running-BERTA-tutorial-part3

Christopher L. Cahill

28 Feb 2022

Purpose

The purpose of this tutorial is to work through the calculations and population dynamics model in the `BERTA_single_lake.stan` file, which is a single lake version of the model fitted in Cahill et al. (2021). Learners should familiar with tutorials 1 and 2 before reading further, as I assume folks already know how to compile a `.stan` file, read data into that file from R, and run that model. I have done my best to limit the total number of equations in this document, but a key bit of this is understanding how to take a mathematical equation and convert that into stan code (and vice versa).

Goals

- Understand the main components of the `BERTA_single_lake.stan` file
- Develop intuition for how the `.stan` file takes data and parameters to generate population dynamics predictions
- Work through the various parts of the model, including data, parameters, model structure and predictions, and the log posterior calculations
- Discuss the algorithm in the generated quantities block that calculates equilibrium Maximum Sustainable Yield MSY and instantaneous fishing mortality F_{MSY} at MSY

A note on indexing

Indexing is tricky when converting between ages of Walleye (age 2-20 in our model) and index a (which ranges from 1-19). This is necessary because indexing in a matrix, vector, or array starts at slot 1 in Stan, and so rather than write the biological value I have written everything in index a values (again, these range form 1-19 in our model but correspond to fish of ages 2-20).

The same goes for years 1980-2028 in our model. Thus, t = 1 equates to year = 1980 while t = 49 equates to 2028. This is a bit tricky to get used to and so I make special note of it up front.

Data block:

The data block in lines 6-44 brings in all relevant information from Fall Walleye Index Netting (FWIN) surveys, life history parameters, and values used for setting lake-specific informative priors.

This section also includes several so-called "control" values that allow users to toggle among different model run options (e.g., Ricker vs. Beverton-Holt recruitment dynamics).

Control variables should all have `_ctl` in the variable name. I do this so that I did not have to recompile the model (which takes a lot of time) for many different model variant runs and tests. Each of the values in the data section should be defined or commented, and thus we will not discuss it further.

Transformed data block:

The transformed data block runs from line 45 to 105. This section is only run a single time when a .stan file is executed, and it transforms data (see data block) and calculates things that you need to run your model. Lines 46-61 declare the integer, real, and vector values that we will use to store the calculations in this section.

If this section confuses you, remember that I did this for ease of managing lakes and FWIN surveys in those lakes across an entire landscape. One could also read these values in as data, but it would take a bit more programming on the R side of things!

```
// calculate vul, length-age, M-age, fec-age
sbro = 0; // initialize
for(a in 1:n_ages){
  v_a[a] = ((linf/lbar)*(1 - exp(-vbk * ages[a])))^phi;
  if(vul_ctl == 0){
    //v_f_a[a] = ((linf/lbar)*(1 - exp(-vbk * ages[a])))^psi; // used in paper
    v_f_a[a] = (1 - exp(-vbk * ages[a]))^psi;  // length^2 selectivity
  }
  if(vul_ctl == 1){
    v_f_a[a] = 1 / (1 + exp(-(ages[a] - ah_vul) / sd_vul)); // logistic selectivity
  }
  l_a[a] = (linf/lbar)*(1 - exp(-vbk * ages[a]));
  M_a[a] = M/l_a[a]^theta;
  w_a[a] = 0.00001*(linf*(1 - exp(-vbk * ages[a])))^wl_beta;
  if(a < a50){
    f_a[a] = 0;
  } else {
    // relative weight at age assumed to follow vb
    f_a[a] = fmax(0, (l_a[a]^wl_beta));
  }
  if(a == 1){
    Lo[a] = 1;
  } else{
    Lo[a] = Lo[a-1]*exp(-M_a[a-1]);
  }
  sbro += f_a[a]*Lo[a];
}
ln_ar_mean = log(cr_prior/sbro);
```

This probably looks complex, but it isn't the end of the world. It is a fancy looking loop across ages a that calculates

