MatLab Programming

This second sheet shows some basic **MatLab** programming methods. We extend the *ode23* options to assist finding specific times. We write a basic program for finding the time and once again graph the solution.

Lake Pollution Example

The lake pollution problem with fairly simple assumptions is shown to not have an explicit solution. Yet in practical cases, we need to find critical levels of pollution. Thus, the numerical power of MatLab becomes invaluable. The model is described by:

$$\frac{dc(t)}{dt} = -(0.001 + 0.0006\sin(0.0172t))\left(c(t) - 8e^{-0.002t}\right).$$

The RHS is written in a MatLab *function*:

```
function yp = lake2(t,y)

% Model for Lake Pollution
 f = 0.001+0.0006*sin(0.0172*t);
 yp = -f*(y - 8*exp(-0.002*t));
end
```

This is readily simulated for 2000 days with the command

```
[t,y] = ode23(@lake2,[0,2000],0);
```

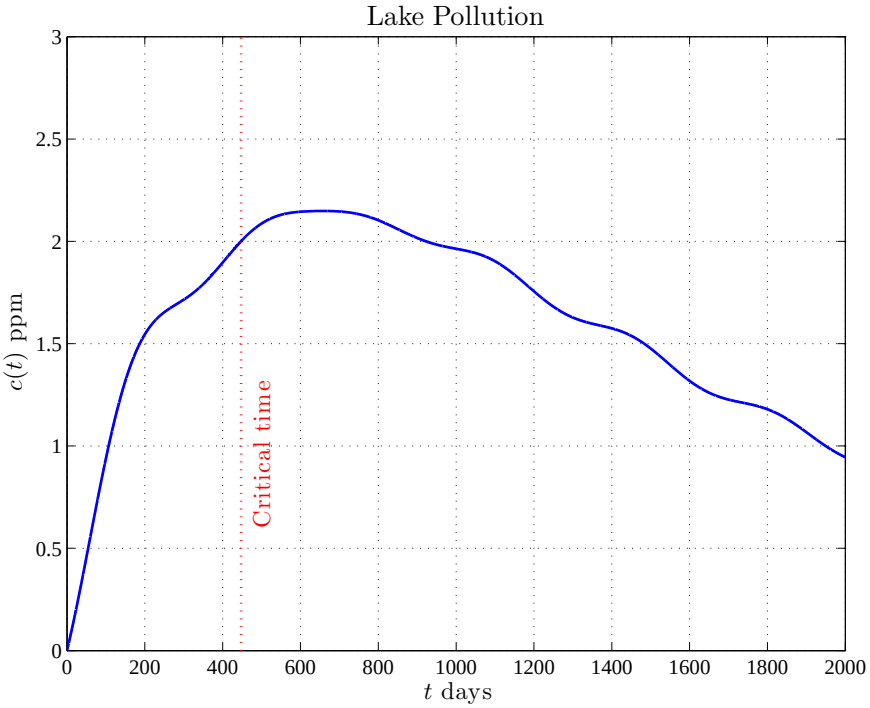
However, we want to determine when it first reaches 2 ppm of pollutant to an accuracy of 0.1 days. Below is a MatLab *script* demonstrating the *while* logical command in a program called *lake_level*.

```
% This script finds numerically when pollution level exceeds 2 ppm
% Divide the time evaluation into steps of 0.1 (20,000 steps)
ts = [0:0.1:2000];
[t1,y1] = ode23(@lake2,ts,0); % Simulate the model (lake2) at given times (ts)
n = 1; yc = y1(n);            % Initialize y for comparison to 2 ppm
while (yc < 2)                % Continue executing this loop until yc > 2
    n = n+1;                  % Increase the index
    yc = y1(n); tc = t1(n);   % Reset the yc and tc value to the new solution value
end
sprintf('t = %.1f.', tc)      % Print out the critical t value
```

