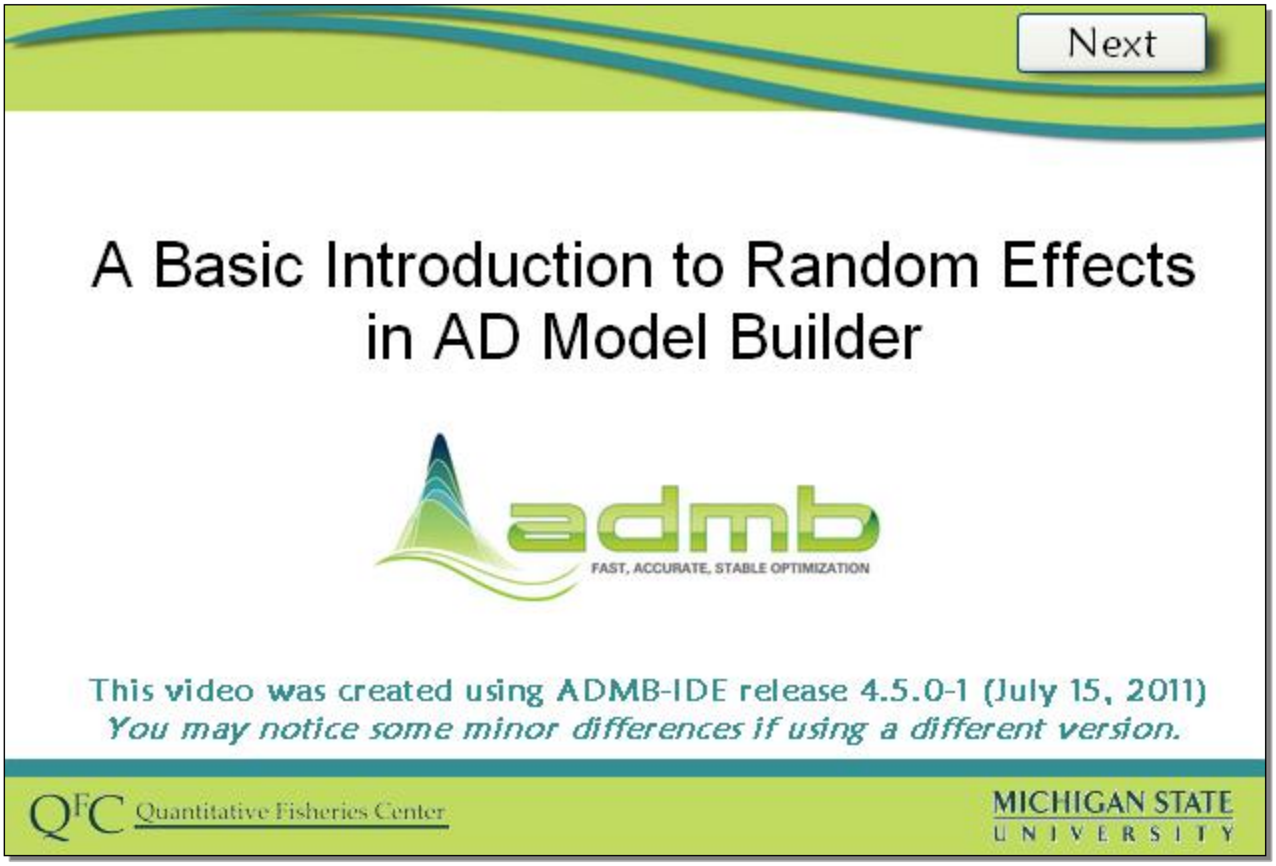
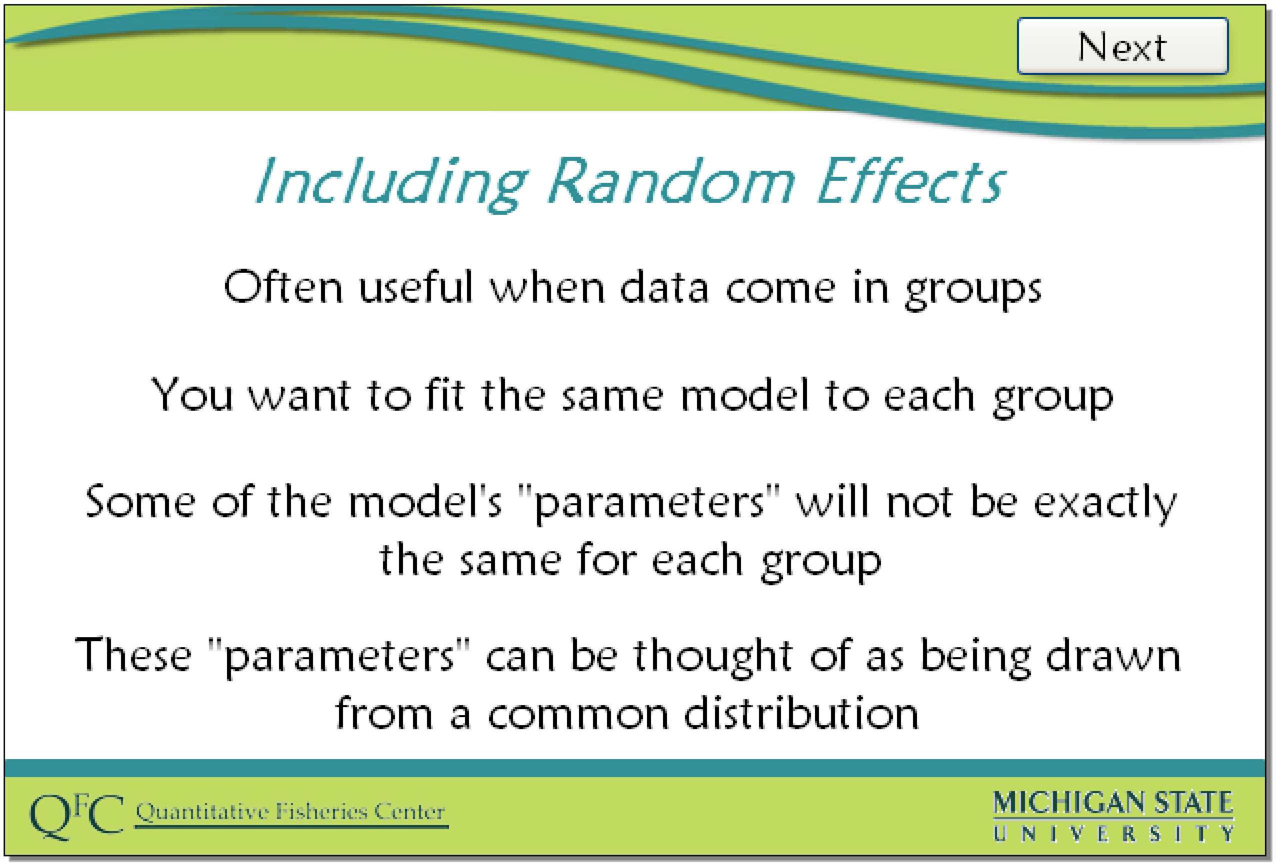
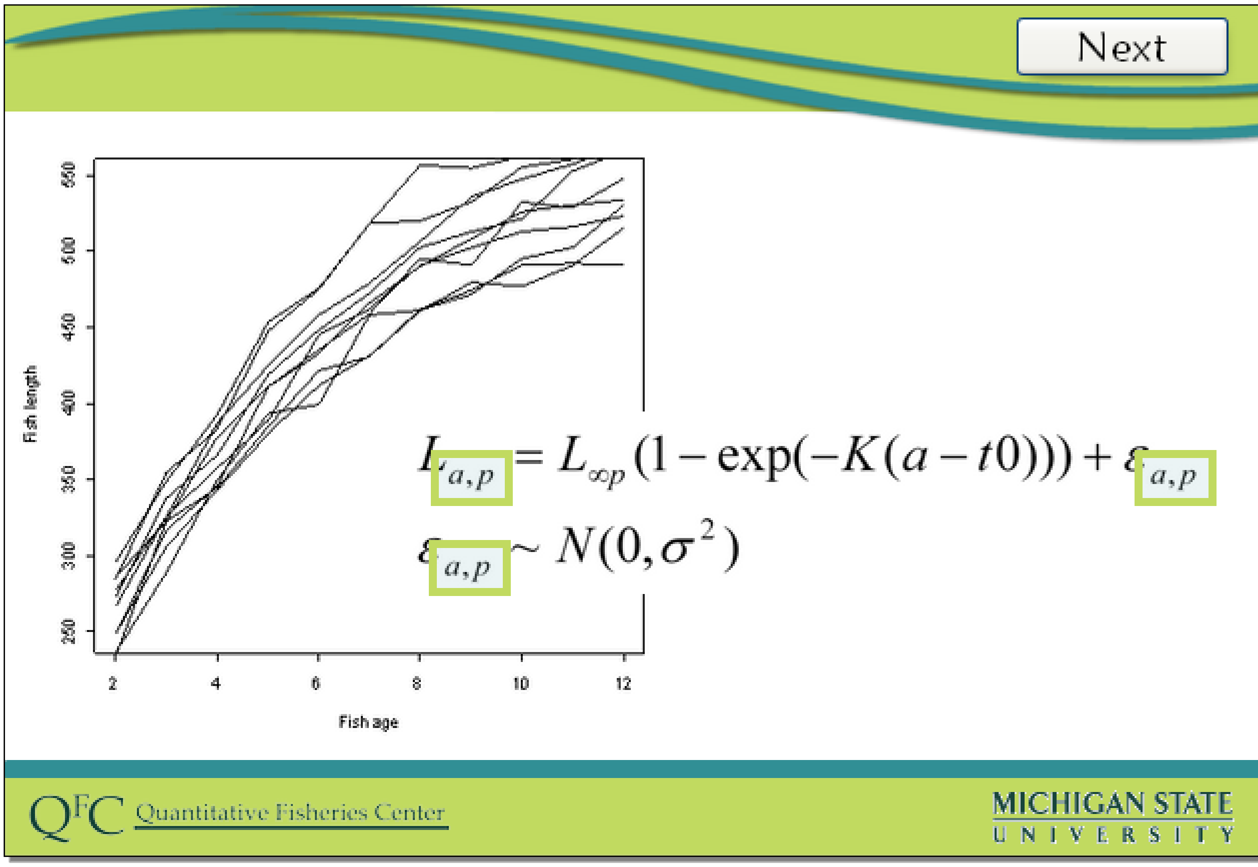


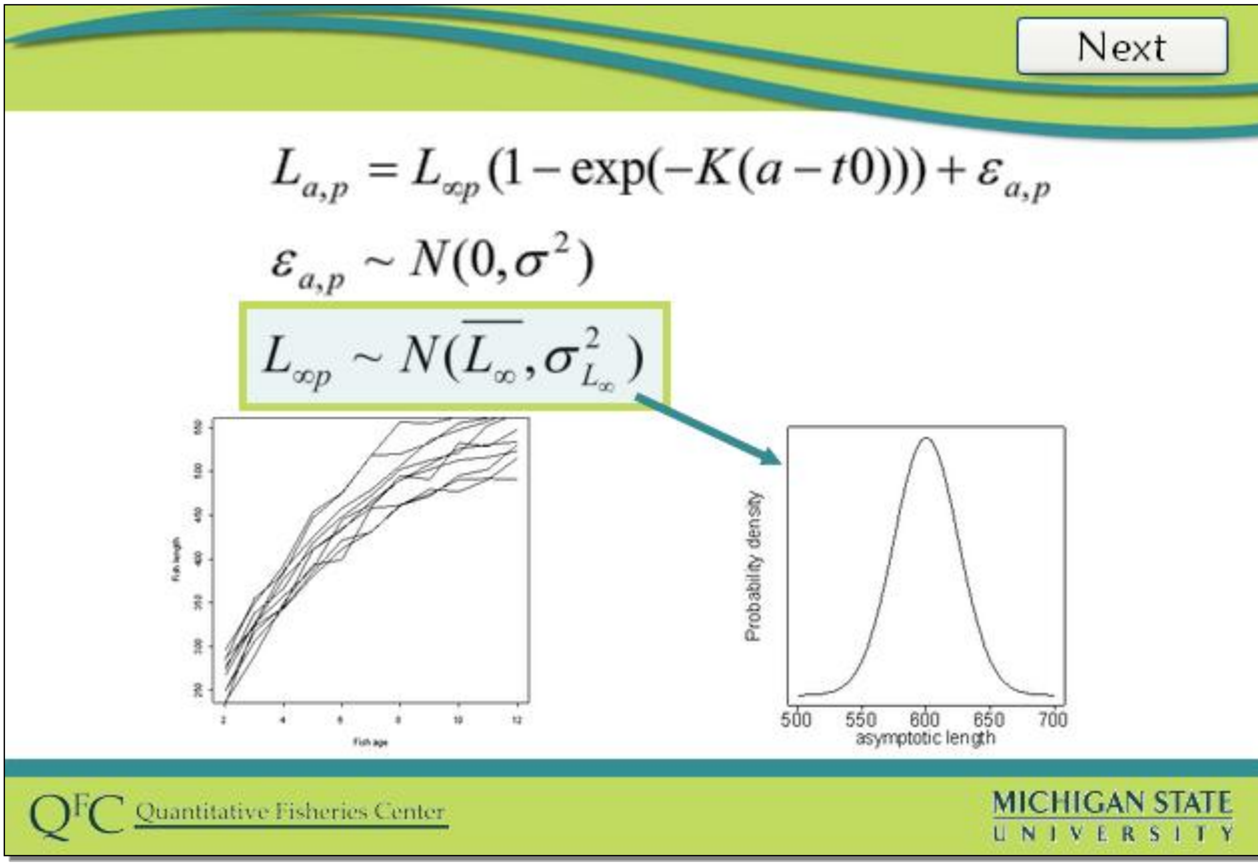
This video shows how to incorporate random effects, also known as random coefficients, into your statistical models using AD Model Builder.



