St. Pierre bank assessment model

Jonathan Babyn

March 24, 2019

Contents

1 Introduction

2 Model Design

2.1 Process Equations

The St. Pierre bank Assessment Model (SPAM) is a state-space stock assessment model implemented in Template Model Builder (TMB). As a State Space Model (SSM) the model can be separated into process equations and observation equations. SPAM follows the same general process equations design as Nielsen and Berg[9], with the log abundance at age a in a given year y, $\log(N_{y,a})$ as

$$\log(N_{y,R}) = \log(\text{SR})) + \eta_{y,R}, \quad \eta_{y,R} \sim \mathcal{N}(0, \sigma_R^2) \quad R < a, \tag{1}$$

$$\log(N_{y,a}) = \log(N_{y-1,a-1}\exp(-Z_{y-1,a-1})) + \eta_{y,a}, \quad \eta_{y,a} \sim \mathcal{N}(0, \sigma_S^2) \quad R < a < A \quad \text{and} \tag{2}$$

$$\log(N_{y,A}) = \log(N_{y-1,a-1}\exp(-Z_{y-1,a-1}) + N_{y-1,A}\exp(-Z_{A,y-1})) + \eta_{y,A} \quad \eta_{y,A} \sim \mathcal{N}(0, \sigma_S^2) \tag{3}$$

with $Z_{y,a} = F_{y,a} + M_{y,a}$, where $F_{y,a}$ is mortality from fishing and natural mortaility $M_{y,a}$. Natural mortality was taken to be a fixed constant.

Equation 1 is the process of recruitment of cod to the fishery, R is the first age included in the model and SR is some form of stock recruitment. SPAM implements four different options for stock recruitment,

$$\text{Random Walk: SR} = N_{y,R} = N_{y-1,R} + \eta_{y,R} \tag{4}$$

$$\text{Ricker: SR} = \alpha\text{SSB}\exp(-\beta\text{SSB}) \tag{5}$$

$$\text{Beverton-Holt: SR} = \frac{\alpha\text{SSB}}{1+\beta(SSB)} \quad \text{and} \tag{6}$$

$$\text{Smooth HS: SR} = \alpha\left(\text{SSB} + \sqrt{\delta^2 + \frac{\gamma^2}{4}} - \sqrt{(\text{SSB} - \delta)^2 + \frac{\gamma^2}{4}}\right) \tag{7}$$

where SSB is Spawning Stock Biomass (SSB) and γ^2 is a fixed parameter that controls the smoothness of the breakpoint and was set equal to 0.1. Equations 2 & 3 are the process of survival, and Equation 3 is an extension for a plus group of cod aged A or older. The entire $\boldsymbol{N}$ abundance matrix is treated as random parameters to be integrated out of the model using the Laplace Approximation, this has the benefit of making adding internal stock recruitment models very simple as the calculation of SSB can be done at any point in the model, as the values of $\boldsymbol{N}$ will be found during optimization. It also greatly reduces model run-time and the number of fixed effect parameters to estimate. Modelling survival as process error also has the benefit of allowing for immigration of fish into the stock of interest, which is something that may be relevant to 3Ps cod as stock mixing has been a concern[8][11].

Fishing mortality was also implemented the same way as in Nielsen & Berg[9]. For a given year of fishing mortality $\boldsymbol{F}_y = (F_{y,1}, \ldots, F_{y,A})'$ is a Multivariate Normal (MVN) random walk,

$$\log(\boldsymbol{F}_y) = \log(\boldsymbol{F}_{y-1}) + \xi_y, \quad \xi_y \sim MVN(\boldsymbol{0}, \boldsymbol{\Sigma_F}) \tag{8}$$

, with $\boldsymbol{\Sigma_F}$ being the covariance matrix. This implementation allows for time varying selectivity, with selectivity being defined as $S_{y,a} = \frac{F_{y,a}}{\sum_a F_{a,y}}$[9]. Since the 3Ps cod fishery has seen a shift in gear types over time, most notably a move away from otter trawls and cod traps to gillnet, time varying selectivity can help deal with this. As in Nielsen and Berg[9], the same four covariance options were implemented. A fifth covariance option was also added which allows for a different

Name	Off-diagonal elements
Independent	$\mathbf{\Sigma}_{a,\tilde{a}} = 0$
Parallel	$\mathbf{\Sigma}_{a,\tilde{a}} = \sigma_a \sigma_{\tilde{a}}$
Compound	$\mathbf{\Sigma}_{a,\tilde{a}} = \rho \sigma_a \sigma_{\tilde{a}}$
AR(1)	$\mathbf{\Sigma}_{a,\tilde{a}} = \rho^{\lvert a-\tilde{a} \rvert} \sigma_a \sigma_{\tilde{a}}$
Custom	$\mathbf{\Sigma}_{a,\tilde{a}} = \rho_a \rho_{\tilde{a}} \sigma_a \sigma_{\tilde{a}}$

Table 1: Covariance matrix options available in SPAM for ages a, $a \neq \tilde{a}$, independent is uncorrelated between ages and the process will develop independently in time, parallel is like $\rho = 1$ and is the same as the assumption of constant selectivity, compound has all the age groups develop correlated in time the same, AR(1) has all the processes be correlated as function of the distance between ages, so closer ages are more closely related. Custom allows having ages have their own correlation parameter.

correlation ρ_a for each age. This was mainly done with the intention of making fishing mortality in age 2 cod uncorrelated with older fish, e.g. $\rho_2 = 0$. Table 1 illustrates the different covariance matrix options available in SPAM.

2.2 Observation Equations

SPAM incorporates data from multiple different research vessel surveys, catch at age proportions and reported landings. For the survey indices $I_{y,a,s}$, which are reported as Mean Numbers Per Tow (MNPT), the value predicted by the model is

$$\log(\hat{I}_{y,a,s}) = \log(q_{a,s}) + \log(N_{y,a}) - t_{s,y} Z_{y,a} \tag{9}$$

where $q_{a,s}$ is the catch-ability parameter for survey s for age a, and $t_{s,y}$ is the median date of the survey during the year in decimal form. As in the Northern Cod Assessment Model (NCAM) currently used for assessment for northern cod, SPAM sets a survey detection limit of 0.0005, with the belief that the cod are there, just in not large enough numbers to be detected by the survey[2]. This says that the data is left censored. $I_{y,a,s} > 0.0005$ are added to the log-likelihood as

$$\sum_a \sum_y \log\left(\varphi\left(\frac{\log(I_{y,a,s}) - \log(\hat{I}_{y,a,s}])}{\sigma_{s,I}}\right)\right) \tag{10}$$

where φ is the PDF of the standard normal distribution and $\sigma_{s,I}$ is the standard deviation of the survey index for survey s. Indices that fall below the detection limit are added to the likelihood as

$$\sum_a \sum_y \log\left(\Phi\left(\frac{\log(I_{y,a,s}) - \log(\hat{I}_{y,a,s})}{\sigma_{s,I}}\right)\right) \tag{11}$$

with Φ being the Cumulative Distribution Function (CDF) of the standard normal distribution.

