Maximize lipid output and biomass production of algae in open pond batch systems for biofuel synthesis

New Mexico

Supercomputing Challenge

Final Report

April 2nd, 2014

Team 60

Las Cruces YWiC

Team members:

Ian Rankin

Ahmed Muhyi

Sophia Sanchez- Maes

Table of Contents

1.0 - Abstract

Green microalgae is widely regarded as a promising biofuel, given it's high lipid yield per area in comparison to other oleaginous crops. The commercial realization of algae biofuel synthesis on a large scale depends primarily on the lipid content and biomass production of the system. In this study, we further explore the optimization of such a system through the use of mathematical models to simulate batch culturing of algae, such as those being explored for continued commercial venture in NM. A computational model was developed to model growth dynamics and neutral lipid synthesis of the green microalgae Pseudochlorococcum sp., grown in batch cultures. Correspondence was established between the computational model and experiment, thus establishing the credibility of subsequent data. Data is collected in the interest of exploring the industrial optimization of algae production with respect to light and nutrients. Lipid synthesis is maximized with 4 hour batch replacement time, with respect to light and nutrients. The computational model will serve to assist industry in establishing commercial feasibility for algae biofuel. The model will further assist the scientific understanding of algae growth dynamics.

2.0 -Background

2.1 Motivation

Aside from it's obvious potential for working towards the pressing national security concern of energy independence, algae are also lauded as the most promising biofuel feedstock to date. With its high lipid yield and simple growth requirements, it's estimated that the average acre of algae can produce 5,000 gallons of biodisel per year, as compared to corns meager 420 gallons of ethanol yearly on the same acre. Additionally, algae's use as a biofuel does not interfere with the food supply. Furthermore, algae can thrive on land unsuitable for other agriculture. Although frequently parched, the one thing we're never short on here in NM is sunshine, which puts our state in one of the most lucrative possible positions for the production of biofuel.

The promise of this potential fuel source is obvious, however, it's not currently commercially viable. In order to become widespread, algae based fuel must compete with fossil fuel. In order to meet this competition, the production of algal biofuel must be optimized at all

phases in the process. We have decided to focus on optimization at it's start, by maximizing algal lipid yield, thereby increasing biofuel output.

2.2 Describing growth dynamics, recent findings

In order to further our goal, and make novel contribution to current research, we first had to understand what we were to be building upon. Numerous researchers have worked on the maximization of lipid yield using methods ranging from nutrient deprivation, to artificial selection. We were particularly interested in the relationships of light and nutrients to lipid yield. This is particularly applicable, as these are considerations faced by industry in the commercialization of biofuels. Researchers have found that lipid production increases when algae are under nutrient stress, such as nitrogen limitation. Additionally, lipid synthesis has been shown to increase with increasing incident irradiance. These correlations are something that we wish to quantify within our computational model, so we can maximize lipid output with respect to those parameters.

3.0 -Mathematical Model for algae growth and lipid production

The mathematical model explored in our project is one based upon the model presented by Dr. Aaron Packer in his article "Growth and neutral lipid synthesis in green microalgae". The functional model serves to calculate the biomass production and neutral lipid synthesis of green microalgae, both integral to the production of algae biofuel. Variable conditions are modeled through the manipulation of input variables. The model allows for the manipulation of two factors in particular, the initial concentration of nitrogen, N(0), and the incident irradiance, I_0. Local seasonal conditions can be accounted for using area values for these parameters.

Our implementation of this model hinges upon, 1. The system of differential equations, 2. Our understanding of the biological intricacies of the model, 3. The exploration and implementation of numerical methods to evaluate the system of differential equations presented in the model, 4. using this understanding in fitting results with experimental data for accuracy.

The model consists four state variables.

A(t)- the biomass concentration excluding neutral lipids (g dw m^{-3})

L(t)- the neutral lipid concentration (g NL m^{-3})

H(t)- chl a content of A (g chl^{-1} dw)

N(t)- the extracellular nitrogen concentration (g N m^{-3}).

3.1 ODEs

The behavior of these state variables with respect to our variable conditions is modeled mathematically through a system of four Ordinary Differential Equations (ODEs).

$$\frac{dA(t)}{dt} = \underbrace{\mu(A,L,H,N)A(t)}_{\text{cell growth}},$$

$$\frac{dL(t)}{dt} = \underbrace{[p(A,L,H,N) - c\mu(A,L,H,N)]A(t)}_{\text{NL synthesis}},$$

$$\frac{dH(t)}{dt} = \underbrace{c\frac{\mu}{p}(A,L,H,N)\rho v(A,N)}_{\text{N uptake devoted to chl a synthesis}} - \underbrace{H(t)\mu}_{\text{growth dilution}},$$

$$\frac{dN(t)}{dt} = \underbrace{-v(A,N)A}_{\text{N uptake}},$$

At left: ODEs from Packer [10]

These four equations are in turn dependant on an entire array of parameters, many of which can be treated as constants because they can be calculated directly from photosynthesis equations. The primary variables we are interested in are the interactions between the initial nitrogen concentration and the lipid output, as well as incident irradiance (light) and lipid output.

Solving the ODEs was a huge problem within the mathematical model. After numerous ugly, and ultimately futile, attempts to solve the system for a nice general solution, we turned to numerical methods of evaluation. These included several Euler methods, which, as we learned in calculus class, hinge upon repeated summation to estimate that area under the curve that we're looking to integrate. This process, and its corresponding code, is further detailed in section 4.4.

$$Q(t) = \frac{A(0)Q_0 + N(0) - N(t)}{A(t)},$$

$$\mu(A,L,H,N) = \min\left\{\mu_m\left(1 - \frac{q}{Q(t)}\right), \frac{p(A,L,H,N)}{c}\right\},$$

$$p(A,L,H,N) = H(t)p_m(A,L,N)\left(1 - \exp\left(\frac{-a\Phi I(A,H)}{p_m(A,L,N)}\right)\right),$$

$$p_m(A,L,N) = \frac{(AQ)^2(t)p_0}{(AQ)^2(t) + q^2(A(t) + L(t))^2},$$

$$I(A,H) = \frac{I_0}{aH(t)A(t)z}(1 - \exp(-aH(t)A(t)z)),$$

$$v(A,N) = \frac{q_M - Q(t)}{q_M - q}\left(\frac{v_m N(t)}{N(t) + v_h}\right).$$

At left: Parameters for ODEs [10]

Below: Explanation of Variables [10]

Parameter	Description	Units
I_0	Incident irradiance	mol photons m^{-2} d^{-1}
z	Light path	m
a	Optical cross section of chl a	m^2 g^{-1} chl
Φ	Quantum efficiency	g C (mol photons) $^{-1}$
q	Minimum/subsistence N quota	g N g^{-1} dw
q_M	Maximum N quota	g N g^{-1} dw
c	C subsistence quota	g C g^{-1} dw
v_m	Maximum uptake rate of nitrogen	g N g dw^{-1} d^{-1}
v_h	Half-saturation coefficient	g N m^{-3}
ρ	Maximum chl:N	g chl a g^{-1} N
μ_m	Maximum N-limited growth rate	d^{-1}
p_0	Maximum photosynthesis rate	g C g^{-1} chl d^{-1}

3.2 Biological relationships

As evidenced by the above figures, specifically, the explanation of variables, there are numerous biological intricacies involved in the model. These equations have been derived from a number of biological models, such as the Poisson single hit model of photosynthesis, the Lamber-Beer

law, and other models establish the basis for all considerations which are accounted for in the model. It's important to understand that the mathematical model is a conglomeration of various models. This is of assistance as we collect data from the computational model, because all results must be explained with respect to this biological basis.

3.3 Verification

The mathematical model was verified for its accuracy by Packer [10]. It demonstrated close alignment with experimental data, except at high values for initial nitrogen concentration. This can be explained by the existence of an additional limiting factor on algae growth. This verification based on experimental data speaks to the accuracy of the resulting computational model.

4.0 -Computational Model

Java was chosen for the project due to it's higher math functions and object orientation. Our model consists of a number of different classes, 1456 lines of code in total.

4.1AlgaeGrowthModel

AlgaeGrowthModel is the main class that holds the algae growth model. It contains all of the methods inside of the differential equations used. It also holds all of the state variables needed to run a simulation. The primary method in AlgaeGrowthModel is the update() call. This update call moves the state variables to the next time slot of the differential equation. Psudocode for this is.

```
public void update()    {
   EulersMethods.eulerMethod(StateVariableX)
// moves StateVariableX to the next time slot
}

private class StateVariableX implements Derivative {
 private double stateVariableX;
}
```

4.2 Pond

Pond is an abstract class that can not be instantiated without being extended by another class. We use the pond class to actually run the growth models.

