White Sturgeon Pop Model: A Brief Overview

Introduction

S. Blackburn (Univ. of Idaho, contracted through CDFW) developed a computational model to understand San Francisco Estuary (SFE) White Sturgeon population dynamics. The model assess populations changes (λ) over varying levels of exploitation (μ; harvest rate) given current and possible size limits.

Herein we demonstrate model operation using (for simplicity) fixed (i.e., assumptions cannot be modified) data inputs. (In the future, we'll implement functionality to allow for some data input manipulation.) The results as presented here project λ — under current fishing regulations — over 20 years using CDFW mark-recapture data 2014-2016. These data were adjusted for gear selectivity using methods written by Millar (https://www.stat.auckland.ac.nz/~millar/selectware/RNext/). The CDFW used modified trammel nets with inner webbing sizes of 6“, 7”, and 8" mesh. Each trammel net typically was configured with 2x8" panels, 1x6" panel, & 1x7" panel (i.e., 8" mesh was fished at twice more the effort than the 6" or 7" mesh; one panel is 150’ in length).

The fixed data inputs carry the assumptions below.
1. 15% of females spawn in a given year
2. for exploitation (μ)
- 100% reporting of highest $ tag (i.e., $150)
- tag loss at 10% & tagging mortality at 1%
3. spawning periodicity & egg to age-1 mortality rates from older (20+ years) literature
4. initial abundance (of total population) set at 48,000
5. female:male ratio 50:50

Load Libraries

Model algorithms and processes were written using R (unclear which version but likely 3.4.x). Additionally, the model uses three (3) functions from the `popbio` package (version 2.4.3).

`betaval()` (https://www.rdocumentation.org/packages/popbio/versions/2.4.3/topics/betaval)
`stretchbetaval()` (https://www.rdocumentation.org/packages/popbio/versions/2.4.3/topics/stretchbetaval)
`pop.projection()` (https://www.rdocumentation.org/packages/popbio/versions/2.4.3/topics/pop.projection)

```
library(popbio)
library(ggplot2)
```

Load Data

For this purpose, we stored all model input data in a `.Rdata` file. Data sourced from S. Blackburn's file `corrected_transient_midfecund_current.R`. We load this here to being. Data are 20 observations (i.e., ages 0-19) by 2 or 3 variables (shown below in each section, 5 rows at a time).

S. Blackburn created a separate `prob_survival_` dataframe for each level of μ (see below for all variable names). For convenience and neatness, we put these variables in a new environment (`prob_survival`), and then removed each from the Global environment. (Later, we'll build functionality so only one “prob_survival” variable is needed.)

```
load(file = "data/model/StartingData.RData")

# putting all prob_survival dataframes in separate env for neatness
prob_surv <- ls(pattern = "prob_survival")

# naming for convenience later in the process (model)
names(prob_surv) <- prob_surv

prob_survival <- new.env()

vapply(prob_surv, FUN = function(x) {
  assign(x = x, value = get(x), envir = prob_survival)
  rm(list = x, envir = .GlobalEnv)
  x
}, FUN.VALUE = character(1L), USE.NAMES = FALSE)
```

```
##  [1] "prob_survival_01mu" "prob_survival_02mu" "prob_survival_03mu"
##  [4] "prob_survival_04mu" "prob_survival_05mu" "prob_survival_06mu"
##  [7] "prob_survival_07mu" "prob_survival_08mu" "prob_survival_09mu"
## [10] "prob_survival_10mu" "prob_survival_11mu" "prob_survival_12mu"
## [13] "prob_survival_15mu" "prob_survival_16mu" "prob_survival_17mu"
## [16] "prob_survival_18mu" "prob_survival_19mu" "prob_survival_20mu"
## [19] "prob_survival_21mu" "prob_survival_22mu" "prob_survival_23mu"
## [22] "prob_survival_24mu" "prob_survival_25mu" "prob_survival_26mu"
## [25] "prob_survival_27mu" "prob_survival_28mu" "prob_survival_29mu"
## [28] "prob_survival_30mu" "prob_survival_nomu" "prob_survival_obs"
```