SPAM calculates the expected catch C in the model using the Baranov catch equation,

$$\hat{C}_{y,a} = N_{y,a}(1 - \exp(-Z_{y,a}))\frac{F_{y,a}}{Z_{y,a}}. \tag{12}$$

It does not fit directly to $\hat{C}$ but instead to the expected Continuation Ratio Logit (CRL) of the catch proportions at age for a given year, $\hat{\boldsymbol{X}}_y$. CRL are defined as $P(Y = a | a \geq a) = \text{logit}(\pi_a)$

where $\pi_a = \frac{p_a}{p_a + \cdots + p_A}$ and p_a is proportion of catch at age a. Using CRLs allows for fitting on catch proportions but does not require knowing or assuming the sampling sizes used to determine the catch proportions. It does require having non-zero catch proportions and small values were added to the observed zero catch proportions, and these were used to create the observed CRL, $\boldsymbol{X}_y$. The observed CRL are assumed be distributed as

$$\boldsymbol{X}_y \sim MVN(\hat{\boldsymbol{X}}_y, \boldsymbol{\Sigma}_{CRL}) \tag{13}$$

where $\boldsymbol{\Sigma}_{CRL}$ can be any one of the same covariance matrix options presented in Table 1 allowing for a correlation structure between ages.

Since SPAM fits catch proportions instead of catch at age, a third element is needed to fix the the magnitudes of abundance. The reported landings that were provided were used to do this. This was done in a similar fashion to how it is outlined in the NCAM paper, except instead of fitting to expected total catch for the year versus the observed total catch, this was instead done on the expected landings versus the observed landings. This was done since earlier catch at age calculations have been speculated to not include fish older than age 14 rather than their just be zero catch at those ages, whereas reported landings would and thus total observed catch would be under-reporting. The landings are added to the log-likelihood in the model as

$$\sum_y \log\left\{\Phi\left(\frac{\log(L_y) + \log(UB) - \log(\hat{L}_y)}{\sigma_L}\right) - \Phi\left(\frac{\log(L_y) + \log(LB) - \log(\hat{L}_y)}{\sigma_L}\right)\right\} \tag{14}$$

where L_y and $\hat{L_y}$ are the observed and predicted landings respectively, UB and LB are upper and lower bound multipliers on what the reported landings actually were. This is similar to Equation 7 of the NCAM paper. There it was noted that there can be issues optimizing the likelihood with this sort of formulation and there a mixture distribution and judicious choices of starting parameters were used to overcome them[2]. I instead opted to fix the problem directly by writing atomic functions for TMB that manually defines the derivatives for Equations 11 & 14 in a more numerically stable manner. This process is described in Appendix A. This allows for SPAM to be run with a wide variety of starting values and without the need for upper or lower bounds on parameters or the need to run the model twice to find reasonable starting values. For the same reason outlined in Cadigan (2016)[2], σ_L is fixed to be $\log(0.02)$ to prevent $\hat{L}_y$ from escaping the bounds too often or exceeding them by too much.

The parameters estimated, including optional parameters by SPAM are noted in Table 2.

Fixed Parameters	Description
σ_R	Std. Dev. of Recruitment Process
σ_S	Std. Dev. of Survival Process
$\boldsymbol{\sigma}_{s,I}$	Vector of Std. Dev. of survey indices
$\boldsymbol{\sigma}_F$	Vector of Std. Dev. for fishing mortality
$\rho_F(\boldsymbol{\rho}_F)$	Correlation parameter for $\boldsymbol{\Sigma}_F$(Vector if custom type,0 if independent)
$\boldsymbol{\sigma}_{CRL}$	Vector of Std. Dev. for CRLs.
$\rho_{CRL}(\boldsymbol{\rho}_{CRL})$	Correlation parameter for $\boldsymbol{\Sigma}_{CRL}$(Vector if custom type, 0 if independent)
$\boldsymbol{q}$	Vector of catchability parameters for the survey indices
α	Optional parameter used in the three stock recruitment functions
β	Optional parameter used in the Ricker and Beverton-Holt stock recruitment functions
δ	Optional parameter used in the smooth hockey stick recruitment function
Random Parameters	
$\boldsymbol{N}$	Matrix of Abundance
$\boldsymbol{F}$	Matrix of Fishing Mortality

Table 2: Mandatory and optional fixed effect parameters estimated in SPAM followed by random parameters estimated.

3 Data Inputs

In addition to the DFO Offshore and Inshore and Offshore survey indices that were provided, I also took the RV survey indices that were the result of the Groundfish Enterprise Allocation Council (GEAC) RV survey performed in 3Ps. GEAC performed their survey in 3Ps from 1997 to 2007, with 2006 excluded and their data was given to DFO to be analyzed in the same manner as DFO's RV survey. These survey indices were taken from the DFO research documents in [7] & [6]. The GEAC indices were included in past 3Ps cod assessments, but have not been included recently, in addition the 1997 survey indices were not used in those assessments due to a much smaller smaller area being surveyed than other years[1]. Due to this I also elected not to use the 1997 GEAC survey indices and I also chose not to use the 2007 GEAC survey indices since they expanded the survey extent and used an entirely different set of equipment than previous years. Juliette Champagnat provided the survey indices from the French RV survey that was performed in 3Ps from 1978 to 1992.

Due to the approach I opted to take with the landings, upper and lower bounds on the landings are required. I consulted with Danny Ings, head of 3Ps cod stock stock assessment at DFO, he provided a set of periods with a general idea of how they relate to each other in width, but not exact bounds. These periods, along with the selection of landings bounds that were tried can be seen in Table 3.

