

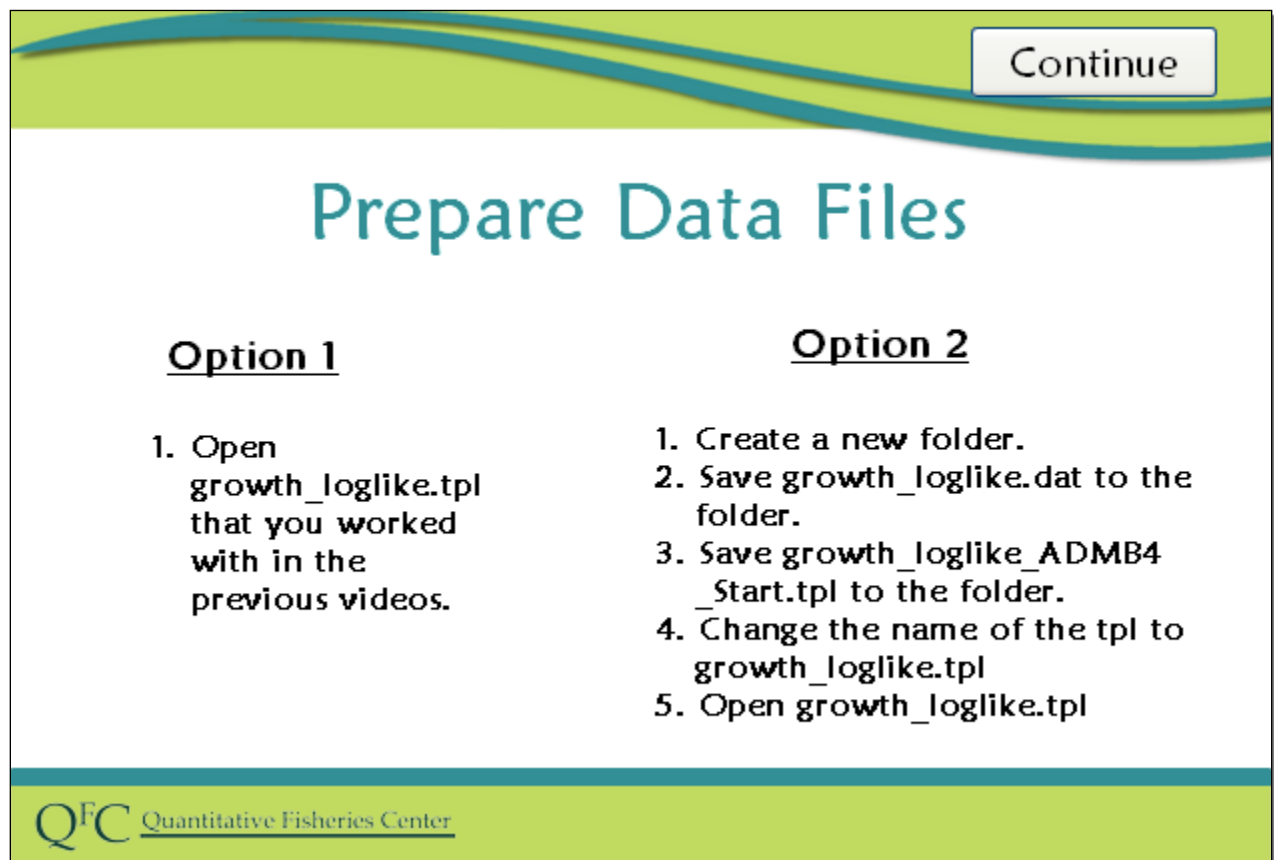
[Continue](#)

Creating your first AD Model Builder application

Part 4 - Begin Modeling



This video was created using ADMB-IDE release 4.4.0-2 (January, 2011)
You may notice some minor differences if using a different version.



The screenshot shows a presentation slide with a green header and footer. The header contains a 'Continue' button. The main title is 'Prepare Data Files'. Below the title, there are two columns: 'Option 1' and 'Option 2'. Option 1 lists a single step to open an existing file. Option 2 lists five steps to create a new folder, save data files, and rename a template file. The footer contains the Quantitative Fisheries Center logo and name.

Continue

Prepare Data Files

Option 1

1. Open growth_loglike.tpl that you worked with in the previous videos.

Option 2

1. Create a new folder.
2. Save growth_loglike.dat to the folder.
3. Save growth_loglike_ADMB4_Start.tpl to the folder.
4. Change the name of the tpl to growth_loglike.tpl
5. Open growth_loglike.tpl

QFC Quantitative Fisheries Center

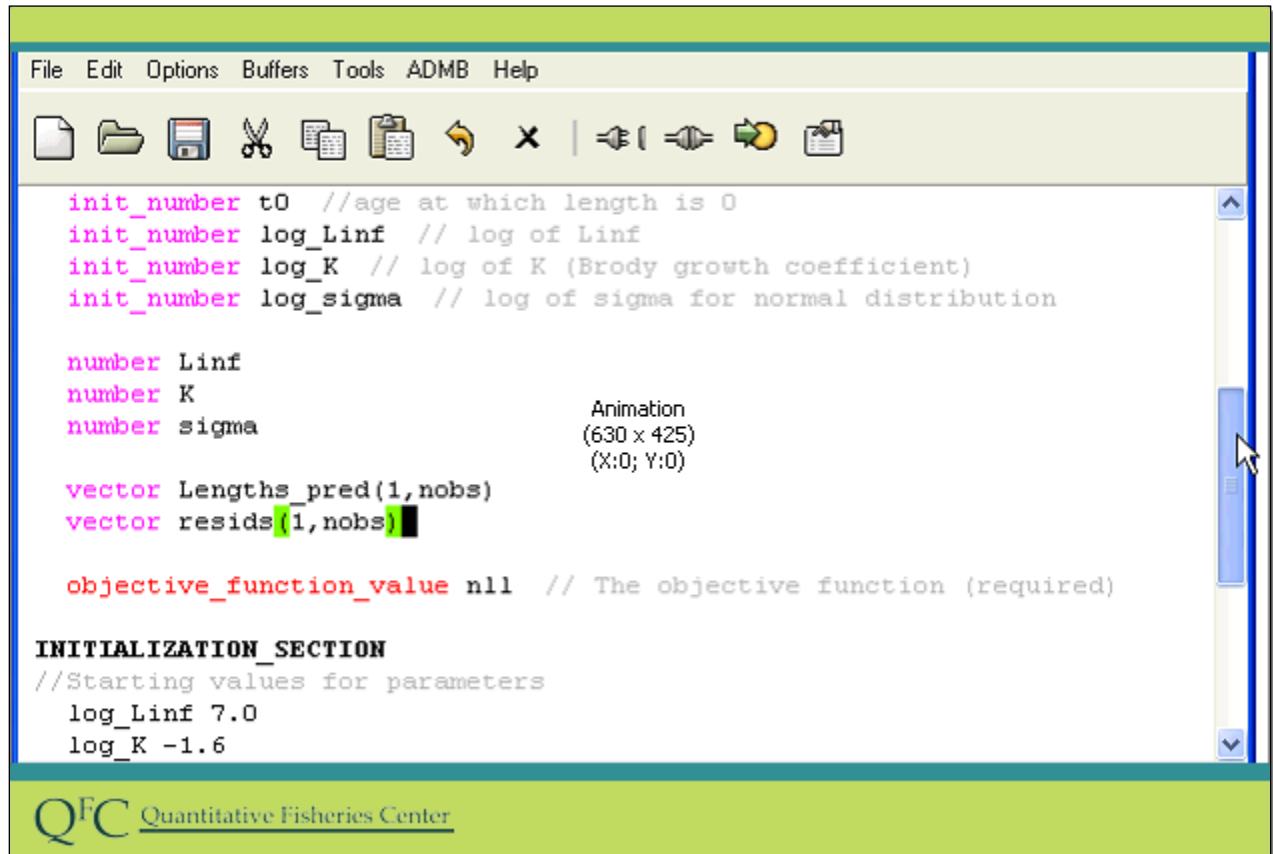
Prepare and open your files. Click next when you are ready.

Open growth_loglike.tpl that you worked with in the previous videos or

1. Create a new folder.
2. Save growth_loglike.dat to the folder.
3. Save growth_loglike_ADMB4_Start.tpl to the folder.
4. Change the name of the tpl to growth_loglike.tpl

Open growth_loglike.tpl

Parameter Section



```
File Edit Options Buffers Tools ADMB Help

init_number t0 //age at which length is 0
init_number log_Linf // log of Linf
init_number log_K // log of K (Brody growth coefficient)
init_number log_sigma // log of sigma for normal distribution

number Linf
number K
number sigma

vector Lengths_pred(1,nobs)
vector resids(1,nobs)

objective_function_value nll // The objective function (required)

INITIALIZATION_SECTION
//Starting values for parameters
log_Linf 7.0
log_K -1.6
```

We are now ready to really start the modeling. We know we are going to generate predicted lengths for each observation and the deviation between observed and predicted values for each observation, i.e., the residuals.

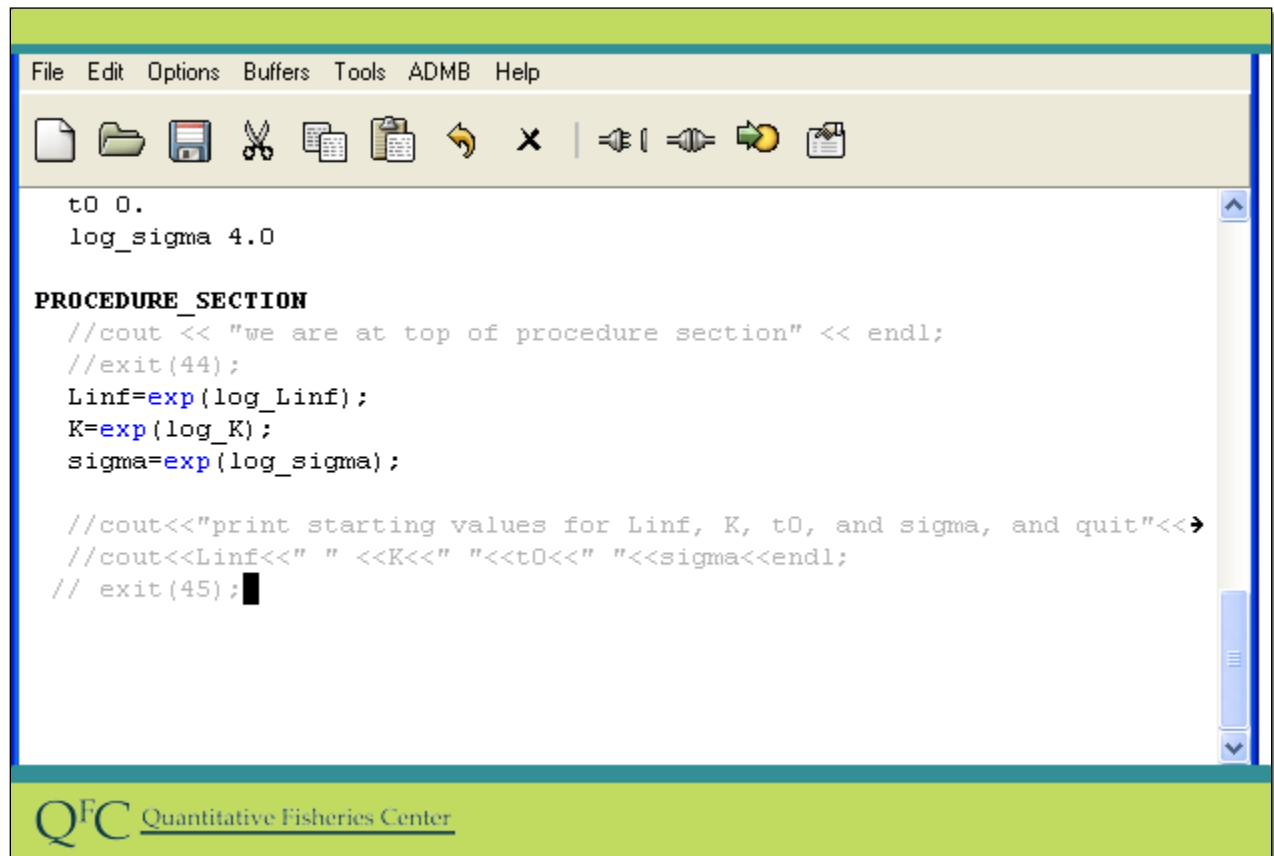
So we add definitions of these variables to the parameter section. Notice we define these in the parameter and not data section because both will change values depending upon parameter values. There is one for each observation so they have the same index minimums and maximums as the age and length data.

