STADEM Vignette

Kevin E. See

2017-11-27

Contents

Introduction **1**

System requirements **1**

Data sources **2**

High-level overview **2**

Compiling data **4**

Run STADEM **5**

Suggestions 5

STADEM output **6**

Additional options **12**

References **12**

Introduction

The STADEM package was developed with the goal of estimating total adult escapement of spring/summer Chinook salmon and steelhead that cross Lower Granite dam (LGD). In addition, to meet desired management and research objectives, total escapement has to include estimates of uncertainty and be parsed into weekly strata by three origin groups; wild, hatchery and hatchery no-clip. To reach this goal, we have developed the **ST**ate space **A**dult **D**am **E**scapement **M**odel (STADEM) model that incorporates fish ladder window counts, data from sampled fish at the LGD adult trap, and observations of previously PIT tagged fish at LGD adult detection sites.

Some of the data needed for STADEM is available at other dams, and the package developers are currently working to develop the ability to query all of the necessary data at other locations. Currently however, the focus remains on Lower Granite dam, the furthest upstream dam returning salmonids encounter on the journey up the Snake River. The following example will show how to run STADEM at Lower Granite for one species and one year, and what some of the output looks like.

System requirements

STADEM relies on the following R packages which can be downloaded via CRAN or by using the function `install.packages()`:

- `dplyr`, `lubridate`, `httr`: can all be installed by installing the `tidyverse` package
- `rjags`
- `jagsUI`
- `boot`

In addition, STADEM requires the JAGS software (Just Another Gibbs Sampler). Please download version >= 4.0.0 via the former link.

Data sources

STADEM relies on several pieces of data, which must be compiled from multiple sources. Many of them are accessed through the Columbia Basin Research Data Access in Real Time (DART) website.

- **Window counts** are available through DART for many dams within the Columbia River basin.
- **Trap data** comes from an adult fish trap. This provides biological information (e.g., origin, genetic stock, age, sex) to allow the decomposition of total escapement into specific groups, as well as a secondary estimate of total escapement, if the trap rate is known or can be reliably estimated.
- **PIT-tag data** comes from fish that were previously PIT tagged, either as juveniles or as adults at one of the dams downstream of Lower Granite. These fish provide information about the proportion of the run that crosses the dam at night, when the counting window is closed, as well as the proportion of the run that has fallen back and re-ascended the dam. This data is also available through DART.

High-level overview

STADEM estimates the total number of fish crossing the dam each week, based on two major data sources: the window counts and the total fish in the trap, while also accounting for two known biological processes: night-time passage and fallback-and-reacension. Using a state-space approach, STADEM assumes that the window counts and the estimates from the trap (fish in the trap divided by trap rate that week) are generated by processes with observation error. In the case of the window counts, there is some inherent error in the counting process, and it fails to account for fish that cross the dam while the window is closed. In the case of the trap, there is sampling variation and uncertainty in what the realized trap rate is. In addition, STADEM accounts for potential double-counting of fish that have fallen back and re-ascended the dam. It then partitions the estimate of total fish over the dam by origin, to provide the needed data for management goals (Figure 1).