We plot this result and include the critical time on the graph with this plot script (*lake2_plot*, available from Math 337 Lecture Notes). **Note:** the script should begin with code described in the **MatLab Introduction** for clearing the figure and Workspace and end with the font control and the output (*print*) commands.

```
mytitle = 'Lake Pollution';    % Title
xlab = '$t$ days';             % X-label
ylab = '$c(t)$ ppm';           % Y-label

lake_level    % Simulate model and find time pollution exceeds 2 ppm (subroutine)

plot(t1,y1,'b-','LineWidth',1.5); % Plot concentration of pollution
hold on                           % Plots Multiple graphs
plot([tc,tc],[0,3],'r:','LineWidth',1.5); % Plot when reaches critical level
```

```
grid                                    % Adds Gridlines
text(500,0.6,'Critical time','rotation',90,'color','red','FontSize',14,...
    'FontName','Times New Roman');

xlim([0  2000]);  % Defines limits of graph
ylim([0  3]);
```



Mercury in Lake Trout

This section shows the programs required to support the lecture notes on the build-up of mercury (Hg) in Lake Trout. There are three critical steps to modeling this phenomenon. The lecture notes contain **4** sets of data: 1. Length vs Age, 2. Weight vs Length, 3. Weight vs Age, 4. Hg Concentration vs Age of Lake Trout from Lake Superior. The modeling includes **2** differential equations and an allometric model (dimensional analysis).

Relation Between Length and Age

The first model uses the **von Bertalanffy equation**, which upon solving the differential equation yields the model:

$$L(t; L_*, b) = L_* \left(1 - e^{-bt}\right).$$

This is an equation giving the length of a fish, L, based on its age, t. It has two parameters that need to be fit to the data, L_* and b. The reader is referred to the lecture notes: **Linear Differential Equations** for details on deriving the model, tables of the data, and a discussion of nonlinear least squares fitting of data. The data can be accessed through the MatLab file, **fishdat.mat**. The data on the length and age of Lake Trout, *Salvelinus namaycush*, in Lake Superior are copied from Kory Groetsch[1] and stored in MatLab variables *tdfish* for age and *ldfish* for length (vectors with 19

[1]Kory Groetsch, Total Mercury and Copper Concentrations in Lake Trout and Whitefish Fillets, Activity: 19-23, From Lake Superior, Environmental Section, Biological Services Division, 1998

elements).

If we define these data points $(t_i, L_i), i = 1..19$, then the error between the measured length, L_i, at time t_i and the model evaluated at t_i is

$$e_i = L_i - L_* \left(1 - e^{-bt_i}\right), \qquad i = 1..19.$$

The **Sum of Square Errors** function satisfies

$$J(L_*, b) = \sum_{i=1}^{19} \left(L_i - L(t; L_*, b)\right)^2 = \sum_{i=1}^{19} e_i^2,$$

and has some scalar value for each pair of the parameter values, (L_*, b), which we select as a vector in MatLab $p = [(L_*, b]$. With p we define a MatLab function for this sum of square errors function between the data and the model $L(t; L_*, b) = L_* \left(1 - e^{-bt}\right) = p(1) \left(1 - e^{-p(2)t}\right)$.

```
function J = sumsq_vonBert(p,tdata,ldata)
% Function computing sum of square errors for von Bertalanffy model
model = p(1)*(1 - exp(-p(2)*tdata));
error = model - ldata;
J = error*error';
end
```

This function takes advantage of the **vector** capabilities of MatLab. The data are stored as vectors. Any of the internal functions, such as *exp*, with a vector argument produces a vector, and scalars are added and subtracted componentwise, *i.e.*,

$$a - exp([x_1, x_2, ..., x_n]) = [a - e^{x_1}, a - e^{x_2}, ..., a - e^{x_n}].$$

In this function, $error = [e_1, e_2, ..., e_n]$ is a row vector, so $error'$, the transpose, is a column vector. The product $error * error'$ is

$$[e_1, e_2, ..., e_n] \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = e_1^2 + e_2^2 + ... + e_n^2,$$

which is the sum of square errors.

