Generate Model Input Data

2019-04-22

```
library(spopmodel)
```

Background

A key step in `spopmodel` employs a female-based Leslie Matrix. Thus, this model requires data about survival probability, reproduction (spawing probability & fecundity), and age distribution (frequency). Here we show — using raw age & length data — the processes by which we generate model inputs. Ultimately, we use these inputs (or starting data) to run simulations (see vignette of same name) that then "feed" the model.

`spopmodel` was developed to assess San Francisco-estuary based population dynamics. Herein we use the data on which the model was built (i.e., White Sturgeon data collected by the California Deparment of Fish & Wildlife [CDFW] in collaboration with the US Fish & Wildlife Service and the University of Idaho [the model developers]).

Note: about data not exact to that used in thesis *Note*: if you already have these data proceed to sims *Note*: ??

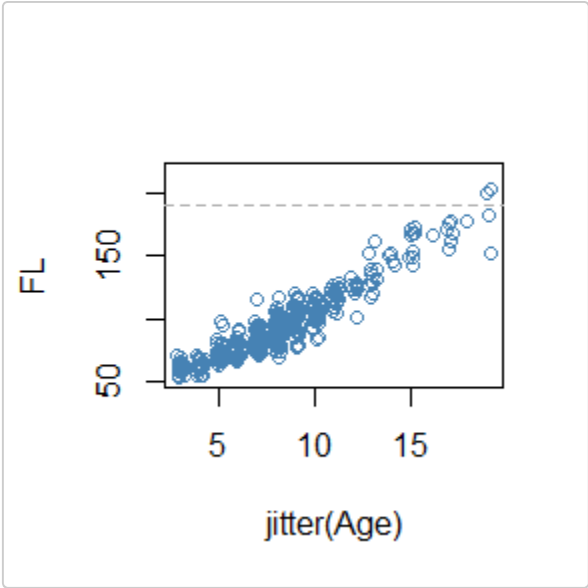
Data

For this demonstration, we'll use dataset `trammel_catch`. These data are from the CDFW's Sturgeon Study. This particular dataset is from 2014-2016 sampling. It contains age & length data, with 361 aged fish from 1000 observations. Age was estimated from fin ray samples collected in the field.

Note: Herein we **do not** adjust catch data for gear selectivity. Likely, such an adjustment would increase catch in most of the length bins (and thus ages). A vignette about performing such adjustment is forthcoming.

Let's plot FL ~ Age to get some sense of range and mean-length-at-age. We added some noise (`jitter()`) to variable Age to reduce over-plotting.

```
plot(FL ~ jitter(Age), data = trammel_catch, col = "steelblue")
abline(h = 190, col = "grey70", lty = 2)
```



We arbitrarily set a horizontal line at 190 cm FL and noted we do not have many *aged* fish ≥ 190. If fact, as we see below, the entire dataset only has 4 fish ≥ 190 cm FL. Thus, to ensure no empty bins in our age-length key, we will (somewhat arbitrarily) set age to 19 for "age-less" fish ≥ 190 cm FL.

Note: In reality, we may decide to age these large fish if we had the means to do so (i.e., had a fin-ray sample). But for this purpose, age-19 is a pretty good surrogate give aged fish of similar size. We also could manipulate bin size (in length frequency) to see if these fish are binned with any aged fish.

```
bool_gte190 <- trammel_catch[["FL"]] >= 190

trammel_catch[bool_gte190, ]
#>     MeshSize  FL Age
#> 456        8 204  19
#> 528        7 194  NA
#> 688        8 200  19
#> 962        8 217  NA
```

We can replace `NA` with 19 simply using `is.na()` along with our boolean above. We display the results to verify our change.

```
bool_ageNA <- is.na(trammel_catch[["Age"]])

trammel_catch[bool_gte190 & bool_ageNA, "Age"] <- 19

# verification
trammel_catch[bool_gte190, ]
#>     MeshSize  FL Age
```

```
#> 456        8 204  19
#> 528        7 194  19
#> 688        8 200  19
#> 962        8 217  19
```

Model Inputs

`spopmodel`'s design requires four data inputs: age distribution; spawning probability; egg count; and survival probability. All four must configure to the age range established in age distribution, which should contain young age classes (i.e., ages-0 to ages-2). These datasets are not fed directly to the model but rather are used to run stochastic simulations. The model directly uses these simulations to generate population growth rate (λ) predictions.