- Spawner biomass per recruit in the unfished condition `sbro`
- Vulnerability at age to the survey nets `v_a[]`
- Vulnerability at age to fishing `v_f_a[]`
- Relative length at age `l_a[]`
- Lorenzen instantaneous natural mortality at age `M_a[]`

- Weight at age `w_a[]`
- Relative fecundity at age `f_a[]`, which is assumed to scale with `w_a[]` above age at 50% maturity `a50`
- Survivorship at age `Lo[]`
- A lake specific informative prior for the mean of the natural log of stock-recruit α `ln_ar_mean`

These are fairly common inputs for an age structured stock assessment model, and so I won't type out the specific equations used here. If you are confused, check out Walters and Martell (2004) box 3.1 or the equations in Cahill et al. (2021). Similarly, these are the same as the "leading parameter vectors" section in the spreadsheets that Carl and I shared with Laura and Andy in January-February 2022.

You should be aware that we are assuming a different vulnerability to fishing `v_f_a[]` function than what was used in Cahill et al. (2021). We did this to ensure the simulation testing and harvest control rule development was more conservative from a risk-of-overfishing perspective, and this change doesn't change the main results from Cahill et al. (2021). All of our model runs for this harvest control rule contract are assuming this logistic fishing vulnerability.

The rest of the transformed data block looks hard but is actually just goofy code that I used to make the "observed" Spawning Stock Biomass (SSB) values for plots:

```
// calculate the rowsums for each survey, observed SSB
// SSB_C(t)=sum over a of fec(a)*C(a,t)/[vage(a)Nnet(t)Paged(t)]
for(i in 1:n_surveys){
  SSB_Cn[i] = 0;
  SSB_Cd[i] = 0;
  SSB_C[i] = 0;
  for(a in 1:n_ages){
   counter = counter + 1;
   SSB_Cn[i] += f_a[a]*caa[i, a]*(1/v_a[a]);
   caa_obs[counter] = caa[i,a];
  }
  SSB_Cd[i] = prop_aged[i]*effort[i];
  SSB_C[i] = SSB_Cn[i] / SSB_Cd[i];
}
```

All this is doing is calculating the "observed" SSB `SSB_C[]` for each year t as

$$SSB_{C_t} = \sum_{a=1}^{a=19} \frac{f_a \cdot catch_{a,t}}{v_a \cdot nnet_t \cdot paged_t} \tag{1}$$

Here, $catch_{a,t}$ is the catch in the FWIN nets of age a fish in year t, $nnet_{a,t}$ and and $paged_{a,t}$ are the number of nets set and the proportion of critters with age estimates in year t, respectively. Again, these values specifically are used for plotting back in R after the model is fitted to data.

Parameters block:

The parameters block has all estimated parameters in a given stan file. In our case, the parameters are:

```
parameters {
  vector<lower=0>[2] v;                          // early period F v[1] or late F v[2]
  real<lower=0> Ro;                              // average unfished recruitment
  real ln_ar;                                    // ln stock-recruit a
  vector[n_years-2] w;                           // recruitment anomalies--first 2 yrs for initiaza
```

```
  real<lower=G_bound[1], upper=G_bound[2]> G;        // is population at equilibrium (1) or declining (<
  //real<lower=0> phi;                               // negative binomial parameter
  //vector<lower=0, upper=1>[n_lakes] su_stock;      // estimate stocked fish survival
}
```

All parameters are described in their comments. In total, how many parameters is this `.stan` file estimating?

Note that `phi` and `su_stock` were things that I was estimating (or trying to) but mostly didn't have success with given the FWIN data. I left them in in case Spok (Andy) wants to play with them sometime.

Transformed parameters block:

This section takes your data, transformed data, and parameters and runs the population dynamics model to create model predictions, which will be contrasted against your observations in the model section that comes directly after the transformed parameters block. This is the workhorse chunk of code that is doing most of the calculations.