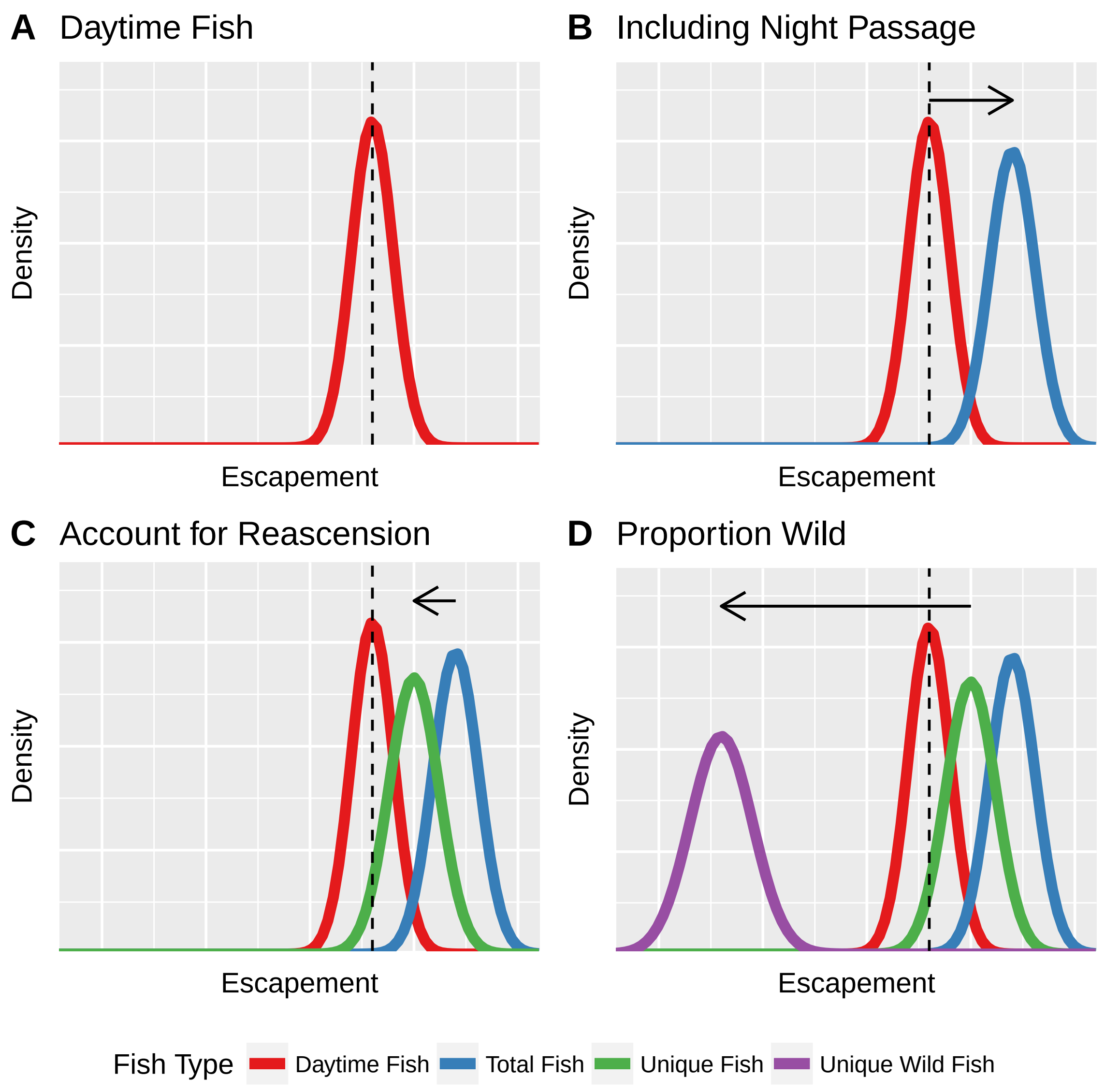


Figure 1: Figure 1: Schematic of how the STADEM model works. Panel A shows the posterior of the estimate of fish crossing while the window is open (dashed line shows observed window counts). That estimate is divided by the nighttime passage rate (B). The total fish is then discounted by the reascension rate to estimate unique fish (C). Those unique fish are then multiplied by the proportion of wild fish (D), to estimate unique wild fish.

