Contents

1 Halibut Drafty Code

1.1 Types of Sample

- Sex
- Year caught
- Length
- May or may not have otolith age
- Otolith age because kin pair member
- DNAge (Not on table now)
- Should be non-lethal variable somewhere

Secret R tip: ?Quotes

1.2 The code we wrote

```
enum sex{
  male == 0,
  female == 1
};

enum mt{
  shared == 0,
  unshared == 1
};


template<class Type>
 get_fec(Type length, int sex){
   Type fec;
   //alpha, wexp are inputs
   Type weight = alpha[sex]*pow(weight,wexp[sex]);
   //ppnmature is an input, proportion mature at length given sex
   Type fec = pow(weight,omega[sex])*ppnmature(length,sex);
   return fec;
 }
```

```
array<Type> meanfec_sa(sexes,A);
 for(int s = 0; s < sexes; ++s){
   for(int a = 0; a < A; ++A){
     Type runningfec = 0.0;
      for(int l = 0; l < L_max; ++l){
runningfec += Pr_l_given_as(a,s)*get_fec(l,s);
      }
      meanfec_sa(s,a) = runningfec;
   }
 }


array<Type> TRO(S,Y);
for(int s = 0; s < sexes; ++s){
  for(int y = 0; y < Y; ++y){
    Type partialRO = 0.0;
    for(int a = 0; a < A; ++a){
      partialRO += N(a,y)*meanfec_sa(s,a);
    }
    TRO(s,y) = partialRO;
  }
}


for(int s1 = 0; s1 < blah.dim(s1); s1++){
  for(int y1 = 0; y1 < blah.dim(y1); y1++){
    for(int l1 = 0; l1 < blah.dim(l1); l1++){
      for(int a1 = 0; a1 < blah.dim(a1); a1++){
for(int y2 = 0; y2 < blah.dim(y2); y2++){
  for(int a2 = 0; a2 < blah.dim(a2); a2++){
    int B2 = y2-a2;
    int B1 = y1-a1;
    Type Prob;
    if(y1 < B2){
      Prob = 0;
    }
    else if(B1 >= B2){
      Prob = 0;
    }
    else{
      Type L_inf_A = l1/(1-exp(-k_[s1]*(a1-t0[s1])));
      //age at juv birth year
      int a_b2 = a1-(y1-B2);
      Type l_b2 = L_inf_A*(1-exp(-k_[s1]*(a1-t0[s1])));
      //what is get_fec and TRO?
      Prob = get_fec(s1,l_b2)/TRO(s1,B2);
    }
```

```
    Pr_MOP_SYLAYA[s1,y1,l1,a1,y2,a2] = Prob;
}
      }
    }
  }
 }

  array<Type> Pr_A_SYL(a,l,s,y);
  for(int l = 0; l < blah.dim(l); l++){
    for(int s = 0; s < blah.dim(s); s++){
      for(int y = 0; y < blah.dim(y); y++){
Type Prob_denom = 0.0;
for(int a = 0; a < blah.dim(a); a++){
  Type Prob_num = (Pr_l_given_as(a,s)*(N(a,s,y)));
  Prob_denom += Prob_num
  Pr_A_SYL(a,l,s,y) = Prob_num;
}
for(int a = 0; a < blah.dim(a); a++){
  Pr_A_SYL(a,l,s,y) /= Prob_denom;
}
      }
    }
  }



  //uncertain adult ages
  for(int s1 = 0; s1 < blah.dim(s1); s1++){
    for(int y1 = 0; y1 < blah.dim(y1); y1++){
      for(int l1 = 0; l1 < blah.dim(l1); l1++){
for(int y2 = 0; y2 < blah.dim(y2); y2++){
  for(int a2 = 0; a2 < blah.dim(a2); a2++){
    Pr_POP_running = 0.0;
    for(int a1 = 0; a1 < someblah.dim(a1); a1++){

      Pr_POP_running += Pr_MOP_SYLAYA[s1,y1,l1,a1,y2,a2]*Pr_A_SYL[a1,l1,s1,y1];
    }
    Pr_POP_sylya[s1,y1,l1,y2,a2] = Pr_POP_running;
  }
}
      }
    }
  }

  //uncertain juv ages
  for(int s1 = 0; s1 < blah.dim(s1); s1++){
