PITcleanr Vignette

Kevin E. See

2017-10-13

Contents

Introduction

The PITcleanr package was developed to help query the necessary data to fit a DABOM model (**D**am **A**dult **B**ranch **O**ccupancy **M**odel) in order to estimate adult escapement to various tributaries above a tagging location. A key assumption of a DABOM model is that fish travel along a single upstream route. Therefore, the model will fail if presented with detections by a single fish in multiple tributaries above a branching node. PITcleanr is designed to help clean the PIT tag detection data to identify non-linear upstream pathways and help the user determine which observations to keep.

System requirements

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

- `dplyr`, `lubridate`, `httr`, `purrr` and `stringr`: can all be installed by installing the `tidyverse` package

In addition, many of the various queries in PITcleanr require connection to the internet.

```
library(tidyverse)
library(httr)
library(stringr)
```

```
library(lubridate)
library(PITcleanr)
```

Data sources

PITcleanr starts with data from an adult trap database, to determine which tags are considered part of the valid tag sample, and what the trap date is for each tag. For the Lower Granite dam, the data can be retreived from the "QCI Downloads" folder on the Idaho Fish and Game's IFWIS website. The trapping data is stored in an Microsoft Access database that is updated weekly and named "LGTrappingExportJodyW.accdb". Permission must be obtained from Idaho Fish and Game before viewing, accessing and retreiving data from the site. This database contains two tables, which can be exported as .csv files. Once this is done, PITcleanr has a function, `filterLGRtrapDB` to determine which tags are valid, and when they were caught in the adult trap.

Other than the adult trap database, PITcleanr relies on queries from the Columbia Basin Research Data Access in Real Time (DART) and the Columbia Basin PIT Tag Information System (PTAGIS).

Workflow

For the purposes of this vignette, we will focus on data to be fed into the Lower Granite version of DABOM, using data from spring/summer Chinook that crossed Lower Granite in the spring and summer of 2015.

```
spp = c('Chinook', 'Steelhead')[1]
yr = 2015
```

Configuration Table

One of the first steps is to compile a configuration file containing information about all of the detection sites that may be present. PITcleanr will create such a configuration file for all sites in the Columbia River basin, using the `buildConfig()` function.

```
my_config = buildConfig()
```

```
## [1] "Querying INT sites' metadata"
## [1] "Querying INT sites' configuration information"
## [1] "Querying MRR sites' metadata"
```

A user can then make custom changes depending on the model set-up. For example, in the Lower Granite version of DABOM, all antennas in VC1 and VC2 are combined into a single node.

```
my_config = my_config %>%
  mutate(Node = ifelse(SiteID == 'VC2',
                       'VC2',
                       Node),
         Node = ifelse(SiteID == 'VC1',
                       'VC1',
                       Node)) %>%
  distinct()

# a sample of this configuration file
head(my_config)
```

```
## # A tibble: 6 x 19
##   SiteID ConfigID AntennaID  Node ValidNode  StartDate EndDate Comment
##    <chr>    <dbl>     <chr> <chr>     <lgl>     <dttm>  <dttm>   <lgl>
## 1    158      100        D1 158A0        NA 2011-11-29      NA      NA
## 2    158      100        D2 158A0        NA 2011-11-29      NA      NA
## 3    158      100        D3 158B0        NA 2011-11-29      NA      NA
## 4    158      100        D4 158A0        NA 2011-11-29      NA      NA
## 5    158      100        D5 158A0        NA 2011-11-29      NA      NA
## 6    158      100        D6 158B0        NA 2011-11-29      NA      NA
## # ... with 11 more variables: SiteType <chr>, SiteName <chr>,
## #   ModelMainBranch <lgl>, AntennaGroup <chr>, ArrayOrder <lgl>,
## #   SiteDescription <chr>, SiteTypeName <chr>, RKM <chr>, RKMTotal <dbl>,
## #   Latitude <dbl>, Longitude <dbl>
```