Slide Code:

```
vector Lengths_pred(1,nobs)
```

```
vector resids(1, nobs)
```

Procedure Section



The screenshot shows the ADMB software interface. The menu bar includes File, Edit, Options, Buffers, Tools, ADMB, and Help. The toolbar contains icons for file operations (new, open, save, cut, copy, paste, undo, redo) and a print icon. The main text area contains the following code:

```
t0 0.
log_sigma 4.0

PROCEDURE_SECTION
//cout << "we are at top of procedure section" << endl;
//exit(44);
Linf=exp(log_Linf);
K=exp(log_K);
sigma=exp(log_sigma);

//cout<<"print starting values for Linf, K, t0, and sigma, and quit"<<endl;
//cout<<Linf<<" " <<K<<" " <<t0<<" " <<sigma<<endl;
// exit(45);
```

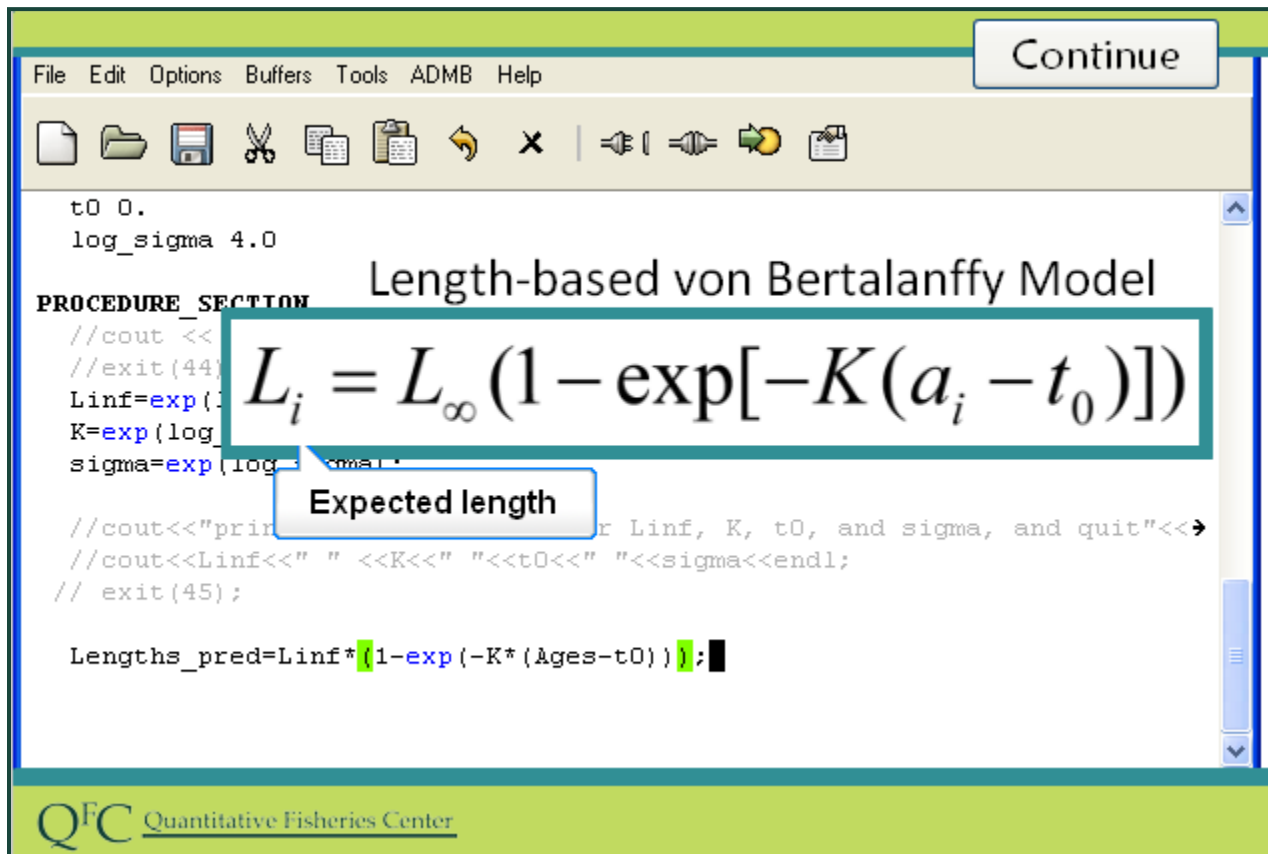
The bottom of the window features the Quantitative Fisheries Center logo and name.

We now go to the procedure section.

We need to comment out our old cout and exit statements.

Slide Code:

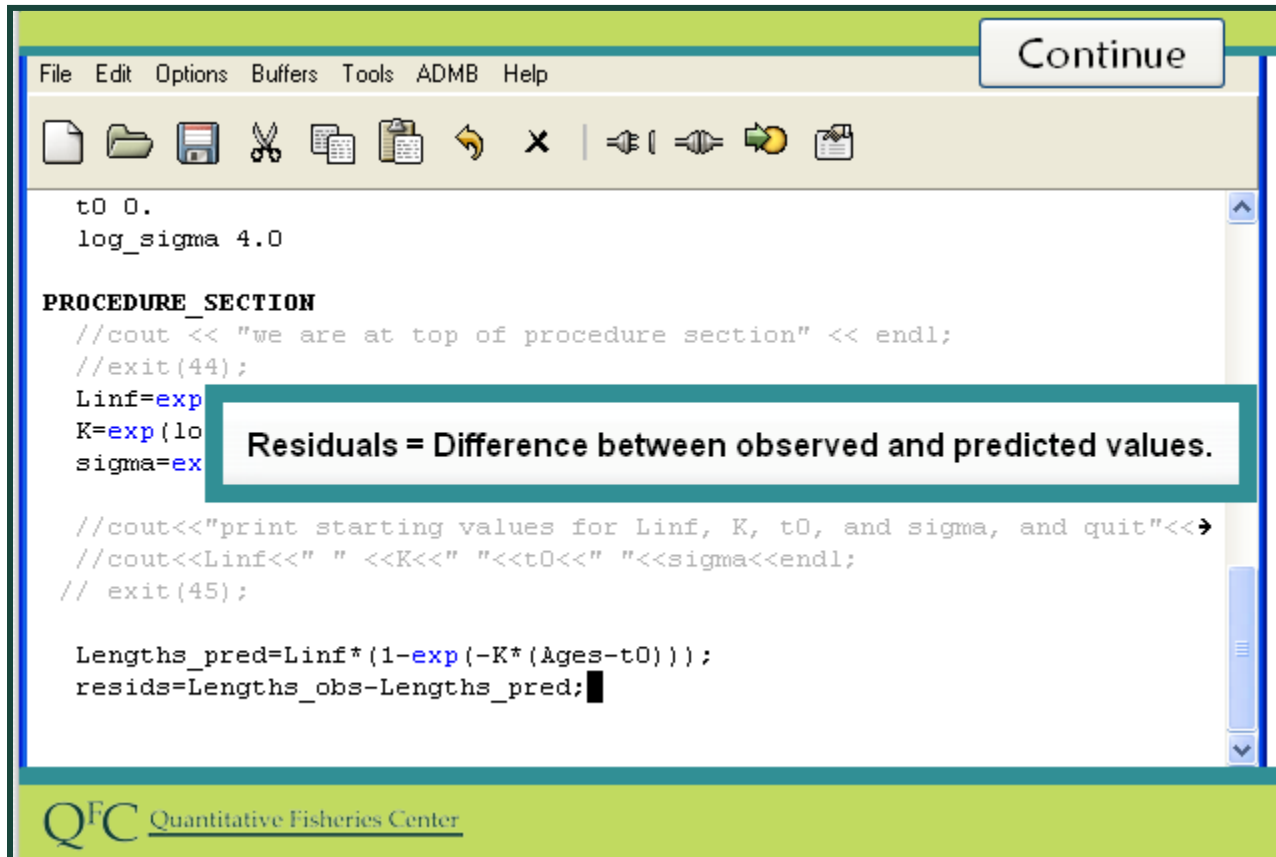
```
// cout<<"print starting.....
// cout<<Linf<<.....
// exit (45);
```



Now we add a line of code to generate predicted values. Notice that admb knows how to work with vectors. Many functions applied to a vector will apply that function to each element of the vector. Likewise if we multiply a vector by a number or subtract a number from a vector we multiply each element of the vector by the number or subtract the number from each element of the vector. As a result we can write just one line of code to produce the desired result – a vector of predicted values. We are however aware that admb will use matrix algebra if it applies so you could get unexpected results when multiplying or dividing matrices or vectors by each other. For example, if you had two matrices of the same size called A and B, A*B would attempt to do matrix multiplication not multiplication of the elements of A by the corresponding element of B. There are special functions for element by element multiplication and division available in admb but we don't get into their use in this video series on developing your first program.

Slide Code:

```
Lengths_pred=Linf*(1-exp(-K*(Ages-t0)));
```



The screenshot shows the ADMB software interface. At the top is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'ADMB', and 'Help'. To the right of the menu bar is a 'Continue' button. Below the menu bar is a toolbar with various icons for file operations and editing. The main area is a code editor with the following C++ code:

```
t0 0.
log_sigma 4.0

PROCEDURE_SECTION
//cout << "we are at top of procedure section" << endl;
//exit(44);
Linf=exp
K=exp(10
sigma=ex

//cout<<"print starting values for Linf, K, t0, and sigma, and quit"<<endl;
//cout<<Linf<<" " <<K<<" " <<t0<<" " <<sigma<<endl;
// exit(45);

Lengths_pred=Linf*(1-exp(-K*(Ages-t0)));
resids=Lengths_obs-Lengths_pred;
```