As previously mentioned, since the raw catch proportions used to determine catch at age are not available, nor their sample sizes, the catch at age was converted to be catch proportions at age then to the catch CRLs used in the model. Ideally the raw catch proportions would be provided and their sample sizes known which would allow for other possibly better options of representing compositional data.

3.1 Smoothing Maturities and Weights

Two important pieces for calculating SSB are the weights used and maturity ogives. The maturity ogives provided are the output of a logistic regression run and updated each year from fish that have had their otoliths aged, lengths measured and maturity status determined. This results in very noisy maturity data set for each age group. For both the commercial and stock weights, output

Period	Danny Ings' Description	Tight	Medium	Wide
1959-1993	Somewhat wide	(0.75,1.25)	(0.5,1.5)	(0.35,1.75)
1994-1996	Narrow	(0.95,1.05)	(0.90,1.10)	(0.80,1.20)
1997-1998(or 1997-1999)	Very Wide	(0.5,1.5)	(0.35,1.75)	(0.25,2.0)
1999-present(or 2000-present)	Narrower than 1959-1993 but wider than 1994-96	(0.85,1.15)	(0.75,1.25)	(0.5,1.5)

Table 3: The periods for landing bounds provided by Danny Ings, along with the description of how they relate and three sets of different upper and lower bound multipliers to be tested on SPAM.

from a cohort model going from 1983 to 2016 were provided, along with commercial and stock weights from 1959 to 2014 and 1959 to 2017 respectively provided by DFO. The weights provided by DFO are noisy, and values prior to 1977 are identical and likely copy and pasted from some mean weights derived at some point. Neither of these situations are ideal. The weights either need to be extended thirty back into the past or improved. While the DFO provided weights unfortunately do not have real observations prior to 1977, the values provided at least seem consistent when compared with the output of the cohort model weights. I chose to try and deal with the noise in the DFO provided weights rather than extend the other weights back as the DFO weights are based on actual commercial sampling efforts rather than being based off research vessel survey data and weight length relationships.

Both the maturities and weights were smoothed using Functional Principal Component Analysis (FPCA). FPCA is a non-parametric technique that has been used to perform noise reduction on curve like data[10]. FPCA has at least been at least applied to fisheries data in the context of evaluating spatio-temporal patterns in the data[4] but it has not to the author's knowledge been applied to fisheries weight or growth data. The noise reduction is done by breaking the data down into it's mean function $\mu(t)$, eigenfunctions $\phi_k(t)$ and Functional Principal Components (FPC), ξ_{ki} and keeping only the number of FPCs that explain most of the variation in the data. Further details on FPCA and this process can be seen in Appendix B.

For the maturities, the logit was taken and the FPCA performed in logit space to ensure that maturity proportions would continue to lie between 0 and 1 after smoothing. The FPCA was performed with each age group being considered an observation measured at each year. Smoothing was allowed on the mean function and a user selected bandwidth of 14 was chosen which removed many of the peaks and valleys. The diagnostic plots of the FPCA performed on the maturities showcasing the mean function, first three eigenfunctions, fraction of variance explained by each FPC are seen in Figure **??**. Over 90% of the variance is explained by the first FPC, because of this only the first FPC was used in creating the smoothed data. Plots comparing the smoothed and raw versions of the maturities by age and year are in Figure **??**.

For the weights the years were treated as observations and the age groups as measurement points. Here the mean function was automatically smoothed using Generalized Cross Validation (GCV). The diagnostic plots for the FPCA on the commercial weights are in Figure **??**. As the commerical weights represent mid-year weights, the points were placed on a grid at half year intervals, e.g. the weights for fish that are aged 3.5 years.

Appendices

A Numerically stable logged normal cumulative distribution function derivatives in TMB

`TMB` uses Automatic Differentation (AD) to generate the first and second derivatives which are used to help with both performing MLE for parameters and integrating random effects out of the likelihood with the Laplace Approximation (LA). `TMB` accomplishes this with a mixture of forward and reverse AD accumulation modes. Further details on AD can be seen in Fournier et al[5]. `TMB` automatically tries to determine the derivatives of the user's template. However, it can not automatically determine the most numerically stable form of the derivative which can often be necessary when working with extremely small values that can commonly occur when working with probabilities. `TMB` also includes the ability to manually define the forward and reverse modes for a given function to overcome this limitation using atomic functions. By default `TMB` includes a built in atomic version of the normal CDF, `pnorm` based on the version in `R`'s `C` language math library. This built-in version does not support the ability to return a logged value of the probability and shares the same limitation of returning 0 or 1 for negative or positive values of standard normal Z scores when $|Z| >\sim 38$ due to floating point limitations. This clearly means that for any $Z < -38$, `TMB` will return a `NaN` in the gradient when trying to perform `log(pnorm(Z))` due to log(0) being negative infinity. In practice however this is actually much worse since the derivative of `log(pnorm(Z))` is

$$\frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}\left(\frac{\text{erf}\left(\frac{z}{\sqrt{2}}\right)}{2}+\frac{1}{2}\right)} \tag{15}$$

and easily results in the division of two extremely small numbers. With floating point limitations this quickly becomes undefined if not handled correctly.

I created an atomic function to specifically deal with the problem of numerical stability when trying to do `log(pnorm(Z))` and similar calculations in the objective function of SPAM such as those done when using a censored likelihood for the detection limit for fitting the observation equations for the survey indices. This requires specifying more numerically stable versions for both the forward and reverse AD modes. For the forward mode this is just the `double C++` function giving the log of the normal CDF and all that requires is calling the version of `pnorm` in `R`'s `C` math library with the `give.log` flag turned on. For the reverse mode just using Equation 15 is not numerically stable due to the division. It can easily be seen that Equation 15 is equivalent to

$$\exp\left(\log\left(\frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}\right)-\log\left(\frac{\text{erf}\left(\frac{z}{\sqrt{2}}\right)}{2}+\frac{1}{2}\right)\right). \tag{16}$$

Working in log space is much less likely to result in under or overflow of floating point values when performing division of small numbers. Using numerically stable versions of the logged normal PDF & CDFs and the form in Equation 16 results in a numerically stable reverse mode derivative for `log(pnorm(Z))`. The atomic function was wrapped in the function `pnorm4` and made available for others to use in the `C++` header file `pnorm4.hpp`. Using `pnorm4(Z)` in place of `log(pnorm(Z))` allows for running one-sided censor likelihood bounds without the need for using two runs of optimization.