Though not directly a model input, we begin by creating a length frequency distribution. As we'll see in the next section, length frequency if useful for — among other things — aging the un-aged portion.

Length Frequency

To age the un-aged portion of our data, we have a couple of methods from which to choose. Both methods — using age-length key or assigning age to each fish — require length frequency. We use `Frequency()` for such a task, setting bin width to 5 cm. (For White Sturgeon, 5 cm is a good starting bin size, but you may want to experiment with various sizes. Here, we will stick with 5 cm.) As shown below, `Frequency()` output provides a list with nine elements.

```
len_freq <- Frequency(trammel_catch[["FL"]], binWidth = 5)

str(len_freq)
#> List of 9
#>  $ breaks  : num [1:34] 53 58 63 68 73 78 83 88 93 98 ...
#>  $ counts  : int [1:33] 7 21 34 55 101 126 111 91 66 77 ...
#>  $ density : num [1:33] 0.0014 0.0042 0.0068 0.011 0.0202 0.0252 0.0222 0.0182 0.0132
  0.0154 ...
#>  $ mids    : num [1:33] 55.5 60.5 65.5 70.5 75.5 ...
#>  $ xname   : chr "trammel_catch[[\"FL\"]]"
#>  $ equidist: logi TRUE
#>  $ xna     : int 0
#>  $ binw    : num 5
#>  $ xstats  :function ()
#>  - attr(*, "class")= chr [1:2] "Frequency" "histogram"
```

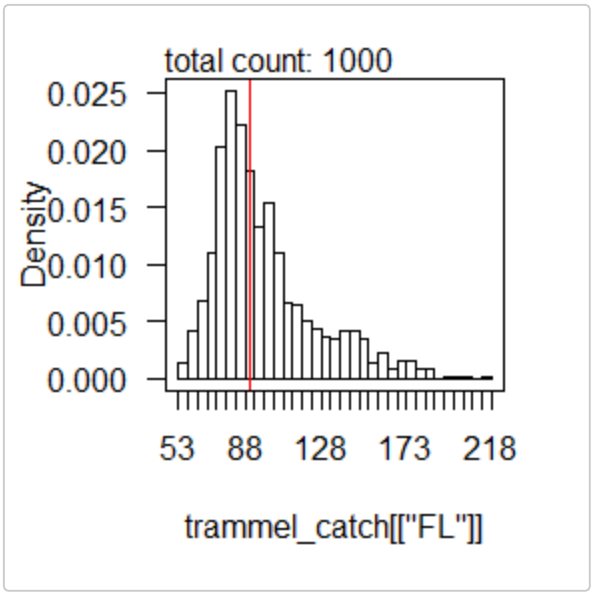
For curiousity, we'll run some descriptive stats on variable FL. We see a range from 53 to 217 cm FL.

```
len_desc_stats <- len_freq$xstats()

# for display
unlist(len_desc_stats)
#>      NAll         N       Min       Max       Avg       Med       Var
#> 1000.0000 1000.0000   53.0000  217.0000   97.5170   90.0000  728.7845
```

We also note from `len_freq` we get breaks (or bins; see `len_freq$breaks`). We'll use this in the next step (below) to age each fish. Plotting length-frequency we see our data is skewed right. The red vertical line denotes the median (90 cm FL).

```
plot(len_freq)
```



Age Distribution

`AgeEach()` is modeled after `FSA::alkIndivAge()`. The stripped-down output (i.e., just the ages) of `AgeEach()` is better suited for `spopmodel`. (**Note** however more testing of `AgeEach()` is still required as of 15-Oct-2018.)

```
ages <- AgeEach(
  data = trammel_catch,
  len = FL,
  age = Age,
  lenBreaks = len_freq$breaks
)
```

Now we can simply call R's `table()` to generate age-frequency. We now have our age-frequency (for ages 3-19). (Sum of `age_freq` should equal 1000. You can double-check if you like.)

```
age_freq <- table(ages[["Ages"]], dnn = NULL)

age_freq
#>   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
#>  28  38  70 110 173 159 148  73  37  25  54  22  32   3  15   4   9
```