    for(int y1 = 0; y1 < blah.dim(y1); y1++){
      for(int l1 = 0; l1 < blah.dim(l1); l1++){
```

```
for(int a1 = 0; a1 < blah.dim(a1); a1++){
  for(int l2 = 0; l2 < blah.dim(l2); l2++){
    for(int s2 = 0; s2 < blah.dim(s2); s2++){
      for(int y2 = 0; y2 < blah.dim(y2); y2++){
//summing over possible juv ages
Type Pr_POP_running = 0.0;
for(int a2p = 0; a2p < A; a2p++){
  Pr_POP_running += Pr_A_SYL[a2p,l2,s2,y2]*Pr_MOP_SYLAYA[s1,y1,l1,a1,y2,a2p]
    }
Pr_POP_sylalsy[s1,y1,l1,a1,l2,s2,y2] = Pr_POP_running;
      }
    }
  }
}
      }


      //HSPs

      // \(Pr[OHSP
      array<Type> Pr_OHSP_sbb(sexes,Y,Y);
      for(int sp = 0; sp < sexes; sp++){
for(int B1 = 0; B1 < Y; B1++){
  for(int B2 = B1; B2 < Y; B2++){
    //Handle same cohort case at the end
    Type Pr_HSP_running = 0.0;
    //potential ages of parent
    for(int app = 0; app < A; app++){
      Type Pr_surv = N(app+(B2-B1),sp,B2)/N(app,sp,B1);
      Type pp_ERO_B2 = 0.0;
      Type denom = 0.0;
      for(int q = 0; q < num_qs; q++){
//length_at_age_q gets length at age given quantile
Pr_pp_is_qth_q_1 = 1/num_qs*get_fec(length_at_age_q(q,app,sp),sp);
denom += Pr_pp_is_qth_q_1;
pp_ERO_B2_if_q = get_fec(length_at_age_q(a,app+(B2-B1),sp),sp);
pp_ERO_B2 += Pr_pp_is_qth_q_1*pp_ERO_B2_if_q;
      }
      pp_ERO_B2 /= denom;
      Pr_pp_is_1s_mum = (meanfec_sa(sp,app)/TRO(sp,B1));
      Pr_HSP_running += N(app,sp,B1)*Pr_pp_is_1s_mum*
Pr_surv*
pp_ERO_B2/TRO(sp,B2);

      //compensate for lucky litter (same cohort)
      if(B2 == B1){
//lucky_litter_factor is a PARAMETER_VECTOR
Pr_HSP_running *= lucky_litter_factor(sp);
```

```
      }
      Pr_OHSP_sbb[sp,B1,B2] = Pr_HSP_running;
    }
  }
}
      }
    }
  }

      // unordered HSPs (should really be done on SO_Kins)
  // Should deal with ordering of POPs
  for(int sp = 0; sp < sexes; sp++){
    for(int B1 = 0; B1 < Y; B1++){
      for(int B2 = B1; B2 < Y; B2++){
//p_false_neg is a data input from kinference or CKMRsim
Pr_HSP_sbb[sp,B2,B1] = Pr_OHSP_sbb[sp,B1,B2];
Pr_HSP_sbb[sp,B1,B2] = Pr_OHSP_sbb[sp,B1,B2];
      }
    }
  }


  array<Type> Pr_HSP_slysb(sexes,L_max,Y,sexes,Y);

  for(int sp = 0; sp < sexes; sp++){
    for(int l1 = 0; l1 < L_max; l1++){
      for(int y1 = 0; y1 < Y; y1++){
for(int s1 = 0; s1 < sexes; s1++){
  for(int B2 = 0; B2 < Y; B2++){

    //summing over possible I1 ages
    Type Pr_HSP_running = 0.0;
    for(int a1 = 0; a1 < A; a1++){
      Pr_HSP_running += Pr_A_SYL[a1,l1,s1,y1]*Pr_HSP_sbb[sp,y1-a1,B2];
}
    Pr_HSP_slysb[sp,l1,y1,s1,B2] = Pr_HSP_running;
  }
}
      }
    }
  }

  //when we don't know both hsp ages

  //grandparent grandoffspring
  array<Type> Pr_GGP_syabs(sexes,Y,A,Y,sexes);
  for(int s1 = 0; s1 < sexes; s1++){
    for(int y1 = 0; y1 < Y; y1++){
```

```
      for(int a1 = 0; a1 < A; a1++){
for(int B2 = 0; B2 < Y; B2++){
