Stream of Consciousness Model Explanation: Nisqually

A document to just dump model explanations according to coding order so the thought process is recorded somewhere anywhere and not lost to time. Created on 9/9/2024

# PinnipedCaseStudies/Nisqually

All files to run the Nisqually iteration of the model are in the GitHub repo “PinnipedCaseStudies” [Nisqually folder.](https://github.com/lizallyn/Pinniped-Case-Studies/tree/main/Nisqually)

Exception is that the underlying data file is in the [Data/Nisqually](https://github.com/lizallyn/Pinniped-Case-Studies/tree/main/Data/Nisqually) folder

# 00 Run The Model

The file “00\_Run\_The\_Model\_Nis\_All.R” sources each of the scripts required to run the model in order. You could just source this one script and the whole thing would run. The bottom of the script is assorted plots and tables for checking that all things ran well.

All files with numbers preceding the file name are specific versions for Nisqually. Ones without numbers can be found in the [Functions](https://github.com/lizallyn/Pinniped-Case-Studies/tree/main/Functions) folder and are used in common with the Locks model runs.

# 01 Set-up Functions

This section incorporates the functions for setting up variables – makeArray and createHarvestPlan (which both just create repetitive arrays) – and the predictFish function which was altered for Nisqually slightly (in ways that will likely be carried back over to the Locks).

predictFish houses the functions that fit a curve to the salmon arrival data for each run in the system and then create a vector of that data over the timeframe that the loop will run over.

# 02 Prep Salmon Data

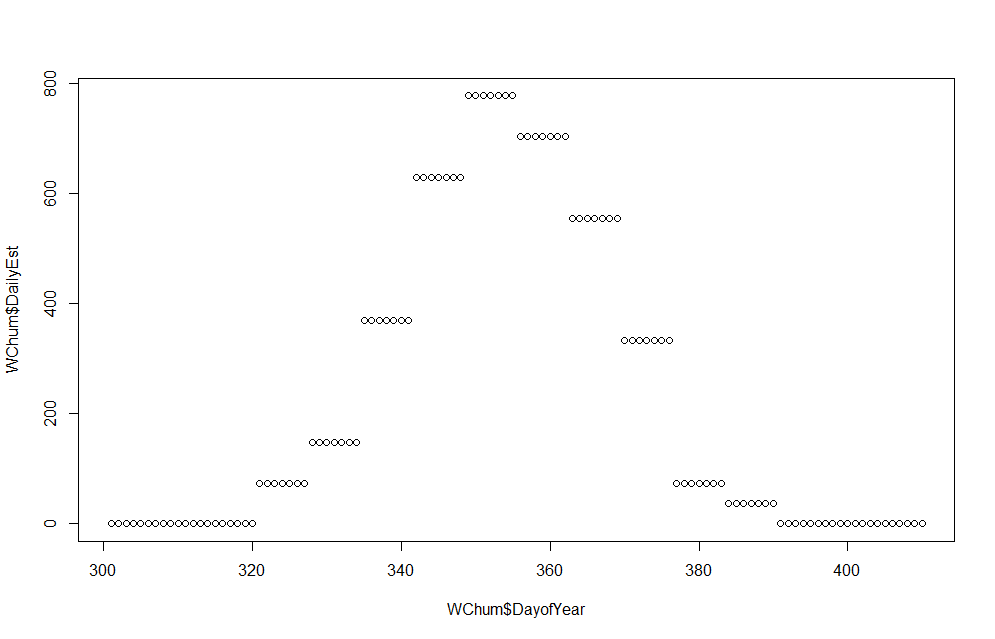
This script takes the salmon source data and turns it into daily predicted salmon arrivals, daily salmon catch, daily harvester presence.

First it calls the data from Data/Nisqually folder and saves it as fish.wide.

