PITcleanr & DABOM User Manual for Upper Columbia Steelhead

Kevin See[1,*], and Mike Ackerman[1]

05 June, 2020

Abstract

This manual contains instructions on how to run the DABOM model to estimate adult abundace for steelhead to locations in the Upper Columbia River basin. We start by describing how to generate a list of valid PIT tags at Priest Rapids Dam to be used in the model and then how to query for detections of those PIT tags using PTAGIS. We then "clean up" the detections using the R package PITcleanr. Finally we describe how to write the JAGS model for use in DABOM, and finally, run DABOM to estimate transition probabilities for steelhead throughout the Upper Columbia system. DABOM movement probabilities can then be multiplied by estimates of escapement at Priest Rapids Dam to get abundance to locations or tributaries.

Contents

[1] Biomark, Inc. 705 South 8th St., Boise, Idaho, 83702, USA

[*] Correspondence: Kevin See <Kevin.See@merck.com>

1 Introduction

This manual describes how to run the **D**am **A**dult **B**ranch **O**ccupancy **M**odel (DABOM) for steelhead crossing over Priest Rapids Dam and into the Upper Columbia River. We start by describing how to query PTAGIS to get all detections of adults at relevant observation sites (e.g., weirs, PIT tag arrays, etc.) for a particular spawning run from a list of "valid" PIT tags. Observation data are then "cleaned up" using the `PITcleanr` R package to determine a final destination or spawning location for each individual and detection data are prepared for use in the `DABOM` R package and model. Next, we describe how to write a JAGS model for use in `DABOM`, and finally, run `DABOM` to estimate detection and movement probabilities in the Upper Columbia River system. Movement probabilities can then be multiplied by an estimate of adult escapement at Priest Rapids Dam to estimate escapement, with uncertainty, at any observation site (or tributary) within the Upper Columbia River.

2 Set-up

2.1 Software

The first step is to ensure that all appropriate software and R packages are installed on your computer. (R) is a language and environment for statistical computing and graphics and is the workhorse for running all of the code and models described here. R packages are collections of functions and data sets developed by the R community for particular tasks. Some R packages used here are available from the general R community (Available CRAN Packages) whereas others (e.g., `PITcleanr`, `DABOM`) are developed by (Kevin See) and contain functions written for cleaning and analysis of PIT tag detection site and observation data.

First, you will need to have R downloaded and installed. Use the "base" distribution and all default installation settings should work just fine. Additionally, although not necessary, we find it very useful to use RStudio as an interface for R. Download the Desktop version of RStudio, and again, default installation settings should work just fine. RStudio provides a graphical user interface (GUI) for R with a text/code editor and allows for direct code execution, management of R packages, a viewing of R objects (e.g., data) in the environment.

Next, you will also need the JAGS software to run DABOM. You can download that from SourceForge. JAGS (Just Another Gibbs Sampler) software is used by `DABOM` for Bayesian inference.

2.2 R Packages

After installing R and Studio, you will also need to install `tidyverse`, a series of R packages that work together for data science (i.e. data cleaning and manipulation), as well as the `jagsUI` package to interface with JAGS. To save some results to Excel files, we use the `WriteXLS` pacakge. The `tidyverse`, `jagsUI` and `WriteXLS` packages are all available from the R community and can be installed by typing the following into your R console:

```r
install.packages("tidyverse")
install.packages("jagsUI")
install.packages('WriteXLS')
```

Next, install `PITcleanr` and `DABOM` from Kevin See's GitHub page here. `PITcleanr` was written primarily to build a "river network" describing the relationships among detection sites in a system, to clean PIT tag detection data to establish capture histories for individuals, and to determine the final destination or spawning location for each fish. `DABOM` is used for writing and running the `DABOM` model and estimating detection and movement probabilities. You can use `devtools` to install both of these packages from GitHub using the following:

```r
install.packages("devtools")
devtools::install_github("KevinSee/PITcleanr")
devtools::install_github("KevinSee/DABOM")
```

Hint: We have experienced errors installing the PITcleanr and DABOM packages related to *"Error: (converted from warning) package 'packagenamehere' was built under R version x.x.x"*. Setting the following environment variable typically suppresses the error and allows you to successfully install the packages.

```
Sys.setenv(R_REMOTES_NO_ERRORS_FROM_WARNINGS = TRUE)
```

When attempting to install PITcleanr or DABOM you may receive an error message similar to *"there is no package called 'ggraph'"*. In that case, try to install the given package using the following and then attempt to install PITclean or DABOM, again.

```
install.package("ggraph") # to install package from R cran
# replace ggraph with the appropriate package name as needed
```

We are always trying to improve the PITcleanr and DABOM R packages to minimize these types of errors.

2.3 devtools Note

IF THE devtools PACKAGE WORKS FINE ABOVE, SKIP THIS SECTION. To use devtools, you may have to download and install Rtools. You can try to use devtools without Rtools, initially, and if PITcleanr and DABOM fail to install correctly, installing Rtools may remedy the situation. We have had mixed results with this in the past.

3 Procedure

Briefly, the steps to process the data for a given run or spawn year include:

1. Generate valid PIT tag list
2. Query PTAGIS for detections
3. Develop the "river network" describing the relationship among detection sites
4. Use PITcleanr to "clean up" detection data
5. Review PITcleanr output to determine final capture histories
6. Run DABOM to estimate detection and movement probabilities
7. Summarise DABOM results
8. Combine DABOM movement probabilities with an estimate of adult escapement at Priest Rapids

Following, we describe each of the steps in detail using the 2018/2019 steelhead run as an example.

4 Valid PIT Tag List and PTAGIS Query

4.1 Tag Information

You will need to compile a list of PIT tags in steelhead that were caught in the Priest Rapids trap for a given run or spawn year, called the valid tag list. Tagged steelhead need to be a random, representative sample of the run, and so should only include tags from the trap. If a fish is caught in the trap and happens to be previously tagged, that tag can be used as part of the valid tag list. However, if a previously tagged fish (e.g. a fish tagged as a juvenile in the upper Methow) is detected crossing Priest Rapids, but is not caught in the trap, it cannot be used for this analysis.

Save this list of valid PIT tag codes as a text file with no headers, to make it easy to upload to a PTAGIS query.

This is also a good opportunity to compile other relevant biological or life history information for each fish in the valid PIT tag list, such as sex, length, weight, age, origin, genetics, etc. That information may be used later to estimate, for example, sex- or age-specific abundance to locations which is useful for productivity monitoring.

```r
library(tidyverse)
library(lubridate) # for dealing with dates, is installed with tidyverse
library(readxl)

# which spawn year are we dealing with?
yr = 2019

# read in biological data from trap
bio_df = read_excel('../data/raw_data/WDFW/BY19 BioData.xlsx',
                       1) %>%
  mutate(BroodYear = "BY19") %>%
  mutate(record_id = 1:n()) %>%
  mutate(Year = paste0("20", str_remove(BroodYear, "^BY")),
         Year = as.numeric(Year)) %>%
  gather(tag_loc, TagID, matches('^PIT')) %>%
  filter(!is.na(TagID)) %>%
  select(record_id,
         BroodYear,
         Year,
         tag_loc,
         TagID,
         Species = `Species(final)`,
         TrapDate = SurveyDate,
         Sex = `Sex(final)`,
         Origin = `Origin(final)`,
         ForkLength,
         Age = `Age (scales)`,
         FinalAge,
         AdClip = `Ad-clip`,
         CWT = `CWT (Sn)`) %>%
  mutate(Age = str_replace(Age, '^r', 'R'))

# pull out PIT tag numbers and save as a text file
bio_df %>%
  filter(!is.na(TagID)) %>%
  select(TagID) %>%
  distinct() %>%
  write_delim(path = paste0('examp_data/UC_Sthd_Tags_', yr, '.txt'),
              delim = '\n',
              col_names = F)
```

4.2 PTAGIS Query

The next step is to query PTAGIS for all detections of the fish included on the valid tag list. PTAGIS is the regional database for fish marked with PIT tags by fisheries management agencies and research organizations in the Columbia River Basin. There, go to the Advanced Reporting page, which can also be found under the Data tab on the homepage. To access Advanced Reporting, you will need a free account from PTAGIS, and to be logged in. Once on the Advanced Reporting page, select "Launch" and create your own query by selecting "Create Query Builder2 Report". We will use a "Complete Tag History" query.