As noted above the age and length data are stored in *tdfish* and *ldfish*, respectively. The optimal solution is the **nonlinear least squares fit** to these data, which is the minimum possible value of the **sum of square errors** function over all possible L_* and b. MatLab has a powerful function, which is capable of numerically finding the minimum of a function, *fminsearch*. You can learn about numerical algorithms for finding minima in Math 541 and details about this particular function with the MatLab *help fminsearch* command. For our problem, we need a reasonable initial guess, $p_0 = [L_{*0}, b_0] = [100, 0.1]$. (A poor initial guess may prevent the algorithm from converging or require multiple iterations.) In addition, since our sum of square errors function requires input of the data, these must be supplied to the *fminsearch* in the *OPTIONS* part of this function (separated by []). Below we show how to execute the MatLab function *fminsearch* with a user defined function *sumsq_vonBert*, including an initial guess and the data sets.

```
[p1,J,flag] = fminsearch(@sumsq_vonBert,[100,0.1],[],tdfish,ldfish)
```

MatLab returns the best fitting parameter values in the vector p_1 (which could be used in another iteration if one is uncertain of convergence), the least sum of square errors, J, and a variable $flag$, which is 1 if MatLab thinks *fminsearch* has converged and 0 if it failed to converge. For our

particular problem, MatLab returns the results $p_1 = [92.401, 0.14553]$, $J = 1, 107.3$, and $flag = 1$, so the best fitting parameters are

$$L_* = 92.401 \qquad \text{and} \qquad b = 0.14553$$

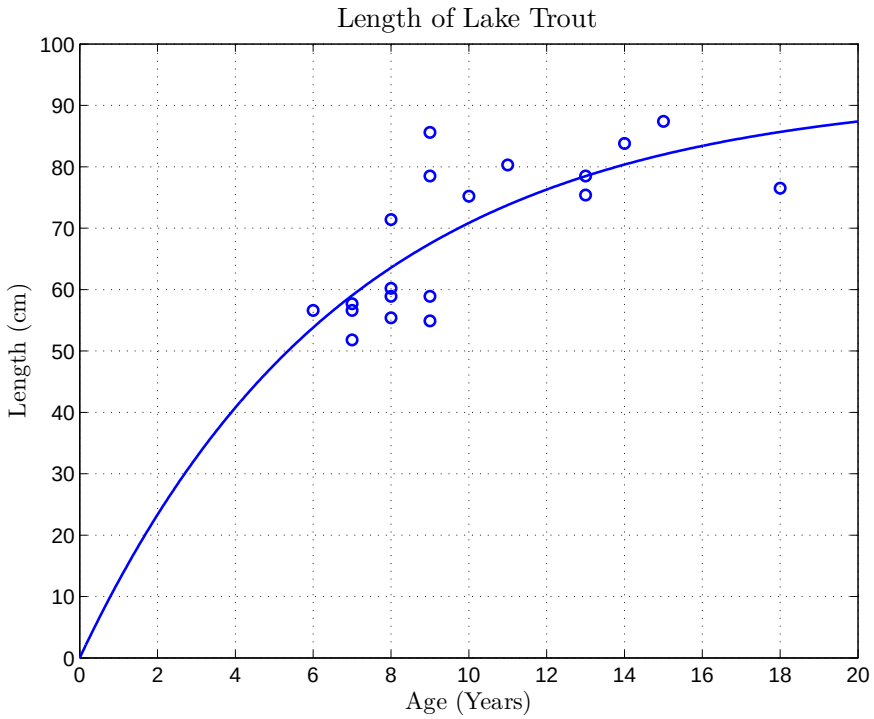
and our best model for the length of Lake Trout as it ages is

