C++ Coursework Documentation

Bryan Beh

March 2019

1 Introduction

The program started from the example exercises that span across chapter 8 and chapter 9 of Modeling Methods for Marine Science[1] From there, it evolved in hopes to equip the user with a toolbox to solve different problems on this physical premise, which can be modified for the application on other physical systems. The program can be broken down to three fundamental parts: model building, error analysis and parameter optimization.

2 Functionality

2.1 Model Building

The premise of the problem is a coupled ODE that defines the concentration of phosphate in the shallow ocean x1(mmmol) and the deep ocean x2(mmmol) at a given time t(yr) by their respective derivatives:

$$\frac{dx_1}{dt} = \frac{F_R * x_R - F_O * x_1 + F_O * x_2 - P}{V_1} \tag{1}$$

$$\frac{dx_2}{dt} = \frac{F_O * x_1 - F_O * x_2 + E * P}{V_2} \tag{2}$$

where F_R is the river water flux, x_R is the concentration of phosphate in the river, F_O is the overturning water flux, V_1 and V_2 is the box volume of the shallow ocean and deep ocean respectively. The productivity rate P defined as $x_1 * V_1/\tau$ where τ is the residence time of phosphate in shallow ocean and E is the remineralization efficiency are both used later in parameter optimization. Figure 1 depicts the physical system in 2D.

The model building section of the program is organized into a class so that the coupled ODEs can be easily called to be used in the Runge-Kutta evaluation. A file writing member function exist to allow easy translation between arrays into text file. A noise generator member function also exist to create a 'random' dataset that simulates experimental data collected from the real world. Lastly, an error member functionfunction is also in placed to allow for calculating the difference between the model dataset object and the experimental dataset object to produce a corresponding error dataset

2.2 Error Analysis

From the error dataset produced by model building class, a stand alone function for file reading can be called to store the error dataset from text file into a nested array. This can then be analysed with a seperate class for error analysis. It contains several member functions that help extract certain properties of the error dataset. A smallest error member function searches through all the data

[1] Glover, David M et al. Modeling Methods For Marine Science. Cambridge University Press, 2011, pp. 194 - 249.

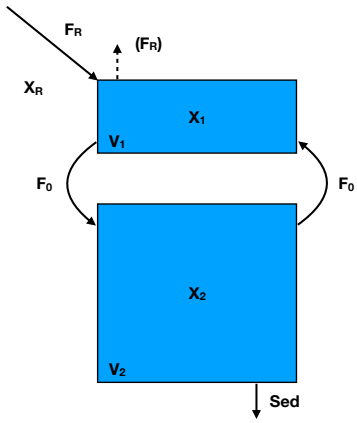


Figure 1: A two-box global ocean phosphate model showing water fluxes. This figure has been taken from the book Modeling Methods for Marine Science (Glover et al, 2011) where this exercise was inspired from.

points for the minimum error at a O(n) time complexity. A group of member functions also exist to calculate a line of best fit into the error dataset through simple linear regression and provide quantitative information on the fit via the R-squared error and the coefficient of determination. It could be used to model and predict the error of the physical system.

2.3 Parameter Optimization

$$J = \sum_{i=1}^{N}((x_{i,1}^{obs} * x_{i,1}^{model})^2 + (x_{i,2}^{obs} * x_{i,2}^{model})^2) \tag{3}$$

The last class for parameter optimization is a child class inherited from the model building class. Its main functionality is to perform parameter optimization on two parameters τ and E from some initial guess. The optimization algorithm implemented is called Gradient Descent. The gradient descent algorithm minimises the cost function defined by another member function which is a least squares expression as shown in equation (3). In the program, E could be optimized to a good precision. However, τ was too insensitive and could not be optimized. This is due to the eigenvalue of E being much larger than τ, resulting in narrow elongated valley in the E and τ parameter space. With a small enough tolerance and no computation rounding errors, the gradient descent method will approach the optimised τ value.

3 Author's Note

It was a great pleasure building this. With more time I will gladly explore implementing the conjugate gradient method for optimization as that will probably resolve the optimization of τ. In building this program, there was no one use-case in mind. However, one example is documented in the Appendix below for the user to follow.