You will see several query indices on the left side of the query builder, but for the purposes of `PITcleanr` and `DABOM`, we only need to deal with a couple of those. First, under "1 Select Attributes" the following fields are required to work with `PITcleanr`:

- Tag
- Mark Rear Type
- Event Type
- Event Site Code
- Event Date Time
- Event Release Date Time
- Antenna
- Antenna Group Configuration

You are welcome to include other fields as well, but the ones listed above must be added. Any additional fields will just be included as extra columns in your query output.

The only other required index is "2 Select Metrics", but that can remain as the default, "CTH Count", which provides one record for each event recorded per tag.

Set up a filter for specific tags (e.g. the valid tag list) by next navigating to the "28 Tag Code - List or Text File" on the left. And then, after selecting "Tag" under "Attributes:", you should be able to click on "Import file…". Simply upload the .txt file you saved in the previous step containing tag codes in the valid tag list. Under "Report Message Name:" near the bottom, name the query something appropriate, such as "PTAGIS_2018_19", and select "Run Report". Once the query has successfully completed, export the output as a .csv file (e.g. "PTAGIS_2018_19.csv") using the default settings:

- Export: Whole report
- CSV file format
- Export Report Title: unchecked
- Export filter details: unchecked
- Remove extra column: Yes

5 PITcleanr

5.1 Processing PTAGIS Detections

The next step is to clean up all the detections listed in the PTAGIS query. Those include every detection on every antenna; we need to condense those to a single detection for each particular array of antennas, even if the fish was detected 7 times on 3 different antennas in that array. We can use the `PITcleanr` package for this. There are a few parts to this particular step. But first, load the appropriate packages into your R environment.

```r
library(PITcleanr)
library(tidyverse)
```

5.1.1 Build Site Configuration

The first necessary step to "cleaning up" or processing the detections is to define which sites we are going to include in the DABOM model. The `PITcleanr` package contains a function, specific to Priest Rapids Dam and the Upper Columbia, to do this, called `writePRDNodeNetwork()`. This function lists all the various sites by their PTAGIS site ID, and shows which other sites a tag would need to pass in order to reach that particular site. The below saves a new object `site_df` contaning that information. If desired, you can use the `view()` function to review `site_df` in a new RStudio window.

```r
site_df = writePRDNodeNetwork()
site_df
#> # A tibble: 76 x 11
#>    SiteID  path          Step1 Step2   Step3  Step4 Step5 Step6 Step7 Step8 Step9
#>    <fct>   <chr>         <chr> <chr>   <chr>  <chr> <chr> <chr> <chr> <chr> <chr>
#>  1 PRA     PRA           PRA   ""      ""     ""    ""    ""    ""    ""    ""
#>  2 BelowJ~ PRA.BelowPr~  PRA   "Below~ "Belo~ ""    ""    ""    ""    ""    ""
#>  3 JD1     PRA.BelowPr~ PRA   "Below~ "JD1"  ""    ""    ""    ""    ""    ""
#>  4 TMF     PRA.BelowPr~ PRA   "Below~ "TMF"  ""    ""    ""    ""    ""    ""
#>  5 ICH     PRA.BelowPr~ PRA   "Below~ "ICH"  ""    ""    ""    ""    ""    ""
#>  6 PRH     PRA.BelowPr~ PRA   "Below~ "PRH"  ""    ""    ""    ""    ""    ""
#>  7 RSH     PRA.BelowPr~ PRA   "Below~ "RSH"  ""    ""    ""    ""    ""    ""
#>  8 PRO     PRA.BelowPr~ PRA   "Below~ "PRO"  ""    ""    ""    ""    ""    ""
#>  9 PRV     PRA.BelowPr~ PRA   "Below~ "PRV"  ""    ""    ""    ""    ""    ""
#> 10 HST     PRA.BelowPr~ PRA   "Below~ "PRV"  "HST" ""    ""    ""    ""    ""
#> # ... with 66 more rows
# view(site_df)
```

The next step is to query PTAGIS for all the metadata associated with these sites. Again, `PITcleanr` includes a function to do this, `buildConfig()`, but you will need an internet connection to run this. The `buildConfig()` function returns information about each site in PTAGIS, including the site code, the various configuration codes, the antenna IDs, when that configuration started and ended (if it has), what type of site it is (interrogation, INT, or mark/recapture/recover, MRR), the site name, the antenna group each antenna is part of, and several other pieces of information. It also assigns a 'model node' to each antenna. The model nodes essentially define which array each antenna is part of within each site. If it is a single array (or perhaps an MRR site), all of the antennas will be assigned to the same model node. If there is a double array, the antennas in the downstream array will be assigned to the "B0" array, and the upstream antennas to the "A0" array. If there is a triple array, by default the middle array is grouped with the upper array, to help simplify the DABOM model structure. Defining upstream and downstream arrays and nodes are a necessary step to estimate detection probabilities at double (or triple) arrays. This file is what will link the PTAGIS detections to the DABOM model nodes.

```
org_config = buildConfig()
#> [1] "Querying INT sites' metadata"
#> [1] "Querying INT sites' configuration information"
#> [1] "Querying MRR sites' metadata"

# manually add site for Colockum Creek (not in PTAGIS)
org_config = org_config %>%
  bind_rows(tibble(SiteID = 'CLK',
                   ConfigID = 100,
                   AntennaID = 'A1',
                   Node = 'CLK',
                   ValidNode = T,
                   # making these up
                   StartDate = as.POSIXct(ymd('20100101')),
                   SiteType = 'INT',
                   SiteName = 'Colockum Creek',
                   AntennaGroup = 'Single Colockum Ck',
                   SiteDescription = 'Tempoary single antenna.',
                   SiteTypeName = 'Instream Remote Detection System',
                   RKM = '740.001',
                   RKMTotal = 741))
```