Psudocode for this is.

```
for (int i = 0; i < (numDaysToRun / stepSize); i++)    {
 model.update();
}
```

4.3 Derivative

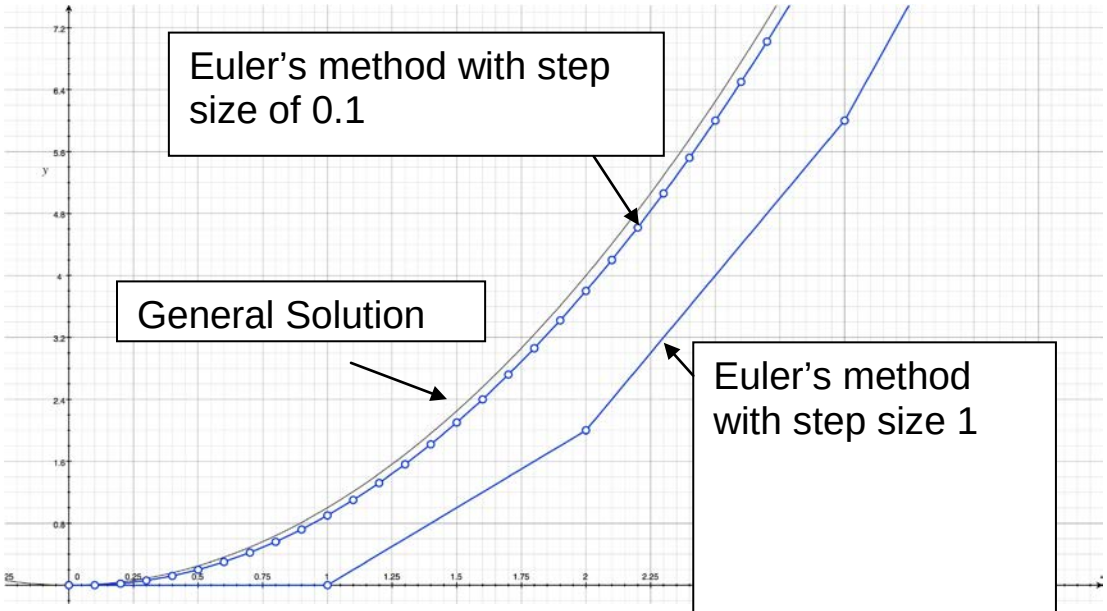
This is an Interface to allow the Euler methods to change the state variables. Code for this is,

```
public interface Derivative {
   public double findDerivative(double increaseStepSize);
   public double getStateVariable();
   public void setStateVariable(double value);
   public void move(double stepSize);
}
```

4.3 EulerMethods

This is a class that is just a container for functions. It holds no data and should only be instantiated once. A very important set of functions inside of here are the Euler's methods. We use the Euler methods to solve the ODE's in the mathematical model. Euler's method works by finding the slope or derivative of the equation at the current time slot and then uses the current slope to predict the value at current time slot + step size. If the equation dy(x)/dx = 2x, then y = x^2, then we solve for y using the Euler methods with an initial value of 0 for both x and y, and a step size of 1. The first time it runs, x = 0, therefore the slope equals 0. We then multiply by the step size, which is 1, and 1*0 equals 0. The next step is x = 1 and y = 0. This time it runs the slope is 2 because dy(x)/dx = 2(1). We then predict that the value of y at x = 2 is y + (slope * step size) is 2. The next time it runs the slope is 4. To move to x = 3, then we add 4*step size(1) + past y(2) = current y(6). When x = 3, the general solution of y = x^2, will equal 9. This is greater than the 6 we found using the Euler's methods. If we use a smaller step size with a step

size of 0.1 we now get a numerical solution of 8.45. As you get a smaller and smaller step size that number becomes infinitely close to the actual solution.



Other numerical methods we coded are the Hu lpoint method. These are improvements on the original Euler's method that allow you to get the same accuracy of a solution with a larger step size. Huen's method works by finding the slope and approximating the next time slot just like Euler's method does. After it figures out that value it then comes up with what it thinks the slope is there, and then averages between the two slopes and then goes back to the last time slot and uses the averaged slope.

Euler's Midpoint method works similar to Huen's in that it approximates what it thinks the values should be in a different location and then uses the slope from those approximate values to recalculate the slope. The difference is that it approximates the midpoint of the step size and then uses the approximate slope found at the midpoint and then uses it over the whole step size. Here is the code for the Euler's method:

```
public void eulersMethod(double stepSize, Derivative... dEquations)    {
      double[] slopes = new double[dEquations.length];

    // finds the slopes
     for (int i = 0; i < dEquations.length; i++)    {
```

```
            slopes[i] = dEquations[i].findDerivative(0);
        }

        // sets the state variables
        for (int i = 0; i < dEquations.length; i++)    {
            dEquations[i].setStateVariable(dEquations[i].getStateVariable()
                    + (slopes[i] * stepSize));
            dEquations[i].move(stepSize);
        }
    }
```

4.5 Graphics

The graphics classes allow the program to generate a graph of the simulation results. By adjusting the code, the window can be adjusted. The program assigns different colors to each line on the graph, corresponding to one of the functions being graphed with respect to time.

4.6 Parallelization

The results found in this project required thousands of individual Simulations be run. This starts taking very large amounts of time. By running them in parallel we can harvest the full potential of modern computers by using all cores instead of only a single core; making run times significantly shorter.

For our parallelization, batch processing was used. Batch processing allows us to run multiple independent Simulations at the same time in batches. By using the Threads built into Java we had a very efficient way to use Parallelization as it taps into the core threads that are built into the operating system. These threads are what the bottom level to tell the processors exactly what to run. For our parallelization we wrote a few different classes detailed below.

4.6.1 Simulation

This class is what contains the thread and gives us a basic API to be able to use the thread easily in other places. It holds the Pond object, the Thread, and the name of the simulation inside. The

Pond object as mentioned before is Runnable which makes it possible for the thread to run the code needed to simulate the growth model. The basic API is,

```
public Simulation(Pond pond, String name)
public void start()
public void waitForCompletion() throws InterruptedException
public boolean isFinished()
public Pond getPond()
public AlgaeGrowthModel getModel()
public String getName()
```

4.6.2 SimulationScheduler

SimulationScheduler is where the public API is to handle large amounts of simulations. SimulationScheduler has an method called add this adds the simulation into a Queue that sets up which simulations actually need to be ran. This Queue is set up as a FIFO for the simulations First In First Out. This Queue is actually stored in the SimulationDataHandler detailed below. The data is then actually processed in a separate class called RunnableScheduler. These three Scheduler classes are in a different Package called scheduler. SimulationScheduler's API is,

```
public SimulationScheduler(Recorder recorder)    // constructor lets you set which
//recorder you want to use
public void add(Simulation sim)       // this is how you add the simulations into //the
program
public void add(String simData)       // This lets you add just the data you need to //start
the program as a String
public Recorder getRecorder()
public void waitForFinish() throws InterruptedException    // This lets you stop //running
while it is processing all of the simulations you added
```

4.6.3 SimulationDataHandler

This class contains all of the data that needs to be passed between the SimulationScheduler and the RunnableScheduler. This class's data is protected. This allows SimulationScheduler and RunnableScheduler to both be able to access the data but not allow any other classes to be able to muck with the internals of the scheduler. Data stored in here is,

```
protected Queue<Object> toDo;
protected Recorder recorder;
```

The Queue holds a type called Object. This type is the generic pointer Java uses to point to a class or object. This type is used to allow us to hold both Strings, and Simulations inside the Queue. What is done with this Object will be detailed below.

4.6.4 RunnableScheduler

This is the part of the scheduler that actually handles starting and stopping the individual Simulations. It only runs 8 Simulations at a time to reduce the amount of memory used in by the growth models. We choose 8 simulations as the computer that the models would be ran on is a quad core with 2 virtual cores in each core. So the computer acts as an 8 “virtual” core computer. It then continually checks each of these 8 simulations stored in an array to see if it has finished running yet. If it has finished running then it sends that simulation to the Recorder which will be detailed below and then pulls the next Simulation to run out of the Queue. When it pulls the next Simulation out it actually pulls out a type called Object. This then can have checks ran to see what type of object it is. We then check to see if it is an Simulation or a String. If it is an Simulation it just sends it straight to the running array of Simulations. If it is a String it first must create the Simulation and give the Pond inside it the string data so that it can setup the simulation. Why it is done like that is because by using the string to tell the Pond the initial setup. The memory of the Simulation does not have to be allocated until it is ready to be run. greatly reducing the amount of memory required by the entire program. Once Runnable Scheduler see’s that both the Queue and its array of running simulation is empty. It stops itself until another Simulation is added to the SimulationScheduler.

5.7 Recorder

The actual class called Recorder in the code is just an interface that has the code,

```
public interface Recorder {
public void record(Simulation sim);
public String toString();
}
```

This should then have a class above it that implements it. By using the recorder we can record the small amount of data we actually need from running the simulation then remove the pointer to it. Java will then at some point come back and deallocate the memory to it, this is done

without us telling it to do so. This is actually a little bit of a problem with using java as we do not actually know when the Java Garbage collector will really come back and deallocate the memory from the heap.