$$L(t) = 92.401\left(1 - e^{-0.14553t}\right).$$

A MatLab script file is developed to graph this best fitting model and the data. This best fitting model is shown with the data in the graph below. The script, *vonBert_plot.m*, is available from the Lecture notes (including initial clear and final text and output controls, which are omitted here for clarity).

```
1
2  clear                              % Clear previous definitions
3  figure(1)                 % Assign figure number
4  clf                       % Clear previous figures
5  hold off                  % Start with fresh graph
6
7  mytitle = 'Length of Lake Trout';          % Title
8  xlab = 'Age (Years)';                      % X-label
9  ylab = 'Length (cm)';                      % Y-label
10
11 load('fishdat');              % Provide vectors of Lake Trout data
12
13 tt = linspace(0,20,500);              % t domain of function
14 Lt = 92.404*(1-exp(-0.14553*tt));     % Function for von Bertalanffy
15
16 plot(tt,Lt,'b-','LineWidth',1.5);  % Plot model
17 hold on                            % Plots Multiple graphs
18 plot(tdfish,ldfish,'bo','LineWidth',1.5);  % Plot data with circles
19
20 grid                                  % Adds Gridlines
21
22 xlim([0  20]);  % Defines limits of graph
23 ylim([0  100]);
24
25 fontlabs = 'Times New Roman';   % Font type used in labels
26 xlabel(xlab,'FontSize',14,'FontName',fontlabs,'interpreter','latex');
27         % x-Label size and font
28 ylabel(ylab,'FontSize',14,'FontName',fontlabs,'interpreter','latex');
29         % y-Label size and font
30 title(mytitle,'FontSize',16,'FontName','Times New Roman','interpreter','latex');
31         % Title size/font
32 set(gca,'FontSize',12);          % Axis tick font size
33
34 print -depsc vonBert.eps                  % Create figure as EPS file
35 %print -djpeg vonBert.jpg                 % Create figure as JPEG file
```

The graph of the von Bertalanffy model with the relevant data is below.



Relation Between Weight and Length

The next step in the modeling for this problem is to find a functional relationship between the weight and the length of the Lake Trout. This is known as **Allometric** modeling or a **Power Law** relationship. Specifically, we examine a relationship of the form

$$W = kL^{a}.$$

If we take the logarithms (natural) of both sides, then this model can be written

$$\ln(W) = a\ln(L) + \ln(k),$$

which is a linear relation between the $\ln(W)$ and the $\ln(L)$. There are standard formula to find the **Linear Least Squares** best fit to linear data. MatLab uses a subroutine *polyfit* to find the best slope and intercept for a line passing through a data set. (**Note:** MatLab uses the natural logarithm in its calculations, so uses the command *log(x)* to mean $\ln(x)$.)

The analysis above shows that the linear least squares fit of a line to the logarithms of the data provides the allometric model. This suggests that MatLab's *polyfit* routine can be used to fit these types of models. The following MatLab function allows the easy input of data to produce the parameters for our allometric model:

```
function [k,a] = powerfit(ldata,wdata)
 % Power law (Allometric) fit for model W = k*L^a
 % Uses linear least squares fit to logarithms of data
 Y = log(wdata);        % Logarithm of W-data
 X = log(ldata);        % Logarithm of L-data
 p = polyfit(X,Y,1);    % Linear fit to X and Y with p = [slope, intercept]
 a = p(1);              % Value of exponent
 k = exp(p(2));         % Value of leading coefficient
end
```

Our data file discussed above has the vector data variables *ltdfish* for length and *wtdfish* for weight (vectors with 25 elements) of Lake Trout data from the Kory Groetsch data. If we execute our *powerfit* function with the following MatLab command:

```
[k,r] = powerfit(ltdfish,wtdfish)
```

MatLab outputs the variables $k = 0.015049$ and $a = 2.8591$. It follows that using this method of finding the linear best fit to the logarithms of the data yields a best **allometric model** of the form:

$$W = 0.015049\, L^{2.8591}.$$

From a modeling perspective, this problem can be examined in two other ways. Similar to the section before, a nonlinear least squares best fit can be used to obtain the unbiased best fit of the model (with respect to the parameters k and a). Alternately, a dimensional analysis supported by the allometric fit above would suggest that the weight of a fish varies like the cube of the length, *i.e.*, self-similarity of fishes would suggest that as the length increases, then the height and width would similarly increase giving a cubic relation between length and weight. This would fix the parameter $a = 3$ and only allow changes in k. Below are two functions for finding the **sum of square errors** for these two models. The function for the 2-parameter problem is

```
function J = sumsq_nonlin(p,ldata,wdata)
% Function computing sum of square errors for allometric model
model = p(1)*ldata.^p(2);
error = model - wdata;
J = error*error';
end

% Obtain the least sum of square errors
% [p1,J,flag] = fminsearch(@sumsq_allom,[k,a],[],ltdfish,wtdfish);
```