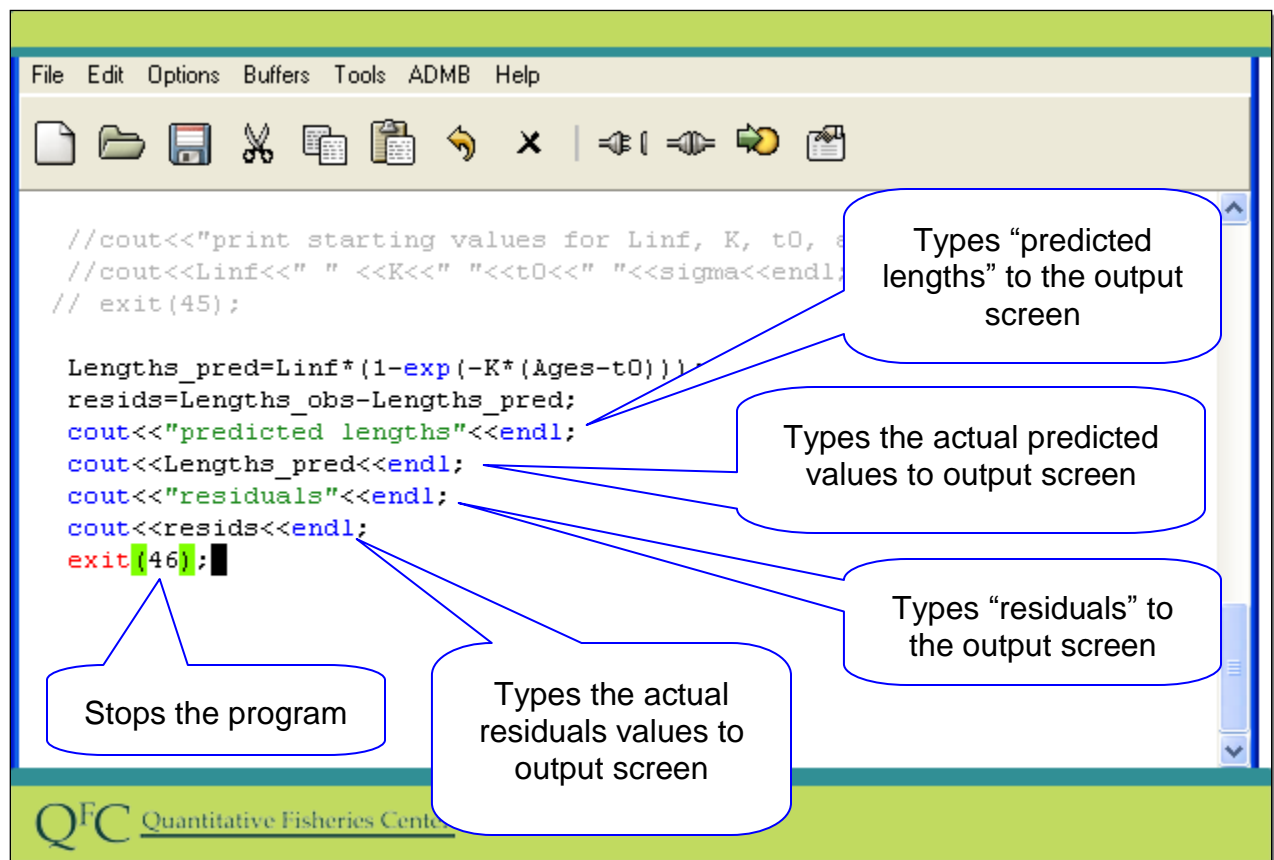
A callout box with a teal border and white background is overlaid on the code, containing the text: **Residuals = Difference between observed and predicted values.**

At the bottom of the window is a green footer bar with the 'QFC Quantitative Fisheries Center' logo and text.

We now add a line to calculate residuals. Notice that matrix subtraction is simply subtraction of corresponding elements of one vector or matrix from another so we again can get away with writing just one line of code to calculate all the residuals.

Slide Code:

```
resids=Lengths_obs-Lengths_pred;
```



At this point we again test that things are working correctly by writing out our predicted values and residuals and using an exit statement.

Slide Code:

```
cout<< "predicted lengths"<<endl;
cout<<Lengths_pred<<endl;
cout<< "residuals"<<endl;
cout<<resids<<endl;
exit (46);
```

The screenshot shows a terminal window with the following code and output:

```
//cout<<"print starting values for L→
// cout<<Linf<<" " <<K<<" "<<t0<<" "→
//exit(45);

Lengths_pred=Linf*(1.-exp(-K*(Ages-t→
resids=Lengths_obs-Lengths_pred;
cout<<"predicted lengths"<<endl;
cout<<Lengths_pred<<endl;
cout<<"residuals"<<endl;
cout<<resids<<endl;
exit(46);
```

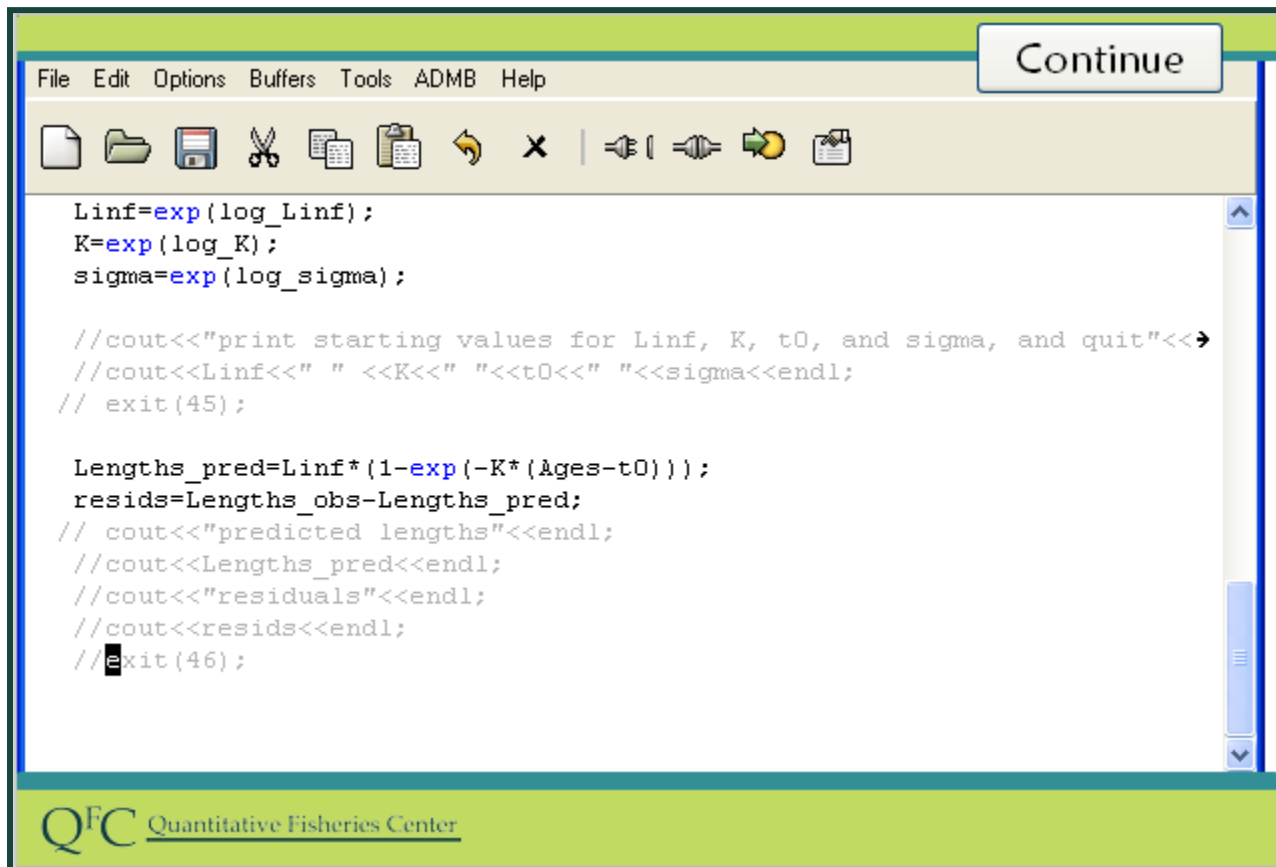
The output on the right side of the window is:

```
predicted lengths
364.321 364.321 364.321 364.321 3
residuals
201.679 228.679 277.679 290.679 2
Process growth_loglike exited abno
```

An arrow points from the `cout<<"predicted lengths"<<endl;` line in the code to the output header `predicted lengths`.

What we should get at this point are the predicted values and residuals based on our starting parameter values.

So build and run the program. We do get to the right spot. The predicted values don't match the data too well as the residuals are consistently positive, but the predicted values do start at a lower value and increase toward the current value of L -infinity. So things do seem to be working and hopefully our admB program will be able update the parameters to better fit the data. But to fit the data we need to define the negative log likelihood as our objective function.

A screenshot of the ADMB software interface. The window has a title bar with a 'Continue' button. Below the title bar is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'ADMB', and 'Help'. Underneath the menu bar is a toolbar with various icons for file operations and editing. The main area is a text editor containing C++ code for a fishery model. The code defines parameters Linf, K, and sigma, calculates predicted lengths, and computes residuals. Comments indicate where to print starting values and predicted lengths. The code ends with an exit statement.

```
Linf=exp(log_Linf);
K=exp(log_K);
sigma=exp(log_sigma);

//cout<<"print starting values for Linf, K, t0, and sigma, and quit"<<endl;
//cout<<Linf<<" " <<K<<" " <<t0<<" " <<sigma<<endl;
// exit(45);

Lengths_pred=Linf*(1-exp(-K*(Ages-t0)));
resids=Lengths_obs-Lengths_pred;
// cout<<"predicted lengths"<<endl;
//cout<<Lengths_pred<<endl;
//cout<<"residuals"<<endl;
//cout<<resids<<endl;
//exit(46);
```

We now need to code in the negative log-likelihood equation.

But first we must comment out the previous cout and exit statements.

Slide Code:

```
// cout<< "predicted lengths"<<endl;
// cout<<Lengths_pred<<endl;
// cout<< "residuals"<<endl;
// cout<<resids<<endl;
// exit (46);
```

Objective Function

The screenshot shows the ADMB software interface. The menu bar includes File, Edit, Options, Buffers, Tools, ADMB, and Help. A toolbar with various icons is below the menu. The main window displays C++ code. A 'Continue' button is in the top right corner. A mathematical formula is overlaid on the code, with arrows pointing from its components to the corresponding code parts:

- The term $n \log \sigma$ is linked to `nll=nobs*log_sigma`.
- The term $\frac{1}{2\sigma^2}$ is linked to `(0.5/square(sigma))`.
- The term $\sum_{i=1}^n (L_i - \hat{L}_i)^2$ is linked to `*norm2(resids);`.

The code in the background includes:

```

Linf=exp(log_Linf);
K=exp(log_K);
sigma=exp(log_sigma);

//cout<<"print sta
//cout<<Linf<<" "
// exit(45);

Lengths_pred=Linf*
resids=Lengths_obs
// cout<<"predicted
//cout<<Lengths_pr
//cout<<"residuals"<<endl;
//cout<<resids<<endl;
//exit(46);

nll=nobs*log_sigma + (0.5/square(sigma))*norm2(resids);

```

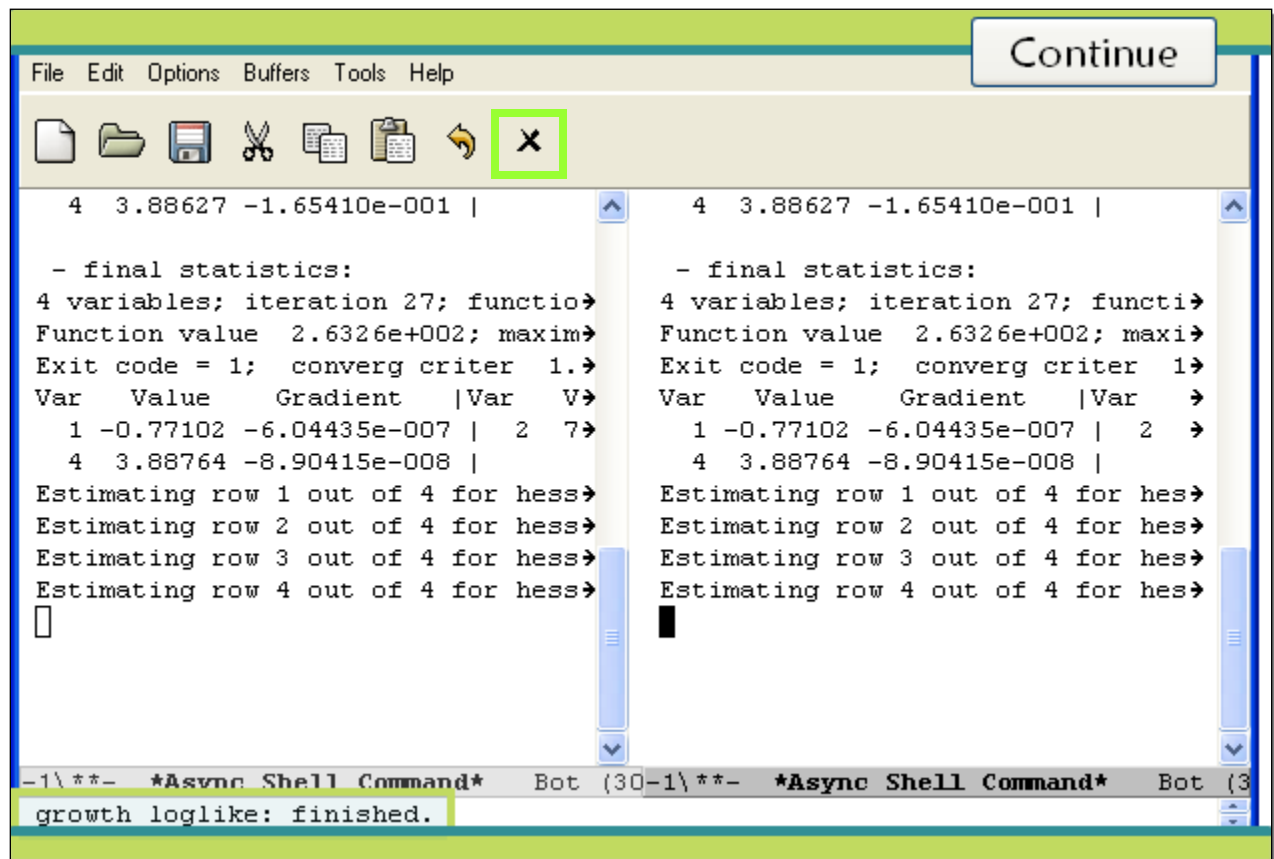
The Quantitative Fisheries Center logo is visible in the bottom left corner.

