hw2_tara

tara

10/25/2023

Due: 10/27/23, 17:00 pm (Eastern Time)

Please provide Gavin with a brief report containing your solutions to the tasks below, and also include your R scripts (or markdown), .cpp file, and any additional files needed to run your assignment. Remember that in addition to performing the technical analyses, communicating those analyses through interpretation and discussion is an important component.

###Review of state-space modeling###
1. Use the LLM of your choice (e.g. Chat-GPT4) to produce a brief (1-2 paragraph) overview of state-space models, their estimation and application to population dynamics model, and the advantages and disadvantages of using state-space models for these applications. Based on your reading and knowledge from the course material, provide a critical review of the LLM-generated response, including correcting any necessary statements, and validate any literature that the model provided. In your report, include what model you used to generate the response, the prompt you provided, as well as the generated answer and your review.

ChatGPT4 response to "What are state-space models?"
"State-space models (SSMs) are a class of statistical models used to describe the evolution of a system over time. They are particularly useful in situations where there is uncertainty about the underlying state of a system and where there may be noisy or imperfect observations. A state-space model consists of two main components: 1. State Equation: This equation describes how the underlying (hidden) state of the system evolves over time. It's often represented as a linear or nonlinear dynamic process. The state at time `t` is denoted by `x(t)`. For example, in a linear Gaussian state-space model, the state equation might be: `x(t+1) = A * x(t) + B * u(t) + w(t)`. Where: A is a transition matrix representing how the state evolves. B represents the effect of an input u(t) on the state, and w(t) is a process noise term. 2. Observation equation: This equation relates the observed data y(t) to the underlying state x(t). It accounts for measurement noise and any other sources of error. For example, in a linear Gaussian state-space model, the observation equation might be: `y(t) = C * x(t) + v(t)`. Where, C is an observation matrix representing the relationship between the state and the observations and v(t) is an observation noise term. In addition to these two main components, state-space models often include assumptions about the initial state x(0) and the distributions of the noise terms (w(t) and v(t)). State-space models are very versatile and can be applied in various fields such as economics (e.g., in modeling economic trends), engineering (e.g., in control systems), ecology (e.g., in population dynamics), and more. They are also used extensively in time series analysis and machine learning. The strength of state-space models lies in their ability to handle dynamic and noisy systems, making them a valuable tool for modeling real-world phenomena with inherent uncertainty."

1. Critique of ChatGPT4 response
ChatGPT4 provides a good basic overview of state-space models. It begins with a concise definition of and proceeds to explain the two main components, the process model (though ChatGPT referred to this as the 'state' model), and the observation model (ChatGPT referred to both models as 'equations'). The examples given for the state equation are somewhat overspecific. The examples illustrate a case where the process model is time-dependent, but the state variables could evolve in relation to space (i.e. geography or euclidean, etc.), or some other process such as evolution or any independent variable. The example used to describe the process model is overly specific as well. The process model needs to include at least one state-dependent variable, where the outcome at the next stage is conditioned on the value of the state variable in

the present stage. The example provided by ChatGPT makes it seem as though several terms need to be state-dependent. `A` does not have to be a transition matrix, but could be a constant,though technically a matrix with one value is still a matrix. The observation model provided by ChatGPT includes observation error ("noise") and correctly relates the state variable to observation at the present stage. This response did not mention structural uncertainty. ChatGPT's response references that state-space models often include assumptions about the distribution of the error terms. ChatGPT did not provide any references used in its explanation. When further prompted to provide its references, ChatGPT responded that it does not have access to its training data and is prohibited from disclosing the specifics of individual training datasets used.

2. Inspect the available data for pups and non-pups. Provide a list of 3-5 summaries of the data, including any conclusions about how the properties of the data might facilitate (or not) application of the state-space model described above, including estimation of the magnitude of process and observation error.

2a. Data summary of pups and nonpups

```
##packages
require(tidyverse)
require(kableExtra)
require(TMB)

setwd("/Users/tdolan/Documents/R-Github/AdvPopModeling/")

###load the data
sslData <- readRDS("/Users/tdolan/Documents/R-Github/AdvPopModeling/hw2/ssl-data.rds")
#sslData <- readRDS("hw2/ssl-data.rds") #relative paths not working.
#build dataframe with all years represented
pups <-as.data.frame(sslData$forrester_pups)%>%rename("icount_p"="icount")
nonpups <-as.data.frame(sslData$forrester_nonpups)%>%rename("icount_np"="icount")
ssl <-full_join(pups, nonpups, by="Year")%>%arrange(Year)
Year = data.frame(Year=seq(min(ssl$Year),max(ssl$Year),1))
ssltmb = right_join(ssl,Year)%>%arrange(Year)

###Summarize data completeness
ssl_sum <-ssltmb %>%summarize(
nyears =n_distinct(Year),
nyears_pups_counted=n_distinct(Year[!is.na(pupcount)]),
nyears_adults_counted=n_distinct(Year[!is.na(adultcount)]),
n_replicate_pupcounts=n_distinct(Year[icount_p>1]),
n_replicate_adultcounts=n_distinct(Year[icount_np>1]),
no_observations=n_distinct(Year[is.na(pupcount)&is.na(adultcount)]))

#average replicate counts from years where there are multiple counts prior to creating the grand mean a
ssl_piv <-select(ssl, -icount_p, -icount_np)%>%
  pivot_longer(c("pupcount","adultcount"), names_to = "stage",values_to = "count")
ssl_sum2 <-ssl_piv%>%group_by(Year,stage)%>%
  summarize(av_count=mean(count,na.rm=T),sd_count=sd(count,na.rm=T),.groups="keep")
sslsum_wide <-select(ssl_sum2, av_count, Year, stage)%>%
  pivot_wider(names_from = stage, values_from = av_count)%>%
  mutate(prod=pupcount/adultcount)
sslsum3 <-ssl_sum2 %>%group_by(stage)%>%summarize(mean_count=mean(av_count,na.rm=T),sd_count=sd(av_coun

##create across year summary table
sslsum3$n_years_with_counts<-c(ssl_sum[3],ssl_sum[2])
sslsum3$n_years_with_replicate_counts<-c(ssl_sum[5],ssl_sum[4])
```