Compiling data

The STADEM package relies on many functions from two other packages, `tidyverse` for many data manipulation functions and `lubridate` for dates and intervals. It also depends upon the `jagsUI` package to run the JAGS program from within R.

```
library(tidyverse)
library(lubridate)
library(STADEM)
library(jagsUI)

# to make processing of results easier
library(stringr)
```

The STADEM package makes it easy to compile all the necessary data in one convenient function, `compileGRAdata`. The user provides the spawn year and species (either "Chinook" or "Steelhead") they are interested in. STADEM operates on a weekly time-step, and the user has the option to determine the day of the week that most strata will begin on, using the `strata_beg` arguement. There are periodic messages describing what is being done within `compileGRAdata`. Within `compileGRAdata`, several internal functions are being called, which may be utilized with the STADEM package for other purposes. They include:

- `getWindowCounts`: query DART for window counts at various dams
- `queryPITtagData`: query DART for data about PIT tags exhibiting night-time passage and fall-back/reascension behavior
- `weeklyStrata`: divide season into weekly strata, with the user defining the day of the week each strata should begin with
- `summariseLGRtrapDaily`: summarise on a daily time-step the .csv file contained within the Lower Granite Dam trap database
- `tagTrapRate`: estimate the fish trap rate based on proportion of PIT tags known to have crossed the dam were caught in the adult trap
- `queryTrapRate`: query DART for intended and realized trap rates, based on time the trap is open

```
# what spawning year? (2010 - 2016)
yr = 2014

# what species?
spp = c('Chinook', 'Steelhead')[1]

# pull together all data
stadem_list = compileGRAdata(yr = yr,
                             spp = spp,
                             strata_beg = 'Mon')
```

The `compileGRAdata()` function returns several pieces of information, consolidated into a named list we have called `stadem_list`:

- `weekStrata`: weekly strata for STADEM, which are `interval` objects from the `lubridate` package.
- `trapData`: data from adult fish trap.
- `dailyData`: data.frame of data, summarised by day.
- `weeklyData`: data.frame of data, summarised by weekly strata.

To run STADEM, only `weeklyData` is needed. STADEM also includes a function to transform all relevant data from the weekly summary to a list ready to be passed to JAGS.

```
# compile everything into a list to pass to JAGS
jags_data_list = prepJAGS(stadem_list[['weeklyData']])
```

Run STADEM

Part of the function `runJAGSmodel` writes the JAGS model as a text file. This requires a filename, and the type of statistical distribution the user would like to use to model the window counts. The options are Poisson (`pois`), negative binomial (`neg_bin`), a more flexible type of negative binomial, described in Lindén and Mäntyniemi (2011) (`neg_bin2`), quasi-Poisson (`quasi_pois`), or as a normal distribution in log-space (`log_space`). Once those have been set, use the `runSTADEMmodel` function to run the model in JAGS. Some of the inputs to this function are:

- `file_name` name of text file to write the JAGS model to (should end in .txt)
- `mcmc_chainlength` total length of MCMC chain
- `mcmc_burn` length of burn-in period for MCMC chain
- `mcmc_thin` thinning rate for with samples of MCMC chain to keep
- `mcmc_chains` how many independent chains to run
- `jags_data` list of data compiled for JAGS, returned by `prepJAGS` function
- `seed` input to `set.seed` function to make the results exactly reproducible
- `weekly_params` Should weekly estimates of escapement be saved, or only season-wide totals?
- `win_model` statistical distribution used to model the window counts

Suggestions

Recommended MCMC parameters are:

- `mcmc_chains`: 4
- `mcmc_chainLength`: 40,000
- `mcmc_burn`: 10,000
- `mcmc_thin`: 30

This provides a sample of 4,000 draws from the posterior distribution. Through trial and error, we have also determined the appropriate burn-in length and thinning interval to meet MCMC posterior checks.