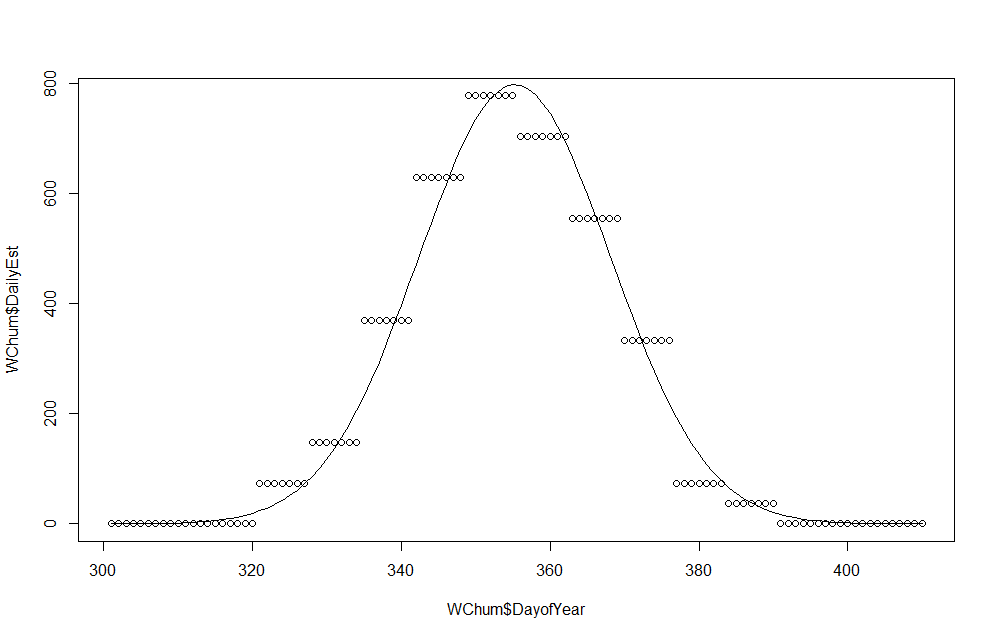
dates\_buffer is how many days on either side of salmon presence in the system do I want to incorporate in the loop. Example: if salmon arrive in the system on day 100 according to the observational data, and my days\_buffer was 20, then the loop would start on day 80 and similarly end 20 days after the salmon leave the system.

Next few rows are formatting dates as dates in R and then fixing the day of year column so that the loop can run through the new year and the day of year will continue past 366. Then it adds rows of no salmon presence to the salmon data frame according to how large days\_buffer is.

For each salmon species, the script defines the time frame that salmon are present e.g. chum\_start, chum\_end. Also defines residence time of the fish in days according to observations/expert opinion/tag data if relevant. This is used to define the escape rate as 1/residence time. Then it creates the data frame for that run of salmon with the new blank rows and creates a rounded integer daily count value. For each run, there is also code that quickly plots this to decide if fitting a normal curve to the arrival data is going to make sense at all. Example for winter chum below.



Next the script focuses on fitting the curve to the arrival data. This starts by defining starting parameters for the optim fitting process. These were initially defined through an iterative process, but now are hard coded in at a good starting point to expedite the fit. Fish.fit.optim is defined in the predictFish script and is used to spit out final parameters for each run. Then there is code to plot predicted curve over the initial plot to check the fit, see chum example below.



If this looks good, then save the predicted daily fish arrival to a new vector to be added to a full data frame of all runs at the bottom of the script.

Once all predicted fish are defined and saved, they are saved together in the Daily\_Fish data frame. This data frame runs from the minimum start day of all runs in the system to the maximum end day of all runs in the system.

Then it adds the fishing rates from each run. In this iteration the fish catch (in count of fish landed) is pulled directly from the csv that was initially called and saved as “fish.wide” (the wide is leftover from an earlier iteration that transformed it into a long version but that’s not necessary anymore and I never changed it). This matches the catch from each day into the Daily\_Fish data frame.

Then we estimate the number of harvesters in the system on each day (humans harvesting pinnipeds that is). This is equal to the count of fishers in the system on each fishery day according to data from Craig that is not in fish.wide but can be found in the Data/Nisqually folder in the file “Chinook and Chum Catches for Liz.xlsx” including comments from Craig. I decided to use an average for Chinook/Coho fishery since Craig seemed to be roughly estimating, and a random sample from the range of values for Chum (1-25) for each day fishers are active (catch in fish.wide is >0).

There are many different excel files from Craig with similar names from this process. Only look at the ones explicitly mentioned/called, the others are older versions and don’t incorporate the correct time frame.

# 03 Set Model Parameters

This section sets the parameters for the model and for the variable initialization process.

Loop parameters define the loop time frame and day range according to the length of fish data in Daily\_Fish. This should make the loop the same length as Daily\_Fish. It allows me to use start\_loop as a way to translate between Day of Year and loop day (t) by subtracting (start\_loop – 1) from any Day of Year value. Days is used in the loop itself.