initial_age_dist

S. Blackburn assigned ages to non-aged fish (~650; using `FSA::alkIndivAge()` (https://www.rforge.net/doc/packages/FSA/alkIndivAge.html)) using an age-length key made from ~350 aged fish. Data were corrected for gear selectivity, and then scaled to a starting abundance (N=48,000). S. Blackburn used a log-linear model to estimate age-1 & age-2 abundance, and then multiplied values by 0.5 (for assumed M:F ratio of 50:50). Age-0 value derived from White Sturgeon (female) fecundity and abundance.

Ages range (here & throughout the model) from 0 to 19. Variable `initial_age_dist` is — by original design — a matrix, but herein we've converted it to a dataframe for convenience and consistency.

age <dbl>	freq <dbl>
0	2.193873e+08
1	3.242553e+03
2	2.013134e+03
3	7.629124e+02
4	8.929627e+02

1-5 of 20 rows Previous **1** 2 3 4 Next

prob_survival_obs

Data are probability of survival for each age. Survival probability calculated using Chapman-Robson peak+1 method and "[SE adjusted] for overdispersion (using chat variance inflation)." Age-0 through -2 values from extant literature.

Many `prob_survival_<NAME>` dataframes are loaded into this session (below we use `prob_survival_obs` as an example). Each one considers the effects of exploitation (μ; and increments by 0.01 from 0 to 0.30) for the cohort susceptible to harvest (i.e., ages 10-15, according to von Bertalanffy growth model). For all other age groups, survival probability remains constant irrespective of μ. Reference to `_obs` indicates observed survival probabilities under current White Sturgeon harvest conditions (i.e., where μ ~0.13).

age <int>	prob <dbl>	SE <dbl>

age <int>	prob <dbl>	SE <dbl>
0	0.00200	0.00300
1	0.25000	0.05000
2	0.84000	0.16800
3	0.94576	0.04281
4	0.94576	0.04281

1-5 of 20 rows Previous **1** 2 3 4 Next

prob_spawn

Data generated using logistic regression on Chapman's (1989, doctoral thesis [I believe]) summarized data (i.e., percent of sexually mature females as a function of fork length [cm]). Standard errors (SE) were predicted from logistic model.

Spawning (according to extant literature, Chapman et al. 1996) is not likely to occur prior to age 10. `prob_spawn` is created as such, where `prob` is 0 for ages 0-9.

	Age <dbl>	prob <dbl>	SE <dbl>
1	0	0.00000	0.00000
2	1	0.00000	0.00000
3	2	0.00000	0.00000
4	3	0.00000	0.00000
5	4	0.00000	0.00000

1-5 of 20 rows Previous **1** 2 3 4 Next

number_eggs

Data are number of eggs per age, derived from linear regression (fork length ~ number of eggs; linear model not shown here but starting data [Devore et al. 1995] are given below). SEs were predicted from the linear model. Again, 0s for ages 0-9. Length ~104 cm FL roughly equivalent to age-10.

	Age <dbl>	count <dbl>	SE <dbl>
1	0	0.00	0.000
2	1	0.00	0.000
3	2	0.00	0.000
4	3	0.00	0.000
5	4	0.00	0.000

1-5 of 20 rows Previous **1** 2 3 4 Next

ForkLen	NumOfEggs
105	63041.86
115	82372.86
125	105256.26
135	131981.72
145	162837.50
155	198110.59

ForkLen	NumOfEggs
165	238086.79
175	283050.78
185	333286.24
195	389075.83

Model

The process, as described below by S. Blackburn, is presented below. `A` refers to a Leslie female-based population matrix.

1. Simulate Ai iid population matrices, i = 1:n, n is the number of simulations, where the random iid vital rates are drawn from the following distrubitions:
    i. prob_survival ~ Beta(a, b) with a and b such that E(prob_survival) = mean, and Var(prob_survival) = SE^2 ia. include recruitment stochasticity (age-0 survival only)
    ii. prob_spawn ~ Beta(a, b) with a and b such that E(prob_spawn) = mean, and Var(prob_spawn) = SE^2

iii. number_eggs ~ StretchBeta(a, b, min, max) with same type of expectation and variance as above. support = [min, max] = [0, 2* ma  s] Use functions betaval() and stretchbetaval() to generate from Beta and StretchBeta distributions, respectively, with the appropriate parameters.

