A Simulation Framework to Evaluate Harvest Control Rules for Alberta Walleye Fisheries

Christopher L. Cahill and Carl J. Walters

20 March 2022

Purpose

The purpose of this tutorial is to develop a management strategy evaluation or simulation framework that can be used to evaluate simple linear harvest control rules for any of the Alberta Walleye fisheries assessed in Cahill et al. (2021). We will use provisional objectives for all our analyses (discussed below), but any explicit objectives could be tested using the framework presented here.

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 is useful for managing natural resources.
- Understand how the `hcr_r()` code is pulling results from the stock assessments and using these as inputs into the 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 age-structured simulation model.
- Understand the tabular or grid search approach to finding linear harvest control rules that perform relatively well.
- Visualize some of the results.

Background

Harvest control rules are agreed-upon, transparent, and repeatable mathematical equations that help managers alter harvest in response to changes in stock size to achieve some explicit goal. Such changes in population size can be due to either environmental stochasticity or the effects of fishing. In Alberta, harvest control rules may be useful for informing Special Harvest License (SHL) harvest tag allocations given explicit management objectives and Fall Walleye Index Netting (FWIN) survey data. The SHL program requires that managers allocate some number of tags (a quota or Total Allowable Catch; TAC) based on their understanding of the risks to the fishery. Once tags have been allocated by regional managers, anglers are then able to enter lottery draw system to try and win a harvest tag, and fish can only be kept if anglers have an SHL tag in the SHL lakes. The development of harvesting rules for the SHL program via a management strategy simulation framework appears important because a recent study showed many Alberta Walleye populations were under-harvested relative to common fisheries reference points (see Cahill et al. 2021).

In general, 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 we are actually designing a control rule (a “controller” or “policy”) to get something we desire from a nonlinear dynamical system (e.g., a fishery).

There exists a rich history and literature of developing harvest control rules or feedback policies to regulate human exploitation in managed systems, and as a result there are several references that will help us better understand some of the issues we will encounter in this tutorial. A few of the references that I have found helpful 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

I recommend reading through these resources if you have the time to do so.

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 airborne. In this example, our goal 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 turn the plane to the right if the wind pushed it to the left, and try to orient upward if the plane were decreasing in altitude for whatever reason. In a harvested population, the goal might be to let the stock size recover if it has been overharvested, or perhaps to harvest more if the stock size was believed 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 harvesting policies generally decrease harvest rate when population size declines, and increase it as population size increases (e.g., Walters 1975; Walters and Hilborn 1978). These findings have often come from dynamic programming solutions, and the solutions to these particular problems often indicate that we should modulate harvest as a simple function of harvestable abundance or biomass (see Mangel 1984; Clark 1974; Clark and De Pree 1979; Moxnes 2002). This finding appears general and is surprising–it is often believed that complex policies are needed to manage complex ecological systems. However, much of the work on harvesting theory during the 1970-80s showed that simple policies performed well or were even optimal in many situations (e.g., Walters 1986; Hilborn and Walters 1992).

A limitation of dynamic programming and many of the methods used to make inferences about optimal harvesting policies (citations above) is that that these approaches can only be applied to relatively simple models and management objectives. In our case, the age-structured Bayesian Estimation of Recruitment Trends in Alberta (BERTA) model is too complex to use with such methods. Thus, we will instead use a simulation technique known as approximation in policy space to find good harvest control rules (see methods in Moxnes 2002; Edwards and Dankel 2016; Bertsekas 2019). 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 there are 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 (e.g., see Conroy and Peterson 2013).

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 2016). It is unlikely that Cahill and Walters (or any single 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 1977), 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 so as to explicitly consider stakeholder or harvester 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:

$$utility_t = catch_t^{pp} \tag{1}$$

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., $1/2 \cdot 100{,}000$), but most people will choose a guaranteed income of around 30,000 dollars vs. the 50:50 chance at winning the 100,000 dollars.