4 Appendix

Figure 2 - 8 are step-by-step sequence a user can use the following program.

```
int main() {

    //Building first two 'real' and 'random' datasets by approximating the coupled ODES with rungekutta
    CPhosphate *real, *random;
    double** val, **store;
    cout << "Building first solution" << endl;
    real = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
```

```
Console
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:47 PM)
Building first solution
Is this real or random data (real/rand)?
real
```

Figure 2: Step 1 - CPhosphate constructor is called and the user can choose between creating a 'real' dataset or a 'noisy' dataset approximated by the Runge-Kutta method.

```
int main() {

    //Building first two 'real' and 'random' datasets by approximating the coupled ODES with rungekutta
    CPhosphate *real, *random;
    double** val, **store;
    cout << "Building first solution" << endl;
    real = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
```

```
Console
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
Runge Kutta 2; step 988:0.42887
Runge Kutta 2; step 989:0.429302
Runge Kutta 2; step 990:0.429733
Runge Kutta 2; step 991:0.430164
Runge Kutta 2; step 992:0.430596
Runge Kutta 2; step 993:0.431027
Runge Kutta 2; step 994:0.431458
Runge Kutta 2; step 995:0.431889
Runge Kutta 2; step 996:0.432321
Runge Kutta 2; step 997:0.432752
Runge Kutta 2; step 998:0.433183
Runge Kutta 2; step 999:0.433614
Runge Kutta 2; step 1000:0.434046
Writing into file; what do you want your file name to be:
real1
```

Figure 3: Step 2 - CPhosphate constructor evaluates x1 and x2 at time t of constant dt steps. A file writing member function is called within the constructor to allow the user to name the file before being written.

```
int main() {

    //Building first two 'real' and 'random' datasets by approximating the coupled ODES with rungekutta
    CPhosphate *real, *random;
    double** val, **store;
    cout << "Building first solution" << endl;
    real = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
```

Console

```
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
9890  0.36543  0.444857
9900  0.308511  0.396779
9910  0.354517  0.437107
9920  0.312108  0.443359
9930  0.329608  0.473944
9940  0.310675  0.463494
9950  0.326511  0.417194
9960  0.349559  0.468159
9970  0.368397  0.389988
9980  0.353258  0.43098
9990  0.3476  0.45447
10000  0.349365  0.405751
Calculating for error between real and random data
Writing into file; what do you want your file name to be:
error
```

Figure 4: Step 3 - Same process is done for 'noisy' dataset with another object. An error member function is now called that calculates the difference between the 'real' and 'noisy' dataset which is written into a file and also stored as a nested array.

```
int main() {

    //Building first two 'real' and 'random' datasets by approximating the coupled ODES with rungekutta
    CPhosphate *real, *random;
    double** val, **store;
    cout << "Building first solution" << endl;
    real = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
```

Console

```
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
9880  0.0285524  0.00291041
9890  0.0303036  0.0155558
9900  0.0269038  0.0329536
9910  0.018815  0.00694273
9920  0.023882  0.0127632
9930  0.00666946  0.0429173
9940  0.0258895  0.0320355
9950  0.010341  0.0146957
9960  0.0124188  0.035838
9970  0.0309697  0.042764
9980  0.0155431  0.00220303
9990  0.0095978  0.0208557
10000  0.0110749  0.0282946
Which error file do you want to read and analyse?:
```

Figure 5: Step 4 - The Analysis class is called for error analysis. A stand alone read file function is first called to read the error file into a nested array for future analysis.

```
int main() {

    //Building first two 'real' and 'random' datasets by approximating the coupled ODES with rungekutta
    CPhosphate *real, *random;
    double** val, **store;
    cout << "Building first solution" << endl;
    real = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
```

Console

```
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
x1 data results
SmallestErr - Minimum error in time: 7150
SmallestErr - Minimum error of x-value: 2.22318e-06
Simple Linear Regression model
m value: 1.53294e-06
b value: 0.00214098
R squared error: 0.0403011
Coefficient of Determination: 0.327009

x2 data results
SmallestErr - Minimum error in time: 0
SmallestErr - Minimum error of x-value: 0
Simple Linear Regression model
m value: 2.10878e-06
b value: 2.29232e-05
```

Figure 6: Step 5 - The constructor is called which processes the respective t, x1 and x2 array from the error dataset for analysis. The error analysis states the minimum error, the line of best fit through Simple Linear Regression, the R squared error, and the coefficient of determination.

```
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
    cout << endl;
    cout << "Optimising Parameters of " << "E = " << E_in << ", Tau = " << tau_in << " to get E = " << E_opt << ",
    optPar* optimisation;
    optimisation = new optPar(tau_in, E_in, tau_opt, E_opt, 10, 10000); //recommended alpha = 0.0001 for this setti

}
```

```
Console
cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
SmallestErr - Minimum error of x-value: 0
Simple Linear Regression model
m value: 2.10878e-06
b value: 2.29232e-05
R squared error: 0.051321
Coefficient of Determination: 0.419306

Optimising Parameters of E = 0.7, Tau = 99.9 to get E = 0.99, Tau = 100
Please key in the tuning parameter alpha:
0.0001
```