  for(int spp = 0; spp < sexes; spp++){
    int B1 = y1-a1;
    //year 1 mature
    int mat_1_y = min_age_mature(s1)+B1;
    Type Pr_GGP_running = 0.0;
    for(int app = B2-y1; app < B2-mat_1_y; app++){
      int Bpp = B2-app;
      Type Pr_pp_is_2s_mum = (meanfec_sa(spp,app)/TRO(spp,B2));
      int a1_bpp = Bpp-B1;
      Type Pr_1_is_pps_mum = (meanfec_sa(s1,a1_bpp)/TRO(s1,Bpp));
      Pr_GGP_running += Pr_pp_is_2s_mum*Pr_1_is_pps_mum;
    }
    Pr_GGP_syabs(s1,y1,a1,B2,spp) = Pr_GGP_running;
  }
}
      }
    }
  }

  array<Type> Pr_SO_kins(smtdna,sexes,Y,A,sexes,Y,A);
    for(int s1 = 0; s1 < sexes; s1++){
      for(int y1 = 0; y1 < sexes; y1++){
for(int a1 = 0; a1 < A; a1++){
  for(int s2 = 0; s2 < sexes; s2++){
    for(int y2 = 0; y2 < Y; y2++){

      for(int a2 = 0; a2 < A; a2++){
int B1 = y1 - a1;
int B2 = y2 - a2;
//shared mt case first
Type Pr_SO_kins_temp = (Pr_HSP_sbb[female,B1,B2]);
if((B2 < B1) & s2 == female){
  Pr_SO_kins_temp += Pr_GGP_syabs[female,y2,a2,B1,female];
}else if((B1 < B2) & s1 == female){
  Pr_SO_kins_temp += Pr_GGP_syabs[female,y1,a1,B2,female];
}
//shared is enum mt var
Pr_SO_kins(shared,s1,y1,a1,s2,y2,a2) = (1-p_false_neg)*Pr_SO_kins_temp;

//unshared mt case
else if((B2 < B1) & s2 == male){
  Pr_SO_kins_temp += Pr_GGP_syabs[male,y2,a2,B1,female]+Pr_GGP_syabs[male,y2,a2,B1,male];
}else if((B1 < B2) & s1 == male){
  Pr_SO_kins_temp += Pr_GGP_syabs[male,y1,a1,B2,female]+Pr_GGP_syabs[male,y1,a1,B2,male];
}
Pr_SO_kins(unshared,s1,y1,a1,s2,y2,a2) = (1-p_false_neg)*Pr_SO_kins_temp;
```

```
      }
    }
  }
}
      }
    }
  }
```

```
  // Age composition data
  for(int s = 0; s < sexes; s++){
    for(int y = 0; y < Y; y++){
      for(int l = 0; l < L_max; l++){
for(int a = 0; a < A; a++){
  //change distro
  nll -= dpois(n_otos_at_age(a,l,s,y),
   //tot number of otoliths
n_otos(l,s,y)*Pr_A_SYL[s,y,l],true);
}
      }
    }
  }
```

```
for(int y1 = 0; y1 < blah.dim(y1); y1++){
  for(int a1 = 0; a1 < blah.dim(a1); a1++){
    for(int y2 = 0; y2 < blah.dim(y2); y2++){
      for(int a2 = 0; a2 < blah.dim(a2); a2++){
for(int sp = 0; sp < sexes; sp++){
  int n_comps = n_comps_HS_yaya[y1,a1,y2,a2];
  int n_SO_kins = n_SO_kins_syaya[sp,y1,a1,y2,a2];
  nll -= dpois(n_SO_kins,n_comps*Pr_SO_kins,true);
      }
    }
  }
}
      }
```

```
for(int s1 = 0; s1 < blah.dim(s1); s1++){
  for(int y1 = 0; y1 < blah.dim(y1); y1++){
    for(int l1 = 0; l1 < blah.dim(l1); l1++){
      for(int a1 = 0; a1 < blah.dim(a1); a1++){
for(int y2 = 0; y2 < blah.dim(y2); y2++){
  for(int a2 = 0; a2 < blah.dim(a2); a2++){
```

```
      int n_comps = n_comps_POP_sylaya[s1,y1,l1,a1,y2,a2];
      int n_POPs = n_POP_sylaya[s1,y1,l1,a1,y2,a2];
      //negative log likelihood
      nll -=  dpois(n_POPs,n_comps*Pr_MOP_SYLAYA[s1,y1,l1,a1,y2,a2],true);
    }
  }
        }
      }
    }
   }

//Post-hoc

  return nll;
}
```