Seal parameters define the number of seals in the source population and the proportion of them that are likely exhibiting salmon specialist behaviors and therefore may have a higher proclivity for going to the gauntlet right at the start of things. The source population here is defined/conceptualized as nearby haulouts where those individuals are likely accessing the gauntlet as a frequent foraging location. Based on observations from Craig, Jed, and Walker of the individuals hauling out in the delta area or visible in the system. Rough estimate.

Sea lions defines the number of California (zc) and steller (ej) sea lions in the source pools. Source pools here are more direct count of individuals observed in the gauntlet yearly currently, based on counts from nearby barge haulout.

Seal consumption parameters defines the delta\_t on the rungeKutta evaluation interval. Also defines the alpha (search and capture rate), Cmax for harbor seals (maximum daily consumption in number of salmon eaten) and gamma (predator dependence). Alpha and gamma are also shared with sea lions, but Cmax for each sea lion species is defined separately.

Seal learning parameters are defined next, which dictate how the y and x change each iteration and how they map onto P.

* Specialist\_prob is the baseline/minimum probability that a specialist individual (specialist seal or any sea lion) goes to the gauntlet.
* Baseline\_x\_val and baseline\_y\_val define the starting or neutral value that the x or y decays back to in absence of recent gauntlet experience. The specialist\_baseline\_y is an option to have them decay to a different value that allows them to essentially ignore or have a higher tolerance of risk. Currently set at 0 to be the same as normal though.
* W represents the relative goodness of the gauntlet compared to other foraging opportunities outside the gauntlet. So in order to have a good foraging day, an individual has to consume more than this much salmon, otherwise they could have gone elsewhere and been as full or fuller.
* Ymin, ymax, xmax define the range of the x and y variables. Intercept\_x\_val is the intercept of the linear relationship between x and P\_x. steepness and threshold\_val and threshold\_specialist describe the logistic relationship between y and P\_y. threshold\_x\_specialist describes the logistic relationship between x and P\_x for specieslists only. More parameters for these relationships are derived from these and are thus in the variables script so they can be manipulated separately and have those manipulations propagate.
* Step and decay describe how x and y change when given new input information from each foraging day. Step on days when they individual went to the gauntlet, decay on days when they didn’t.
* Buffer Px and Py are the minimum value that each can go to. This allows specialists to never develop an aversion to hunt activity or to chose the gauntlet even in the face of poor foraging performance.

Social learning parameters include the number of seals to copy (which is based on the haulout size). Also includes the mean and beta values that describe the beta distribution used for the receptivity term.

Next the escape rates for each salmon run are defined using 1/residence time in days. Natural mortality is an utter guess. No data there and such a small time/space snapshot. Conceptually this includes disease, injury, other predators. For chum I guess this also includes failure to spawn for reasons other than behavioral disturbance or being eaten.

Sealion\_arrival\_buffer is the number of days the sea lions might arrive before the chum arrive. This is used to add the impact that an impending chum arrival has on the other runs in that system, i.e. natural run chinook. 10 days is an utter guess. Very dependent on run size and timing of other runs I would bet, but no data for that explicitly.

Hunting parameters include the zone efficiency, or what proportion of the existing pinnipeds that went to the gauntlet might be able to be removed through a management removals directed effort. Steepness and efficiency describe the logistic relationship between the number of harvester vessels in the gauntlet and the number of pinniped harvested – logistic so it saturates. Scenario defines that we are describing a scenario where only boat based harvest is happening. This is why zone\_efficiency is NA right now.

# 04 Initialize Variables

This script starts by defining parameters that depend on other parameters. This ensures that they can be manipulated independently in the single parameter manipulations and actually propagate that change through the model.

Number of sea lions to copy is all of the existing sea lions in each source pool.

The number of seal specialists is the number of seals multiplied by the proportion of specialists

Sea lion arrival date and date adjusted for loop day uses the sealion arrival buffer value

Harvest days for each pinniped species are defined separately but identically in the current run by using the number of harvesters that was put into the Daily\_Fish data frame during the data prep script. Any day with > 0 harvesters is a day with pinniped harvest potential.

Slope\_x\_val and xmin describe the range of x and how it maps onto P\_x. slope is derived from the intercept, and xmin is derived from them both, which is why this is in the following script.

The next few lines bundle all the parameters for the functions that calculate the change in x and y each day and the ones that map x and y onto their corresponding P value so that they aren’t all being listed as separate arguments in those functions in the loop. This is purely for the simplicity/aesthetics/tidiness in the loop script.