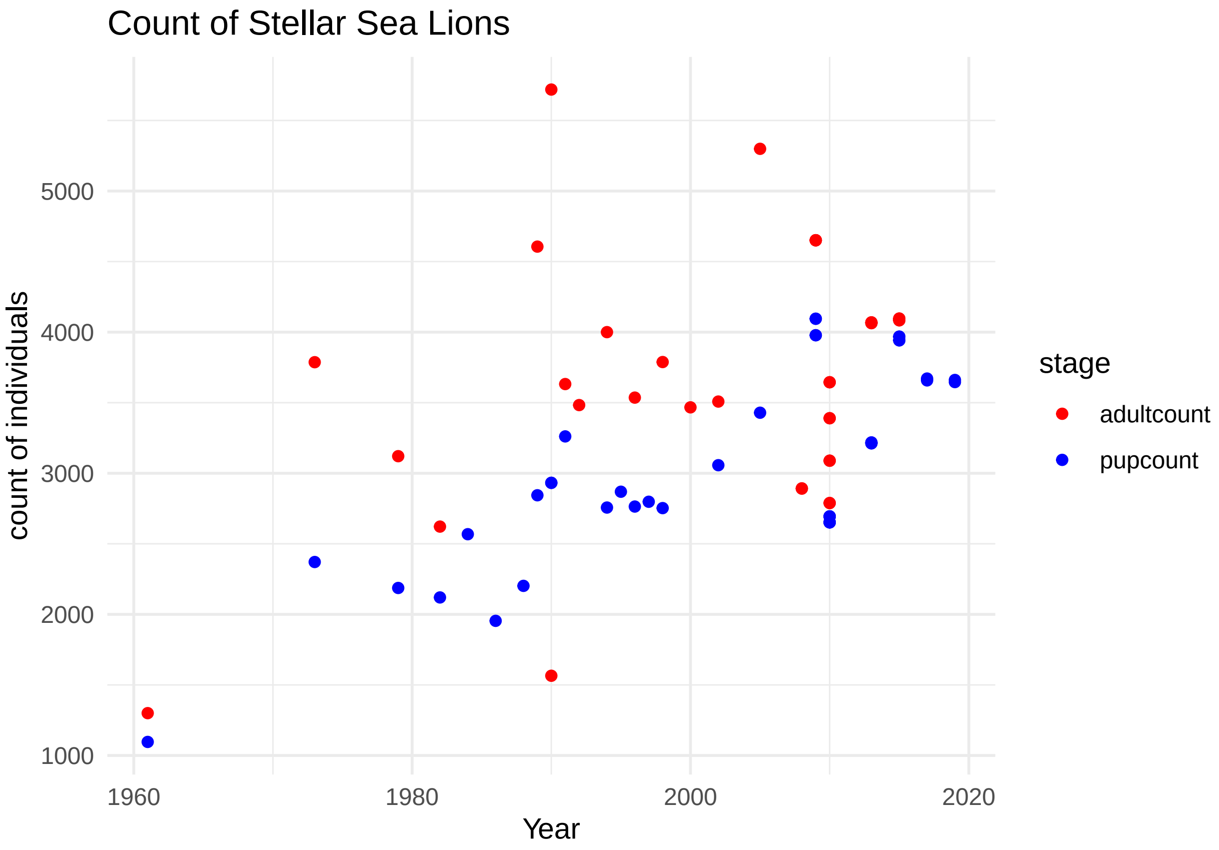
Table 1: Summary of Stellar sea lion sampling 1961-2019

	nonpups	pups
mean count	3617	2833
std. dev. count	845	683
n years sampled	19	23
n years with >1 replicate count	7	7
total years	58	58