5.0 -Methods

5.1 Experimental structure

Data is collected on the lipid production, biomass production and total algae growth as well as the total algae dry weight. The values of these are determined by the outputs of the computational model and graphed. Values for all initial parameters, but the one being tested are treated as constants using default values obtained from the article [10]. The model is set up to simulate batch culturing, with a time of 4 hours used between batches. The step size for the Euler methods was set to .05 for all trials. In the first experiment, initial nitrogen concentration is varied from 1-1000, incremented by one. This required thousands of simulation runs from which a maximum was determined.

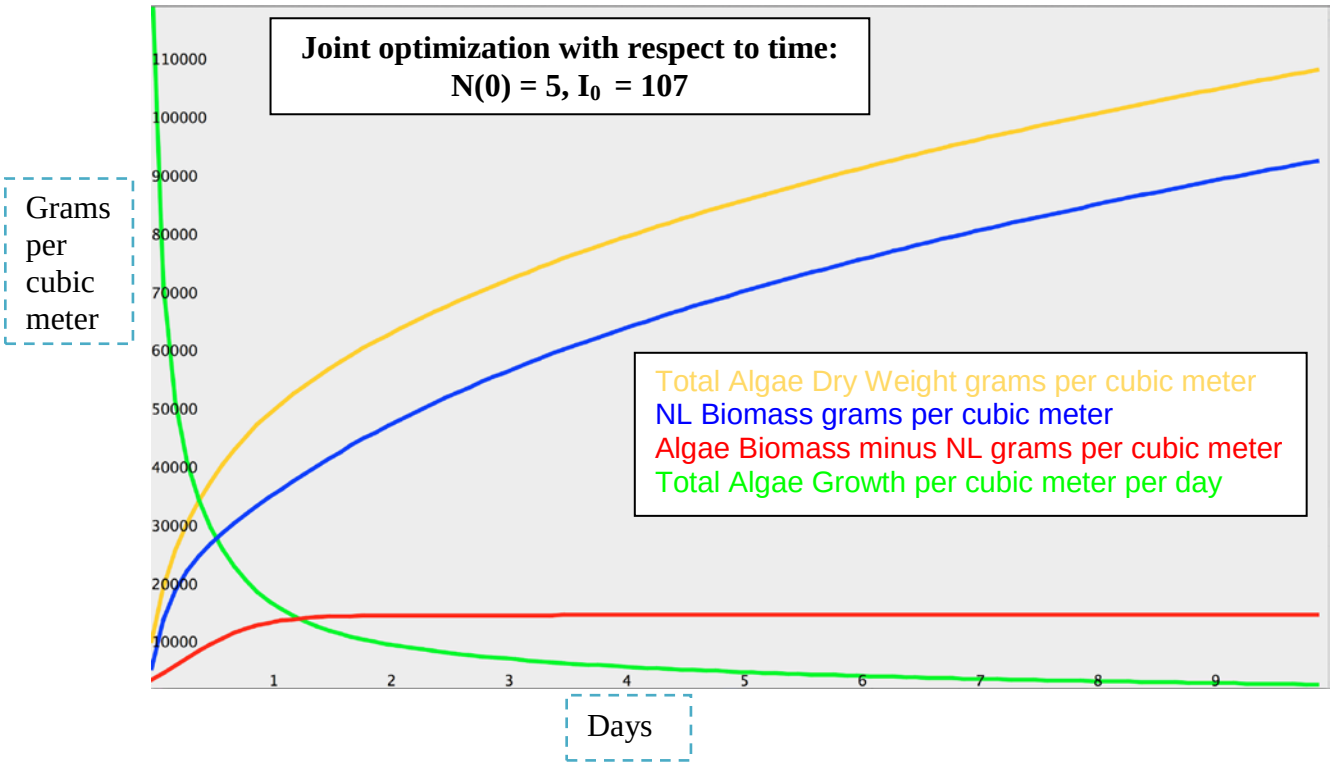
In the second experiment, incident irradiance is varied from 1-140 mol photons $m^{-2}d^{-1}$, incrementing by one, from the results of this set of simulations, another maximum was found. These maximums were put together so as to achieve a joint optimization under our parameters.

At left: Simulation default parameters as obtained from Packer [10].

Parameter	Value	Units
(a) Parameter values		
I_0	51.8	mol photon m^{-2} d^{-1}
z	0.03	m
a	4.82	m^2 g^{-1} chl
c	0.610	g C g^{-1} dw
q	0.0278	g N g^{-1} dw
q_M	0.0935	g N g^{-1} dw
v_m	5.96×10^{-1}	g N g^{-1} dw d^{-1}
v_h	1.03×10^{-5}	g N m^{-3}
ρ	0.283	g chl g^{-1} N
μ_m	3.26	d^{-1}
p_0	90.1	g C g^{-1} chl d^{-1}
Φ	9.84×10^{-2}	mol C mol^{-1} photon

6.0 -Results

	Initial A(t)	Initial NL(t)	Initial H(t)	Initial N(t)	Incident irradiance	Hours growing	Avg NL grams per cubic meter per day	Notes
1	34410	5	0.5	5	51.8	4.48	1.601082788837135E8	Degraded with large initial A(t)
2	4593	5	0.5	5	51.8	3.04	4925848.322539088	Optimized initial A(T)
3	4593	5	0.5	2409	51.8	0.07	1.0986E+07	Degraded with large initial N(t)
4	4593	5	0.5	5	51.8	3.04	4925848.322539088	Optimized initial N(T)
5	4593	5	0.499999	5	51.8	3.04	4925848.322539088	Optimized initial H(T)
6	4593	6569	0.5	5	51.8	3.36	5.757E+09	Optimized initial NL(T)
7	4593	6569	0.5	5	106	3.36	1.753E+10	Optimized Incident irradiance
8	4593	6574	0.45	5	105	12.24	2.103E+11	Optimized together



6.1 Conclusions

With 107 mol photons $m^{-2}d^{-1}$ as the lipid maximizing irradiance level, we investigate. This level of irradiance is relatively high. This high level of sunlight makes sense, because that energy serves to excite the electrons within the algae, and store energy in the form of ATP through photosynthesis. However, lipid yields decreased at higher irradiance. This is likely due to the effects of photo-inhibition, which reduces the photosynthetic capacity of the algae in response to light induced damage to photosystem II, which is light sensitive.

Table of Irradiance Values for Las Cruces, NM [2] : **Irradiance (mol photons $m^{-2}d^{-1)}$**

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sept
Irradiance	51.57	64.31	85.15	101.14	110.35	109.58	99.61	89.02	81.34
Month	Oct	Nov	Dec						
Irradiance	68.91	55.25	46.66						

Looking at the above table of local irradiance values shows us that April, May and June will produce the highest lipid yield here in an open system. Those bright sunny months have irradiance values closest to optimum and have higher lipid yield than the others.

Another option for producers, given this finding, is to augment natural light during the winter months to increase lipid yield. This could be accomplished through supplementing the light intensity using artificial light, or by increasing the winter photoperiod artificially.

The optimum nitrogen value of 5 g N m^{-3} , a seemingly low value, can also be explained biologically. As explained in section 2.2, nitrogen limitation has been shown to increase lipid production. This could be due to the metabolic pathway that causes algae to normally accumulate lipids during the cell cycle prior to cell division, and the stressed conditions block cell cycle progression, thereby increasing the overall lipid content of the culture.

7.0 -Achievements

We have successfully created a computational model of algal growth dynamics and lipid synthesis. We've established its correspondence with experimental data, and thus have verified its accuracy and real-world applicability. We have collected data optimizing lipid yield under our assumptions. These relationships are logical under biological rationale, further pointing to the success of the computational model.

Using our model, area-specific data can be collected to assist the optimization of algae commercially. This is achieved through the modification of the input variables, including incident irradiance, light angle, and nutrient levels. The resulting data will assist in the industrial engineering of open pond batch culture algae farms. Much remains to be discovered in regards to algal population dynamics, and this model will aid in the scientific understanding of lipid synthesis with respect to light and nutrients; an understanding that will continue to assist the biofuel industry.

8.0 -Future Work

All optimization that can be achieved with our current model is production optimal, that is, our model will maximize biofuel output by maximizing lipid production in a given scenario. However, industry requires another level of optimization: economic optimization. The economic model, to be an extension of our current model, will take a look at the bigger picture of algae production in the form of the bottom line. This part will utilize data on the production scenarios, such as the cost required to sustain conditions specified by the input variables of the previous model. It will utilize this data to estimate cost per gallon of the algae oil in that production scenario. This will allow us to create an economic optimum, so that we might further verify the commercial viability of out production optimums, as the larger goal is to assist in the commercial viability of algae fuels.