For the purposes of understanding this section of code, let's assume for the time being that we have a single value for each of the parameters described in the previous section. This will help us see what is going on in terms of the calculations and the loops of misery.

Let's start with some simple stuff–lines 116-150 simply declare vectors, real values, and arrays necessary to store our calculations. Note the comments after each bit (to tell you what is going on).

```
transformed parameters {
  vector<lower=0>[n_years] F_vec;                   // Insantaneous fishing mortality
  real Nat_array[n_ages, n_years];                  // Numbers at age
  vector [n_years] SSB;                             // Spawning stock biomass
  vector<lower=0>[n_obs] caa_pred;                  // predicted catch at age
  real<lower=0> br;                                 // derived stock-recruit b from sbro, Ro
  vector<lower=0>[n_years] R2;                      // kick out recruits rather than N(a,t)
  vector[n_ages] Su_Fearly;                         // survivorship including fishing--early
  vector[n_ages] Su_Flate;                          // survivorship including fishing--late
  real sbrf_early;                                  // spawning biomass per recruit fished
  real sbrf_late;                                   // spawning biomass per recruit fished
  real<lower=0> pinit;                              // how much depletion from Ro
  real<lower=0> Rinit;                              // initial recruitment
  vector[n_surveys] SSB_obs;                        // SSB observed
  vector[n_years] pred_N_catch;                     // predicted catch N/ha, *not vulN*
  vector[n_years] pred_B_catch;                     // predicted catch biomass/ha
  real<lower=0> SPR;                                // spawning potential ratio
  real<lower=0> SSB_bar;                            // average ssb survey years
  real<lower=0> SBR;                                // spawning biomass ratio
  real<lower=0> counter_SSB;                        // hack for SBR calcs
  real<lower=0> cr;                                 // compensation ratio kick out to check math
  real<lower=0> sbro_report;                        // report sbro
  real ln_ar_mean_report;                           // report the ar mean
  vector<lower=0>[n_ages] l_a_report;               // report length at age a
  vector<lower=0>[n_ages] Lo_report;                // report survivorship unfished (F=0)
  vector<lower=0>[n_ages] v_a_report;               // report net vulnerability age a
  vector<lower=0>[n_ages] v_f_a_report;             // report angling vulnerability age a
  vector<lower=0>[n_ages] f_a_report;               // report fec at age a
  vector<lower=0>[n_ages] w_a_report;               // report weight at age a
  vector<lower=0>[n_ages] M_a_report;               // report M at age a
```

```
  // calculate sbrf
  sbro_report = sbro;
  ln_ar_mean_report = ln_ar_mean;
  sbrf_early = 0;
  sbrf_late = 0;
  //... lots more code after
}
```

Some of these values are used for "counting" (e.g., see `counter_SSB`) which helps simplify some calculations, and other values are "report" values (e.g., see `l_a_report`). Report values are sort of silly, but it turns out that you cannot record a value from the transformed data section without copying it into some sort of matrix or vector or array in the transformed parameters section. Again, this is a silly quirk of programming in Stan, but this should help you understand the *why* behind these bits of code.

Once again, it is important to remember that these things are *calculated* from your data, parameters (see previous section), and model structure (discussed below). It is probably best to call these things *derived* values.

Continuing on in the transformed parameters section we see the following loop across ages a in lines 151-171:

```
for(a in 1:n_ages){
    if(a == 1){
      Su_Fearly[a] = 1;
      Su_Flate[a] = 1;
      } else {
        Su_Fearly[a] = Su_Fearly[a-1]*exp(-M_a[a-1] - v_f_a[a-1]*v[1]);
        Su_Flate[a] = Su_Flate[a-1]*exp(-M_a[a-1] - v_f_a[a-1]*v[2]);
      }
    sbrf_early += f_a[a]*Su_Fearly[a];
    sbrf_late += f_a[a]*Su_Flate[a];

    // fill out report vectors
    l_a_report[a] = l_a[a];
    Lo_report[a] = Lo[a];
    v_a_report[a] = v_a[a];
    v_f_a_report[a] = v_f_a[a];
    f_a_report[a] = f_a[a];
    w_a_report[a] = w_a[a];
    M_a_report[a] = M_a[a];
  }
  SPR = sbrf_late / sbro;
  //... lots more code after
```