The next few lines build a bunch of blank arrays that will save various rates and counts from each day and/or individual pinniped so they can be plotted and summarized and analyzed after the run.

The identifiers of the specialist seals are calculated here. Then those identities are used to replace the baseline or starting values of other variables with the specialist-specific rate (i.e. baseline\_y, buffer\_Pymin, threshold).

Harvest creates that harvest plan for each pinniped based on the days when harvest happens for each species and the scenario defined in the parameter script.

# 05 Loop Functions

These are mostly generic and pulled over from the Locks run. But each function accomplishes a part of the loop. I’ll describe more during the loop script itself.

The only Nisqually-specific function here is the salmonSpeciesUpdate function, which is specific to the runs of salmon included in this system. This function just grabs the arriving salmon counts from the Daily\_Fish data and adds them to the existing gauntlet salmon. This is a separate function purely for aesthetics/tidiness in the loop.

# 06 Run The Loop

The loop runs from 1:days-1, so the t counting system is based on t+1 not t-1.

First the salmon are updated using the salmonSpeciesUpdate function which adds arriving salmon from Daily\_Fish data frame for the corresponding row t. These values are unpacked into the corresponding gauntlet salmon counter list.

Next the pinnipeds make their initial foraging location decision. This uses the function decideForagingDestination which essentially flips a coin on the prob\_gauntlet using runif(). This has to happen first, even though it almost always gets overwritten by the copying, because if num\_2\_copy is <0 then there needs to be a default value.

Then there is a round of copying or social contagion of the foraging decisions. This uses the function collusion(). Collusion calculates a beta curve based on the parameters provided, and returns the density at the given individual probability x value. That density is the receptivity value, which is used to scale the way that the social information is incorporated into the new prob\_gauntlet of that individual. The social information is simply the mean of the prob\_gauntlets of a randomly selected group of pinnipeds, where the size of the group is determined through num\_2\_copy. This function spits out the new P\_social (which is the prob\_gauntlet with social info included) and then uses that to decide the foraging destination using that new prob.

This whole social collusion process only happens for sea lions once they have arrived at the system, i.e. t > sealion\_arrival\_loopday. If that’s true, the same process as above happens for both sea lion species.

Next there is a whole code chunk that just keeps track of which and how many pinnipeds are at the gauntlet.

Then the catch rate for the day is calculated using the catch data from Craig. The landed salmon for that day of the year is divided by the gauntlet\_salmon for that run to get a rate which is used in the rungeKutta function.

All the information about how many salmon and pinnipeds are at the gauntlet and the various rates are fed to the rungeKutta function. This calculates dx/dt at 4 specific points within the time interval delta\_t (1 hour) and calculates each of the salmon rates that we care about for all runs and pinniped species.

The next chunk then takes the results from that function call and saves them to the relevant variables. Consumed salmon are distributed among the pinnipeds that came to the gauntlet.

Pinnipeds are harvested according to the harvest plan. getHarvested takes the scenario information and calculates that harvest under that scenario. For “boat”, the harvest is calculated using a saturation curve as defined in parameters. This rounds down “floor” so that the harvest cannot end up being more than the pinnipeds present due to rounding error. The number harvested is then assigned to specific pinnipeds for each species, and their IDs are added to the kill\_list for their species.

Next, we update learning. This is done for each species separately by looping through each individual. The y shape parameters for normal seals were not bundled in the variable script because they depend on whether the seal is a specialist or not, so those are bundled within the loop. Sea lions were bundled in variables because they are all specialists. All the parameters are passed to updateLearning, which does the following:

* calculates the adjusted consumption (salmon consumed minus w), then uses that consumption C to calculate d\_x using the learnX function.
* learnX takes an individual pinniped’s information and calculates the change in x based on their foraging location and foraging success and existing x value.
* learnY takes an individual pinniped’s information and the hunting record from the day and calculates the change in y based on their experience with harvest risk and existing y value.
* Updates x\_t and y\_t with the dx or dy calculated above
* Calculates P\_x and P\_y using the appropriate function depending on whether they are a specialist or not.
* Returns a tibble of the values calculated above

The outputs from updateLearning are then assigned to the relevant variable. If the pinniped individual died, then each value that influences their future decisions is replaced with “NA”.

Then the loop repeats!

# 07 Plotting

This section creates plotting functions, then runs those functions for each functional group in separate scripts. This is only separated out because I have a burning hatred for unendingly long plotting scripts. HATe.

Below this is a section where I call the various tables and plots to check that everything ran well.