2. Assemble vital rates into post-breeding matrices. This means we include newborns (age-0) in our census, they are 1st age class
3. Use pop.projection() function to iterate each population j = 1:t times by multiplying matrix Ai by the population vector at time j. Because the same Ai will be used accross the t times, we are interested when t is small (to look at short-term changes in the population)
   3a. Calculate observed lambda for each matrix at time t by comparing the population change from time t-1 to time t (which is already done by pop.proj in pop.changes)
4. Calculate the mean log lambda for each Ai
5. For the overall value, look at the geometric mean of the lambdas.

Herein, we've created some custom functions (see `functions-model_simsurv.R` (https://github.com/jasondubois/SturgeonPopModel/blob/master/source/functions-model_simsurv.R)) to simplify a bit this multi-stepped process.

The model includes stochasticity, and as such we need to first set the number of iterations (`iters`). We can adjust this value accordingly, but for ease of demonstration we'll set it to 100 (for best results it should be upwards of 5K or more).

```
iters <- 100
# iters <- 5000
```

Survival Rates

For each `prob_survival_<NAME>` variable, we simulate (`n=iters`) survival rates (using mean [see `prob` field] & standard deviation [see `SE` field]) for each row (i.e., `age`). We do this using `popbio::betaval()` & the custom `SurvivalSims()`. `prob_survival_obs` are the observed survival rates under current conditions (i.e., fishing regulations & exploitation). All others vary μ (from 0 [no μ] to 0.30 or 30%).

```
# span = period (in years on average) of successful recruitment; e.g., 5 =
# successful recruitment once every 5 years
sims_survival <- SurvivalSims(
  probSurv = prob_surv,
  env = prob_survival,
  iterations = iters,
  span = 5
)
```

Spawning

Here we simulate spawning (by `age`) using probability of spawning in `prob_spawn`. We do this using the custom `BetavalSims()`.

```
# simulate probability of spawning
sims_spawning <- BetavalSims(
  data = prob_spawn,
  mn = prob,
  sdev = SE,
  iterations = iters
)
```

Fecundity

We first simulate the number of eggs (using custom `SBetavalSims()`), and then we calculate simulated fecundity. For now, we assume a sex ratio of 50%.

```
# Simulate the number of eggs
sims_num_eggs <- SBetavalSims(
  data = number_eggs,
  mn = count,
  sdev = SE,
  iterations = iters
)
```

```
## Warning: `maxb` set at 3 * count
```

```
# assumed 50% sex ratio
sex_ratio <- 0.5

# Simulate fecundity rate from prob_spawning_sims and number_eggs_sims
sims_fecundity <- sims_num_eggs * sims_spawning * sex_ratio

# if needed, we can dispose of sims_num_eggs & sims_spawning, as these variables
# are not needed anymore & may consume memory

# rm(sims_num_eggs, sims_spawning)
```

Leslie Matrix & Model Results

The crux of this model is a post-breeding female-based Leslie matrix. We create the matrix and get results using custom `GetLambda()`. Here is what S. Blackburn says about this step of the process (model).

> W[hite Sturgeon] caught right AFTER (post) their “birthday” can contribute newborn (age 0) animals to next year’s census by surviving to their next birthday, then reproducing ON that birthday. Ex: fecundity of 9-year-olds (10th fecundity row element in matrix) is: survival from 9-10 (S9, 10th survival element), times fecundity on 10th bday (f10, 11th repro element)

Note: `GetLambda()` is a bit convoluted and is very slow when iterations increase. In future versions, we'll try to improve performance and readability.