The first bit of this code is setting up two vectors `Su_Fearly` and `Su_Flate`, which correspond to survivorship in the fished condition for both F_{early} and F_{late}. This calculation is straightforward–the first value or age of the vector is set at 1.0 and then the remaining survivorship values are calculated recursively. In math-speak, this early period survivorship vector is calculated as:

$$Lx_{F_{early_a}} = 1.0 \qquad \text{for a} = 1 \tag{2}$$

$$Lx_{F_{early_a}} = Lx_{F_{early_{a-1}}} \cdot e^{-M_{a-1} - v_{f_{a-1}} \cdot F_{early}} \qquad \text{for a} > 1 \tag{3}$$

The next bit of code after the survivorship in the early or late period fished condition (i.e., $Lx_{F_{early_a}}$ or $Lx_{F_{late_a}}$) calculates a term called `sbrf_early` and `sbrf_late`. The names of these are "spawning biomass per recruit" given either the early or late period fishing mortality rate. These names are an unfortunate technical debt from earlier coding and are technically incorrect.

What we are calculating in lines 159-160 actually should be called relative egg production per recruit. If you go back up to the calculations in the transformed data section, you will see that this model assumes egg production is proportional to the weight of mature fish, which is assumed to be a knife-edge function of age.

Regardless of the name, these `sbrf_early` and `sbrf_late` terms are calculated as the sumproduct of the `f_a` and either the early or late period surivorship vectors:

$$sbrf_{early} = \sum_{a=1}^{a=19} f_a \cdot L_{F_{early_a}} \qquad \text{for early period years 1980-1996} \tag{4}$$

$$sbrf_{late} = \sum_{a=1}^{a=19} f_a \cdot L_{F_{late_a}} \qquad \text{for late period years 1997-2018+} \tag{5}$$

If you think critically about this, you should be starting to see that the fancy sumproduct symbols in "math speak" are just a for loop across the index of the summation sign and some sort of multiplication on the right hand side of the calculation, which are stored or "summed up" across all indeces for the loop in "programming speak."

The rest of the code section above is just transferring calculated values to report vectors so that we can kick them out to R after our analysis as discussed above, and so we will move on.

The next chunk of code is:

```
// Calculate recruitment b's, Rinit's
if(rec_ctl==0){ // ricker b
  br = (ln_ar + log(sbro)) / (Ro*sbro);
}
if(rec_ctl==1){ // bev-holt b
  br = (exp(ln_ar)*sbro - 1)/ (Ro*sbro);
}
if(Rinit_ctl == 0){
  Rinit = G*Ro;
}
if(Rinit_ctl == 1){
  if(rec_ctl==0){ // ricker Req
    Rinit = (ln_ar + log(sbrf_early)-log(G)) / (br*sbrf_early);
  }
  if(rec_ctl==1){ // bev-holt Req
    Rinit = (exp(ln_ar)*(sbrf_early-1)) / (br*sbrf_early);
  }
}
pinit = Rinit / Ro;
```

This chunk of code is using `rec_ctl` to toggle between the Ricker and Beverton-Holt stock-recruitment models. Here we are setting `br`, which is the β term in the stock-recruitment relationship, as a function of `sbro`, `ln_ar`, and `Ro`. Because the math for the Ricker and Beverton-Holt models differs, you have to calculate these `br` terms differently.

The `Rinit_ctl` value is always set at 0 for the model runs in Cahill et al. 2021 and thus our fits to be used with the harvest control rule simulation. Thus, the initial recruitment `Rinit` is set as `Ro` multiplied by some "depletion" parameter `G`.