9.0 -Appendix

9.1 Acknowledgements

We would like to thank our parents, as well as our teacher Rebecca Galves and coach Noor Muhyi for their contributions to our cause. We would also like to thank NMSU professors Dr. Shuguang Deng from the Chemical Engineering Department, and Dr. Martha Mitchell, NMSU College of Engineering Associate Dean of Research. Additional thanks to Dr. Nagamany Nirmalakhandan of the Civil Engineering department, and Dr. Peter Lammers, technical director of the Algal Bioenergy Program, for assisting us.

9.2 References

[1] - Bouterfas, R., Belkoura, M., & Dauta, A. (2006). *The effects of irradiance and photoperiod on the growth rate of three freshwater green algae isolated from a eutrophic lake*. Retrieved from http://www.limnetica.com/Limnetica/Limne25/L25b647_Effect_irradiance_photoperiod_growth_rate.pdf

[2] - Boxwell, M. (n.d.). *Solar Irradiance.* Retrieved from http://solarelectricityhandbook.com/solar-irradiance.html

[3] - Demirbas, A., & Demirbas, M. F. (2011). Importance of algae oil as a source of biodiesel. *Energy Conversion and Management*, *52*(1), 163-170. Retrieved from http://www.sciencedirect.com/science/article/pii/S0196890410002761

[4] - Gao, Y. , Gregor, C., Liang, Y., Tang, D., Tweed, C. (2012). *Algae biodiesel - a feasibility report.* Retrieved from http://journal.chemistrycentral.com/content/6/S1/S1

[5] - Griffiths, M. J., & Harrison, S. T. L. (2008, July 31). *Lipid productivity as a key characteristic for choosing algal species for biodiesel production*. Retrieved from http://link.springer.com/article/10.1007/s10811-008-9392-7

[6] Hannon, M., Gimpel, J., Mayfield, S. Biofuels 1(5), 763-784. (2010) Biofuels from algae: Challenges and potential. Retrieved from http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3152439/

[7] - Jones, C., & Mayfield, S. (2012). Algae biofuels: Versatility for the future of bioenergy. *Current Opinion in Biotechnology*, *23*(3), 346-351. Retrieved from http://www.sciencedirect.com/science/article/pii/S0958166911007099

[8] - Liu, X., Clarens, A. F., & Colosi, L. M. (2012). Algae biodiesel has potential despite inconclusive results to date. *Bioresource Technology*, *104*, 803-806. Retrieved from http://www.sciencedirect.com/science/article/pii/S0960852411015653

[9] - Mata, T. M., Martins, A. A., & Caetano, N. S. (2010). Microalgae for biodiesel production and other applications: A review. *Renewable and Sustainable Energy Reviews*, *14*(1), 217-232. Retrieved from http://www.sciencedirect.com/science/article/pii/S1364032109001646

[10] - Packer, A., Li, Y., Andersen, T., Hu, Q., Kuang, Y., & Sommerfeld, M. (2011). Growth and neutral lipid synthesis in green microalgae: A mathematical model. *Bioresource Technology*, *102*(1), 111-117. Retrieved from http://www.sciencedirect.com/science/article/pii/S0960852410010102

[11] - Scott, S. A., Davey, M. P., Dennis, J. S., Horst, I., Howe, C. J., Lea-Smith, D. J., & Smith, A. G. (2010). Biodiesel from algae: Challenges and prospects.*Current Opinion in Biotechnology*, *21*(3), 227-286. Retrieved from http://www.sciencedirect.com/science/article/pii/S0958166910000443

[12] - Sheehan, J., Camobreco, V., Duffield, J., Graboski, M., & Shapouri, H. (1998). *Life cycle inventory of biodiesel and petroleum diesel for use in an urban bus.* Retrieved from http://www.nrel.gov/docs/legosti/fy98/24089.pdf

[13] - Sheehan, J., Dunahay, T., Benemann, J., & Roessler, P. U.S. Department of Energy, (1998). *A look back at the u.s. department of energy's aquatic species program: Biodiesel from algae*. Retrieved from National Renewable Energy Labratory website: http://www.nrel.gov/biomass/pdfs/24190.pdf