We also recommend using the negative binomial distribution (`win_model = neg_bin`) to model the window counts. In our experience, all options other than the Poisson distribution provide similar estimates, with similar estimates of uncertainty. The Poisson distribution does not allow for the possibility of overdispersion in the window count data, leading to smaller uncertainty estimates which may not be appropriate. However, we encourage investigation of how different distribution choices may affect estimates for particular datasets, and for users to critically consider the appropriate modeling choice.

```r
# name of JAGS model to write
model_file_nm = 'STADEM_JAGS_model.txt'

# what distribution to use for window counts?
win_model = c('pois', 'neg_bin', 'neg_bin2', 'quasi_pois', 'log_space')[2]

#-----------------------------------------------------------------
# run STADEM model
#-----------------------------------------------------------------
mod = runSTADEMmodel(file_name = model_file_nm,
                     mcmc_chainLength = 40000,
                     mcmc_burn = 10000,
                     mcmc_thin = 30,
                     mcmc_chains = 4,
                     jags_data = jags_data_list,
                     seed = 5,
                     weekly_params = T,
                     win_model = win_model)
```

STADEM output

The JAGS object returned by `runSTADEMmodel` contains many parameter estimates. Some of the most important are the total escapement of various fish.

```
mod$summary[grep('X.tot', rownames(mod$summary)),] %>%
  pander::pander()
```

Table 1: Table continues below

	mean	sd	2.5%	25%	50%	75%	97.5%
X.tot.all	98504	3712	91521	95990	98317	100926	106056
X.tot.day	91642	3443	85149	89284	91434	93854	98724
X.tot.night	6863	579.6	5852	6474	6816	7206	8126
X.tot.reasc	9437	863.1	7968	8835	9364	9951	11355
X.tot.new.all	89068	3473	82546	86720	88932	91369	96152
X.tot.new.wild	25123	1073	23168	24384	25104	25813	27426
X.tot.new.hatch	55646	2367	51209	54025	55563	57226	60338
X.tot.new.hnc	8298	428.5	7515	8003	8279	8572	9182
X.tot.night.wild	1748	159.3	1483	1644	1736	1836	2088
X.tot.reasc.wild	2851	319.9	2327	2625	2813	3036	3596

	Rhat	n.eff	overlap0	f
X.tot.all	1.001	1795	0	1
X.tot.day	1.001	1885	0	1
X.tot.night	1.004	975	0	1
X.tot.reasc	1.002	1845	0	1
X.tot.new.all	1.001	2102	0	1
X.tot.new.wild	1.002	1087	0	1
X.tot.new.hatch	1	4000	0	1
X.tot.new.hnc	1.001	1822	0	1
X.tot.night.wild	1.005	913	0	1
X.tot.reasc.wild	1.001	3323	0	1

Another table that summarises some of the output by week can be found by using the code below.

```
week_df = mod$summary[grepl('^X.all\\[', rownames(mod$summary)) |
                         grepl('^X.day\\[', rownames(mod$summary)) |
                         grepl('^X.night\\[', rownames(mod$summary)) |
                         grepl('^X.reasc\\[', rownames(mod$summary)) |
                         grepl('X.new.tot\\[', rownames(mod$summary)),] %>%
  as.data.frame() %>%
  mutate(var = rownames(.),
         week = as.integer(str_extract(var, "[0-9]+")),
         param = str_extract_all(var, "[:alpha:]+", simplify = T)[,2]) %>%
  tbl_df() %>%
  mutate(param = recode(param,
                        'all' = 'Total',
                        'day' = 'Day',
                        'night' = 'Night',
                        'reasc' = 'Reascension',
                        'new' = 'Unique')) %>%
  select(param, week, mean, sd)

# table with point estimates only
pt_est_tab = week_df %>%
  select(-sd) %>%
  mutate(mean = round(mean)) %>%
  spread(param, mean) %>%
  left_join(stadem_list[['weeklyData']] %>%
              filter(window_open | trap_open) %>%
              mutate(week = 1:n())) %>%
  select(Week = week,
         Win.Cnt = win_cnt,
         Total,
         Day,
         Night,
         Reascension,
         Unique)

# table with point estimates and standard errors
est_tab = week_df %>%
  mutate(prnt_val = paste0(round(mean), ' (', round(sd, 1), ')')) %>%
  select(-sd, -mean) %>%
  spread(param, prnt_val) %>%
  left_join(stadem_list[['weeklyData']] %>%
              filter(window_open | trap_open) %>%
              mutate(week = 1:n())) %>%
  select(Week = week,
         Win.Cnt = win_cnt,
         Total,
         Day,
         Night,
         Reascension,
         Unique)

est_tab %>%
  pander::pander(big.mark = ',')
```