When `Rinit_ctl` value was set to one we were testing the ability of the model to be "fixed" at a low or fished equilibrium point using the mathematics in box 3.1 in Walters and Martell 2004. It didn't work well, but I've left it in here for now in case I ever need to come back to it.

The last bit of this code calculates `pinit`, which simply is the depletion proportion from `Ro`. We didn't use this in our paper and I won't use it any more in this script, but it is one of those nifty things you can do with Bayesian analyses.

The next bit of code is:

```
// Initialize F(t) fishing rate vector from start year to 2018
for(t in 1:n_years){
  if(t < which_year){
    F_vec[t] = v[1];
  }
  if(t >= which_year){
    F_vec[t] = v[2];
  }
  SSB[t] = 0;
  pred_N_catch[t] = 0;
  pred_B_catch[t] = 0;
}
```

All this code is doing is initializing a fishing rate vector `F_vec[]`, and the if-statement is toggling between the early and late period fishing mortality rates. Additionally, this loop is initializing three additional vectors: `SSB[]`, `pred_N_catch[]`, and `pred_B_catch[]`.

This initialization thing might seem stupid but the reason you have to do this is because these three vectors are calculated as sumproducts of various stuff below, and if you don't set these vectors to 0.0 C++ really decides to screw your life up. In particular, what happens is that the program will keep adding to SSB or pred_N_catch or pred_B_catch for each iteration of the .stan file. This might not seem like a problem until you recall (in a previous tutorial) we talked about how this file is actually being run many, many times during a full Bayesian analysis. This results in nonsensical values.

Thus, we set them to zero here to avoid depression and misery.

The next section of code features even more initialization:

```
// Initialize the N(at) array age structure for t = 1,2
cr = exp(ln_ar)*sbro;
for(t in 1:2){
Nat_array[1, t] =  Rinit;
if(t == 1){
  for(a in 2:n_ages){
    Nat_array[a, t] = Nat_array[a-1, t]*exp(-M_a[a-1] - v_f_a[a-1]*F_vec[1]);
    SSB[t] += Nat_array[a ,t]*f_a[a];
    pred_N_catch[t] += Nat_array[a ,t]*v_f_a[a];
    pred_B_catch[t] += Nat_array[a ,t]*v_f_a[a]*w_a[a];
  }
  pred_N_catch[t] = pred_N_catch[t]*(1-exp(-F_vec[1]));
  pred_B_catch[t] = pred_B_catch[t]*(1-exp(-F_vec[1]));
}
if(t == 2){
  for(a in 2:n_ages){
    Nat_array[a, t] = Nat_array[a, 1];
  }
```

```
}
SSB[t] = SSB[1];
pred_N_catch[t] = pred_N_catch[1];
pred_B_catch[t] = pred_B_catch[1];
}

// Calculate the N(at) array and derived outputs
R2[1] = Nat_array[1, 1];
R2[2] = Nat_array[1, 2];

SSB_bar = 0;
counter_SSB = 0;
```

All of this scary looking code is just initializing our model, similar to the last section.

First, we calculate compensation ratio `cr` as $cr = \alpha \cdot sbro$.

Next, we loop through t = 1,2 to initialize all our fancy vectors and matrices for the first two time steps.

In addition to initializing across t = 1,2, we need to fill in our `Nat_array[]` across rows for a = 2-19 (i.e., ages 3-20) To do this, we loop across a = 2-19 and calculate the number of critters expected to be there given the number in the previous age as:

$$N_{a,t=1} = Rinit \qquad \text{for a} = 1 \tag{6}$$

$$N_{a,t=1} = N_{a-1,t=1} \cdot e^{-M_{a-1} - vf_{a-1} \cdot Fvec_{t=1}} \qquad \text{for a} > 1 \tag{7}$$

We then fill in the second column of `Nat_array[]` by setting it equal to the first column where t = 1:

$$N_{a,t=2} = N_{a,t=1} \tag{8}$$

We briefly discussed this at the beginning of this tutorial, but why do we initialize the first two t values? Because we are assuming critters recruit to the survey gear at age 2. So this model sets the first two recruitment values in `Nat_array[a = 1, t = 1,2]` to Rinit.