Now we write the objective function.

The first term is just the sample size times log of sigma. 0.5 divided by the square of sigma is the same as 1 divided by the quantity two times sigma squared. Square is a function that squares its argument. Norm2 is a function that squares all the elements within and sums them up. So this is the sum of squared residuals. That is the same as the sum of the squared deviations between observed and expected values. So this line produces the desired normal log-likelihood.

Slide Code:

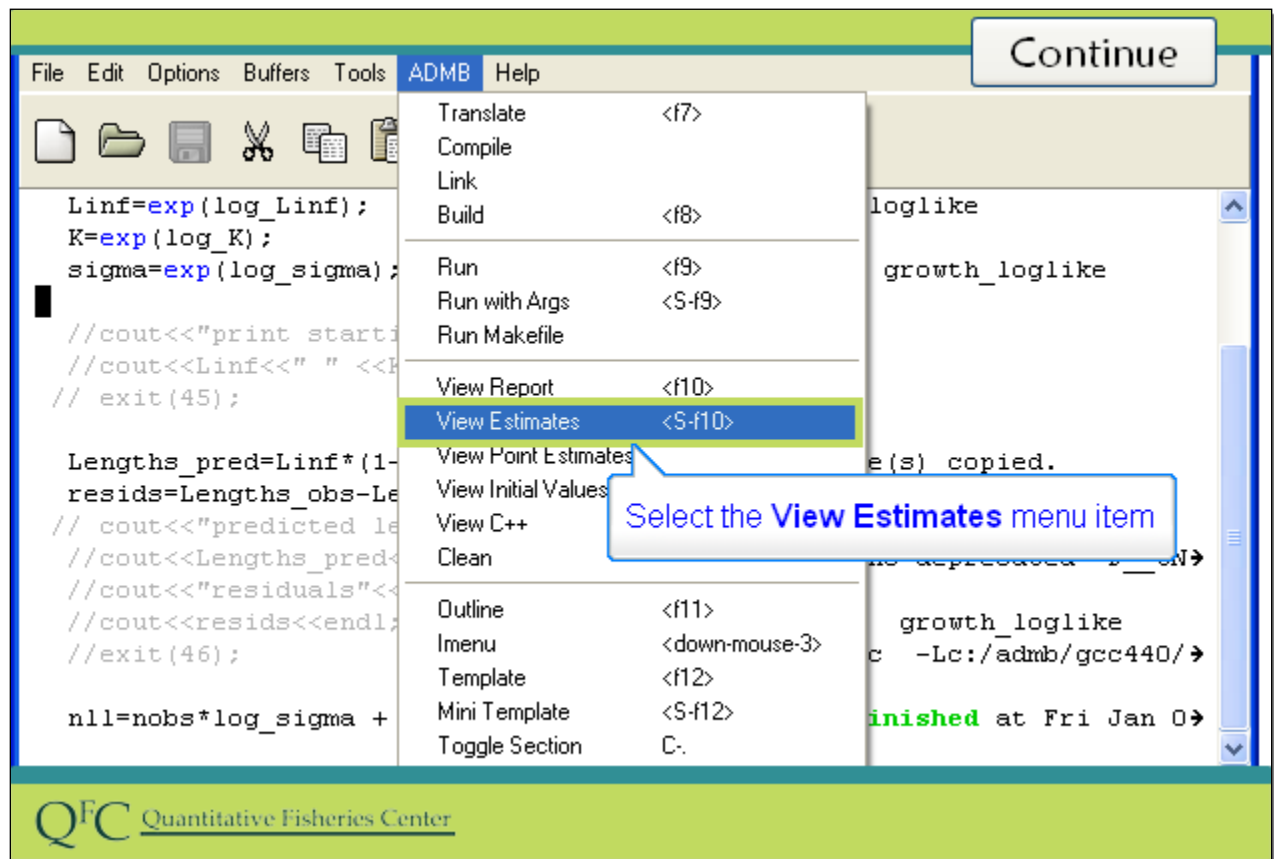
```
nll=nobs*log_sigma + (0.5/square(sigma))*norm2(resids);
```



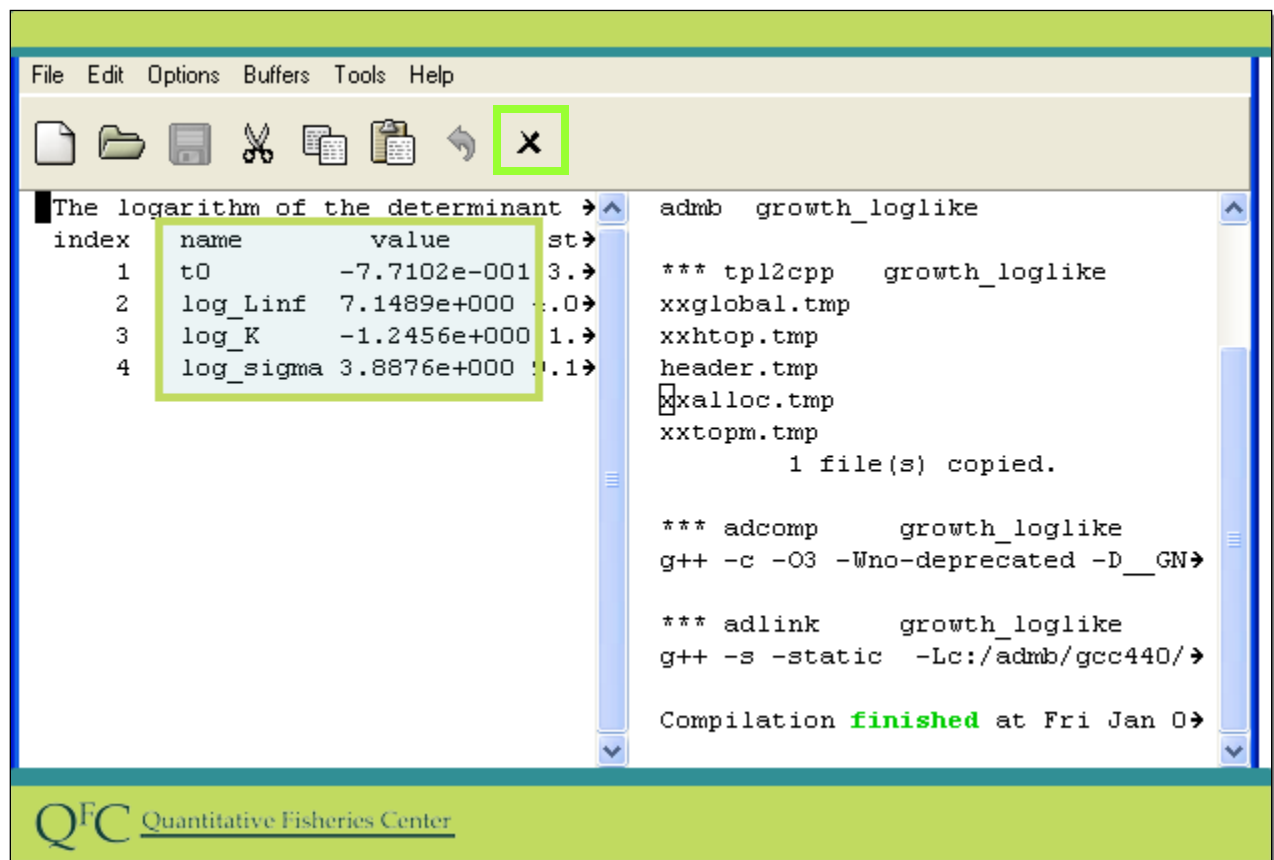
At this point we should be ready to let admb estimate the parameters. We have set starting values, and connected our parameters to the resulting value of the negative log-likelihood. So again let us build and run this.

The build seems to have worked. When we run it a bunch of information gets sent to a buffer describing the progress during estimation. When done a message gets sent to the “mini-buffer” usually at the bottom of your interface saying “growth_loglike: finished.”

This is generally a good sign since there are no messages saying it exited abnormally or warnings about failures to converge. At this point we can look at the parameter file containing the parameter estimates and the ending maximum gradient value. It is worth checking that this is less than the convergence criterion value. Sometimes you might miss a warning but if this is greater than the convergence criterion you know there was one you missed. The default convergence criterion is 1 time 10 to the minus 4, or 1E-4.



A file with somewhat more useful results is the cor or correlation file. Select the ADMB menu and select view estimates.

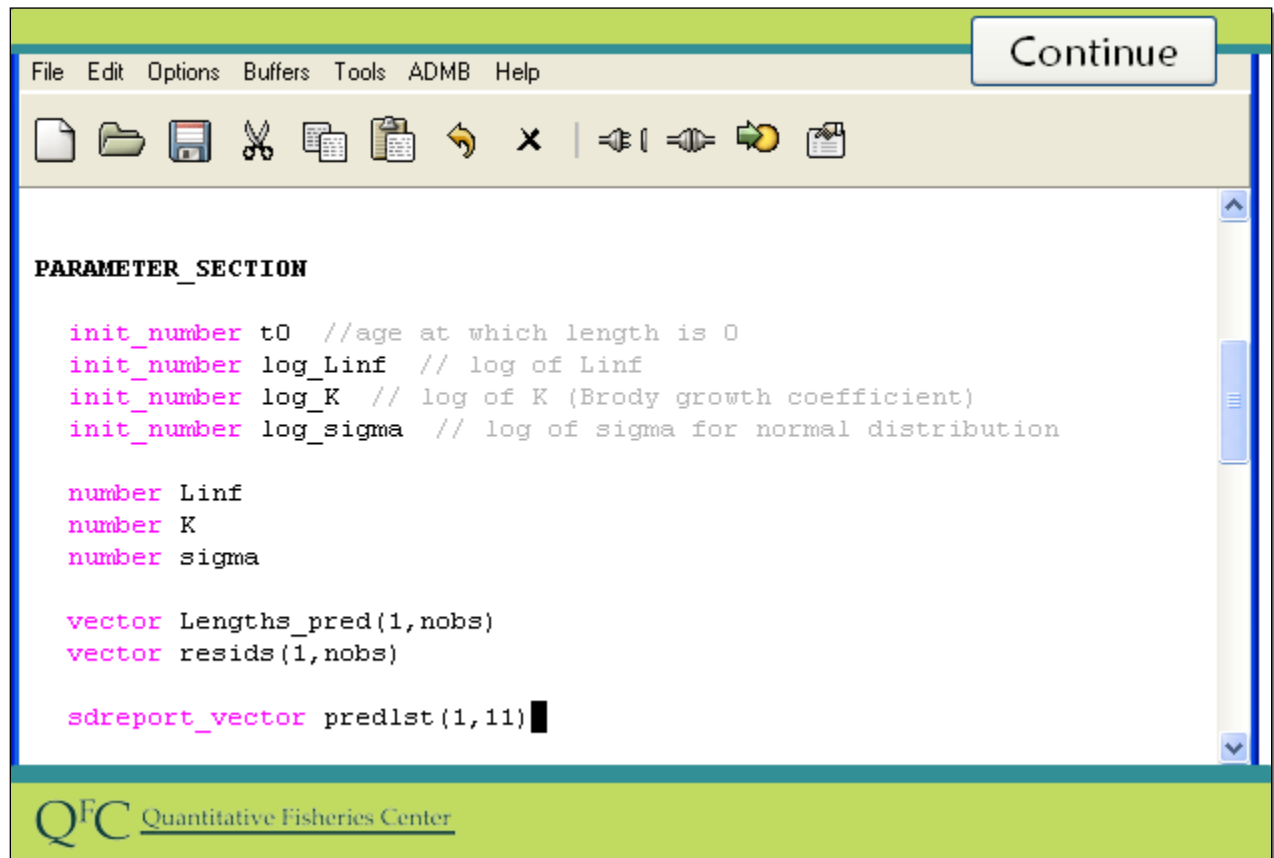


This file contains the estimates, the asymptotic standard errors for the parameter estimates, and the asymptotic correlation matrix.