Figure 7: Step 6 - A child class of CPhosphate, optPar is called for parameter optimisation. The setting above shows an initial guess of E = 0.7 and Tau = 99.9. We want to eventually reach the optimised values of E = 0.99 and Tau = 100 through minimizing the cost function.

```
    cout << "Building second solution" << endl;
    random = new CPhosphate(0, 0, 0, 100, 0.99, 10, 10000);
    cout << "Calculating for error between real and random data" << endl;
    val = real->error(*random);

    //reads error file for analysis, gives user an option to analyze other files but will not make much sense
    Analysis *x1Analyse, *x2Analyse;
    store = fileRead(1000);
    cout << endl;
    cout << "Displaying data analysis" << endl;
    cout << endl;
    cout << "x1 data results" << endl;
    x1Analyse = new Analysis(store[0], store[1], 1000);
    cout << endl;
    cout << "x2 data results" << endl;
    x2Analyse = new Analysis(store[0], store[2], 1000);

    //parameter optimisation with different tau and reminEff values, simulates scenario when actual parameters are
    double E_in, tau_in, E_opt, tau_opt;
    tau_in = 99.9;
    E_in = 0.7;
    tau_opt = 100;
    E_opt = 0.99;
    cout << endl;
    cout << "Optimising Parameters of " << "E = " << E_in << ", Tau = " << tau_in << " to get E = " << E_opt << ",
    optPar* optimisation;
    optimisation = new optPar(tau_in, E_in, tau_opt, E_opt, 10, 10000);//recommended alpha = 0.0001 for this setti

}
```

```
Console
<terminated> (exit value: 0) cPModel [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/cPModel/Debug/cPModel (07/03/2019, 7:51 PM)
3.98178e-06
4.00634e-06
3.4803e-06
count 631: 1.01994e-06
3.4803e-06
2.82696e-06
4.23162e-06
3.98203e-06
4.00608e-06
3.48029e-06
count 632: 9.83539e-07
Optimised tau parameter: 99.9006
Optimised remineralisation coefficient parameter: 0.990113
Terminated cost function value: 3.48029e-06
```

Figure 8: Step 7 - The constructor calls for the gradient descent member function and minimizes the cost member function at each iteration until it reaches a relative improvement lower than 10^{-6}. The remineralisation coefficient is optimised to a precision of ± 0.0001. However, tau is barely optimised and this is due to the difference in their eigenvalues.

```
double costfunc(double x, double y){
    return(100*pow((y-x*x),2) + pow((1-x),2));
}

double* gradDes(double x_in, double y_in){
    int counts;
    double val_x, val_y, dx, dy, J_old, J_new, rel_imp, dxdt, dydt, alpha;
    double* parContainer;
    val_x = x_in;
    val_y = y_in;
    rel_imp = 1;
    dx = x_in / 10000; //scale three magnitudes down
    dy = y_in / 10000;
    counts = 0;

    cout << "Please key in the tuning parameter alpha: " << endl; //needs manual tuning
    cin >> alpha;

    while(rel_imp > pow(10,-6)){
        counts += 1;
        J_old = costfunc(val_x, val_y);
        dxdt = (costfunc(val_x+dx,val_y) - costfunc(val_x-dx,val_y))/(2*dx);
        dydt = (costfunc(val_x,val_y+dy) - costfunc(val_x,val_y-dy))/(2*dy);
        val_x = val_x - alpha*dxdt;
        val_y = val_y - alpha*dydt;
        J_new = costfunc(val_x,val_y);
```

```
Console
<terminated> (exit value: 0) optimiseTest [C/C++ Application] /Users/bryanbeh/Desktop/Advanced Programming/optimiseTest/De
count 500243: 1.69842e-05
count 500244: 1.49937e-05
count 500245: 1.32257e-05
count 500246: 1.10135e-05
count 500247: 9.02343e-06
count 500248: 7.03358e-06
count 500249: 5.04392e-06
count 500250: 3.27685e-06
count 500251: 1.28761e-06
count 500252: 9.23353e-07
Optimised x parameter: 1
Optimised y parameter: 1
Terminated cost function value: 3.9968e-16
```

Figure 9: Gradient Descent method evaluated on the Rosenbrock function where x and y are correctly optimised.