Remember, programming indeces go by row (ages) and then column (year). We discussed this above, but you must also remember that the first age is indexed as a = 1, but this actually represents age 2 Walleye (see above). Similarly, t starts at 1 (year 1980) and runs to 49 (corresponding to year 2028).

Once we have filled in or initialized the `Nat_array[]` for t = 1,2 and all ages a, we can calculate the first two Spawning Stock Biomass SSB values and the predicted number or biomass of fish vulnerable to fishing in each year as:

$$SSB_t = \sum_{a=2}^{a=20} f_a \cdot N_{a,t} \qquad \text{spawning stock biomass in year t} \tag{9}$$

$$catch(numbers)_t = \sum_{a=2}^{a=20} N_{a,t} \cdot vf_a \cdot (1 - e^{-Fvec_t}) \qquad \text{predicted numbers harvested in year t} \tag{10}$$

$$catch(biomass)_t = \sum_{a=2}^{a=20} N_{a,t} \cdot vf_a \cdot w_a \cdot (1 - e^{-Fvec_t}) \qquad \text{predicted biomass harvested in year t} \tag{11}$$

The final four lines of stan code chunk above simply kick out the recruits as a vector called `R2`, which is shorthand for age two recruits. I did this for plotting purposes. The next two lines initialize some more values that are used for plotting. So much initialization, but we are done with that hogwash for now.

If you take a step back and think, this is actually pretty crazy. Lines 115-235 in the transformed parameters section are *just initializing all of your matrices and vectors so that we can run the population dynamics model*, which really is only about 20 lines of code and is what we jump into now in lines 237-258:

```
for(t in 3:n_years){
  if(rec_ctl==0){// ricker
    Nat_array[1, t] = SSB[t-2]*exp(ln_ar - br*SSB[t-2] + w[t-2]);
  }
  if(rec_ctl==1){// beverton-holt
    Nat_array[1, t] = SSB[t-2]*exp(ln_ar + w[t-2]) / (1 + br*SSB[t-2]);
  }
  for(a in 2:n_ages){
    Nat_array[a, t] = Nat_array[a-1, t-1]*exp(-M_a[a-1] - v_f_a[a-1]*F_vec[t-1]);
    SSB[t] += Nat_array[a ,t]*f_a[a];
    pred_N_catch[t] += Nat_array[a ,t]*v_f_a[a];
    pred_B_catch[t] += Nat_array[a ,t]*v_f_a[a]*w_a[a];
  }
  pred_N_catch[t] = pred_N_catch[t]*(1-exp(-F_vec[t]));
  pred_B_catch[t] = pred_B_catch[t]*(1-exp(-F_vec[t]));
  R2[t] = Nat_array[1, t];
  // calculate mean SSB across survey years
  if(t >= survey_yrs[1] && t <= survey_yrs[2]){
    counter_SSB += 1;
    SSB_bar += SSB[t];
  }
}
SSB_bar = SSB_bar / counter_SSB;
SBR = SSB_bar / (Ro*sbro);
```

This code is doing much of the heavy lifting in this program. For each year from year 3 to n_years, it is

- setting the number of recruits using either Ricker or Beverton-Holt stock-recruit functions
- recursively calculating how many critters remain given the numbers at age, vulnerability at age (fishing), and instantaneous mortality rates
- calculating the SSB, predicted(numbers) catch, and predicted(biomass) of catch exactly as it was calculated in the previous section (see equations 9-11)
- Setting the newly calculated age-2 recruit value in R2 given the stock-recruit functions
- It then calculates `SSB_bar`, which is just the average SSB during the survey years for a given lake (i.e., the first and last survey years are what I mean here). This was used for plotting.