Node Network

The next step is to define how the various sites that are to be used with DABOM are related to each other. PITcleanr contains a function to describe that network for the Lower Granite version of DABOM, `writeLGRNodeNetwork()`. This serves two purposes: it defines which sites are to be used in the DABOM model, and it describes how they are related to each other along the stream network.

```
site_df = writeLGRNodeNetwork()

# a sample of what this looks like
head(site_df)
```

```
## # A tibble: 6 x 11
##   SiteID                        path      Step1    Step2 Step3 Step4 Step5
##   <fctr>                       <chr>      <chr>    <chr> <chr> <chr> <chr>
## 1    LAP       Clearwater.Lapwai.LAP Clearwater   Lapwai
## 2    MIS       Clearwater.Lapwai.LAP Clearwater   Lapwai   LAP
## 3    SWT   Clearwater.Lapwai.LAP.SWT Clearwater   Lapwai   LAP
## 4    WEB   Clearwater.Lapwai.LAP.SWT Clearwater   Lapwai   LAP   SWT
## 5    JUL     Clearwater.Potlatch.JUL Clearwater Potlatch
## 6    KHS Clearwater.Potlatch.JUL.KHS Clearwater Potlatch   JUL
## # ... with 4 more variables: Step6 <chr>, Step7 <chr>, Step8 <chr>,
## #   Step9 <chr>
```

Parent-Child Table

Next, we build a parent-child dataframe describing which child nodes are directly upstream of each parent node in our system. The `createParentChildDf` function contains several arguements:

- `sites_df`: This is the dataframe we built using the `writeLGRNodeNetwork()` function.
- `config`: This is the configuration file we built using the `buildConfig()` function.
- `startSite`: This defines which site all paths will start from. This is typically the dam where the adult trap is located.
- `startDate`: The first date (in YYYYMMDD format) when fish may start being caught in the adult trap. Configurations that ended prior to this date are excluded.

```
parent_child = createParentChildDf(site_df,
                                   my_config,
                                   startSite = 'GRA',
                                   startDate = ifelse(spp == 'Chinook',
```

```
                                                    paste0(yr, '0301'),
                                                    paste0(yr-1, '0701')))
```

Valid Paths

Once the parent-child dataframe has been created, PITcleanr uses the `getValidPaths` function to construct a dataframe of all valid paths a fish may follow.

```
valid_paths = getValidPaths(parent_child)
```

Valid Tags

From the trap database, PITcleanr searches for tags that are part of the valid tag list for a particular species and year, and returns a dataframe with their tag code and the date of trapping. PITcleanr contains an example of the trap database (`trap_chnk2015`), which only contains spring/summer Chinook in 2015. Set `path` equal to the file path, including the file name and extension, of the actual table (named tblLGDMasterCombineExportJodyW.csv in my version of the trap database). Currently it must be in CSV format.

```
data(trap_chnk2015)
head(trap_chnk2015)
```

```
## # A tibble: 6 x 52
##   MasterID CollectionDate SpawnYear LGDTrapID BioSamplesID
##      <int>          <dttm>     <chr>     <int>        <chr>
## 1   260635     2015-04-23    SY2015        NA         <NA>
## 2   260640     2015-04-23    SY2015        NA         <NA>
## 3   260636     2015-04-23    SY2015        NA         <NA>
## 4   260639     2015-04-23    SY2015        NA         <NA>
## 5   260671     2015-04-23    SY2015        NA         <NA>
## 6   260672     2015-04-23    SY2015        NA         <NA>
## # ... with 47 more variables: CollectionLocation <chr>, LGDFLmm <int>,
## #   LGDWeight <chr>, SRR <chr>, LGDSpecies <int>, PtagisSpecies <int>,
## #   GenSpecies <chr>, LGDRun <int>, PtagisRun <int>, GenRun <chr>,
## #   LGDRear <chr>, PtagisRear <chr>, GenRear <chr>, LGDLifeStage <chr>,
## #   LGDSex <chr>, GenSex <chr>, GenStock <chr>, GenStockProb <dbl>,
## #   GenParentHatchery <chr>, GenBY <int>, GenPBT_ByHat <chr>,
## #   GenPBT_RGroup <chr>, GenComments <chr>, LGDFishComments <chr>,
## #   BioScaleFinalAge <chr>, PtagisEventSites <chr>,
## #   PtagisLastEventSite <chr>, PtagisLastEventDate <dttm>,
## #   PtagisEventLastSpawnSite <chr>, RepeatSpawner <chr>,
## #   BiosamplesValid <lgl>, LGDValid <int>, LGDInjuryiesAll <chr>,
## #   LGDMarksAll <chr>, LGDMarkAD <chr>, LGDTagsAll <chr>, LGDNumPIT <chr>,
## #   LGDNumRT <chr>, LGDNumJaw <chr>, LGDOpComments <chr>,
## #   PTAgisSxCGRAObs <chr>, OpTrapSampleRate <dbl>, OpWaterTemp <chr>,
## #   CreatedDate <dttm>, CreatedUser <chr>, Editdate <dttm>, EditUser <chr>
```

```
valid_df = filterLGRtrapDB(path = NULL,
                           species = spp,
                           spawnYear = yr)
```