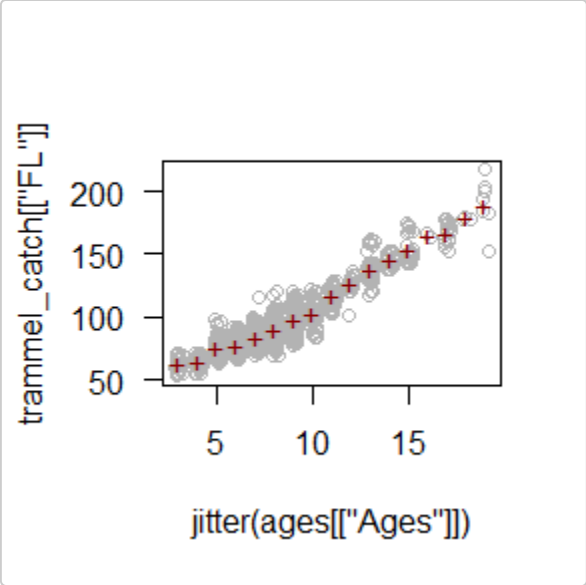
```
# mean length at each age (3-19)
mean_len_age <- aggregate(
  trammel_catch[["FL"]],
  by = list(Age = ages[["Ages"]]),
  FUN = mean
)

# something more descriptive than 'FL'
colnames(mean_len_age)[2] <- "MeanFL"

mean_len_age
#>    Age    MeanFL
#> 1    3  63.21429
#> 2    4  64.13158
#> 3    5  75.92857
#> 4    6  77.58182
#> 5    7  84.28902
#> 6    8  89.80503
#> 7    9  98.68243
#> 8   10 103.02740
#> 9   11 117.18919
#> 10  12 126.52000
#> 11  13 137.96296
#> 12  14 145.54545
#> 13  15 153.18750
#> 14  16 165.33333
#> 15  17 167.26667
#> 16  18 178.75000
#> 17  19 189.00000
```

Let's plot our raw length ~ age data, overlayed with mean-length-at-age (red '+' sign). It looks reasonable and as we expected.

```
plot(jitter(ages[["Ages"]]), trammel_catch[["FL"]], col = "grey70", las = 1)
points(mean_len_age, pch = "+", col = "darkred")
```



We now need an age distribution based on a starting population. We use `AgeDist()` passing our `age_freq` variable and keeping defaults `abund = 48000` and `fracFemale = 0.5`. The output (as shown below) is the basis for how data `age_dist` was created. (With `spopmodel` library loaded, view `age_dist` & compare to `age_distribution`.)

```
age_distribution <- AgeDist(ageFreq = age_freq)

age_distribution
#> $EstAgeAbund
#>    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
#> 1344 1824 3360 5280 8304 7632 7104 3504 1776 1200 2592 1056 1536  144  720
#>   18   19
#>  192  432
#>
#> $CountFemByAge
#>    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
#>  672  912 1680 2640 4152 3816 3552 1752  888  600 1296  528  768   72  360
#>   18   19
#>   96  216
#>
#> $TotalN
#> [1] 1000
#>
#> $OverallAbund
#> [1] 48000
```

```
#>
#> $FracFemale
#> [1] 0.5
```

Though we have our age-distribution, we still need values for ages 0-2. Per model development, age-1 & -2 values were obtained using linear regression (specifically log-linear).

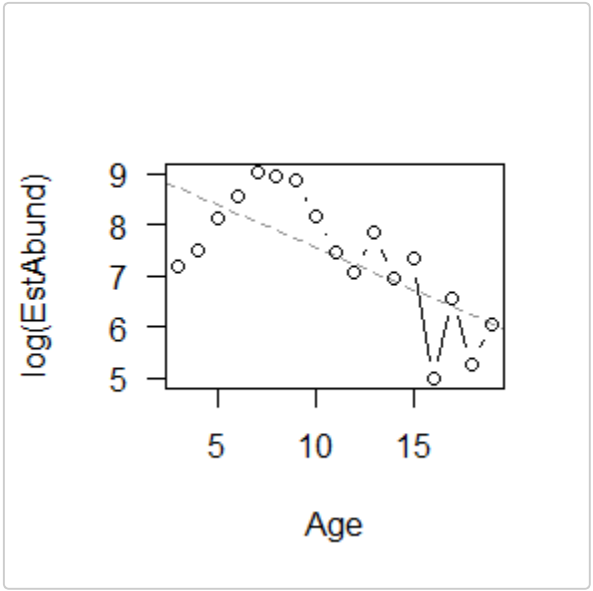
For convenience, we assign estimated age abundance (`EstAgeAbund`) to variable `est_abundance`. Estimated age abundance is simply age-frequency (as a proportion of total) multiplied by our starting abundance (in this case 4.810^{4}). We assign our x & y values accordingly, use R's `lm()` for linear regression, and then plot the results. As we expect, frequency declines with age.

```
est_abundance <- age_distribution[["EstAgeAbund"]]

# variables for plotting & linear regression
yval <- log(as.vector(est_abundance))
xval <- as.numeric(names(est_abundance))

mod <- lm(yval ~ xval)

plot(
  x = xval,
  y = yval,
  type = "b",
  xlab = "Age",
  ylab = "log(EstAbund)",
  panel.first = abline(mod, col = "grey60", lty = 2),
  # panel.first = grid(),
  las = 1
)
```



