

# Developing Harvest Control Rules for Alberta Walleye Fisheries

Christopher L. Cahill and Carl J. Walters

20 March 2022

## IN PROGRESS—Cahill must finish by March 31

### TODO

- parallel ordering of MAY / HARA
- Carl's point about including precautionary rules
- proofread – check citations

### Purpose

The purpose of this tutorial is to develop a management strategy evaluation or simulation framework that can be used to evaluate harvest control rules for any of the Alberta Walleye fisheries assessed in Cahill et al. 2021 using either yield or catch-based objectives.

### Goals

- Provide readers with some background on the development of policies for fisheries management
- Understand what a harvest control rule is and why such rules are useful for managing natural resources
- Understand how the code is pulling results from the stock assessments and using these as inputs into our harvest control rule simulation program
- Work through the model structure of the harvest control rule simulator, including calculations of maximum average yield and HARA utility
- Develop intuition for the catch and release mortality components of the simulation model
- Understand the tabular or grid search approach to “optimizing” simple linear harvest control rules

### Background

Harvest control rules are agreed-upon, transparent, and repeatable mathematical equations that help managers alter harvest levels in response to changes in stock size to achieve some explicit goal. Changes in population size may be due to either environmental stochasticity or the effects of fishing. In Alberta, harvest control rules may be particularly useful for informing special harvest license allocations given some agreed upon and explicit management objectives and Fall Walleye Index Netting (FWIN) survey data. The development of explicit harvesting rules is important because a finding of Cahill et al. 2021 was that many Alberta Walleye lakes appeared underharvested relative to common fisheries reference points.

Fisheries science has a rich history of using harvest control rules or feedback policies to design effective resource management systems, and there are some key references that will help you better understand many of the issues that we will discuss in this tutorial. These references are:

1. Walters 1986, chapter on feedback policies
2. Hilborn and Walters 1992, chapter on designing effective fisheries management systems
3. Quinn and Deriso 1999, chapter 11 on optimal harvesting
4. Walters and Martell 2001, chapters 2-4

In general, you will find that it is much more difficult to design effective fisheries management systems and harvest control rules than it is to simply assess a particular fishery. This is because what we are actually doing is attempting to design a control rule (a “controller”) for a nonlinear dynamical system.

## **A nonmathematical description of the problem that we would share with Grandma**

You can think of the harvest control rule problem like trying to design an autopilot system (or a control rule or policy) that keeps a plane air born—our goal in this case is to create a mathematical rule that relates measurements from the environment taken by sensors (altitude, jet speed, wind speed, wind direction, etc.) to actions the plane can take to alter its course of movement (increase speed, decrease speed, turn left/right/up/down). The point here that really matters is that we are trying to design a feedback policy, which relates where we are (in the environment) to what we want to achieve.

In this case, the “environment” simply refers to the physical equations governing the flight of a plane. In our fisheries situation, it refers to all of the equations and ecological processes describing the dynamics of a harvested population. Whereas there are sensors on the plane which tell us where the plane is and what the state of the environment is outside of that plane, we use surveys and sometimes stock assessments to determine the state of a population in fish and wildlife management settings.

In the oversimplified case of the autopilot system, the goal is not to crash. Intuitively, this might mean that a good feedback policy for the plane would seek to orient the plane to the right if the wind pushed it to the left, and try to maneuver upward if the plane were decreasing in altitude for whatever reason. In a fishery, the goal might be to let the stock size recover if it has been overfished, or to harvest more if the stock size were thought to be high.

## **Harvesting theory, dynamic programming, and optimal solutions**

A great deal of theoretical work on optimal harvesting strategies for dynamic populations shows that optimal policies generally decrease harvest rate when population size declines, and increase it as population size increases. These findings have often come from dynamic programming solutions, and the solutions for these particular problems are often simple functions of harvestable abundance or biomass. This finding appears general and was surprising—it was generally believed that complex policies would be required to manage complex ecological systems. However, work on harvesting theory during the 1970-80s showed that simple policies performed well or were even optimal in many situations (e.g., Walters 1975; Walters and Hilborn 1978; Mangel XXXX; Clark XXXX; Moxnes XXXX).

A limitation of dynamic programming and the related methods discussed above is that these approaches can only be applied to relatively simple models and management objectives. The age-structured Bayesian Estimation of Recruitment Trends in Alberta (‘BERTA’) model is too complex to use with such methods. Consequently, we will instead use a simulation technique known as approximation in policy space to find good harvest control rules (see methods in Moxnes XXXX; Edwards and Dankel 20XX; new fancy book) for the BERTA models.

We briefly hit on this above, but there is one more difficulty facing us and that is that there are multiple (often conflicting) objectives in resource management. We note that the management strategy simulation approach we use in this tutorial can cope with this issue quite nicely if folks have explicit objectives.

## On the need for explicit objectives

In the above toy airplane autopilot example, we wouldn't get very far if we didn't agree up front that the goal was to keep the plane flying within some (acceptable) flight parameters. The same is true for designing harvest control rules for renewable resources: if we do not agree upon how to manage a population a priori, we will not get very far. If we want to design effective fisheries management systems, we at least need to agree upon how best to manage those fisheries. This is easier said than done, and individuals have dedicated their entire careers toward developing structured decision making programs to meaningfully engage relevant stakeholders and develop such objectives.

For Alberta Walleye, there are presently no agreed upon *explicit* objectives for fisheries management. Believe it or not, this is pretty common for many fisheries. Thus, we will introduce two concepts—Maximum Average Yield (MAY) and Hyperbolic Absolute Risk Aversion Utility (HARA)—as starting points for Alberta Walleye fisheries. As stated above, objectives should be set through something like a structured decision making program (see Edwards and Dankel 20XX). It is unlikely that Cahill and Walters (or any manager or researcher for that matter) should be setting the objectives for Alberta fisheries.

Our approach here is to specify two provisional, albeit conflicting objectives to demonstrate how Alberta Walleye fisheries could be managed given explicit objectives. Later on we will simulate across a range of policy options to find policies that do well in our simulations given these provisional objectives. We discuss both of these objectives in the following section.

### Maximum Average Yield (MAY) objective

Maximum Average Yield is the stochastic analog of Maximum Sustainable Yield (MSY). In a harvested population, this is the highest yield that can be taken on average from a population experiencing average environmental conditions. While there are many criticisms of MSY or MAY (e.g., Larkin 19XX), such yield-maximization policies demonstrate how much yield can be achieved from a given fishery if a MAY policy was adopted. When you simulate yield maximization policies or attempt to solve this problem using dynamic programming, what you find is that the optimal policy often is some type of “fixed escapement” policy. That is, there is a lower limit reference point below which no harvesting occurs, and above this lower limit reference point you tend to harvest at a rate near  $F_{msy}$ , or the instantaneous fishing mortality rate thought to achieve MSY.

While such policies maximize the average yield that can be taken from dynamic, uncertain, or even unpredictable fish populations, they have the downside that they result in increased variability to harvesters in terms of fisheries closures as stock size drops below a fixed lower limit reference point. Remember, if the stock drops below the lower limit reference point the MAY policy shuts off harvesting completely.

### Hyperbolic Absolute Risk Aversion (HARA) utility objective

Hyperbolic Absolute Risk Aversion for catch (herein-after, HARA) is an objective that values consistency in catch through time from the perspective of a harvester. It is critical to include risk-averse utility for catch as a performance measure representing the interests of stakeholders and harvesters so as to explicitly consider stakeholder interests. The simplest way to do this is to calculate utility each year using a hyperbolic absolute risk averse utility function, which boils down to the following equation:

$$utility_t = catch_t^{pp}$$

In this case,  $catch_t$  is simply yield in year  $t$ , and the power term  $pp$  is a risk aversion parameter that for most people is near 0.3-0.5. You can empirically assess  $pp$  by asking people to choose between two income options: a 50:50 gamble between 0 and 100,000 dollars, or a sure income of around 30,000 dollars. The expected value of the income outcome of the 50:50 gamble is 50,000 (i.e.,  $1/2 * 100,000$ ), but most people

will choose a guaranteed income of around 30,000 dollars vs. the 50:50 chance at winning the 100,000 dollars (myself included).