9.3 Code

```
/* This Class is a class to allow us to be able to graph the algae growths
*/

package Graphics;

import java.awt.Color;
import java.util.ArrayList;
import javax.swing.*;

public class Grapher {

public static final int WINDOW_X = 1400;
public static final int WINDOW_Y = 850;
public static final int MAX_Y = 120000;
public static final int MIN_Y = -50;
public static final int NUM_GRAPH_POINTS = 100;
public static final int NUM_Y_POINTS = 10000
;
public static final int Y_OFFSET = 25;
public static final Color[] colors = {Color.black, Color.red, Color.blue, Color.green,
Color.orange, Color.PINK, Color.yellow};
public Grapher() {
graphs = new ArrayList<double[]>();
frame = new JFrame();
frame.setSize(WINDOW_X, WINDOW_Y);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
addYLabels();
}
public void addYLabels() {
double yWindowRatio = (double) WINDOW_Y / (MAX_Y - MIN_Y);
for (int i = 0; i < MAX_Y; i+= NUM_Y_POINTS) {
String str = String.valueOf(i);
Label label = new Label(str, 0, (int)((MAX_Y - i) * yWindowRatio) ,Color.BLACK);
frame.add(label);
frame.setVisible(true);
}
}
public void addXLabels(int length) {
for (int i = 1; i < length; i++) {
String str = String.valueOf(i);
Label label = new Label(str, (i * (WINDOW_X / length)), WINDOW_Y - Y_OFFSET, Color.BLACK);
frame.add(label);
frame.setVisible(true);
}
}
public void addGraph(ArrayList<Double> graph)
{
double[] newGraph = new double[graph.size()];
for (int i = 0; i < graph.size(); i++) // converts ArrayList to a regular array
{
newGraph[i] = graph.get(i);
}
graphs.add(newGraph); // adds regular array to list of graphs
}
public void addGraph(double[] graph)
{
graphs.add(graph);
drawGraph(graph, WINDOW_X,WINDOW_Y, MAX_Y, MIN_Y);
}
// clears screen then goes through each graph and tells it to update
private void updateGraphs()
{
```

```
//clearScreen();
for (int i = 0; i < graphs.size(); i++)
{
drawGraph(graphs.get(i), 1,1, 10, -10);
}
}
private void drawGraph(double[] graph, int xWindowSize, int yWindowSize, double yMax, double yMin)
{
double yWindowRatio = (double) yWindowSize / (yMax - yMin);
double xWindowRatio = (double) xWindowSize / (graph.length);
//double xWindowRatio = (double) xWindowSize / (graph.length / 100);
double lastY = (yMax - graph[0]) * yWindowRatio;
double lastX = 0.0 / xWindowRatio;
//for (int i = 1/*(graph.length / 100)*/; i < graph.length; i++)//= (graph.length / 100))
for (int i = (graph.length / NUM_GRAPH_POINTS); i < graph.length; i += (graph.length / NUM_GRAPH_POINTS))
{
double plotY = (yMax - graph[i]) * yWindowRatio;
double plotX = (double) i * xWindowRatio;
frame.add(new Line((int) lastX,(int) lastY,(int) plotX,(int) plotY, colors[colorIndex]));
frame.setVisible(true);
lastX = plotX;
lastY= plotY;
}
colorIndex++;
if (colorIndex >= colors.length) colorIndex = 0;
}
private ArrayList<double[]> graphs;
private JFrame frame;
private int colorIndex = 1;
}
package Graphics;

import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JComponent;

public class Label extends JComponent{

public Label(String string, int xLocation, int yLocation, Color color) {
this.string = string;
this.xLocation = xLocation;
this.yLocation = yLocation;
this.color = color;
}
public void paintComponent(Graphics g) {
g.setColor(color);
g.drawString(string, xLocation, yLocation);
}
private String string;
private int xLocation;
private int yLocation;
private Color color;
}

package Graphics;

import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JComponent;

public class Line extends JComponent {

public Line(int xZero, int yZero, int xOne, int yOne, Color color)
{
```

```
this.xZero = xZero;
this.yZero = yZero;
this.xOne = xOne;
this.yOne = yOne;
this.color = color;
}
public void paintComponent(Graphics g)
{
g.setColor(color);
g.drawLine(xZero, yZero, xOne, yOne);
}
private int xZero;
private int yZero;
private int xOne;
private int yOne;
private Color color;
}

package GrowthModel;

/* All of these methods are based off of mathematical model we found in this paper
* http://www.ncbi.nlm.nih.gov/pubmed/20619638
*
* This class contains all of the functions and state variables needed to run the
* Simulations
*/

public class AlgaeGrowthModel {
/* This is the constructor that sets up the growth model
* You need to give it the concentrations or contents of algae, Neutral Lipid, Chlorophyll, Nitrogen. at the start
* of the simulation.
* Written Ian Rankin (10/29/13) */
public AlgaeGrowthModel(double biomassConc, double NLConc, double ChlContent, double NConc) {
currentBiomassConc = biomassConc;
startBiomassConc = biomassConc;
currentNLConc = NLConc;
startNLConc = NLConc;
currentChlContent = ChlContent;
startChlContent = ChlContent;
currentNConc = NConc;
startNConc = NConc;
biomassFunc = new dBiomassConc();
NLFunc = new dNLConc();
chlFunc = new dChlContent();
NFunc = new dNConc();
startQt = .0935;
hasStarted = false;
if (solver == null) {
solver = new EulersMethods();
}
}

//------------------------------------------------ Get Methods ----------------------------------------------
// since all of these are concentrations to get the mass they need to be multiplied by the volume. (I think)....
// this returns the dry weight (I think) of all of the algae in the pond
public double getAlgaeBiomass() {
return (currentBiomassConc + currentNLConc);
}
// This returns the concentration of the algae minus the neutral lipids
public double getBiomass() {
return currentBiomassConc;
}
// This returns the mass of the neutral lipids
public double getNLConc() {
return currentNLConc;
}
```

```
public double getChlContent() {
return currentChlContent;
}
public double getNitrogenConc() {
return currentNConc;
}
public double getIncidentIrradiance() {
return this.incidentIrradiance;
}
public double getStartBiomass() {
return startBiomassConc;
}
public double getStartNL() {
return startNLConc;
}
public double getStartChl() {
return startChlContent;
}
public double getStartN() {
return startNConc;
}
public double getStartQt() {
return startQt;
}
//_____________________________________________________ Set Methods
_____________________________________________
public void setStepSize(double stepSize) {
this.stepSize = stepSize;
}
public boolean setStartBiomass(double biomass) { // DO NOT USE UNLESS SETTING AT START
if (!hasStarted) {
startBiomassConc = biomass;
currentBiomassConc = biomass;
}
return !hasStarted;
}
public boolean setStartNL(double NL) { // DO NOT USE UNLESS SETTING AT START
if (!hasStarted) {
startNLConc = NL;
currentNLConc = NL;
}
return !hasStarted;
}
public boolean setStartChl(double Chl) { // DO NOT USE UNLESS SETTING AT START
if (!hasStarted) {
startChlContent = Chl;
currentChlContent = Chl;
}
return !hasStarted;
}
public boolean setStartN(double N) { // DO NOT USE UNLESS SETTING AT START
if (!hasStarted) {
startNConc = N;
currentNConc = N;
}
return !hasStarted;
}
public boolean setStartQt(double Qt) { // DO NOT USE UNLESS SETTING AT START
if (!hasStarted) {
startQt = Qt;
}
return !hasStarted;
}
public void setIrradiance(double irradiance) {
this.incidentIrradiance = irradiance;
}
```

```
public void setLightPath(double lightPath) {
this.lightPath = lightPath;
}
//---------------------------------------------- updating methods ----------------------------------------------
/* update()
* This is a basic function to update the model to the next time slot
* It calculates all of the main equations and them updates the current concentrations of the model */
public void update()
{
solver.eulersMidpointMethod(stepSize, biomassFunc, NLFunc, chlFunc, NFunc);
hasStarted = true;
}
//---------------------------------------- private classes for Euler solvers -----------------------------------
private class dBiomassConc implements Derivative {
public double findDerivative(double increaseStepSize) {
return getDCellGrowth();
}
public double getStateVariable() {
return currentBiomassConc;
}
public void setStateVariable(double value) {
currentBiomassConc = value;
}
public void move(double stepSize) {
}
}
private class dNLConc implements Derivative {

public double findDerivative(double increaseStepSize) {
return getDNLSynthesis();
}

public double getStateVariable() {
return currentNLConc;
}
public void setStateVariable(double value) {
currentNLConc = value;
}
public void move(double stepSize) {
}
}
private class dChlContent implements Derivative {

public double findDerivative(double increaseStepSize) {
return getDChlContent();
}

public double getStateVariable() {
return currentChlContent;
}
public void setStateVariable(double value) {
currentChlContent = value;
}
public void move(double stepSize) {
}
}
private class dNConc implements Derivative {

public double findDerivative(double increaseStepSize) {
return getDNitrogenConc();
}

public double getStateVariable() {
return currentNConc;
}
public void setStateVariable(double value) {
```

```
currentNConc = value;
}
public void move(double stepSize) {
}
}
//---------------------------------------- Derivative get Functions ----------------------------------------------
public double getDCellGrowth() {
return dCellGrowth(getMuALHN(), currentBiomassConc);
}
private double getDNLSynthesis() {
return dNLSynthesis(getPALHN(), CQuota, getMuALHN(), currentBiomassConc);
}
private double getDChlContent() {
double muALHN = getMuALHN();
return dChlContent(CQuota, muALHN, getPALHN(), maxChlN, getVAN(),
currentChlContent, muALHN);
}
private double getDNitrogenConc() {
return dNitrogenConc(getVAN(), currentBiomassConc);
}
// _____________________________________ other math functions