Note, the `org_config` object contains **every** INT and MRR detection site included in PTAGIS. Note that in the code above we added information about a site that was not registered in PTAGIS. You now have the opportunity to modify this configuration file however you would like, re-assigning various antennas or sites to different nodes. For this version of DABOM, we have some suggestions, such as combing some of the carcass recovery sites with upstream arrays, or grouping all of the sites at the Leavenworth National Fish Hatchery into a single node, called "LNF". Many of the modifications below help simplify the nodes throughout the network.

```r
configuration = org_config %>%
  filter(!(SiteID == 'WAN' & SiteType == 'MRR'),
         !(SiteID == 'TMF' & SiteType == 'MRR'),
         !(SiteID == 'PRO' & SiteType == 'MRR')) %>%
  mutate(Node = ifelse(SiteID %in% c('RIA', 'RRF', 'WEA', 'PRV'),
                       SiteID,
                       Node)) %>%
  mutate(Node = ifelse(SiteID == 'PRDLD1',
                       'PRA',
                       Node)) %>%
  mutate(Node = ifelse(Node == "LWE",
                       'LWEB0',
                       Node),
         Node = ifelse(SiteID %in% c('TUF', 'TUMFBY', 'TUM'),
                       'TUM',
                       Node),
         Node = ifelse(SiteID == 'LNF' & AntennaID %in% c('01', '02'),
                       'LNFA0',
                       Node),
         Node = ifelse(SiteID == 'LNF' & AntennaID %in% c('03', '04'),
                       'LNFB0',
                       Node),
         Node = ifelse(SiteID == 'LEAV',
                       'LNFA0',
                       Node),
         Node = ifelse(SiteID == 'ICL' & ConfigID == 100,
                       'ICLB0',
                       Node),
         Node = ifelse(SiteID == 'CHIWAC',
                       'CHWA0',
                       Node),
         Node = ifelse(SiteID == 'CHIWAR',
                       'CHLA0',
                       Node),
         Node = ifelse(SiteID == 'CHIKAC',
                       'CHUA0',
                       Node),
         Node = ifelse(SiteID == 'WHITER',
                       'WTLA0',
                       Node),
         Node = ifelse(SiteID == 'LWENAT',
                       'LWNA0',
                       Node),
         Node = ifelse(SiteID == 'NASONC',
                       'NALA0',
                       Node),
         # any fish seen at Dryden dam should also be seen at LWE
         Node = ifelse(SiteID == 'DRY',
                       'LWEA0',
                       Node),
         # any fish seen at Chiwawa acclimation pond gets moved to CHL
         Node = ifelse(SiteID == 'CHP',
                       'CHLA0',
                       Node),
         Node = ifelse(SiteID == 'EBO',
                       'RRF',
                       Node),
         Node = ifelse(SiteID == 'EHL' & ConfigID == 100 & AntennaID == '02',
                       'EHLB0',
                       Node),
         Node = ifelse(SiteID == 'EHL' & ConfigID == 100 & AntennaID == '01',
                       'EHLA0',
                       Node),
         Node = ifelse(SiteID == 'EHL' & ConfigID == 110 & AntennaID == '03',
                       'EHLB0',
                       Node),
         Node = ifelse(SiteID == 'EHL' & ConfigID == 110 & AntennaID %in% c('01', '02'),
                       'EHLA0',
                       Node),
         Node = ifelse(SiteID == 'WEA' & AntennaID == 'C1',
                       'WVTB0',
                       Node),
         Node = ifelse(SiteID == 'WEA' & AntennaID == 'C2',
                       'WVTA0',
                       Node),
         Node = ifelse(Node == "LMR",
                       'LMRB0',
                       Node),
         Node = ifelse(SiteID == 'LBC' & ConfigID == 100,
                       'LBCB0',
                       Node),
         Node = ifelse(SiteID == 'MRC',
                       'MRCB0',
                       Node),
         Node = ifelse(SiteID %in% c('SSC', '18N', 'MHB', 'M3R', 'MWF'),
                       'MRCA0',
                       Node),
         Node = ifelse(SiteID == 'MSH' & AntennaID %in% c('02', '03'),
                       'MSHB0',
                       Node),
         Node = ifelse(SiteID == 'MSH' & AntennaID %in% c('01'),
                       'MSHA0',
                       Node),
         Node = ifelse(SiteID == 'MSH' & AntennaID == '00',
                       'METHB0',
                       Node),
         Node = ifelse(SiteID == 'METH',
                       'METHA0',
                       Node),
         Node = ifelse(SiteID == 'LLC' & ConfigID == 100,
                       ifelse(AntennaID == 'D3',
                              'LLCB0',
                              'LLCA0'),
                       Node),
         Node = ifelse(Node == "SCP",
                       'SCPB0',
                   Node),
         Node = ifelse(Node == "OMK",
                       'OMKB0',
                       Node),
         Node = ifelse(SiteID %in% c('OFB', 'OMF'),
                       'OMKA0',
                       Node),
         Node = ifelse(SiteID == 'ZSL',
                       ifelse(grepl('Weir 3', AntennaGroup, ignore.case = T),
                              'ZSLB0',
                              'ZSLA0'),
                       Node),
         Node = ifelse(SiteID == 'SA1' & ConfigID == 110,
                       'SA1B0',
                       Node),
         Node = ifelse(SiteID == 'OKC' & ConfigID == 100,
                       'OKCB0',
                       Node),
         Node = ifelse(SiteID == 'RCT' & ConfigID == 100,
                       'RCTB0',
                       Node),
         Node = ifelse(SiteID == 'BPC' & ConfigID == 100,
                       ifelse(AntennaID %in% c('C3'),
                              'BPCB0',
                              'BPCA0'),
                       Node),
         Node = ifelse(SiteID == 'PRH' & AntennaID %in% c('F1', 'F2', 'F3', 'F4'),
                       'PRHB0',
                       Node),
         Node = ifelse((SiteID == 'PRH' & AntennaID %in% c('F5', 'F6', '01', '02')) | SiteID %in% c('DD
                       'PRHA0',
                       Node),
         Node = ifelse(SiteID == 'PRO' & SiteType == 'INT',
                       'PROB0',
                       Node),
         Node = ifelse(SiteID %in% c('CHANDL', 'SAT', 'TOP', 'SUN', 'LNR', 'ROZ', 'LMC', 'TAN') | SiteI
                       'PROA0',
                       Node),
         Node = ifelse(SiteID == 'ICH',
                       'ICHB0',
                       Node),
         Node = ifelse(grepl('522\\.', RKM) & RKMTotal > 538,
                       'ICHA0',
                       Node),
         Node = ifelse(SiteID == 'MDR',
                       'MDRB0',
                       Node),
         Node = ifelse(SiteID %in% c('LWD', 'BGM', 'NBA', 'MCD'),
                       'MDRA0',
                       Node),
         Node = ifelse(SiteID == 'HST',
                       'HSTB0',
                       Node),
         Node = ifelse(SiteID %in% c('BBT', 'COP', 'PAT'),
                       'HSTA0',
                       Node),
         Node = ifelse(SiteID == 'JD1',
                       'JD1B0',
                       Node),
         Node = ifelse(SiteID %in% c('30M', 'BR0', 'JDM', 'SJ1', 'SJ2', 'MJ1'),
                       'JD1A0',
                       Node),
         Node = ifelse(SiteID != 'JD1' & as.integer(stringr::str_split(RKM, '\\.', simplify = T)[,1]) <
                       'BelowJD1',
                       Node)) %>%
  distinct()

# correct a couple RKM values
configuration = configuration %>%
  mutate(RKM = ifelse(SiteID == 'SA1',
                      '858.041.003',
                      RKM),
         RKMTotal = ifelse(SiteID == 'SA1',
                           902,
                           RKMTotal)) %>%
  mutate(RKM = ifelse(SiteID == 'TON',
                      '858.133.001',
                      RKM),
         RKMTotal = ifelse(SiteID == 'TON',
                           992,
                           RKMTotal)) %>%
  mutate(RKM = ifelse(grepl('WVT', Node),
                      '829.001',
                      RKM),
         RKMTotal = ifelse(grepl('WVT', Node),
                           830,
                           RKMTotal))
```