The MatLab function for the cubic model with only 1-parameter is

```
function J = sumsq_cubic(p,ldata,wdata)
% Function computing sum of square errors for cubic allometric model
model = p*ldata.^3;
error = model - wdata;
J = error*error';
end

% Obtain the least sum of square errors
% [p1,J,flag] = fminsearch(@sumsq_cubic,k,[],ltdfish,wtdfish);
```

MatLab Vector Operations: The previous two functions use MatLab's ability to perform componentwise operations on vectors. These allometric models have L^a, so when $L = [L_1, L_2, ..., L_n]$ is a vector, MatLab provides an easy way to form the vector $L^a = [L_1^a, L_2^a, ..., L_n^a]$. This is done using the period combined with the caret, so typing `L.^a` yields $[L_1^a, L_2^a, ..., L_n^a]$. Similarly, if $a = [a_1, a_2, ..., a_n]$ and $b = [b_1, b_2, ..., b_n]$, then the componentwise product formed by `a.*b` yields the vector $[a_1b_1, a_2b_2, ..., a_nb_n]$. Similarly, the componentwise quotient formed by `a./b` yields the vector $[a_1/b_1, a_2/b_2, ..., a_n/b_n]$.

We take advantage of a MatLab **script program** to execute all of the parameter searches for all three models, find the best fitting parameters (and least sum of square errors), and create a plot of the models along with the data. Below is the script file (*allo_plot.m*) with comments inside describing what is being executed.

```
mytitle = 'Allometric Models for Lake Trout';   % Title
xlab = 'Length (cm)';                % X-label
ylab = 'Weight (g)';                 % Y-label

load('fishdat');   % Loads stored data file

```

```
[k,a] = powerfit(ltdfish,wtdfish);  % Finds best linear model to log of data
allom = k*ltdfish.^a;               % Defines allometric model at data
err = allom-wtdfish;                % Finds error between model and data
J1 = err*err';                      % Finds sum of square errors
[p1,J2,flag] = fminsearch(@sumsq_nonlin,[k,a],[],ltdfish,wtdfish);
  % Find nonlinear least squares fit to allometric model to data
[p2,J3,flag] = fminsearch(@sumsq_cubic,k,[],ltdfish,wtdfish);
  % Find nonlinear least squares fit to cubic model to data
lt = linspace(0,100,500);       % length domain
W1t = k*lt.^a;                  % Allometric model - over domain
W2t = p1(1)*lt.^p1(2);          % Nonlinear best fit - over domain
W3t = p2*lt.^3;                 % Cubic model - over domain

plot(lt,W1t,'b-','LineWidth',1.5);  % Plot allometric model
hold on                             % Plots Multiple graphs
plot(lt,W2t,'k-','LineWidth',1.5);  % Plot nonlinear fit model
plot(lt,W3t,'r-','LineWidth',1.5);  % Plot cubic fit model
plot(ltdfish,wtdfish,'ro','LineWidth',1.5);  % Plot data
grid                                % Adds Gridlines
legend({'Allometric','Nonlinear Fit','Cubic Fit','Data'},'location','Northwest','
    FontSize',10,'FontName','Times New Roman');  % Create legend
xlim([0 100]);  % Defines limits of graph
ylim([0 5500]);
```

After executing this script we obtain the three best fitting models with their least sum of square errors, J_i. The parameter values and J_i are found stored in the MatLab **Workspace**. The best **allometric model** is

$$W = 0.015049\, L^{\,2.8591}, \qquad \text{with} \qquad J_1 = 3.5147 \times 10^6.$$