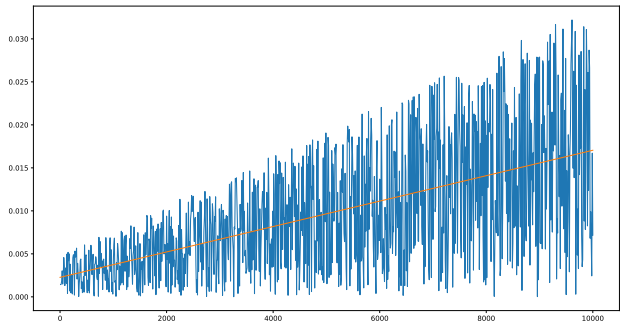
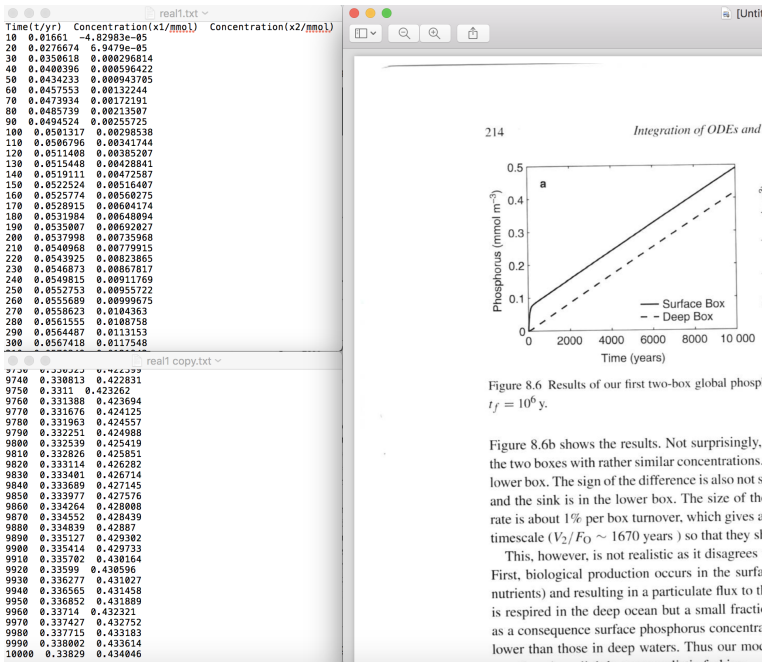


Figure 10: Line of best fit from error dataset using Simple Linear Regression plotted in Python



real1.txt

Time(t/yr)	Concentration(x1/mmol)	Concentration(x2/mmol)
10	0.01661	-4.82983e-05
20	0.0276674	6.9479e-05
30	0.0350618	0.000296814
40	0.0400396	0.000596422
50	0.0434233	0.000943705
60	0.0457553	0.00132244
70	0.0473934	0.00172191
80	0.0485739	0.00213507
90	0.0494524	0.00255725
100	0.0501317	0.00298538
110	0.0506796	0.00341744
120	0.0511408	0.00385207
130	0.0515448	0.00428841
140	0.0519111	0.00472587
150	0.0522524	0.00516407
160	0.0525774	0.00560275
170	0.0528915	0.00604174
180	0.0531984	0.00648094
190	0.0535007	0.00692027
200	0.0537998	0.00735968
210	0.0540968	0.00779915
220	0.0543925	0.00823865
230	0.0546873	0.00867817
240	0.0549815	0.00911769
250	0.0552753	0.00955722
260	0.0555689	0.00999675
270	0.0558623	0.0104363
280	0.0561555	0.0108758
290	0.0564487	0.0113153
300	0.0567418	0.0117548

real1 copy.txt

9740	0.330813	0.422831
9750	0.3311	0.423262
9760	0.331388	0.423694
9770	0.331676	0.424125
9780	0.331963	0.424557
9790	0.332251	0.424988
9800	0.332539	0.425419
9810	0.332826	0.425851
9820	0.333114	0.426282
9830	0.333401	0.426714
9840	0.333689	0.427145
9850	0.333977	0.427576
9860	0.334264	0.428008
9870	0.334552	0.428439
9880	0.334839	0.42887
9890	0.335127	0.429302
9900	0.335414	0.429733
9910	0.335702	0.430164
9920	0.33599	0.430596
9930	0.336277	0.431027
9940	0.336565	0.431458
9950	0.336852	0.431889
9960	0.33714	0.432321
9970	0.337427	0.432752
9980	0.337715	0.433183
9990	0.338002	0.433614
10000	0.33829	0.434046

Figure 8.6 Results of our first two-box global phosp
$t_f = 10^6$ y.

Figure 8.6b shows the results. Not surprisingly,
the two boxes with rather similar concentrations.
lower box. The sign of the difference is also not s
and the sink is in the lower box. The size of th
rate is about 1% per box turnover, which gives a
timescale ($V_2/F_O \sim 1670$ years) so that they sl
This, however, is not realistic as it disagrees
First, biological production occurs in the surfa
nutrients) and resulting in a particulate flux to tl
is respired in the deep ocean but a small fracti
as a consequence surface phosphorus concentra
lower than those in deep waters. Thus our mo

Figure 11: Data produced by different set of fixed parameters compared to the book dataset. Reason that the same parameters were not used is because the textbook data is generated from a simpler version of the coupled ODE. There is no direct comparison of data for the current modified ODE in the program but the similarities of dataset should show confidence regardless.