This scenario implies  $pp = 0.57$ . You can calculate someone's  $pp$  by asking how much guaranteed reward (catch, income) they would need to receive before they were unwilling to take the “unsure” bet—so if you said your  $X = 20,000$  dollars (or catch of fish), your  $pp$  would be

$$pp = \frac{\ln(0.5)}{(\ln(20) - \ln(100))} = 0.43$$

And note that we have substituted “20” for 20,000 and “100” for 100,000 because the math works out to be the same. When we ask fishermen what  $X$  they would need to not fish, typical answers have been in the 20,000-30,000 range, i.e. having some income or catch is much more important than maximizing average income if that means having highly variable income or catch. When framed this way, HARA utility is a powerful tool for capturing aversion to variability in catch.

When HARA utility is maximized as an objective, it tends to do two things relative to the MAY objective discussed above. First, the HARA objective tends to shift the lower limit reference point toward zero. This is because if we “shut off” fishing we lose a great deal of utility, which I hope makes sense. The second thing that this objective tends to do when you undertake formal optimization (i.e., dynamic programming) is that it generally speaking lowers the rate at which you increase harvest as stock size increases. We will demonstrate this via some simple R code shortly and via simulation later on.

## Simulating some simple (fake) MAY vs. HARA policies

We picked these two objectives (MAY, HARA) to represent management tradeoffs (see Walters and Martell 2001). MAY represents maximum yield that say a corporation exploiting many populations at once might choose to maximize expected catch from a collection of populations, while HARA represents a better policy for a risk averse fisher or angler. To harvest under a harvest control rule that was developed to achieve MSY or MAY is to concede that a fishery will need to be closed for potentially many years if stock sizes drop below the lower limit reference point. This is particularly problematic for fisheries with highly variable recruitment dynamics like Alberta Walleye, as closures might be triggered simply due to natural variability.

Conversely, to harvest under a HARA objective and corresponding policy is to harvest at a lower rate but to continue to harvest when biomass drops below the lower limit reference point required to achieve MAY. These are key lessons from theoretical studies of harvested populations (e.g., see Walters and Hilborn 1976; 1978; Walters 1986), and we will demonstrate these principles in excruciating detail as we proceed with our Alberta Walleye example below.

## Let's plot it in R

If you are a visual learner, you might have an easier time with the following hypothetical depiction of fixed escapement MAY vs. a HARA policy in R.

Let's load some packages:

```
# packages
library(tidyverse)
```

And then simulate and plot some dead simple harvest control rules (I made these up):

```

vB <- seq(from=0.1, to =100, length.out = 100) # biomass vulnerable to fishing
b_lrp <- c(0, 30) # lower limit reference point
c_slopes <- c(0.2, 1) # slope of harvest control rule

TAC_HARA <- c_slopes[1] * (vB - b_lrp[1]) # total allowable catch HARA
U_HARA <- TAC_HARA / vB # exploitation rate HARA

TAC_MAY <- c_slopes[2] * (vB - b_lrp[2])
TAC_MAY[which(TAC_MAY < 0)] <- NA # Set negative values to NA (don't harvest below blrp)
U_MAY <- TAC_MAY / vB

# make a big data frame
data_HARA <- data.frame(vB, TAC = TAC_HARA, U = U_HARA, policy = "HARA")
data_MAY <- data.frame(vB, TAC = TAC_MAY, U = U_MAY, policy = "MAY")
data <- rbind(data_HARA, data_MAY)

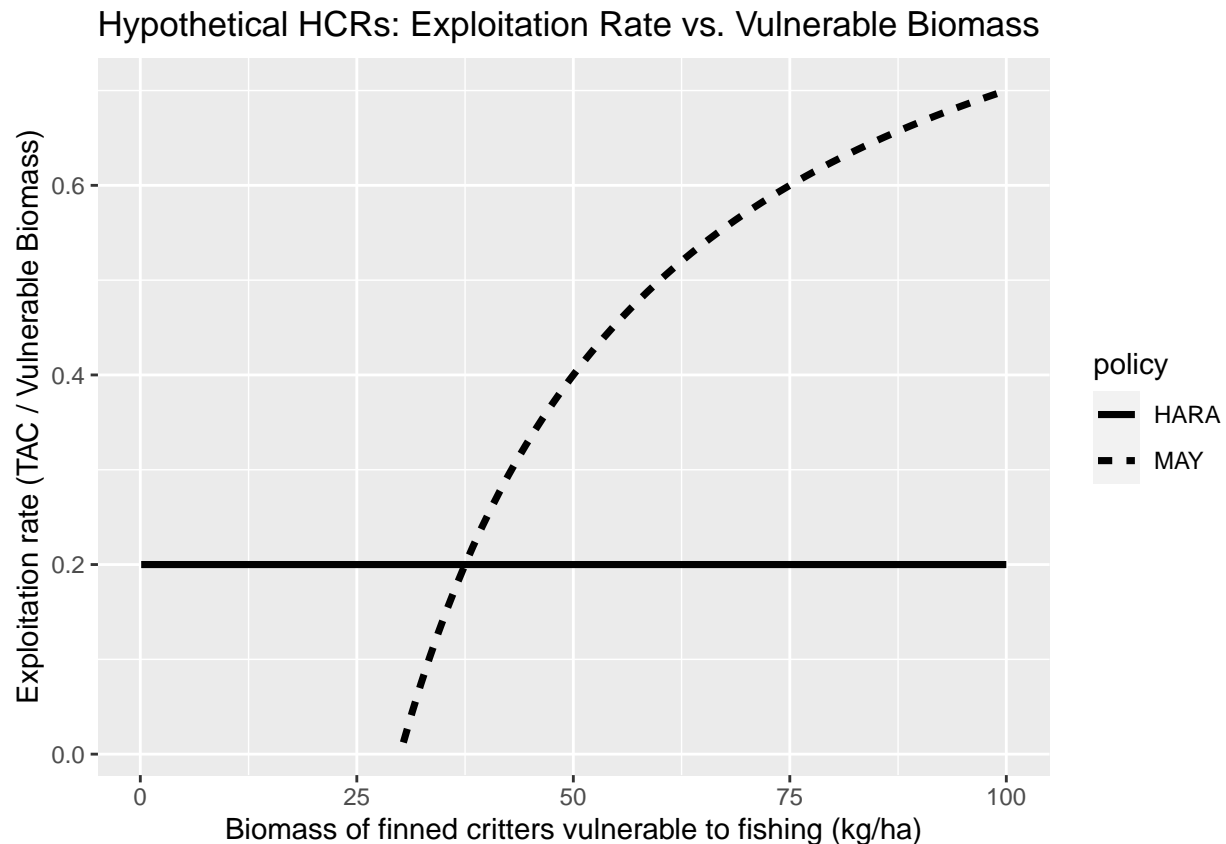
data %>%
  ggplot(aes(x=vB, y = TAC, group = policy))+
  geom_line(aes(linetype = policy), lwd = 1.25) +
  ylab("Total allowable catch (kg/ha)") +
  xlab("Biomass of finned critters vulnerable to fishing (kg/ha)") +
  ggtitle("Hypothetical HCRs: Total Allowable Catch vs. Vulnerable Biomass")

```



In this example, there is a lower limit reference point  $b_{lrp}$  and a slope at which you increase harvest as stock size increases ( $c_{slope}$  in the code). Next we can plot exploitation rate as a function of stock size for these simple examples using this code:

```
data %>%
  ggplot(aes(x=vB, y = U, group = policy))+
  geom_line(aes(linetype = policy), lwd = 1.25) +
  ylab("Exploitation rate (TAC / Vulnerable Biomass)") +
  xlab("Biomass of finned critters vulnerable to fishing (kg/ha)") +
  ggtitle("Hypothetical HCRs: Exploitation Rate vs. Vulnerable Biomass")
```



Think about these last two plots a bit—it is important to wrap your head around what this is showing you. Most of the harvest control rule simulation script that we will go through below is shifting the intercepts and slopes of these lines from Figure 1 to find simple linear policies that maximize specific objectives like MAY or HARA utility.