It's accuracy was checked by comparing the results of the gradient and hessian generated by `TMB` for `pnorm4` against the numerical first and second derivatives of `pnorm4` done by the `R` software

package `numDeriv` along with the brute force first and second derivatives of the normal log(CDF) calculated by the open-source computer algebra system `Maxima` with 5000 digits of floating point precision.

A.1 Extending stability to censored bounds

Censored likelihood bounds pose a similar problem to the above. When using the built-in functions to add normally distributed censored log-likelihood bounds this can be done like `log(pnorm(ZU))-log(pnorm(ZU))` where `ZU` & `ZL` are Z scores of the upper and lower bounds respectively. Just replacing the calls to `pnorm` with the more stable `pnorm4` and using the brute force approach or `logspace_sub` to perform the subtraction is not stable either. This is because for $Z > 38$ `pnorm` will return 1, or 0 for the logged version which will then result in trying to take the log of 0, again leading to `NaN` in the gradient.

The normal CDF of the upper and lower bounds can be thought of as $\Phi(ZU) = 1 - \epsilon_u$ and $\Phi(ZL) = 1 - \epsilon_l$ where the $0 < \epsilon < 1$ and the censored log-likelihood bound is

$$\log(\Phi(ZU) - \Phi(ZL)) = \log(1 - \epsilon_u - (1 - \epsilon_l)) = \log(\epsilon_l - \epsilon_u). \tag{17}$$

It can also be seen that

$$\log(\Phi(-ZL) - \Phi(-ZU)) = \log(\epsilon_l - \epsilon_u) \quad \text{since} \quad \Phi(-Z) = 1 - \Phi(Z) = 1 - (1 - \epsilon) = \epsilon. \tag{18}$$

Since the problem of `NaN`s in the gradient is occurs because the relatively large value of 1 overwhelms the incredibly small values of ϵ for large positive values of Z again due to floating point limitations. This was solved by using equation 18 for positive values of Z instead but with probabilities returned in logged form and the subtraction done with `logspace_sub`. This prevents the small ϵs from disappearing in the subtraction.

Then a stable reverse mode similar to the one for `pnorm4` was made, and this was wrapped in the function `censored_bounds`, checked for accuracy the same way and again made available for everyone to use in `pnorm4.hpp`.

B Functional Principal Component Analysis

Functional Data Analysis (FDA) is a branch of statistics that deals with data that takes values in an infinite dimensional or functional space. Growth and maturity curves are one-dimensional examples of functional data. Several classical statistical techniques have been extended to work with functional data, Principal Component Analysis (PCA) being among them. In Principal Component Analysis (PCA) the goal is to perform dimension reduction while maximizing the variation explained by each dimension, Functional Principal Component Analysis (FPCA) extends this to functional data. Let X be a random function defined on the function grid $[0, T]$ with unknown smooth mean function

$$E[X(t)] = \mu(t) \quad t \in [0, T], \tag{19}$$

and covariance function

$$\text{Cov}(X(s), X(t)) = G(s, t) \quad s, t \in [0, T]. \tag{20}$$

$G(s, t)$ can be written in it's orthogonal expansion as

$$G(s, t) = \sum_{k=1}^{\infty} \lambda_k \phi_k(s) \phi_k(t) \tag{21}$$

where λ_k are the eigenvalues, and ϕ_k are eigenfunctions that form an othonormal basis with a unit norm. Using the orthogonal expansion allows re-writing each functional observation $X_i(t)$ in the Karhunen-Loève representation

$$X_i(t) = \mu(t) + \sum_{k=1}^{\infty} \xi_{ik}\phi_k(t) \quad \text{where} \quad \xi_{ik} = \int (X_i(t) - \mu(t))\phi_k(t)dt. \tag{22}$$

The ξ_{ik} are the Functional Principal Components (FPC)[3]. Like PCA Principal Components, FPCs can also be clustered using clustering algorithms to find groups in the data.

The Karhunen-Loève representation enables a few different things. When using estimated forms of the mean function, FPCs, and eigenfunctions which can be found a number of ways such as numerical integration, using a limited number of FPCs that explain most of the variation in the data can be used to generate noise reduced versions of the data while keeping the parts of the process that explain the most variability[10]. If L is the number of FPC that explain for example 90% of the variation in the data, then noise reduced observations can be made by

$$\hat{X}_i(t) = \hat{\mu}(t) + \sum_{k=1}^{L} \hat{\xi}_{ik}\hat{\phi}_k(t). \tag{23}$$

This same method can be used to impute missing parts of the function as well[3].

References

[1] John Brattey et al. *Assessment of the cod (Gadus morhua) stock in NAFO Subdiv. 3Ps in October 2005*. Fisheries & Oceans Canada, Science, Canadian Science Advisory Secretariat, 2005.

[2] N. G. Cadigan. "A state-space stock assessment model for northern cod, including under-reported catches and variable natural mortality rates". *Canadian Journal of Fisheries and Aquatic Sciences* 73 (2016).

[3] Jeng-Min Chiou et al. "A functional data approach to missing value imputation and outlier detection for traffic flow data". *Transportmetrica B: Transport Dynamics* 2.2 (Mar. 2014), pp. 106–129. ISSN: 2168-0582. DOI: 10.1080/21680566.2014.892847. URL: http://dx.doi.org/10.1080/21680566.2014.892847.

[4] Clare B. Embling et al. "Investigating fine-scale spatio-temporal predator-prey patterns in dynamic marine ecosystems: a functional data analysis approach". *Journal of Applied Ecology* 49 (2012), pp. 481–492.

[5] David A. Fournier et al. "AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models". *Optimization Methods and Software* 27.2 (Apr. 2012), pp. 233–249. ISSN: 1029-4937. DOI: 10.1080/10556788.2011.597854. URL: http://dx.doi.org/10.1080/10556788.2011.597854.

[6] John McClintock. *The Fall 2007 NAFO Subdivision 3Ps GEAC Survey: Catch Results for Atlantic Cod (Gadus Morhua), American Plaice (Hippoglossoides Platessoides F.), Witch Flounder (Glyptosephalus Synoglossus L.), and Haddock (Melanogrammus Aeglefinus)*. Canadian Science Advisory Secretariat= Secrétariat canadien de consultation ..., 2011.