We use our linear model to predict ages 1 & 2. These values, however, include males & females. We need only females and will combine ages 1 & 2 with ages 3-19 (females only).

```
age1_2 <- exp(predict(object = mod, newdata = list(xval = c(1, 2))))

females <- c(
  age1_2 * age_distribution$FracFemale,
  age_distribution$CountFemByAge
)

females
#>        1        2        3        4        5        6        7        8
#> 4368.617 3699.198  672.000  912.000 1680.000 2640.000 4152.000 3816.000
#>        9       10       11       12       13       14       15       16
#> 3552.000 1752.000  888.000  600.000 1296.000  528.000  768.000   72.000
#>       17       18       19
#>  360.000   96.000  216.000
```

Spawning Probability

Later we will calculate age-0 distribution (frequency) and append `females` accordingly. To do so we need spawning probability, which we will create now.

For our White Sturgeon population, we do not have age-specific spawning probabilities. Thus, we rely on our age & length data plus probability data from previous research.

Running `SpawningProb()` will issue a warning saying "Using `SpawningProbWST` for `pMat`". `pMat` is short for probability of maturity and in leiu of our own dataset, we use data from Champman (1989), termed herein `SpawningProbWST`. These data can be found here.

Internally, `SpawningProb()` performs `glm()` (as probability ~ length) using `SpawningProbWST`, and then from this model predicts probabilty given mean-length-at-age. The resulting `Prob` in `p_spawn` is the predicted probability multiplied by the argument supplied to `mature` (in this case default 0.15, or 15% of all females spawn annually). `Err` is probability multiplied by 0.20.

Note: `glm()` inside `SpawningProb()` may issue the following waring: non-integer #successes in a binomial glm! As of 24-Oct-2018, we are still looking into this.

```
bool_mlaa_gt9 <- mean_len_age$Age > 9
```

```
# using default argument for mature parameter (0.15)
p_spawn <- with(data = mean_len_age[bool_mlaa_gt9, ], expr = {
  SpawningProb(len = MeanFL, age = Age)
})
#> Warning: Using `SpawningProbWST` for `pMat`.
#> Warning in eval(family$initialize): non-integer #successes in a binomial
```

```
#> glm!
```

```
p_spawn
#>            l Age        Prob          Err
#> 1   103.0274  10 0.004043737 0.0008087475
#> 2   117.1892  11 0.012151673 0.0024303346
#> 3   126.5200  12 0.023842223 0.0047684447
#> 4   137.9630  13 0.048751695 0.0097503390
#> 5   145.5455  14 0.070837466 0.0141674932
#> 6   153.1875  15 0.093843047 0.0187686095
#> 7   165.3333  16 0.122773676 0.0245547353
#> 8   167.2667  17 0.126119561 0.0252239121
#> 9   178.7500  18 0.139655362 0.0279310724
#> 10 189.0000  19 0.145341746 0.0290683492
```

We've subsetted on fish "> age-9" because — for White Sturgeon — females mature beginning around age-10. So to complete our dataset (i.e., probability of spawning) we need to `rbind()` ages 0-9.