## Thinking through one more harvest control rule

While the policies above arise from explicit objectives and dynamic programming solutions (again, take my word for it or go read all that stuff I referenced at the beginning of the tutorial), we will take this opportunity to work through one more policy that you might encounter in fisheries. This is DFO's so-called precautionary harvest control rule (DFO XXXX), which appears to be loosely based on work from Restrepo XXXX. This policy has been adopted for fisheries throughout Canada (and regardless of the management objectives of those fisheries). The DFO policy specifies a rectilinear harvest control rule, and while it is often unclear what is specifically supposed to be precautionary about this rule we will simulate it anyway to show you what it is doing.

The DFO policy increases exploitation rate from zero to  $U_{MSY}$  as biomass increases from a lower limit reference point to an upper limit reference point. These reference points are typically set at  $0.4 \cdot B_{MSY}$  and  $0.8 \cdot B_{MSY}$ . Note that  $U_{MSY}$  and  $B_{MSY}$  typically need to be estimated in an assessment model.

In our simple example, we will say  $U_{MSY} = 0.2$ , and that  $B_{MSY} = 30$  kg/ha. Thus, we would specify the DFO harvest control rule as follows:

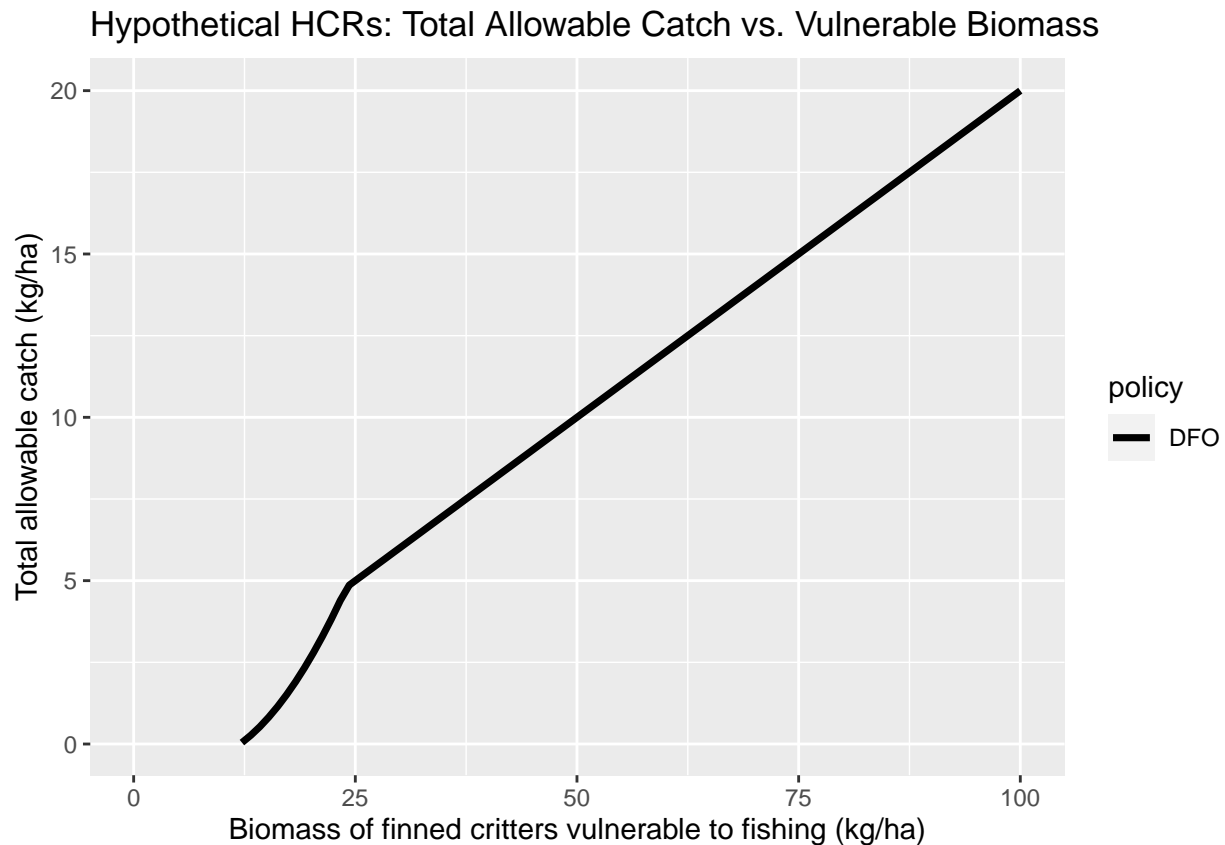
```
U_MSY <- 0.2
B_MSY <- 30
b_lrp <- 0.4*B_MSY
u_lrp <- 0.8*B_MSY

U_DFO <- U_MSY * (vB - b_lrp) / (u_lrp - b_lrp)
U_DFO[which(U_DFO < 0)] <- NA # Set negative values to NA
U_DFO[which(U_DFO > U_MSY)] <- U_MSY # Set negative values to NA

# calculate the TAC as U*vB
data_DFO <- data.frame(vB, TAC = U_DFO*vB, U = U_DFO, policy = "DFO")
data <- rbind(data, data_DFO)
```

We can then plot it like we did before:

```
data %>%
  filter(policy == "DFO") %>%
  ggplot(aes(x=vB, y = TAC, group = policy))+
  geom_line(aes(linetype = policy), lwd = 1.25) +
  ylab("Total allowable catch (kg/ha)") +
  xlab("Biomass of finned critters vulnerable to fishing (kg/ha)") +
  ggtitle("Hypothetical HCRs: Total Allowable Catch vs. Vulnerable Biomass")
```



This plot is admittedly a bit weird looking, but that's just how this policy plays itself out. We can also plot the exploitation rate vs. vulnerable biomass for this policy as well:

```
data %>%
  filter(policy == "DFO") %>%
  ggplot(aes(x=vB, y = U, group = policy))+
  geom_line(aes(linetype = policy), lwd = 1.25) +
  ylab("Exploitation rate (TAC / Vulnerable Biomass)") +
  xlab("Biomass of finned critters vulnerable to fishing (kg/ha)") +
  ggtitle("Hypothetical HCRs: Exploitation Rate vs. Vulnerable Biomass")
```



And you can see where the name “rectilinear” harvest control rule comes from. So in this example, there’s no harvest below some lower limit reference point, and then we “cap” the maximum harvest rate at  $U_{MSY}$ .

Remember, this is the harvest control rule used for most fisheries managed via a harvest control rule by DFO. What does this rectilinear rule imply from a HARA or MAY perspective? What does it imply for other objectives?

Is there a biological basis for the  $0.4 \cdot B_{MSY}$  and  $0.8 \cdot B_{MSY}$  cutoff values?

## How do we use our Bayesian assessment models to inform harvest control rules?

Before we begin working through the harvest control rule simulation script, we need to understand what we currently have in terms of the analyses we have completed up to this point. In previous tutorials we have estimated a straightforward age-structured stock assessment model for Alberta Walleye. We have saved the posteriors from these model fits for each lake in their own .rdata files in the `fits` folder on the Github for this project.