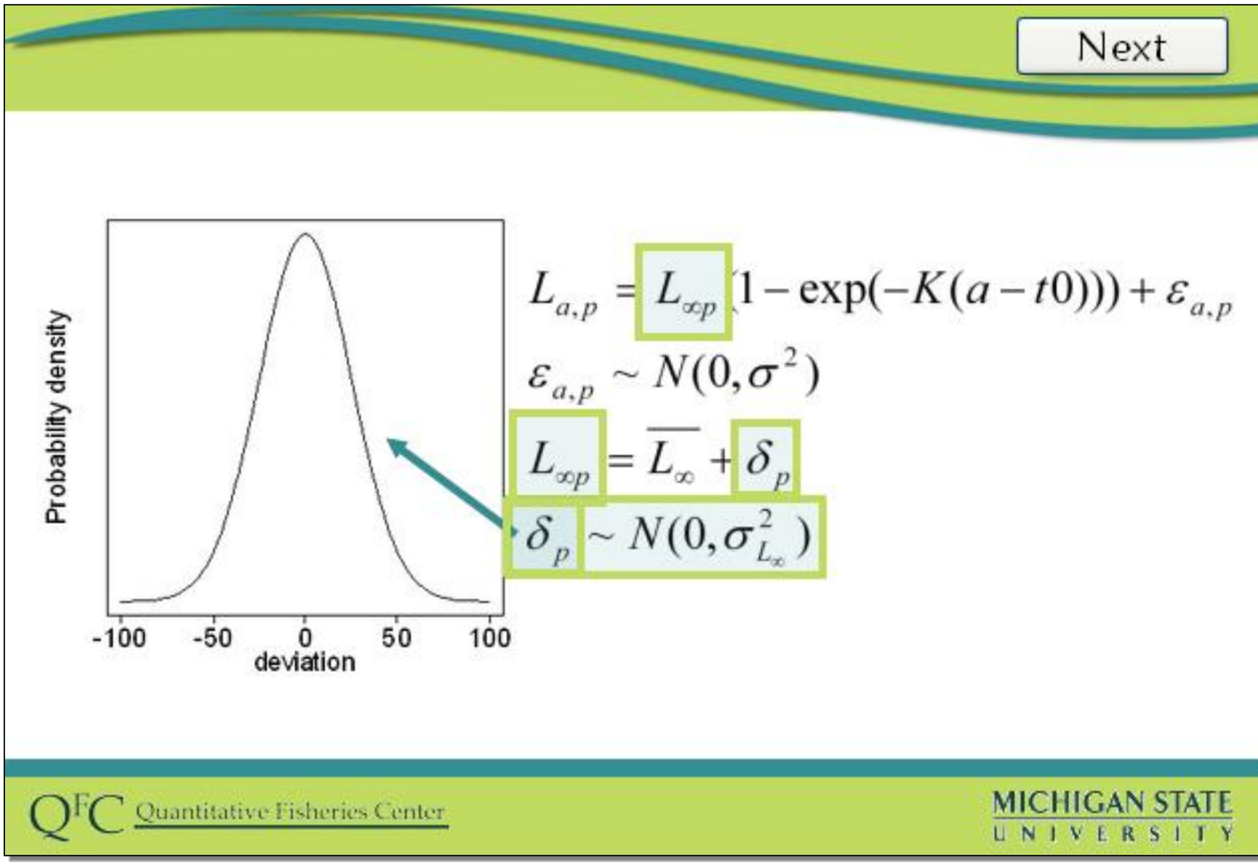
Random effects can be especially useful when your data naturally fall into groups where the functional forms describing relationships are the same for each group, but you do not expect all of the so-called parameters to be identical. If it is reasonable to think of the parameters describing the same thing, say asymptotic length, as being drawn from a common probability distribution, this makes these random effects. For a frequentist, we would no longer call these parameters, because parameters cannot be random.



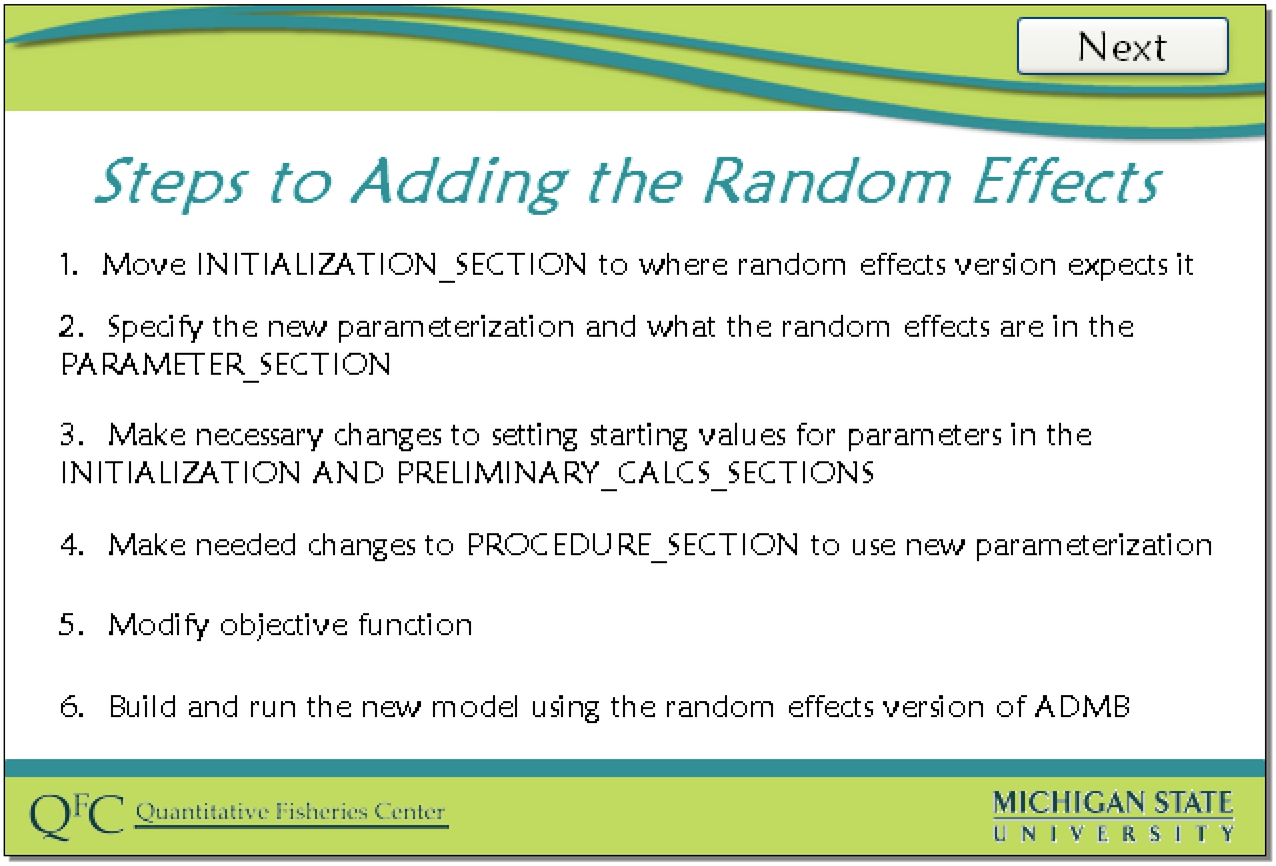
We previously fit the data using the von Bertalanffy model represented by this equation. In this equation, we only have subscripts for age and pond and not individual observation because our data have exactly one observation per pond and age combination. When we previously fit these data we freely estimated ten asymptotic lengths, one for each pond. Our equation says that the observed length depends on the pond specific asymptotic length, and common Brody growth coefficient K and t naught parameters, as well as observational error. We have previously made no probability claims about how the L-infinity values are related to one another.



We will now augment the model with a probability statement about the L-infinity values. In particular, we assume that the individual pond L-infinities come independently from a common normal distribution sharing a mean, L-infinity-bar, and sharing a variance, sigma squared L-infinity.



It is often convenient to treat the deviations from the mean as the random effects, rather than the original so-called parameters. Here the original parameters are represented by the L-infinities and the deviations are the deltas. In this case, each modeled random effect is still normal and still has variance sigma-squared sub L-infinity but each random effect has mean zero. Although the model is really unchanged, we cast it like this to more closely match how we are going to code the model. In this equation the observed data are the lengths represented by L sub a and p for each age and pond. The parameters are Linfinity-bar, K, t-naught, sigma-squared, and sigma-squared-sub Linfinity. The random effects will be the delta sub p's.



We will start with a tpl we had previous developed to fit the data with pond specific L-infinities in our videos on looping. The steps we follow to develop and run the random effects model include first moving the initialization section to where the random effects version expects it, and then specifying the random effects and modifying the declared parameters to match the random effects model in the parameter section. Once we have defined the parameters we need to make necessary changes to how the starting values for parameters are set in the initialization section. We then make needed changes to the procedure section to use the new way the model is parameterized, and deal with some specifics of the random effects version. Finally, also in the procedure section we modify the objective function to take into account the distributional assumption for random effects. Once we make all these changes, we build the program using the random effects version by selecting random effects mode, and run our new code. We will use cout and exit statements to test things one time before we are completely done.

Next

$$L_{a,p} = L_{\infty p}(1 - \exp(-K(a - t0))) + \varepsilon_{a,p}$$

$$\varepsilon_{a,p} \sim N(0, \sigma^2)$$

$$L_{\infty p} = \overline{L_\infty} + \delta_p$$

$$\delta_p \sim N(0, \sigma^2_{L_\infty})$$

$\overline{L_\infty}$ estimate on log - scale

K estimate on log - scale

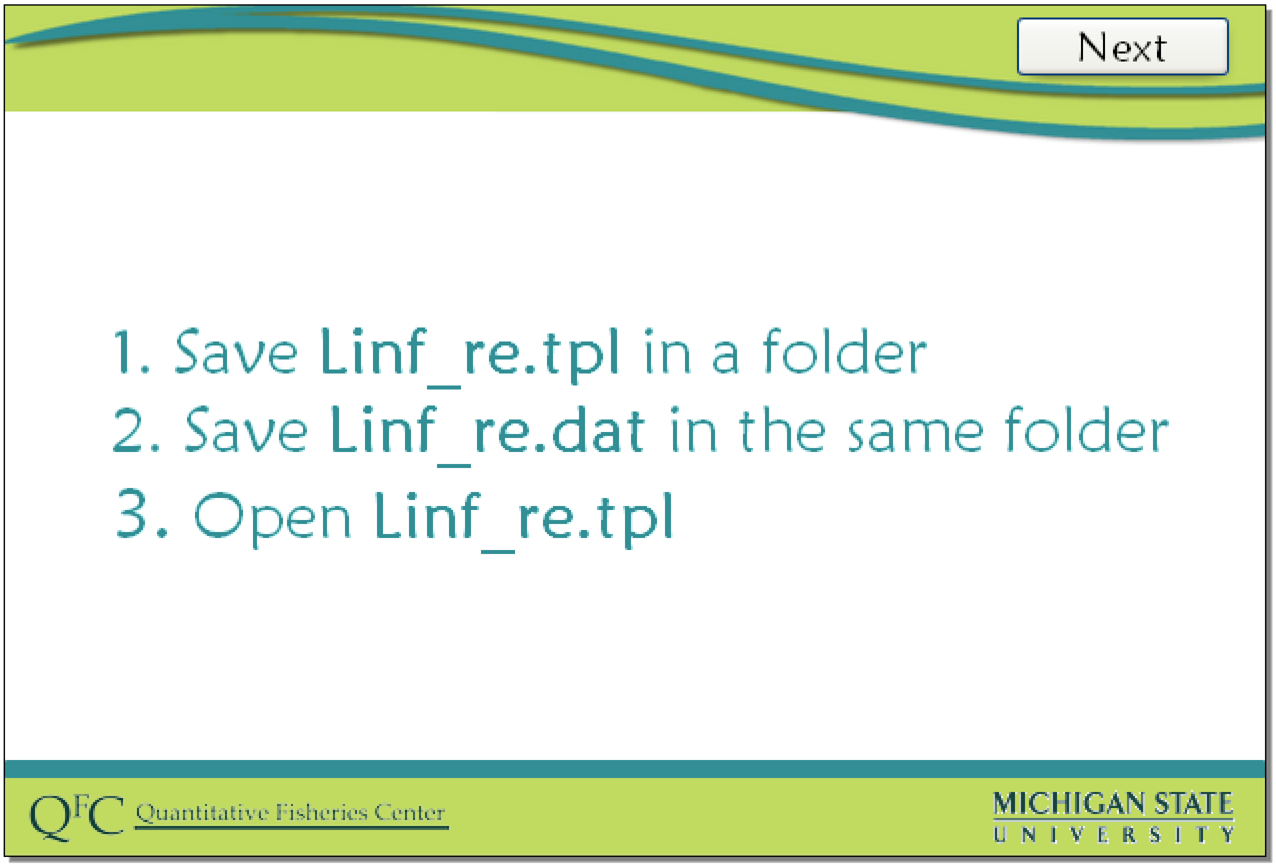
$t0$ estimate without change of scale

σ^2 estimate as log - scale standard deviation

$\sigma^2_{L_\infty}$ estimate as log - scale standard deviation

QFC Quantitative Fisheries Center MICHIGAN STATE UNIVERSITY

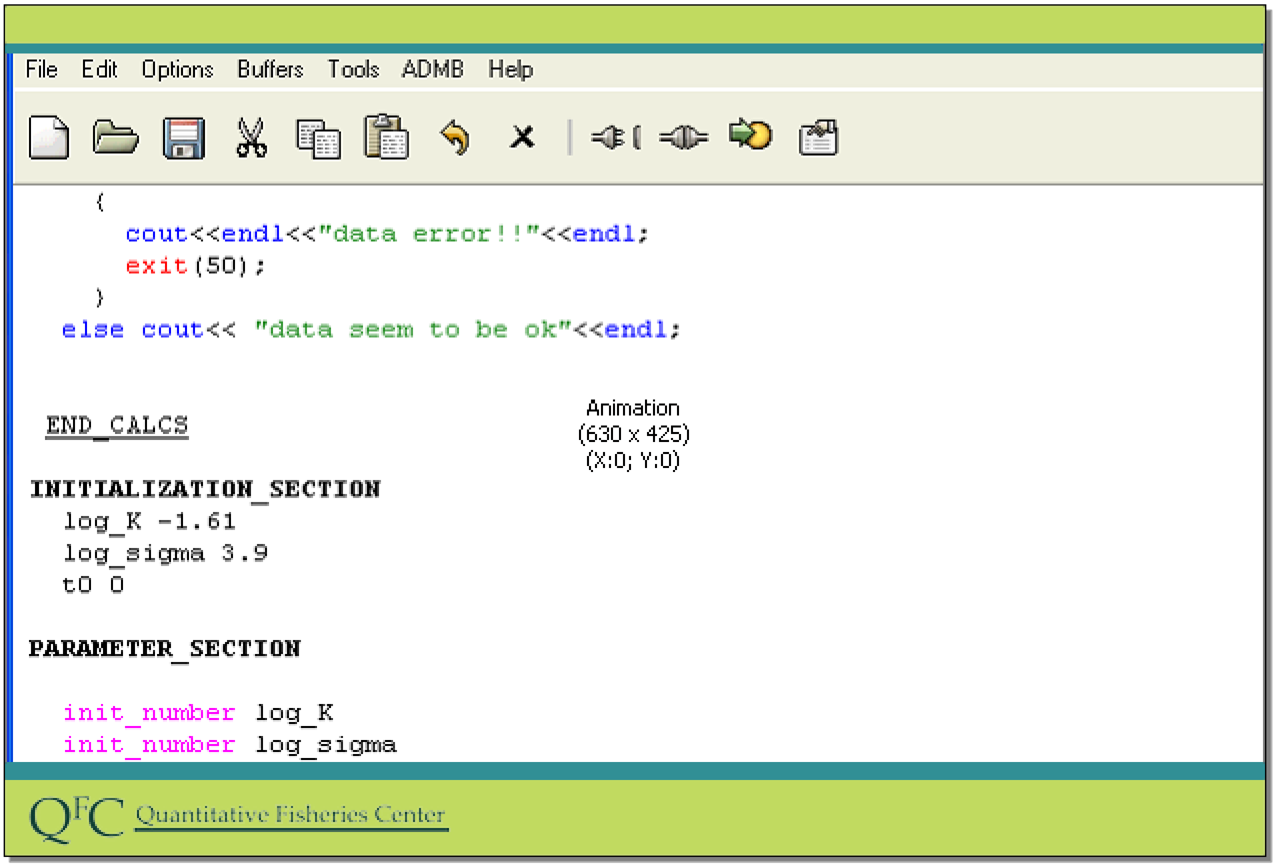
Our first task then is to modify the existing code so we now estimate the new parameters and declare the random effects. We will estimate Linfinity-bar and K on a log scale. We will estimate t-naught on its original scale because it can take negative values. We will estimate both the observational error variance and the random effect variances as log-scale standard deviations.