This `valid_df` dataframe contains lots of biological information for all the valid tags that may be useful later, which is why we maintain it in its entirety. However, for DABOM, we only need the PIT tag code and

the trap date.

```
# pull valid tags from trap database, get trap date
valid_tag_df = summariseValidTagsLGR(valid_df)
```

Complete Tag History

The next step is to query PTAGIS for the complete tag history of all valid tags in our sample. This can be done in two ways.

Through PTAGIS

The valid tag numbers now need to be exported into a '.txt' file for loading into PTAGIS to perform a 'Complete Tag History' query. We only need to export the contents of the TagID field which contains the unique PIT-tag codes and save them in a text file with no headers or rownames.

```
tag_codes <- valid_tag_df$TagID
# write.table(tag_codes,
#             file = '../tag_codes.txt',
#             quote = FALSE,
#             sep = '\n',
#             row.names = FALSE,
#             col.names = FALSE)
```

After logging into PTAGIS, select the "Data" tab, "Advanced Reporting" and then the "Launch" button. Now, select the "New Query Builder2 Report" icon and the "Complete Tag History" option. The **DABOM** functions require the following attributes to be selected; *Tag*, *Event Date Time*, *Event Site Code*, *Antenna*, and *Antenna Group Configuration*. Other attributes can be selected and they can appear in any order. After selecting attributes, upload *tag_codes.txt* in the 'Tag Code - List or Text File' section and then run the report and save the output. Next read the outputted observation file into R or use the example observation file, "chnk15_obs".

```
data(chnk2015_obs)
observation = chnk2015_obs
```

Through DART

DART has built an API query allowing users to query the complete tag history of any particular tag. Be warned however, on one test machine, this query could return the records of about 10 tags per minute, so for several thousand tags, it will take awhile.

```
observation = valid_tag_df %>%
  select(TagID) %>%
  as.matrix() %>%
  as.character() %>%
  as.list() %>%
  map_df(.f = queryCapHist,
         configuration = my_config)
```

Filter Tag History

The next step is to assign each observation to an appropriate node from the DABOM model, as defined in the configuration file. The `assignNodes()` function will do just this. It requires the complete tag histories,

the dataframe of valid tags and their trap date, the site configuration dataframe, the parent-child dataframe to filter out observations from sites not contained in the DABOM version. If `truncate` is set to `TRUE` (the default), the `assignNodes()` function will filter out observations that occurred prior to the trap date, and remove consecutive observations at the same node to simplify the file.

```
valid_obs_dat = assignNodes(valid_tag_df,
                            observation,
                            configuration = my_config,
                            parent_child,
                            truncate = T)
```

Determine Valid Observations

Fish Paths

The final step is to determine if each observation of a particular tag is following a "one-way" path upstream. The logic behind this function was first proposed and implemented by Greg Kliewer and worked within a SQLite database. We have ported that logic into more standard R functions. The `writeFishPaths()` function does this by determining the final node a fish was observed at, and querying the valid path dataframe for all the downstream sites that may be encountered on the way to that node. It also proposes an extended path of the first set of nodes upstream of the final node, and acknowledges that observations along that extended path may also be valid (e.g. a fish swims a particular route upstream, spawns and then swims partway downstream, past a node or two. The most upstream node should be considered a valid observation, and the likely spawning location.).