2 A Basic CKMR model in TMB

Here is a simple CKMR model using TMB based off what I've done to maybe help show how the peices fit together. This uses some data from one of my simulations so there's 3 age classes and sampling is done non-lethally.

This model only uses POPs so it can't estimate survival and that's assumed known and passed into the model (the `Z`)). It's also assumed to be the same every year. If HSPs were added it is possible to estimate `Z`.

```
// This is needed to actually load in the TMB library
#include <TMB.hpp>

//Your TMB objective function (the model) will always start with these two lines
template<class Type>
Type objective_function<Type>::operator() (){

  //Parameter and data setup

//The optimizer works best on -Inf to Inf.
// By passing in parameters on log space and transforming inside the model
// we keep the optimizer happy while ensuring things that should be positive are
// always postive.

  PARAMETER(log_init_tot_N);
  PARAMETER_VECTOR(log_fecundity);

  vector<Type> fecundity = exp(log_fecundity);

  DATA_INTEGER(Y);
  DATA_INTEGER(A);
  DATA_VECTOR(Z);
```

```
DATA_VECTOR(init_frac);
DATA_IMATRIX(POPs);

//Building the Numbers at age matrix
matrix<Type> log_N(A,Y);

//setting initial numbers
for(int a = 0; a < A; ++a){
  log_N(a,0) = log_init_tot_N+log(init_frac(a));
  }

//Our negative log likelihood, what we are maximizing
Type nll = 0.0;

//Population dynamics
//We need the pop dynamics to figure out the numbers at age for our expected probabilities

//Propogate the numbers at age for each year:
for(int y = 1; y < Y; ++y){
  //Recruitment, probably should be a random effect...
  //Here I just set it to be the number of offspring from last year
  Type TRO  = 0.0;
  for(int a = 0; a < A; ++a){
    TRO += fecundity(a)*exp(log_N(a,y-1));
  }
  log_N(0,y) = log(TRO);

  for(int a = 1; a < A; ++a){
    //fraction of fish that survive from last year to the current year
    log_N(a,y) = log_N(a-1,y-1)-Z(a-1);
  }
}

//The actual CKMR bit
// I do one matrix rather than a million nested for loops
// The POPs matrix contains all the data we need, like a flattened array
for(int i = 0; i < POPs.rows(); ++i){
  //pull out the correct covariates/data from each row:
  int juv_by = POPs(i,0);
  int par_by = POPs(i,1);
  int par_sy = POPs(i,3);
  //How many real POPs we found
  Type n_POPs = POPs(i,5);
  Type n_comps = POPs(i,4);

  //Finding the expected probability
  //Get the TRO:
  Type TRO = 0.0;
```