We'll preserve `p_spawn` for future use and name our new dataframe `prob_spawn2` so we don't confuse with `prob_spawn` internal to `spopmodel`. In fact, it's not a bad idea to compare `prob_spawn` with `prob_spawn2`, as both datasets should be roughly the same.

```
prob_spawn2 <- rbind(
  data.frame(Age = 0:9, Prob = 0, Err = 0),
  p_spawn[, c("Age", "Prob", "Err")]
)
```

```
prob_spawn2
#>    Age        Prob          Err
#> 1    0 0.000000000 0.0000000000
#> 2    1 0.000000000 0.0000000000
#> 3    2 0.000000000 0.0000000000
#> 4    3 0.000000000 0.0000000000
#> 5    4 0.000000000 0.0000000000
#> 6    5 0.000000000 0.0000000000
#> 7    6 0.000000000 0.0000000000
#> 8    7 0.000000000 0.0000000000
#> 9    8 0.000000000 0.0000000000
#> 10   9 0.000000000 0.0000000000
#> 11  10 0.004043737 0.0008087475
#> 12  11 0.012151673 0.0024303346
#> 13  12 0.023842223 0.0047684447
#> 14  13 0.048751695 0.0097503390
#> 15  14 0.070837466 0.0141674932
#> 16  15 0.093843047 0.0187686095
#> 17  16 0.122773676 0.0245547353
```

```
#> 18  17 0.126119561 0.0252239121
#> 19  18 0.139655362 0.0279310724
#> 20  19 0.145341746 0.0290683492
```

Age Distribution: age-0

Now we return to age distribution. Estimating age-0 frequency is a bit more involved. We need first to calculate mean length at age, given our age and length data.

Here we employ Devore's (et al. 1995) equation to calculate number of eggs based on fork length ($eggs = 0.072 \times l_i^{2.94}$, where l_i is mean length at age. Like maturity, we subset on fish > age-9.

```
eggs_female <- 0.072 * (mean_len_age[bool_mlaa_gt9, "MeanFL"]^2.94)
```

Now that we have number of eggs per spawning female, we need to multiply this by the number of females and spawning probability.

```
age0 <- sum(p_spawn[["Prob"]] * females[10:19] * eggs_female)
```

We get a value of 60,312,160 age0 fish. We will append this to our `females` varialbe. Our age distribution really should be a dataframe, so we can construct that now. We name this dataframe `age_dist2` so as not to confuse it with `age_dist` internal to `spopmodel`. (The two dataframes should be roughly the same, although you likely will find a big (and yet unexplained) difference between age-0 values.)

```
age_dist2 <- data.frame(
  Age = as.numeric(c(0, names(females))),
  Freq = c(age0, unname(females)),
  row.names = NULL
)

age_dist2
#>    Age         Freq
#> 1    0 60312160.028
#> 2    1     4368.617
#> 3    2     3699.198
#> 4    3      672.000
#> 5    4      912.000
#> 6    5     1680.000
#> 7    6     2640.000
#> 8    7     4152.000
#> 9    8     3816.000
#> 10   9     3552.000
```

```
#> 11  10     1752.000
#> 12  11      888.000
#> 13  12      600.000
#> 14  13     1296.000
#> 15  14      528.000
#> 16  15      768.000
#> 17  16       72.000
#> 18  17      360.000
#> 19  18       96.000
#> 20  19      216.000
```

Egg Count

We need age-specific egg count. Absent this, the process to generate such data is similar to the one that creates spawning probabilty. Here we use `EggCount()`, which runs linear regression on Devore (et al. 1995) fecundity data (here in named `FecundityWST`) and then predicts fecundity given mean-length-at-age.

Age, we `rbind()` ages-0 through -9 data as 0, given maturity begins around age-10. We assign this to variable `num_eggs` so as not to confused with `numbers_eggs` interal to `spopmodle`.

Note: the use of linear regression is peculiar given Devore's egg count equation (above). We are following model protocol, but you may want to experiment by plugging mean-length-at-age directly into Devore's equation.

```
# egg count age-10 to age-19
egg_count <- with(data = mean_len_age[bool_mlaa_gt9, ], expr = {
  EggCount(len = MeanFL, age = Age)
})
#> Warning: Using `FecundityWST` for `numEggs`.

num_eggs <- rbind(
  data.frame(Age = 0:9, Count = 0, Err = 0),
  egg_count[, c("Age", "Count", "Err")]
)

num_eggs
#>    Age     Count       Err
#> 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
#> 6    5      0.00     0.000
#> 7    6      0.00     0.000
#> 8    7      0.00     0.000
#> 9    8      0.00     0.000
```

```
#> 10   9      0.00     0.000
#> 11  10  29800.54 11238.621
#> 12  11  80725.15  8901.094
#> 13  12 114277.95  7572.648
#> 14  13 155425.88  6356.981
#> 15  14 182691.87  5933.044
#> 16  15 210172.02  5898.947
#> 17  16 253847.41  6646.076
#> 18  17 260799.51  6840.788
#> 19  18 302092.61  8295.396
#> 20  19 338950.74  9886.751
```

Survival Probability

Here we create the final of four data inputs: survival probability. Because exploitation (μ) affects overall survival, we need to derive survival probability for every level of μ . We do this using `SurvivalProb()`.

