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:

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

The RHS is written in a MatLab function:

```matlab
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

```matlab
[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.

```matlab
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

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

n = 1; yc = y(1);
while (yc < 2) % Continue executing this loop until yc > 2
    n = n+1;
eyc = y1(n); tc = t1(n);
end
```

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.
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:

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

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, \( L_\ast \) 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

---

1Kory Groetsch, Total Mercury and Copper Concentrations in Lake Trout and Whitefish Fillets, Activity: 19-23, From Lake Superior, Environmental Section, Biological Services Division, 1998
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
\[
e_i = L_i - L_s \left(1 - e^{-bt_i}\right), \quad i = 1..19.
\]

The **Sum of Square Errors** function satisfies
\[
J(L_s, b) = \sum_{i=1}^{19} (L_i - L(t; L_s, b))^2 = \sum_{i=1}^{19} e_i^2,
\]
and has some scalar value for each pair of the parameter values, \((L_s, b)\), which we select as a vector in MatLab \(p = [(L_s, b)]\). With \(p\) we define a MatLab function for this sum of square errors function between the data and the model
\[
L(t; L_s, b) = L_s \left(1 - e^{-bt}\right) = p(1) \left(1 - e^{-p(2)t}\right).
\]

```matlab
function J = sumsq_vonBert(p, tdata, ldata)
    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
\[
\begin{bmatrix}
e_1 \\
e_2 \\
\vdots \\
e_n
\end{bmatrix}
\begin{bmatrix}
e_1 \\
e_2 \\
\vdots \\
e_n
\end{bmatrix}^T = e_1^2 + e_2^2 + ... + e_n^2,
\]

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 \(L_s\) 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, \(p_0 = [L_s0, b_0] = [100, 0.1]\). (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.

```matlab
[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

$$L_\ast = 92.401 \quad \text{and} \quad b = 0.14553$$

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).

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

```matlab
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 \texttt{ltdfish} for length and \texttt{wtdfish} for weight (vectors with 25 elements) of Lake Trout data from the Kory Groetsch data. If we execute our \texttt{powerfit} function with the following MatLab command:

\begin{verbatim}
[k, r] = powerfit(ltdfish, wtdfish)
\end{verbatim}

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 \textbf{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, \textit{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 \textbf{sum of square errors} for these two models. The function for the 2-parameter problem is

\begin{verbatim}
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);
\end{verbatim}

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

\begin{verbatim}
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);
\end{verbatim}

\textbf{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 \( a \cdot b \) 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 \textbf{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 (\texttt{allo_plot.m}) with comments inside describing what is being executed.
% Finds best linear model to log of data
[k, a] = powerfit(ltdfish, wtdfish);
% Defines allometric model at data
allom = k * ltdfish.^a;
% Finds error between model and data
err = allom - wtdfish;
% Finds sum of square errors
J1 = err * err';
% Find nonlinear least squares fit to allometric model to data
[p1, J2, flag] = fminsearch(@sumsq_nonlin, [k, a], [], ltdfish, wtdfish);
% Find nonlinear least squares fit to cubic model to data
[p2, J3, flag] = fminsearch(@sumsq_cubic, k, [], ltdfish, wtdfish);

lt = linspace(0, 100, 500);
% Allometric model - over domain
W1t = k * lt.^a;
% Nonlinear best fit - over domain
W2t = p1(1) * lt.^p1(2);
% Cubic model - over domain
W3t = p2 * lt.^3;

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

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

The best nonlinear least square fitting allometric model is

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

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

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

Clearly, the second model is the best fitting model from an 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 $\alpha_1$ for $W(t) = \alpha_1 (92.401 (1 - e^{-0.14553t}))^3$.

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, $W(t) = \alpha_2 (1 - e^{-\beta t})^3$. 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

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

with $W_\ast = 6295.4$ 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\textsubscript{2} intake) should be roughly proportional to the weight of the fish. (Alternate models might consider [Kleiber’s law](https://en.wikipedia.org/wiki/Kleiber%27s_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:

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

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:

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

The concentration in the fish satisfies

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

With the values of $W_\ast$ and $b$ determined by our weight model, this becomes a one parameter, $\kappa$, 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.

```matlab
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_\ast, 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)
```
This program is used in the MatLab script with `fminsearch` to find the best fitting models and graph the models and the data.

```
ct = model./wt; % c(t)
error = ct - hgdata;
J = error*error;
end
```

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