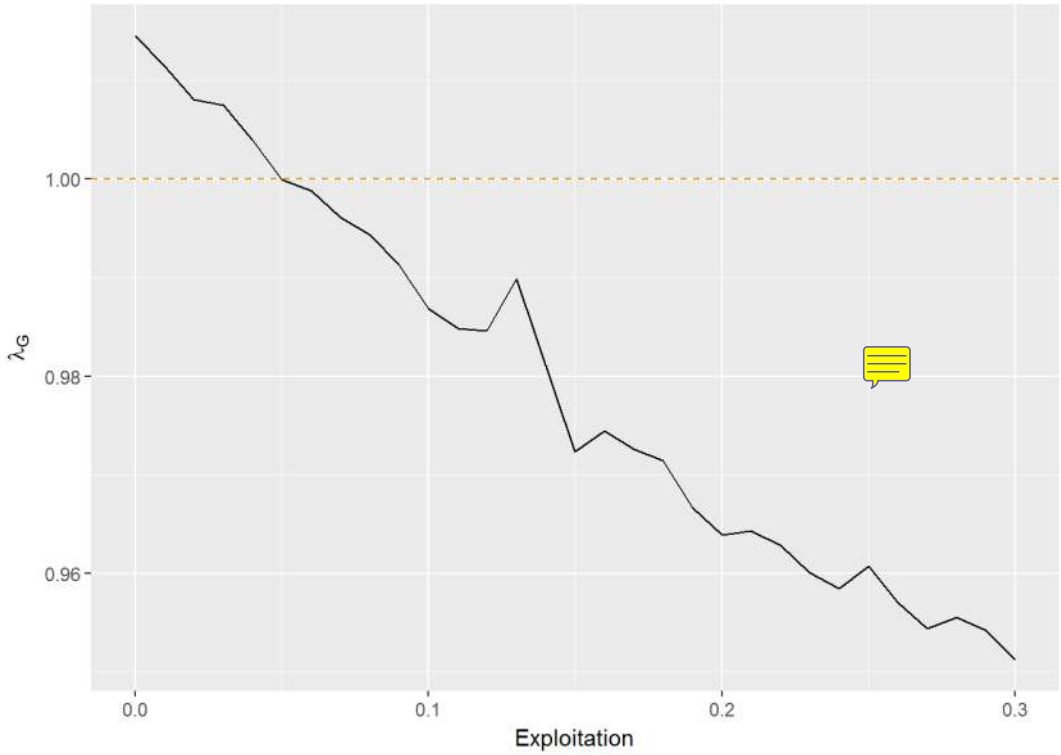
Note: the `iterations` parameter in `GetLambda()` is the period (in years) over which population changes are projected. (For now, we've hard-coded this value, but in the future we will allow for variation.)

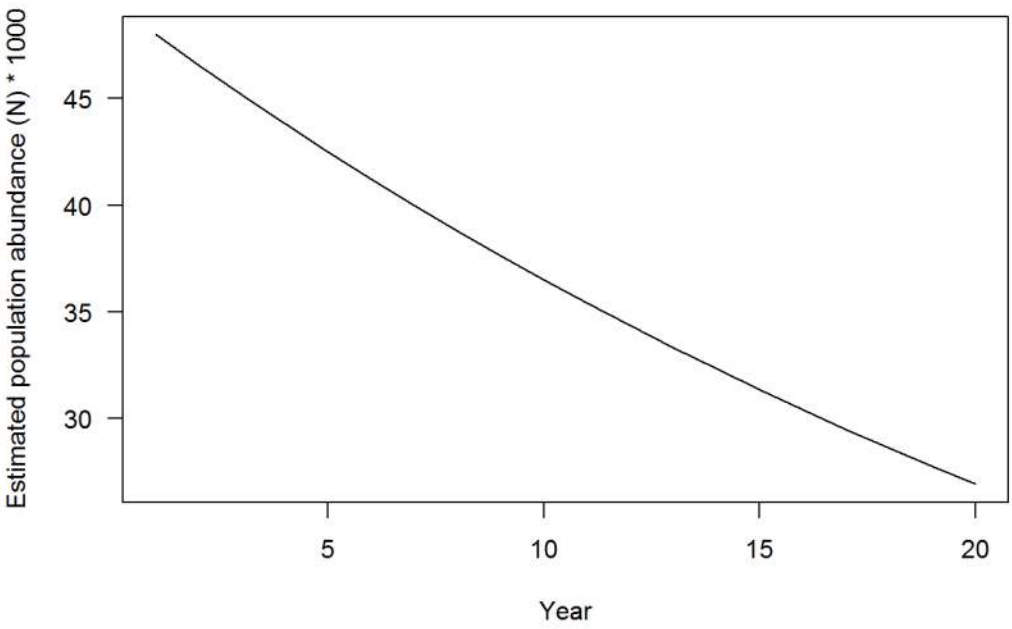
```
# get lambda for each level of exploitation & current observations
results <- lapply(prob_surv, function(x) {
  GetLambda(
    data = prob_survival[[x]],
    mn = prob,
    sdev = SE,
    fecundSims = sims_fecundity,
    survSims = sims_survival[[x]],
    n = initial_age_dist[["freq"]],
    iterations = 20
  )
})

# clean up for ease of display
results <- do.call(what = rbind, args = results)

results[["Level"]] <- vapply(
  strsplit(rownames(results), split = "_"),
  FUN = "[", 3,
  FUN.VALUE = character(1L)
)

rownames(results) <- NULL
```