Table 3: Table continues below

Week	Win.Cnt	Total	Day	Night	Reascension
1	0	2 (1.9)	2 (1.8)	0 (0.3)	2 (1.5)
2	0	2 (1.8)	2 (1.8)	0 (0.2)	1 (1.4)
3	0	2 (1.5)	2 (1.5)	0 (0.1)	1 (1.2)
4	1	2 (1.1)	2 (1.1)	0 (0.2)	1 (0.9)
5	4	5 (2.1)	5 (2)	0 (0.4)	3 (1.8)
6	94	86 (17.9)	82 (16.9)	4 (2.8)	51 (19.8)
7	423	433 (84.6)	411 (79.9)	22 (10.8)	186 (64.1)
8	2,363	2729 (511.5)	2584 (482.7)	144 (51.8)	405 (123.1)

Week	Win.Cnt	Total	Day	Night	Reascension
9	19,408	16753 (1812.2)	15812 (1718)	940 (175.5)	1473 (246.5)
10	26,430	24543 (2316.2)	22718 (2138.9)	1825 (276.9)	1075 (206.9)
11	12,276	13896 (1340.3)	12932 (1248.7)	964 (161.2)	1435 (223.3)
12	6,147	8375 (1307)	7596 (1187.9)	779 (165.2)	1075 (216.1)
13	5,492	5939 (647.1)	5385 (590.4)	554 (103.3)	665 (117.7)
14	4,883	5790 (627.1)	5387 (586)	403 (77.3)	514 (99.6)
15	4,335	5284 (679.3)	5005 (645.9)	279 (65.8)	397 (90.2)
16	3,278	4441 (629.1)	4216 (599.7)	225 (56.6)	363 (85.6)
17	2,613	3339 (456.7)	3122 (428.4)	217 (56.4)	334 (79.9)
18	2,728	2993 (621.3)	2799 (574.7)	194 (95.8)	396 (225.3)
19	1,671	1853 (377.2)	1719 (342.2)	134 (101.5)	369 (274.9)
20	853	951 (210.4)	875 (183.8)	76 (74.4)	251 (188.2)
21	273	329 (89)	299 (73.5)	30 (41.4)	109 (74.9)
22	305	415 (107.5)	375 (92.2)	40 (51.5)	166 (100.1)
23	258	342 (69.2)	311 (61.3)	31 (33.5)	163 (73.4)

Unique
1 (1.2)
1 (1.1)
1 (0.8)
1 (0.7)
2 (1.4)
35 (17.7)
246 (71.7)
2324 (449)
15279 (1663.1)
23468 (2222.6)
12462 (1211)
7300 (1147.3)
5275 (581.8)
5275 (579.7)
4887 (634.7)
4077 (582.9)
3006 (416.8)
2597 (579.8)
1484 (407.1)
701 (239.4)
220 (93.8)
249 (110.6)
179 (75.8)