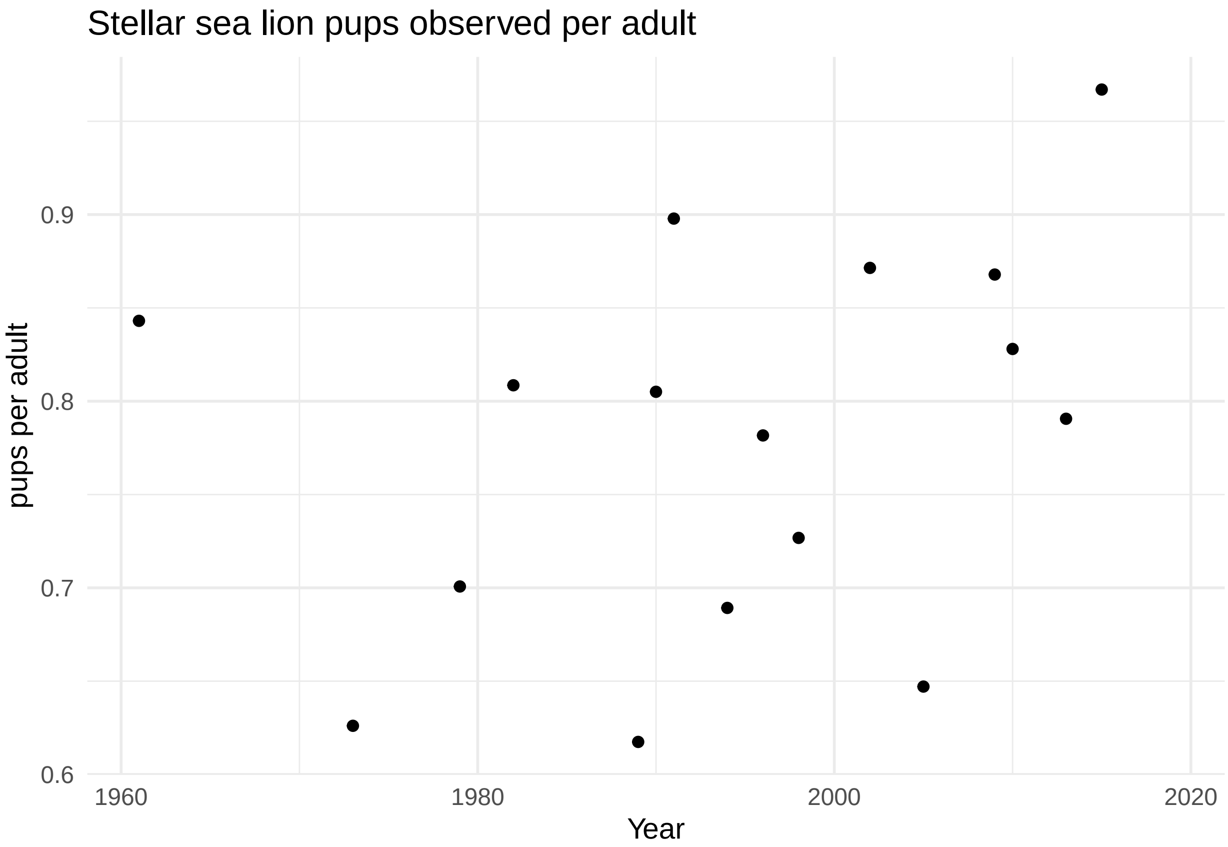
```r
sum_tab <-mutate(sslsum3, total_years= max(ssl$Year)-min(ssl$Year)) %>% t()
sum_tab<-as.data.frame(sum_tab[-1,])
colnames(sum_tab)<-c("nonpups","pups")
rownames(sum_tab)<-c("mean count", "std. dev. count", "n years sampled", "n years with >1 replicate cou

sum_tab %>%
  mutate(across(c("nonpups","pups"),as.numeric))%>%
  mutate(across(c("nonpups","pups"),round))%>%
  kbl(caption="Summary of Stellar sea lion sampling 1961-2019") %>% kable_styling()
```

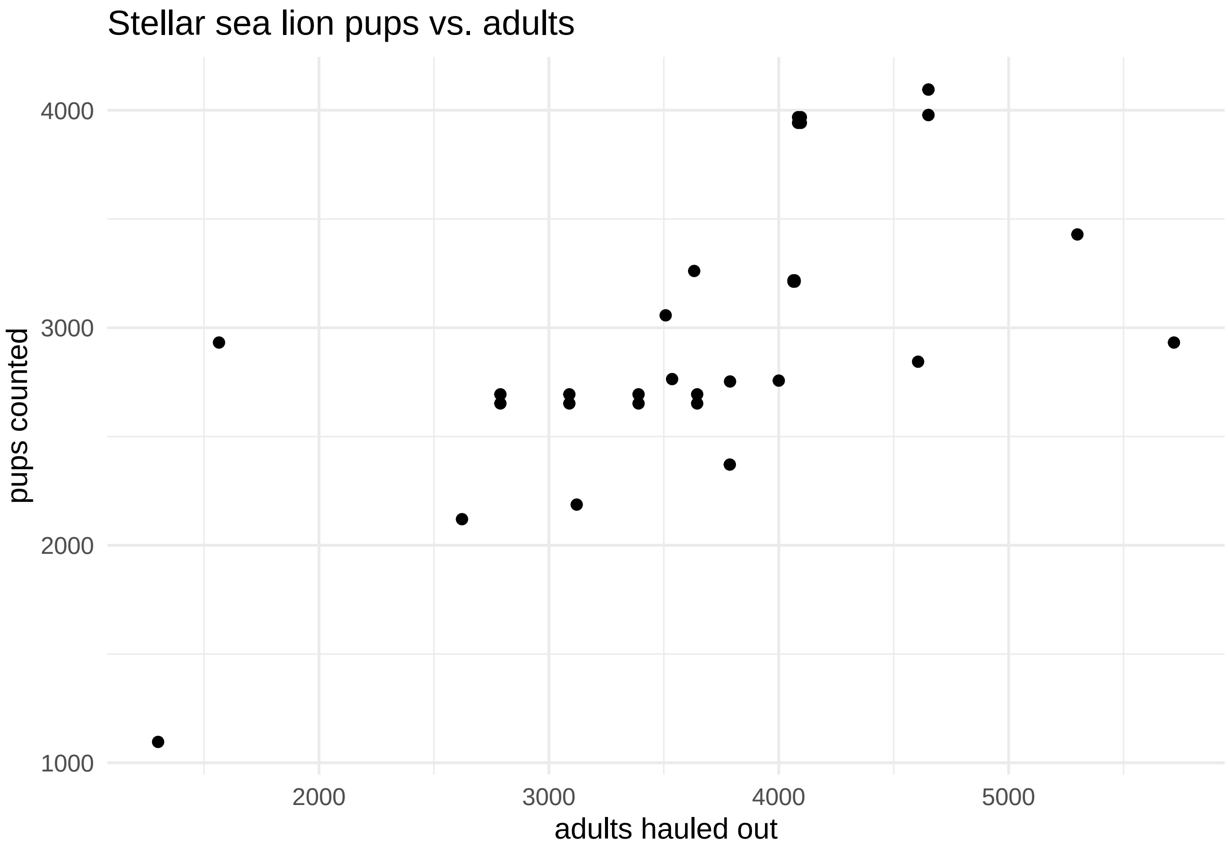
```r
#plot all data
ggplot(aes(Year,count, col=stage),data=ssl_piv)+
  geom_point()+
    scale_color_manual(values = c("red", "blue"))+
  #geom_smooth(aes(Year,count, col=stage),formula = count~Year, method=lm)+
  labs(title="Count of Stellar Sea Lions")+
  ylab("count of individuals")+
  theme_minimal()
```



```
#pups per adult
sslsum_wide%>%
  ggplot()+
  geom_point(aes(Year,prod))+
  labs(title="Stellar sea lion pups observed per adult")+
  ylab("pups per adult")+
  theme_minimal()
```



```
#pups v. adults
ssl%>%
  ggplot()+
  geom_point(aes(adultcount,pupcount))+
  labs(title="Stellar sea lion pups vs. adults")+
  ylab("pups counted")+xlab("adults hauled out")+
  theme_minimal()
```