If you open the cor file using the admb menu make sure to close the file before you rerun your model with changes as it may not open a new file the next time you try if the previous one is still open.

So our model is working, at least we think. But our output so far is pretty limited. We only have parameter estimates. To even know what L infinity is we need a calculator to back-transform the estimated log_Linf. In addition to backtransformed parameters, we might want to get our hands on residuals and predicted values, and perhaps some measure of uncertainty for predicted values.

Parameter Section



```
PARAMETER_SECTION

init_number t0 //age at which length is 0
init_number log_Linf // log of Linf
init_number log_K // log of K (Brody growth coefficient)
init_number log_sigma // log of sigma for normal distribution

number Linf
number K
number sigma

vector Lengths_pred(1,nobs)
vector resids(1,nobs)

sdreport_vector predlst(1,11)
```

To this end we now learn about sd report variables and also the report section.

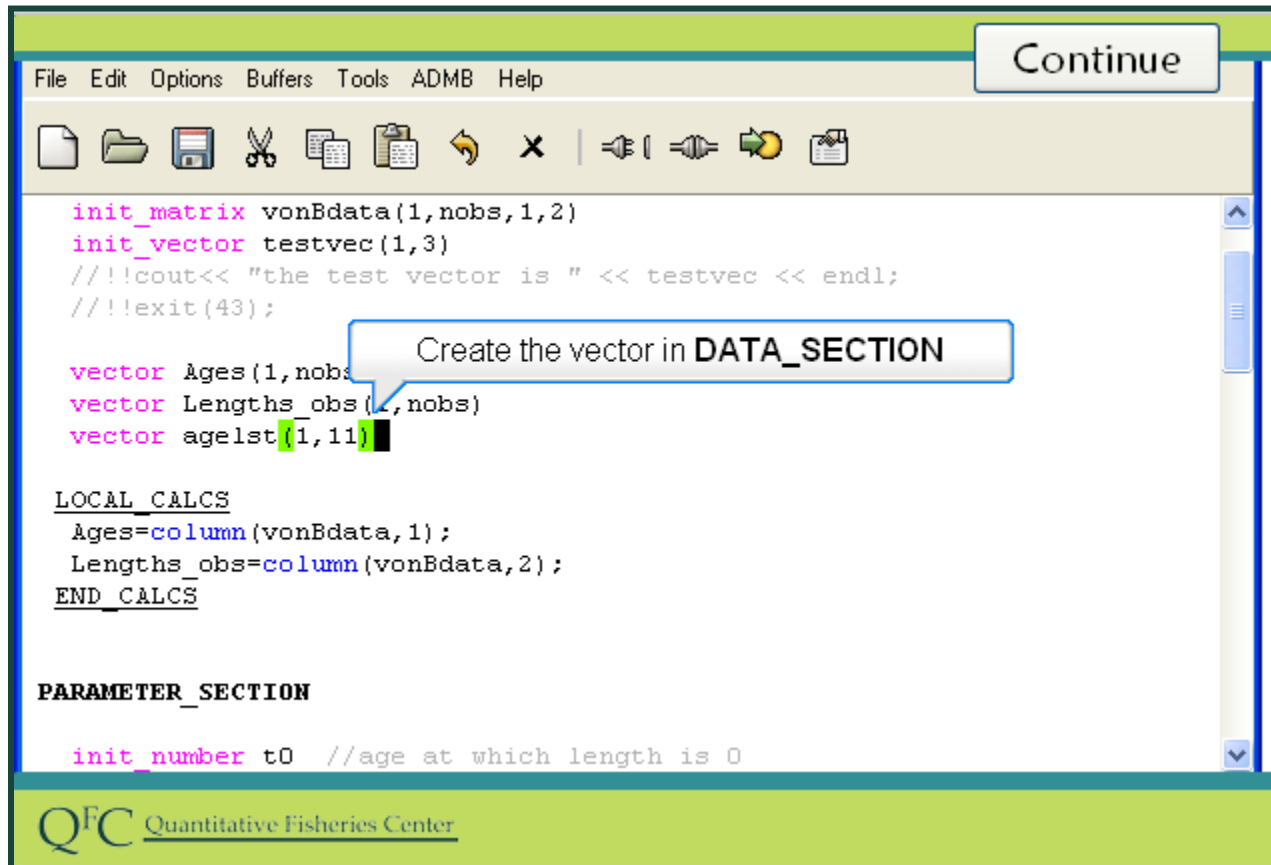
We will define an sd_report_vector for predicted lengths. We create a new vector for this and intend it to hold predictions for ages of 1 through 11.

Any variable we define as sd_report type will automatically have asymptotic standard errors estimated for it and be incorporated into the asymptotic correlation matrix. We create this rather than just redefining the type for the predicted values for the data because that other vector has repeat values for the same age and we only need to assess the uncertainty for an age once. Also we can generate uncertainty for predicted lengths given age even for an age that is not seen in the data, here for example age 1. Of course these are only believable if its reasonable that our same model would apply to those unseen ages.

Slide Code:

```
sdreport_vector predlst(1,11)
```

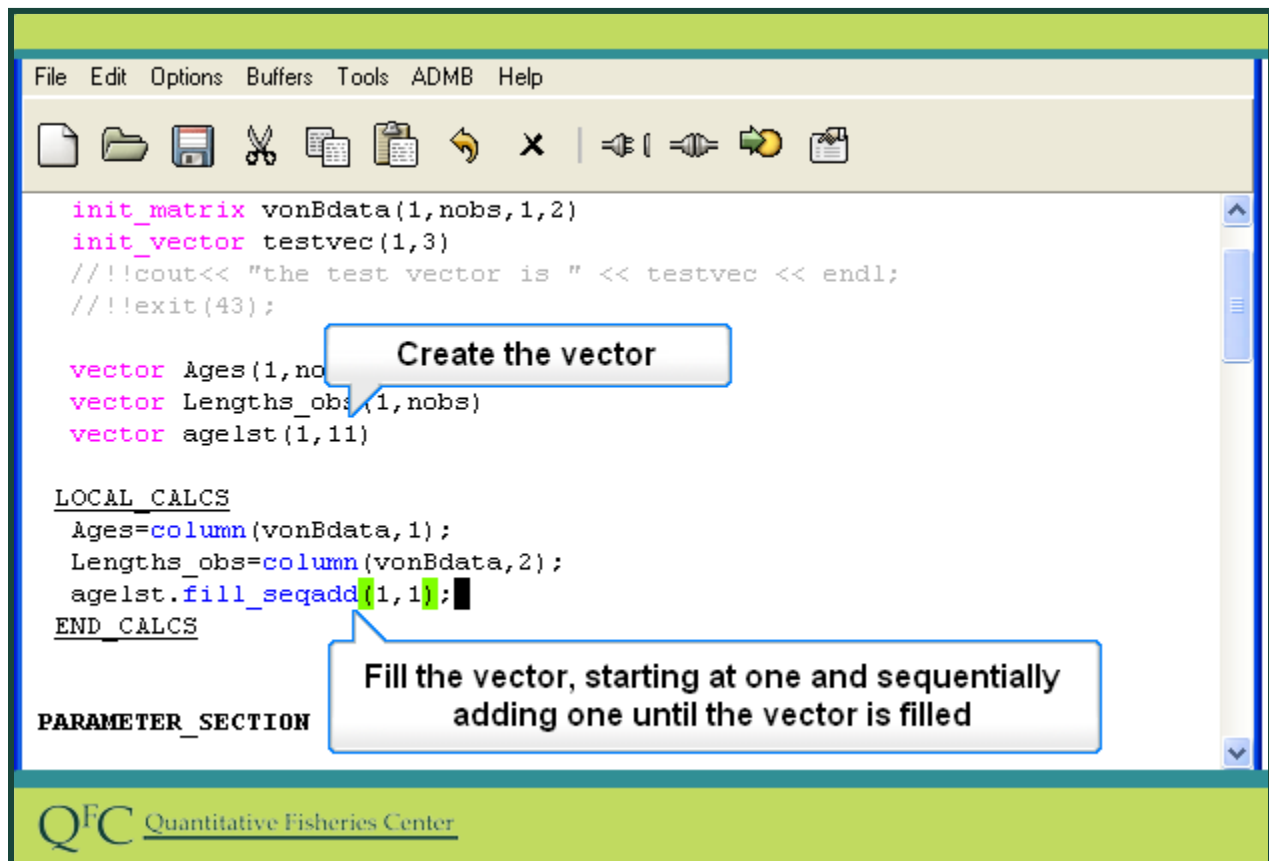
Data Section



We also create a vector to hold ages we want the predictions for, which should just get filled with values of 1 through 11. We define this vector up in the data section and fill it with values there because the ages don't depend upon the parameters or change values during the procedure section.

Slide Code:

```
vector agelst(1,11)
```



Then within the local calcs block in the data section we add a line a code to fill the vector with the desired values.

This syntax applies `fill_seqadd` to the vector `agelist` and fills it with an additive sequence starting at 1, the first argument, and sequentially adding 1 to the value it puts into each subsequent element until the vector is filled. Admb has a number of other fill commands like this described in the manual, which can be very useful.

Slide Code:

```
agelst.fill_seqadd (1,1);
```


Procedure Section

```
File Edit Options Buffers Tools ADMB Help
Continue

//cout << "we are at top of procedure section" << endl;
//exit(44);
Linf=exp(log_Linf);
K=exp(log_K);
sigma=exp(log_sigma);

//cout<<"print starting values for Linf, K, t0, and sigma, and quit"<<endl;
//cout<<Linf<<" " <<K<<" " <<t0<<" " <<sigma<<endl;
// exit(45);

Le
re

predlst=Linf*(1.-exp(-K*(agelst-t0)));

// cout<<"predicted lengths
//cout<<Lengths_pred<<endl;
//cout<<"residuals"<<endl;
```