5.1.2 Parent-Child Table

The next step is to build a parent-child table that describes which nodes are upstream of which nodes. In most cases, when modeling returning adults, the parent node is the first node the adult crosses when returning upstream to spawn and the child node is the next node upstream. The exception being nodes that occur outside of the Upper Columbia (e.g., JD1, ICH) to account for adults that are tagged at Priest Rapids Dam, but then later detected outside of the Upper Columbia. The `PITcleanr` function `createParentChildDf()` does this, taking as inputs the data frame of sites in our model and the configuration file we just created. The other input is the starting date (in `YYYYMMDD` format) for this model run, because the function uses that to find the appropriate configuration of the antennas. For this version, we define that starting date as June 1 of the year prior to the spawn year we are interested in.

```
# start date is June 1 of the previous year
# the paste0 function simply pastes together our 'yr' and '0601' into the 'YYYYMMDD' format
start_date = paste0(yr - 1, '0601')

# build parent-child table
parent_child = createParentChildDf(site_df,
                                   configuration,
                                 startDate = start_date)
```

5.1.3 Clean PTAGIS Data

The final step of this data cleaning process is run the PTAGIS detections through the `processCapHist_PRD()` function in `PITcleanr` which will assign each detection to a node in the model, and then collapse the output so we are left, for each tag, only one detection on a node before that tag is sighted on a different node.

First, use the `read_csv()` function to read in all of the detections we output from our PTAGIS query above. In the example below, we use the `paste0()` function and some additional code to recreate the filename of the .csv output we created above. Alternatively, you can just use `read_csv()` and the directory and filename of your .csv output to read in your detections and create an object called `observations`.

```
# get raw observations from PTAGIS
# These come from running a saved query on the list of tags to be used
observations = read_csv(paste0('../data/raw_data/PTAGIS/UC_Sthd_', yr, '_CTH.csv')) %>%
    filter(`Event Site Code Value` != "ORPHAN")

# process those observations with PITcleanr, using Upper Columbia-specific function
proc_list = processCapHist_PRD(startDate = start_date,
                               configuration = configuration,
                               parent_child = parent_child,
                               observations = observations,
                               # use this to filter out observations past June 1
                               last_obs_date = format(ymd(start_date) + years(1) + months(1), "%Y%m%d")
                               truncate = T,
                               site_df = site_df,
                               step_num = 1,
                               save_file = F)
```

The output of the `processCapHist_PRD()` function (`proc_list` in our example) is a list, containing four elements:

- *ValidPaths*: a data frame containing the path a fish would need to take to reach each detection node, including all the nodes it would need to pass along the way (regardless of whether it is detected at each). The *ValidPaths* can be accessed using `proc_list$ValidPaths` or `proc_list[[1]]`.
- *NodeOrder*: a data frame containing each node in the model, the node order (how many nodes to cross to arrive there from Priest Rapids Dam), the "path" a tag would take to get there consisting of all the nodes it could be detected at along the way, which site that node is associated with, and the RKM of that site from PTAGIS metadata. The *NodeOrder* can easily be accessed using `proc_list$NodeOrder` or `proc_list[[2]]`.
- *ValidObs*: a data frame containing all the observations to be deemed "valid". This consists of translating detections from site/antenna combinations into detections on nodes and then simplifing consecutive hits on the same node. The *ValidObs* can easily be accessed using `proc_list$ValidObs` or `proc_list[[3]]`.
- *ProcCapHist*: The "cleaned" capture histories, with a row for each detection that has been kept. It shows the tag ID, the date that tag was in the trap, the first and last observed date and time on that node, the PTAGIS site ID associated with that node, whether the tag was moving upstream or downstream (based on the previous observation), and two columns containing **ProcStatus** for "processed status". **AutoProcStatus** is `PITcleanr`'s best guess as to whether the observation on that node should be kept (`TRUE`) or discarded (`FALSE`). **UserProcStatus** is where the end user can define that for themselves. The *ProcCapHist* can be accessed using `proc_list$ProcCapHist` or `proc_list[[4]]`.

After this step, the results of *ProcCapHist* can be saved to a .txt, .csv, .xls, or .xlsx file, to be examined by a fisheries biologist. To save the results, simply change the `save_file` argument in `processCapHist_PRD()`

Table 1: Example of PITcleanr output for one PIT tag.

TagID	TrapDate	ObsDate	lastObsDate	SiteID	Node	AutoProcStatus
3DD.00779FDCBB	2018-07-11	2018-07-11 13:14:07	2018-07-11 14:32:13	PRDLD1	PRA	TRUE
3DD.00779FDCBB	2018-07-11	2018-07-14 12:44:40	2018-07-14 12:44:40	RIA	RIA	TRUE
3DD.00779FDCBB	2018-07-11	2018-07-25 14:09:19	2018-07-25 14:42:11	TUF	TUM	FALSE
3DD.00779FDCBB	2018-07-11	2019-04-23 19:07:18	2019-04-23 19:07:18	ICL	ICLB0	TRUE
3DD.00779FDCBB	2018-07-11	2019-04-23 19:07:33	2019-04-23 19:07:33	ICL	ICLA0	TRUE

to `T` or `TRUE` and add the `file_name` containing the file name (with possible extension) and optionally, the directory, to be saved to (e.g. *"output/PITcleanr/UC_Steelhead_2019.csv"*). NOTE: to save an .xls or .xlsx file may require installation of Perl depending on your OS.

5.2 Examine PITcleanr Output

The *ProcCapHist* in `proc_list` now contains the cleaned, processed capture histories for each PIT tag with the **AutoProcStatus** column containing `PITcleanr`'s best guess of whether the observation should be used in the DABOM model and a **UserProcStatus** column allowing the end user to make their own determination of whether an observation should be used. For all tags with no issues (i.e. the detections move steadily upstream after Priest Rapids Dam), the **UserProcStatus** column has been set to `TRUE`. However, for any tags with potential detections in question, the **UserProcStatus** is (blank). In this case, the user merely needs to open the output, perhaps in Excel, and filter the **UserProcStatus** selecting all the rows with a (blank). Initally, that will include all the detections for the tags in question. By examing the dates and the nodes, and possibly considering the suggestions made in the **AutoProcStatus** column by `PITcleanr`, the user needs to fill in each blank with either `TRUE` or `FALSE` to show whether the detection at the node should be kept or ignored for the DABOM model, respectively.

One of the assumptions in the DABOM model is that fish are making a one-way upstream migration, which ends in their spawning location. So if a fish is detected moving past the TUM array, for example, and later seen moving past the ICL site, both of those observations cannot be kept in the model. Based on the observation dates (**ObsDate** and **lastObsDate**), the user will need to decide where the final spawning location was for that fish. If it was past TUM, then the rows where the **SiteID** is ICL should be marked `FALSE` in the "UserProcStatus" column, and the other one marked `TRUE`. Instead, if it appears the fish spawned in Icicle Creek, then the TUM rows should be marked `FALSE`. The default action taken by the **AutoProcStatus** column is to keep the latest observation, so it would default to keeping the ICL observations and dropping the TUM ones.