The best nonlinear least square fitting **allometric model** is

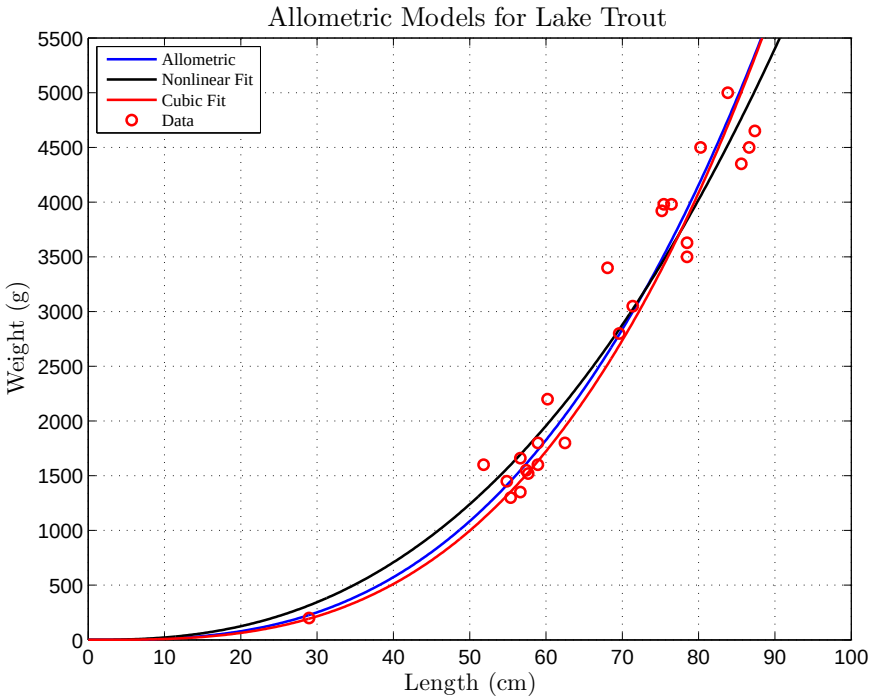
$$W = 0.068695\, L^{\,2.5052}, \qquad \text{with} \qquad J_2 = 2.8683 \times 10^6.$$

The best **allometric model**, where the exponent is assumed to be $a = 3$, is

$$W = 0.0079791\, L^{\,3}, \qquad \text{with} \qquad J_3 = 3.9113 \times 10^6.$$

Clearly, the second model is the best fitting model from a unbiased perspective having the least square errors. However, the last model is not that far away and makes more sense from a physical argument.

The script also produced the graph below, which shows that all of the three models are fairly similar. This gives visual evidence that using the third model in the next section is reasonable.



Accumulation of Mercury with Age

The model for accumulation of Mercury as the fish ages is discussed in the lecture notes: **Linear Differential Equations**. It begins with a best fitting model for the weight vs age data. Below are two ways to create the best fitting cubic model with the weight vs age data. The first technique uses the cubic model described above with the von Bertalanffy model. A one parameter search is applied to the cube of the von Bertalanffy model, so find the best α_1 for $W(t) = \alpha_1 \left(92.401 \left(1 - e^{-0.14553t}\right)\right)^3$. When this approach is taken with *fminsearch*, the result is $\alpha_1 = 0.0079798$, so the weight of a fish satisfies the following composite function:

$$W(t) = 6295.4 \left(1 - e^{-0.14553t}\right)^3,$$

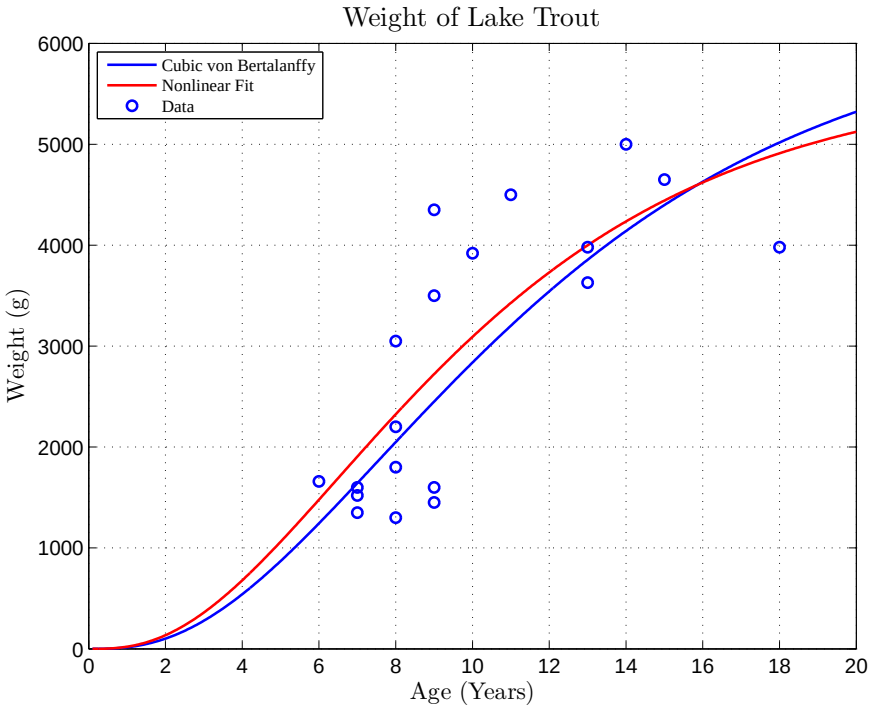
where t is the age of the fish, giving a sum of square errors of $J = 1.3168 \times 10^7$. A second method is to parallel the fitting of the length vs age for the von Bertalanffy equation with two parameters, $W(t) = \alpha_2 \left(1 - e^{-\beta_2}\right)^3$. With a nonlinear least squares fit of this model to the weight vs age data (almost identical to *sumsq_vonBert* M-script, so omitted), the composite function for weight of a fish satisfies:

$$W(t) = 5677.67 \left(1 - e^{-0.16960t}\right)^3,$$

with a sum of square errors of $J = 1.2049 \times 10^7$. These two composite functions are graphed to show this fit of the weight of the fish as it ages. (The graphing script is similar to the one for the von Bertalanffy model, so is omitted here.) The graphs are very similar, and there is not a significant difference in the sum of square errors. The calculations will use the model generalized model derived from the von Bertalanffy equation

$$W(t) = W_* \left(1 - e^{-bt}\right)^3,$$

with $W_* = 6295.4$ or 5677.67 and $b = 0.14553$ or 0.16960, corresponding to the von Bertalanffy or nonlinear least squares fits, respectively.



The lecture notes discuss how mercury (Hg) accumulates in the body from feeding and water passing over the gills. Since fish are cold blooded animals, their energy expenditure (balanced by food and O_2 intake) should be roughly proportional to the weight of the fish. (Alternate models might consider **Kleiber's law** or more mercury laden food sources with aging.) The lecture notes point out that Hg doesn't tend to leave the body after entering. This suggests the rate of Hg entering the body of a fish should be proportional to the weight of the fish, giving the differential equation:

$$\frac{dH}{dt} = \kappa W(t) = \kappa W_* \left(1 - e^{-bt}\right)^3,$$

where $H(t)$ is the amount of Hg in the fish. This differential equation is readily solved with details in the lecture notes. The resulting equation for the amount of Hg in the fish is:

$$H(t) = \frac{\kappa W_*}{6b}\left(6bt + 18e^{-bt} - 9e^{-2bt} + 2e^{-3bt} - 11\right).$$

The concentration in the fish satisfies

$$c(t) = \frac{H(t)}{W(t)}.$$

With the values of W_* and b determined by our weight model, this becomes a one parameter, κ, search to fitting the data on the concentration of mercury in fish as they age. Below are the MatLab programs used to fit the data for the concentration of mercury in fish as they age. The first program computes the sum of square errors.