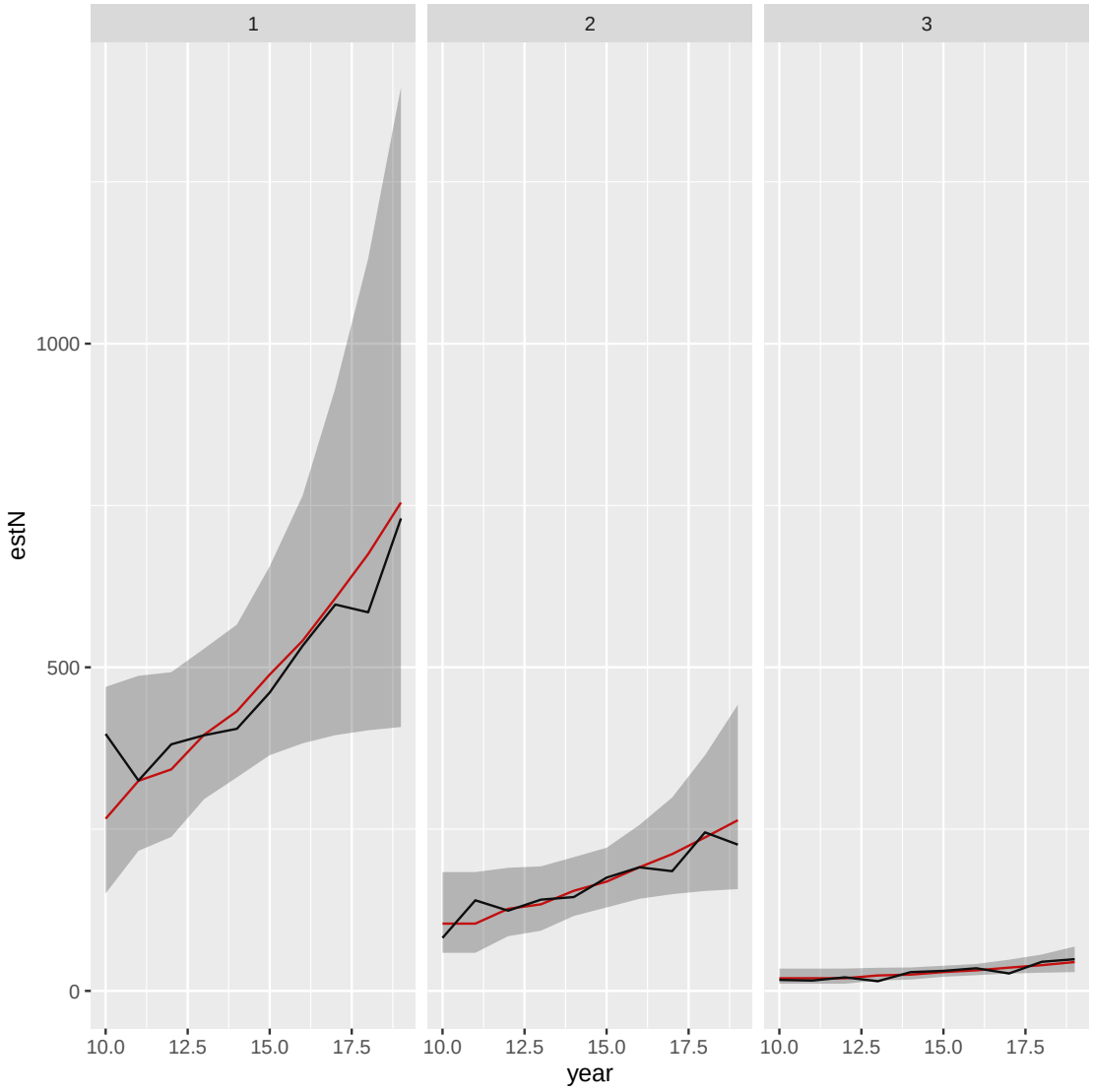
```
    for(int a = 0; a < A; ++a){
      TRO += fecundity(a)*exp(log_N(a,juv_by));
    }
    int p_age = juv_by-par_by-1; //the 1 is because C++ starts everything at 0
    Type eprob = (2.0*fecundity(p_age))/TRO;

    // Since non-lethal sampling adjust for survival of parent
    if(par_sy < juv_by){
      //parents age at sampling
      int p_age_sy = par_sy-par_by-1;
      Type Pr_surv = exp(log_N(p_age,juv_by)-log_N(p_age_sy,par_sy));
      eprob *= Pr_surv;
    }

    //Add the contribution to the likelihood
    nll -= dpois(n_POPs,n_comps*eprob,true);
  }

  matrix<Type> N(A,Y);
  N = log_N.array().exp();


  //Get anything we want out of the model and back into R
  REPORT(N);
  REPORT(fecundity);
  //Get the standard error for that quanity:
  ADREPORT(log_N);
  ADREPORT(log_fecundity);

  return nll;

}
```