_____________________________________________
public double getQt() {
return Qt(startBiomassConc, startQt, startNConc, currentNConc, currentBiomassConc);
} // this is the one with Qt
private double getMuALHN() {
return muALHN(CQuota, minNQuota, maxNLimitedGrowth, getQt(), getPALHN());
}
private double getPALHN() {
return pALHN(getPmALN(), getIAH(), currentChlContent, opticalCross, quantumEfficiency);
}
private double getPmALN() {
return pmALN(currentBiomassConc, getQt(), maxPhotosynthesisRate, minNQuota, currentNLConc);
}
private double getIAH() {
return IAH(incidentIrradiance, opticalCross, currentChlContent, currentBiomassConc, lightPath);
}
private double getVAN() {
return vAN(maxNQuota, getQt(), minNQuota, maxUptakeN, currentNConc, halfSaturation);
}
//_____________________________________________ Formula's
_____________________________________________
// These are all public because they do not muck with the internals
// and then we can get the functions from outside of the object.
/* dCellGrowth
* Written Ian Rankin (10/25/13)
* return cell growth */
public double dCellGrowth(double muALHN, double currentBiomassConc)
{
return (muALHN * currentBiomassConc);
}

/* dNLSynthesis
* Written Ian Rankin (10/25/13)
* returns derivative of Neutral Lipid synthesis */
public double dNLSynthesis(double pALHN, double c, double muALHN, double currentBiomassConc)
{
return (pALHN - (c * muALHN)) * currentBiomassConc; // changed 4/1
}
/* dChlSynthesis
* Written Ian Rankin (10/29/13)
* returns derivative of Chlorophyll content */
public double dChlContent(double carbonQuota, double muALHN, double pALHN, double maxChl,
double vAN, double currentChlContent, double muALHNMaybe)
{
return (carbonQuota * (muALHN / pALHN) * maxChl * vAN) - (currentChlContent * muALHNMaybe);
```

```
}
public double dNitrogenConc(double vAN, double ASomething)
{
return -vAN * ASomething;
}

/* Qt
* Started Ian Rankin (10/25/13)
* returns Q(t) equation */
public double Qt(double biomassConcAtTimeZero, double QSubscriptZero, double nitrogenAtTimeZero,
double currentNitrogenConc, double currentBiomassConc)
{
return ((biomassConcAtTimeZero * QSubscriptZero) + nitrogenAtTimeZero - currentNitrogenConc)
/ currentBiomassConc;
}
public double muALHN (double c, double q, double um, double Qt, double pALHN)
{
return Math.min(um * ( 1 - (q / Qt) ), pALHN/c); // fixed 3/29 Sophia S.M. was Math.min(um * ( 1 - q / Qt ), pALHN/c);
}
public double pALHN (double pmALN, double IAH, double Ht, double a, double phi)
{ // Ht is dynamic variable H(t)
return Ht * pmALN * ( 1 - Math.exp((-a * phi * IAH)/pmALN));
}
/* pmALN
* written Ian Rankin (mostly) (11/12/13)
* returns pm(A,L,N) function */
public double pmALN (double At, double Qt, double pSubscriptZero, double q, double Lt)
{
double AQTSquared = (At * Qt) * (At * Qt); // may be wrong..
//double AQTSquared = Math.pow ( (At*Qt), 2 );

double denominator = AQTSquared + (q * q) * ((At + Lt) * (At + Lt)); // I think this is right (Ian)
//double denominator = (AQTSquared + ((q * q) * (At) +Lt)); // not sure if that is right (Ian 4/1/13)
//fixed denominator, was originally(AQTSquared + ((q * q) * At) +Lt) Sophia S.M. ;
//return (AQTSquared * pSubscriptZero) / (denominator * denominator);
return (AQTSquared * pSubscriptZero) / denominator;
}
public double IAH (double ISubscriptZero, double a, double Ht, double At, double z)
{
return (ISubscriptZero / (a * Ht * At * z)) * (1 - Math.exp(-a * Ht * At * z));
}
public double vAN(double qM, double Qt, double q, double vm, double Nt, double vh)
{
return ((qM - Qt) / (qM - q)) * (( vm * Nt )/ (Nt + vh)); // fixed 3/29 Sophia S.M.
}

//---------------------------------------- Instance Variables ------------------------------------------------
private double currentBiomassConc;
private double currentNLConc;
private double currentChlContent;
private double currentNConc;
private double startBiomassConc;
private double startNLConc;
private double startChlContent;
private double startNConc;

private dBiomassConc biomassFunc;
private dNLConc NLFunc;
private dChlContent chlFunc;
private dNConc NFunc;
private static EulersMethods solver;
private boolean hasStarted;
private double incidentIrradiance = GrowthModelConstants.DEFUALT_INCIDENT_IRRADIANCE;
private double lightPath = GrowthModelConstants.DEFUALT_LIGHT_PATH;
private double opticalCross = GrowthModelConstants.DEFUALT_OPTICAL_CROSS;
private double quantumEfficiency = GrowthModelConstants.DEFUALT_QUANTUM_EFFICIENCY;
```

```
private double minNQuota = GrowthModelConstants.DEFUALT_MIN_N_QUOTA;
private double maxNQuota = GrowthModelConstants.DEFUALT_MAX_N_QUOTA;
private double CQuota = GrowthModelConstants.DEFUALT_C_QUOTA;
private double maxUptakeN = GrowthModelConstants.DEFUALT_MAX_UPTAKE_RATE_N;
private double halfSaturation = GrowthModelConstants.DEFUALT_HALF_SATURATION;
private double maxChlN = GrowthModelConstants.DEFUALT_MAX_CHL_N; //??
private double maxNLimitedGrowth = GrowthModelConstants.DEFUALT_MAX_N_LIMITED_GROWTH;
private double maxPhotosynthesisRate = GrowthModelConstants.DEFUALT_MAX_PHOTOSYNTHESIS_RATE;
private double startQt;
private double stepSize = 1; // This is something to let us do more detailed simulations
}

package GrowthModel;

/* This is just and example pond abstract using a really basic model
* ALL PONDS must implement Runnable
*
*/

public class BasicAlgaePond extends Pond {
public BasicAlgaePond()
{
super(new AlgaeGrowthModel(1,1,1,1));
}
public BasicAlgaePond(AlgaeGrowthModel model) {
super(model);
}
/* run() is a unique method that MUST be in a pond abstract
* in fact if it isn't there the compiler will complain because anything
* that implements Runnable must have this method
*
* Run() is the staring place for when you start a Simulation it looks
* for the Run() method and starts there use it when you want to start
* the pond simulation
*/
public void run() {
model.setStepSize(.1);
for (int i = 0; i < timesToRun; i++) {
model.update();
}
}
private int timesToRun = 1200; // this is just a constant that could be changed if you wanted
}


package GrowthModel;

/* this is for the Euler methods
* using this allow you to use different differential equations for
* the Euler solvers */
public interface Derivative {
public double findDerivative(double increaseStepSize);
public double getStateVariable();
public void setStateVariable(double value);
public void move(double stepSize);
}


package GrowthModel;

public class EulersMethods {
/* Eulers method
* this calculates the slope at current time. then it predicts what the next time
* slot by multiplying the slope by the stepSize. */
public void eulersMethod(double stepSize, Derivative... dEquations) {
double[] slopes = new double[dEquations.length];
```

```
// finds the slopes
for (int i = 0; i < dEquations.length; i++) {
slopes[i] = dEquations[i].findDerivative(0);
}
// sets the state variables
for (int i = 0; i < dEquations.length; i++) {
dEquations[i].setStateVariable(dEquations[i].getStateVariable()
+ (slopes[i] * stepSize));
dEquations[i].move(stepSize);
}
}
/* Heuns Method
* this is an improvement of Eulers Method.
* It predicts the slope of the next time slot using Eulers method.
* it then avg's the slope of the current timeSlot with the predicted slope of the next Time slot
* and then uses the avg slope to calculate the next time slots "y" */
public void huensMethod(double stepSize, Derivative... dEquations) {
double[] currentValues = new double[dEquations.length];
double[] slopes = new double[dEquations.length];
// stores the current state variables and finds the slopes at the current time slot
for (int i = 0; i < dEquations.length; i++) {
currentValues[i] = dEquations[i].getStateVariable();
slopes[i] = dEquations[i].findDerivative(0);
}
// updates the state variables to the predicted values
for (int i = 0; i < dEquations.length; i++) {
dEquations[i].setStateVariable(dEquations[i].getStateVariable()
+ (slopes[i] * stepSize));
}
// finds the predicted slope of the next time slot and avg's the two
for (int i = 0; i < dEquations.length; i++) {
slopes[i] = (dEquations[i].findDerivative(stepSize) + slopes[i]) / 2;
}
// updates the state variables using the avg slope and replace predicted values
for (int i = 0; i < dEquations.length; i++) {
dEquations[i].setStateVariable(currentValues[i] + (slopes[i] * stepSize));
}
}
/* Eulers Midpoint Method
* this calculates the next time slot by finding a predicted mid point slope
* this is just like the Eulers method except it predicts the midpoint slope of the step size
* and uses it instead of the current slope */
public void eulersMidpointMethod(double stepSize, Derivative... dEquations) {
double[] currentValues = new double[dEquations.length];
double[] midSlopes = new double[dEquations.length];
// stores the current state variables
for (int i = 0; i < dEquations.length; i++) {
currentValues[i] = dEquations[i].getStateVariable();
}
// finds the predicted state variable values at the midpoint using Eulers method
eulersMethod((stepSize * .5), dEquations);
// finds the midpoint slopes
for (int i = 0; i < dEquations.length; i++) {
midSlopes[i] = dEquations[i].findDerivative(0);
}
// updates the state variables using the midpoint slope and replace predicted values
for (int i = 0; i < dEquations.length; i++) {
dEquations[i].setStateVariable(currentValues[i] + (midSlopes[i] * stepSize));
dEquations[i].move(stepSize * .5);
}
}
}

package GrowthModel;

public interface GrowthModelConstants {
```

```