5.2.1 Summarise Information for Each Tag

At this point, some summary information can be obtained for each tag in the valid tag list, including potential spawning (i.e. final) location. This is based on the furthest model node that the tag was detected at, after filtering out unwanted observations (see Examine PITcleanr Output). The `summariseTagData()` function also takes biological information obtained at the trap (e.g. the `bio_df` object created above), and the output can be used to summarise, for example, sex ratios, age or length distributions, etc. for various nodes in the network.

```
tag_summ = proc_list$ProcCapHist %>%
  filter(AutoProcStatus) %>%
  mutate(UserProcStatus = AutoProcStatus) %>%
  summariseTagData(trap_data = bio_df %>%
                     filter(TagID %in% proc_list$ProcCapHist$TagID) %>%
                     group_by(TagID) %>%
                     slice(1) %>%
```

Table 2: Example of tag summaries.

TagID	LastObs	BranchNum	Group	AssignSpawnSite	AssignSpawnNode	TagPath
3D9.1C2DE4963C	2019-04-18 00:19:47	9	PRA	MRC	MRCB0	PRA, RRF, W
3DD.003BC64E05	2018-09-18 10:22:38	9	PRA	WEA	WEA	PRA, RRF, W
3DD.003BC72619	2018-08-30 11:55:57	NA	NA	PRDLD1	PRA	PRA
3DD.003BC73175	2019-04-04 19:43:46	9	PRA	OMK	OMKA0	PRA, RIA, RF
3DD.003BC81E88	2019-05-11 05:46:04	9	PRA	OBF	OBF	PRA, RIA, RF
3DD.003BC8232B	2019-04-07 22:27:13	9	PRA	OMK	OMKA0	PRA, RIA, RF
3DD.003BC825F0	2019-05-10 19:49:21	9	PRA	OMK	OMKA0	PRA, RIA, RF

```
                      ungroup())
```

```
tag_summ %>%
  slice(1:7) %>%
  kable(booktabs = T,
        caption = 'Example of tag summaries.') %>%
  kable_styling()
```

Congratulations! You have now prepared all of your data and detections to build and run the DABOM model.

6 DABOM

6.1 DABOM Inputs

```
library(DABOM)
library(jagsUI)
```

The DABOM model requires as input a capture or detection history of every valid tag. This is a series of `1`'s and `0`'s depicting whether that tag was detected (or not) on each node in the network. The `DABOM` package includes a function `createDABOMcapHist()` to create these detection histories, based on the output from `PITcleanr`. To make the model easier to manipulate in the future, e.g. if more sites/arrays are added, we have included an option to split the complete detection histories into separate matrices to feed into DABOM. The argument `split_matrices = TRUE` shown below in the `createDABOMcapHist()` function does just that. Alternatively, to create a single matric with the entire detection history for each tag (e.g., for diagnostic purposes), simply set that argument to `FALSE`.

In this example, we simply set the **UserProcStatus** to be the same as the **AutoProcStatus** (`mutate(UserProcStatus = AutoProcStatus`) containing `PITcleanr`'s suggestion on whether an observation should be used in DABOM, and then remove any observation where **UserProcStatus** equals `FALSE`. Alternatively, one could use any **UserProcStatus** determined by the user after examining the detection histories (see Examine PITcleanr Output).

```
# pull out the detections we are going to keep in the processed capture history
proc_ch <- proc_list$ProcCapHist %>%
  mutate_at(vars(UserProcStatus),
            list(as.logical)) %>%
    mutate(UserProcStatus = if_else(is.na(UserProcStatus),
                                    AutoProcStatus,
                                    UserProcStatus)) %>%
    filter(UserProcStatus) %>%
  filter(TagID %in% unique(bio_df$TagID))
```

```r
# put them into matrix form to be fed into JAGS
dabom_list = createDABOMcapHist(proc_ch,
                                proc_list$NodeOrder,
                                split_matrices = T)
```

This version of DABOM was created to be run on both hatchery and wild origin fish, using all tags and observations together to estimate detection probabilities, but allowing for different movement rates between hatchery and wild fish, since presumably they may be going to very different places. Therefore, one of the inputs is a vector containing the origin for each fish (1 for wild fish, 2 for hatchery), called `fishOrigin`, which must be added to the input list `dabom_list`.

```r
dabom_list$fishOrigin = createDABOMcapHist(proc_ch,
                                proc_list$NodeOrder,
                                split_matrices = F) %>%
  left_join(bio_df %>%
              select(TagID, Origin)) %>%
  select(TagID, Origin) %>%
  mutate(Origin = recode(Origin,
                         'W' = 1,
                         'H' = 2)) %>%
  pull(Origin)
```

We have compiled all of the necessary inputs for DABOM. Next, we need to generate the DABOM model that the JAGS software will use.

6.2 DABOM JAGS Model

To run the DABOM model, we need to write the text file with the Priest Rapids DABOM model that the JAGS software will use. The `DABOM` R package contains a function `writeDABOM_PRA` to write a generic version for the Priest Rapids Dam version of DABOM:

```r
# file path to the default and initial model
basic_modNm = 'examp_data/PRD_DABOM.txt'

# write the Priest Rapids Dam DABOM model for JAGS
writeDABOM_PRA(file_name = basic_modNm)
```

We then need to update that "default" DABOM model based on our tag detections. For starters, if no tags were detected at a particular node in the model, we need to set that node's detection probability to 0, since it would be difficult if not impossible to estimate. Or that node / site may not have existed or been in operation for the year we are running the model. Similarly, if a terminal site contains only a single array, or only had detections on a single array, that array has its detection probability fixed to 100%, because without detections upstream of a site there is no way to estimate the detection probability there. Setting the detection probability to 100% may lead to a conservative estimate of escapement past that particular site (i.e. escapement was *at least* this much), but since many terminal sites are further upstream in the watershed, where most detection probabilities are likely close to 100% already, this may not be a bad assumption to make. The function `fixNoFishNodes()` in `DABOM` re-writes the default JAGS model with one specific to the year we are running, and saves it as a different text file.

```r
# filepath for specific JAGS model code for species and year
mod_path = paste0('examp_data/PRA_Steelhead_', yr, '.txt')

# writes species and year specific jags code
fixNoFishNodes(basic_modNm,
               mod_path,
              proc_ch,
              proc_list$NodeOrder)
#> Fixed AEN, MRCA0, CHWB0, CHWA0, CLK, EHLB0, EHLA0, ENMB0, ENMA0, ENFB0, ICMB0, ICMA0, ICUB0, ICUA0,
#>
#> Fixed BelowJD1 at 100% detection probability, because it is a single array with no upstream detectio
#>
#> Fixed ENF at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed FST at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed JD1 at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed METH at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed NMC at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed OBF at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed TMF at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed TNK at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed TON at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed TWISPW at 100% detection probability, because it is a single array with no upstream detections
#>
#> Fixed WFC at 100% detection probability, because it is a single array with no upstream detections.
#>
#> Fixed upstream movement past site ICU to 0 because no detections there or upstream.
#>
#> Fixed upstream movement past site PEU to 0 because no detections there or upstream.
```

Based on the code in that modified model text file for our DABOM run, we can then extract the names of the parameters we wish to save.

```
# Tell JAGS which parameters in the model that it should save.
jags_params = setSavedParams(model_file = mod_path)
```

Now we're ready to set some intial values for the JAGS model. We don't set the values for the detection or movement probabilities; however, we do need to set the initial values for where fish are that have been detection. Otherwise, JAGS will throw an error because a tag has a randomly selected initial value (e.g. Chiwawa River), but it was detected elsewhere (e.g. Little White River).

```
# Creates a function to spit out initial values for MCMC chains
init_fnc = setInitialValues_PRA(dabom_list)
```