2b. Data properties which might help/hinder application of the state space model

The Stellar sea lion dataset has highly inconsistent sampling, with many long periods of missing data, incomplete observations for certain years and multiple observations for others. Where there are certain years with multiple observations for that year, we may have a greater ability to understand observation error, relative to years where there are only a single observation. In years where we have only adult counts or only pup counts, it might be difficult to fit the process model because adult count is linked to pup survival in the process model. Adult observability and fecundity are treated as constant with respect to the state variables, so this may affect the uncertainty around those parameter estimates. However, state space models (SSM) offer an improvement over other methods in this respect because, intrinsic to the conceptual framework of SSM, the value of the state vector at a given point in time is seldom known with certainty.

3. Develop a TMB model that can fit the state-space model described above to the Forrester Island sea lion data

There are multiple associated .cpp files that will be used to run various iterations of this model. However, I have also included code chunks with the .cpp code in this RMarkdown file, using the argument eval=F so that these chunks are not evaluated, but using echo=T so that you can more easily see my work and thought process. I like to build the simplest version of the model first, then increase complexity with each iteration. Some of these model iterations will not run, but I am including them in hope of partial credit.

3a. First iteration (Model A)

Here is the model code copied from the file "hw2A_tara.cpp". In this iteration, when there are multiple counts for a single year, an average is used. Missing data are excluded from the model: only years where both pups and nonpups were sampled are included. There are no likelihood penalties for pup survival (phi_p) or fraction of nonpups hauled out (q) based on the pilot study. Phi_p and q are assigned the values from the pilot study and are not estimated within the model.

```
#include <TMB.hpp>
#include <vector>
```

```
#include <cmath>
#include <iostream>

template<class Type> Type square(Type x){return x*x;}
template<class Type>
Type objective_function<Type>::operator() ()
{
  //DATA//
  DATA_VECTOR(pups_data); //pups
  DATA_VECTOR(nonpups_data); //nonpups
  DATA_VECTOR(year); //time step
  int n = year.size();

  //PARAMETERS//
  PARAMETER_VECTOR(pups); //unobserved state variable, actual number of pups
  PARAMETER_VECTOR(nonpups); //unobserved state variable, actual number of pups
  PARAMETER(f); //fecundity
  PARAMETER(phi_np); //nonpup survival
  PARAMETER(sigma2_p); //process error on obs model pups
  PARAMETER(sigma2_np); //process error on obs model nonpups
  PARAMETER(tau2); //variance of the process error

//DECLARE TEMP VARIABLES
//everything has to be either data or a parameter or a temp variable.
 using namespace density;

vector<Type> nll =0;
std::vector<double> ypups(79, log(1000)); //sample mean pupcount in log space.
std::vector<double> ynp(79, log(1300)); //sample mean adultcount in log space.
Type psi_pups = 0.0;
Type psi_np = 0.0;
Type eps_p = 1000;
Type eps_np = 2000;
Type q = 0.3;
Type phi_p = 0.6;

//backtransform_params
Type ln_psi_pups = log(psi_pups);
Type ln_psi_np = log(psi_np);
Type ln_eps_p = log(eps_p);
Type ln_eps_np = log(eps_np);

// PROCESS MODEL
// there is a process model output for every year, not for every observation.
for (int i=0; i<n;i++) { //looping over years
  pups(i+1)=nonpups(i)*f + ln_psi_pups;
  nonpups(i+1)=pups(i)*phi_p + nonpups(i)*phi_np + ln_psi_np;
  nll-=dnorm(psi_pups,0,tau2,true);
  nll-=dnorm(psi_np,0,tau2,true);
  nll-=sum(dnorm(log(pups(i+1)),log(f*nonpups(i)),psi_pups,true));
  nll-=sum(dnorm(log(nonpups(i+1)),log(phi_p*pups(i))+log(phi_np*nonpups(i)),psi_np,true));
}
```

```
// OBSERVATION MODEL
for (int i=0; i<n;i++){
  ypups(i) = pups(i)+ln_eps_p; //observations of pups are the state variable pups plus error.
  nll-=sum(dnorm(log(pups_data(i)),log(ypups),eps_p,true)); //dnorm(conditioned, expectation,variance)
  nll-=dnorm(eps_p,0,sigma2_p,true); //observation error
  ynp(i) = q*nonpups(i)+ln_eps_np; //observations of nonpups are the state variable nonpups plus error.
  nll-=sum(dnorm(log(nonpups_data(i)),log(ynp(i)*q),eps_np,true)); //dnorm(conditioned, expectation,var
  nll-=dnorm(eps_np,0,sigma2_p,true); //observation error
}


//OUTPUTS
// put ADREPORT and REPORT here.
ADREPORT(phi_np);
ADREPORT(sigma2_p);
ADREPORT(sigma2_np);
ADREPORT(tau2);
  return nll;
}
```