Use predlst in place of Lengths_pred

use agelst in place of Ages

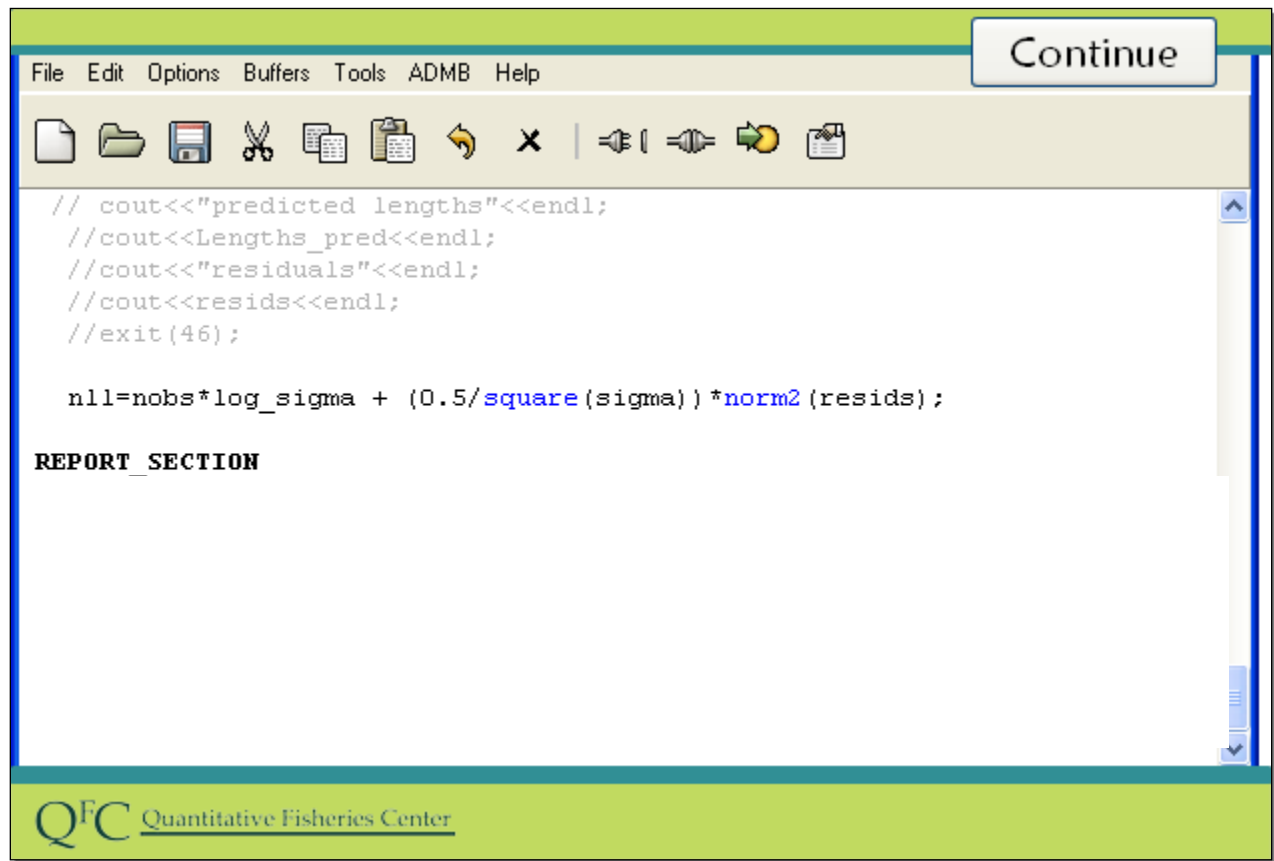
QFC Quantitative Fisheries Center

Slide notes

Next we go down to the procedure section and calculate the predicted values for these 11 ages following basically the same procedure used to calculate predicted values of the observed ages now just using agelst in place of Ages and predlst in place of Lengths_pred.

Slide Code:

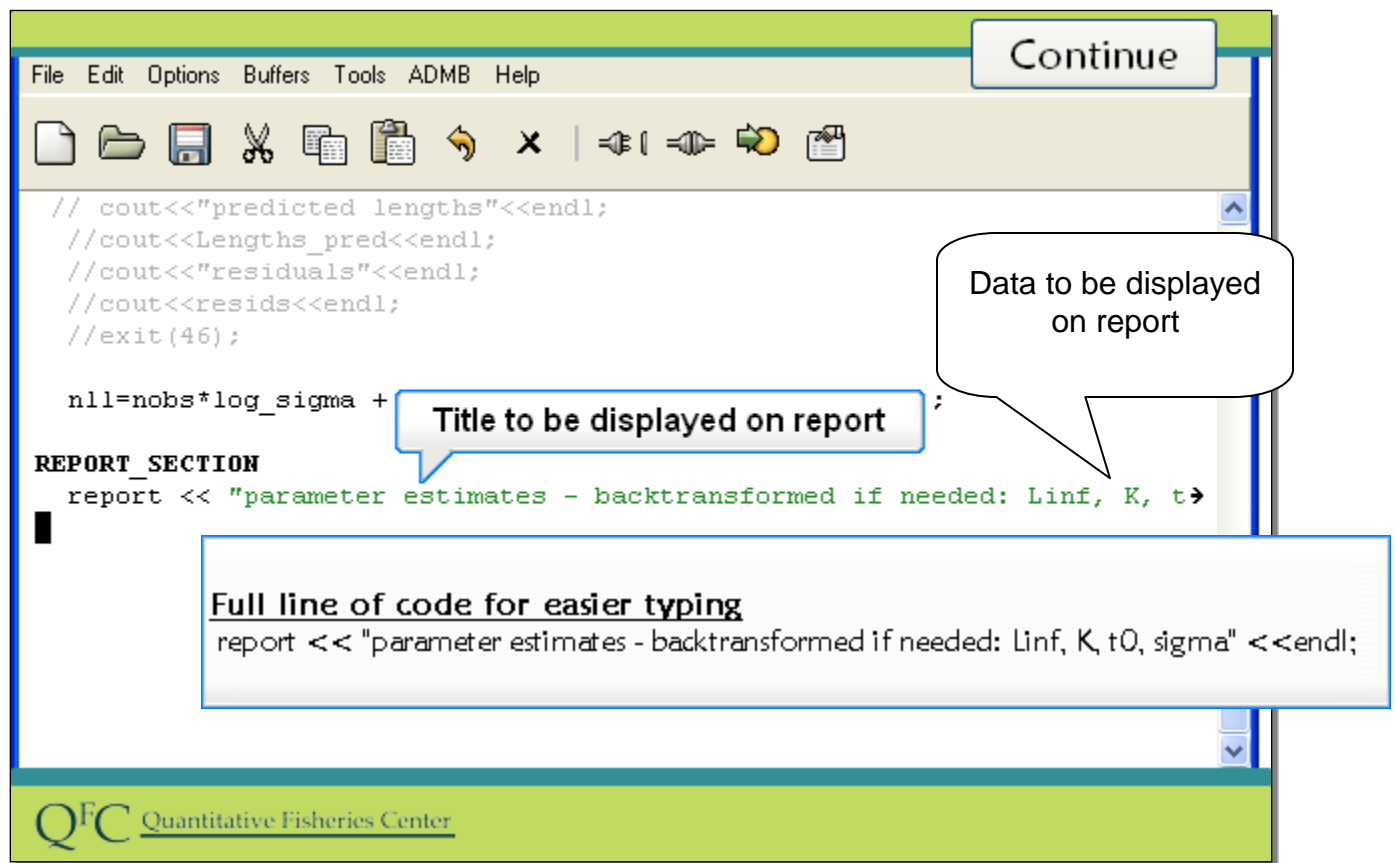
```
predlst=Linf*(1.-exp(-K*(agelst-t0)));
```



Finally we add a report section (directly under nll function in the Procedure Section) and simply write out some things we have calculated based on the parameters and are interested in seeing when the program is done. Begin the report section in all capitals and again there are no indents for section names.

Slide Code:

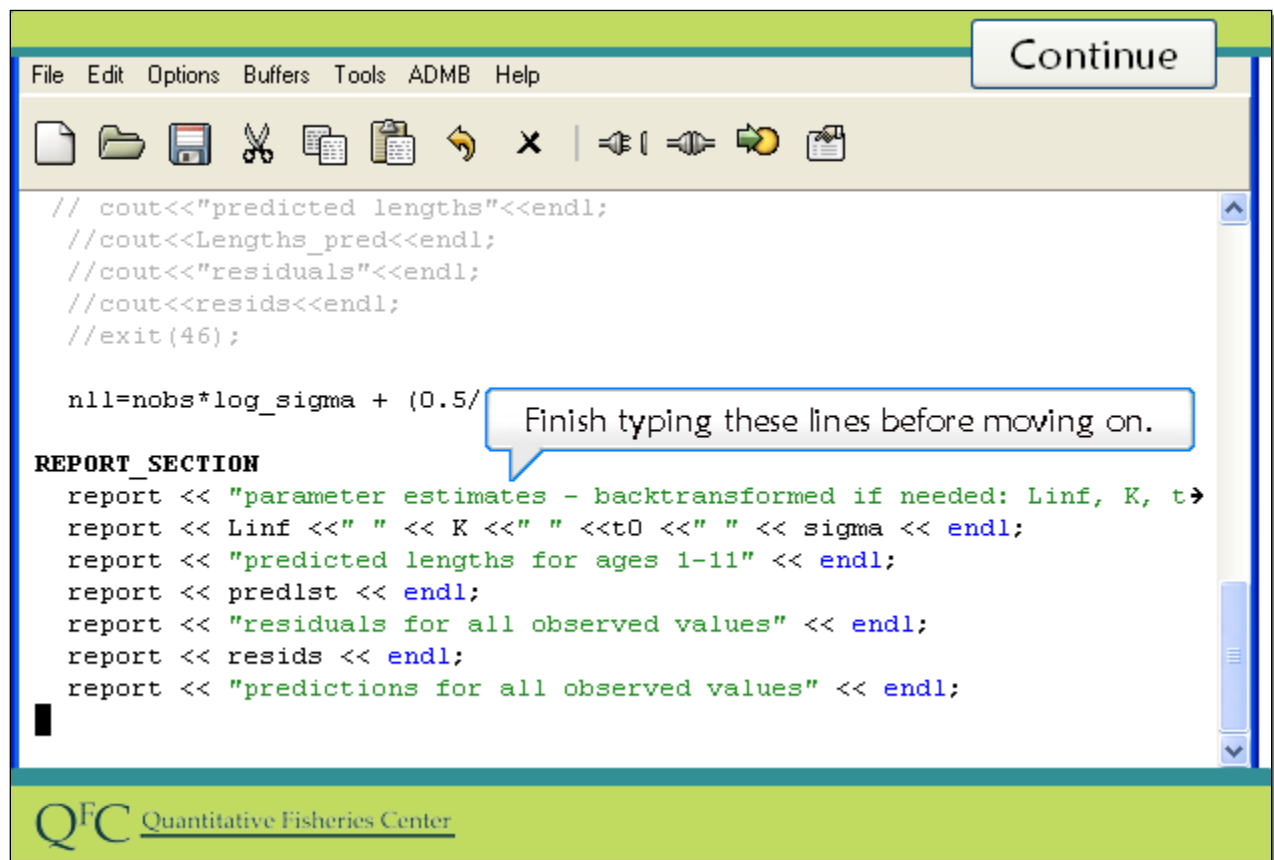
REPORT_SECTION



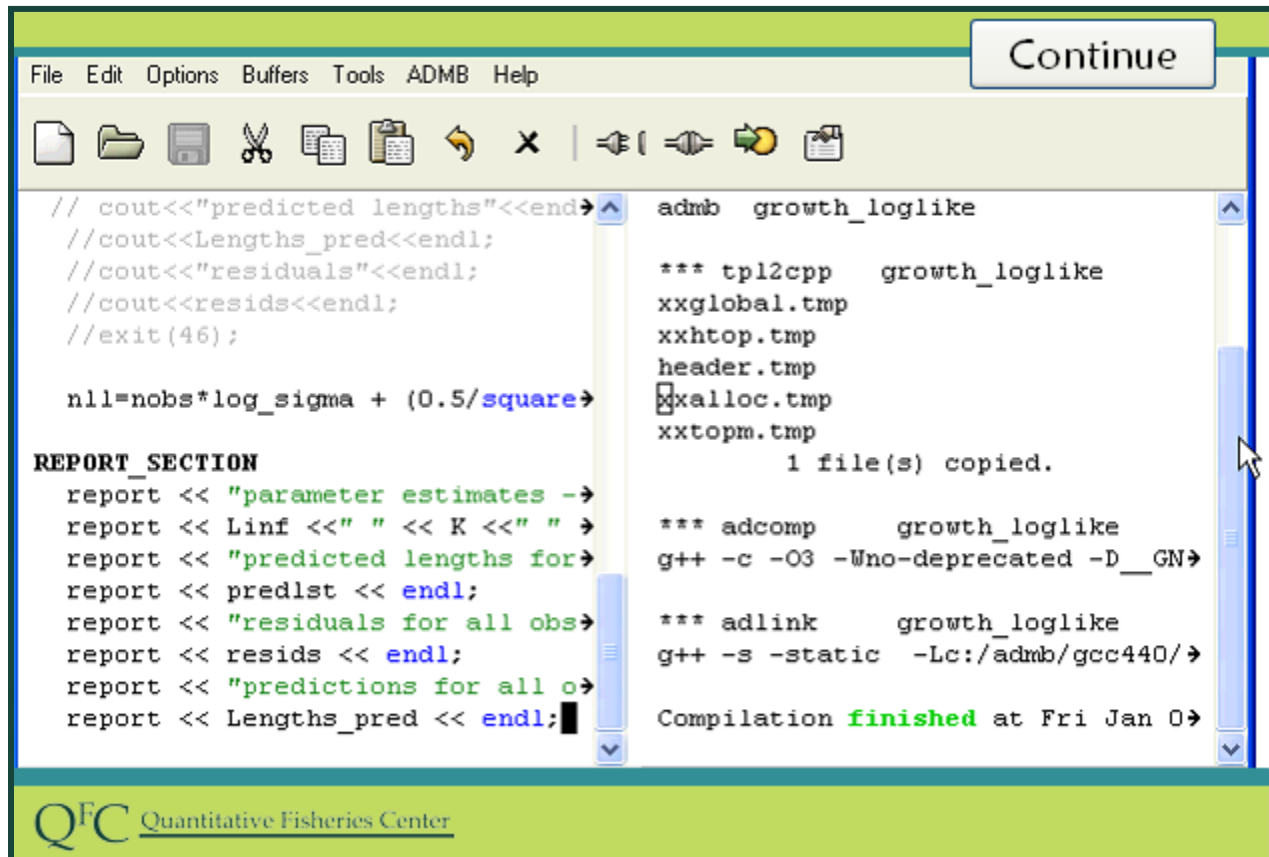
The lines of code we add within this section that write out output follow basically the same syntax we used with the cout command but now we replace cout with report.