[7] John McClintock. *Year Eight of the NAFO Subdivision 3Ps Fall GEAC Surveys: Catch Results for Atlantic Cod (Gadus morhua), American Plaice (Hippoglossoides platessoides F.), Witch Flounder (Glyptocephalus cynoglossus L.), and Haddock (Melanogrammus aeglefinus)*. Canadian Science Advisory Secretariat= Secrétariat canadien de consultation . . ., 2011.

[8] Red Méthot et al. "Spatio-temporal distribution of spawning and stock mixing of Atlantic cod from the northern Gulf of St. Lawrence and southern Newfoundland stocks on Burgeo Bank as revealed by maturity and trace elements of otoliths". *Journal of Northwest Atlantic Fishery Science* 36 (2005), pp. 31–42.

[9] Anders Nielsen and Casper W. Berg. "Estimation of time-varying selectivity in stock assessments using state-space models". *Fisheries Research* 158 (2014).

[10] James O Ramsay and Bernard W Silverman. *Applied functional data analysis: methods and case studies*. Springer, 2007.

[11] R.M. Rideout et al. *Assessing the status of the cod (Gadus morhua) stock in NAFO Subdivision 3Ps in 2016*. Fisheries and Oceans Canada, Science, Newfoundland & Labrador Region, 2017.

