT	2.3	34.9415054086	230.3103515416
164	US_TROUT	2.4	34.7809792199	225.255451411
164	US_TROUT	2.5	34.5782235448	220.3598954893
164	US_TROUT	2.6	34.3348625943	215.6186608135
164	US_TROUT	2.7	34.0525205795	211.0268827579
164	US_TROUT	2.8	33.7328217112	206.579850043
164	US_TROUT	2.9	33.3773902007	202.2729999018
164	US_TROUT	3	32.9878502588	198.1019133981
164	US_TROUT	3.1	32.5658260966	194.0623108925
164	US_TROUT	3.2	32.1129419251	190.1500476517
164	US_TROUT	3.3	31.6308219553	186.3611095953
164	US_TROUT	3.4	31.1210903983	182.691609178
164	US_TROUT	3.5	30.585371465	179.1377814004
164	US_TROUT	3.6	30.0252893665	175.695979946
164	US_TROUT	3.7	29.4424683137	172.3626734402
164	US_TROUT	3.8	28.8385325178	169.1344418271
164	US_TROUT	3.9	28.2151061897	166.0079728599
164	US_TROUT	4	27.5738135404	162.9800587033
164	US_TROUT	4.1	26.9162787809	160.0475926413
164	US_TROUT	4.2	26.2441261223	157.2075658902
164	US_TROUT	4.3	25.5589797756	154.4570645114

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_TROUT	4.4	24.8624639518	151.7932664215
164	US_TROUT	4.5	24.1562028619	149.2134384969
164	US_TROUT	4.6	23.4418207169	146.7149337695
164	US_TROUT	4.7	22.7209417278	144.2951887107
164	US_TROUT	4.8	21.9951901057	141.9517206014
164	US_TROUT	4.9	21.2661900616	139.6821249847
164	US_TROUT	5	20.5355658064	137.4840731984
164	US_TROUT	5.1	19.8049415513	135.3553099865
164	US_TROUT	5.2	19.0759415071	133.2936511843
164	US_TROUT	5.3	18.350189885	131.2969814784
164	US_TROUT	5.4	17.629310896	129.3632522355
164	US_TROUT	5.5	16.914928751	127.490479401
164	US_TROUT	5.6	16.208667661	125.6767414629
164	US_TROUT	5.7	15.5121518372	123.9201774807
164	US_TROUT	5.8	14.8270054905	122.2189851757
164	US_TROUT	5.9	14.1548528319	120.5714190819
164	US_TROUT	6	13.4973180725	118.9757887552
164	US_TROUT	6.1	12.8560254232	117.4304570389
164	US_TROUT	6.2	12.232599095	115.9338383839
164	US_TROUT	6.3	11.6286632991	114.4843972219
164	US_TROUT	6.4	11.0458422464	113.08064639
164	US_TROUT	6.5	10.4857601478	111.7211456045
164	US_TROUT	6.6	9.9500412145	110.4044999837
164	US_TROUT	6.7	9.4403096575	109.1293586161
164	US_TROUT	6.8	8.9581896877	107.8944131747
164	US_TROUT	6.9	8.5053055162	106.6983965745
164	US_TROUT	7	8.083281354	105.5400816724

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_TROUT	7.1	7.6937414121	104.4182800084
164	US_TROUT	7.2	7.3383099016	103.3318405858
164	US_TROUT	7.3	7.0186110334	102.2796486906
164	US_TROUT	7.4	6.7362690185	101.2606247474
164	US_TROUT	7.5	6.492908068	100.2737232122
164	US_TROUT	7.6	6.2901523929	99.3179314991
164	US_TROUT	7.7	6.1296262042	98.3922689421
164	US_TROUT	7.8	6.0129537129	97.495785788
164	US_TROUT	7.9	5.9417591301	96.6275622227
164	US_TROUT	8	5.9176666667	95.786707427
164	US_TROUT	8.1	5.9133475135	94.9723586629
164	US_TROUT	8.2	5.9005725598	94.1836803881
164	US_TROUT	8.3	5.8796155639	93.4198633988
164	US_TROUT	8.4	5.8507502843	92.6801239998
164	US_TROUT	8.5	5.8142504796	91.9637031998
164	US_TROUT	8.6	5.7703899081	91.2698659332
164	US_TROUT	8.7	5.7194423285	90.5979003054
164	US_TROUT	8.8	5.6616814991	89.947116863
164	US_TROUT	8.9	5.5973811785	89.3168478858
164	US_TROUT	9	5.5268151252	88.7064467021
164	US_TROUT	9.1	5.4502570975	88.1152870248
164	US_TROUT	9.2	5.3679808541	87.5427623094
164	US_TROUT	9.3	5.2802601534	86.9882851312
164	US_TROUT	9.4	5.1873687538	86.4512865826
164	US_TROUT	9.5	5.0895804139	85.9312156897
164	US_TROUT	9.6	4.9871688922	85.4275388467
164	US_TROUT	9.7	4.880407947	84.9397392685

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_TROUT	9.8	4.769571337	84.4673164606
164	US_TROUT	9.9	4.6549328206	84.0097857052
164	US_TROUT	10	4.5367661562	83.5666775645
164	US_TROUT	10.1	4.4153451024	83.1375373983
164	US_TROUT	10.2	4.2909434177	82.721924898
164	US_TROUT	10.3	4.1638348605	82.3194136347
164	US_TROUT	10.4	4.0342931893	81.929590622
164	US_TROUT	10.5	3.9025921625	81.5520558915
164	US_TROUT	10.6	3.7690055388	81.1864220831
164	US_TROUT	10.7	3.6338070765	80.8323140475
164	US_TROUT	10.8	3.4972705342	80.4893684608
164	US_TROUT	10.9	3.3596696703	80.1572334524
164	US_TROUT	11	3.2212782432	79.8355682433
164	US_TROUT	11.1	3.0823700116	79.524042797
164	US_TROUT	11.2	2.9432187339	79.2223374806
164	US_TROUT	11.3	2.8040981685	78.9301427369
164	US_TROUT	11.4	2.665282074	78.6471587666
164	US_TROUT	11.5	2.5270442088	78.3730952213
164	US_TROUT	11.6	2.3896583314	78.1076709048
164	US_TROUT	11.7	2.2533982003	77.850613485
164	US_TROUT	11.8	2.118537574	77.6016592148
164	US_TROUT	11.9	1.9853502109	77.3605526606
164	US_TROUT	12	1.8541098696	77.1270464412
164	US_TROUT	12.1	1.7250903085	76.9009009733
164	US_TROUT	12.2	1.5985652861	76.681884226
164	US_TROUT	12.3	1.4748085609	76.4697714827
164	US_TROUT	12.4	1.3540938914	76.2643451103

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
164	US_TROUT	12.5	1.236695036	76.0653943362
164	US_TROUT	12.6	1.1228857533	75.8727150319
164	US_TROUT	12.7	1.0129398017	75.6861095037
164	US_TROUT	12.8	0.9071309398	75.5053862896
164	US_TROUT	12.9	0.8057329259	75.330359963
164	US_TROUT	13	0.7090195186	75.1608509425
164	US_TROUT	13.1	0.6172644763	74.9966853077
164	US_TROUT	13.2	0.5307415576	74.8376946205
164	US_TROUT	13.3	0.449724521	74.6837157525
164	US_TROUT	13.4	0.3744871248	74.5345907174
164	US_TROUT	13.5	0.3053031276	74.3901665093
164	US_TROUT	13.6	0.2424462879	74.2502949452
164	US_TROUT	13.7	0.1861903642	74.1148325135
164	US_TROUT	13.8	0.1368091149	73.9836402263
164	US_TROUT	13.9	0.0945762986	73.8565834771
164	US_TROUT	14	0.0597656736	73.7335319023
164	US_TROUT	14.1	0.0326509986	73.6143592481
164	US_TROUT	14.2	0.0135060319	73.4989432403
164	US_TROUT	14.3	0.0026045321	73.3871654591
171	US_BIGMU	0	39.64445595	404.943
171	US_BIGMU	0.1	39.64445595	395.1791789
171	US_BIGMU	0.2	39.64445595	385.6999401
171	US_BIGMU	0.3	39.64445595	376.4969889
171	US_BIGMU	0.4	39.64445595	367.5622724
171	US_BIGMU	0.5	39.64445595	358.8879726
171	US_BIGMU	0.6	39.64445595	350.4664992
171	US_BIGMU	0.7	39.64445595	342.2904832