In math form, the stock-recruitment functions can be written as:

$$R_t = SSB_{t-2} \cdot e^{ln(\alpha) - br \cdot SSB_{t-2} + w_{t-2}} \quad \text{Ricker stock-recruit function} \quad (12)$$

$$R_t = SSB_{t-2} \cdot e^{ln(\alpha + w_{t-2})/(1 + br \cdot SSB_{t-2})} \quad \text{Beverton-Holt stock-recruit function} \quad (13)$$

where R_t is just the number of age two recruits.

Once we set the number of recruits, we recursively fill in the age-structured population dyanmics model using equations similar to what we have already used above:

$$N_{a,t} = R_{t-2} \qquad \text{for a} = 1, \text{t} > 2 \tag{14}$$

$$N_{a,t} = N_{a-1,t-1} \cdot e^{-M_{a-1} - vf_{a-1} \cdot Fvec_t} \qquad \text{for a} > 1, \text{t} > 2 \tag{15}$$

To put this in words, once we set the number of new recruits for a given year, we simply 'fill in' the rest of the numbers of critters at each age. To do this, you take the number of fish in the previous time step t-1 and age cohort a - 1, and move them forward in the numbers at age array. The number of fish in the previous step is exposed to some instantaneous mortality (which is estimated) and vulnerability to fishing that those fish experienced in that previous step.

Once you've done that you simply just repeat the process until you reach the total number of years you want to run your model for. In our case that is `n_years`. Everything else used in our model is a derived calculation originating from this relatively straightforward age structured model.

We now need to create the model predictions that we will contrast against the observed catch at age data from the FWIN surveys. The final chunk of code in the transformed parameters section (lines 262-279) is the following:

```
// Calculate the preds vector
// C(a,t)=N(a,t)*nnet(t)Paged(t)*v_a(a)
for(hack in 1:1){// hack to initialize j = 0
  int j = 0;
  for(i in 1:n_surveys){
    for(a in 1:n_ages){
      j += 1;
      caa_pred[j] = 0;                          // initialize
      caa_pred[j] = Nat_array[a, year[i]]*      // numbers(a,t)
      prop_aged[i]*                             // prop_aged
      effort[i]*                                // survey effort
      v_a[a];                                   // vulnerability to survey gear
    }
    if(get_SSB_obs==1){
      SSB_obs[i] = SSB_C[i];
    }
  }
}
```

This chunk of code is admittedly pretty jank. I needed to create a vector of catch at age predictions for the model section below. Note that this catch at age vector `caa_pred` is used to contrast the observed catch at age in the FWIN nets in the Poisson likelihood in the model block (we'll discuss this shortly). The main reason for doing things this way is that it speeded up the model (i.e., it allowed me to use a vectorized likelihood call in the model block).

The key trick for understanding this section of code is recognizing that each lake had a finite number of surveys `n_surveys` and given those surveys we can calculate a predicted catch at each age in the FWIN nets for each surveyed year as:

$$caa_{pred_t} = \sum_{a=1}^{a=19} N_{a,t} \cdot paged_t \cdot nnets_t \cdot v_a \tag{16}$$

Note here that caa_{pred_t} is a vector that has dimensions of the number of ages in the model (ages 2-20) multiplied by number of surveys that occurred for a given lake. That was tricky to code, hence the weird looking loops.

Also, the if() loop is simply another plotting tool, and you don't need to worry about it–it is literally just transferring the `SSB_C` values to an `SSB_obs` vector.

Boom, we are done with that miserable transformed parameters section.

Model block:

The model block takes the data you read in, the predictions you generate (via parameter values and your model structure in the transformed parameters section), and contrasts them in a likelihood. It then sums up the log of the priors and the log of the likelihood to generate a log posterior automatically.