Complete these tasks if you want to follow along and press next when you are ready.

Slide Action:

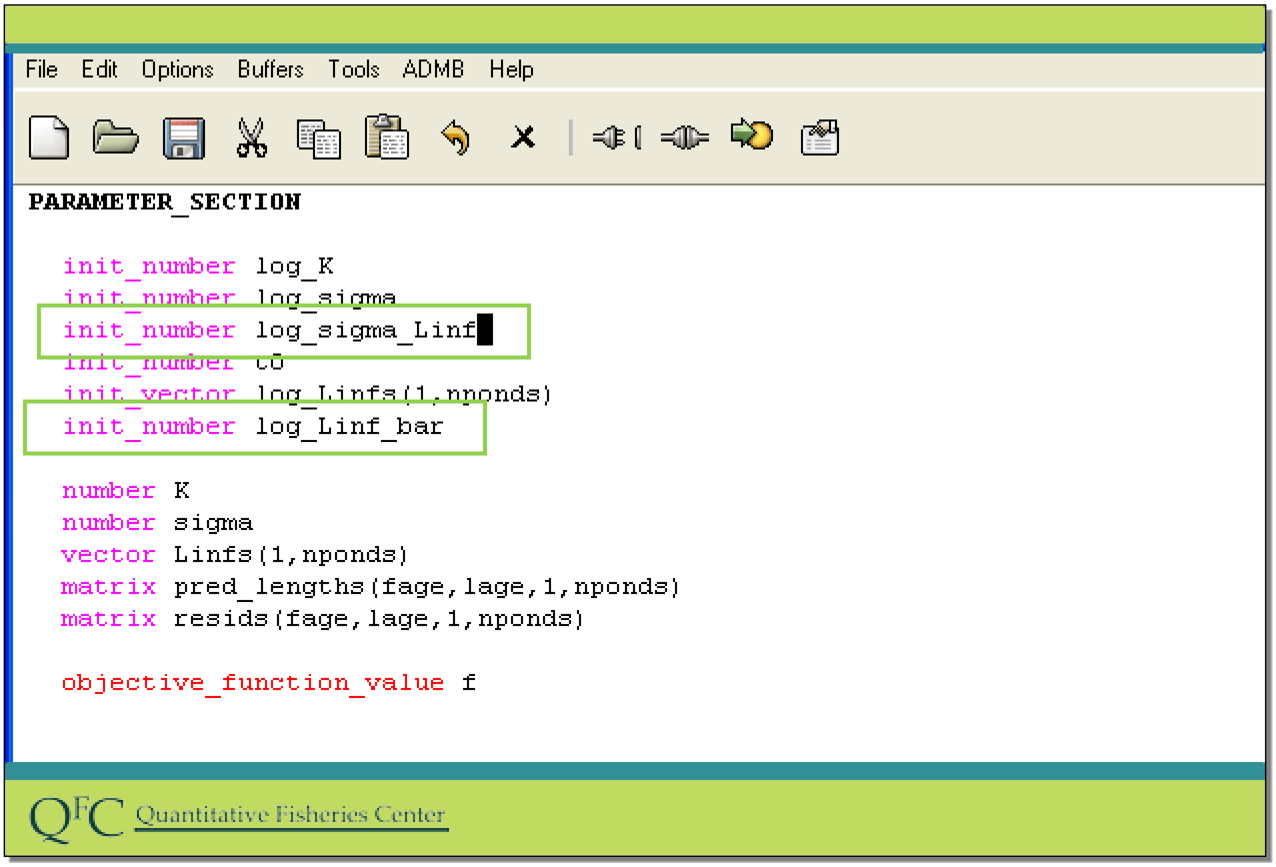
1. **Save Linf_re.tpl in a folder**
2. **Save Linf_re.dat in the same folder**
3. **Open Linf_re.tpl**



The random-effects version of AD model builder expects the initialization section to come before the parameter section. So we cut it from where it is and paste it in the expected place.

Slide Action:

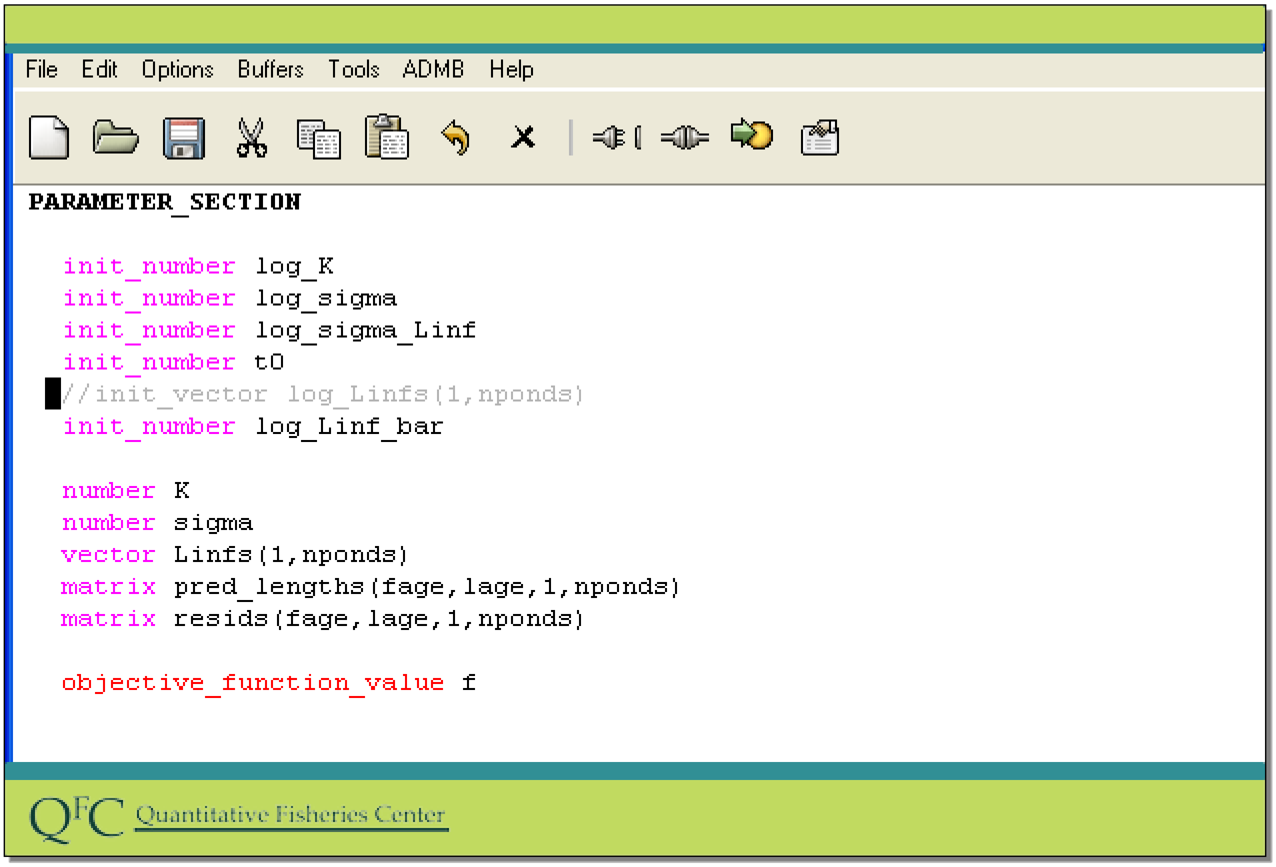
1. **Highlight and cut the INITIALIZATION_SECTION**
2. **Scroll up and paste it just above the PARAMETER_SECTION**



Next we will make changes in our PARAMETER_SECTION. We create the new parameters log_Linf_bar and log_sigma_Linf

Slide Code:

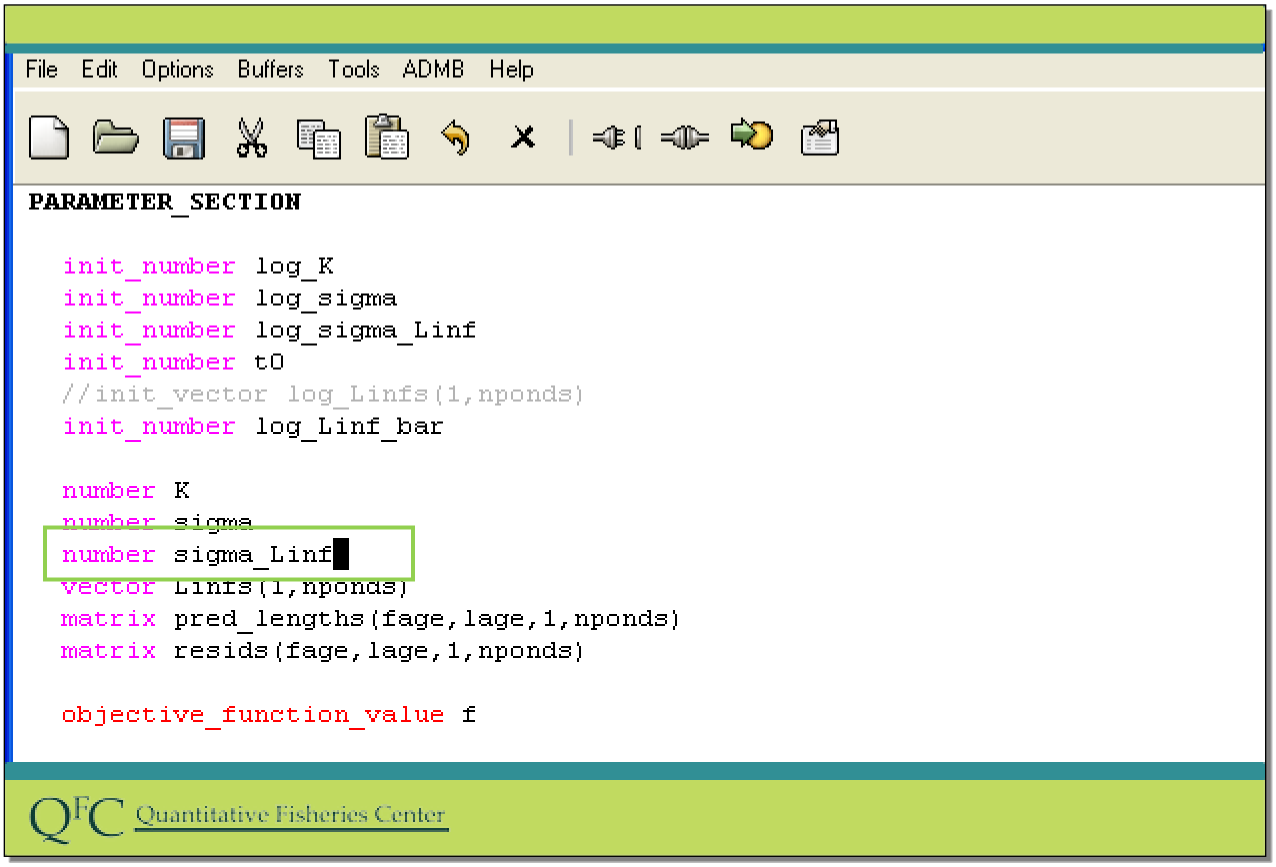
```
init_number log_Linf_bar
init_number log_sigma_Linf
```



and comment out the definition for log_Linfs.

Slide Code:

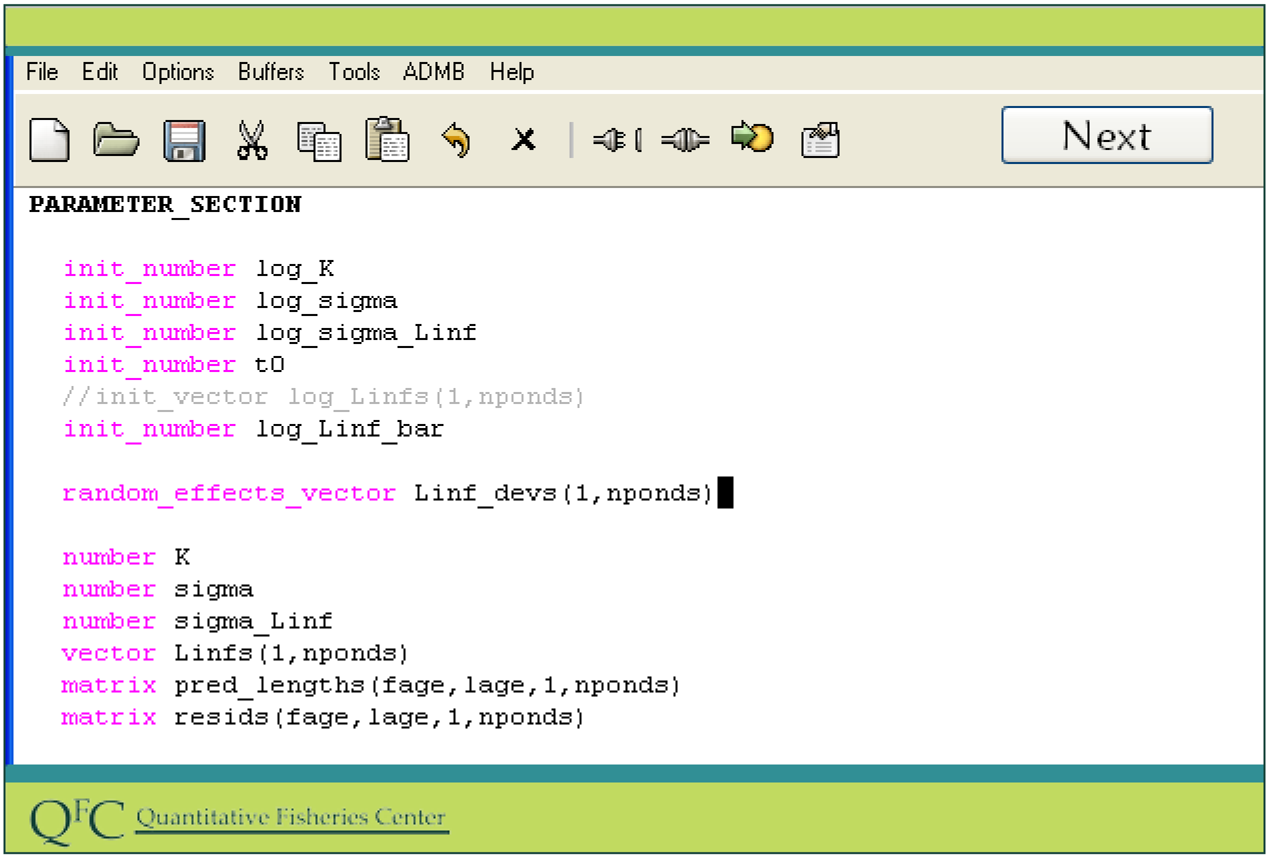
```
// init_vector  log_Linfs(1,nponds)
```



We will also need a back transformed version of the L-infinity standard deviation which we add to the parameter section, calling it sigma_Linf.