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	0.8	39.64445595	334.3527703
171	US_BIGMU	0.9	39.64445595	326.6464147
171	US_BIGMU	1	39.64445595	319.1646732
171	US_BIGMU	1.1	39.64445595	311.9009991
171	US_BIGMU	1.2	39.64445595	304.8490365
171	US_BIGMU	1.3	39.64445595	298.0026146
171	US_BIGMU	1.4	39.64445595	291.3557426
171	US_BIGMU	1.5	39.64445595	284.9026044
171	US_BIGMU	1.6	39.64445595	278.6375533
171	US_BIGMU	1.7	39.64445595	272.5551071
171	US_BIGMU	1.8	39.64445595	266.6499436
171	US_BIGMU	1.9	39.64445595	260.9168956
171	US_BIGMU	2	39.64445595	255.3509464
171	US_BIGMU	2.1	39.58598514	249.9472258
171	US_BIGMU	2.2	39.41863764	244.7010053
171	US_BIGMU	2.3	39.15451086	239.6076943
171	US_BIGMU	2.4	38.80570221	234.662836
171	US_BIGMU	2.5	38.3843091	229.8621035
171	US_BIGMU	2.6	37.90242894	225.2012961
171	US_BIGMU	2.7	37.37215914	220.6763354
171	US_BIGMU	2.8	36.80559711	216.2832619
171	US_BIGMU	2.9	36.21484027	212.0182316
171	US_BIGMU	3	35.61198601	207.8775125
171	US_BIGMU	3.1	35.00913175	203.8574813
171	US_BIGMU	3.2	34.4183749	199.9546203
171	US_BIGMU	3.3	33.85181288	196.1655145
171	US_BIGMU	3.4	33.32154308	192.4868482

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	3.5	32.83966292	188.9154026
171	US_BIGMU	3.6	32.41826981	185.4480525
171	US_BIGMU	3.7	32.06946116	182.0817639
171	US_BIGMU	3.8	31.80533438	178.8135911
171	US_BIGMU	3.9	31.63798688	175.6406745
171	US_BIGMU	4	31.57951606	172.5602377
171	US_BIGMU	4.1	31.572478	169.5695851
171	US_BIGMU	4.2	31.551527	166.6660999
171	US_BIGMU	4.3	31.5169078	163.8472414
171	US_BIGMU	4.4	31.46886518	161.110543
171	US_BIGMU	4.5	31.4076439	158.4536102
171	US_BIGMU	4.6	31.33348871	155.8741179
171	US_BIGMU	4.7	31.24664437	153.369809
171	US_BIGMU	4.8	31.14735565	150.9384923
171	US_BIGMU	4.9	31.03586731	148.5780402
171	US_BIGMU	5	30.9124241	146.2863872
171	US_BIGMU	5.1	30.77727079	144.0615282
171	US_BIGMU	5.2	30.63065213	141.9015162
171	US_BIGMU	5.3	30.47281289	139.8044612
171	US_BIGMU	5.4	30.30399783	137.7685283
171	US_BIGMU	5.5	30.12445171	135.7919359
171	US_BIGMU	5.6	29.93441928	133.8729545
171	US_BIGMU	5.7	29.73414532	132.0099049
171	US_BIGMU	5.8	29.52387457	130.2011568
171	US_BIGMU	5.9	29.3038518	128.4451276
171	US_BIGMU	6	29.07432176	126.7402807
171	US_BIGMU	6.1	28.83552923	125.0851242

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	6.2	28.58771896	123.47821
171	US_BIGMU	6.3	28.33113571	121.9181319
171	US_BIGMU	6.4	28.06602423	120.4035247
171	US_BIGMU	6.5	27.7926293	118.9330632
171	US_BIGMU	6.6	27.51119567	117.5054607
171	US_BIGMU	6.7	27.22196811	116.1194679
171	US_BIGMU	6.8	26.92519136	114.7738721
171	US_BIGMU	6.9	26.62111019	113.4674959
171	US_BIGMU	7	26.30996937	112.1991961
171	US_BIGMU	7.1	25.99201365	110.9678629
171	US_BIGMU	7.2	25.66748779	109.7724189
171	US_BIGMU	7.3	25.33663656	108.6118181
171	US_BIGMU	7.4	24.99970471	107.4850448
171	US_BIGMU	7.5	24.656937	106.3911131
171	US_BIGMU	7.6	24.3085782	105.3290658
171	US_BIGMU	7.7	23.95487306	104.2979736
171	US_BIGMU	7.8	23.59606635	103.2969343
171	US_BIGMU	7.9	23.23240283	102.3250718
171	US_BIGMU	8	22.86412724	101.3815358
171	US_BIGMU	8.1	22.49148437	100.4655007
171	US_BIGMU	8.2	22.11471896	99.57616493
171	US_BIGMU	8.3	21.73407578	98.71275026
171	US_BIGMU	8.4	21.34979958	97.87450119
171	US_BIGMU	8.5	20.96213513	97.06068423
171	US_BIGMU	8.6	20.57132719	96.27058728
171	US_BIGMU	8.7	20.17762052	95.50351897
171	US_BIGMU	8.8	19.78125987	94.75880809

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	8.9	19.38249001	94.03580301
171	US_BIGMU	9	18.9815557	93.33387108
171	US_BIGMU	9.1	18.5787017	92.65239807
171	US_BIGMU	9.2	18.17417277	91.99078769
171	US_BIGMU	9.3	17.76821367	91.34846101
171	US_BIGMU	9.4	17.36106916	90.72485597
171	US_BIGMU	9.5	16.952984	90.11942689
171	US_BIGMU	9.6	16.54420296	89.53164402
171	US_BIGMU	9.7	16.13497078	88.96099303
171	US_BIGMU	9.8	15.72553223	88.40697457
171	US_BIGMU	9.9	15.31613208	87.86910387
171	US_BIGMU	10	14.90701507	87.34691027
171	US_BIGMU	10.1	14.49842598	86.83993685
171	US_BIGMU	10.2	14.09060956	86.34773998
171	US_BIGMU	10.3	13.68381058	85.86988897
171	US_BIGMU	10.4	13.27827379	85.4059657
171	US_BIGMU	10.5	12.87424395	84.95556422
171	US_BIGMU	10.6	12.47196582	84.51829042
171	US_BIGMU	10.7	12.07168417	84.09376166
171	US_BIGMU	10.8	11.67364376	83.68160647
171	US_BIGMU	10.9	11.27808934	83.2814642
171	US_BIGMU	11	10.88526567	82.89298473
171	US_BIGMU	11.1	10.49541752	82.51582811
171	US_BIGMU	11.2	10.10878965	82.14966432
171	US_BIGMU	11.3	9.725626807	81.79417297
171	US_BIGMU	11.4	9.346173765	81.44904299
171	US_BIGMU	11.5	8.97067528	81.11397237

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	11.6	8.599376112	80.78866792
171	US_BIGMU	11.7	8.232521022	80.47284499
171	US_BIGMU	11.8	7.870354769	80.16622723
171	US_BIGMU	11.9	7.513122114	79.86854633
171	US_BIGMU	12	7.161067818	79.57954182
171	US_BIGMU	12.1	6.81443664	79.29896081
171	US_BIGMU	12.2	6.473473342	79.02655779
171	US_BIGMU	12.3	6.138422683	78.76209439
171	US_BIGMU	12.4	5.809529423	78.5053392
171	US_BIGMU	12.5	5.487038324	78.25606755
171	US_BIGMU	12.6	5.171194144	78.01406132
171	US_BIGMU	12.7	4.862241646	77.77910876
171	US_BIGMU	12.8	4.560425588	77.55100426
171	US_BIGMU	12.9	4.265990732	77.32954823
171	US_BIGMU	13	3.979181837	77.1145469
171	US_BIGMU	13.1	3.700243664	76.90581212
171	US_BIGMU	13.2	3.429420974	76.70316126
171	US_BIGMU	13.3	3.166958526	76.50641697
171	US_BIGMU	13.4	2.913101081	76.31540712
171	US_BIGMU	13.5	2.668093399	76.12996456
171	US_BIGMU	13.6	2.432180241	75.94992701
171	US_BIGMU	13.7	2.205606366	75.77513695
171	US_BIGMU	13.8	1.988616536	75.60544143
171	US_BIGMU	13.9	1.78145551	75.44069195
171	US_BIGMU	14	1.584368049	75.28074437
171	US_BIGMU	14.1	1.397598913	75.12545871
171	US_BIGMU	14.2	1.221392863	74.9746991

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik
171	US_BIGMU	14.3	1.055994658	74.82833362
171	US_BIGMU	14.4	0.9016490593	74.6862342
171	US_BIGMU	14.5	0.7586008272	74.54827649
171	US_BIGMU	14.6	0.6270947218	74.41433979
171	US_BIGMU	14.7	0.5073755034	74.28430688
171	US_BIGMU	14.8	0.3996879325	74.15806399
171	US_BIGMU	14.9	0.3042767692	74.03550065
171	US_BIGMU	15	0.221386774	73.91650962
171	US_BIGMU	15.1	0.1512627071	73.80098677
171	US_BIGMU	15.2	0.0941493289	73.68883102
171	US_BIGMU	15.3	0.0502913997	73.57994423
171	US_BIGMU	15.4	0.0199336797	73.47423113
171	US_BIGMU	15.5	0.0033209294	73.37159919

B.2.2 Selecting parameter values for interpolation testing

The depths of the desired light proportions were then manually calculated, and the data points closest in depth were selected. For example, for US_SPARK day 160, the depth of 1% light was calculated to be about 13.4 meters, so the values from pprinputs_Colin.xlsx at depth 13.4 were used (Table B.2).