The posteriors that we have saved capture uncertainty in Walleye population dynamics given our model structure, priors, and data for a given lake. Thus, we can use results from these assessments to initialize a harvest control rule simulation, and run these simulations for many random draws from the posterior to characterize uncertainty in our policy development. We will undertake a specific type of management strategy evaluation known as retrospective harvest control rule simulation. All this means is that once we initialize our simulation model using our stock assessment results, we will repeat recruitment anomaly sequences  $w_t$  some number of times into the future. We will play around with this a bit, but the primary goal here is to develop harvest policies that are robust to the recruitment dynamics that were estimated in Cahill et al. 2021.

Conceptually, the way we will do this is as follows:

1. Initialize our population dynamics simulation model using assessment model results for some lake.
2. Project this model into the future using (nearly) the same equations as in Cahill et al. 2021.
3. For each policy we want to test the performance of, we initialize our simulation model using some number of random draws of the posterior, run the simulation forward in time while implementing that specific policy for each random draw, and record average performance in terms of our objectives over years of our simulation and number of draws.
4. Look across all policies we tested, and look up the policy that maximized average yield or HARA utility, respectively.
5. Go for a long run and drink tasty beer, never think about designing controllers for nonlinear dynamical systems again.
6. Upon recovering, go back and deal with the rectilinear policy, and see how well the DFO rule works for Alberta Walleye relative to the simple rules above.

This will get a little tedious as we work through code, but the key point to remember here is that we are trying to create a rule for managers that says “if our objective is  $x$  and our net catch is  $y$ , how many fish should we allocate in year  $t$ ?” We will relax a variety of assumptions around this (assessment interval, survey variability, and catch and release mortality), but the main point is we are trying to come up with an equation that helps folks achieve some objective(s) given a net catch.

Additionally, while we are only going to use two conflicting objectives, it is straightforward to do this for any specific objective(s) you want. Phrased differently, if you have explicit multi-criteria objectives and some weighting scheme you could easily adapt this script to test the performance of such an fancy objective.

## The `hcr.R` simulation script

The `hcr.R` script works like the `run.R` script in our earlier tutorials. The `get_hcr()` function takes in some inputs, reads in a specific Bayesian model fit from a given lake, extracts the relevant stuff that we need to parameterize an age structured model, and then runs our simulations forward in time for some number of draws and records performance of that simulation. It then saves the simulation output as a `.rds` file with a unique file name (based on whatever simulation choices you chose) in a `sim/` folder. The full function is available here:

<https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L17-L359>

This function was primarily developed to test the simple linear harvest control rules (see MAY vs. HARA examples above), but it can also be used to test the DFO-style rectilinear harvest control rule. For now, we’ll ignore the rectilinear rule entirely and just focus on the simple linear rules. Once we work through the linear harvest control rules and get a handle on how this works, we’ll loop back at the end of this tutorial deal with the rectilinear policy. Eventually we will be able to compare the performance of the linear vs. rectilinear harvest control rules for Alberta Walleye fisheries.

Before we start jumping through this function line by line, I want you to look at the following lines of code below the function here.

<https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L361-L432>

Please note you don't have to know what all of this is doing yet. The main reason to show you this is that there is eventually going to be stuff in our R environment (a bunch of simulation control parameters and our Bayesian model fits in list form) that our `get_hcr()` function is going to interact with. If I didn't show you this up front, you might get confused as we work through the function. So keep that in mind.

The first chunk of code in `get_hcr()` is the following:

```
get_hcr <- function(which_lake = "lac_ste._anne", ass_int = 1,
                    sd_survey = 0.05, d_mort = 0.3,
                    rule = c("linear", "rectilinear")) {
  # many more lines of code...
```

These are inputs you can change depending on your what you want. The arguments that you can play around with are:

- `which_lake`, a string that specifies the lake name that you want to run. This needs to be lowercase and have underscores in place of spaces.
- `ass_int`, describes the interval at which we will assess the fishery.
- `sd_survey`, describes the amount of noise in the survey that we base our harvest control rules off of.
- `d_mort`, the discard mortality rate for released fish.
- `rule`, specifies between linear or rectilinear harvest control rules.

The next few lines of the function specify a bunch of additional simulation parameters:

```
#-----
# initialize stuff
#-----
n_draws <- hcr_pars$n_draws
n_repeats <- hcr_pars$n_repeats
retro_initial_yr <- hcr_pars$retro_initial_yr
retro_terminal_yr <- hcr_pars$retro_terminal_yr
n_sim_yrs <- hcr_pars$n_sim_yrs
ages <- hcr_pars$ages
n_ages <- length(ages)
initial_yr <- hcr_pars$initial_yr
initial_yr_minus_one <- hcr_pars$initial_yr_minus_one
ret_a <- hcr_pars$ret_a
Ut_overall <- hcr_pars$Ut_overall
sbo_prop <- hcr_pars$sbo_prop
rho <- hcr_pars$rho
sd_wt <- hcr_pars$sd_wt
psi_wt <- hcr_pars$psi_wt
n_historical_yrs <- length(retro_initial_yr:retro_terminal_yr)
grid_size <- hcr_pars$grid_size
```

These lines of code are looking up values from a tagged list called `hcr_pars`. I set the values in this function later on in the script in these lines:

<https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L386-L403>

You can change these if you want, but you'll notice that I haven't put these explicitly in the function call for `get_hcr()`. I did this because you should probably know what you are doing if you want to change these parameters. This might seem annoying, but the reason I did this is because if I needed to change any of

these values I needed all the following code in this function to update automatically. This was the best way I could think to do that, though I'm sure there are other (and perhaps more elegant) solutions.

I don't want you to worry too much about these for the moment. You may not like these initial values. So be it. The point of coding it this way is so that we can loop through and run many different simulations with many different input parameters.

The next code block is just taking the `which_lake` string, and using it to grab the appropriate Bayesian model fit (which eventually will be stored in our R environment from the fits folds):

```
#-----  
# subset lake-specific posterior from all fits  
#-----  
lake_str <- gsub(" ", "_", which_lake)  
fit_idx <- grep(lake_str, names(fits))  
fit <- fits[[fit_idx]]  
rec_ctl <- ifelse(grepl("bh", names(fits)[fit_idx]), "bh", "ricker")
```

The first line makes sure there are no spaces in the variable `which_lake`. The next line determines which of the fits (in the R environment) corresponds to the `which_lake` we wanted to run the simulation for. We then read in the fit corresponding to our lake, and then declare a control variable called `rec_ctl` indicating whether the fitted Bayesian model used a Beverton-Holt (i.e., “bh”) or Ricker stock-recruit model (i.e., “ricker”). Sweet.

The next ten or so lines are used for the rectilinear harvest control rule, and so we will ignore this for now. Note this code is only triggered when `rule = “rectilinear”`:

<https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L49-L62>

And so we will ignore them for now for sanity purposes. Again, we will come back to this later on.

Lines 64-144 or so are pretty annoying. Quite literally all these lines of code are doing is extracting relevant values from our Bayesian model fits that we are going to use in our simulations. Some of these terms were read into our Bayesian model as data, while others were explicitly estimated as parameters (and thus values will vary per each draw of the posterior).

I am using the `tidybayes` package to manage Bayesian model output. There are many other ways to do this, but I like this way, and so now you have to learn it if you want to figure out what I did. Let's assume for now that `n_draws = 1`. The next chunk of code is:

```
# extract estimated and derived parameters from BERTA  
devs <- fit %>%  
  spread_draws(Ro, ln_ar, br) %>%  
  sample_draws(n_draws)  
  
draw_idx <- unique(devs$.draw) # which posterior rows did sample_draws() take?
```

This piece of code takes a specific model fit, pipes it into the function `spread_draws()` to access this fit's posterior, which is then passed to `sample_draws()`. The `sample_draws()` function randomly samples some number of draws `n_draws` from the posterior (i.e., rows of the posterior where columns are parameter names and rows are parameter values). Note also that we are only grabbing three parameters in this case— $R_0$ ,  $\ln(ar)$ , and  $br$ . These values are stored in the object `devs`.