Currently, `SurvivalProb()` assumes a constant survival rate (`sRate`; & error `sRateErr`) for all ages supplied to `ages`. For ages subject to harvest (i.e., legal-sized fish), we subtract from survival rate (`sRate`) fishing mortality (*F*), as calculated below.

$$F = \frac{\mu \times Z}{A}$$

where $Z = -log(S)$, S is `sRate`
$A = 1 - S$

Note: the derviation for standard error of fish affected by harvest is not well understood (by me - J. DuBois, that is) at this point. We simply multiply `sRateErr` by 0.45. This will suffice for now until we (err…I) better understand the method used to derive this SE. I used 0.45 because the values in `prob_survival` are 0.04281 for non-harvestable fish ≥ age-3 and 0.01932 for legal-sized fish, 0.04281 * 0.45 is roughly 0.019.

Note: we use default values for ages 0-2. By "default" I mean values culled from literature. See S. Blackburn's Masters Thesis for references.

`SurvivalProb()` accepts a list as an argument for parameter `mu`. In fact, it's very likely you'll want to pass a list with varying `mu`s for use in simulations. Here we create a very small list (3 levels of `mu`) just for ease of demonstation. In reality, this model was developed on `mu` 0 to 0.30, by 0.01. `SurvivalProb()` names list items using "mu_" followed by the level of `mu`.

```
mus <- as.list(seq(from = 0.01, to = 0.03, by = 0.01))

prob_survival2 <- SurvivalProb(
  ages = 3:19,
  sRate = 0.946,
  sRateErr = 0.03,
```

```
  mu = mus,
  agesMu = 10:15
)

str(prob_survival2)
#> List of 3
#>  $ mu_0.01:'data.frame': 20 obs. of  3 variables:
#>   ..$ Ages: int [1:20] 0 1 2 3 4 5 6 7 8 9 ...
#>   ..$ Prob: num [1:20] 0.002 0.25 0.84 0.946 0.946 0.946 0.946 0.946 0.946 0.946 ...
#>   ..$ Err : num [1:20] 0.003 0.05 0.168 0.03 0.03 0.03 0.03 0.03 0.03 0.03 ...
#>  $ mu_0.02:'data.frame': 20 obs. of  3 variables:
#>   ..$ Ages: int [1:20] 0 1 2 3 4 5 6 7 8 9 ...
#>   ..$ Prob: num [1:20] 0.002 0.25 0.84 0.946 0.946 0.946 0.946 0.946 0.946 0.946 ...
#>   ..$ Err : num [1:20] 0.003 0.05 0.168 0.03 0.03 0.03 0.03 0.03 0.03 0.03 ...
#>  $ mu_0.03:'data.frame': 20 obs. of  3 variables:
#>   ..$ Ages: int [1:20] 0 1 2 3 4 5 6 7 8 9 ...
#>   ..$ Prob: num [1:20] 0.002 0.25 0.84 0.946 0.946 0.946 0.946 0.946 0.946 0.946 ...
#>   ..$ Err : num [1:20] 0.003 0.05 0.168 0.03 0.03 0.03 0.03 0.03 0.03 0.03 ...
```

Below is a snippet of dataframe `mu_0.01` to show ages not affected by harvest (7-9) and those affected by harvest (10-12).

Note: Per model design (and mostly due to data gaps) we use (1) constant survival rate for ages 3-19, less of course fishing moratility and (2) constant standard error throughout, irrespective of mu. It might be an interesting exercise to vary age-specfic survival, perhaps at least for older fish.

```
# just a snippet for display
prob_survival2$mu_0.01[8:13, ]
#>    Ages      Prob    Err
#> 8     7 0.9460000 0.0300
#> 9     8 0.9460000 0.0300
#> 10    9 0.9460000 0.0300
#> 11   10 0.9357199 0.0135
#> 12   11 0.9357199 0.0135
#> 13   12 0.9357199 0.0135
```

References

LEFT OFF HERE 24-Oct-2018