Table B.2: Example data for interpolation testing

day_of_year	lake_ID	depth	benthic_pmax	benthic_ik	optical_depth
160	US_SPARK	0	14.7563703653	404.943	1.00
160	US_SPARK	0.7	16.8567990535	333.4427073373	0.80
160	US_SPARK	2	57.9816558706	238.6559725805	0.50
160	US_SPARK	4	36.722999797	154.9128285247	0.25
160	US_SPARK	6.7	21.8775756059	103.6069722003	0.10
160	US_SPARK	13.4	0.0012484529	73.3339661329	0.01
164	US_CRYST	0	19.1303546145	404.943	1.00
164	US_CRYST	0.7	19.1303546145	335.9472556194	0.80
164	US_CRYST	2.1	19.195871963	237.6607327084	0.50
164	US_CRYST	4.2	28.0230188436	153.9133678026	0.25
164	US_CRYST	7	11.3840047286	103.3291310615	0.10
164	US_CRYST	13.9	0.0041578269	73.3897556364	0.01
164	US_TROUT	0	35.1534649462	404.943	1.00
164	US_TROUT	0.7	35.1534649462	337.6588712091	0.80
164	US_TROUT	2.2	35.0581778999	235.5297823354	0.50
164	US_TROUT	4.3	25.5589797756	154.4570645114	0.25
164	US_TROUT	7.2	7.3383099016	103.3318405858	0.10
164	US_TROUT	14.3	0.0026045321	73.3871654591	0.01

B.2.3 Devlin output values

Values for benthic primary production calculated using the selected data were then compared against calculated values for the same lakes, for the same days, provided by Dr. Devlin (Table B.3).

TABLE B.3: pproutputs_Colin.xlsx

	Lake_ID	DOY	Bppr.m2
1	US_BIGMU	171	286.8234852
2	US_BIGMU	193	210.3905481
3	US_BIGMU	194	226.7434736
4	US_BIGMU	209	244.0238483
5	US_BIGMU	212	365.0429936
6	US_BIGMU	224	216.4839574
7	US_CRYST	164	174.271404
8	US_CRYST	188	146.203903
9	US_CRYST	200	151.8723555
10	US_CRYST	228	232.9958418
11	US_LITTL	166	195.3752144
12	US_LITTL	189	219.2919569
13	US_LITTL	201	216.2048878
14	US_LITTL	229	256.0982216
15	US_SPARK	160	254.6715329
16	US_SPARK	172	413.6384345
17	US_SPARK	186	305.3512556
18	US_SPARK	202	508.7571973
19	US_SPARK	215	232.8215528
20	US_SPARK	227	344.684941
21	US_SPARK	234	335.484913

	Lake_ID	DOY	Bppr.m2
22	US_TROUT	164	188.1237123
23	US_TROUT	171	263.8388339
24	US_TROUT	190	181.3083071
25	US_TROUT	214	252.1843583
26	US_TROUT	225	209.797893
27	US_TROUT	229	249.9694686

B.3 Testing phytoplanktonic primary production

B.3.1 Data sources

Validating phytoplanktonic primary production required assembling test parameter values from several sources.

- "Phyte_90s.csv" and later, "PI2000sPhoto.csv", received from Dr. Vadeboncoeur: α, P_{max}, and β for each layer, for each pond, for each day.
- NTL LTER Light Extinction database: light extinction coefficient
- North Temperate Lakes LTER: Primary Production - Trout Lake Area 1986 - 2007 database: day length, noon light, thermal layer depths. Also, this database contains the output values against which we compared the program output.

B.3.1.1 Phyte_90s.csv

Initial testing revealed some trouble with this data file.

Table B.4: Phyte_90s.csv

Lake_ID	Year	DOY	stratum	PMAX	alpha	b
US_CRYST	1995	152	1	12.573396381	0.0363236129	0.0082111384
US_CRYST	1995	166	1	14.225868948	0.0261208166	0.0062117318
US_CRYST	1995	180	1	16.018816695	0.0223830795	0.0102509115
US_CRYST	1995	194	1	28.459350024	0.0468462303	0.0214344156
US_CRYST	1995	208	1	16.024860567	0.0364274582	0.007306914
US_CRYST	1995	222	1	14.372101026	0.0298177915	0.0104311194
US_CRYST	1995	236	1	8.8476727941	0.0346486734	0.0060404583
US_CRYST	1996	162	1	13.784678074	0.0352721626	0.0042233295
US_CRYST	1996	178	1	8.617852999	0.036758373	0.0045144984
US_CRYST	1996	191	1	17.287281061	0.0302336037	0.0089433254
US_CRYST	1996	206	1	20.58951494	0.0268257768	0.0235773361
US_CRYST	1996	220	1	13.594276403	0.0240822796	0.0087340022
US_CRYST	1996	234	1	8.005227938	0.0138782989	0.0091783448
US_CRYST	1996	247	1	11.499952727	0.0170815045	0.0097610806
US_CRYST	1997	168	1	11.556004241	0.018277843	0.0107500029
US_CRYST	1997	168	2	1.5670108571	0.0146318474	0
US_CRYST	1997	168	3	3.1126622755	0.036283937	0
US_CRYST	1997	184	1	11.270889126	0.0190547074	0.0090754423
US_CRYST	1997	184	2	2.8475925592	0.0272034085	0
US_CRYST	1997	184	3	5.1100807173	0.0981272838	0
US_CRYST	1997	196	1	94212.039355	0.0161562011	155.27309818
US_CRYST	1997	196	2	2.8750034575	0.0240525258	0
US_CRYST	1997	196	3	5.0245789789	0.1001343716	0
US_CRYST	1997	241	1	9.9180808556	0.0161458046	0.009809034
US_CRYST	1997	241	2	2.8489820164	0.0360166018	0
US_CRYST	1997	241	3	6.7462266128	0.1716064712	0

Lake_ID	Year	DOY	stratum	PMAX	alpha	b
US_SPARK	1995	151	1	7.1341068624	0.0211979226	0.0091903853
US_SPARK	1995	151	2	2.5062080985	0.0423488837	0
US_SPARK	1995	151	3	6.4504900739	0.1524206706	0
US_SPARK	1995	165	1	4.1138675444	0.05	0.01
US_SPARK	1995	165	2	2.6367659654	0.0412667109	0
US_SPARK	1995	165	3	4.9593398366	0.1646230769	0
US_SPARK	1995	178	1	10	0.0286343053	0.0072316059
US_SPARK	1995	178	2	3.6368813967	0.0400185714	0
US_SPARK	1995	178	3	5.4157050245	0.180425915	0
US_SPARK	1995	200	1	9.0945703699	0.0431750747	0.0040652956
US_SPARK	1995	200	2	5.0492213331	0.0825628069	0
US_SPARK	1995	200	3	4.6112769781	0.1483984278	0
US_SPARK	1995	206	1	13.322303252	0.0283504584	0.0084212269
US_SPARK	1995	206	2	7.316552368	0.0740148879	0
US_SPARK	1995	206	3	4.762666608	0.2573020687	0
US_SPARK	1995	220	1	10	0.0527520084	0.0117905301
US_SPARK	1995	220	2	4.7261424229	0.0596439151	0
US_SPARK	1995	220	3	5.372189975	0.1481807297	0
US_SPARK	1995	235	1	2.3730629686	0.05	-0.000426468
US_SPARK	1995	235	2	3.8564724596	0.0616566022	0
US_SPARK	1995	235	3	3.4566806267	0.1388458024	0
US_SPARK	1995	249	1	10	0.0283421335	0.0128089635
US_SPARK	1995	249	2	5.7327735511	0.0718012316	0
US_SPARK	1995	249	3	3.2072132826	0.1321555135	0
US_SPARK	1996	162	1	8.6652021967	0.0208517299	0.0098886533
US_SPARK	1996	178	1	6.286996317	0.0528598156	0.0046701023
US_SPARK	1996	191	1	10.030815192	0.0267824463	0.0089770579

Lake_ID	Year	DOY	stratum	PMAX	alpha	b
US_SPARK	1996	206	1	11.008079696	0.0537939663	0.0096198259
US_SPARK	1996	220	1	10	0.0254586929	0.0147626957
US_SPARK	1996	234	1	7.8750480924	0.0195903035	0.009702043
US_SPARK	1996	247	1	10	0.0209116833	0.0138980781
US_SPARK	1997	169	1	18.077442559	0.0506943157	0.0129456862
US_SPARK	1997	183	1	12.63253707	0.0465696954	0.0039492152
US_SPARK	1997	198	1	11.648883738	0.0096215863	0.0098179513
US_SPARK	1997	217	1	8.4323448014	0.0159388408	0.0083223363
US_SPARK	1997	240	1	7.6886631247	0.0175255487	0.0096519661
US_TROUT	1995	152	1	11.304991124	0.038387625	0.0127845788
US_TROUT	1995	166	1	7.6051364551	0.0317827985	0.0049831535
US_TROUT	1995	180	1	13.071108786	0.0336535724	0.0131332661
US_TROUT	1995	194	1	10.887479149	0.041878557	0.0075241832
US_TROUT	1995	208	1	10.010981486	0.0401656208	0.0079225391
US_TROUT	1995	222	1	12.104947287	0.0652697765	0.0069690877
US_TROUT	1995	236	1	11.542937115	0.100041022	0.008292905
US_TROUT	1996	151	1	13.668738793	0.0803275996	0.0088362319
US_TROUT	1996	151	2	6.9097106277	0.0805479164	0
US_TROUT	1996	151	3	8.691306875	0.1243841651	0
US_TROUT	1996	163	1	14.596926921	0.0307438836	0.0115350469
US_TROUT	1996	163	2	5.5135426394	0.1215081471	0
US_TROUT	1996	163	3	6.2883413054	0.1707940388	0
US_TROUT	1996	175	1	15.554650813	0.0763082338	0.0114774342
US_TROUT	1996	175	2	7.3102949126	0.1263593754	0
US_TROUT	1996	175	3	8.7853658209	0.2949131248	0
US_TROUT	1996	190	1	91019.132356	0.0334036768	154.68276001
US_TROUT	1996	190	2	5.9689679977	0.0969767449	0