Slide Code:

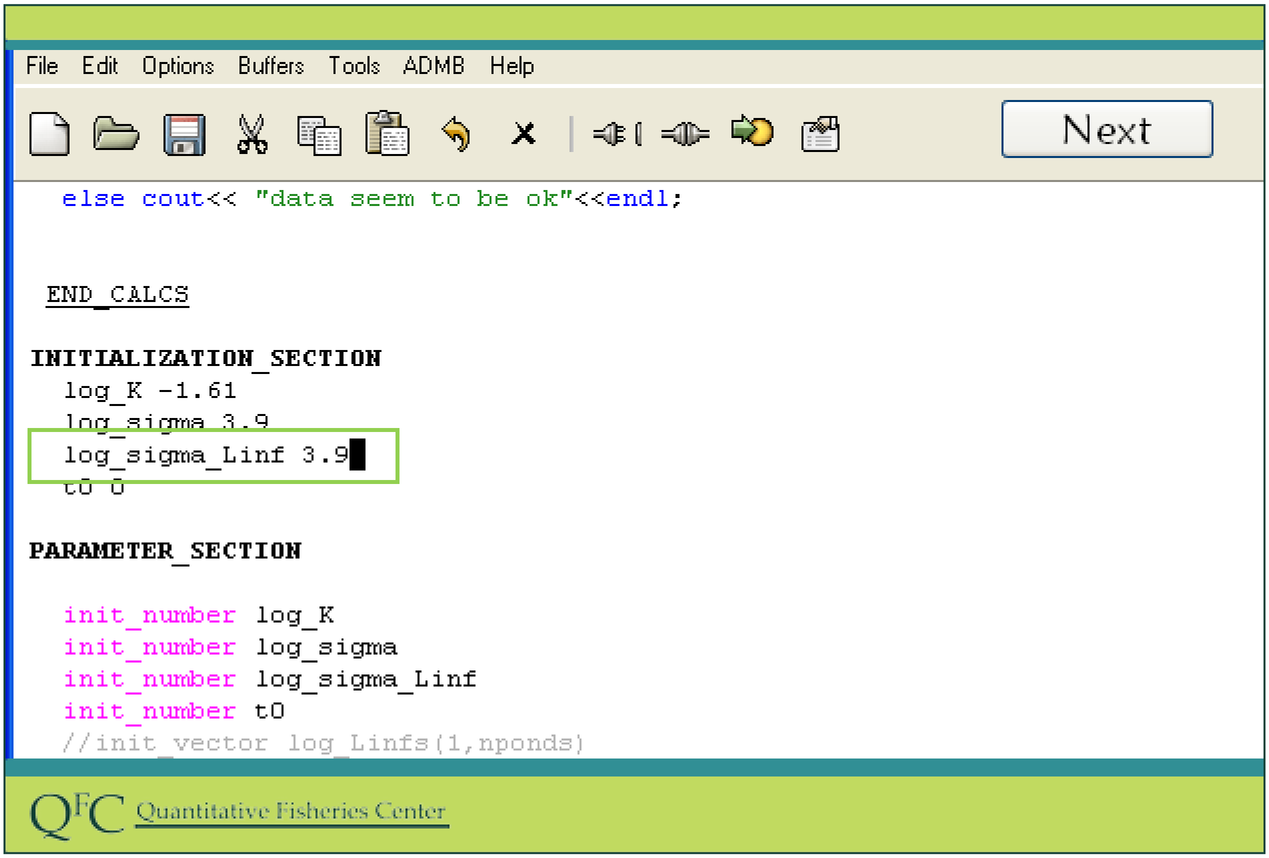
```
number  sigma_Linf
```



Finally, we add a line to the parameter section defining the random effects. This follows the basic syntax for a vector, but by using the key word random_effects_vector we tell the random effects version of ad-model builder that this contains random effects.

Slide Code:

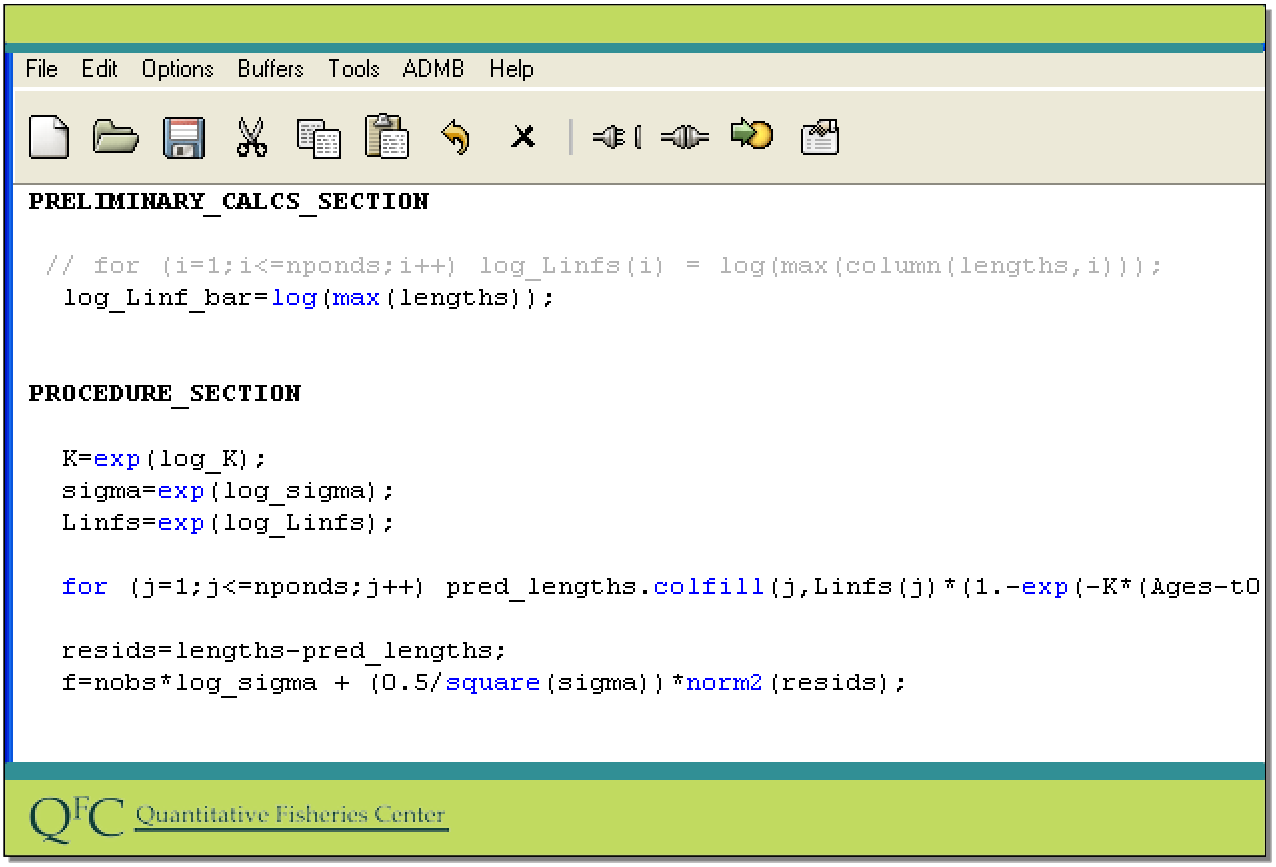
random_effects_vector Linf_devs(1,nponds)



Now we go back to the initialization section. We add in a starting value for log_sigma_Linf. Somewhat arbitrarily we simply use the same starting value for this as we had previously used for the standard deviation of observed lengths. If we run into trouble during estimation we might consider being more sophisticated about this, for example by setting the standard deviation based on how variable lengths of older fish are among ponds.

Slide Code:

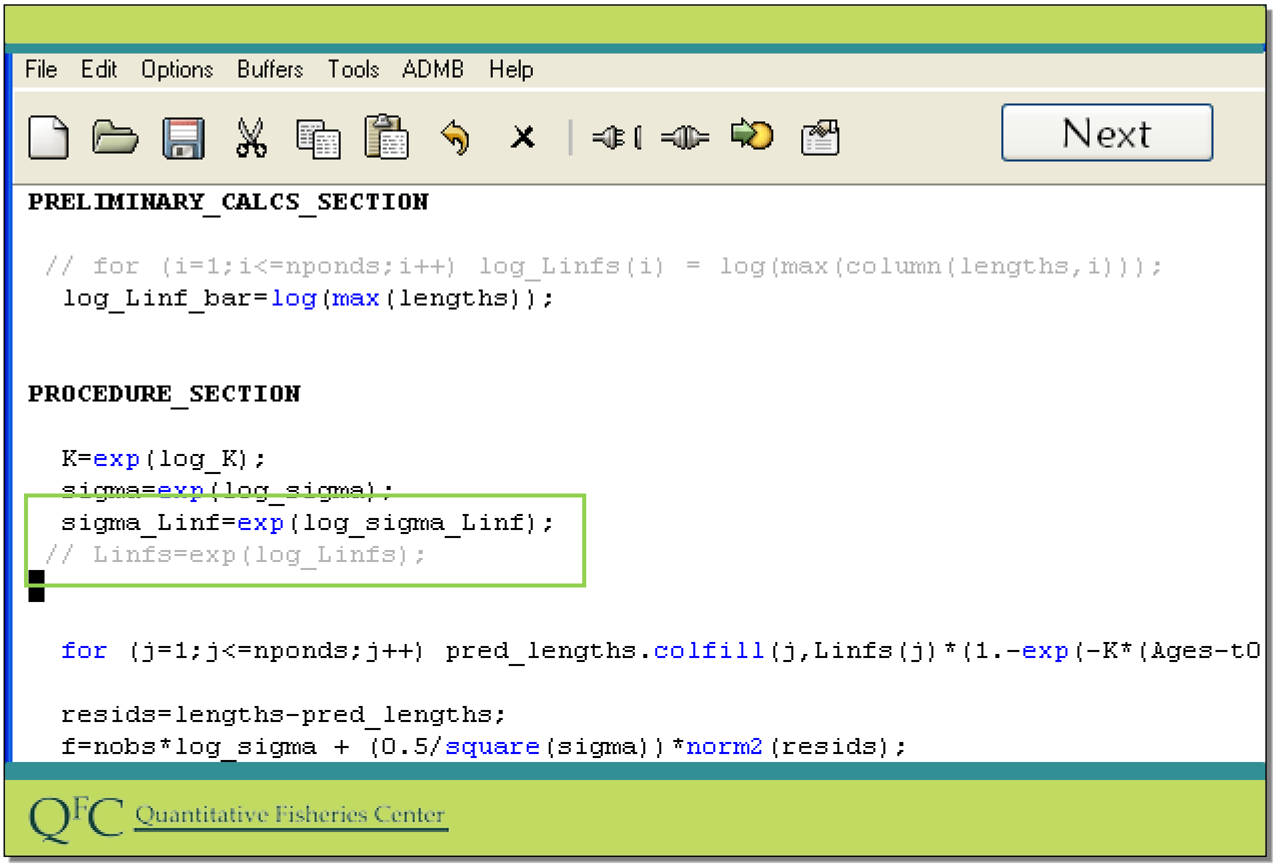
log_sigma_Linf 3.9



We now go to the preliminary calcs section and comment out the existing line that set initial values for Linfinity for each pond. We replace this with a line setting the initial value for log_Linf_bar to the log of the maximum observed length in the data.

Slide Code:

```
//  for(i=1;i<=nponds;i++)  log_Linfs(i) = log(max(column(lengths,i)));
  log_Linf_bar = log(max(lengths));
```

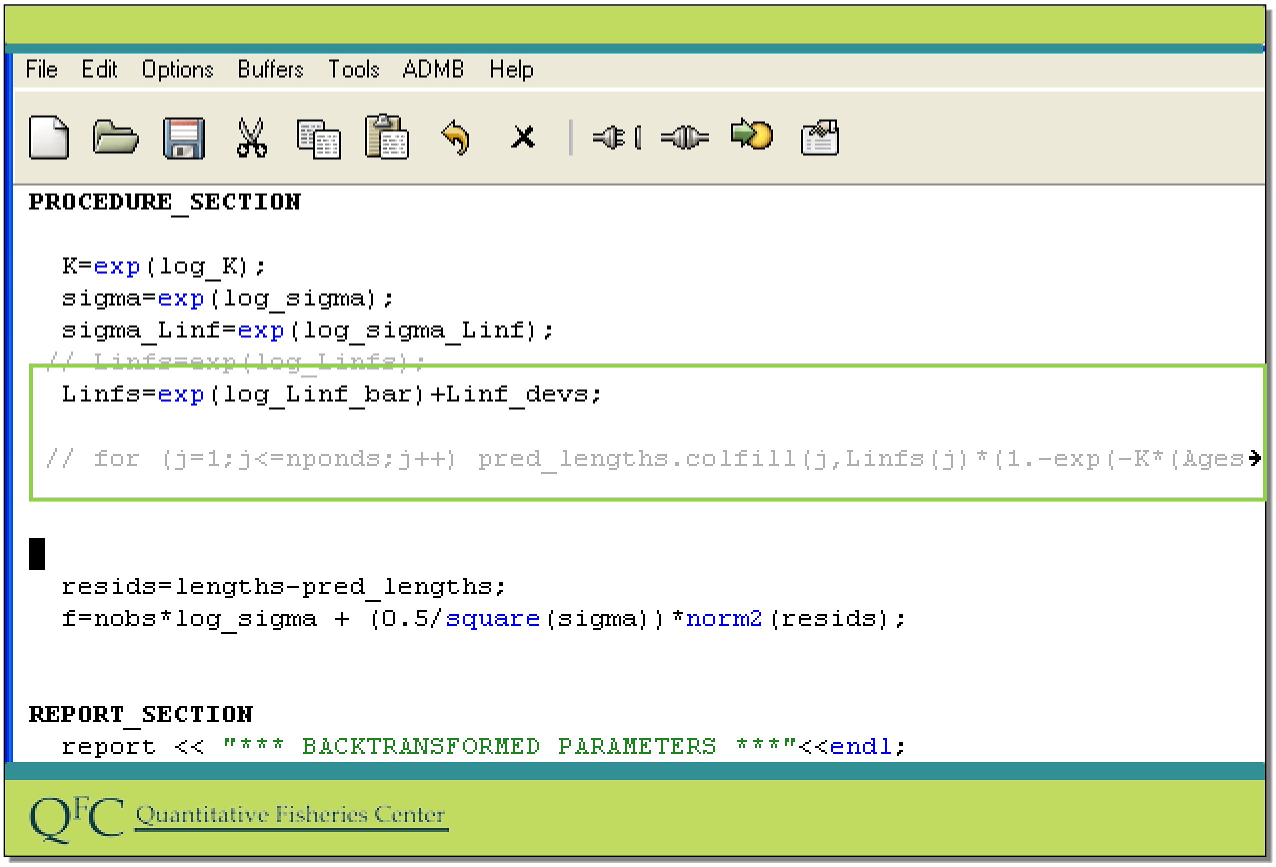


At this point we want to test the changes we have made so far. However, to test the changes we need to do a bit more to avoid some errors.