Slide Code:

```
report << "parameter estimates - backtransformed if needed: Linf, K, t0, sigma" <<endl;
```

**Slide Code:**

```
report << Linf << " " << K << " " << t0 << " " << sigma << endl;
report << "predicted lengths for ages 1-11" << endl;
report << predlst << endl;
report << "residuals for all observed values" << endl;
report << resids << endl;
report << "predictions for all observed values" << endl;
```



The screenshot shows the ADMB software interface. The top menu bar includes File, Edit, Options, Buffers, Tools, ADMB, and Help. A toolbar with various icons is located below the menu. The main window is divided into two panes. The left pane contains C++ code for reporting results, including comments and a REPORT_SECTION block. The right pane shows the compilation process, with messages from tpl2cpp, adcomp, and adlink, indicating that the compilation is finished.

```
// cout<<"predicted lengths"<<endl;
//cout<<Lengths_pred<<endl;
//cout<<"residuals"<<endl;
//cout<<resids<<endl;
//exit(46);

nll=nobs*log_sigma + (0.5/square)

REPORT_SECTION
report << "parameter estimates ->
report << Linf <<" " << K <<" " >
report << "predicted lengths for>
report << predlst << endl;
report << "residuals for all obs>
report << resids << endl;
report << "predictions for all o>
report << Lengths_pred << endl;
```

```
admb growth_loglike

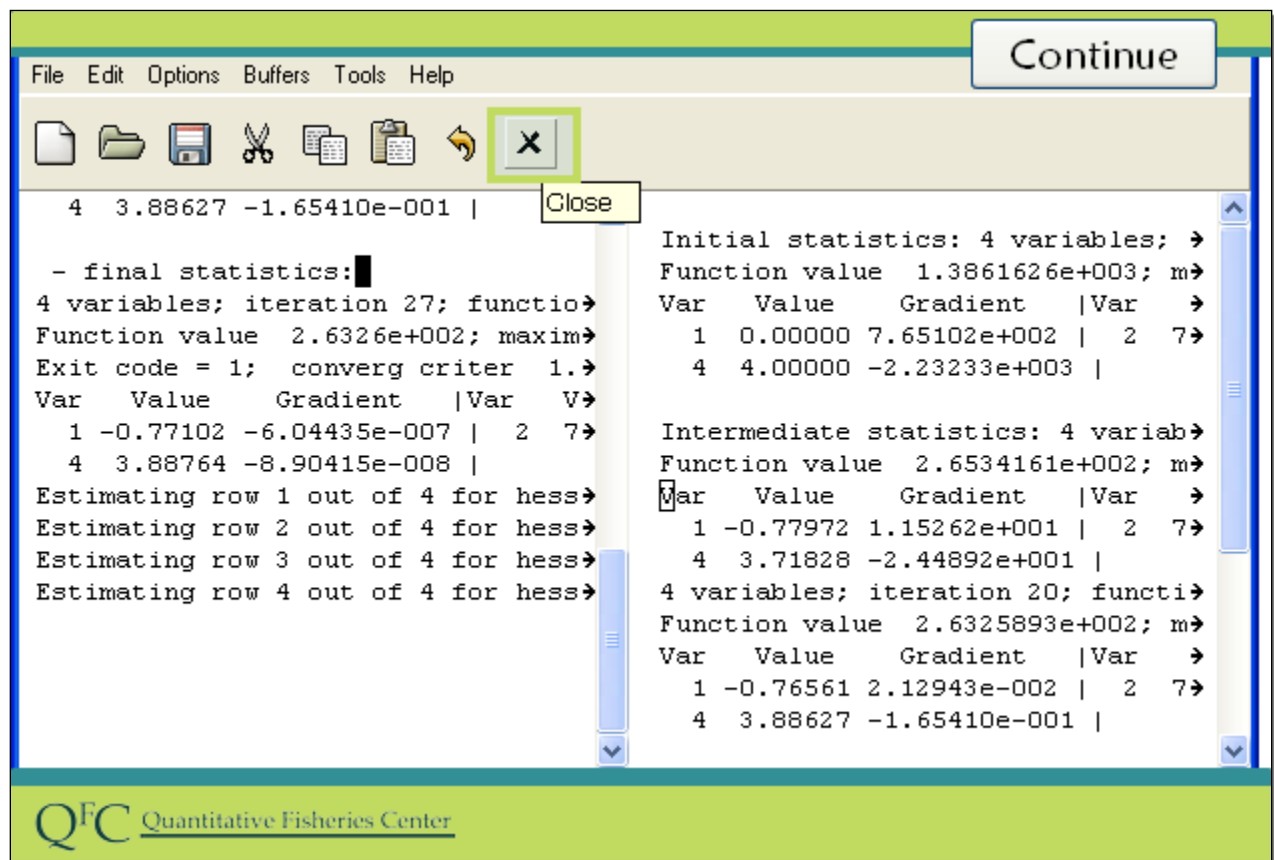
*** tpl2cpp growth_loglike
xxglobal.tmp
xxhtop.tmp
header.tmp
xalloc.tmp
xxtopm.tmp
1 file(s) copied.

*** adcomp growth_loglike
g++ -c -O3 -Wno-deprecated -D__GN>

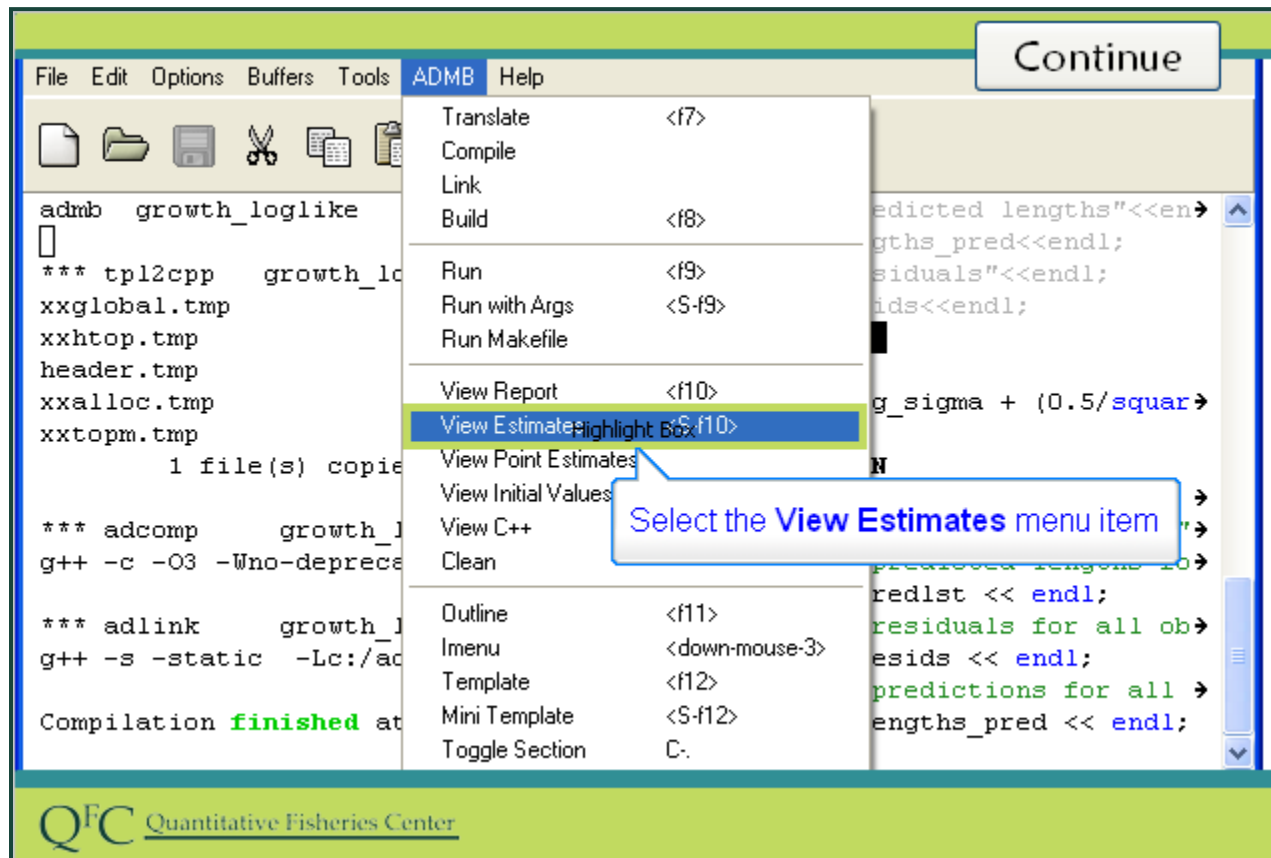
*** adlink growth_loglike
g++ -s -static -Lc:/admb/gcc440/>

Compilation finished at Fri Jan 0>
```

We are now ready to run our final version of the program and see what we get. Again build it. We look at messages and it seems like it was built correctly.



We now run it which seems to correctly run to obtain final estimates. Close out.