Lake_ID	Year	DOY	stratum	PMAX	alpha	b
US_TROUT	1996	190	3	7.8129073902	0.2976636715	0
US_TROUT	1996	207	1	7.2324258694	0.0383440142	0.0044932935
US_TROUT	1996	207	2	5.2808973752	0.1423595764	0
US_TROUT	1996	207	3	5.0932830489	0.2641653504	0
US_TROUT	1996	217	1	6.3213273392	0.0469303643	0.0046077733
US_TROUT	1996	217	2	4.0218311047	0.0770101649	0
US_TROUT	1996	217	3	8.2778970082	0.276099706	0
US_TROUT	1996	231	1	6.1905421438	0.0224594263	0.0069299365
US_TROUT	1996	231	2	3.3237494346	0.0730561346	0
US_TROUT	1996	231	3	6.5686037623	0.3126898418	0
US_TROUT	1996	249	1	4.7150417594	0.025131018	0.0039852174
US_TROUT	1996	249	2	2.9011782681	0.0642018875	0
US_TROUT	1996	249	3	4.4098461976	0.2397650136	0
US_TROUT	1997	169	1	10	0.0225843374	0.0111050324
US_TROUT	1997	183	1	7.3508776037	0.0126714094	0.0085295407
US_TROUT	1997	198	1	4.9936757646	0.0290874318	0.0036201461
US_TROUT	1997	217	1	136101.89157	0.0227843412	283.1010228
US_TROUT	1997	240	1	7.5892235297	0.0184313211	0.0090666144

B.3.1.2 PI2000sPhoto.csv

After our initial testing revealed some problems with the Phyte_90s.csv file, Dr. Vadeboncoeur sent this updated file instead.

Table B.5: PI2000sPhoto.csv

Lake_ID	Year	stratum	DOY	alpha	PMax	b
US_CRYST	2006	1	125	0.1602297768	16.554412222	0.01

Lake_ID	Year	stratum	DOY	alpha	PMax	b
US_CRYST	2006	1	166	0.15	3.3792540235	0.01
US_CRYST	2006	1	180	0.15	3.0369320671	0.01
US_CRYST	2006	1	194	0.15	4.3890081246	0.01
US_CRYST	2006	1	215	0.15	6.1172215153	0.01
US_CRYST	2006	1	236	0.15	3.7347620216	0.01
US_CRYST	2006	1	258	0.15	2.915312696	0.01
US_CRYST	2006	1	286	0.0660409067	8.2623244229	0.0071957009
US_CRYST	2006	1	298	0.1176278269	14.566978218	0.0173092881
US_CRYST	2006	1	312	0.1014295551	8.5001284103	0.0092415157
US_CRYST	2007	1	130	0.0626266032	9.1062897726	0.0038570449
US_CRYST	2007	1	144	0.15	6.5544808785	0.01
US_CRYST	2007	1	158	0.15	2.8344517901	0.01
US_CRYST	2007	1	172	0.15	2.8066647971	0.01
US_CRYST	2007	1	186	0.15	1.9996969087	0.01
US_CRYST	2007	1	200	0.15	4.8848821453	0.01
US_CRYST	2007	1	214	0.15	7.0391446058	0.01
US_CRYST	2007	1	250	0.0755148266	19.461754345	0.0250651304
US_CRYST	2007	1	277	0.15	6.2773597642	0.01
US_CRYST	2007	1	290	0.4497200022	10.720819201	-0.000306045
US_CRYST	2007	1	318	0.1007193777	11.49911349	0.0181334373
US_CRYST	2007	2	130	0.1650338932	15.109665007	0.0211231913
US_CRYST	2007	2	144	0.0592173571	8.8589406256	0.0153646851
US_CRYST	2007	2	158	0.15	1.0956298622	-0.000840962
US_CRYST	2007	2	172	0.15	5.0064516128	0.01
US_CRYST	2007	2	186	0.15	4.1968458743	0.01
US_CRYST	2007	2	200	0.0423214499	10.784871015	0.0064696765
US_CRYST	2007	2	214	0.1366021013	12.023183504	0.0091554575

Lake_ID	Year	stratum	DOY	alpha	PMax	b
US_CRYST	2007	2	250	0.3812096996	2445518.2194	13294.834529
US_CRYST	2007	2	277	0.3603688549	80.588538836	0.2014807966
US_CRYST	2007	3	130	0.4461090039	17.833458457	0.0328539807
US_CRYST	2007	3	144	0.3499378407	15.655071602	0.0315975357
US_CRYST	2007	3	158	0.0738659068	5.8446420677	0.0045379415
US_CRYST	2007	3	172	0.1600834275	8.2198239341	0.0220964213
US_CRYST	2007	3	186	0.1471416153	5.5216665082	0.0093711939
US_CRYST	2007	3	200	0.1584570744	18.399596077	0.0912030798
US_CRYST	2007	3	215	0.2732634525	712843.97486	4727.9882416
US_CRYST	2007	3	250	0.5156251652	24.577455726	0.0503462443
US_CRYST	2007	3	277	0.5113346132	28.759151776	0.126532002
US_CRYST	2007	3	290	0.1580677081	14.730207402	0.0573888747
US_SPARK	2006	1	124	0.1267238718	12.906516311	0.012670822
US_SPARK	2006	1	166	0.15	4.2662205282	0.01
US_SPARK	2006	1	180	0.15	4.4964439652	0.01
US_SPARK	2006	1	194	0.15	0.6737367792	-0.000986364
US_SPARK	2006	1	215	0.15	1.0509203452	-0.001095913
US_SPARK	2006	1	236	0.15	1.0266685273	-0.001489197
US_SPARK	2006	1	258	0.15	0.5863926599	0.01
US_SPARK	2006	1	286	0.0656564174	5.8468519499	0.0063079241
US_SPARK	2006	1	298	0.0639972993	7.7417564285	0.0128724067
US_SPARK	2006	1	312	0.0824337148	12.28538274	0.0253259753
US_SPARK	2007	1	129	0.0508538121	15.861954271	0.0269617057
US_SPARK	2007	1	141	0.042588377	13.896212504	0.0191933074
US_SPARK	2007	1	157	0.0451498763	16.799401765	0.0101434738
US_SPARK	2007	1	171	0.15	4.5324668562	0.01
US_SPARK	2007	1	184	0.0629915897	521467.88552	907.92272727

Lake_ID	Year	stratum	DOY	alpha	PMax	b
US_SPARK	2007	1	198	0.15	6.5200144232	0.01
US_SPARK	2007	1	212	0.15	6.4864334169	0.01
US_SPARK	2007	1	249	0.0554963449	14.301058604	0.0179155693
US_SPARK	2007	1	264	0.0513174149	8.8573931466	0.0074325196
US_SPARK	2007	1	276	0.087511062	9.3311060625	0.0088855652
US_SPARK	2007	1	291	0.1150163643	15.64067322	0.0230995671
US_SPARK	2007	1	318	0.1920553462	17.706957911	0.0254278794
US_TROUT	2006	1	125	0.1206097614	24.024920142	0.0499766637
US_TROUT	2006	1	165	0.043798591	484397.64726	822.17306503
US_TROUT	2006	1	178	0.0433993213	500798.13619	1082.0746022
US_TROUT	2006	1	192	0.15	8.2118500238	0.01
US_TROUT	2006	1	214	0.0707433606	277779.49976	1063.4999141
US_TROUT	2006	1	233	0.0782248633	13.378916201	0.0095851973
US_TROUT	2006	1	257	0.0568171888	6.5406728738	0.0028796722
US_TROUT	2006	1	284	0.0915769106	14.461243262	0.0206921378
US_TROUT	2006	1	297	0.1070694793	12.110729057	0.0140592551
US_TROUT	2006	1	311	0.1207949876	14.073180447	0.0159987087
US_TROUT	2006	2	165	0.0660183652	306675.88249	1317.9909123
US_TROUT	2006	2	178	0.0744622137	8.0554908224	0.0063234473
US_TROUT	2006	2	192	0.0670318543	36.958221902	0.1239224014
US_TROUT	2006	2	214	0.0991024244	10.293488109	0.0131759805
US_TROUT	2006	2	233	0.1155144289	7.2869912125	0.0143748473
US_TROUT	2006	2	257	0.0907828928	6.8138365442	0.0112702511
US_TROUT	2006	3	165	0.1562807792	8.4780081569	0.0116129962
US_TROUT	2006	3	178	0.1247226255	12.0876065	0.0440370205
US_TROUT	2006	3	233	0.143478966	5.6102781162	0.0282008512
US_TROUT	2007	1	129	0.1999886062	31.322210987	0.0718914355

Lake_ID	Year	stratum	DOY	alpha	PMax	b
US_TROUT	2007	1	141	0.0957554967	20.090774735	0.0334835435
US_TROUT	2007	1	157	0.0682293698	22.575694983	0.0562142885
US_TROUT	2007	1	171	0.0655885164	239768.50746	846.37170221
US_TROUT	2007	1	184	0.0552929666	76.194913885	0.0092179829
US_TROUT	2007	1	198	0.0601039781	557359.05868	1609.6139975
US_TROUT	2007	1	212	0.0537609224	451029.39955	1020.5236441
US_TROUT	2007	1	249	0.1682042923	15.402787794	0.0135501309
US_TROUT	2007	1	264	0.1182475788	19.424189397	0.0221070191
US_TROUT	2007	1	276	0.1127368721	18.69729194	0.0244706301
US_TROUT	2007	1	291	0.1177342831	17.453688448	0.0288637783
US_TROUT	2007	1	318	0.1533776669	13.393504344	0.0069275339

B.3.2 Phytoplanktonic primary production test data assembly process

The process of assembling test data:

B.3.3 Checking for problem values

Starting with the file PI2000sPhoto.csv, we check for any values that may have problems. If we see a default value, then we assume that the parameter estimation process failed, and we throw out the whole day's measurements.