It returns a dataframe with a row for each observed node, showing the tag code, the node, the minimum observed date-time, and two columns called "AutoProcStatus" and "UserProcStatus". "AutoProcStatus" is PITcleanr's best attempt to determine which observations should be deleted before bringing the data into DABOM. UserProcStatus is there for the user to change things. Any tag that has at least one flagged observation (AutoProcStatus == `FALSE`) will have all of the observations labeled as blanks in the UserProcStatus column. This allows the user to filter initially on the rows with UserProcStatus == '', and see all the observations for each flagged tag. The user can then determine which observations should be included as "valid" by marking the UserProcStatus as `TRUE`, or which ones should be deleted by marking the UserProcStatus as `FALSE`, guided by the suggestions in the AutoProcStatus column, and by the user's knowledge of fish behavior and the system in question.

```
fish_paths = writeFishPaths(valid_obs_dat,
                            valid_paths)
```

```
## [1] "2017-10-13 13:34:50  1 of 3026 tags. Current tag: 384.3B2395D0A4"
## [1] "2017-10-13 13:35:13  500 of 3026 tags. Current tag: 384.3B23AD2E1B"
## [1] "2017-10-13 13:35:35  1000 of 3026 tags. Current tag: 3DD.00773A0BCE"
## [1] "2017-10-13 13:36:00  1500 of 3026 tags. Current tag: 3DD.00773AA565"
## [1] "2017-10-13 13:36:24  2000 of 3026 tags. Current tag: 3DD.00773AD944"
## [1] "2017-10-13 13:36:48  2500 of 3026 tags. Current tag: 3DD.00773B083A"
## [1] "2017-10-13 13:37:11  3000 of 3026 tags. Current tag: 3DD.00773BB274"
```

```
# how many tags are flagged for further review?
fish_paths %>%
  filter(UserProcStatus == '') %>%
  summarise(n_flagged_tags = n_distinct(TagID))
```

```
## # A tibble: 1 x 1
##   n_flagged_tags
##            <int>
```

```
## 1             44
```

```
# what percentage is that?
fish_paths %>%
  select(TagID, UserProcStatus) %>%
  distinct() %>%
  xtabs(~ UserProcStatus, .) %>%
  prop.table() %>%
  round(3)
```

```
## UserProcStatus
##        TRUE
## 0.015 0.985
```

In this example, PITcleaner has flagged 44 tags as having less than straightforward detection paths, out of 3026 total tags.

Spawner Paths

Ryan Kinzer put together another function to determine the direction of travel for each individual fish by checking the current and previous node locations against the valid path strings. We've adapted it as part of PITcleanr, and called it `writeSpwnPaths`. For each observation, if the node is within a valid path (regardless of what direction the tag appears to be moving), it is labeled as a valid path. However, if the observation node is downstream of a previously identified upstream node, the direction of travel is labeled as "Down" and the *ModelObs* column is marked as `FALSE`.

```
spwn_paths = writeSpwnPaths(valid_obs_dat,
                            valid_paths)

# how many tags are flagged for further review?
spwn_paths %>%
  filter(!ValidPath | !ModelObs) %>%
  summarise(n_flagged_tags = n_distinct(TagID))
```

```
## # A tibble: 1 x 1
##   n_flagged_tags
##            <int>
## 1            272
```

```
# # determine spawning location (most upstream detection)
# spwn_paths %>%
#   filter(ModelObs) %>%
#   group_by(TagID) %>%
#   filter(NodeOrder == max(NodeOrder)) %>%
#   ungroup() %>%
#   select(TagID, SiteID, Node, ObsDate) %>%
#   distinct()
```

The User's Remaining Work

The results of these two functions, `writeFishPaths` and `writeSpwnPaths` can be combined to help the user determine which observations to ultimately discard before feeding the data to DABOM. We recommend saving the output as an Excel file with filtering capabiliities. The *UserComment* field is meant to record reasons why certain observations are deleted.

We imagine the workflow to involve initially filtering on all the blank UserProcStatus rows. Then, for each tag, determine whether each observation should be kept (mark *UserProcStatus* == `TRUE`) or discarded (mark *UserProcStatus* == 'FALSE') based on suggestions from the *AutoProcStatus*, *ValidPaths* and *ModelObs* fields. For any observation destined to be discarded, a reason should be provided in the *UserComment* for re-producibility. When no blank *UserProcStatus* rows remain, save the file. It is now ready for importation into DABOM.

```
library(WriteXLS)

save_df = spwn_paths %>%
  select(TagID, TrapDate, ObsDate:SiteID, Node, SiteName, SiteDescription, NodeOrder:ModelObs) %>%
  full_join(fish_paths %>%
              rename(ObsDate = MinObsDate)) %>%
  arrange(TrapDate, TagID, ObsDate)

WriteXLS('save_df',
         'PITcleanr_output.xlsx',
         AdjWidth = T,
         AutoFilter = T,
         BoldHeaderRow = T,
         FreezeCol = 1,
         FreezeRow = 1)
```