4a. Prepare the data for TMB model A and run the model

```
#create the dataset
sslTMB_A <- sslsum_wide %>%select(-prod)
  #This dataset averages across counts when there are replicate counts
sslTMB_A <- sslTMB_A[complete.cases(sslTMB_A), ] #Missing data are removed from this version.
sslTMB_A$Year <-seq(0,nrow(sslTMB_A)-1,1) #Turn "Year" into a vector starting at zero for easy indexing

##compile the script
compile("/Users/tdolan/Documents/R-Github/AdvPopModeling/hw2/hw2A_tara.cpp")
dyn.load(dynlib("hw2/hw2A_tara"))

#feed in data
data <- tibble(pups_data = sslTMB_A$pupcount,
               nonpups_data = sslTMB_A$adultcount,
               year =sslTMB_A$Year)
#feed in params
params <-list(pups=rep(0,length(data$year)),
              nonpups=rep(0,length(data$year)),
              f=0,
              #phi_p=0,
              phi_np=0,
              logsigma2_p=0,
              logsigma2_np=0,
              #q=0,
              tau2=0)

#make a function object and declare random variables.
modelA<- MakeADFun(data,
                parameters,
                DLL="hw2A_tara")#not really sure what the random effects would be here.

#fit the model(?)
```

```
fitA <- nlminb(modelA$par, modelA$fn, modelA$gr)

#Get parameter uncertainties and convergence diagnostics
sdrA <- sdreport(modelA)
sdrA
summary(sdrA)
```

3b. Second iteration (Model B)

This is a more complex version of the model that uses all observations, instead of taking an average for each year. It also incorporates likelihood penalties on pup survival (phi_p) and adult observability (fraction of nonpups hauled out, q) based on sample data from a pilot study. This version estimates parameters q and f within the model, instead of setting them equal to the constants from the pilot study. This version does not run.

```
#include <TMB.hpp>
#include <vector>
#include <cmath>
#include <iostream>

template<class Type> Type square(Type x){return x*x;}
template<class Type>
Type objective_function<Type>::operator() ()
{
  //DATA//
  DATA_VECTOR(pups_data); //pups
  DATA_VECTOR(nonpups_data); //nonpups
  DATA_VECTOR(year); //time step
  DATA_VECTOR(pup_years); //years in which pup counts were made
  DATA_VECTOR(adult_years); //years in which adult counts were made
  int year = year.size(); //list years including replicate counts (??)
  int pup_obs = pup_years; // year associated with every pup count, even with replicates.
  int adult_obs = adult_years; // year associated with every adult count, even with replicates.

  //PARAMETERS//
  PARAMETER_VECTOR(pups); //unobserved state variable, actual number of pups
  PARAMETER_VECTOR(nonpups); //unobserved state variable, actual number of pups
  PARAMETER(f); //fecundity
  PARAMETER(phi_p); //pup survival - could use a constant from the pilot experiment at 0.6
  PARAMETER(phi_np); //nonpup survival
  PARAMETER(sigma2_p); //process error on obs model pups
  PARAMETER(sigma2_np); //process error on obs model nonpups
  PARAMETER(q); //observability of nonpups - could use a constant from the pilot experiment at 0.3
  PARAMETER(tau2); //variance of the process error

//DECLARE TEMP VARIABLES
//everything has to be either data or a parameter or a temp variable.
using namespace density

Type nll =0.0;
std::vector<double> ypups(79, log(1000)); //sample mean pupcount in log space.
std::vector<double> ynp(79, log(1300)); //sample mean adultcount in log space.
Type psi_pups = 0.0;
Type psi_np = 0.0;
Type eps_p = 1000; //sample std dev.
Type eps_np = 2000; //sample std dev.

//backtransform_params
Type ln_psi_pups = log(psi_pups);
Type ln_psi_np = log(psi_np);
Type ln_eps_p = log(eps_p);
Type ln_eps_np = log(eps_np);

// PROCESS MODEL
// there is a process model output for every year, not for every observation.
for (int i=0; i<uniqueYear;i++) { //looping over unique years.
  pups(i+1)= nonpups(i)*f + ln_psi_pups;
  nonpups(i+1)= pups(i)*phi_p + nonpups(i)*phi_np + ln_psi_np;
  nll-=dnorm(psi_pups,0,tau2);
  nll-=dnorm(psi_np,0,tau2);
  nll-=sum(dnorm(log(pups(i+1)),log(f*nonpups(i)),psi_pups,true));
  nll-=sum(dnorm(log(nonpups(i+1)),log(phi_p*pups(i)+phi_np*nonpups(i)),psi_np,true));
}

// OBSERVATION MODEL - I decided to split this into two for loops, so that we could loop over correct n
// observation model for pups
for(int i=0;i<pup_obs;i++){ //looping over observations of pups, even though there are multiple observa
  //if(i < 1){
    //continue;  //skip years where there are blanks
  //} else
  ypups(i) = pups(i)+ln_eps_p; //observations of pups are the state variable pups plus error.
  nll-=sum(dnorm(log(pups_data(i)),log(ypups(i)),eps_p,true)); //dnorm(conditioned, expectation,varianc
  nll-=dnorm(eps_p,0,sigma2_p); //observation error
}

// observation model for nonpups
for(int i=0;i<adult_obs;i++){ //looping over number of observations of nonpups, even though there are mu
  //if(i < 1){
    //continue;  //skip years where there are blanks
  //} else
  ynp(i) = q*nonpups(i)+ln_eps_np; //observations of nonpups are the state variable nonpups plus error.
  nll-=sum(dnorm(log(nonpups_data(i)),ynp(i)*q,eps_np,true)); //dnorm(conditioned, expectation,variance
  nll-=dnorm(eps_np,0,sigma2_p); //observation error
}


//likelihood penalties
  nll-=dnorm(phi_p,0.6,0.05); // penalty on pup survival(phi_p) mean=0.6, sd=0.05
  nll-=dnorm(log(q),log(0.3),log(0.1));//penalty on *fraction* of nonpups hauled out. mean=0.3, sd=0.1

//OUTPUTS
// put ADREPORT and REPORT here.
  ADREPORT(pups);
  ADREPORT(nonpups);
  ADREPORT(f);
  ADREPORT(phi_p);
  ADREPORT(phi_np);
  ADREPORT(sigma2_p);
  ADREPORT(sigma2_np);
  ADREPORT(q);
  ADREPORT(tau2);
  return nll;
}
```