A user might also like to make time-series plots of estimates, to compare with window counts and/or trap estimates (Figure 2).

```r
week_est = mod$summary[grep('^X.all', rownames(mod$summary)),] %>%
  as.data.frame() %>%
  mutate(var = rownames(.),
         week = as.integer(str_extract(var, "[0-9]+")),
         param = str_extract_all(var, "[:alpha:]+", simplify = T)[,3],
         param = ifelse(param == '', 'all', param)) %>%
  tbl_df() %>%
  select(var, param, week, everything()) %>%
  left_join(stadem_list[['weeklyData']] %>%
              filter(window_open | trap_open) %>%
              mutate(week = 1:n()))
```

```r
# plot time-series of model estimates, window counts and trap estimates
week_est %>%
  filter(param == 'all') %>%
  ggplot(aes(x = Start_Date,
             y = `50%`)) +
  geom_ribbon(aes(ymin = `2.5%`,
                  ymax = `97.5%`),
              alpha = 0.2) +
  geom_line(aes(y = win_cnt / (day_tags / tot_tags),
                color = 'Window (adj)')) +
  geom_point(aes(y = win_cnt / (day_tags / tot_tags),
                 color = 'Window (adj)')) +
  geom_line(aes(y = win_cnt,
                color = 'Window (raw)')) +
  geom_point(aes(y = win_cnt,
                 color = 'Window (raw)')) +
  geom_line(aes(y = trap_est,
                color = 'Trap')) +
  geom_point(aes(y = trap_est,
                 color = 'Trap')) +
  geom_line(aes(color = 'Model')) +
  geom_point(aes(color = 'Model')) +
  scale_color_manual(values = c('Model' = 'black',
                                'Window (raw)' = 'lightblue',
                                'Window (adj)' = 'blue',
                                'Trap' = 'red')) +
  theme_bw() +
  theme(legend.position = 'bottom') +
  labs(x = 'Date',
       y = 'Estimate',
       color = 'Source',
       title = paste('All', spp, 'in', yr))
```

It's also possible to examine estimates of night-time passage or re-ascension rates. The plot below shows the observed values as points, the estimated rates as lines, with 95% credible intervals around them.

```r
rate_est = mod$summary[grepl('^day.true', rownames(mod$summary)) |
                         grepl('^reasc.true', rownames(mod$summary)),] %>%
  as.data.frame() %>%
  mutate(var = rownames(.),
         week = as.integer(str_extract(var, "[0-9]+")),
         param = str_extract_all(var, "[:alpha:]+", simplify = T)[,1]) %>%
  tbl_df() %>%
  select(var, param, week, everything()) %>%
  left_join(stadem_list[['weeklyData']] %>%
              filter(window_open | trap_open) %>%
              mutate(week = 1:n()))
```