Finally, we can compile all the input data necessary to run the JAGS model using the `createJAGSinputs_PRA()` function. This includes the capture history matrices, the origin of each tag, and the priors for the Dirichlet vector that goes with each node with multiple branches. That prior is merely `1`'s and `0`'s, turning branches off if there were no observed tags there, and preventing the model from trying to estimate movement parameters to places with no detections.

```r
# Create all the input data for the JAGS model
jags_data = createJAGSinputs_PRA(dabom_list)
```

6.3 Run DABOM

To actually run DABOM, we use the `jags.basic()` function in the `jagsUI` package. Part of the inputs to that function include the Monte Carlo Markov chain (MCMC) parameters. We recommend:

- 4 chains (`n.chains = 4`)
- 5,000 iterations (`n.iter = 5000`)
- 2,500 of which are burn-in (`n.burnin = 2500`)
- Keep every 10 iterations (`n.thin = 10`)

This provides a total of 1,000 draws from the posterior. In our experience, this leads to convergence of all the parameters, without taking too long to run. Setting the seed in `R` means you can reproduce the exact MCMC draws.

```r
# Run the model
set.seed(12)
dabom_mod <- jags.basic(data = jags_data,
                        inits = init_fnc,
                        parameters.to.save = jags_params,
                        model.file = mod_path,
                        n.chains = 4,
                        n.iter = 5000,
                        n.burnin = 2500,
                        n.thin = 10,
                        DIC = T)
```

What is returned (from the `jags.basic` function) is an `mcmc.list` object. The user can generate many diagnostic plots and statistics, using packages such as `coda`, `mcmcr`, `shinystan`, or even `postpack` available here on www.github.com/.

6.4 Summarise DABOM results

Now that we have completed our DABOM run, we can summarise a number of results from the `dabom_mod` `mcmc.list` object we created above including detection and movement (i.e. transition) probabilties. In addition, transition probabilities can be multiplied by some estimate of escapement in the system (e.g. Priest Rapids Dam) to get abundance to any node or group of nodes (e.g. a tributary).

6.4.1 Detection Probabilities

Detection probabilities are easily summarised using the function `summariseDetectProbs()`:

```r
# summarise detection probabilities
detect_summ = summariseDetectProbs(dabom_mod = dabom_mod,
                                   capHist_proc = proc_list$proc_ch,
                                   cred_int_prob = 0.95) %>%
  # remove nodes that were not saved by JAGS
  filter(!is.na(mean))
```

```r
detect_summ %>%
  filter(sd > 0) %>%
  arrange(desc(sd)) %>%
  kable(digits = 3,
        booktabs = T,
        caption = 'Summary of detection parameters.') %>%
  kable_styling()
```

6.4.2 Transition Probabilities

The movement or transition probabilities have to be extracted, and then multiplied appropriately, so that the movement past a site far upstream in the network also accounts for moving past all of the downstream sites before that one. There is a function specific to the Priest Rapids Dam version of DABOM `compileTransProbs_PRA` to do this for you. The **param** column in the `trans_summ` summary table below refers to the probability of a tag moving *past* that site, or if the parameter ends in "_bb", the probability of falling into the black box above that site.

```
# compile all movement probabilities, and multiply them appropriately
trans_df = compileTransProbs_PRA(dabom_mod)
```

```
# summarize transition probabilities
trans_summ = trans_df %>%
  group_by(Origin, param) %>%
  summarise(mean = mean(value),
            median = median(value),
            mode = estMode(value),
            sd = sd(value),
            lowerCI = coda::HPDinterval(coda::as.mcmc(value))[,1],
            upperCI = coda::HPDinterval(coda::as.mcmc(value))[,2]) %>%
  mutate_at(vars(mean, median, mode, sd, matches('CI$')),
            list(~ if_else(. < 0, 0, .))) %>%
  ungroup()

trans_summ %>%
  filter(sd > 0) %>%
  kable(digits = 3,
        booktabs = T,
        caption = 'Summary of movement parameters.') %>%
  kable_styling()
```

7 Abundance Estimates

```
# install STADEM
devtools::install_github("KevinSee/STADEM")
```

Finally, we want to generate estimates of abundance for nodes or groups of nodes (e.g. tributaries or populations). To accomplish this, the next step is to multiply the transition probabilities from `DABOM` by the total escapement at Priest Rapids Dam. There is an additional R package, `STADEM` (*St*ate-space *A*dult *D*am *E*scapement *M*odel) that contains a function `getWindowCounts()` to query the window counts at a variety of dams within the Columbia River Basin, including Priest Rapids Dam, on a daily time-step. Those daily counts can be summed up for the season:

```
# load STADEM
library(STADEM)

# window count
tot_win_cnt = getWindowCounts(dam = 'PRD',
                              spp = 'Steelhead',
                              start_date = paste0(yr-1, '0601'),
```

Table 3: Summary of detection parameters.

Node	n_tags	mean	median	mode	sd	lowerCI	upperCI
CHMA0	2	0.666	0.698	0.879	0.231	0.219	0.997
CHMB0	2	0.674	0.722	0.835	0.230	0.251	0.999
HSTB0	1	0.494	0.490	0.483	0.224	0.118	0.929
PRV	1	0.505	0.494	0.454	0.221	0.099	0.900
LLCA0	2	0.708	0.747	0.902	0.218	0.283	1.000
LLCB0	2	0.695	0.720	0.910	0.214	0.287	0.999
ANTB0	2	0.708	0.745	0.906	0.213	0.291	1.000
ANTA0	2	0.716	0.758	0.918	0.209	0.310	1.000
HSTA0	2	0.747	0.793	0.918	0.195	0.363	1.000
MCLA0	3	0.756	0.799	0.922	0.192	0.382	1.000
MCLB0	3	0.753	0.798	0.922	0.190	0.387	1.000
LBCA0	3	0.761	0.803	0.931	0.184	0.390	1.000
LBCB0	3	0.771	0.810	0.927	0.182	0.417	1.000
ENSB0	3	0.490	0.495	0.511	0.166	0.165	0.782
PRHB0	3	0.806	0.851	0.942	0.165	0.471	0.999
SA1A0	2	0.369	0.357	0.318	0.159	0.098	0.679
PRHA0	3	0.795	0.827	0.930	0.158	0.480	1.000
ENSA0	2	0.370	0.360	0.318	0.157	0.058	0.650
CHUA0	6	0.676	0.685	0.735	0.151	0.374	0.935
ENAB0	5	0.590	0.597	0.604	0.148	0.296	0.846
ENAA0	6	0.684	0.692	0.675	0.146	0.375	0.924
PESB0	6	0.500	0.505	0.514	0.136	0.248	0.758
RCTA0	5	0.839	0.871	0.952	0.136	0.571	1.000
RCTB0	5	0.842	0.874	0.955	0.134	0.566	1.000
NAUA0	5	0.849	0.889	0.950	0.132	0.584	1.000
NAUB0	5	0.850	0.884	0.954	0.131	0.568	1.000
OKCA0	8	0.625	0.633	0.676	0.128	0.376	0.855
CHUB0	7	0.774	0.791	0.822	0.127	0.520	0.992
ICLB0	6	0.782	0.799	0.847	0.127	0.535	0.982
NALA0	7	0.796	0.814	0.867	0.122	0.554	0.985
SA1B0	6	0.861	0.893	0.959	0.119	0.625	1.000
CHLA0	10	0.568	0.566	0.566	0.116	0.354	0.798
PESA0	11	0.863	0.895	0.948	0.115	0.637	0.999
CHLB0	12	0.672	0.676	0.677	0.114	0.453	0.872
SA0B0	6	0.877	0.913	0.965	0.113	0.651	1.000
CRUB0	7	0.872	0.904	0.963	0.113	0.638	1.000
MRWB0	13	0.430	0.424	0.409	0.111	0.207	0.627
SA0A0	6	0.878	0.910	0.965	0.111	0.651	1.000
CRUA0	7	0.872	0.896	0.953	0.103	0.675	1.000
BPCA0	10	0.831	0.850	0.897	0.102	0.634	0.988
MRWA0	10	0.341	0.337	0.346	0.100	0.155	0.537
UWE	10	0.847	0.865	0.925	0.098	0.659	0.990
ICLA0	7	0.886	0.915	0.963	0.097	0.686	1.000
NALB0	8	0.897	0.922	0.972	0.094	0.702	1.000
OKCB0	12	0.905	0.932	0.971	0.085	0.730	1.000
ENLA0	17	0.558	0.558	0.558	0.085	0.395	0.719
CRWA0	14	0.826	0.835	0.842	0.084	0.656	0.965
ENLB0	20	0.655	0.657	0.656	0.082	0.478	0.798
SCPA0	10	0.912	0.936	0.968	0.080	0.743	1.000
BPCB0	11	0.912	0.936	0.973	0.080	0.751	1.000