Note: Before throwing out problem values, there are 93 data rows. After this step, there are 37 rows.

Table B.6: Assembling data: initial problem check

Lake_ID	Year	stratum	DOY	alpha	PMax	b	NOTES
US_CRYST	2006	1	125	0.1602297768	16.55441222	0.01	Default values detected. Throw out.
US_CRYST	2006	1	166	0.15	3.379254024	0.01	Default values detected. Throw out.
US_CRYST	2006	1	180	0.15	3.036932067	0.01	Default values detected. Throw out.
US_CRYST	2006	1	194	0.15	4.389008125	0.01	Default values detected. Throw out.
US_CRYST	2006	1	215	0.15	6.117221515	0.01	Default values detected. Throw out.
US_CRYST	2006	1	236	0.15	3.734762022	0.01	Default values detected. Throw out.
US_CRYST	2006	1	258	0.15	2.915312696	0.01	Default values detected. Throw out.
US_CRYST	2006	1	286	0.0660409067	8.262324423	0.0071957009	
US_CRYST	2006	1	298	0.1176278269	14.56697822	0.0173092881	
US_CRYST	2006	1	312	0.1014295551	8.50012841	0.0092415157	
US_CRYST	2007	1	130	0.0626266032	9.106289773	0.0038570449	
US_CRYST	2007	2	130	0.1650338932	15.10966501	0.0211231913	
US_CRYST	2007	3	130	0.4461090039	17.83345846	0.0328539807	
US_CRYST	2007	1	144	0.15	6.554480879	0.01	Default values detected. Throw out.
US_CRYST	2007	2	144	0.0592173571	8.858940626	0.0153646851	Default values detected. Throw out.
US_CRYST	2007	3	144	0.3499378407	15.6550716	0.0315975357	Default values detected. Throw out.
US_CRYST	2007	1	158	0.15	2.83445179	0.01	Default values detected. Throw out.
US_CRYST	2007	2	158	0.15	1.095629862	-0.000840962	Default values detected. Throw out.

Lake_ID	Year	stratum	DOY	alpha	PMax	b	NOTES
US_CRYST	2007	3	158	0.0738659068	5.844642068	0.0045379415	Default values detected. Throw out.
US_CRYST	2007	1	172	0.15	2.806664797	0.01	Default values detected. Throw out.
US_CRYST	2007	2	172	0.15	5.006451613	0.01	Default values detected. Throw out.
US_CRYST	2007	3	172	0.1600834275	8.219823934	0.0220964213	Default values detected. Throw out.
US_CRYST	2007	1	186	0.15	1.999696909	0.01	Default values detected. Throw out.
US_CRYST	2007	2	186	0.15	4.196845874	0.01	Default values detected. Throw out.
US_CRYST	2007	3	186	0.1471416153	5.521666508	0.0093711939	Default values detected. Throw out.
US_CRYST	2007	1	200	0.15	4.884882145	0.01	Default values detected. Throw out.
US_CRYST	2007	2	200	0.0423214499	10.78487102	0.0064696765	Default values detected. Throw out.
US_CRYST	2007	3	200	0.1584570744	18.39959608	0.0912030798	Default values detected. Throw out.
US_CRYST	2007	1	214	0.15	7.039144606	0.01	Default values detected. Throw out.
US_CRYST	2007	2	214	0.1366021013	12.0231835	0.0091554575	Default values detected. Throw out.
US_CRYST	2007	3	215	0.2732634525	712843.9749	4727.988242	Default values detected. Throw out.
US_CRYST	2007	1	250	0.0755148266	19.46175435	0.0250651304	
US_CRYST	2007	2	250	0.3812096996	2445518.219	13294.83453	Implausible value for Pmax. Throw out
US_CRYST	2007	3	250	0.5156251652	24.57745573	0.0503462443	
US_CRYST	2007	1	277	0.15	6.277359764	0.01	Default values detected. Throw out.
US_CRYST	2007	2	277	0.3603688549	80.58853884	0.2014807966	Default values detected. Throw out.
US_CRYST	2007	3	277	0.5113346132	28.75915178	0.126532002	Default values detected. Throw out.
US_CRYST	2007	1	290	0.4497200022	10.7208192	-0.000306045	negative beta doesn't make sense

Lake_ID	Year	stratum	DOY	alpha	PMax	b	NOTES
US_CRYST	2007	3	290	0.1580677081	14.7302074	0.0573888747	<—- Missing metalimnion?
US_CRYST	2007	1	318	0.1007193777	11.49911349	0.0181334373	
US_SPARK	2006	1	124	0.1267238718	12.90651631	0.012670822	
US_SPARK	2006	1	166	0.15	4.266220528	0.01	Default values detected. Throw out.
US_SPARK	2006	1	180	0.15	4.496443965	0.01	Default values detected. Throw out.
US_SPARK	2006	1	194	0.15	0.6737367792	-0.000986364	Default values detected. Throw out.
US_SPARK	2006	1	215	0.15	1.050920345	-0.001095913	Default values detected. Throw out.
US_SPARK	2006	1	236	0.15	1.026668527	-0.001489197	Default values detected. Throw out.
US_SPARK	2006	1	258	0.15	0.5863926599	0.01	Default values detected. Throw out.
US_SPARK	2006	1	286	0.0656564174	5.84685195	0.0063079241	
US_SPARK	2006	1	298	0.0639972993	7.741756429	0.0128724067	
US_SPARK	2006	1	312	0.0824337148	12.28538274	0.0253259753	
US_SPARK	2007	1	129	0.0508538121	15.86195427	0.0269617057	
US_SPARK	2007	1	141	0.042588377	13.8962125	0.0191933074	
US_SPARK	2007	1	157	0.0451498763	16.79940177	0.0101434738	
US_SPARK	2007	1	171	0.15	4.532466856	0.01	Default values detected. Throw out.
US_SPARK	2007	1	184	0.0629915897	521467.8855	907.9227273	
US_SPARK	2007	1	198	0.15	6.520014423	0.01	Default values detected. Throw out.
US_SPARK	2007	1	212	0.15	6.486433417	0.01	Default values detected. Throw out.
US_SPARK	2007	1	249	0.0554963449	14.3010586	0.0179155693	

Lake_ID	Year	stratum	DOY	alpha	PMax	b	NOTES
US_SPARK	2007	1	264	0.0513174149	8.857393147	0.0074325196	
US_SPARK	2007	1	276	0.087511062	9.331106063	0.0088855652	
US_SPARK	2007	1	291	0.1150163643	15.64067322	0.0230995671	
US_SPARK	2007	1	318	0.1920553462	17.70695791	0.0254278794	
US_TROUT	2006	1	125	0.1206097614	24.02492014	0.0499766637	
US_TROUT	2006	1	165	0.043798591	484397.6473	822.173065	Implausible value for Pmax. Throw out
US_TROUT	2006	2	165	0.0660183652	306675.8825	1317.990912	Implausible value for Pmax. Throw out
US_TROUT	2006	3	165	0.1562807792	8.478008157	0.0116129962	Implausible value for Pmax. Throw out
US_TROUT	2006	1	178	0.0433993213	500798.1362	1082.074602	Implausible value for Pmax. Throw out
US_TROUT	2006	2	178	0.0744622137	8.055490822	0.0063234473	Implausible value for Pmax. Throw out
US_TROUT	2006	3	178	0.1247226255	12.0876065	0.0440370205	Implausible value for Pmax. Throw out
US_TROUT	2006	1	192	0.15	8.211850024	0.01	Default values detected. Throw out.
US_TROUT	2006	2	192	0.0670318543	36.9582219	0.1239224014	Default values detected. Throw out.
US_TROUT	2006	1	214	0.0707433606	277779.4998	1063.499914	Implausible value for Pmax. Throw out
US_TROUT	2006	2	214	0.0991024244	10.29348811	0.0131759805	Implausible value for Pmax. Throw out
US_TROUT	2006	1	233	0.0782248633	13.3789162	0.0095851973	
US_TROUT	2006	2	233	0.1155144289	7.286991213	0.0143748473	
US_TROUT	2006	3	233	0.143478966	5.610278116	0.0282008512	
US_TROUT	2006	1	257	0.0568171888	6.540672874	0.0028796722	
US_TROUT	2006	2	257	0.0907828928	6.813836544	0.0112702511	