We then run one more line of code that figures out which specific draw(s) or rows of the posterior we randomly selected, and we store this value as `draw_idx`. This is important when we sample more than one parameter because we want our sample values to come from the same (randomly selected) rows of the posterior, which preserves co-variation among the parameter estimates used in our simulation (cool).

Next we basically do the same thing for all the other stuff we want to draw in from our Bayesian model fit, and we make sure we sample the same rows of the posterior as we did above via the `filter()` call:

```
w_devs <- fit %>%
  spread_draws(w[year]) %>%
  filter(.draw %in% draw_idx) %>% # force same .draw to be taken as above
  mutate(year = year + initial_yr_minus_one) # make years 1980-2028

# extract nta from stan
nta_stan <- fit %>%
  spread_draws(Nat_array[age, year]) %>%
  pivot_wider(names_from = age, values_from = Nat_array) %>%
  filter(.draw %in% draw_idx) %>%
  mutate(year = year + initial_yr_minus_one)

# which columns have ages:
age_cols <- which(!is.na(str_extract(
  string = colnames(nta_stan),
  pattern = "[0-9]|10[0-9]"
)))

# rename age columns to correct ages 2-20
colnames(nta_stan)[age_cols] <- ages
```

So here we are extracting the recruitment anomalies  $w_t$ 's and the estimated numbers at age by year matrix. We then rename the Stan fit indeces to make them intelligible. Remember, Stan begins its indeces at 1, so we have to do this extra step if we want Walleye ages to range from 2-20 and years to range from 1980-2028. We then extract even more stuff from the posterior to use in our management strategy evaluation simulator:

```
# extract MAP estimate of Ro from posterior (for indexing bmin)
Ro_summary <- rstan::summary(fit, pars = "Ro")$summary
Ro_map <- Ro_summary[, "mean"]

# extract the age structure corresponding to 1990
nta_init <- nta_stan %>% filter(year == retro_initial_yr)

# extract w estimates from 1990-2015
w_devs <- w_devs %>%
  filter(year %in% retro_initial_yr:retro_terminal_yr)
```

Here you'll note that the code is a little different, and that we are extracting a summary value for  $R_0$  rather than an entire posterior or some random subset of that posterior. In particular, we are extracting the most probable single estimate of  $R_0$  for use a little later on.

You'll note we are also subsetting our Bayesian posterior model fits a bit. Instead of using the full 1980-2018 numbers at age matrix and  $w_t$  values, we'll subset these estimates to correspond to years 1990-2015. We do this for two reasons.

First, most of the Fall Walleye Index Netting surveys had information on Walleye age structure and recruitment back to at least 1990. While there were some systems where survey data carried information on recruitment and age structure back to the early 1980s, our choice of 1990 seemed to be a safe early year bound based on our data explorations.

Second, recruitment is usually difficult to estimate until cohorts of fish have been present in a fishery for a few years, which means recruitment estimates from recent years (i.e., 2015-2018) are generally unreliable (see Hilborn and Walters 1992; Walters and Martell 2001).

As our main goal was to design harvest control rules robust to the recruitment variation present in these systems, we felt it was best to use this 1990-2015 range to initialize our age-structured simulation below because these were the estimates that we could trust the most given our data and models.

Next, we join all of that stuff into one tibble:

```
# join all those devs into one big tibble called "post"
suppressMessages(
  post <- left_join(w_devs, nta_stan,
    join_by = c(.draw, year)
  )
)

suppressMessages(
  post <- left_join(post, devs,
    join_by = .draw
  ) %>% arrange(.draw)
)
```

Note, the `supressMessages()` function is just telling the join commands to shut up and not print a bunch of garbage to the console while we run this chunk of code. I personally don't like it when my programs back sass me, and so I suppress their output here.

Next up we extract a bunch of leading parameters from the Bayesian model fit. These are terms that were read in as data, and thus these are not changing on each draw of the posterior. So, we'll just bring them in based on a single draw of the posterior:

```
# extract leading parameters from stan to get vbro
leading_pars <- fit %>%
  spread_draws(
    Lo_report[age],
    l_a_report[age],
    v_a_report[age],
    v_f_a_report[age],
    f_a_report[age],
    w_a_report[age],
    M_a_report[age]
  ) %>%
  filter(.draw %in% draw_idx[1]) # indexed on 1 bc not changing
leading_pars$age <- ages # correct ages
w_a <- leading_pars$w_a_report
f_a <- leading_pars$f_a_report

v_survey <- leading_pars$v_a_report
v_fish <- leading_pars$v_f_a_report
Lo <- leading_pars$Lo_report
M_a <- leading_pars$M_a_report
vbro <- sum(Lo * v_survey * w_a)
sbro <- sum(f_a * Lo)

# put some extra stuff in post
post$sbro <- sbro
post$Ro_map <- Ro_map
post$vbro <- vbro # biomass vulnerable per recruit *surveys*
post$vbo <- sum(Ro_map * Lo * v_fish * w_a) # biomass vulnerable to *fishing*
```

So basically we are just extracting a bunch of the stuff from our fit that we discussed in detail during the previous tutorial. If you don't remember, these terms are:

- `w_a` is a vector of relative weight at ages.
- `f_a` is a vector of relative fecundity at ages.
- `v_survey` is a vector describing Walleye vulnerability to the survey gear at each age.
- `v_fish` is a vector describing Walleye vulnerability to fishing at each age.
- `Lo` is survivorship at age in the unfished condition.
- `M_a` is instantaneous natural mortality age age (based on the Lorenzen equation).
- `vbro` is vulnerable biomass per recruit in the unfished condition.
- `sbro` is spawners per recruit in the unfished condition.

and we then put the stuff we need into the `post` tibble for use later on. Remember: this is just us reading in relevant information, parameters, data, whatever from the BERTA model fits and setting it up to use with our simulations below.

Now that once we have read in all of the stuff that we need to run the simulations, we are going to set up the simple linear policies for which we want to test performance in terms of how well they achieve the MAY and HARA objectives.

The way this works is that for each policy we want to test, and then for each draw of the posterior we want to simulate across, we'll run the model some number of years into the future. We'll then record the expected performance of our policies and pick the best policy overall for a specific objective. The linear policies we showed above have two parameters: *b<sub>lrp</sub>* and *c<sub>slope</sub>*, which correspond to the x-axis intercept (the fixed escapement point below which fishing is shut off) and the slope of the line (or the rate at which harvesting increases as stock size increases), respectively.

We'll plot a few of these in R to remind you what is going on:

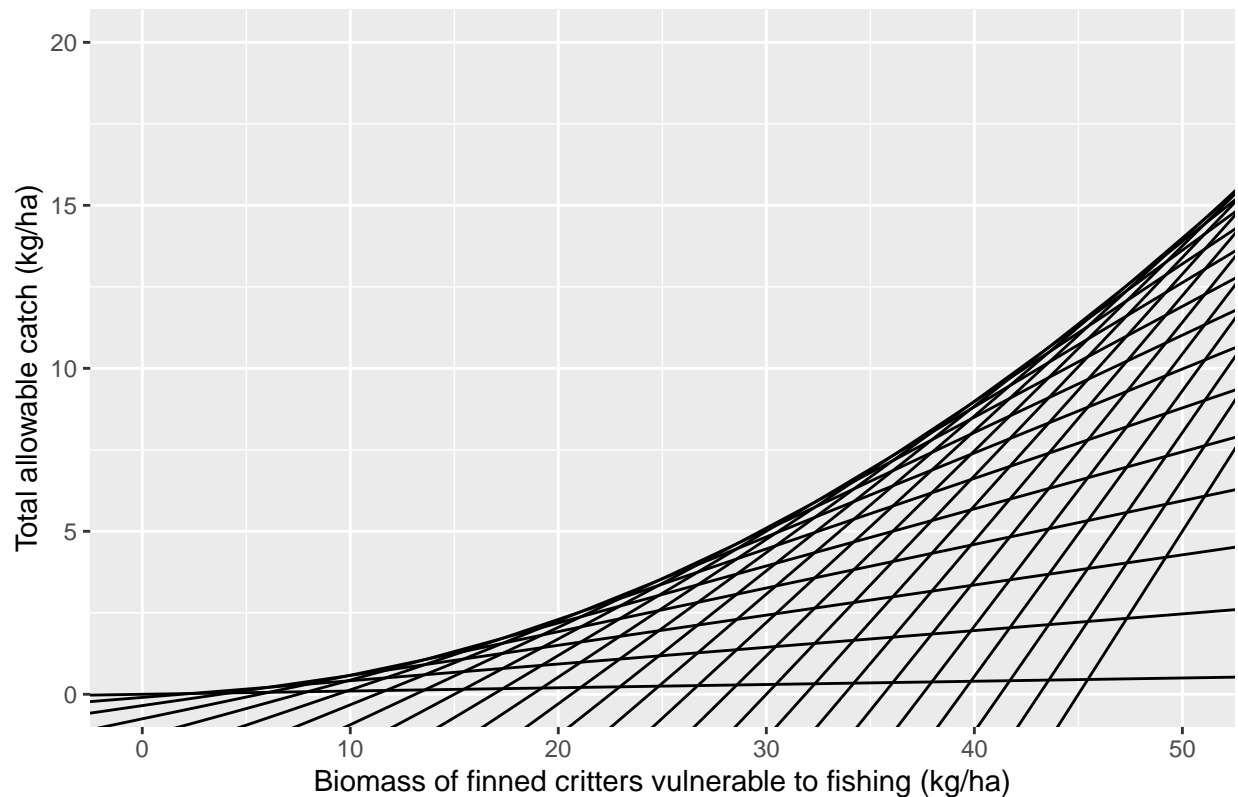
```
b_lrp <- seq(from = 0, to = 45, length.out = 25) # lower limit reference point
c_slopes <- seq(from=0.01, to = 1.0, length.out = length(b_lrp)) # slope of hcr

# solve c_slopes*(vb-b_lrp) for vb = 0
yints <- -b_lrp*c_slopes

my_lines <- data.frame(yints, b_lrp, c_slopes)
my_lines$policy <- as.factor(1:nrow(my_lines))
my_lines$vB <- seq(from = 0, to = 100, length.out = nrow(my_lines))
my_lines$TAC <- seq(from=0, to = 60, length.out = nrow(my_lines))

my_lines %>%
  ggplot(aes(x=vB, y = TAC))+
  geom_abline(mapping = aes(intercept = yints, slope = c_slopes,
                           group = policy)) +
  scale_x_continuous(limits = c(0, 50)) +
  scale_y_continuous(limits = c(0, 20)) +
  ylab("Total allowable catch (kg/ha)") +
  xlab("Biomass of finned critters vulnerable to fishing (kg/ha)") +
  ggtitle("Visualizing a few possible linear policies")
```

## Visualizing a few possible linear policies



Each of these lines has a specific lower limit reference point  $b_{lrp}$  and a slope to that particular line  $c_{slope}$ . What we need to do then, is test each of these policies against our simulation and see which one wins in terms of achieving our objective(s).

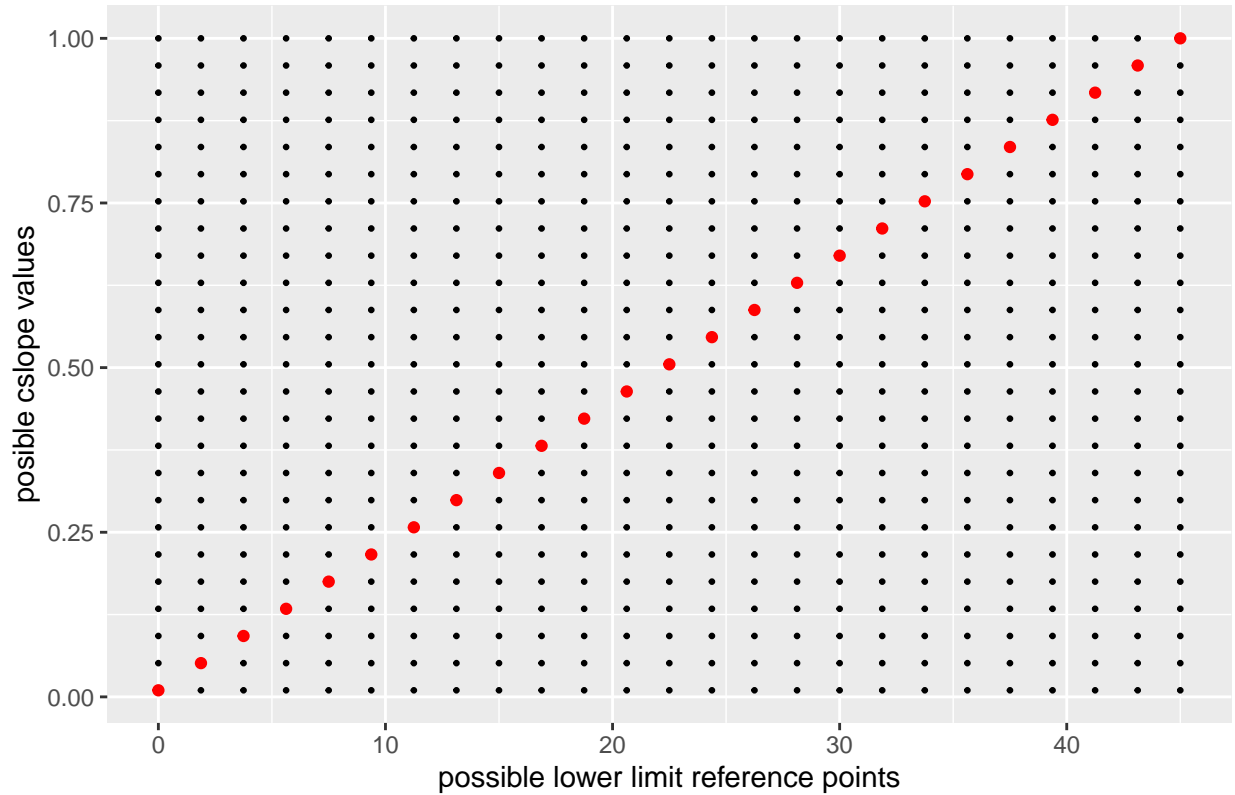
Note that this is not all possible linear policies. I just took a few and plotted them to give you a sense of what I am talking about—there are too many to plot on this single graph. What we really need to do is take a range of plausible  $b_{lrp}$  and  $c_{slope}$  values, and then test each possible combination of these values.

You can, however, visualize or think about the solution space of all possible linear harvesting policies in terms of a matrix:

```
# get all combinations of b_lrp, c_slope
my_grid <- expand.grid(b_lrp = b_lrp, c_slope = c_slopes)

my_grid %>%
  ggplot(aes(x = b_lrp, y = c_slope)) +
  geom_point(size=0.5) +
  geom_point(data = my_lines, aes(x = b_lrp, y = c_slopes),
            colour = "red") +
  xlab("possible lower limit reference points") +
  ylab("possible c_slope values") +
  ggtitle("The continuous space of all possible simple linear policies")
```

## The continuous space of all possible simple linear policies



*# red points identify the subset of policies from the previous line plot*  
*# black points are policies that weren't plotted on the previous line plot*

The key thing to recognize is that each possible combination of  $b_{lrp}$  and  $c_{slope}$  defines a single harvest policy (i.e., each point on the plot). We want to test the performance of each of these policies, so we need to figure out a range of policies to iterate across for each lake.