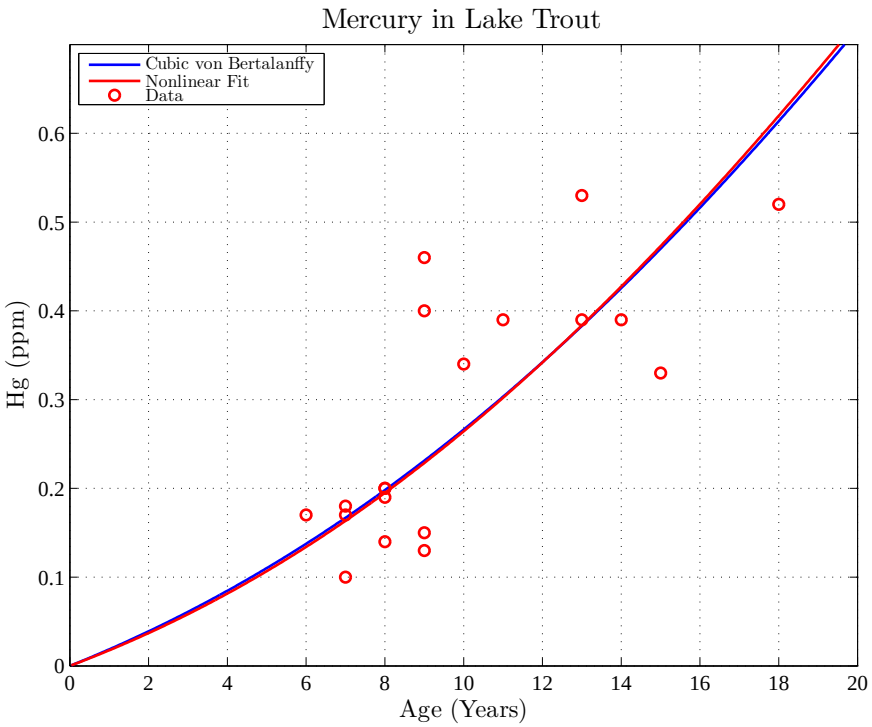
```
function J = sumsq_Hg(p,tdata,hgdata,w)
% Function computing sum of square errors for von Bertalanffy model
% p is kappa, tdata and hgdata are fish data, w = [W*,b]
model = (p*w(1)/(6*w(2)))*(6*w(2)*tdata + 18*exp(-w(2)*tdata) - ...
     9*exp(-2*w(2)*tdata) + 2*exp(-3*w(2)*tdata)-11);  % H(t)
wt = w(1)*(1 - exp(-w(2)*tdata)).^3;                   % W(t)
```

```
ct = model./wt;                                          % c(t)
error = ct - hgdata;
J = error*error';
end
```

This program is used in the MatLab script with *fminsearch* to find the best fitting models and graph the models and the data.

```
mytitle = 'Mercury in Lake Trout';       % Title
xlab = 'Age (Years)';                    % X-label
ylab = 'Hg (ppm)';                       % Y-label

load('fishdat');          % Provide vectors of Lake Trout data

tt = linspace(0,20,500);          % t domain of function

w1 = [6295.4,0.14553];
[p1,J1,flag]=fminsearch(@sumsq_Hg,0.1,[],tdfish,hgdfish,w1);
model1 = (p1*w1(1)/(6*w1(2)))*(6*w1(2)*tt + 18*exp(-w1(2)*tt) - ...
    9*exp(-2*w1(2)*tt) + 2*exp(-3*w1(2)*tt)-11);
wt1 = w1(1)*(1 - exp(-w1(2)*tt)).^3;
ct1 = model1./wt1;
w2 = [5677.67,0.16960];
[p2,J2,flag]=fminsearch(@sumsq_Hg,0.1,[],tdfish,hgdfish,w2);
model2 = (p2*w2(1)/(6*w2(2)))*(6*w2(2)*tt + 18*exp(-w2(2)*tt) - ...
    9*exp(-2*w2(2)*tt) + 2*exp(-3*w2(2)*tt)-11);
wt2 = w2(1)*(1 - exp(-w2(2)*tt)).^3;
ct2 = model2./wt2;

plot(tt,ct1,'b-','LineWidth',1.5);  % Plot model cubic von Bertalanffy
hold on                            % Plots Multiple graphs
plot(tt,ct2,'r-','LineWidth',1.5);  % Plot model nonlinear fit
plot(tdfish,hgdfish,'ro','LineWidth',1.5);  % Plot data with red circles

grid                                % Adds Gridlines
legend({'Cubic von Bertalanffy','Nonlinear Fit','Data'},'location','Northwest', '
    FontSize',10,'FontName','Times New Roman');  % Create legend
xlim([0  20]);  % Defines limits of graph
ylim([0  0.7]);
```

This MatLab script finds that using the weight model with the cubic best fitting von Bertalanffy model gives $\kappa = 0.071406$ with a least sum of square errors to the data of $J = 0.17113$, while the weight model with the cubic nonlinear fit model gives $\kappa = 0.066953$ with a least sum of square errors to the data of $J = 0.17427$. The graph is shown below with the data, and it is clear that choice of model is almost indistinguishable between these two models compared to the data.


Mercury in Lake Trout
Cubic von Bertalanffy
Nonlinear Fit
Data
Hg (ppm)
Age (Years)
0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14
16
18
20