4b. Prepare the data for TMB Model B and fit the model

```
#add -1 instead of NA to facilitate handling missing data.
ssltmb[is.na(ssltmb)]<- -1
ssltmb <-mutate(ssltmb, Year=Year-1961)#Turn "Year" into a vector starting at zero for easy indexing.
#ssltmb$Index <-seq(0,nrow(ssltmb)-1,1) #create an index

#implement the indexing method presented in class and in "hw2-snippet-20231026.Rmd"
pup_years <-ssltmb$Year[!is.na(ssltmb$pupcount)]
adult_years <-ssltmb$Year[!is.na(ssltmb$adultcount)]
#n_years <-unique(ssltmb$Year)

##compile the script
compile("hw2/hw2B_tara.cpp")
dyn.load(dynlib("hw2/hw2B_tara"))

#feed in data
data <- tibble(pups_data = ssltmb$pupcount,
               nonpups_data = ssltmb$adultcount,
               year =ssltmb$Year,
               pup_years=pup_years,
               adult_years=adult_years)

#feed in params initialized with realistic data
params <-list(pups=rep(0,length(data$year)),
              nonpups=rep(0,length(data$year)),
              f=0.7, #using our mean npups per adult hauled out, though possibly overestimating
              phi_p=0.6, #from the pilot study
              phi_np=0,
              logsigma2_p=0,
              logsigma2_np=0,
              q=0.3, #from the pilot study
              tau2=0)

#make a function object and declare random variables.
obj<- MakeADFun(data,
                parameters,
                DLL="hw2B_tara",
                random=c("pups","nonpups"))#not really sure how/if to specify state variables as random

#fit the model(?)
fit <- nlminb(model$par, model$fn, model$gr)

#Get parameter uncertainties and convergence diagnostics
sdr <- sdreport(obj)
sdr
```

```
summary(sdr)
```

5. Conduct a stochastic projection of the population through 2030. Summarize the population status.

I was not able to get the model to run, but here I have provided some pseudocode to illustrate how I would proceed if I had a working model.

```
years <-seq(2019-2030)
pred_pup <-c() #predictions of pups
pred_nonpup <-c() #predictions of nonpups
pred_ypups <-c() #predictions of pup observations
pred_ynp <-c() #predictions of nonpup observations

#model parameters that I would get from the model output
q
phi_p
phi_np
f
sigma2_p
sigma2_no
tau2

#progress the model forward

for (i in 1:length(years)){
  if (i==1){ #the first year is 2019, which we will have the pups and nonpups value from the model outp
  #so in this case we would populate the first row of the data vectors with data from the model output.
  }else{
  #process model
  pred_pup(i+1)= pred_nonpup(i)*f + log(dnorm(1,mean=0,sd=tau2))
  pred_nonpup(i+1)= pred_pup(i)*phi_p + pred_nonpup(i)*phi_np + log(dnorm(1,mean=0,sd=tau2))
  #observation model
  pred_ypups(i) = pred_pup(i)+log(dnorm(1,mean=0,sd=sigma2))
  pred_ynp(i) = q*pred_nonpup(i)+log(dnorm(1,mean=0,sd=sigma2))
}}
```

6. Using the same .cpp program, fit (a) an observation error only model (no process error), and(b) a process error only model. Summarize the results and comment on the implications for the estimates of population parameters, the population dynamics, and the prediction of population size in 2030.

6a. How would fitting a model with observation error only affect the model results, predictions and estimates

Process error represents stochasticity in the underlying system. A model without process error is purely deterministic. The uncertainty will be missing the component contributed by process error and thus could be an underestimate. Ignoring process error could lead to a model that fits observed data but might not accurately represent the underlying process, and thus could lead to poor out-of-sample predictions.