The next chunk of simulation code does declares a possible range of  $b_{lrp}$  and  $c_{slope}$  values:

```
bmin_max_value <- Ro_map * sum(Lo * v_fish * w_a)
```

Here, we are just setting the maximum  $b_{lrp}$  value at  $B_0$ , which is the biomass vulnerable to fishing in the unfished condition and is which calculated as the sum product of survivorship at age, fishing vulnerability at age, and relative weight at age:

$$B_0 = R_0 \cdot \sum_{age=2}^{age=20} L_{oa} \cdot v_{fa} \cdot w_a \quad (1)$$

It should make intuitive sense that we would say that the lower limit reference point shouldn't be higher than  $B_0$ , or at least I hope this is intuitive. The next chunk of code in this hog of a function is:

```
if (which_lake == "calling lake") {
  bmin_max_value <- 100
}
```



```

    } else {
      bmin_max_value <- ifelse(bmin_max_value > 75, 75, bmin_max_value)
    }
    c_slope_seq <- seq(from = 0.01, to = 1.0, length.out = round(grid_size / 2))
    bmin_seq_low <- seq(from = 0, to = 20, length.out = round(grid_size / 2))
    bmin_seq_high <- seq(from = 20.5, to = bmin_max_value,
                        length.out = grid_size - length(bmin_seq_low))
    bmin_seq <- c(bmin_seq_low, bmin_seq_high)

```

First, we are setting up a simple conditional statement to handle `bmin_max_value` for situations where it gets too high. This is ugly, and required different condition handling for calling lake vs. other systems for reasons we can talk about in our tutorial in person.

Next, we are setting up a sequence of `c_slope` values. Then we do something weird relative to what I have shown you. If you look closely, we are setting up two `$b_{lrp}` sequences, and then smushing them together. If you read code, you'll see that we are stacking half of the lower limit reference points for our policies in the range of 0-20, and half in the range 20.5-bmin\_max\_value.

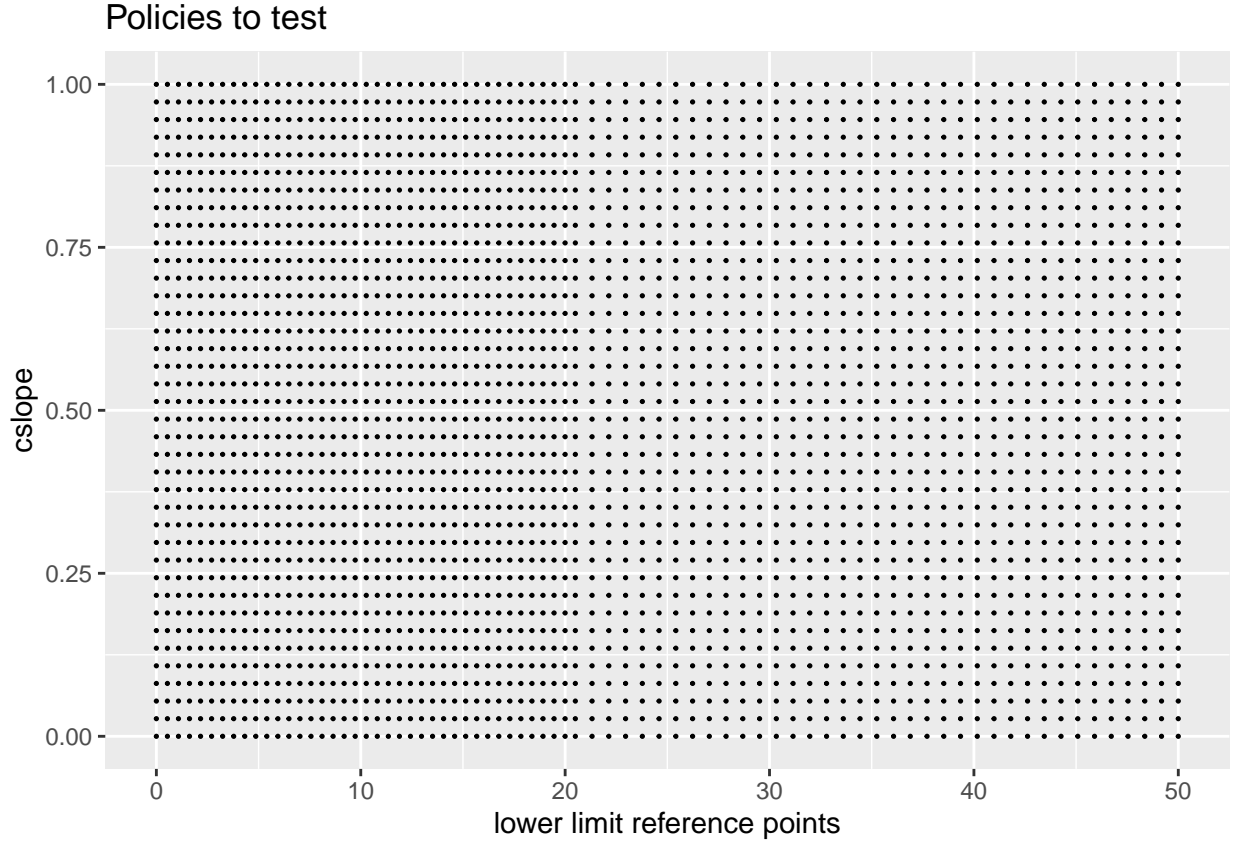
We can actually visualize what this is actually doing by picking some values for the `grid_size` and `bmin_max_value` parameters:

```

grid_size = 75
bmin_max_value <- 50 #let's just pick this to show what's happening
c_slope_seq <- seq(from = 0.0, to = 1.0, length.out = round(grid_size / 2))
bmin_seq_low <- seq(from = 0, to = 20, length.out = round(grid_size / 2))
bmin_seq_high <- seq(from = 20.5, to = bmin_max_value,
                    length.out = grid_size - length(bmin_seq_low))
bmin_seq <- c(bmin_seq_low, bmin_seq_high)
# get all combinations of blrp, cslope
my_grid <- expand.grid(b_lrp = bmin_seq, c_slope = c_slope_seq)

my_grid %>%
  ggplot(aes(x = b_lrp, y = c_slope)) +
  geom_point(size=0.25) +
  xlab("lower limit reference points") +
  ylab("cslope") +
  ggtitle("Policies to test")

```



And if you look closely you can see that this code is stacking more points (policies) into the lower portion of this plot. We did this for computational purposes. We wanted more resolution at low stock sizes, and wanted to maintain some sort of computational efficiency. Thus, we stacked more policies to test in the lower portion of that plot.

Remember, the specific linear policies for a lake depend on a given lake's estimated  $R_0$  (see above) and won't necessarily correspond to these points here.

So now we basically just need to loop through each of these policies, run our simulation (for some number of draws from the posterior, and some number of years into the future). We jump into this bit of the simulation script now.

```
tot_y <- tot_u <- prop_below <- TAC_zero <-
  matrix(0, nrow = length(c_slope_seq), ncol = length(bmin_seq))
yield_array <- vB_fish_array <-
  array(0, dim = c(length(c_slope_seq), length(bmin_seq), n_sim_yrs))
wt_seqs <- matrix(NA, nrow = n_sim_yrs, ncol = n_draws)
```

This is just us creating a bunch of empty matrices and a couple of 3d arrays that we will fill below in the loops of misery (below). If you are not used to arrays, you can think of a 3 dimensional array like a book where each page is a matrix with rows and columns.

We need 3d arrays for some of our model output. For example, saving output for MAY or HARA utility for each year in our simulation and for each  $c_{slope}$ ,  $b_{lrp}$  combination requires three dimensions.