Level	Sims	MeanLambda	MedLambda	LBLambda	UBLambda
01mu	100	1.0114693	1.0071259	0.9537162	1.161933
02mu	100	1.0081064	1.0017804	0.9500041	1.149249
03mu	100	1.0074822	1.0014263	0.9433445	1.155495
04mu	100	1.0040064	0.9974665	0.9498985	1.141207
05mu	100	0.9999253	0.9924679	0.9505151	1.145142
06mu	100	0.9987928	0.9919891	0.9348335	1.146885
07mu	100	0.9961561	0.9899727	0.9500004	1.133752
08mu	100	0.9943256	0.9881472	0.9464035	1.139134
09mu	100	0.9913158	0.9836748	0.9404351	1.129576
10mu	100	0.9869173	0.9832045	0.9253567	1.119324
11mu	100	0.9848205	0.9809276	0.9283475	1.124150
12mu	100	0.9846312	0.9786456	0.9244305	1.120223
15mu	100	0.9723378	0.9690639	0.8906302	1.102077
16mu	100	0.9744934	0.9711951	0.9262265	1.087056
17mu	100	0.9725816	0.9690767	0.9092515	1.093828
18mu	100	0.9714548	0.9667097	0.8923140	1.089841
19mu	100	0.9666320	0.9660593	0.8956398	1.087505
20mu	100	0.9639198	0.9611920	0.8977269	1.083929
21mu	100	0.9643287	0.9627481	0.9051310	1.078247

Level	Sims	MeanLambda	MedLambda	LBLambda	UBLambda
22mu	100	0.9628260	0.9637878	0.8963261	1.081247
23mu	100	0.9601089	0.9574854	0.8853988	1.086650
24mu	100	0.9584820	0.9595920	0.8892673	1.083042
25mu	100	0.9607701	0.9561430	0.9031030	1.083831
26mu	100	0.9570957	0.9555673	0.9053700	1.056027
27mu	100	0.9543960	0.9535648	0.8731736	1.056655
28mu	100	0.9555672	0.9544079	0.8838600	1.045471
29mu	100	0.9542165	0.9535827	0.9045002	1.050457
30mu	100	0.9512265	0.9507197	0.8894188	1.057073
nomu	100	1.0144756	1.0046918	0.9541966	1.181512
obs	100	0.9898701	0.9795174	0.8995244	1.144800

Plot of λ_G (geometric mean) as a function of exploitation over 20 year period. A λ of 1 is a stable population, below 1 is a declining population. Under current conditions, in 20 years our starting population (N=48,000) will look something like the second plot below.





ran: 2018-05-15 11:16:15
CDFW, Sportfish Unit