The model block for BERTA is:

```
model {
  // priors:
  v[1] ~ normal(v_prior_early, prior_sigma_v[1]);
  v[2] ~ normal(v_prior_late, prior_sigma_v[2]);
  Ro ~ lognormal(Ro_mean, Ro_sd);
  ln_ar ~ normal(ln_ar_mean, ln_ar_sd);
  G ~ normal(0,prior_sigma_G);
  to_vector(w) ~ normal(prior_mean_w, prior_sigma_w);

  // likelihood
  caa_obs ~ poisson(caa_pred);

  // NOTE: Stan is creating log(posterior) = log(likelikhood) + log(priors)

  // extra stuff for negative binomial model and stocking survival
  // phi ~ cauchy(0,3);
  // su_stock ~ beta(2,2);
  // caa_obs ~ neg_binomial_2(caa_pred, phi);
}
```

I always define my priors first (i.e., v[1], v[2], Ro, ln_ar, G, and w[]), and then define my likelihood. In this case, we take that `caa_pred` we calculated at the bottom of the transformed parameters section and contrast it against `caa_obs` in a Poisson likelihood. Cool.

That's it–everything through this section is what you need to generate the model predictions and fit BERTA. The rest of the stuff commented out is either to tell you what is happening in the background or is old code to fit the model with a negative binomial likelihood or to fit a stocking survival term explicitly.

We now jump into the generated quantities section, which is where we calculate some useful values for our Kobe plots (e.g., see Figure 7 in Cahill et al. 2021).

Generated quantities block

This block is where we calculate the late period fishing mortality relative to F_{msy} and average biomass during survey years relative to our best estimate of MSY. These are equilibrium estimates, and they rely on the clever Botsford incidence calculations in box 3.1 in Walters and Martell 2004.

The first five lines of code in the generated quantities block just declare some variables, so we'll start with the algorithm that is actually doing something interesting:

```
// Fmsy, MSY subroutine
Fmsy = 0;
MSY = 0;
for(i in 1:length_Fseq){
  real sbrf = 0;
  real ypr = 0;
  real su = 1;
  real Req = 0;
  real Yeq = 0;
  for(a in 1:n_ages){
    sbrf += su*f_a[a]; // accumulate spawning biomass per recruit
    ypr += su*(1-exp(-Fseq[i]*v_f_a[a]))*w_a[a];
    su = su*exp(-M_a[a] - Fseq[i]*v_f_a[a]);
  }
  if(rec_ctl == 0){ // ricker
    Req = log( exp(ln_ar)*(sbrf) ) / (br*sbrf); // Botsford predction of Req for F[i]
  }
  if(rec_ctl == 1){ // bh
    Req = ( exp(ln_ar)*sbrf-1.0 ) / (br*sbrf);
  }
  Yeq = Req*ypr; // predicted equilibrium yield
  if(Yeq > MSY){
    MSY = Yeq;
    // jitter values to make a purdy histogram:
    Fmsy = Fseq[i] + 0.01*(uniform_rng(0,1)-0.5);
  } else {
    // Yeq for this F is lower than highest value already found,
    // so can exit the subroutine
    continue;
  }
}
```

The algorithm above looks complicated but it really is not. This algorithm does the following:

- loops across a sequence of instantaneous fishing mortality values (from low to high–see Fseq)
- calculates spawner biomass per recruit, yield per recruit, suvivorship given one Fseq (mortality rate) value
- calculates the long-term equilibrium recruitment given that age structure
- calculates predicted equilibrium yield as equilibrium recruitment $\times$ yield per recruit
- overwrites MSY and F_{MSY} if equilibrium yield is greater than MSY
- exits the routine when it finds a value that cannot be improved upon (i.e., this is what `continue` is doing)

From there, we simply record Flate/Fmsy and Average Biomass in surveyed years / MSY.

Questions

There are no questions for this–go drink much beer or ride a bike or whatever you like to do because we are done with this miserable model.

Resources:

Cahill et al. 2021. Unveiling the recovery dynamics of walleye after the invisible collapse. CJFAS.

Walters and Martell. 2004. Fisheries Ecology and Management.