Lake_ID	Year	stratum	DOY	alpha	PMax	b	NOTES
US_TROUT	2006	1	284	0.0915769106	14.46124326	0.0206921378	
US_TROUT	2006	1	297	0.1070694793	12.11072906	0.0140592551	
US_TROUT	2006	1	311	0.1207949876	14.07318045	0.0159987087	
US_TROUT	2007	1	129	0.1999886062	31.32221099	0.0718914355	
US_TROUT	2007	1	141	0.0957554967	20.09077474	0.0334835435	
US_TROUT	2007	1	157	0.0682293698	22.57569498	0.0562142885	
US_TROUT	2007	1	171	0.0655885164	239768.5075	846.3717022	Implausible value for Pmax. Throw out
US_TROUT	2007	1	198	0.0601039781	557359.0587	1609.613998	Implausible value for Pmax. Throw out
US_TROUT	2007	1	212	0.0537609224	451029.3996	1020.523644	Implausible value for Pmax. Throw out
US_TROUT	2007	1	249	0.1682042923	15.40278779	0.0135501309	
US_TROUT	2007	1	264	0.1182475788	19.4241894	0.0221070191	
US_TROUT	2007	1	276	0.1127368721	18.69729194	0.0244706301	
US_TROUT	2007	1	291	0.1177342831	17.45368845	0.0288637783	
US_TROUT	2007	1	318	0.1533776669	13.39350434	0.0069275339	

B.3.4 Match with NTL light extinction data

Once we have thrown out problematic values, we can match with the NTL LTER light extinction database.

Note: before matching with NTL LTER light extinction data there were 37 rows of data. After this step there are 31 left.

TABLE B.7: Assembling data: matching with NTL Light Extinction database

Year	stratum	alpha	PMax	b	Lake_ID	DOY	extcoef	NOTES
2006	1	0.0660409067	8.262324423	0.0071957009	US_CRYST	286	0.225	No NTL data found. Used data from day 291
2006	1	0.1176278269	14.56697822	0.0173092881	US_CRYST	298	0.225	No NTL data found. Used data from day 291
2006	1	0.1014295551	8.50012841	0.0092415157	US_CRYST	312	0.279	No NTL data found. Used data from day 307
2007	1	0.0626266032	9.106289773	0.0038570449	US_CRYST	130	0.353	No issues!
2007	2	0.1650338932	15.10966501	0.0211231913	US_CRYST	130	0.353	
2007	3	0.4461090039	17.83345846	0.0328539807	US_CRYST	130	0.353	
2007	1	0.1007193777	11.49911349	0.0181334373	US_CRYST	318	0.277	No NTL data found. Used data from day 318
2006	1	0.1267238718	12.90651631	0.012670822	US_SPARK	124	0.377	No NTL data found. Used data from day 122
2006	1	0.0656564174	5.84685195	0.0063079241	US_SPARK	286	NO DATA	No data found within 7 days of this date.
2006	1	0.0639972993	7.741756429	0.0128724067	US_SPARK	298	0.248	Error noted on database
2006	1	0.0824337148	12.28538274	0.0253259753	US_SPARK	312	NO DATA	No data found within 7 days of this date.
2007	1	0.0508538121	15.86195427	0.0269617057	US_SPARK	129	0.34	
2007	1	0.042588377	13.8962125	0.0191933074	US_SPARK	141	0.363	
2007	1	0.0451498763	16.79940177	0.0101434738	US_SPARK	157	0.391	
2007	1	0.0554963449	14.3010586	0.0179155693	US_SPARK	249	0.273	No NTL data found. Used data from day 254
2007	1	0.0513174149	8.857393147	0.0074325196	US_SPARK	264	0.343	No NTL data found. Used data from day 267
2007	1	0.087511062	9.331106063	0.0088855652	US_SPARK	276	0.262	No NTL data found. Used data from day 267
2007	1	0.1150163643	15.64067322	0.0230995671	US_SPARK	291	0.312	No NTL data found. Used data from day 296

Year	stratum	alpha	PMax	b	Lake_ID	DOY	extcoef	NOTES
2007	1	0.1920553462	17.70695791	0.0254278794	US_SPARK	318	0.411	No NTL data found. Used data from day 316
2006	1	0.1206097614	24.02492014	0.0499766637	US_TROUT	125	0.434	
2006	1	0.0782248633	13.3789162	0.0095851973	US_TROUT	233	0.381	
2006	2	0.1155144289	7.286991213	0.0143748473	US_TROUT	233	0.381	
2006	3	0.143478966	5.610278116	0.0282008512	US_TROUT	233	0.381	
2006	1	0.0568171888	6.540672874	0.0028796722	US_TROUT	257	0.393	No NTL data found. Used data from day 263
2006	2	0.0907828928	6.813836544	0.0112702511	US_TROUT	257	0.393	
2006	1	0.0915769106	14.46124326	0.0206921378	US_TROUT	284	0.389	No NTL data found. Used data from day 290
2006	1	0.1070694793	12.11072906	0.0140592551	US_TROUT	297	0.624	No NTL data found. Used data from day 303
2006	1	0.1207949876	14.07318045	0.0159987087	US_TROUT	311	NO DATA	No data found within 7 days of this date.
2007	1	0.1999886062	31.32221099	0.0718914355	US_TROUT	129	0.413	No NTL data found. Used data from day 128
2007	1	0.0957554967	20.09077474	0.0334835435	US_TROUT	141	0.35	No NTL data found. Used data from day 143
2007	1	0.0682293698	22.57569498	0.0562142885	US_TROUT	157	0.336	No NTL data found. Used data from day 155
2007	1	0.0552929666	76.19491389	0.0092179829	US_TROUT	184	0.275	
2007	1	0.1682042923	15.40278779	0.0135501309	US_TROUT	249	0.354	No NTL data found. Used data from day 255
2007	1	0.1182475788	19.4241894	0.0221070191	US_TROUT	264	0.397	No NTL data found. Used data from day 269
2007	1	0.1127368721	18.69729194	0.0244706301	US_TROUT	276	0.374	No NTL data found. Used data from day 269
2007	1	0.1177342831	17.45368845	0.0288637783	US_TROUT	291	0.428	No NTL data found. Used data from day 297
2007	1	0.1533776669	13.39350434	0.0069275339	US_TROUT	318	0.517	No NTL data found. Used data from day 317

B.3.5 Calculate day length, noon light

Next, we match the data with the NTL Primary Production database. This database has values for primary production and light at half-hour intervals.

By observation of recorded light levels, we estimate the length of day for the date in question.

By plotting light levels and observing the shape, we estimate the peak light level for the date in question.

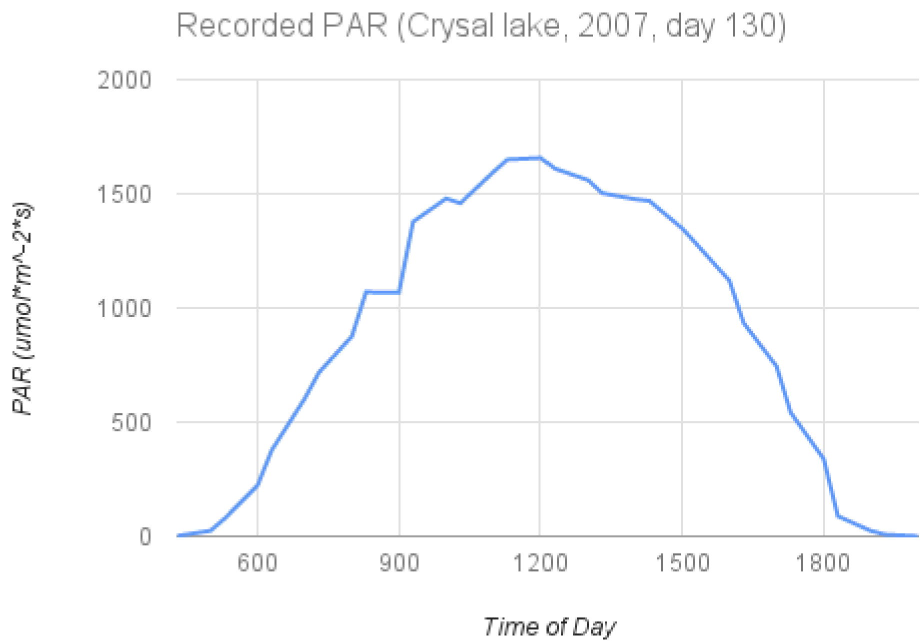

FIGURE B.1: By plotting light values from the NTL LTER database, we can estimate day length and noon light levels: 16 hours and about 1657 micromoles*m^{-2}*hr^{-1} in this case.

To calculate the depth of each thermal layer, we reverse the process used by the NTL LTER database to convert primary production from "per meter cubed" to "per meter squared".