2.1 Running this model from R

```
library(TMB)

##load in the required data for the model, an object called stuff

load("basicCKMRdat.Rdata")

##Compile the model
compile("basicCKMR.cpp")

##Load the compiled model
dyn.load(dynlib("basicCKMR"))
```

```
##annoying bit because there isn't enough data to freely estimate the initial numbers at age
##It's basically the fraction of the numbers at age in the population
##Can be done inside with Z if estimated but easier here since Z isn't
init_frac = c(1,exp(-stuff$Z[1]),exp(-stuff$Z[1]-stuff$Z[2]))
init_frac = init_frac/sum(init_frac)

##set up data and parameters to give to TMB model, must be as named lists
## matching what's in the model
dat = list()
dat$Y = 10
dat$A = 3
dat$Z = c(0.9395,1.670,0)
dat$init_frac = init_frac
##The actual important data, number of POPs found and number of comparisons
dat$POPs = as.matrix(stuff$POPs)

##Our initial guess for parameters
parm = list()
parm$log_init_tot_N = log(500)
parm$log_fecundity = log(c(1.1,1.1,1.1))

##Make the objective function with TMB
obj = MakeADFun(dat,parm)

##optimize the model to get the most likely parameters given the data
opt = nlminb(obj$par,obj$fn,obj$gr)

##get the data out from the model, make confidence intervals
report = obj$report()
sdr = sdreport(obj)
logN = as.data.frame(summary(sdr)[which(rownames(summary(sdr)) == "log_N"),])
logN$year = sort(rep(10:19,3))
logN$age = rep(1:3,10)
logN$lower = exp(logN[,1] - 1.96*logN[,2])
logN$upper = exp(logN[,1] + 1.96*logN[,2])
logN$estN = exp(logN[,1])

logFec = as.data.frame(summary(sdr)[which(rownames(summary(sdr)) == "log_fecundity"),])
logFec = logFec[1:3,]
logFec$fec = exp(logFec[,1])
logFec$lower = exp(logFec[,1] - 1.96*logFec[,2])
logFec$upper = exp(logFec[,1] + 1.96*logFec[,2])


##compare with true N and fecundity
trueN = stuff$trueN
logN$trueN = as.vector(trueN)
trueFec = stuff$trueFec
```

```
library(ggplot2)
Nplot = ggplot(logN) + geom_line(aes(x=year,y=estN),color="red") + geom_line(aes(x=year,y=trueN)
```



The red line is the estimate from the model, black is simulation and the ribbon is the 95% CI. The model gets the abundance back well over the years.

Note that the fecundity for age 3 estimate is too small and it still lies outside the 95% CI. This isn't a big sample from a small population and age 3's are particularly rare (there's about 1 or 2 a year in the sample). I also didn't place any constraints on fecundity other than it being positive.