First, we back-transform sigma_Linf, then we comment out the previous back-transformation of Linfs

Slide Code:

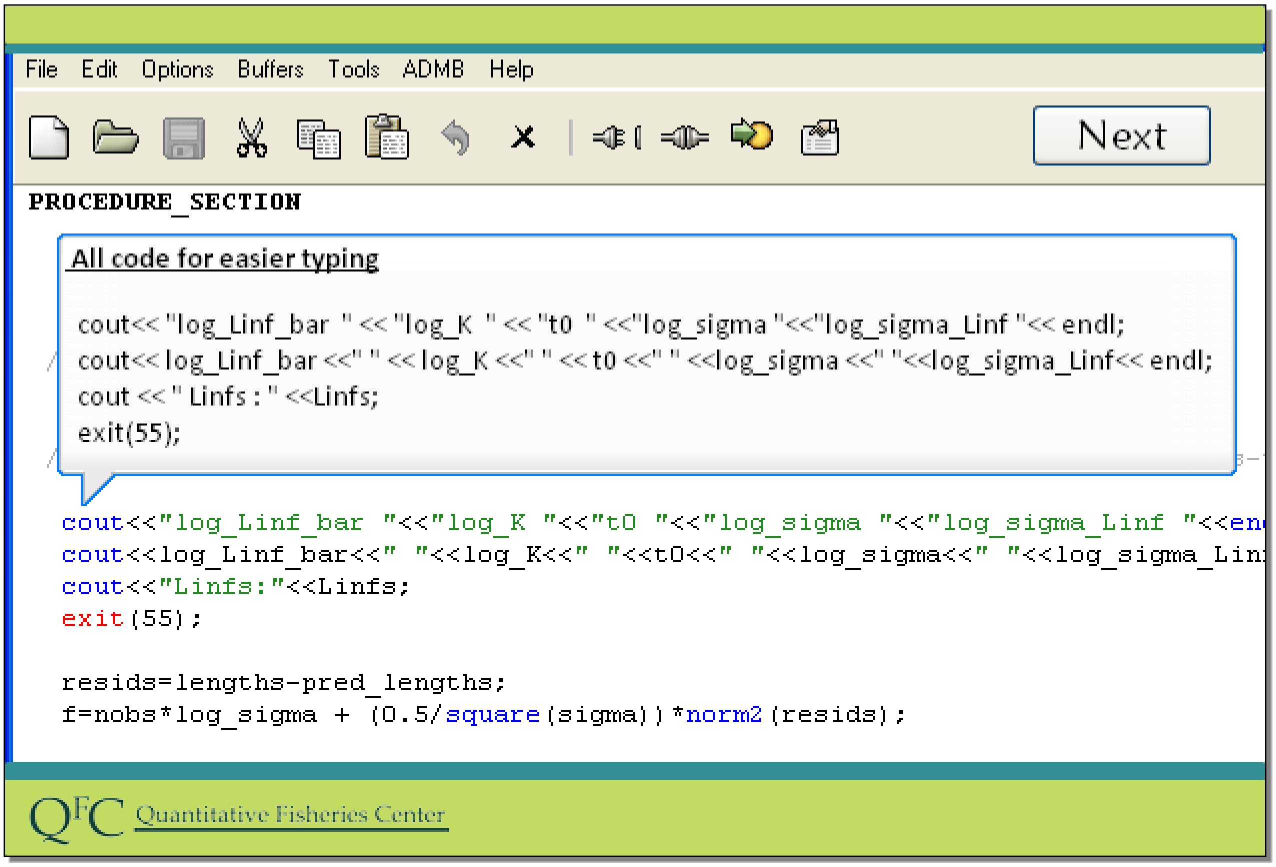
```
sigma=exp(log_sigma);
// Linfs=exp(log_Linfs);
```



and replace this with a calculation based on log_Linf_bar and the random effect vector Linf_devs. We also comment out the line where predicted lengths are calculated. We had to comment out that line because the random effects version of AD Model builder does not understand the colfill command. This highlights that when we choose to run the random effects version of ad model builder we are essentially using a distinct package. While most common functions we use are present in the random effects version not all functions are. After we do our testing we will replace this line with alternative code.

Slide Code:

```
Linfs=exp(log_Linf_bar) +Linf_devs;
// for (j=1; j<=nponds;j++) …………
```

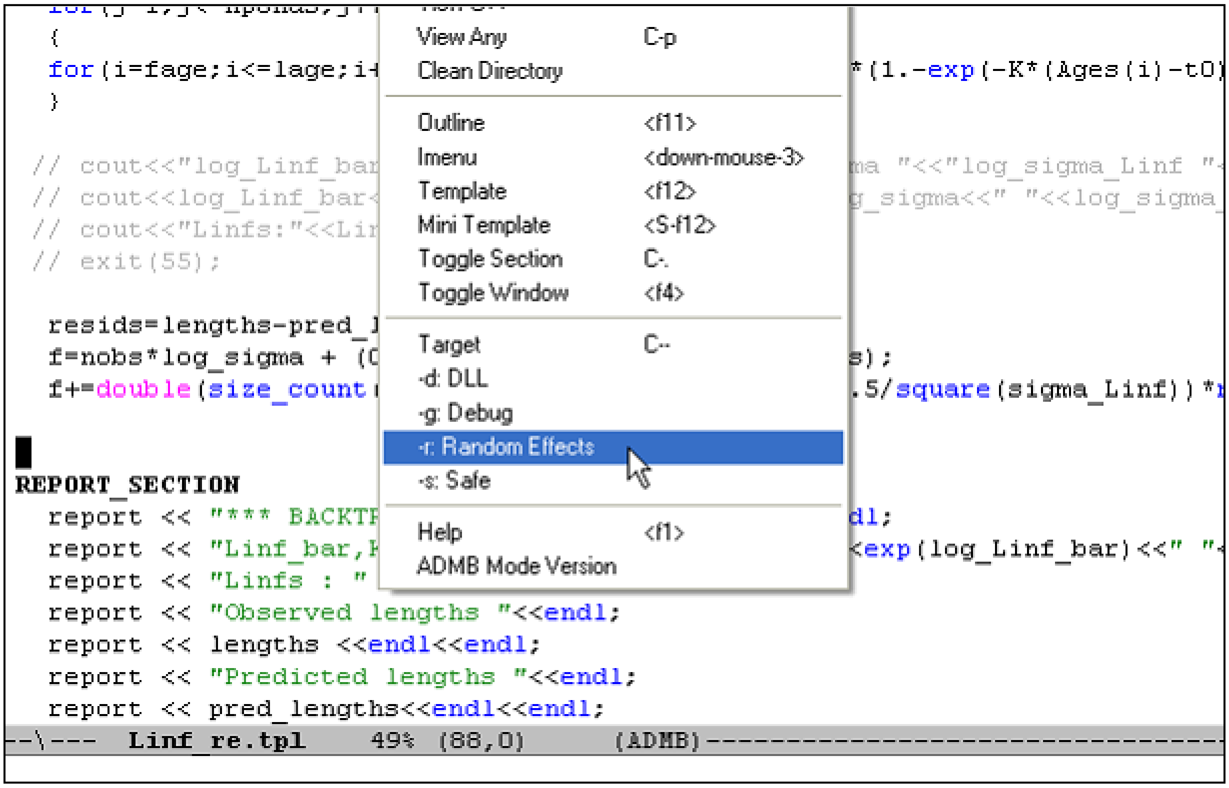


We now add some cout commands and an exit command to make sure that things seem to be running ok to this point.

Type these lines and click next to continue.

Slide Code:

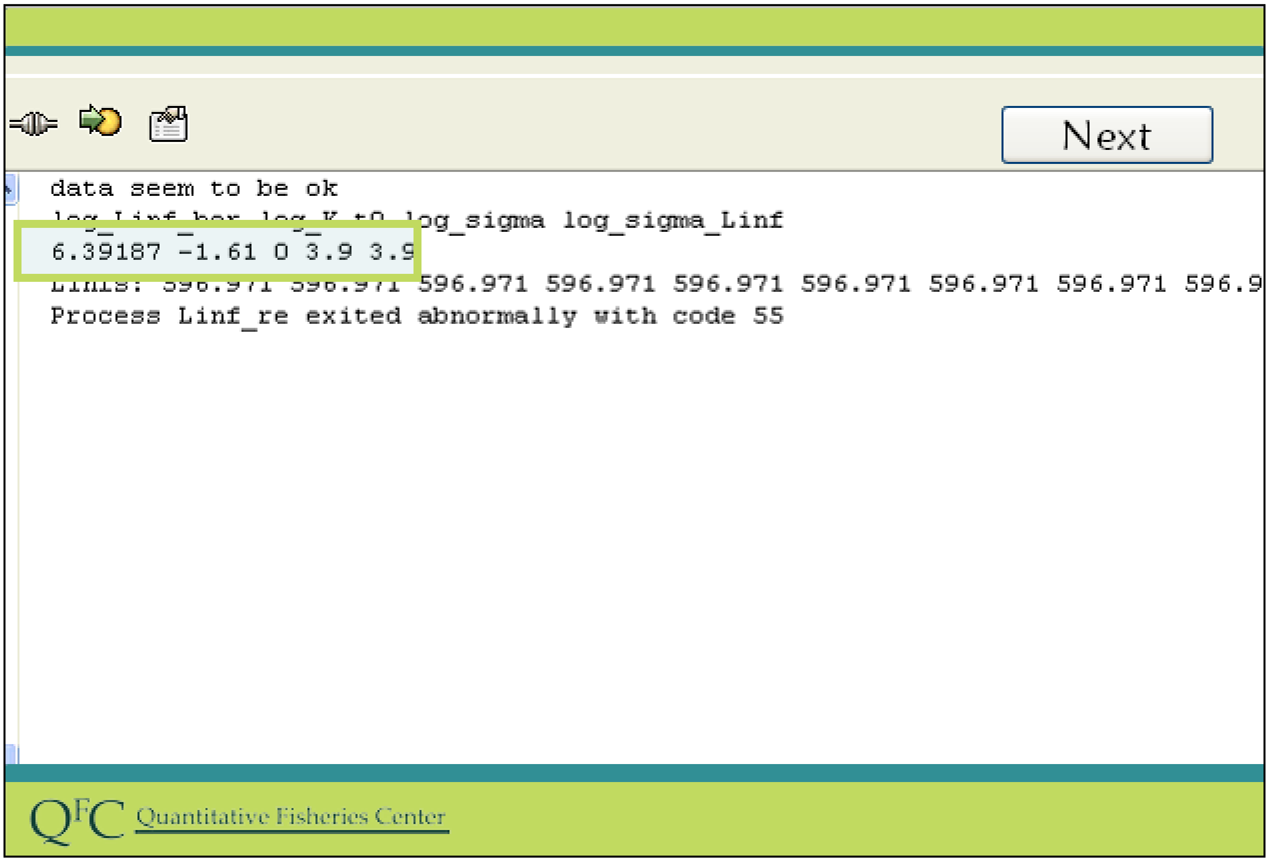
```
cout<< "log_Linf_bar  " << "log_K  " << "t0  " <<"log_sigma "<<"log_sigma_Linf "<< endl;
cout<< log_Linf_bar <<" " << log_K <<" " << t0 <<" " <<log_sigma <<" "<<log_sigma_Linf<< endl;
cout << " Linfs : " <<Linfs;
exit(55);
```



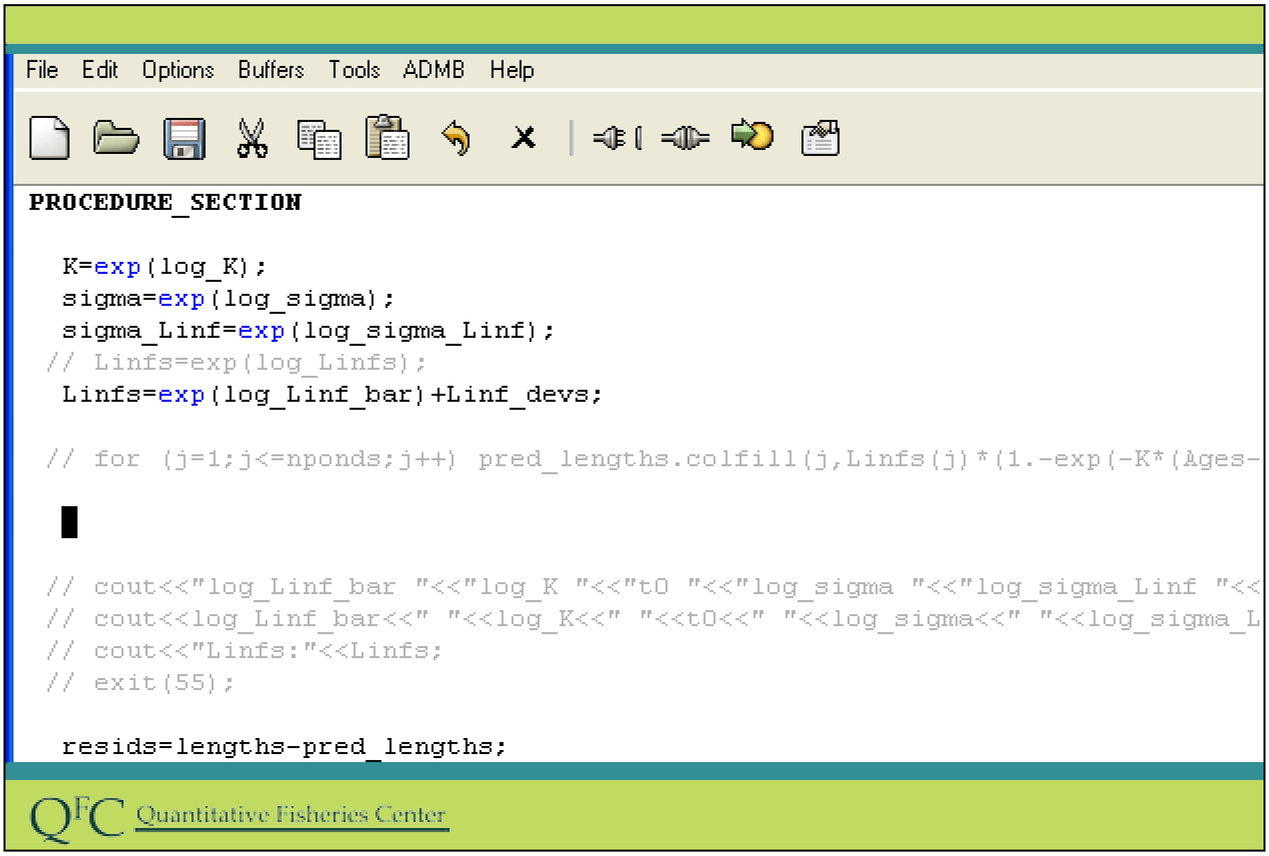
We need to run our program with random effects. Go to ADMB, then down to the target section and select dash r random effects (-r: random effects)

Slide Action:

1. **Go to the ADMB menu**
2. **Go down to –r: random effects**



Now we build and run the program. The screen output appears to have the right starting values for the various parameters. Initially all the L-infinities are the same because by default the random effects vector was started as all zeros.