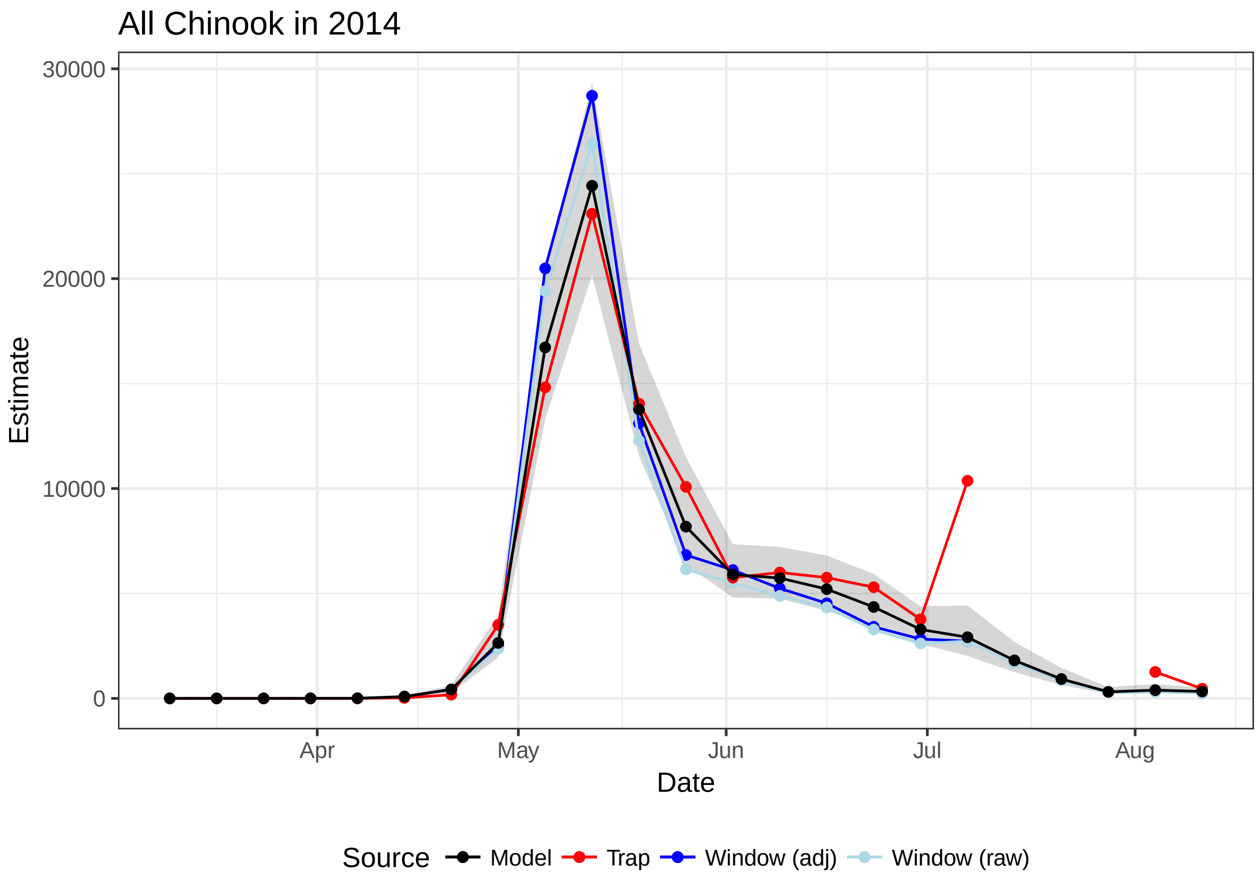
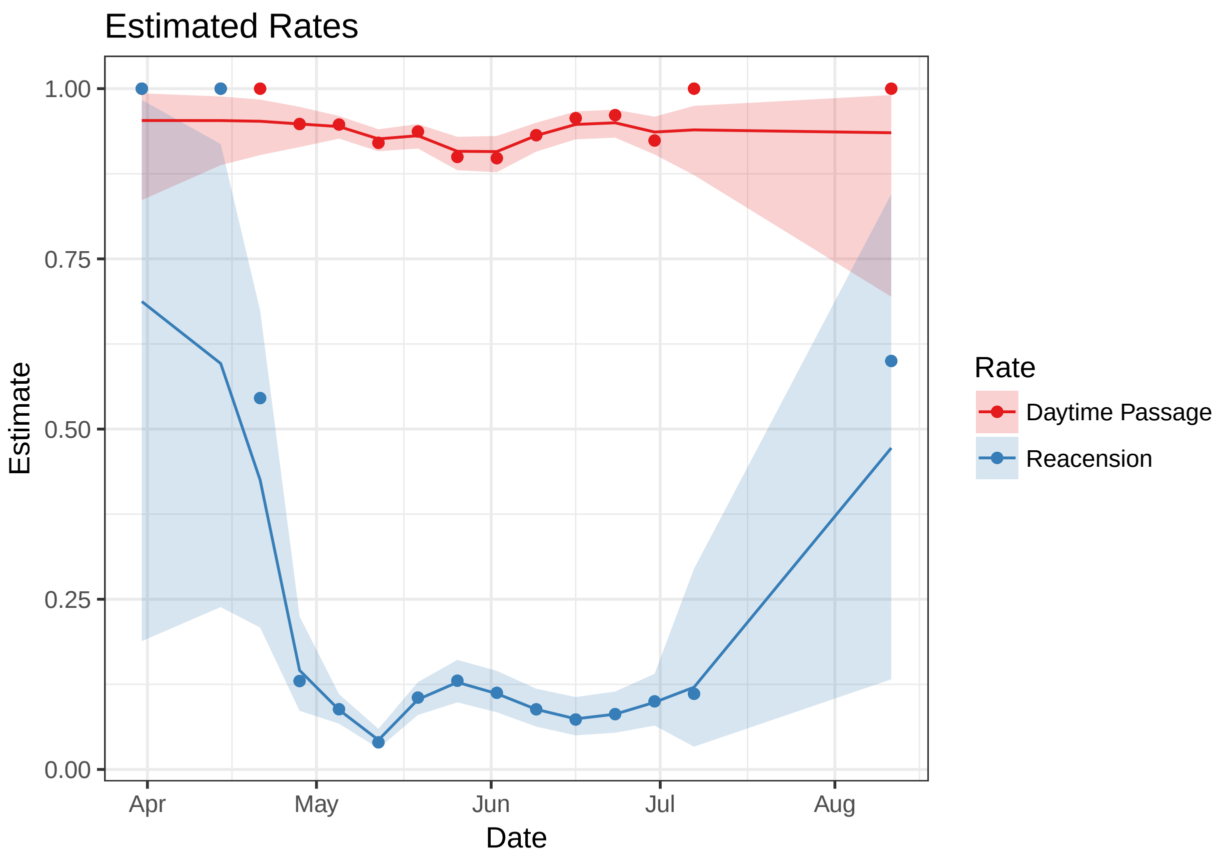