Appendices

C C++ Code used

C.1 fit.cpp (Model Main)

```
#include <TMB.hpp>
#include "pnorm4.hpp"
#include "Fcorr.hpp"
#include "MVMIX.hpp"

//Convert a matrix to it's proportions by row
template<class Type>
matrix<Type>  propM(matrix <Type > M){
  int A = M.cols();
  int Y = M.rows();

  matrix<Type> retM(Y,A);

  for(int y = 0;y < Y;y++){
    vector<Type> cy = M.row(y);
    Type tot = cy.sum();
    vector<Type> prop = cy/tot;
    retM.row(y) = prop;
  }

  return retM;
}
```

```
/*Calculate population weighted average F for some set of ages
 bAge is the beginning age to calculate average F over, eAge is the
    end Age.*/
template<class Type>
vector<Type> aveF(matrix<Type> F, matrix<Type> N, int bAge,int
   eAge){
  int Y = F.rows();
  int dist = (eAge-bAge)+1;
  matrix<Type> aveF = F.array()*N.array();
  //Get the block of the matrix starting at year 0 going to year Y,
     column of bAge extending the number of columns between eAge
     and bAge and take the sum rowwise.
  vector<Type> aveF_XY = aveF.block(0,bAge,Y,dist).rowwise().sum();
  vector<Type> tn = N.block(0,bAge,Y,dist).rowwise().sum();
  aveF_XY = aveF_XY/tn;
  return aveF_XY;

}


/*Calculate CRLs from catch proportions*/
template<class Type>
matrix<Type> makeCRLs(matrix <Type > cProps){
  int A = cProps.cols();
  int Y = cProps.rows();

  matrix<Type> crls(Y,A-1);

  for(int a = 0; a < A-1;a++){
    for(int y = 0; y < Y;y++){
      vector<Type> cprow = cProps.row(y);
      //get p_a+...+p_A
      Type denom = cprow.segment(a,A-a).sum();
      Type num = cProps(y,a);
      Type pi = num/denom;
      crls(y,a) = log(pi/(1-pi));
    }
  }

  return crls;
}

/*Beverton-Holt Recruitment function*/
template<class Type>
Type bHolt(Type SSBly,vector <Type> bhParm){
  Type ret = bhParm(0) + log(SSBly) -log(1.0+exp(bhParm(1))*SSBly);
  return ret;
}
```

```
/*Ricker recruitment function*/
template<class Type>
Type ricker(Type SSBly,vector <Type> rickParm){
  Type ret = rickParm(0) + log(SSBly) - exp(rickParm(1))*SSBly;
  return ret;
}

//' Smooth Hockey Stick Recruitment Function
template<class Type>
Type sHS(Type SSBly,vector <Type> segParm,Type gammaSq){
  //Alpha and delta must be postive.
  Type delta = exp(segParm(1));
  Type alpha = exp(segParm(0));

  Type gm = gammaSq/4.0;
  Type sq1 = sqrt(delta*delta + gm);
  Type sq2 = sqrt((SSBly-delta)*(SSBly-delta) +gm);
  Type ret = log(alpha*(SSBly+sq1-sq2));
  return ret;
}


template<class Type>
Type objective_function<Type>::operator() ()
{
  enum fleetType {
    survey = 0,
    Catch = 1,
    land = 2
  };

  enum recType {
    rw = 0,
    bh = 1,
    rick = 2,
    smoothHS = 3
  };


  // input data;
  DATA_MATRIX(M);
  DATA_MATRIX(weight);
  DATA_MATRIX(mat);
  DATA_MATRIX(midy_weight);
  DATA_VECTOR(log_index);
  DATA_IVECTOR(i_zero); //A lot of this could be put in one IMATRIX?
  DATA_IVECTOR(iyear);
```

```
DATA_IVECTOR(iage);
DATA_IVECTOR(isurvey);
DATA_IVECTOR(iq);
DATA_VECTOR(fs);
DATA_IVECTOR(ft);
DATA_VECTOR(lowerMult); //Lower multiplier for censored bounds
DATA_VECTOR(upperMult); //Upper multiplier for censored bounds
DATA_IVECTOR(keyF);
DATA_INTEGER(recflag);
DATA_INTEGER(corflag);
DATA_INTEGER(corflagCRL);
DATA_SCALAR(gammaSq); //For smoothHS recruitment function,
   controls smoothness of breakpoint
DATA_VECTOR(crlVec); //Vector of Continuation Ratio Logits for
   catch
DATA_IVECTOR(aveFages); //Ages in model ages to calculate average
   F over...0 bAge, 1 for eAge


DATA_VECTOR_INDICATOR(keep,log_index); //Used for one step
   predict, does not need to be read in from R
DATA_VECTOR_INDICATOR(keepCRL,crlVec);

int n = log_index.size();

int A = mat.cols();
int Y = mat.rows();
vector<Type> index = exp(log_index);

//define parameters;
PARAMETER(log_std_log_R);
PARAMETER_VECTOR(log_qparm);
PARAMETER_VECTOR(log_std_index);
PARAMETER_VECTOR(log_std_logF);
PARAMETER_VECTOR(log_std_CRL);
PARAMETER_VECTOR(log_std_landings); //std. dev for landings, size
   0 if fit_land=0, else 1
PARAMETER(log_sdS);
PARAMETER_VECTOR(rec_parm);
PARAMETER_VECTOR(tRhoF);
PARAMETER_VECTOR(tRhoCRL);

//Random Effects
PARAMETER_MATRIX(log_F);
PARAMETER_MATRIX(log_N);

Type std_log_R = exp(log_std_log_R);
vector<Type> std_index = exp(log_std_index);
```

```
vector<Type> std_logF = exp(log_std_logF);
vector<Type> std_CRL = exp(log_std_CRL);
vector<Type> std_landings = exp(log_std_landings);
Type sdS = exp(log_sdS);

matrix<Type> F(Y,A);
matrix<Type> Z(Y,A);
matrix<Type> EC(Y,A);
matrix<Type> ECW(Y,A);

vector<Type> Elog_index(n);
vector<Type> std_index_vec(n);

//**********  start the engine ***************;

using namespace density;
Type nll = 0.0;

//compute F & Z
for(int a = 0;a < A;++a){
  for(int y = 0;y < Y;++y){
    //keyF really cuts down on number of random parameters
    F(y,a) = exp(log_F(y,keyF(a)));
    Z(y,a) = F(y,a) + M(y,a);
  }
}

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

//Because of the magic from making log_N a parameter you can
   calculate SSB whenever you want.
matrix<Type> B_matrix = weight.array()*N.array();
matrix<Type> SSB_matrix = mat.array()*B_matrix.array();
vector<Type> biomass = B_matrix.rowwise().sum();
vector<Type> ssb = SSB_matrix.rowwise().sum();
vector<Type> log_biomass = log(biomass);
vector<Type> log_ssb = log(ssb);



//Recruitment, adding recruitment curves much easier because ssb
   is already available and will be optimized later
Type predN;
for(int y =1;y < Y;y++){
  switch(recflag){
  case rw:
    predN = log_N(y-1,0);
    break;
```

```
  case bh:
    predN = bHolt(ssb(y),rec_parm);
    break;
  case rick:
    predN = ricker(ssb(y),rec_parm);
    break;
  case smoothHS:
    predN = sHS(ssb(y),rec_parm,gammaSq);
    break;
  default:
    error("Not right rec type");
    break;
  }

  nll -= dnorm(log_N(y,0),predN,std_log_R,true);
}


//Fill in N, add survival process
for(int y = 1;y < Y;++y){
  for(int a = 1;a < A;++a){
    predN = log_N(y-1,a-1)-Z(y-1,a-1);
    if(a == A-1){//Plus group
 predN = logspace_add(predN,log_N(y-1,a)-Z(y-1,a));
    }
    nll -= dnorm(log_N(y,a),predN,sdS,true);
  }
}

//F correlation matrix making, sigmaFgen lives in Fcorr.hpp
matrix<Type> sigmaF =
   sigmaFgen(log_F.cols(),corflag,log_std_logF,tRhoF);
REPORT(sigmaF);

//Add the F random walk to the likelihood.
density::MVNORM_t<Type> Fdens(sigmaF);
for(int y = 1; y < Y;y++){
  nll += Fdens(log_F.row(y)-log_F.row(y-1));
}

//Expected catch, for making expected catch CRLs
//Expected landings too
for(int y = 0;y <Y;y++){
  for(int a = 0;a <A;a++){
    EC(y,a) = N(y,a)*(1.0-exp(-1.0*Z(y,a)))*F(y,a)/Z(y,a);
    ECW(y,a) = EC(y,a)*midy_weight(y,a);
  }
}
```

```
vector<Type> landings_pred = ECW.rowwise().sum();
vector<Type> total_expected_catch = EC.rowwise().sum();
matrix<Type> log_EC = EC.array().log();

vector<Type> log_landings_pred = log(landings_pred);
vector<Type> log_total_expected_catch = log(total_expected_catch);

REPORT(log_landings_pred);
REPORT(log_total_expected_catch);

// Landings, Survey index predictions
int ia,iy;
for(int i = 0;i < n;++i){
  int fType = ft(i);
  ia = iage(i);
  iy = iyear(i);

  //Observation equations for survey and landings
  switch(fType){
  case survey:
    std_index_vec(i) = std_index(isurvey(i));
    Elog_index(i) = log_qparm(iq(i)) + log_N(iy,ia) -
       fs(i)*Z(iy,ia);
    if(i_zero(i) == 1){ //if zero use left-censor bound
 nll -=
    keep(i)*pnorm4(log_index(i),Elog_index(i),std_index_vec(i),true);
    }
    else{
 nll -=
    keep(i)*dnorm(log_index(i),Elog_index(i),std_index_vec(i),true);
    }
    break;
  case land:
    Elog_index(i) = log_landings_pred(iy);
    std_index_vec(i) = std_landings(0);
    nll -=
       keep(i)*censored_bounds(log_index(i),Elog_index(i),std_index_vec(i),-l
    break;
  default:
    error("Fleet Type not implemented");
    break;
  }

}

matrix<Type> EPC = propM(EC);  //Expected Catch Proportions
matrix<Type> ECRL = makeCRLs(EPC); //Expected Catch Continuation
```

```
   Ratio Logits

matrix<Type> S = propM(F); //Selectivity
   F_{y,a}/(sum_{a}(F_{y,a}))

REPORT(S);
REPORT(EPC);
REPORT(ECRL);

//Correlation matrix for CRLs, might as well reuse sigmaFgen...
matrix<Type> sigmaCRL =
   sigmaFgen(ECRL.cols(),corflagCRL,log_std_CRL,tRhoCRL);
REPORT(sigmaCRL);

//MV NORMAL MIXTURE distribution, second argument is the mixture
   probability, grabbed from SAM lives in MVMIX.hpp
//MVNORM_t in TMB doesn't support OSA residuals yet, MVMIX_t from
   SAM does however there is a slight speed loss
MVMIX_t<Type> CRLdens(sigmaCRL,0); //cause this is a MV MIXTURE
   duh. I could maybe actually try to use this part?
for(int y = 0;y < Y;y++){
  //IT'S VERY VERY IMPORTANT CATCH CRLs ARE SORTED YEAR THEN AGE
     HERE.
  vector<Type> ECRLr = ECRL.row(y);
  nll +=
     CRLdens(crlVec.segment(y*(A-1),(A-1))-ECRLr,keepCRL.segment(y*(A-1),(A-1
}



//pop size weighted ave F;

vector<Type> aveFXY = aveF(F,N,aveFages(0),aveFages(1));
vector<Type> log_aveFXY = log(aveFXY);

REPORT(std_logF);
REPORT(std_log_R);
REPORT(std_index);

REPORT(N);
REPORT(F);
REPORT(Z);
REPORT(B_matrix);
REPORT(SSB_matrix);
REPORT(biomass);
REPORT(ssb);
REPORT(aveFXY);
```

```
  REPORT(EC);
  REPORT(ECW);

  REPORT(landings_pred);

  REPORT(Elog_index);

  REPORT(log_F);

  REPORT(log_std_index);
  REPORT(log_qparm);

  ADREPORT(log_landings_pred);
  ADREPORT(log_biomass);
  ADREPORT(log_ssb);
  ADREPORT(log_aveFXY);
  ADREPORT(log_qparm);

  ADREPORT(std_logF);
  ADREPORT(std_log_R);
  ADREPORT(std_index);

  return nll;
}
```