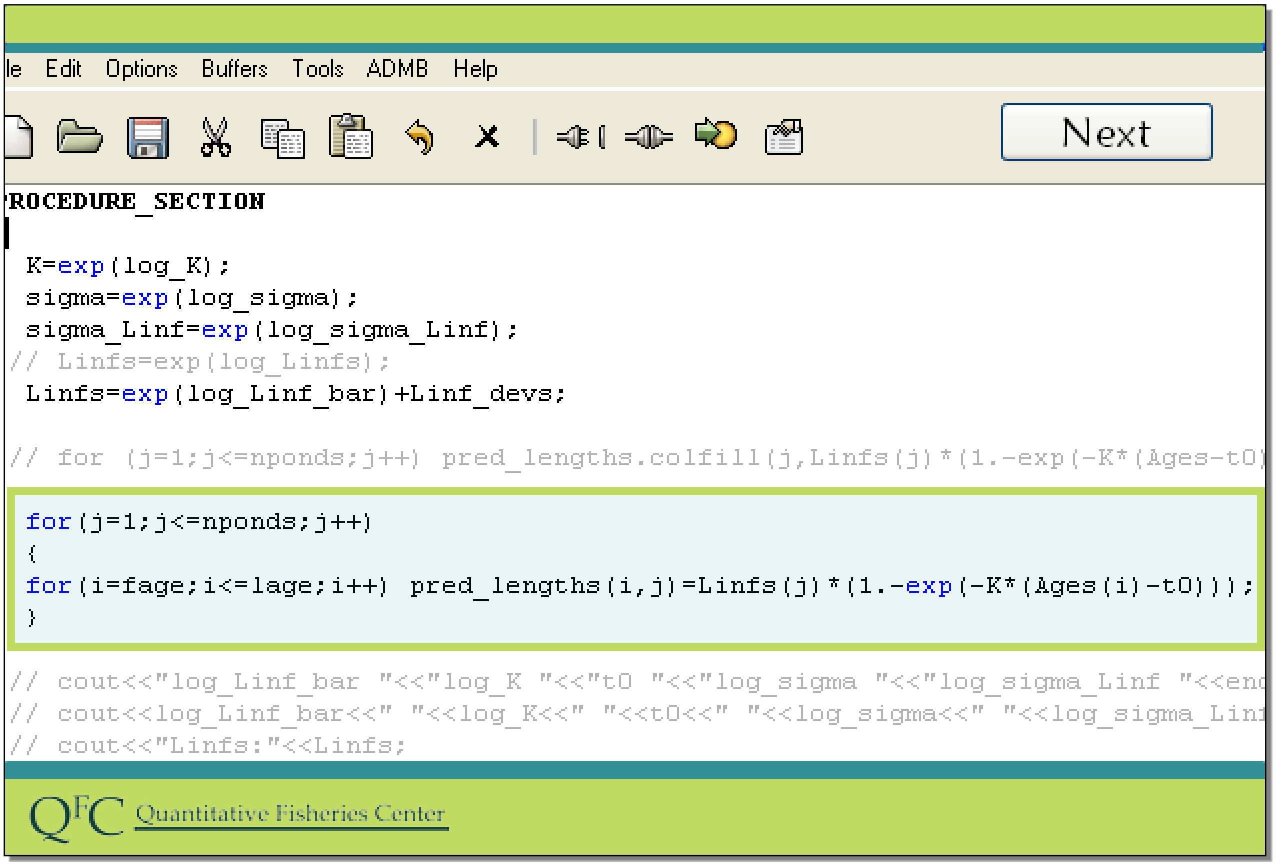


We now will finish writing the code. First we comment out our cout and exit commands before we forget.

Slide Code:

```
// cout<< "log_Linf_bar  " << "log_K  " << "t0  " <<"log_sigma "<<"log_sigma_Linf "<< endl;
// cout<< log_Linf_bar <<" " << log_K <<" " << t0 <<" " <<log_sigma <<" "<<log_sigma_Linf<< endl;
// cout << " Linfs : " <<Linfs;
// exit(55);
```

Slide 47 - Slide 47

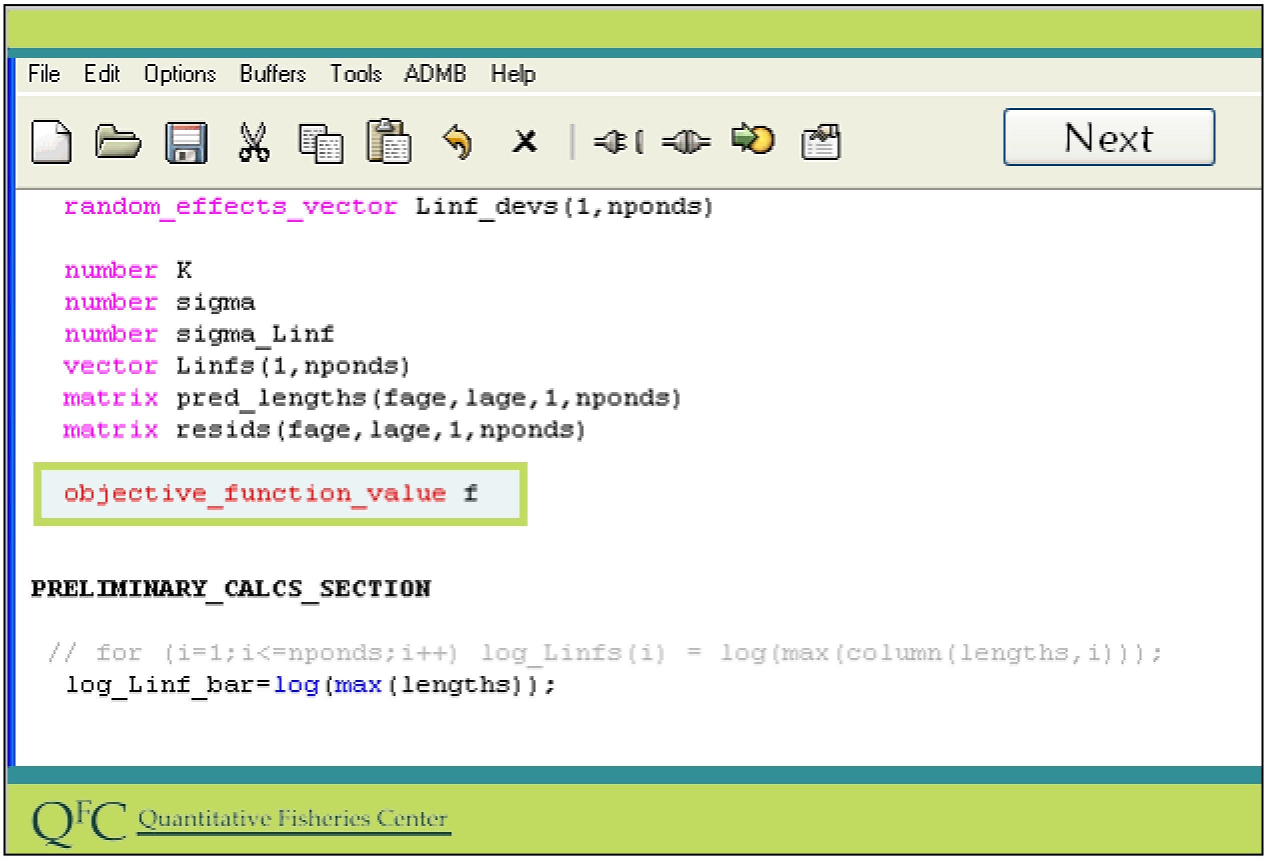


Next we replace the line containing the colfill command with a double loop filling in predicted values for each pond and age:

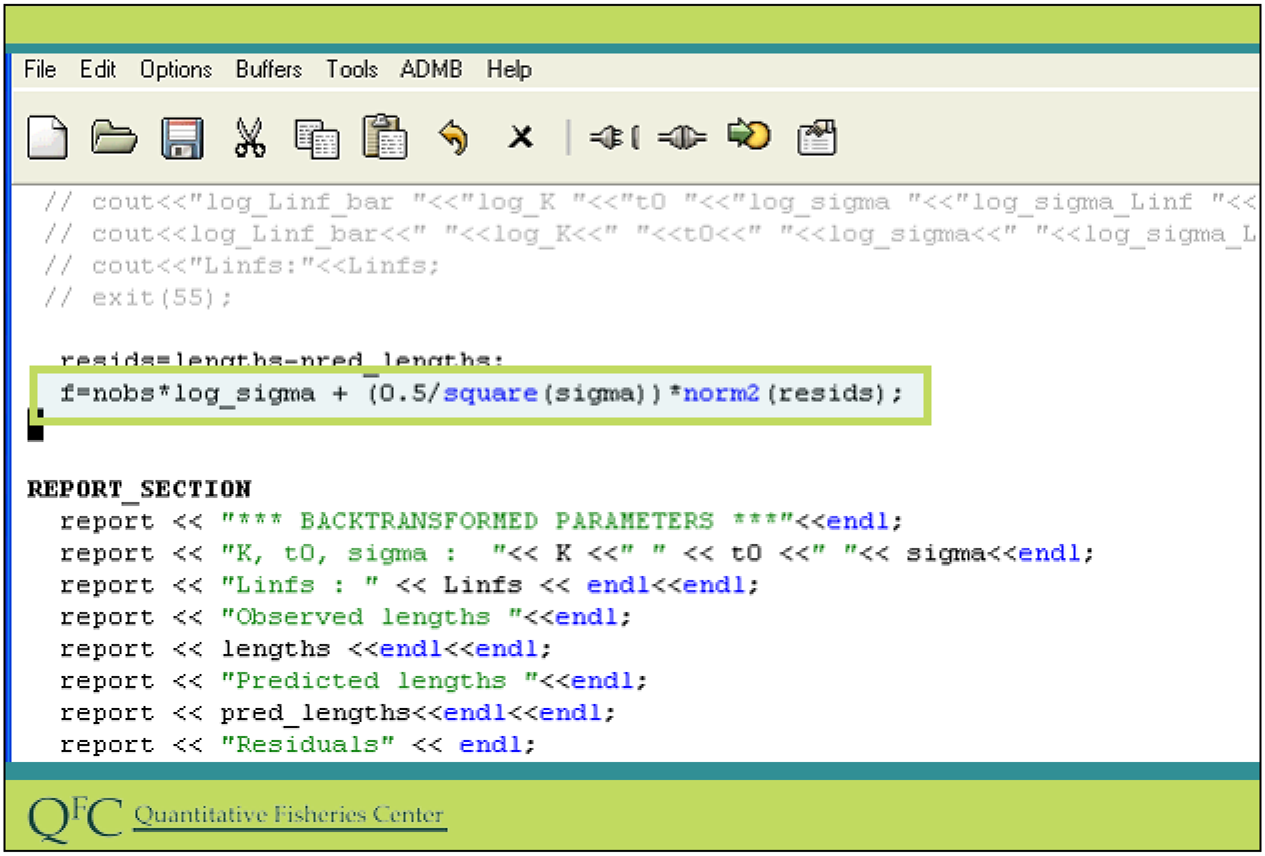
Type these lines then click next to continue.

Slide Code:

```
for(j=1;j<=nponds;j++)
{
for (i=fage; i<=lage;i++)  pred_lengths (i, j) =Linfs(j) * (1.-exp(-K*(Ages(i)-t0)));
}
```



The last essential thing we need to do is modify the objective function, represented by the variable f in our code. Up to now we have not done anything to tell our program what distribution our random effects will have. Most often when modeling random effects, normal distributions are used and that is what we are assuming here. But AD model builder is completely flexible about this so we have to specify the distribution. AD Model builder expects the objective function to consist of the negative log-likelihood of the data, and the negative log of the probability density for the random effects.

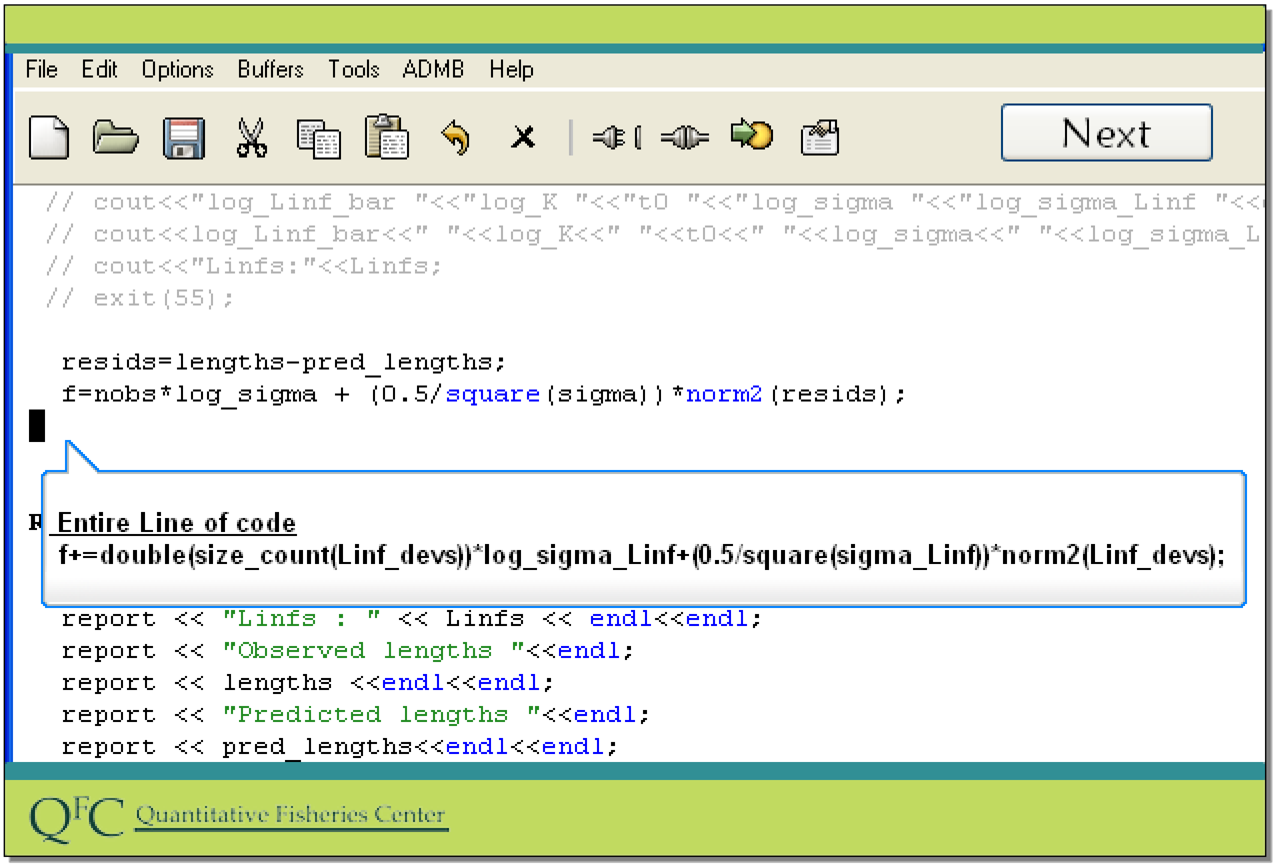



```
// cout<<"log_Linf_bar "<<"log_K "<<"t0 "<<"log_sigma "<<"log_sigma_Linf "<<
// cout<<log_Linf_bar<<" "<<log_K<<" "<<t0<<" "<<log_sigma<<" "<<log_sigma_L
// cout<<"Linfs:"<<Linfs;
// exit(55);

  resids=lengths-pred_lengths;
  f=nobs*log_sigma + (0.5/square(sigma))*norm2(resids);

REPORT_SECTION
  report << "*** BACKTRANSFORMED PARAMETERS ***"<<endl;
  report << "K, t0, sigma :  "<< K <<" " << t0 <<" "<< sigma<<endl;
  report << "Linfs : " << Linfs << endl<<endl;
  report << "Observed lengths "<<endl;
  report << lengths <<endl<<endl;
  report << "Predicted lengths "<<endl;
  report << pred_lengths<<endl<<endl;
  report << "Residuals" << endl;
```