Now when we view estimates through the admb menu

File Edit Options Buffers Tools Help

Continue

admb growth_loglike

*** tpl2cpp growth_loglike
xxglobal.tmp
xxhtop.tmp
header.tmp
xxalloc.tmp
xxtopm.tmp
1 file(s) copied.

*** adcomp growth_loglike
g++ -c -O3 -Wno-deprecated -D__GNU__

*** adlink growth_loglike
g++ -s -static -Lc:/admb/gcc440/lib

Compilation finished at Fri Jan 07

The logarithm of the determinant

index	name	value	s
1	t0	-7.7102e-00	3
2	log_Linf	7.1489e+000	4
3	log_K	-1.2456e+000	1
4	log_sigma	3.8876e+000	9
5	predlst	5.0815e+002	3
6	predlst	6.9932e+002	1
7	predlst	8.4269e+002	8
8	predlst	9.5021e+002	1
9	predlst	1.0308e+003	1
10	predlst	1.0913e+003	1
11	predlst	1.1367e+003	1
12	predlst	1.1707e+003	1
13	predlst	1.1962e+003	2
14	predlst	1.2153e+003	2
15	predlst	1.2296e+003	3

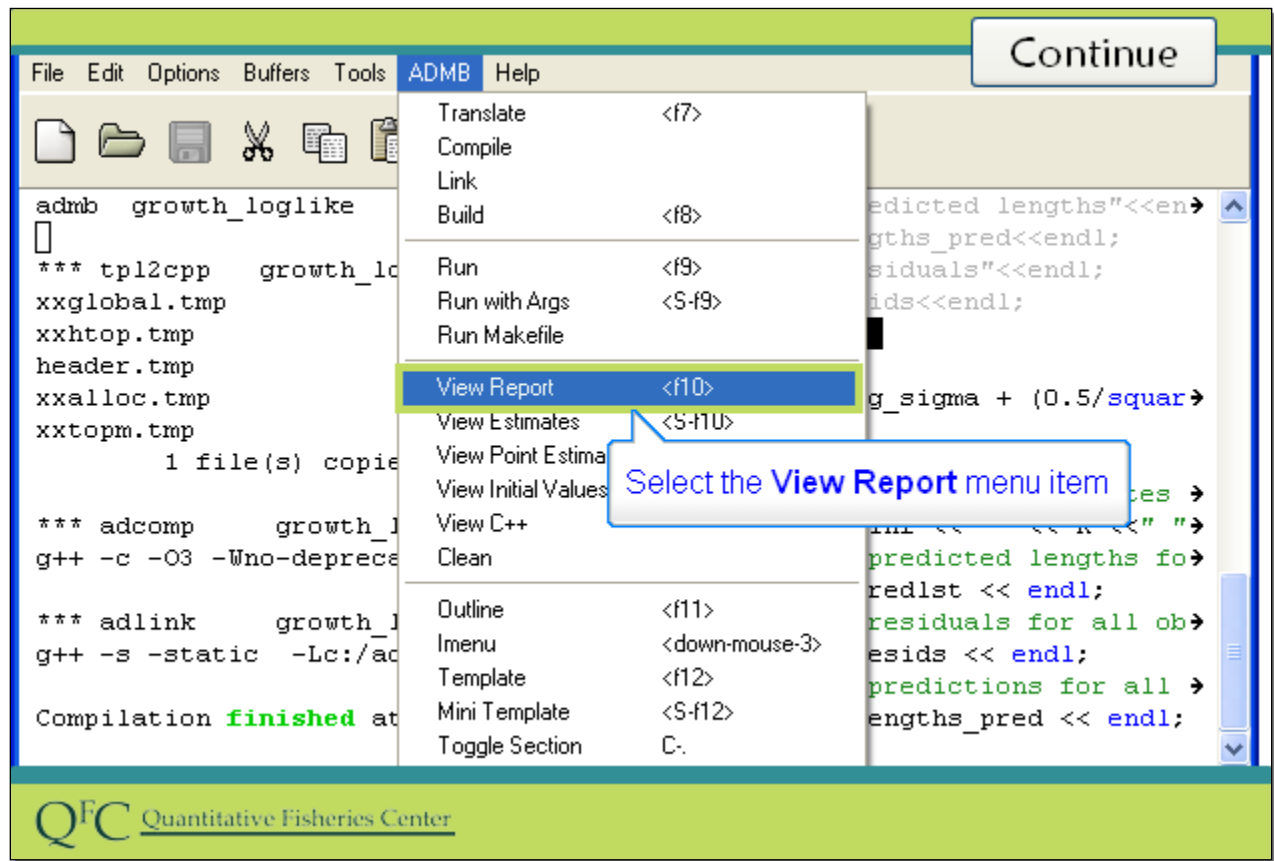
Asymptotic standard errors for the 11 elements of predlst

QFC Quantitative Fisheries Center

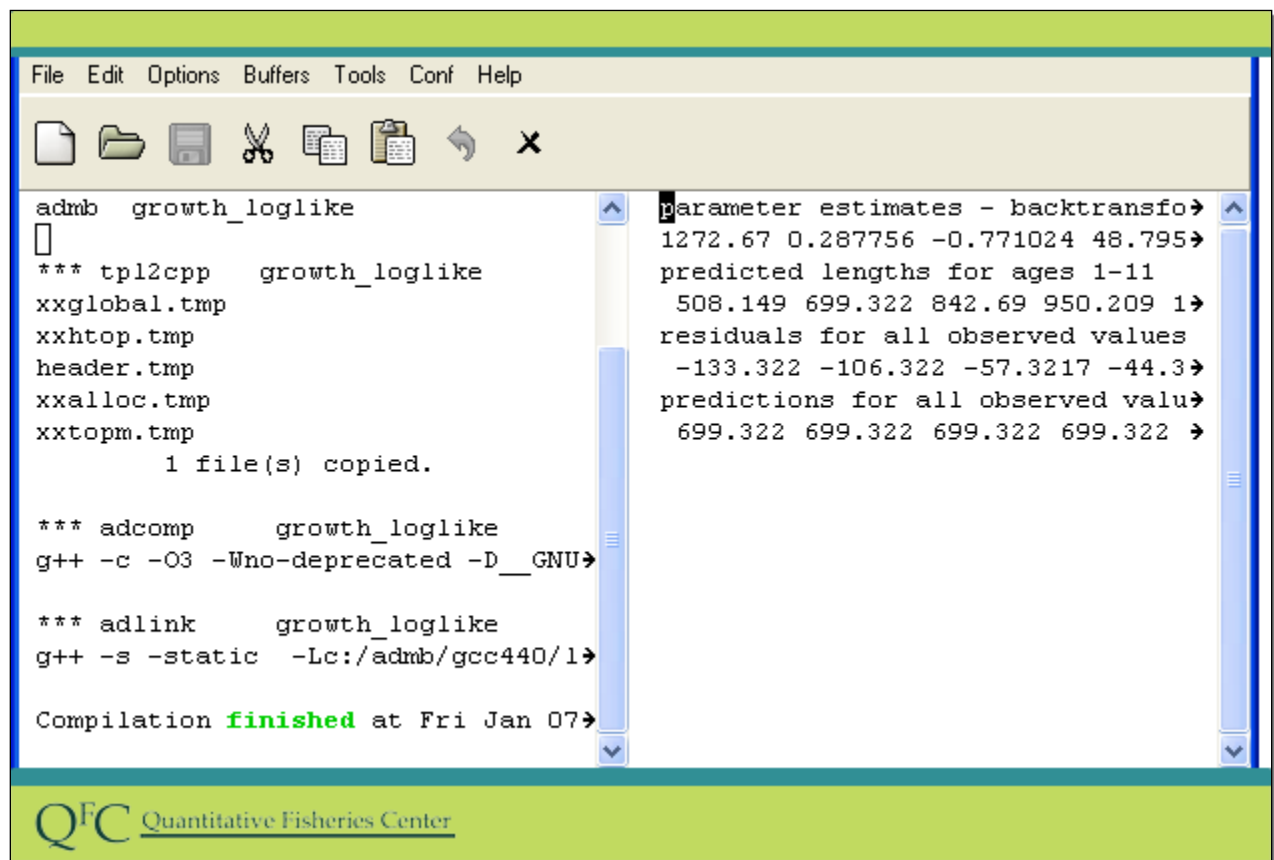
We see that the resulting file contains estimates and asymptotic standard errors for the 11 elements of predlst and that the 11 calculated predlst values are also now part of the asymptotic correlation matrix.

As a technical note I note that these are calculated using the delta method based on the asymptotic variance covariance matrix for the parameters.

Close out



We can also use the admb menu to view the report file,

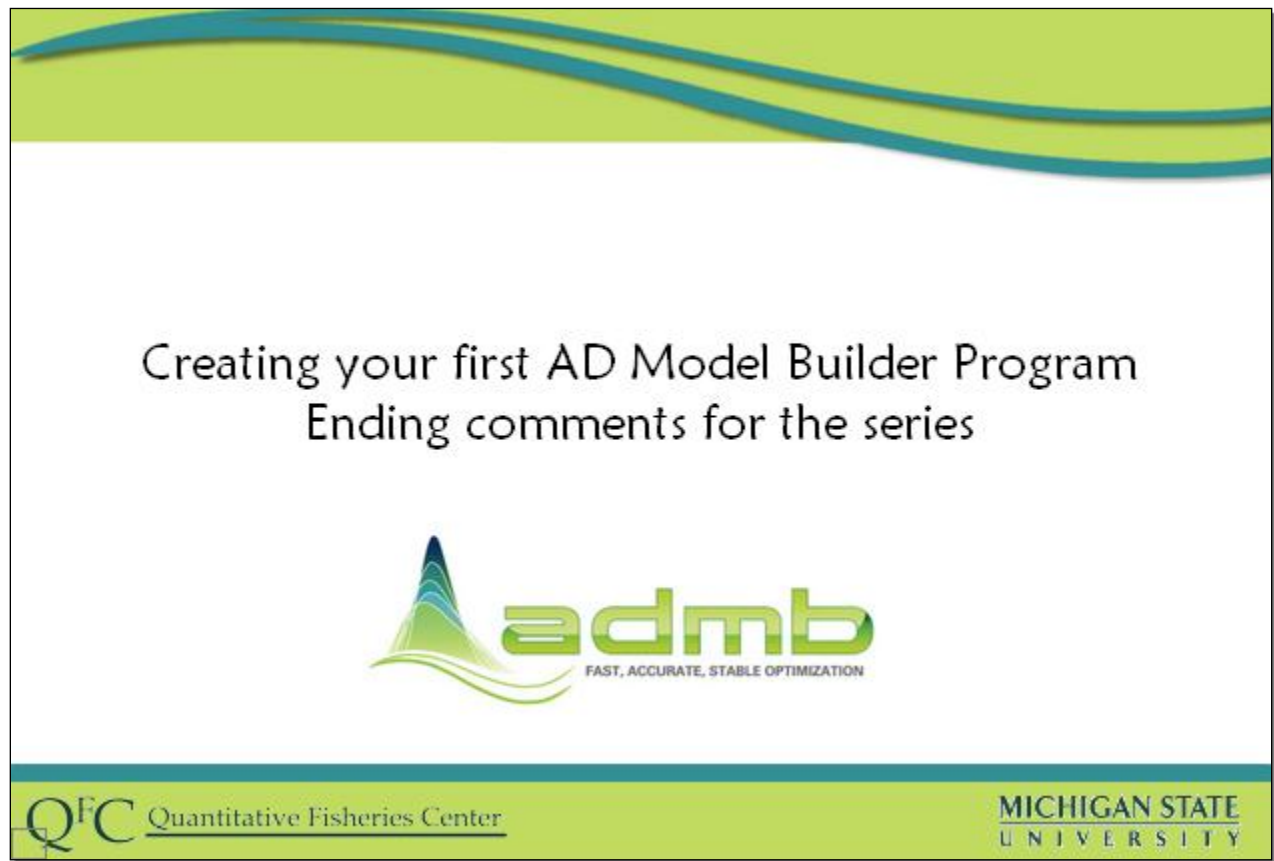


```
admb growth_loglike
*** tpl2cpp growth_loglike
xxglobal.tmp
xxhtop.tmp
header.tmp
xxalloc.tmp
xxtopm.tmp
1 file(s) copied.

*** adcomp growth_loglike
g++ -c -O3 -Wno-deprecated -D__GNU__
*** adlink growth_loglike
g++ -s -static -Lc:/admb/gcc440/lib
Compilation finished at Fri Jan 07

parameter estimates - backtransfo
1272.67 0.287756 -0.771024 48.795
predicted lengths for ages 1-11
508.149 699.322 842.69 950.209 1
residuals for all observed values
-133.322 -106.322 -57.3217 -44.3
predictions for all observed valu
699.322 699.322 699.322 699.322
```

which has the output we requested to be printed to the report file.



We are now done with this set of videos. Feel free to explore the results further to evaluate whether there are patterns in residuals suggesting perhaps a different model should be considered either for the predicted relationship or for the distribution of errors. You could also easily try other starting values to check to see if the model converges back to the same parameter estimates. We hope this step by step implementation of an admb program will help you get started using this software package. There is much more involved in getting this nonlinear regression implemented in admb than would be the case using canned software such as Proc NLS in SAS or the nls function in R. However admb is much more flexible as you consider alternative models and our intent here was to help you work through the coding with a very simple example model.