C.2 pnorm4.hpp

```
//Log of dnorm
template<class Type>
Type ldnorm(Type x){
  Type ret = 0.5*(-x*x-log(2.0*M_PI));
  return ret;
}

//Atomic function for log(pnorm)
//Needed to get good stable derivatives for use with pnorm4
//Atomic double specifes the forward mode derivative, which is just
   pnorm(x,log.p=TRUE) = log(pnorm)
//Reverse is the explicit derivative of log(pnorm) = dnorm/pnorm =
   exp(log(dnorm)-log(pnorm)), more stable derivatives at
//extreme values.
TMB_ATOMIC_VECTOR_FUNCTION(
            // ATOMIC_NAME
            lpnorm1
            ,
            // OUTPUT_DIM
            1
            ,
```

```
            // ATOMIC_DOUBLE
            ty[0] = atomic::Rmath::Rf_pnorm5(tx[0],0,1,1,1);
            ,
            // ATOMIC_REVERSE
            Type pn = ty[0]; //Get the log-supporting pnorm
            px[0] = exp(ldnorm(tx[0])-pn)*py[0]; //derivative of
               log(pnorm) in a more computatially stable way
            )

//Type function to link to atomic
template<class Type>
Type lpnorm1(Type x){
  CppAD::vector<Type> tx(1);
  tx[0] = x;
  return lpnorm1(tx)[0];
}

//vectorize it
VECTORIZE1_t(lpnorm1);

//A version of pnorm taking 4 arguments,
//q quantile of the normal distribution
//mean the mean of the normal distribution
//sd the standard deviation
//give_log if 1, returns log(pnorm(q,mean,sd))
template<class Type>
Type pnorm4(Type q, Type mean = 0.,Type sd = 1.,int give_log = 1){
  if(give_log == 0){
    return pnorm(q,mean,sd);
  }else{
    Type m = (q-mean)/sd;
    return lpnorm1(m);
  }
}

VECTORIZE1_t(pnorm4);
VECTORIZE4_ttti(pnorm4);


//Version for censored upper and lower bounds

//logspace_sub to do log(exp(log(x))-exp(log(y))) safely
extern "C" {
    /* See 'R-API: entry points to C-code' (Writing R-extensions) */
    double Rf_logspace_sub (double logx, double logy);
}

//Here x is std. normal
```

```
//This is the forward double function needed by the atomic function
//Returns log(exp(log(pnorm(x+upper)))-exp(log(pnorm(x-lower))))
   but in a much more compuationally stable way.
double censored_bou(double x,double lowerb,double upperb){
  double upper,lower;
  if(x > 0){
    lower = atomic::Rmath::Rf_pnorm5(-(x+upperb),0,1,1,1);
    upper = atomic::Rmath::Rf_pnorm5(-(x-lowerb),0,1,1,1);
  }else{
    upper = atomic::Rmath::Rf_pnorm5(x+upperb,0,1,1,1);
    //Rcout << "Upper bound: " << upper << std::endl;
    lower = atomic::Rmath::Rf_pnorm5(x-lowerb,0,1,1,1);
    //Rcout << "Lower bound: " << lower << std::endl;
  }
  double ret = Rf_logspace_sub(upper,lower);
  //Rcout << "What we ret: " << ret << std::endl;
  return ret;
}


TMB_ATOMIC_VECTOR_FUNCTION(
            // ATOMIC_NAME
            censored_b
            ,
            // OUTPUT_DIM
            1
            ,
            // ATOMIC_DOUBLE
            ty[0] = censored_bou(tx[0],tx[1],tx[2]);
            ,
            // ATOMIC_REVERSE
            Type cb = ty[0]; //Get the censored bounds from the
               forward mode.
            Type d1 = ldnorm(tx[0]+tx[2]);
            Type d2 = ldnorm(tx[0]-tx[1]);
            Type f1 = d1-cb;
            //Rcout << "This is f1: " << f1 << std::endl;
            Type f2 = d2-cb;
            //Rcout << "This is f2: " << f2 << std::endl;
            px[0] = (exp(f1)-exp(f2))*py[0];
            )

template<class Type>
Type censored_b(Type x,Type lowerb,Type upperb){
  CppAD::vector<Type> tx(3);
  tx[0] = x;
  tx[1] = lowerb;
  tx[2] = upperb;
```

```
  return censored_b(tx)[0];
}

//Function to calculate
   log(pnorm(x+upper,mu,sd)-pnorm(x-lower,mu,sd)) with stable
   derivatives
template<class Type>
Type censored_bounds(Type x,Type mu,Type sd,Type lower,Type upper){
  Type z = (x-mu)/sd;
  Type upperb = upper/sd;
  Type lowerb = lower/sd;
  return censored_b(z,lowerb,upperb);
}
```