// these default values are found out of the paper as values they used
public final static double DEFUALT_INCIDENT_IRRADIANCE = 51.8;
public final static double DEFUALT_LIGHT_PATH = 0.03;
public final static double DEFUALT_OPTICAL_CROSS = 4.82;
public final static double DEFUALT_QUANTUM_EFFICIENCY = 984;
public final static double DEFUALT_MIN_N_QUOTA = 0.0278;
public final static double DEFUALT_MAX_N_QUOTA = 0.0935;
public final static double DEFUALT_C_QUOTA = 0.61;
public final static double DEFUALT_MAX_UPTAKE_RATE_N = 59.6;
public final static double DEFUALT_HALF_SATURATION = 10300;
public final static double DEFUALT_MAX_CHL_N = 0.283;
public final static double DEFUALT_MAX_N_LIMITED_GROWTH = 3.26;
public final static double DEFUALT_MAX_PHOTOSYNTHESIS_RATE = 90.1;
}


package GrowthModel;
import java.util.*;

public class IncidentIrradianceFinder {

public IncidentIrradianceFinder()
{
iradanceValues = new HashMap<String, Double>();
iradanceValues.put("january", 3.36); // units are kWh/m^2/day
iradanceValues.put("febuary", 4.19);
iradanceValues.put("march", 5.55);
iradanceValues.put("april", 6.59);
iradanceValues.put("may", 7.19);
iradanceValues.put("june", 7.14);
iradanceValues.put("july", 6.49);
iradanceValues.put("august", 5.80);
iradanceValues.put("september", 5.30);
iradanceValues.put("october", 4.49);
iradanceValues.put("november", 3.60);
iradanceValues.put("december", 3.04);
}
public double getValue (String month)
{
month.toLowerCase();
return convertUnits (iradanceValues.get(month));
}
private double convertUnits (double irradiance)
{
return 0;// iradiance ();
}
/*
private double convertToJoulesToMole (double iradiance)
{
}
*/
private Map<String, Double> iradanceValues;
}


package GrowthModel;

abstract public class Pond implements Runnable {

public Pond(AlgaeGrowthModel model)
{
this.model = model;
}
@Override
abstract public void run();
public AlgaeGrowthModel model;
```

```
}


package GrowthModel;

public class PondForSimulationUse extends Pond {

// this has to be added to model the amount of time to harvest the first time we ran this optimazation
// we found that it was about 5 minutes would be the best amount of time. which would not work as it
// takes much longer to harvest the algae then 5 minutes
// this is set at 4 hours
// .1666666666667
public static final double TIME_TO_HARVEST = .16666666666667;
/* String data should be given as 2.2 4.6 1.1 8.2 10 .1 which is biomass NL Chl N daysToRun stepSize
* respectively
*
*/
public PondForSimulationUse(String data) {
super(new AlgaeGrowthModel(1,1,1,1));
String[] dataSplit = data.split(" ");
model.setStartBiomass(Double.parseDouble(dataSplit[0]));
model.setStartNL(Double.parseDouble(dataSplit[1]));
model.setStartChl(Double.parseDouble(dataSplit[2]));
model.setStartN(Double.parseDouble(dataSplit[3]));
model.setStartQt(Double.parseDouble(dataSplit[4]));
daysToRun = Double.parseDouble(dataSplit[5]);
stepSize = Double.parseDouble(dataSplit[6]);
if (dataSplit.length > 6) {
model.setIrradiance(Double.parseDouble(dataSplit[7]));
}
maxNLGrowthPerDay = -1;
maxTime = -1;
}
public double getNL() {
return model.getNLConc();
}
public double getMaxNLGrowthPerDay() {
return maxNLGrowthPerDay;
}
public double getMaxNLTime() {
return maxTime / (daysToRun/ stepSize);
}

public void run() {
model.setStepSize(stepSize);
for (int i = 0; i < (daysToRun / stepSize); i++) {
model.update();
if (growthPerDay(model.getNLConc(), i) > maxNLGrowthPerDay) {
maxNLGrowthPerDay = growthPerDay(model.getNLConc(), i);
maxTime = i;
}
}
}
private double growthPerDay(double NL, int time) {
double NLGrowth = NL - model.getStartNL();
double days = time / (daysToRun/ stepSize);
days += TIME_TO_HARVEST; // this accounts for that the time to harvest is not 0
if (days == 0) return -1;
return NLGrowth / days;
}

private double daysToRun;
private double stepSize;
private double maxNLGrowthPerDay;
private int maxTime;
}
```

```
package GrowthModel;

/* This pond is a basic pond simulation that has an array of data
* that has all of the values of algaeBiomass over the time of the simulation
*
* this also creates the model object
*/

public class PondWithGraphData extends Pond {

public PondWithGraphData(String data)
{
super(new AlgaeGrowthModel(.01,.01,.01,.25));
String[] dataSplit = data.split(" ");
int timesToRun = Integer.parseInt(dataSplit[0]);
double stepSize = Double.parseDouble(dataSplit[1]);
constructor(timesToRun, stepSize);
}
public PondWithGraphData(int timesToRun, double stepSize) {
super(new AlgaeGrowthModel(4593, 5, 0.5, 5)); // .01,.01,.01,.25
constructor(timesToRun, stepSize); // maybe right initial values 150,5,.01,250
}
public PondWithGraphData(int timesToRun, double stepSize, double A, double NL, double Chl, double N) {
super(new AlgaeGrowthModel(A, NL, Chl, N));
constructor(timesToRun, stepSize);
}

private void constructor(int timesToRun, double stepSize)
{
this.model.setStepSize(stepSize);
this.timesToRun = (int)((double)timesToRun / stepSize);
biomassData = new double[this.timesToRun];
biomassGrowthData = new double [this.timesToRun];
NLData = new double[this.timesToRun];
chlContentData = new double[this.timesToRun];
nitrogenConcData = new double[this.timesToRun];
daysToRun = timesToRun;
}
public void run() {
for (int i = 0; i < this.timesToRun; i++)
{
biomassData[i] = model.getBiomass();
biomassGrowthData[i] = model.getDCellGrowth();
NLData[i] = model.getNLConc();
chlContentData[i] = model.getChlContent();
nitrogenConcData[i] = model.getNitrogenConc();
model.update();
}
}
public double[] getTotalDryWeight() {
double[] algaeWeight = new double[biomassData.length];
for (int i = 0; i < biomassData.length; i++) {
algaeWeight[i] = biomassData[i] + NLData[i];
}
return algaeWeight;
}
public double[] getBiomassData() {
return biomassData;
}
public double[] getBiomassGrowthData() {
return biomassGrowthData;
}
public double[] getNLData() {
return NLData;
}
public double[] getChlContentData() {
```

```
return chlContentData;
}
public double[] getNitrogenConcData() {
return nitrogenConcData;
}
public double getDaysToRun() {
return daysToRun;
}
public void print() {
System.out.println("Biomass Data");
for (int i = 0; i < timesToRun; i++) {
System.out.println(biomassData[i]);
}
}

private double biomassData[];
private double biomassGrowthData[];
private double NLData[];
private double chlContentData[];
private double nitrogenConcData[];
private double daysToRun;
private int timesToRun;
}

package GrowthModel;

import java.util.ArrayList;

public class RecordBiomass implements Recorder {

public RecordBiomass() {
biomass = new ArrayList<Double>();
names = new ArrayList<String>();
}
public void record(Simulation sim) {
biomass.add(sim.getModel().getAlgaeBiomass());
names.add(sim.getName());
}
public String toString() {
String str = new String();
for (int i = 0; i < biomass.size(); i++) {
String line = names.get(i) + " - biomass = " + biomass.get(i);
str += line;
System.out.println(line);
}
return str;
}

private ArrayList<Double> biomass;
private ArrayList<String> names;
}

package GrowthModel;


public interface Recorder {

public void record(Simulation sim);
public String toString();
}


package GrowthModel;

import java.util.ArrayList;
```

```
public class RecordGraphData implements Recorder {

public RecordGraphData() {
biomassGraphs = new ArrayList<double[]>();
NLGraphs = new ArrayList<double[]>();
ChlGraphs = new ArrayList<double[]>();
NGraphs = new ArrayList<double[]>();
}
public void record(Simulation sim) { // should do checker to make sure PondWithGraphData
PondWithGraphData pond = (PondWithGraphData) sim.getPond();
biomassGraphs.add(pond.getBiomassData());
NLGraphs.add(pond.getNLData());
ChlGraphs.add(pond.getChlContentData());
NGraphs.add(pond.getNitrogenConcData());
}
public ArrayList<double[]> biomassGraphs;
public ArrayList<double[]> NLGraphs;
public ArrayList<double[]> ChlGraphs;
public ArrayList<double[]> NGraphs;
}

package GrowthModel;

public class RecordMaxGrowth implements Recorder {

public void record(Simulation sim) {
PondForSimulationUse pond = (PondForSimulationUse)sim.getPond();
if (maxPond == null) maxPond = pond;
else if (pond.getMaxNLGrowthPerDay() > maxPond.getMaxNLGrowthPerDay()) {
maxPond = pond;
maxSimName = sim.getName();
maxQZero = pond.model.getStartQt();
}
}
public String toString() {
return maxSimName;
}
public PondForSimulationUse getMaxPond() {
return maxPond;
}
public double getQZero() {
return maxQZero;
}
private double maxQZero = -1;
private PondForSimulationUse maxPond;
private String maxSimName = "";
}

package GrowthModel;

/* Simulation this is an abstract way of looking at running simulations in parallel
* Written by Ian Rankin (10/31/13)
*
* To use a simulation you create a new simulation and then pass it the AlgaeGrowthModel
* you are going to use and pass it a object that implements Runnable.
* The object that implements Runnable should abstractly be modeling an algae pond.
*
* Sample code to run a simulation using the BasicAlgaePond class as its runnable
*
* AlgaeGrowthModel model = new AlgaeGrowthModel(1, 1, 1, 1);
* Simulation simulation = new Simulation(new BasicAlgaePond(model));
* simulation.start();
*
* // once you use start you can not change anything until the
* // model is done (Sort of) (You can ish just don't) */
```

```
public class Simulation {

public Simulation(Pond pond, String name) {
this.pond = pond;
this.name = name;
}
public void start()
{
thread = new Thread(pond);
thread.start();
}

public void waitForCompletion() throws InterruptedException {
thread.join();