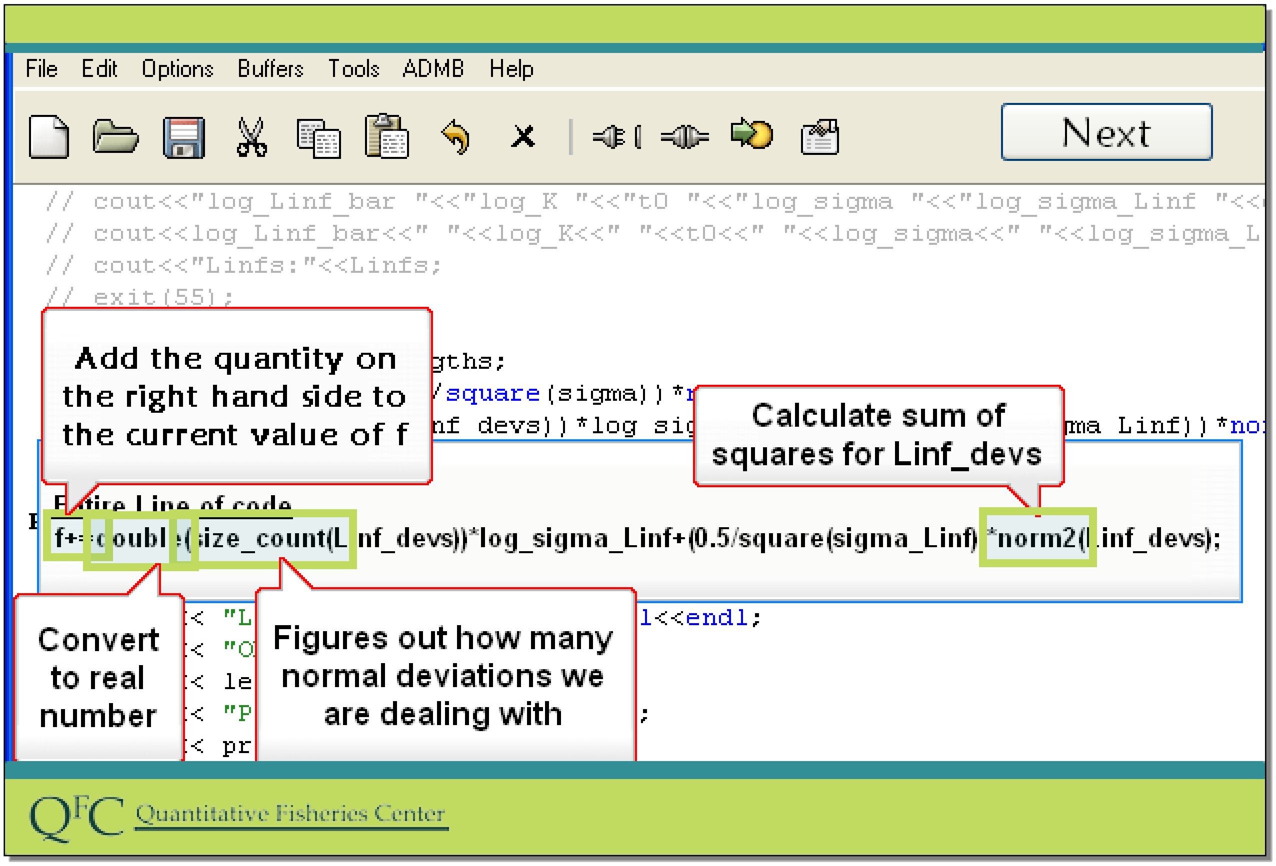
The objective function is being calculated at the end of the procedure section. The line of code that is already there includes only the negative log-likelihood of the data, which depends on the values of the parameters and the random effects. We need to add to the negative log of the probability density for the random effects.



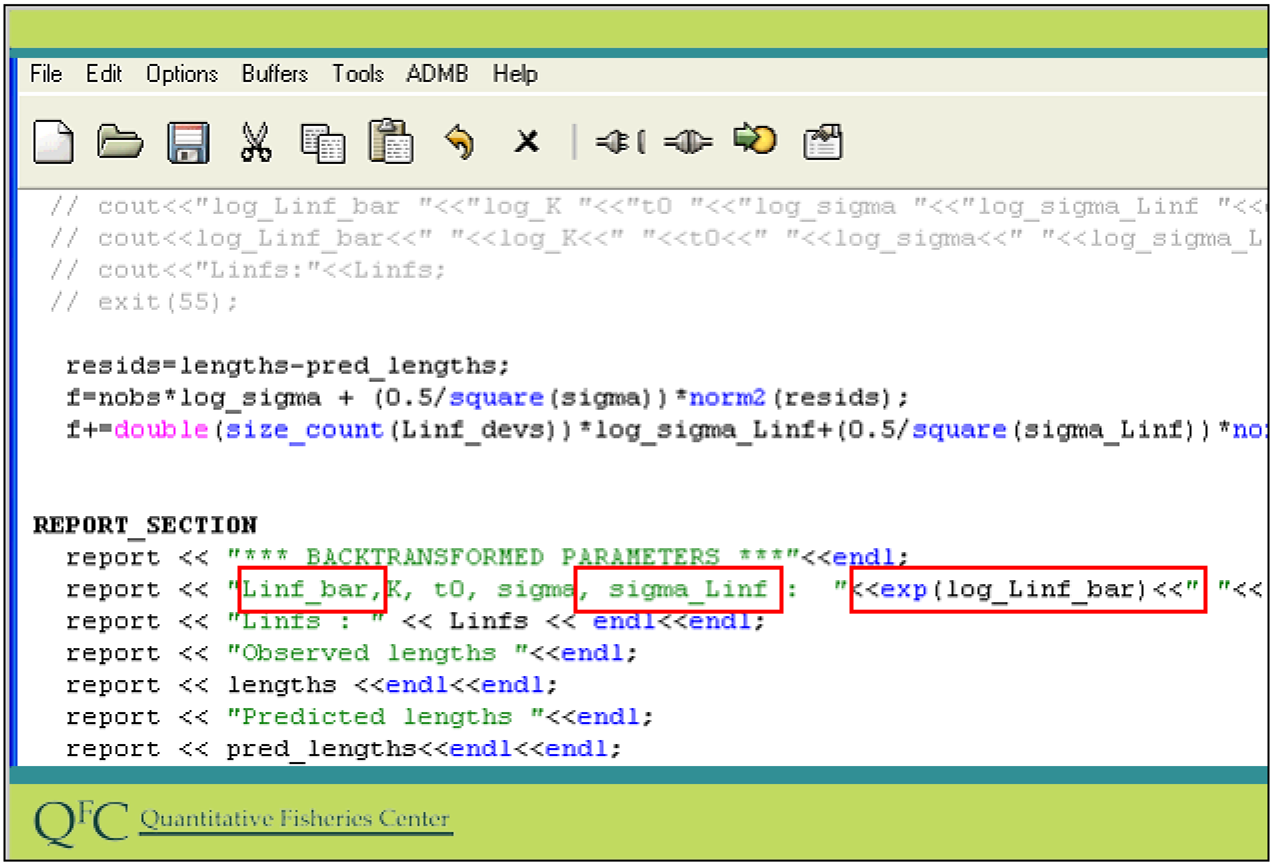
Add this line of code. Click next to continue

Slide Code:

f+=double(size_count(Linf_devs))*log_sigma_Linf+(0.5/square(sigma_Linf))*norm2(Linf_devs);



Notice this has basically the same structure as the first component, not surprisingly since we are again assuming a normal distribution. Note that plus equals means add the quantity on the right hand side to the current value of f. Size count figures out how many normal deviations we are dealing with. Double converts this to real number and this plays same role as nobs did in the first component. Log_sigma_Linf and sigma_Linf replaces log_sigma and sigma. And we use norm2 to calculate the sum of squares for Linf_devs instead of for the residuals.



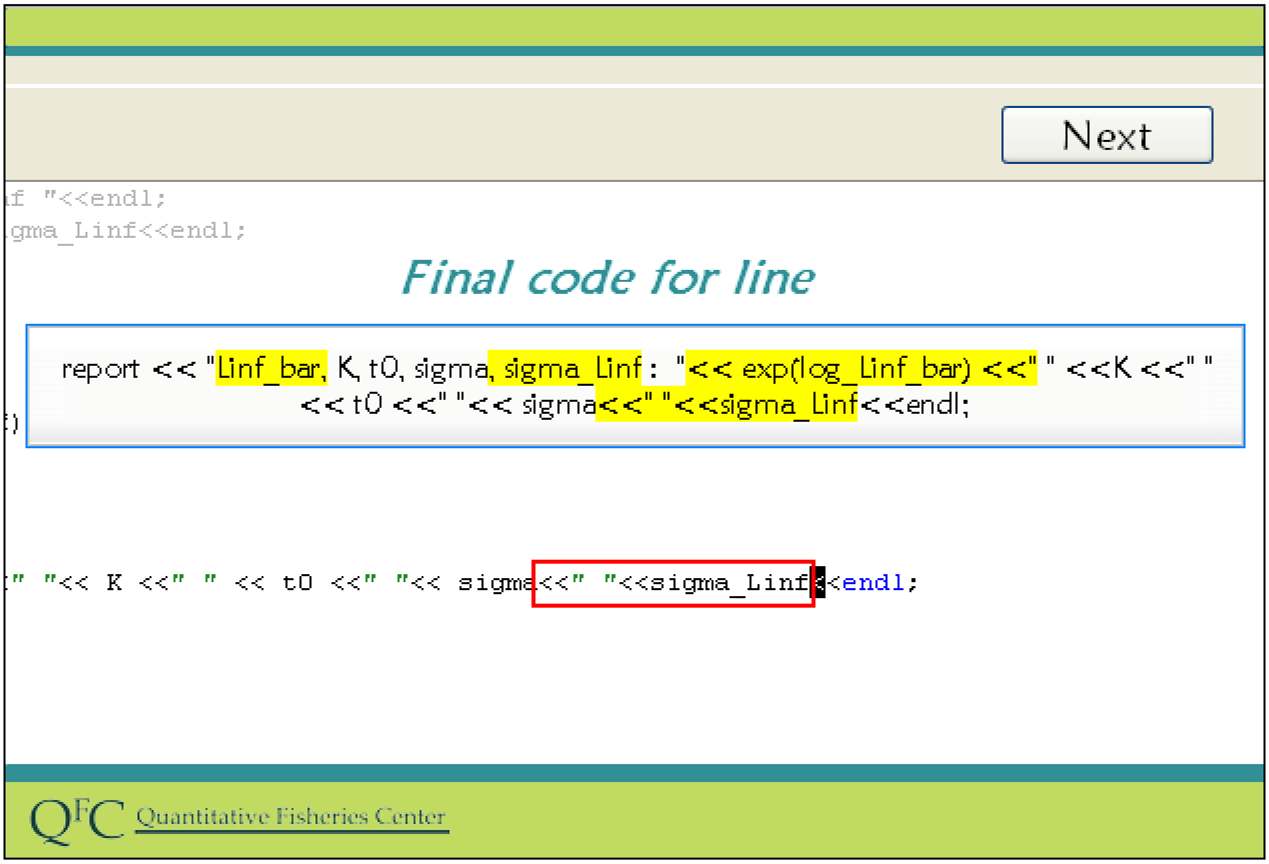
Finally, just for completeness, we modify the report section so it outputs back-transformed values of L-infinity-bar and sigma_Linfinity, which are new parameters for this version of the model.

Slide Code:

Linf_bar,

, sigma_Linf

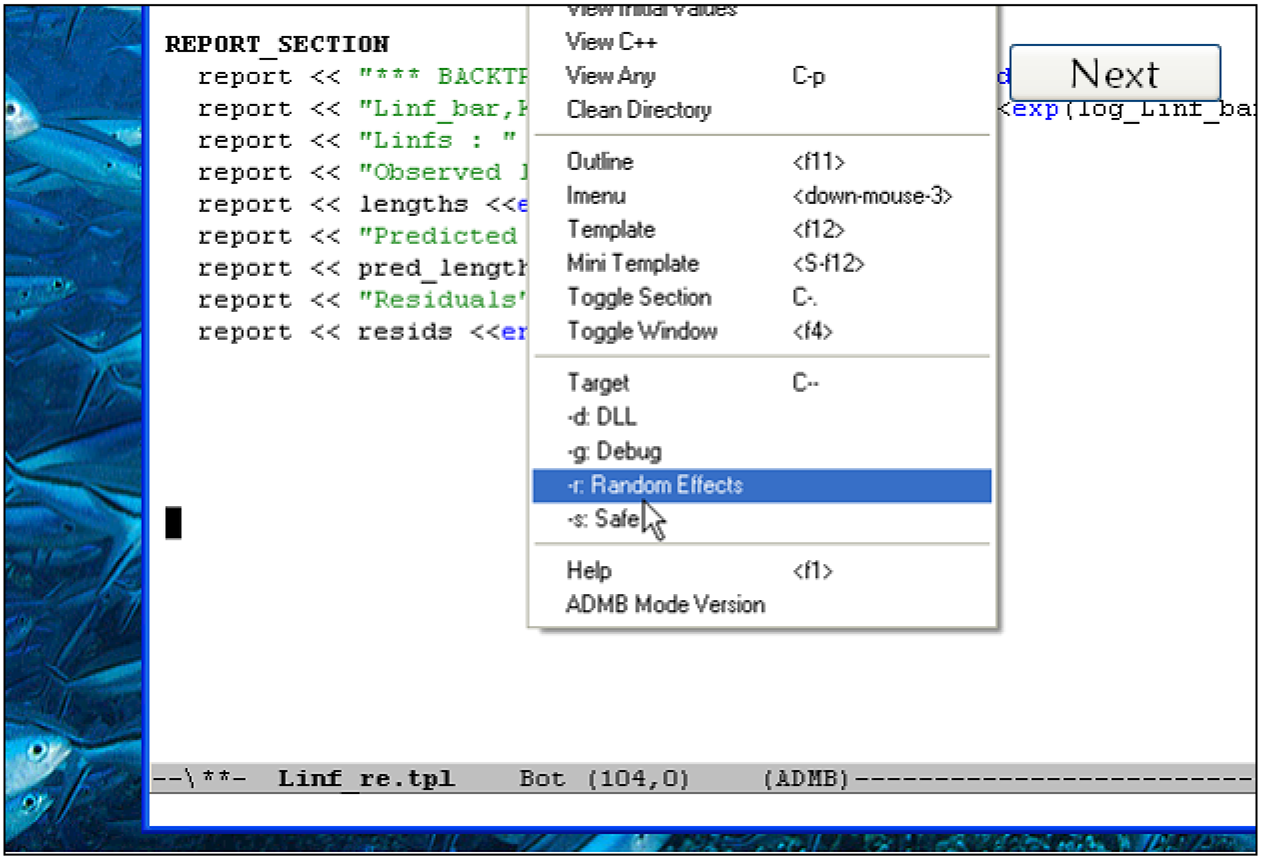
<<exp(log_Linf_bar)<<"



Type the changes to the code. The additional information is highlighted. Click next to continue.