The process used by the NTL LTER database was to simply multiply primary production (per meter cubed) by the thermal layer's vertical interval, the distance from the top to the bottom of the layer:

$pp\_layer\_m3 * layer\_interval = pp\_layer\_m2$

We solve this for layer_interval: $layer\_interval = pp\_layer\_m2/pp\_layer\_m3$

Once we calculate the intervals, we then calculate and store the depth of the bottom edge of each layer.

At the end of all this, we are left with 31 rows of data.

Table B.8: The result of finding peak PAR (noon light), length of day, and stratum interval

DOY	stratum	stratum_interval	noon light	length of day	NOTES
286	1	15.51	343.77	12.5	
298	1	18.47	902.97	11.5	
312	1	NO DATA	NO DATA	NO DATA	No data.
130	1	5	1657.3	16	
130	2	2	1657.3	16	
130	3	13	1657.3	16	
318	1	18.96	488.93	10.5	No data. Used Day 317.
124	1	9.04	1364	15.5	
298	1	17.58	902.97	11.5	
129	1	7	1515.1	15.5	
141	1	5.99	1677.6	16	
157	1	5.966	1641.3	16	
249	1	8.49	1577	13.5	
264	1	9.53	1011.9	13	
276	1	10.59	984.52	12.5	
291	1	11.65	695.28	10.5	
318	1	NO DATA	NO DATA	NO DATA	No data.
125	1	35	1229	15	
233	1	8.15	1617.1	14.5	
233	2	3.89	1617.1	14.5	
233	3	23.5	1617.1	14.5	
257	1	10	1344	13.5	
257	2	NO DATA	1344	13.5	
284	1	25.02	274.91	11.5	
297	1	35.04	713	11	
129	1	2.5	1515.1	15	
141	1	10.51	1677.6	15	
157	1	11.52	1641.3	15.5	
184	1	6.96	1443.9	16	
249	1	11.51	1577	13.5	
264	1	13.09	1011.9	12	
276	1	14.53	984.52	12.5	
291	1	17.51	695.28	10.5	
318	1	35.03	488.93	9.5	No data. Used Day 317.

B.3.6 Layer depths

Using the intervals of each layer, starting with the epilimnion (the top layer), we calculate and store the depth of the bottom edge of each layer.

TABLE B.9: Stratum/layer depths for each lake and day.

Year	DOY	Lake_ID	stratum	stratum_depth
2006	286	US_CRYST	1	15.51
2006	298	US_CRYST	1	18.47
2007	130	US_CRYST	1	5
2007	130	US_CRYST	2	2
2007	130	US_CRYST	3	13
2007	318	US_CRYST	1	18.96
2006	124	US_SPARK	1	9.04
2006	298	US_SPARK	1	17.58
2007	129	US_SPARK	1	7
2007	141	US_SPARK	1	5.99
2007	157	US_SPARK	1	5.966
2007	249	US_SPARK	1	8.49
2007	264	US_SPARK	1	9.53
2007	276	US_SPARK	1	10.59
2007	291	US_SPARK	1	11.65
2006	125	US_TROUT	1	35
2006	233	US_TROUT	1	8.15
2006	233	US_TROUT	2	3.89
2006	233	US_TROUT	3	23.5
2006	257	US_TROUT	1	10
2006	284	US_TROUT	1	25.02
2006	297	US_TROUT	1	35.04
2007	129	US_TROUT	1	2.5
2007	141	US_TROUT	1	10.51
2007	157	US_TROUT	1	11.52
2007	184	US_TROUT	1	6.96
2007	249	US_TROUT	1	11.51
2007	264	US_TROUT	1	13.09
2007	276	US_TROUT	1	14.53
2007	291	US_TROUT	1	17.51
2007	318	US_TROUT	1	35.03

At this point, there are 31 data rows left. However, of all the days listed, the number that have data for all three thermal layers is only 2.

Template user input file

The user input file consist of a minimum of 4 sheets, in the following order: “pond_data”,“benthic_photo_data”,“phytoplankton_photo_data”,“shape_data”.

C.1 Specification

TABLE C.1: Input file specification

Name of sheet	**Name of field**	**explanation**	**Units**	**Data format**
(pond_data)				
	year	4-digit year that the measurement was taken. Standard A.D. Calendar, e.g. “2015”		integer
	day_of_year	Day of year, expressed as a number. e.g. Jan 01 = “1”, Feb 29 = “60”	N/A (unitless, ordinal)	integer

Name of sheet	**Name of field**	**explanation**	**Units**	**Data format**
	lake_ID	ID of lake. e.g. “Sparkling lake, Wisconsin, USA” = “US_SPARK”. Along with day_of_year, forms the primary key.		string
	light_attenuation_coefficient	Coefficient used for calculating light with depth. Measured directly, or calculated from chlorophyll levels	inverse meters (m−1)	float
	noon_surface_light	Light intensity of surface light at solar noon.	micromoles per square meter per second(μ mol*m^{-2}*s^{-1})	float

Name of sheet	Name of field	explanation	Units	Data format
	length_of_day	How long the sun shines on the lake. Example: 15.7 = 15.7 hours of daylight.	hours (h)	float
	latitude	South of the equator is negative, north of the equator is positive. Used for calculating seasonal light changes	decimal degrees, -90 to 90	float
(benthic_photo_data)				
	year	4-digit year that the measurement was taken. Standard A.D. Calendar, e.g. "2015"		integer

Name of sheet	**Name of field**	**explanation**	**Units**	**Data format**
	day_of_year	Same as on pond_data sheet. Values must match exactly between this sheet and pond_data to properly link them.	Same as on pond_data sheet.	Same as on pond_data sheet.
	lake_ID	Same as on pond_data sheet. Values must match exactly between this sheet and pond_data to properly link them.	Same as on pond_data sheet.	Same as on pond_data sheet.

Name of sheet	Name of field	explanation	Units	Data format
	light_penetration_proportion	The proportion of surface light which penetrates to the depth of this measurement. Note: proportion =Iz/I0=e^{-kd*z}. Valid values are from 0.0 to 1.0. Example: a value of "0.75" would mean that light at this depth is 75% of light at the surface.	meters (m)	float
	benthic_pmax	P/I (photosynthesis/irradiance) curve parameter: Maximum possible benthic productivity at depth z.	milligrams of carbon per meter squared per hour (mg C*m^{-2}*h^-1)	float

Name of sheet	Name of field	explanation	Units	Data format
	benthic_ik	P/I curve parameter: Light intensity at onset of photosynthetic saturation. In nature, this generally falls between 0.14 and 0.72 (Kalff)	micromoles per square meter per second(μ mol*m^-2*s^-1)	float
(phytoplankton_photo_data)				
	year	4-digit year that the measurement was taken. Standard A.D. Calendar, e.g. “2015”		integer

Name of sheet	**Name of field**	**explanation**	**Units**	**Data format**
	day_of_year	Same as on pond_data sheet. Values must match exactly between this sheet and pond_data to properly link them.	Same as on pond_data sheet.	Same as on pond_data sheet.
	lake_ID	Same as on pond_data sheet. Values must match exactly between this sheet and pond_data to properly link them.	Same as on pond_data sheet.	Same as on pond_data sheet.
	thermal_layer	0=epilimnion layer, 1 = metalimnion layer, 2 = hypolimnnion layer	N/A (numeric id)	integer
	depth	lower edge of thermal layer	meters	double

Name of sheet	**Name of field**	**explanation**	**Units**	**Data format**
	light_penetration_proportion	The proportion of surface light which penetrates to the depth of this measurement. Note: proportion =Iz/I0=e^(-kd*z). Valid values are from 0.0 to 1.0. Example: a value of "0.75" would mean that light at this depth is 75% of light at the surface.	N/A (unitless proportion)	float
	phyto_pmax_biomass	P/I (photosynthesis/irradiance) curve parameter: Maximum phytoplankton primary production at this depth	mg C*m^-3/hr	float

Name of sheet	Name of field	explanation	Units	Data format
	phyto_alpha	P/I (photosynthesis/irradiance) curve parameter: Slope of photosynthesis/irradiance curve before light saturation point.	(mg C*m^-3/hr)/(μmol*m^-2*s^-1)	float
	phyto_beta	P/I (photosynthesis/irradiance) curve parameter: Slope of photosynthesis/irradiance curve after the point light levels inhibit photosynthesis	(mg C*m^-3/hr)/(μmol*m^-2*s^-1)	float
(shape_data)				

Name of sheet	Name of field	explanation	Units	Data format
	lake_ID	Same as on pond_data sheet. Values must match exactly between this sheet and pond_data to properly link them.	Same as on pond_data sheet.	Same as on pond_data sheet.
	depth	depth in meters	meters	float
	water_surface_area	area of water at this depth.	square meters (m^2)	float

C.2 Sheet 1: pond_data

This sheet holds data not relating specifically to Benthic primary production, Phytoplanktonic primary production, or pond shape.

Table C.2: "pond_data" sheet, empty

year	day_of_year	lake_ID	light_attenuation_coefficient	noon_surface_light	length_of_day

Table C.3: "pond_data" sheet, filled

year	day_of_year	lake_ID	light_attenuation_coefficient	noon_surface_light	length_of_day
1995	165	US_SPARK	0.351	1980	16

C.3 Sheet 2: benthic_photo_data

This sheet holds data specifically relating to benthic photosynthesis.