}
public boolean isFinished() {
return !thread.isAlive();
}
public Pond getPond() {
return pond;
}
public AlgaeGrowthModel getModel() {
return pond.model;
}
public String getName() {
return name;
}
private String name;
private Pond pond;
private Thread thread;
}

/* RunnableScheduler
* This class is a protected class that should never be run without simulationScheduler and SchedulerDataHandler attached
* this class is what allows SimulationScheduler to be able to control the starting and recording of simulations.
*
* It does this by being a thread itself and every time it runs it checks if all of the simulaions running
* are done. If they are done then it looks to see if there is another simulation queued in the data handler
* if there is it starts that simulation. if there isn't then it sets it too NULL and once all simulations running
* are NULL then it stops the thread.
*
* You can set Simulations to be Simulations passed into it. of you can add a String with the data needed to save memory
*
*/

package scheduler;

import GrowthModel.Simulation;
import GrowthModel.*;

public class RunnableScheduler implements Runnable {

protected RunnableScheduler(SchedulerDataHandler dataHandler) {
handler = dataHandler;
numSimulationsRunning = 0;
simulations = new Simulation[numSimulationsToRun];
}
protected RunnableScheduler(SchedulerDataHandler dataHandler, int numSimulations) {
handler = dataHandler;
numSimulationsToRun = numSimulations;
numSimulationsRunning = 0;
simulations = new Simulation[numSimulationsToRun];
}

public void run() {
System.out.println("WEE");
```

```
while ((!handler.toDo.isEmpty()) || (!(numSimulationsRunning == 0))) // stops the execution if
{
/*
// there is no more simulations to run and is currently not running anything
if (numSimulationsRunning < numSimulationsToRun) { // if it has room to run something
if (!handler.toDo.isEmpty()) { // if it has something else to run
startNextSimulation(getEmptyIndex());
}
}
*/
for (int i = 0; i < simulations.length; i++) {
if (simulations[i] == null) {
startNextSimulation(i);
}
else {
if (simulations[i].isFinished()) {
simulationDone(i);
}
}
}
}
}
public int getNumRunning() {
return numSimulationsRunning;
}
private void simulationDone(int index) {
numSimulationsRunning--;
handler.recorder.record(simulations[index]);
startNextSimulation(index);
}
private void startNextSimulation(int index) {
if (!handler.toDo.isEmpty()) {
simulations[index] = getNextSimulation();
simulations[index].start();
numSimulationsRunning++;
}
else simulations[index] = null;
}
private int getEmptyIndex() {
for (int i = 0; i < simulations.length; i++) {
if (simulations[i] == null) return i;
}
return -1; // oh uh!
}
private Simulation getNextSimulation()
{
Object polledObj = handler.toDo.poll();
Simulation nextSim = null;
if (polledObj instanceof Simulation)
{
nextSim = (Simulation) polledObj;
}
else if (polledObj instanceof String)
{
String stringData = (String) polledObj;
nextSim = newSimFromString(stringData);
}
return nextSim;
}

private Simulation newSimFromString(String data)
{
Pond pond = new PondForSimulationUse(data);
return new Simulation(pond, data);
}
private SchedulerDataHandler handler;
```

```
private Simulation simulations[];
private int numSimulationsRunning;
private int numSimulationsToRun = 8; // set to a default value that can be changed only by the constructor
// why 8? this is because the computer being used to run these simulations is a quad core with each
// processor having 2 virtual cores for a total of 8 virtual cores.
}

// This contains the data that needs to be transfered between RunnableScheduler
// and SimulationScheduler.

package scheduler;

import java.util.ArrayDeque;
import java.util.Queue;

import GrowthModel.Recorder;
import GrowthModel.Simulation;

public class SchedulerDataHandler {

protected SchedulerDataHandler(Recorder recorder) {
this.recorder = recorder;
toDo = new ArrayDeque<Object>();
}
protected Queue<Object> toDo;
protected Recorder recorder;
}

/* SimulationScheduler
* The Simulation Scheduler is a way to handle all of
*
* sample code.
*
* SimulationScheduler scheduler = new SimulationScheduler(new RecordBiomass());
* for (int i = 0; i < 25; i++) {
* AlgaeGrowthModel model = new AlgaeGrowthModel(1, 1, 1, 1, 1, 1 , 1);
* scheduler.add(new Simulation(model, new BasicAlgaePond(model), ("Simulation " + i)));
* }
* scheduler.waitForFinish();
* scheduler.getRecorder().toString();
*
*/

package scheduler;

import GrowthModel.AlgaeGrowthModel;
import GrowthModel.BasicAlgaePond;
import GrowthModel.RecordBiomass;
import GrowthModel.Simulation;
import GrowthModel.Recorder;

public class SimulationScheduler {
/* Constructors
* These create new data Handlers and runnable Schedulers
* both require a data recorder to be passed to it.
* one allows you to change the number of threads that can be ran at the same time */
public SimulationScheduler(Recorder recorder) {
handler = new SchedulerDataHandler(recorder);
runnableScheduler = new RunnableScheduler(handler);
}
public SimulationScheduler(Recorder recorder, int numThreadsToRun) {
handler = new SchedulerDataHandler(recorder);
runnableScheduler = new RunnableScheduler(handler, numThreadsToRun);
}
/* add is the most important method in the scheduler.
* add allows you to add a new simulation to queue up to run
```

```
* if it isn't already running it forces RunnableSchuduler to start */
public void add(Simulation sim) {
handler.toDo.add(sim);
startThreadIfDead();
}
public void add(String simData) {
handler.toDo.add(simData);
startThreadIfDead();
}
// returns the recorder.
public Recorder getRecorder() {
return handler.recorder;
}
// this tells you if there are any simulations currently running
public boolean isRunningSimulations() {
return ((thread != null) && (thread.isAlive()));
}
// this tells you if it has finished running everything in it
public boolean hasFinished() {
return handler.toDo.isEmpty() && (runnableScheduler.getNumRunning() == 0);
}
//This makes you wait until everything in the scheduler is done
public void waitForFinish() throws InterruptedException
{
while (!hasFinished()) {
Thread.sleep(500); // totally do not know about this number
}
}

// pretty self explanatory
private void startThreadIfDead() {
if ((thread == null))
{
thread = new Thread(runnableScheduler);
// thread.setPriority(Thread.MIN_PRIORITY); // maybe...
thread.start();
}
}
private SchedulerDataHandler handler;
private Thread thread;
private RunnableScheduler runnableScheduler;
}

import Graphics.Grapher;
import GrowthModel.PondForSimulationUse;
import GrowthModel.PondWithGraphData;
import GrowthModel.RecordMaxGrowth;
import GrowthModel.Simulation;
import scheduler.SimulationScheduler;

/* Main this is where the main function live plus any other high level functions for testing and such
*
* The current setup is creating 10 simulations to run a very basic growth model
* I mostly used it as a way of testing the frameworks
*/

public class Main {
public static void main(String[] args) throws InterruptedException // throws InterruptedException
{
SimulationScheduler scheduler = new SimulationScheduler(new RecordMaxGrowth());
// these optimized individual parts of the model
//for (double i = 1; i < 20000; i += 1) { // biomass
//for (double i = 1; i < 1000; i += 1) { // N
//for (double i = 0; i < 1; i += .025) { // Chl
//for (double i = 0; i < 10000; i += 1) { // NL
//for (double i = 0; i < 140; i++) { // Incident Irradiance
```

```
//scheduler.add(Double.toString(i) +" 5 0.5 5 .0935 15 0.05");
//scheduler.add("4593 5 0.5 "+ Double.toString(i)+ " .0935 15 0.05");
//scheduler.add("4593 5 "+ Double.toString(i)+ " 5 .0935 15 0.05");
//scheduler.add("4593 "+ Double.toString(i)+ " .5 5 .0935 15 0.05");
//scheduler.add("4593 6569 .5 5 .0935 15 0.05 " + Double.toString(i));
//}
// this ran a huge optimizer
for (double i = 4585; i < 4600; i++) { // biomass 15
for (double j = 6560; j < 6575; j++) { // NL 15
for (double k = .4; k < .6; k += .05) { // chl 4
for (double l = 3; l < 8; l++) { // N 5
for (double m = 100; m < 110; m++) { // Incident Irradiance 10
scheduler.add(Double.toString(i) + " " + Double.toString(j) + " " +
Double.toString(k) + " " + Double.toString(l) + " .0935 5 .05 " + Double.toString(m));
}
}
}
}
}
scheduler.waitForFinish();
RecordMaxGrowth recorder = (RecordMaxGrowth)scheduler.getRecorder();
PondForSimulationUse maxPond = recorder.getMaxPond();
System.out.println(maxPond.toString());
System.out.println("Max Growth Per Day " + maxPond.getMaxNLGrowthPerDay());
System.out.println("Time it is for " + maxPond.getMaxNLTime());
System.out.println("Biomass " + maxPond.model.getStartBiomass());
System.out.println("NL " + maxPond.model.getStartNL());
System.out.println("Chl " + maxPond.model.getStartChl());
System.out.println("N " + maxPond.model.getStartN());
System.out.println("Incident irradiance " + maxPond.model.getIncidentIrradiance());

//------------------------------------------------------------------------
// this runs the max simulation again and shows it in a graph
Grapher graph = new Grapher();
PondWithGraphData pond = new PondWithGraphData(10, .005, maxPond.model.getStartBiomass(),
maxPond.model.getStartNL(), maxPond.model.getStartChl(), maxPond.model.getStartN());
Simulation sim = new Simulation(pond, "Sim");
sim.start();
sim.waitForCompletion();
graph.addXLabels(10);
graph.addGraph(pond.getBiomassData());
graph.addGraph(pond.getNLData());
graph.addGraph(pond.getBiomassGrowthData());
graph.addGraph(pond.getTotalDryWeight());
}
}
```