C.3 Fcorr.hpp

```
template<class Type>
Type rhoTrans(Type x){
  return Type(2)/(Type(1) + exp(-Type(2)*x))-Type(1);
}

VECTORIZE1_t(rhoTrans);


/*
  Generate correlation matrix for F's

  param nage number of ages
  param corrFlag Type of correlation structure
  param tRhoF vector of untransformed rhos for corr structure

*/
template<class Type>
matrix<Type> sigmaFgen(int nage,int corrFlag, vector<Type>
   logsdF,vector<Type> tRhoF){

  enum corrType{
      independent = 0,
      parallel = 1,
      compound = 2,
      ARone = 3,
      custom = 4
  };

  matrix<Type> sigma(nage,nage);
  vector<Type> sdF = exp(logsdF);
  vector<Type> rhoF = rhoTrans(tRhoF);
```

```
sigma.setZero();

switch(corrFlag){
case independent:
  sigma.diagonal() = sdF*sdF;
  break;
case parallel:
  sigma.diagonal() = sdF*sdF;
  for(int i=0; i < nage; i++){
    for(int j=0; j < i; j++){
 sigma(i,j) = sdF(i)*sdF(j);
 sigma(j,i) = sigma(i,j);
    }
  }
  break;
case compound:
  sigma.diagonal() = sdF*sdF;
  for(int i = 0;i < nage; i++){
    for(int j = 0; j < i; j++){
 sigma(i,j) = rhoF(0)*sdF(i)*sdF(j);
 sigma(j,i) = sigma(i,j);
    }
  }
  break;
case ARone:
  sigma.diagonal() = sdF*sdF;
  for(int i = 0;i < nage; i++){
    for(int j = 0; j < i; j++){
 sigma(i,j) = pow(rhoF(0),Type(i-j))*sdF(i)*sdF(j);
 sigma(j,i) = sigma(i,j);
    }
  }
  break;
case custom:
  sigma.diagonal() = sdF*sdF;
  for(int i = 0;i < nage; i++){
    for(int j = 0; j < i; j++){
 sigma(i,j) = rhoF(i)*rhoF(j)*sdF(i)*sdF(j);
 sigma(j,i) = sigma(i,j);
    }
  }
  break;


default:
  error("F␣correlation␣mode␣not␣supported");
    break;
}
```

```
  return sigma;
}
```

C.4 MVMIX.hpp

This code was taken from the SAM github page as it supports OSA residuals and the current MVNORM_t provided in TMB does not. The mixture probability was set to 0 in the model. It is also licensed as GPL-2 and thus fine for inclusion here.

```
/*Code ripped from SAM's
   SAM/stockassessment/inst/include/define.hpp github file.
  I'm not going to attempt writing my own MVNORM with support for
     OSA residuals for a class project...
  Maybe for fun later...*/

template<class Type>
Type logspace_add_p (Type logx, Type logy, Type p) {
  return log((Type(1)-p)*exp(logy-logx)+p)+logx; // the order of x
     and y is taylored for this application
}

template<class Type>
Type logdrobust(Type x, Type p){
  Type ld1=dnorm(x,Type(0.0),Type(1.0),true);
  if(p<Type(1.0e-16)){
    return ld1;
  }else{
    Type ld2=dt(x,Type(3),true);
    Type logres=logspace_add_p(ld2,ld1,p);
    return logres;
  }
}
VECTORIZE2_tt(logdrobust)

template <class Type>
class MVMIX_t{
  Type halfLogDetS;
  Type p1;                    /*fraction t3*/
  matrix<Type> Sigma;
  vector<Type> sd;
  matrix<Type> L_Sigma;
  matrix<Type> inv_L_Sigma;
public:
  MVMIX_t(){}
  MVMIX_t(matrix<Type> Sigma_, Type p1_){
    setSigma(Sigma_);
    p1=p1_;
  }
```

```
matrix<Type> cov(){return Sigma;}
void setSigma(matrix<Type> Sigma_){
  Sigma = Sigma_;
  sd = sqrt(vector<Type>(Sigma.diagonal()));
  Eigen::LLT<Eigen::Matrix<Type,Eigen::Dynamic,Eigen::Dynamic> >
     llt(Sigma);
  L_Sigma = llt.matrixL();
  vector<Type> D=L_Sigma.diagonal();
  halfLogDetS = sum(log(D));
  inv_L_Sigma = atomic::matinv(L_Sigma);
}
void setSigma(matrix<Type> Sigma_, Type p1_){
  setSigma(Sigma_);
  p1=p1_;
}
/** \brief Evaluate the negative log density */
Type operator()(vector<Type> x){
  vector<Type> z = inv_L_Sigma*x;
  return -sum(logdrobust(z,p1))+halfLogDetS;
}
Type operator()(vector<Type> x, vector<Type> keep){
  matrix<Type> S = Sigma;
  vector<Type> not_keep = Type(1.0) - keep;
  for(int i = 0; i < S.rows(); i++){
    for(int j = 0; j < S.cols(); j++){
 S(i,j) = S(i,j) * keep(i) * keep(j);
    }
    //S(i,i) += not_keep(i) *
       pow((Type(1)-p1)*sqrt(Type(0.5)/M_PI)+p1*(Type(1)/M_PI),2);
       //(t(1))
    S(i,i) += not_keep(i) *
       pow((Type(1)-p1)*sqrt(Type(0.5)/M_PI)+p1*(Type(2)/(M_PI*sqrt(Type(3)))
  }
  return MVMIX_t<Type>(S,p1)(x * keep);
}

vector<Type> simulate() {
  int siz = Sigma.rows();
  vector<Type> x(siz);
  for(int i=0; i<siz; ++i){
    Type u = runif(0.0,1.0);
    if(u<p1){
      x(i) = rt(3.0);
    }else{
      x(i) = rnorm(0.0,1.0);
    }
  }
  x = L_Sigma*x;
```

```
    return x;
  }
};

template <class Type>
MVMIX_t<Type> MVMIX(matrix<Type> Sigma, Type p1){
  return MVMIX_t<Type>(Sigma,p1);
}
```

C.5 C++ code used outside model

I decided it would be faster to simply convert the code I wrote to do CRLs inside the model for use in R when I had to switch from doing all the CRL related stuff inside the model to both inside and out for the sake of OSA residuals. This is just a version of makeCRLs that uses the Eigen library directly and does not take the log for the CRL as that's done later inside R. Using `sourceCpp` from `Rcpp` package these can be used as functions inside R. They are quick but have zero checks.

C.6 R Code used