Table 4: Summary of movement parameters.

Origin	param	mean	median	mode	sd	lowerCI	upperCI
Hatchery	BelowJD1	0.004	0.003	0.003	0.002	0.000	0.009
Hatchery	dwnStrm	0.156	0.156	0.155	0.014	0.131	0.184
Hatchery	ENL_bb	0.002	0.002	0.001	0.002	0.000	0.005
Hatchery	ICL_bb	0.008	0.007	0.007	0.003	0.002	0.013
Hatchery	LMR_bb	0.026	0.026	0.026	0.006	0.016	0.038
Hatchery	LWE_bb	0.019	0.019	0.020	0.005	0.009	0.029
Hatchery	MRC_bb	0.034	0.034	0.036	0.017	0.002	0.062
Hatchery	OKL_bb	0.016	0.015	0.015	0.005	0.007	0.024
Hatchery	past_BPC	0.010	0.010	0.010	0.004	0.004	0.018
Hatchery	past_BVC	0.011	0.011	0.011	0.004	0.005	0.019
Hatchery	past_CHL	0.015	0.015	0.014	0.005	0.007	0.025
Hatchery	past_CHM	0.003	0.002	0.002	0.002	0.000	0.007
Hatchery	past_CHU	0.005	0.004	0.003	0.002	0.001	0.010
Hatchery	past_CRU	0.001	0.001	0.000	0.001	0.000	0.003
Hatchery	past_CRW	0.003	0.002	0.002	0.002	0.000	0.006
Hatchery	past_ENL	0.011	0.011	0.010	0.004	0.004	0.019
Hatchery	past_FST	0.015	0.015	0.014	0.005	0.007	0.024
Hatchery	past_GLC	0.011	0.011	0.010	0.004	0.004	0.018
Hatchery	past_HST	0.002	0.001	0.001	0.001	0.000	0.005
Hatchery	past_ICH	0.094	0.094	0.096	0.011	0.072	0.113
Hatchery	past_ICL	0.008	0.007	0.007	0.003	0.002	0.013
Hatchery	past_JD1	0.003	0.002	0.002	0.002	0.000	0.007
Hatchery	past_LLC	0.004	0.004	0.003	0.003	0.000	0.009
Hatchery	past_LMR	0.168	0.168	0.167	0.014	0.139	0.194
Hatchery	past_LWE	0.064	0.063	0.062	0.009	0.046	0.081
Hatchery	past_MAD	0.006	0.005	0.005	0.003	0.001	0.010
Hatchery	past_MCL	0.003	0.002	0.001	0.002	0.000	0.007
Hatchery	past_METH	0.010	0.009	0.008	0.004	0.004	0.018
Hatchery	past_MRC	0.131	0.131	0.127	0.013	0.104	0.153
Hatchery	past_MRW	0.025	0.025	0.023	0.008	0.011	0.041
Hatchery	past_MSH	0.028	0.024	0.014	0.016	0.005	0.058
Hatchery	past_NAL	0.005	0.005	0.004	0.003	0.001	0.010
Hatchery	past_NAU	0.002	0.001	0.001	0.001	0.000	0.004
Hatchery	past_OBF	0.012	0.012	0.012	0.004	0.005	0.019
Hatchery	past_OKC	0.008	0.008	0.007	0.003	0.003	0.014
Hatchery	past_OKL	0.093	0.092	0.090	0.011	0.073	0.115
Hatchery	past_OMK	0.035	0.035	0.034	0.007	0.023	0.050
Hatchery	past_PES	0.003	0.002	0.002	0.002	0.000	0.006
Hatchery	past_PRH	0.005	0.005	0.004	0.003	0.001	0.011
Hatchery	past_PRO	0.005	0.005	0.004	0.003	0.001	0.011
Hatchery	past_PRV	0.003	0.002	0.002	0.002	0.000	0.007
Hatchery	past_RCT	0.004	0.003	0.002	0.002	0.000	0.008
Hatchery	past_RIA	0.784	0.784	0.783	0.016	0.756	0.817
Hatchery	past_RRF	0.707	0.707	0.704	0.017	0.675	0.741
Hatchery	past_RSH	0.042	0.042	0.042	0.007	0.028	0.057
Hatchery	past_SA0	0.002	0.001	0.001	0.002	0.000	0.005
Hatchery	past_SA1	0.003	0.002	0.002	0.002	0.000	0.006
Hatchery	past_SCP	0.012	0.011	0.010	0.004	0.005	0.018
Hatchery	past_TNK	0.005	0.005	0.004	0.002	0.001	0.010
Hatchery	past_TON	0.006	0.005	0.004	0.003	0.001	0.011

```
                                    end_date = paste0(yr, '0531')) %>%
  summarise_at(vars(win_cnt),
               list(sum)) %>%
  pull(win_cnt)
```

We can also query data related to the re-ascension rate, what percentage of the fish counted at the window have fallen back and re-ascended the dam, and are therefore counted twice at the window. This is also done through a query in the STADEM package.

```
# reascension data
reasc_data = queryPITtagData(damPIT = 'PRA',
                             spp = 'Steelhead',
                             start_date = paste0(yr-1, '0601'),
                             end_date = paste0(yr, '0531'))
```