Slide Code:

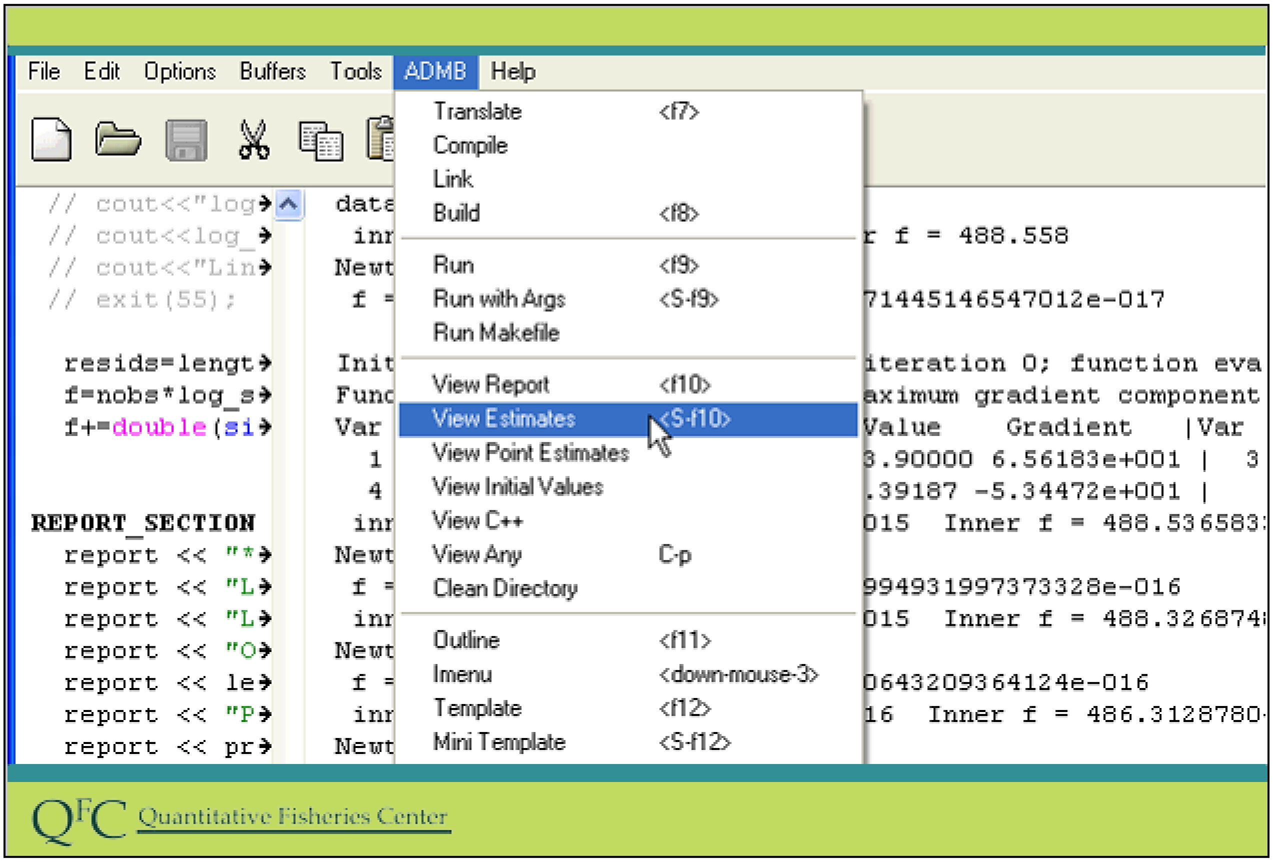
<<" "<<sigma_Linf



Go to ADMB and check to see if there is a check mark next to -r: random effects. If not make sure you select it.

Slide Action:

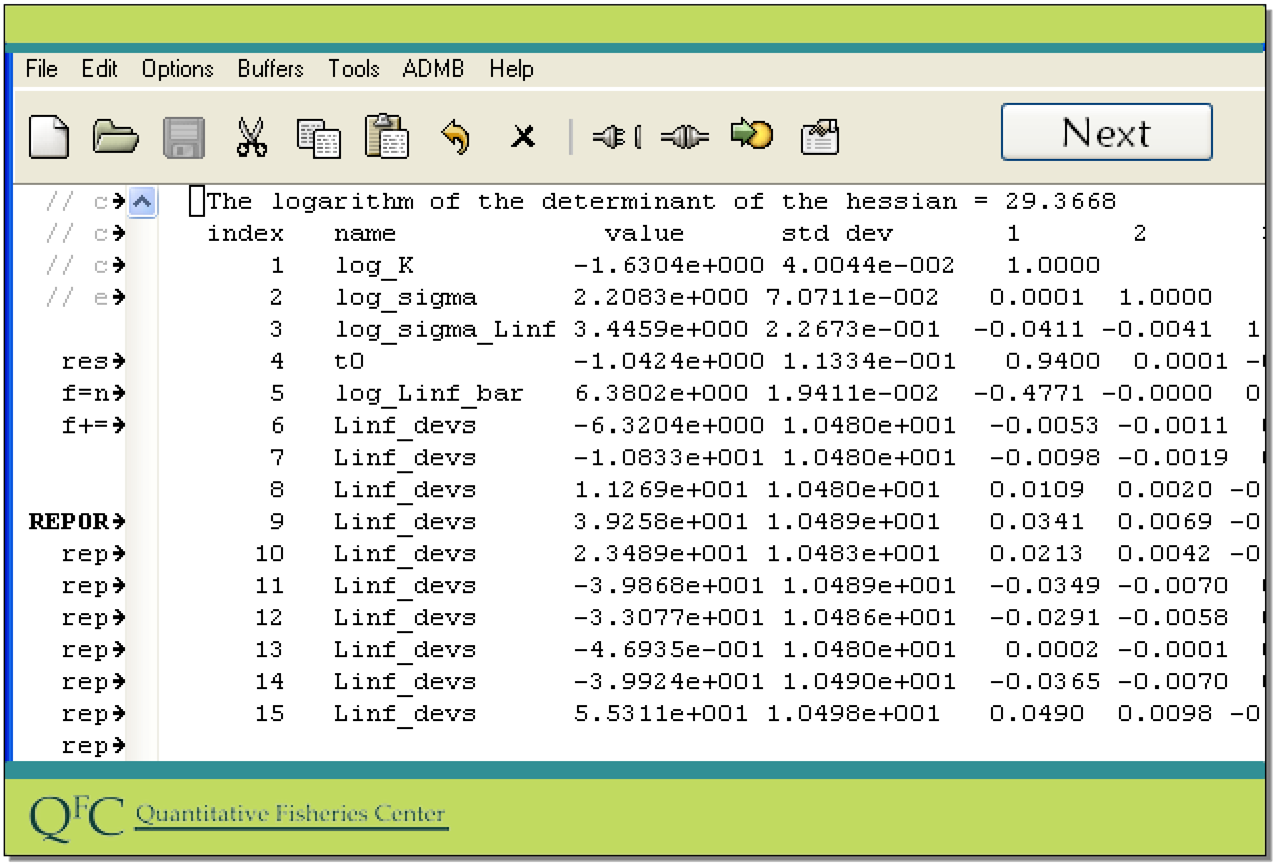
1. **Double check that there is a check next to –r: random effects in ADMB menu. If not check it**
2. **Build**
3. **Run program**



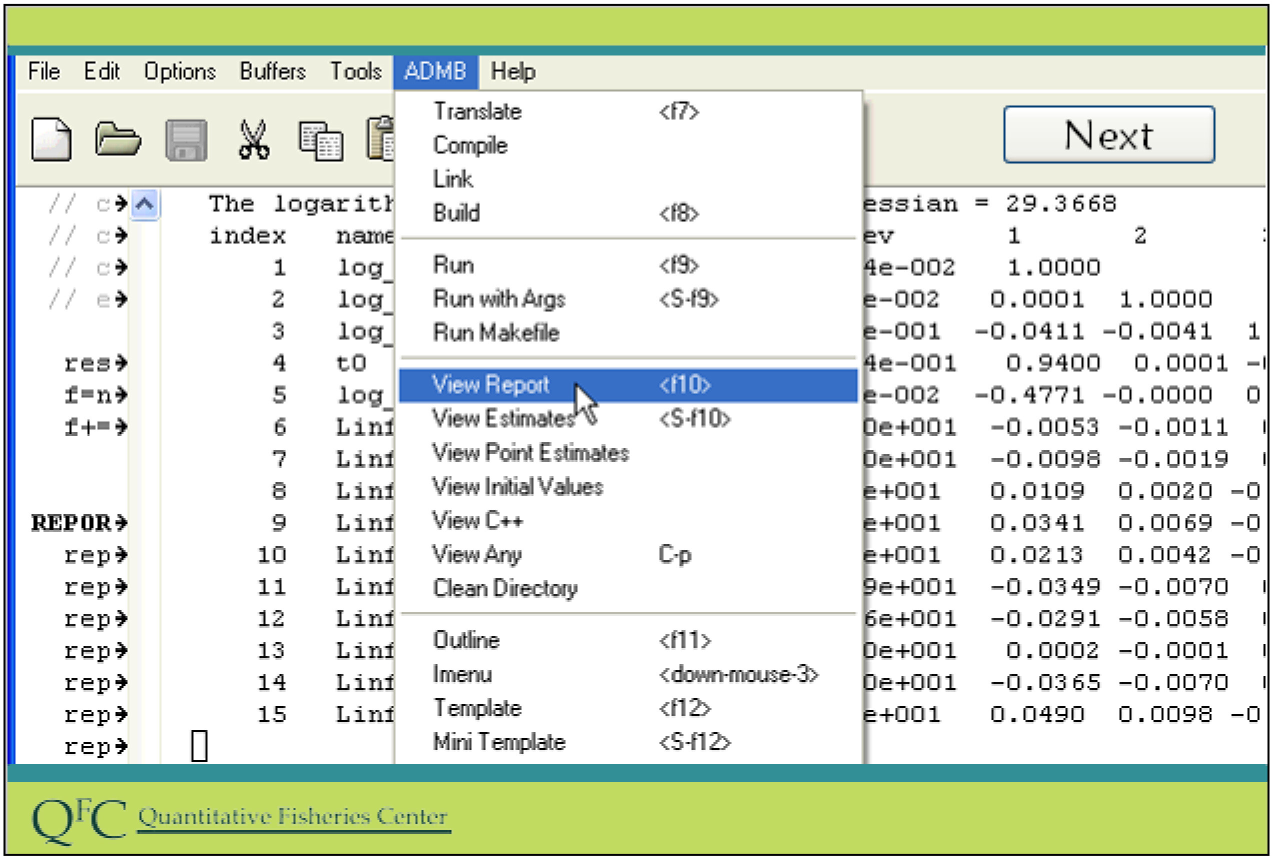
We now can view the cor file by going to ADMB then view estimates.

Slide Action:

Go to ADMB menu and select View Estimates



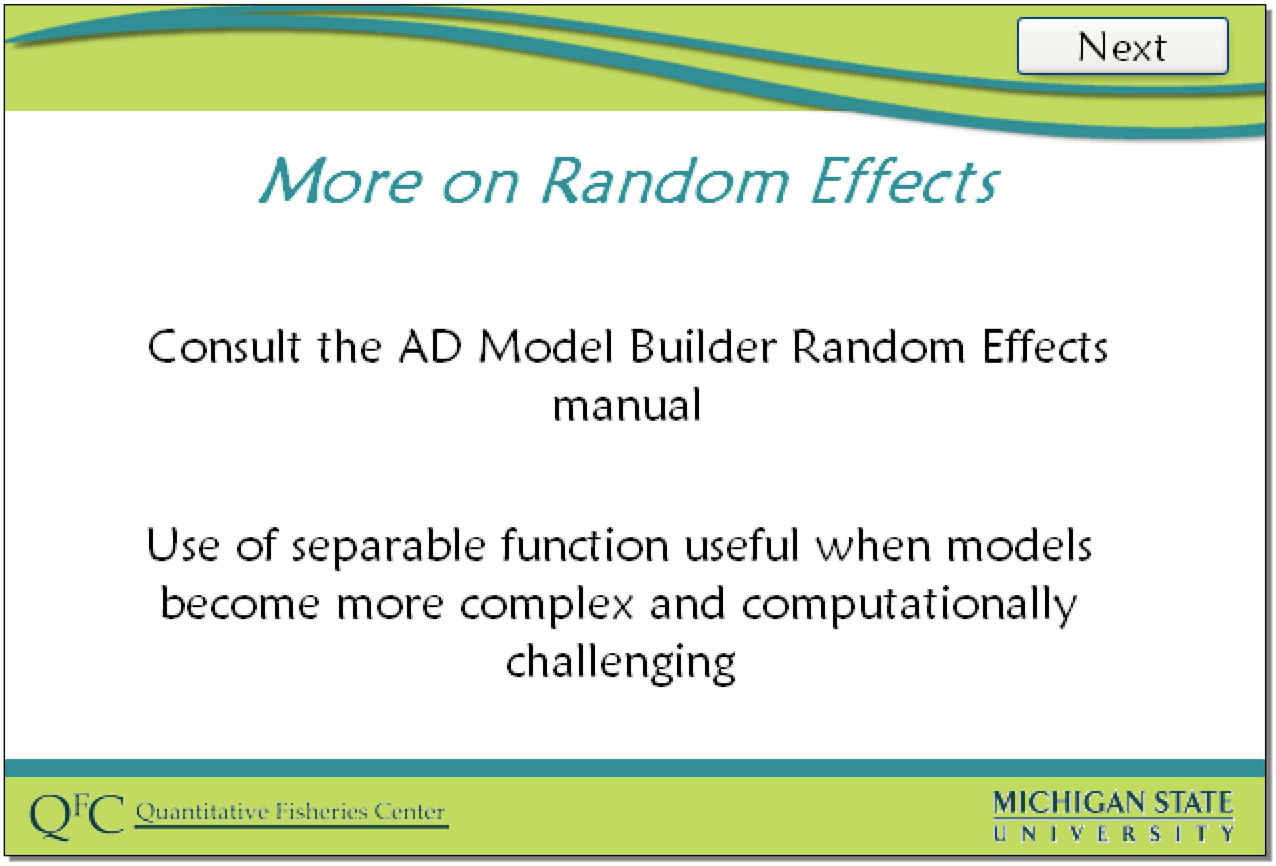
We now have estimates, asymptotic standard errors, and a correlation matrix for all the parameters and random effects. It is worth knowing that the parameter estimates are obtained by maximum likelihood, with the random effects integrated out. The integral is obtained by the Laplace (lap-less) approximation. With the maximum likelihood estimates fixed, the random effect estimates are those that maximize the probability density for the random effects. When estimation was occurring you might have noticed that there was an inner and outer loop step in estimation. During the inner loop the random effects were being adjusted with the parameters fixed at their current estimates. In the outer loop the parameters were being adjusted. The details of the numerical methods used are beyond the scope of this introductory video. The AD Model builder provides more detail on the estimation approach.



We now also take a quick look at the report file and see that the back transformed values and other reported quantities make sense.

Slide Action:

Go to ADMB menu and select View Report



We have barely scratched the surface of using admb for models including random effects. Once you have worked with this simple example and understand it we urge you to read the AD Model builder random effects manual, particularly the material on separable functions. The way we have coded the problem we have done nothing to tell the software which random effects influence which observation. The AD model builder approach allows you to specify this using separable functions, which can allow the software to solve much more difficult problems or to come up with solutions more quickly.

You have now completed this basic introduction to the random effects version of AD model builder.