6a. Model C: Observation error only model c++ code

To get rid of process error, I removed the error term from the process model and set variance=0 in the dnorm function.

```
#include <TMB.hpp>
#include <vector>
#include <cmath>
#include <iostream>

//OBSERVATION ERROR ONLY
template<class Type> Type square(Type x){return x*x;}
template<class Type>
Type objective_function<Type>::operator() ()
{
  //DATA//
  DATA_VECTOR(pups_data); //pups
  DATA_VECTOR(nonpups_data); //nonpups
  DATA_VECTOR(year); //time step
  DATA_VECTOR(pup_years); //years in which pup counts were made
  DATA_VECTOR(adult_years); //years in which adult counts were made
  int year = year.size(); //list years including replicate counts (??)
  int pup_obs = pup_years; // year associated with every pup count, even with replicates.
  int adult_obs = adult_years; // year associated with every adult count, even with replicates.

  //PARAMETERS//
  PARAMETER_VECTOR(pups); //unobserved state variable, actual number of pups
  PARAMETER_VECTOR(nonpups); //unobserved state variable, actual number of pups
  PARAMETER(f); //fecundity
  PARAMETER(phi_p); //pup survival - could use a constant from the pilot experiment at 0.6
  PARAMETER(phi_np); //nonpup survival
  PARAMETER(sigma2_p); //process error on obs model pups
  PARAMETER(sigma2_np); //process error on obs model nonpups
  PARAMETER(q); //observability of nonpups - could use a constant from the pilot experiment at 0.3
  PARAMETER(tau2); //variance of the process error

//DECLARE TEMP VARIABLES
//everything has to be either data or a parameter or a temp variable.
using namespace density

Type nll =0.0;
std::vector<double> ypups(79, log(1000)); //sample mean pupcount in log space.
std::vector<double> ynp(79, log(1300)); //sample mean adultcount in log space.
Type eps_p = 1000; //sample std dev.
Type eps_np = 2000; //sample std dev.

//backtransform_params
Type ln_psi_pups = log(psi_pups);
Type ln_psi_np = log(psi_np);
Type ln_eps_p = log(eps_p);
Type ln_eps_np = log(eps_np);

// PROCESS MODEL
// To get rid of process error I set the variance to zero
for (int i=0; i<uniqueYear;i++) { //looping over unique years.
  pups(i+1)= nonpups(i)*f;
  nonpups(i+1)= pups(i)*phi_p + nonpups(i)*phi_np;
  nll-=sum(dnorm(log(pups(i+1)),log(f*nonpups(i)),0,true));
  nll-=sum(dnorm(log(nonpups(i+1)),log(phi_p*pups(i)+phi_np*nonpups(i)),0,true));
}

// OBSERVATION MODEL - I decided to split this into two for loops, so that we could loop over correct n
// observation model for pups
for(int i=0;i<pup_obs;i++){ //looping over observations of pups, even though there are multiple observa
  //if(i < 1){
    //continue;  //skip years where there are blanks
  //} else
  ypups(i) = pups(i)+ln_eps_p; //observations of pups are the state variable pups plus error.
  nll-=sum(dnorm(log(pups_data(i)),log(ypups(i)),eps_p,true)); //dnorm(conditioned, expectation,varianc
  nll-=dnorm(eps_p,0,sigma2_p,true); //observation error
}

// observation model for nonpups
for(int i=0;i<adult_obs;i++){ //looping over number of observations of nonpups, even though there are mu
  //if(i < 1){
    //continue;  //skip years where there are blanks
  //} else
  ynp(i) = q*nonpups(i)+ln_eps_np; //observations of nonpups are the state variable nonpups plus error.
  nll-=sum(dnorm(log(nonpups_data(i)),ynp(i)*q,eps_np,true)); //dnorm(conditioned, expectation,variance
  nll-=dnorm(eps_np,0,sigma2_p,true); //observation error
}


//likelihood penalties
  nll-=dnorm(phi_p,0.6,0.05,true); // penalty on pup survival(phi_p) mean=0.6, sd=0.05
  nll-=dnorm(log(q),log(0.3),log(0.1),true);//penalty on *fraction* of nonpups hauled out. mean=0.3, sd

//OUTPUTS
// put ADREPORT and REPORT here.
  ADREPORT(pups);
  ADREPORT(nonpups);
  ADREPORT(f);
  ADREPORT(phi_p);
  ADREPORT(phi_np);
  ADREPORT(q);
  ADREPORT(tau2);
  return nll;
}
```

6a. Fit observation error only model

```
##compile the script
compile("hw2/hw2B_tara.cpp")
dyn.load(dynlib("hw2/hw2C_tara"))

#feed in data
data <- tibble(pups_data = ssltmb$pupcount,
               nonpups_data = ssltmb$adultcount,
               year =ssltmb$Year,
               pup_years=pup_years,
               adult_years=adult_years)
```

```
#feed in params initialized with realistic data
params <-list(pups=rep(0,length(data$year)),
              nonpups=rep(0,length(data$year)),
              f=0.7, #using our mean npups per adult hauled out, though possibly overestimating
              phi_p=0.6, #from the pilot study
              phi_np=0,
              q=0.3, #from the pilot study
              tau2=0)

#make a function object and declare random variables.
obj<- MakeADFun(data,
                parameters,
                DLL="hw2C_tara",
                random=c("pups","nonpups"))#not really sure how/if to specify state variables as random

#fit the model(?)
fit <- nlminb(model$par, model$fn, model$gr)

#Get parameter uncertainties and convergence diagnostics
sdr <- sdreport(obj)
sdr
summary(sdr)
```

6a. How would fitting a model with observation error only affect the model results, predictions and estimates

Observation error represents uncertainty in the accuracy and precision of our ability to take real world measurements of an underlying process. A model without observation error will be missing the component of uncertainty contributed by imprecise, inappropriate (for the system) and/or inaccurate measurement and thus could be an underestimate. A model without observation error might mistake the variability in observed data for intrinsic variability of the system (stochastic process). If the sample variance is small compared to the variability of the system, this may lead to overconfident estimates. Ignoring observation error could lead to a model that does not capture the full extent of the noise in the observed data, leading to a poor fit. Predictions may be less accurate without observation error since the model does not fully represent how the observed data are generated. A model without observation error assumes that initial values of state variables are known with perfect precision. Misspecification of initial conditions can lead to inconsistent and unreliable long term predictions.