Now the window counts can be adjusted for re-ascension rates, as well as divided by origin, with appropriate standard errors.

```
library(msm) # for the deltamethod

# adjust for re-ascension and origin
org_escape = reasc_data %>%
    mutate(SpawnYear = yr,
           TagIDAscentCount = ifelse(is.na(TagIDAscentCount),
                                     0, TagIDAscentCount),
           ReAscent = ifelse(TagIDAscentCount > 1, T, F)) %>%
    group_by(Species, SpawnYear, Date) %>%
    summarise(tot_tags = n_distinct(TagID),
              reascent_tags = n_distinct(TagID[ReAscent])) %>%
    ungroup() %>%
    group_by(Species, SpawnYear) %>%
    summarise_at(vars(matches('tags')),
                 list(sum),
                 na.rm = T) %>%
    ungroup() %>%
    mutate(reascRate = reascent_tags / tot_tags,
           reascRateSE = sqrt(reascRate * (1 - reascRate) / tot_tags),
           totWinCnt = tot_win_cnt,
           adjWinCnt = tot_win_cnt * (1 - reascRate),
           adjWinCntSE = tot_win_cnt * reascRateSE) %>%
    bind_cols(bio_df %>%
                group_by(Origin) %>%
                summarise(nTags = n_distinct(TagID)) %>%
                pivot_wider(names_from = "Origin",
                            values_from = "nTags",
                            values_fill = list(nTags = as.integer(0))) %>%
                ungroup() %>%
                mutate(propW = W / (W + H),
                       propH = 1 - propW,
                       propOrgSE = sqrt((propW * (1 - propW)) / (W + H)))) %>%
    mutate(Hescp = propH * adjWinCnt,
           HescpSE = deltamethod(~ x1 * x2,
                                 mean = c(propH, adjWinCnt),
                                 cov = diag(c(propOrgSE, adjWinCntSE)^2))) %>%
    mutate(Wescp = propW * adjWinCnt,
           WescpSE = deltamethod(~ x1 * x2,
                                 mean = c(propW, adjWinCnt),
                                 cov = diag(c(propOrgSE, adjWinCntSE)^2))) %>%
    select(Species, SpawnYear, matches('escp')) %>%
    pivot_longer(-(Species:SpawnYear),
                 names_to = "var",
                 values_to = "value") %>%
    mutate(Origin = if_else(grepl('^H', var),
                            'Hatchery',
                            'Natural'),
           param = if_else(grepl('SE$', var),
                           'tot_escp_se',
                           'tot_escp')) %>%
    select(-var) %>%
    pivot_wider(names_from = "param",
                values_from = "value")
```

We then take each draw from the transition probability posteriors, and multiply it by the appropriate (by origin) adjusted total escapement. This provides posteriors of escapement past every detection point, which can then be summarised.

```
# translate movement estimates to escapement
# how many samples were taken from the posteriors?
n_samps = trans_df %>%
  group_by(Origin, param) %>%
  summarise(nIters = n()) %>%
  ungroup() %>%
  pull(nIters) %>%
  unique()

set.seed(5)
escape_summ = org_escape %>%
  split(list(.$Origin)) %>%
  map_df(.id = 'Origin',
         .f = function(x) {
           tibble(totEsc = rnorm(n_samps, x$tot_escp, x$tot_escp_se)) %>%
             mutate(iter = 1:n_samps)
         }) %>%
  left_join(trans_df %>%
              group_by(Origin, param) %>%
              mutate(iter = 1:n()) %>%
              select(-chain) %>%
              rename(prob = value) %>%
              slice(sample.int(max(iter), n_samps)) %>%
              ungroup()) %>%
  mutate(value = totEsc * prob) %>%
  group_by(Origin, param) %>%
  summarise(mean = mean(value),
            median = median(value),
            mode = estMode(value),
            sd = sd(value),
            # skew = moments::skewness(value),
            # kurtosis = moments::kurtosis(value),
            lowerCI = coda::HPDinterval(coda::as.mcmc(value))[,1],
            upperCI = coda::HPDinterval(coda::as.mcmc(value))[,2]) %>%
  mutate_at(vars(mean, median, mode, sd, matches('CI$')),
            list(~ if_else(. < 0, 0, .))) %>%
  ungroup()
```

If we'd like to summarise escapement at the population scale, that's just a matter of extracting some estimates and adding a few of them together (i.e. include CLK with Wenatchee estimates, and FST with Okanogan estimates).

```
# generate population level estimates
pop_summ = org_escape %>%
  split(list(.$Origin)) %>%
  map_df(.id = 'Origin',
         .f = function(x) {
           tibble(totEsc = rnorm(n_samps, x$tot_escp, x$tot_escp_se)) %>%
             mutate(iter = 1:n_samps)
         }) %>%
  left_join(trans_df %>%
              group_by(Origin, param) %>%
              mutate(iter = 1:n()) %>%
              select(-chain) %>%
              rename(prob = value) %>%
              slice(sample.int(max(iter), n_samps)) %>%
              ungroup()) %>%
  mutate(value = totEsc * prob) %>%
  select(Origin, iter, param, value) %>%
  pivot_wider(names_from = "param",
              values_from = "value") %>%
  mutate(Wenatchee = past_LWE + past_CLK,
         Entiat = past_ENL,
         Methow = past_LMR,
         Okanogan = past_OKL + past_FST,
         `Below Priest` = dwnStrm,
         `Wells Pool` = WEA_bb) %>%
  select(Origin, iter, Wenatchee:`Wells Pool`) %>%
  pivot_longer(cols = Wenatchee:`Wells Pool`,
               names_to = 'Population',
               values_to = "escp") %>%
  group_by(Population, Origin) %>%
  summarise(mean = mean(escp),
            median = median(escp),
            mode = estMode(escp),
            sd = sd(escp),
            # skew = moments::skewness(escp),
            # kurtosis = moments::kurtosis(escp),
            lowerCI = coda::HPDinterval(coda::as.mcmc(escp))[,1],
            upperCI = coda::HPDinterval(coda::as.mcmc(escp))[,2]) %>%
  mutate_at(vars(mean, median, mode, sd, matches('CI$')),
            list(~ if_else(. < 0, 0, .))) %>%
  ungroup()
```

```
pop_summ %>%
  kable(digits = 0,
        caption = 'Population-scale escapement estimates.') %>%
```

Table 5: Population-scale escapement estimates.

Population	Origin	mean	median	mode	sd	lowerCI	upperCI
Below Priest	Hatchery	514	513	515	47	430	611
Below Priest	Natural	205	204	205	31	144	261
Entiat	Hatchery	37	36	35	13	14	61
Entiat	Natural	109	108	108	22	70	152
Methow	Hatchery	556	555	555	48	464	650
Methow	Natural	406	403	396	42	317	481
Okanogan	Hatchery	357	355	346	38	283	430
Okanogan	Natural	196	193	186	30	143	258
Wells Pool	Hatchery	1248	1248	1248	66	1116	1364
Wells Pool	Natural	177	175	171	27	125	233
Wenatchee	Hatchery	210	208	206	30	154	270
Wenatchee	Natural	191	190	191	28	137	248

```
  kable_styling()
```

7.1 Save Results

Finally, we can save these results to an Excel spreadsheet, including the population estimates, estimates past each detection site, and the detection probabilities for each node.

```
# write results to an Excel file
save_list = list('Population Escapement' = pop_summ %>%
                   # select(-skew, -kurtosis) %>%
                   mutate_at(vars(-pop),
                             list(round),
                             digits = 1),
                 'All Escapement' = escape_summ %>%
                   # select(-skew, -kurtosis) %>%
                   mutate_at(vars(-location),
                             list(round),
                             digits = 1),
                 'Detection' = detect_summ %>%
                   mutate_at(vars(-Node),
                             list(round),
                             digits = 3))

WriteXLS(x = save_list,
         ExcelFileName = paste0('examp_data/PRA_est_', spp, '_', yr, '_', format(Sys.Date(), '%Y%m%d'),
         AdjWidth = T,
         AutoFilter = F,
         BoldHeaderRow = T,
         FreezeRow = 1)
```