Table C.4: "benthic_photo_data" Template, empty

year	day_of_year	lake_ID	light_penetration_proportion	benthic_pmax	benthic_ik

Table C.5: "benthic_photo_data" Template, empty

year	day_of_year	lake_ID	light_penetration_proportion	benthic_pmax	benthic_ik
1995	165	US_SPARK	1	14.75637037	404.943
1995	165	US_SPARK	0.50	57.98165587	238.6559726
1995	165	US_SPARK	0.25	36.7229998	154.9128285
1995	165	US_SPARK	0.10	21.87757561	103.6069722
1995	165	US_SPARK	0.01	0.0012484529	73.33396613

C.4 Sheet 3: phytoplankton_photo_data

This sheet holds data specifically relating to phytoplanktonic photosynthesis.

TABLE C.6: "phyto_photo_data" sheet, empty

year	day_of_year	lake_ID	thermal_layer	depth	phyto_pmax_biomass	phyto_alpha	phyto_beta

TABLE C.7: "phyto_photo_data" sheet, filled

year	day_of_year	lake_ID	thermal_layer	depth	phyto_pmax_biomass	phyto_alpha	phyto_beta
1995	165	US_SPARK	1	5.0242825017	4.113867544	0.05	0.01
1995	165	US_SPARK	2	10.0322137967	2.636765965	0.0412667109	0
1995	165	US_SPARK	3	20.118170781	4.959339837	0.1646230769	0

C.5 Sheet 4: shape_data

.

This sheet holds data specifically relating to the shape of the lake basin.

Table C.8: "shape_data" sheet, empty

lake_ID	depth	water_surface_area

Table C.9: "shape_data" sheet, filled

lake_ID	depth	water_surface_area
US_SPARK	0	640000
US_SPARK	1	600960
US_SPARK	2	565120
US_SPARK	3	536960
US_SPARK	4	516480
US_SPARK	5	492800
US_SPARK	6	467840
US_SPARK	7	442880
US_SPARK	8	421120
US_SPARK	9	404480
US_SPARK	10	374400
US_SPARK	11	345600
US_SPARK	12	318720
US_SPARK	13	293120
US_SPARK	14	268800
US_SPARK	15	243200
US_SPARK	16	190720
US_SPARK	17	130560
US_SPARK	18	74240
US_SPARK	19	33280
US_SPARK	20	0

Parameter addition checklist

Checklist used to ensure all relevant classes and templates are updated properly when a parameter is added to the model.

```
#########################################################################
#Places that must be updated when adding parameters to pond
#########################################################################

#say parameter name is "parameter_name"...

#example_data.xls
        __put in a column (OR COLUMNS, IF PRIMARY KEY) for it. Title it "parameter_name"
        #if you put it on multiple sheets, try to have it in the same position

#template.xls
        __put in a column for it (OR COLUMNS, IF PRIMARY KEY), IN THE SAME LOCATION(s) AS EXAMPL


#data_reader.py
        __ data index (position in spreadsheet, BASED ON THE FILES ABOVE):  parameter_name_index
        __ "make pond objects" section/shape data section/Benthic data section/Phytoplankton dat
                        row_parameter_name_value = row[self.parameter_name_index].value
                        pond = Pond(row_parameter_name_value, row_year_value, row_lakeID_value,
                        #IF IT IS PART OF THE PRIMARY KEY, double-check to make sure all "are th
                        pond = next((i for i in pondList if (i.get_parameter_name()== row_parame




#pond.py
        __variables section: parameter_name = "", or parameter_name=[], or parameter_name={} or
        __constants section, for max/min values: PARAMETER_NAME_MAX_VALUE = 5, PARAMETER_NAME_MI
        __getter*
        __setter*
        __deleter*
        __validator (if numerical value)
        __constructor:     def __init__(self,...,parameter_name = ""):
                                              self.set_parameter_name(parameter_name)
        __everywhere that uses the constructor (find all references to "Pond" object?).
```

```
*Once you've made the variable, pyDev can add all these automatically using the "add properties"



#flask_app.py
        __index_view:
                make list: pond_parameter_name_list =[]
                append copies to list in loop:
                add it to the session dict, like so: session['pond_parameter_name_list'] = pond_
        __export_view:
                add a column index for it: parameter_name_column = some_other_parameter_column+1
                get it out of the session dict: parameter_name_list = session['pond_parameter_na
                write it to worksheet: write_column_to_worksheet(worksheet, parameter_name_colum

#primary_production.html
        __print it out, from session dict: session['parameter_name'][i]
```

Program requirements

Automatically-generated requirements using:

```
pip freeze
```

in the server virtualenv.

```
Cheetah==2.4.4
Flask==0.10.1
Jinja2==2.8
Landscape-Client==14.12
MarkupSafe==0.23
MySQL-python==1.2.3
PAM==0.4.2
Pillow==2.3.0
PyYAML==3.10
Pygments==1.6
Sphinx==1.2.2
Twisted-Core==13.2.0
Twisted-Names==13.2.0
Twisted-Web==13.2.0
Werkzeug==0.11.1
apt-xapian-index==0.45
argparse==1.2.1
beautifulsoup4==4.2.1
chardet==2.0.1
cloud-init==0.7.5
colorama==0.2.5
configobj==4.7.2
cvxopt==1.1.4
decorator==3.4.0
docutils==0.11
gunicorn==17.5
html5lib==0.999
ipython==1.2.1
itsdangerous==0.24
joblib==0.7.1
jsonpatch==1.3
```

```
jsonpickle==0.9.2
jsonpointer==1.0
matplotlib==1.3.1
meld3==0.6.10
nose==1.3.1
numexpr==2.2.2
numpy==1.8.2
numpydoc==0.4
oauth==1.0.1
openpyxl==1.7.0
pandas==0.13.1
patsy==0.2.1
pexpect==3.1
prettytable==0.7.2
pyOpenSSL==0.13
pycurl==7.19.3
pygobject==3.12.0
pyparsing==2.0.1
pyserial==2.6
python-apt==0.9.3.5ubuntu1
python-dateutil==1.5
python-debian==0.1.21-nmu2ubuntu2
pytz==2012c
pyzmq==14.0.1
requests==2.2.1
roman==2.0.0
scipy==0.13.3
simplegeneric==0.8.1
simplejson==3.3.1
six==1.5.2
ssh-import-id==3.21
statsmodels==0.5.0
stevedore==0.14.1
supervisor==3.0b2
sympy==0.7.4.1
tables==3.1.1
tornado==3.1.1
urllib3==1.7.1
virtualenv==1.11.4
virtualenv-clone==0.2.4
virtualenvwrapper==4.1.1
wheel==0.24.0
wsgiref==0.1.2
```

```
wxPython==2.8.12.1
wxPython-common==2.8.12.1
xlrd==0.9.2
xlwt==0.7.5
zope.interface==4.0.5
```

Bibliography

Baird, M. (2001). Technical Description of the Ecological Model.

Carpenter, S. (1983). Lake geometry: implications for production and sediment accretion rates. *Journal of Theoretical Biology*.

Devlin, S. (2015). Littoral-benthic primary production estimates: Sensitivity to simplifications with respect to periphyton productivity and basin morphometry. *Limnology and Oceanography: Methods*.

Fee, E. J. (1969). A numerical model for the estimation of photosynthetic production, integrated over time and depth, in natural waters. *Limnology and Oceanography*, 14(6):906–911.

Fee, E. J. (1971). Fee, Everett J; "Digital Computer Programs for Estimating Primary Production, Integrated Over Depth and Time, In Water Bodies." 1971: 49. Web. 16 May 2015.

Frenette, J. and Demers, S. (1993). Lack of agreement among models for estimating the photosynthetic parameters. *Limnology and . . . .*

Jassby, A. and Platt, T. (1976). Mathematical formulation of the relationship between photosynthesis and light for phytoplankton. *qyoiukn.aslo.net*.

Lindeman, R. L. (1942). The Trophic-Dynamic Aspect of Ecology. *Ecology*, 23(4):399–417.

NTL (2008). North Temperate Lakes LTER: Primary Production - Trout Lake Area 1986 - 2007 | North Temperate Lakes.

Platt, T., Gallegos, C., and Harrison, W. (1980). Photoinhibition of photosynthesis in natural assemblages of marine phytoplankton. *Journal of Marine Research*, 38:687 – 701.

Rueter, J. P. S. U. (2008). Constructing a P vs. I curve.

USGS (1997). Why Primary Production is Important.

Vadeboncoeur, Y. and Jeppesen, E. (2003). From Greenland to green lakes: Cultural eutrophication and the loss of benthic pathways in lakes. *Limnol. ...*.

Vadeboncoeur, Y., Lodge, D., and Carpenter, S. (2001). Whole-lake fertilization effects on distribution of primary production between benthic and pelagic habitats. *Ecology*.

Vadeboncoeur, Y., Peterson, G., Vander Zanden, M. J., and Kalff, J. (2008). Benthic algal production across lake size gradients: interactions among morphometry, nutrients, and light. *Ecology*, 89(9):2542–52.