Figure 2: Figure 2: Time-series plot showing estimates of total escapement for Chinook in 2014 , including raw window counts, window counts adjusted for nighttime passage, trap estimates and STADEM estimates.

```
rate_est %>%
  filter(tot_tags > 0) %>%
  ggplot(aes(x = Start_Date,
             y = `50%`)) +
  geom_ribbon(aes(ymin = `2.5%`,
                  ymax = `97.5%`,
                  fill = param),
              alpha = 0.2) +
  geom_line(aes(color = param)) +
  geom_point(aes(y = day_tags / tot_tags,
                 color = 'day')) +
  geom_point(aes(y = reascent_tags / tot_tags,
                 color = 'reasc')) +
  theme_bw() +
  scale_color_brewer(palette = 'Set1',
                     name = 'Rate',
                     labels = c('day' = 'Daytime Passage',
                                'reasc' = 'Reacension')) +
  scale_fill_brewer(palette = 'Set1',
                    name = 'Rate',
                    labels = c('day' = 'Daytime Passage',
                               'reasc' = 'Reacension')) +
  labs(y = 'Estimate',
       x = 'Date',
       title = 'Estimated Rates')
```



Additional options

The user does have the option to not use fish caught in the trap as a secondary estimate of escapement. To turn this feature off, set the argument `trap_est` equal to `FALSE` in the `runSTADEMmodel` function. This will constrain the statistical distribution used to model the window counts as Poisson, because there is no other way to estimate the variance in those counts.

If the user would like to summarise STADEM estimates of total unique escapement, by week, (e.g. as an input to the `SCOBI` pacakge), there is a function to do that, `STADEMtoSCOBI`.

```
scobi_input = STADEMtoSCBOI(stadem_mod = mod,
                            lgr_weekly = stadem_list[['weeklyData']])

head(scobi_input)
```

```
## # A tibble: 6 x 6
##   model_week  StartDate Strata Estimate         SE Collapse
##        <int>     <date>  <dbl>    <dbl>      <dbl>    <lgl>
## 1          1 2014-03-10     10        1  1.2409201       NA
## 2          2 2014-03-17     11        1  1.1140242       NA
## 3          3 2014-03-24     12        1  0.8447712       NA
## 4          4 2014-03-31     13        1  0.7439236       NA
## 5          5 2014-04-07     14        2  1.4422442       NA
## 6          6 2014-04-14     15       35 17.6787653       NA
```

References

Lindén, Andreas, and Samu Mäntyniemi. 2011. "Using the Negative Binomial Distribution to Model Overdispersion in Ecological Count Data." *Ecology* 92 (4). Ecologial Society of America: 1414–21.