6b. Process error only model c++ code

```
#include <TMB.hpp>
#include <vector>
#include <cmath>
#include <iostream>

//PROCESS ERROR ONLY MODEL//

template<class Type> Type square(Type x){return x*x;}
template<class Type>
Type objective_function<Type>::operator() ()
{
  //DATA//
  DATA_VECTOR(pups_data); //pups
  DATA_VECTOR(nonpups_data); //nonpups
  DATA_VECTOR(year); //time step
  DATA_VECTOR(pup_years); //years in which pup counts were made
  DATA_VECTOR(adult_years); //years in which adult counts were made
  int year = year.size(); //list years including replicate counts (??)
  int pup_obs = pup_years; // year associated with every pup count, even with replicates.
  int adult_obs = adult_years; // year associated with every adult count, even with replicates.

  //PARAMETERS//
  PARAMETER_VECTOR(pups); //unobserved state variable, actual number of pups
  PARAMETER_VECTOR(nonpups); //unobserved state variable, actual number of pups
  PARAMETER(f); //fecundity
  PARAMETER(phi_p); //pup survival - could use a constant from the pilot experiment at 0.6
  PARAMETER(phi_np); //nonpup survival
  PARAMETER(sigma2_p); //process error on obs model pups
  PARAMETER(sigma2_np); //process error on obs model nonpups
  PARAMETER(q); //observability of nonpups - could use a constant from the pilot experiment at 0.3
  PARAMETER(tau2); //variance of the process error

//DECLARE TEMP VARIABLES
//everything has to be either data or a parameter or a temp variable.
using namespace density

Type nll =0.0;
std::vector<double> ypups(79, log(1000)); //sample mean pupcount in log space.
std::vector<double> ynp(79, log(1300)); //sample mean adultcount in log space.
Type psi_pups = 0.0;
Type psi_np = 0.0;

//backtransform_params
Type ln_psi_pups = log(psi_pups);
Type ln_psi_np = log(psi_np);

// PROCESS MODEL
// there is a process model output for every year, not for every observation.
for (int i=0; i<uniqueYear;i++) { //looping over unique years.
  pups(i+1)= nonpups(i)*f + ln_psi_pups;
  nonpups(i+1)= pups(i)*phi_p + nonpups(i)*phi_np + ln_psi_np;
  nll-=dnorm(psi_pups,0,tau2,true);
  nll-=dnorm(psi_np,0,tau2,true);
  nll-=sum(dnorm(log(pups(i+1)),log(f*nonpups(i)),psi_pups,true));
  nll-=sum(dnorm(log(nonpups(i+1)),log(phi_p*pups(i)+phi_np*nonpups(i)),psi_np,true));
}

// OBSERVATION MODEL - I decided to split this into two for loops, so that we could loop over correct n
// observation model for pups
for(int i=0;i<pup_obs;i++){ //looping over observations of pups, even though there are multiple observa
  ypups(i) = pups(i) //observations of pups are the state variable pups plus error.
  nll-=sum(dnorm(log(pups_data(i)),log(ypups(i)),0,true)); //dnorm(conditioned, expectation,variance)
}

// observation model for nonpups
for(int i=0;i<adult_obs;i++){ //looping over number of observations of nonpups, even though there are mu
  ynp(i) = q*nonpups(i); //observations of nonpups are the state variable nonpups plus error.
  nll-=sum(dnorm(log(nonpups_data(i)),ynp(i)*q,0,true)); //dnorm(conditioned, expectation,variance)
}


//likelihood penalties
  nll-=dnorm(phi_p,0.6,0.05,true); // penalty on pup survival(phi_p) mean=0.6, sd=0.05
  nll-=dnorm(log(q),log(0.3),log(0.1),true);//penalty on *fraction* of nonpups hauled out. mean=0.3, sd

//OUTPUTS
// put ADREPORT and REPORT here.
  ADREPORT(pups);
  ADREPORT(nonpups);
  ADREPORT(f);
  ADREPORT(phi_p);
  ADREPORT(phi_np);
  ADREPORT(sigma2_p);
  ADREPORT(sigma2_np);
  ADREPORT(q);
  ADREPORT(tau2);
  return nll;
}
```

6b. Fit process error only model

```
##compile the script
compile("hw2/hw2D_tara.cpp")
dyn.load(dynlib("hw2/hw2D_tara"))

#feed in data
data <- tibble(pups_data = ssltmb$pupcount,
               nonpups_data = ssltmb$adultcount,
               year =ssltmb$Year,
               pup_years=pup_years,
               adult_years=adult_years)

#feed in params initialized with realistic data
params <-list(pups=rep(0,length(data$year)),
              nonpups=rep(0,length(data$year)),
              f=0.7, #using our mean npups per adult hauled out, though possibly overestimating
              phi_p=0.6, #from the pilot study
              phi_np=0,
              logsigma2_p=0,
              logsigma2_np=0,
              q=0.3, #from the pilot study
              tau2=0)

#make a function object and declare random variables.
obj<- MakeADFun(data,
                parameters,
                DLL="hw2D_tara",
                random=c("pups","nonpups"))#not really sure how/if to specify state variables as random

#fit the model(?)
```

```r
fit <- nlminb(model$par, model$fn, model$gr)

#Get parameter uncertainties and convergence diagnostics
sdr <- sdreport(obj)
sdr
summary(sdr)
```