## The loops of misery

Here's where the simulation actually starts to happen (finally). Everything up to this point was just reading in data or model output and setting it up to run our simulation. So now we just need to run our simulation over some time horizon for each policy (determined in this case by the  $c_{slope}$ ,  $b_{lrp}$  combinations), for each draw of the posterior (which was set by the  $n_{draws}$  parameter).

```
#-----  
# run retrospective simulation for each cslope, bmin, draw, and sim yr  
#-----  
for (i in seq_along(c_slope_seq)) {  
  c_slope <- c_slope_seq[i]  
  for (j in seq_along(bmin_seq)) {  
    b_lrp <- bmin_seq[j]  
    set.seed(24) # challenge each bmin, cslope combo with same set of rec seqs  
    wt_re_mat <- matrix(rnorm(n = n_sim_yrs * n_draws, mean = 0, sd = sd_wt),  
                        nrow = n_sim_yrs, ncol = n_draws) # generate random deviates  
    for (k in seq_len(n_draws)) {  
      # pick a single draw  
      sub_post <- subset(post, post$.draw == unique(post$.draw)[k])  
      # lots more code below...    }  
  }  
}
```

In words, this code is doing the following:

- For each  $c_{slope}$  i
- For each  $b_{lrp}$  j
- Set the random number generator seed (ensures all simulations are comparable)
- Generate a matrix of random deviates called `wt_re_mat`, with dimensions of the number of simulation years and the number of draws from the posterior
- For each draw of the posterior k
- Set up a bunch of stuff based on this specific draw to use in your simulation (not shown – code below)

So as we have been saying, we are going to run our simulation for each of these things.

The next bit of code takes a specific draw of the posterior, and uses it to set some more parameters and variables before we begin to loop through each year of our simulation, t:

```
# set leading parameters from sampled draw  
rec_a <- exp(sub_post$ln_ar[1])  
rec_b <- sub_post$br[1]  
Ro <- sub_post$Ro[1]  
sbo <- Ro * sbro  
vbo <- Ro * vbro
```

Here we are plucking out the recruitment  $\alpha$  and  $\beta$  parameters and the average number of recruits in the unfished condition  $R_0$  from one draw of the posterior. We then calculate two derived variables `sbo` and `vbo`, which correspond to the total spawner biomass in the unfished condition and the biomass vulnerable to fishing in the unfished condition, respectively. We'll need these for our age structured calculations below.

## Setting up the recruitment anomaly sequences

The next thirty lines of code are where a bunch of important stuff happens. In particular, we set the recruitment anomaly sequences in this chunk of code. Remember, we have  $w_t$  values for 1990-2015 because we subsetting the estimated  $w_t$ 's for these years. This means we have 26 years of estimated recruitment anomalies.

There are many ways we could proceed with this. We could, for example, simply repeat the  $w_t$  sequence some number of times into the future. In fact, Carl and I have done exactly this for pink shrimp fisheries in British Columbia, because we have longer time series of trustworthy recruitment anomaly estimates for these fisheries (e.g., 30+ years).

It was our opinion after examining the data and model outputs for Alberta walleye that it would be better to model recruitment

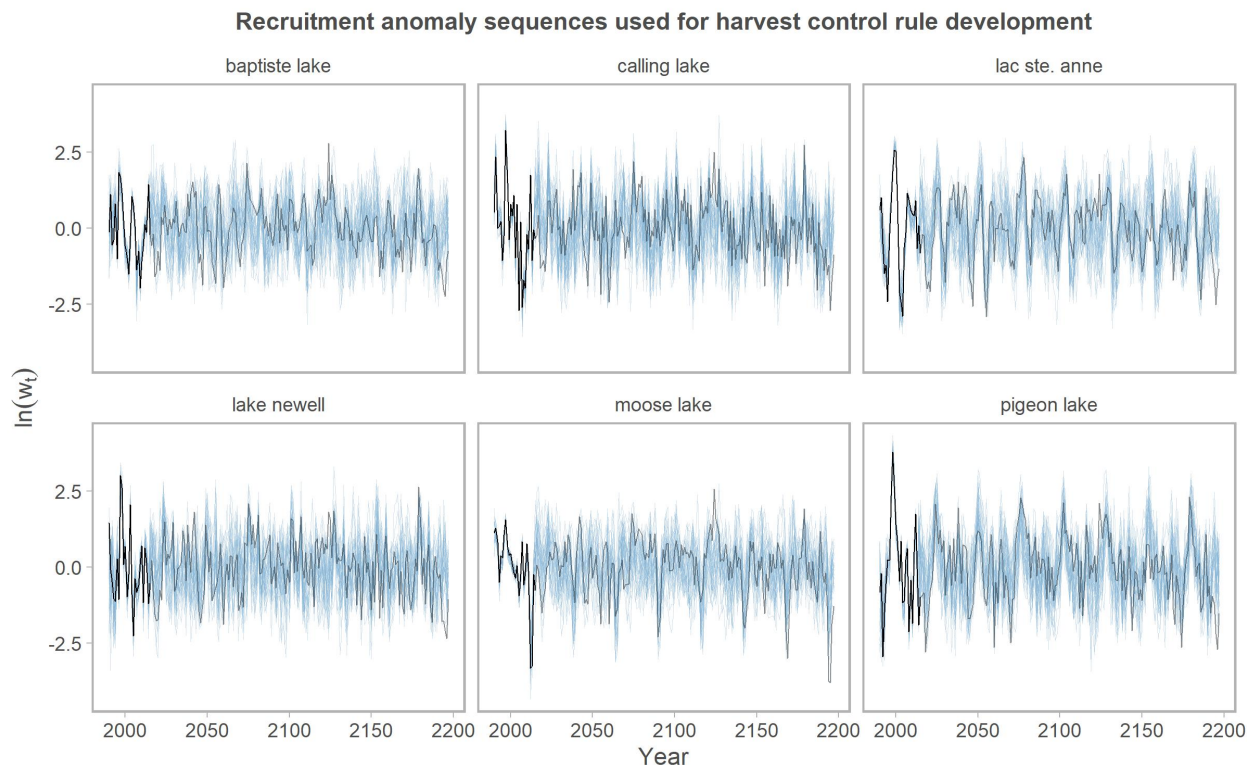


Figure 1: picture

## junk writing

A bewildering array of harvest control rule shapes and forms exists (see review in Deroba and Bence 20XX). These rules relate how much should be allocated for harvest relative to the biomass or number of critters vulnerable to harvesting. Here is an example of a simple harvest control rule:

Early optimization work on harvesting theory shows that optimal rules given simple models and single objectives typically decrease harvest rate as population size decreases and increase harvest rate as population size increases (citations). It doesn't have to be complicated. Many rules, such as the generic rule proposed in Fisheries and Oceans Canada (DFO) 20XX, suggest that rectilinear harvest control rules are "precautionary." However, what *specifically* is precautionary about such rules is rarely

rarely specify what is precautionary about the specific rule beyond references to Restrepo XXXX.

(and

Given these considerations, we will primarily focus on the simulation evaluation of simple linear harvest control rules. If it isn't broke, don't fix it sort of thing.

and repeat this process many times (e.g., 30-100 random draws). Each time we record the performance of a policy

each of some number (e.g., say 30 or 100) of random values from our assessment posteriors, run our model into the future.

## **Resources:**

Cahill et al. 2021 CJFAS.

Department of Fisheries and Oceans 2006. A harvest strategy compliant with the precautionary approach.

Deroba and Bence XXXX. Harvest control rules.

Edwards and Dankel 20XX. Fisheries Management Science.

Gelman et al. 2013. Bayesian Data Analysis.

Larkin. Epitaph on Maximum Sustainable Yield.

Moxnes XXXX.

Restrepo XXXX. Precautionary stuff.

Walters and Hilborn 1976. Adaptive control of fishing systems.

Walters and Hilborn 1978. Ecological optimization and adaptive management.

Hilborn and Walters 1992. Quantitative Fisheries Stock Assessment.

Walters 1986. Adaptive Management of Renewable Resources.

Walters and Martell. 2001. Fisheries Ecology and Management.