This scenario above 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

$$pp = \frac{ln(0.5)}{(ln(20) - ln(100))} = 0.43 \tag{2}$$

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. 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 should make 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 that exploits many populations or resources 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 recruitment variability (see Cahill et al. 2021).

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. Again, I want to drive home the point that 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

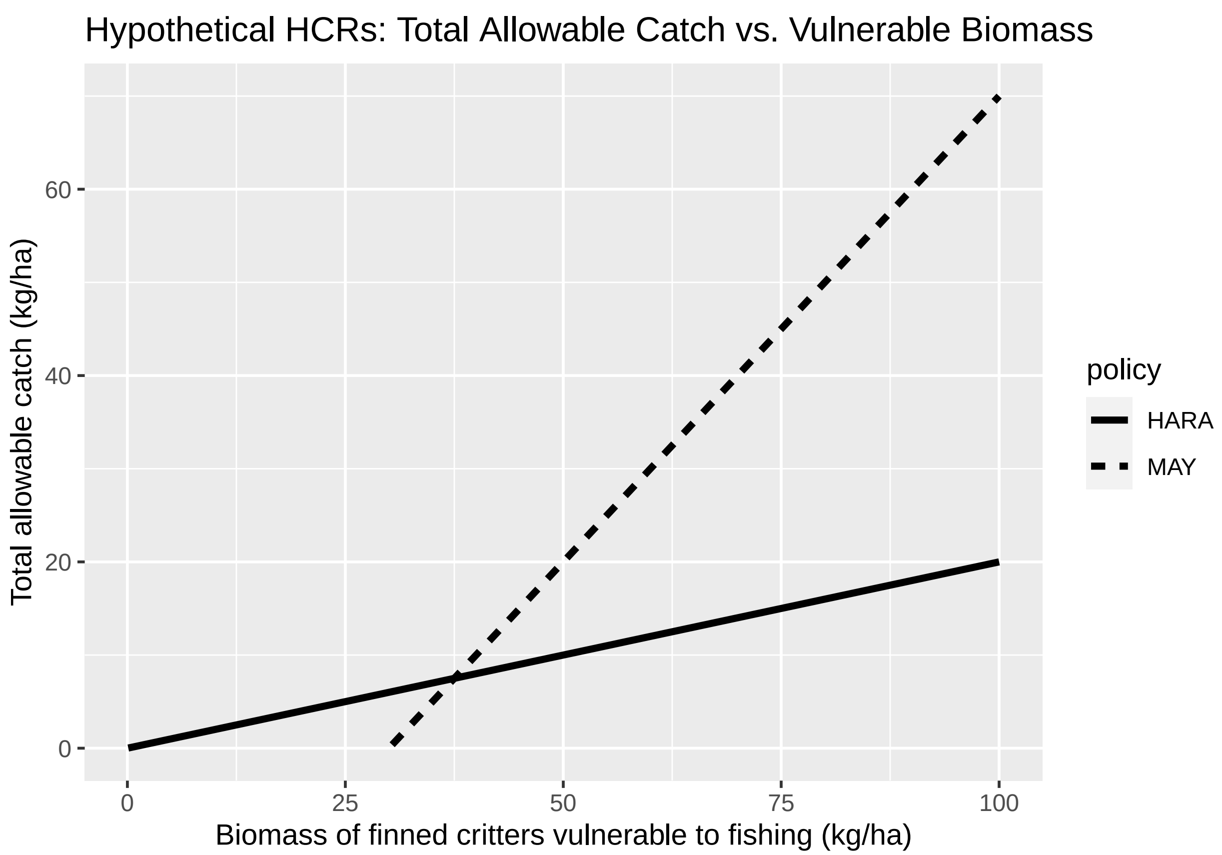
If you are a visual learner, you might have an easier time with the following hypothetical depiction of fixed escapement MAY vs. a HARA policy in R.

Let's load some packages:

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

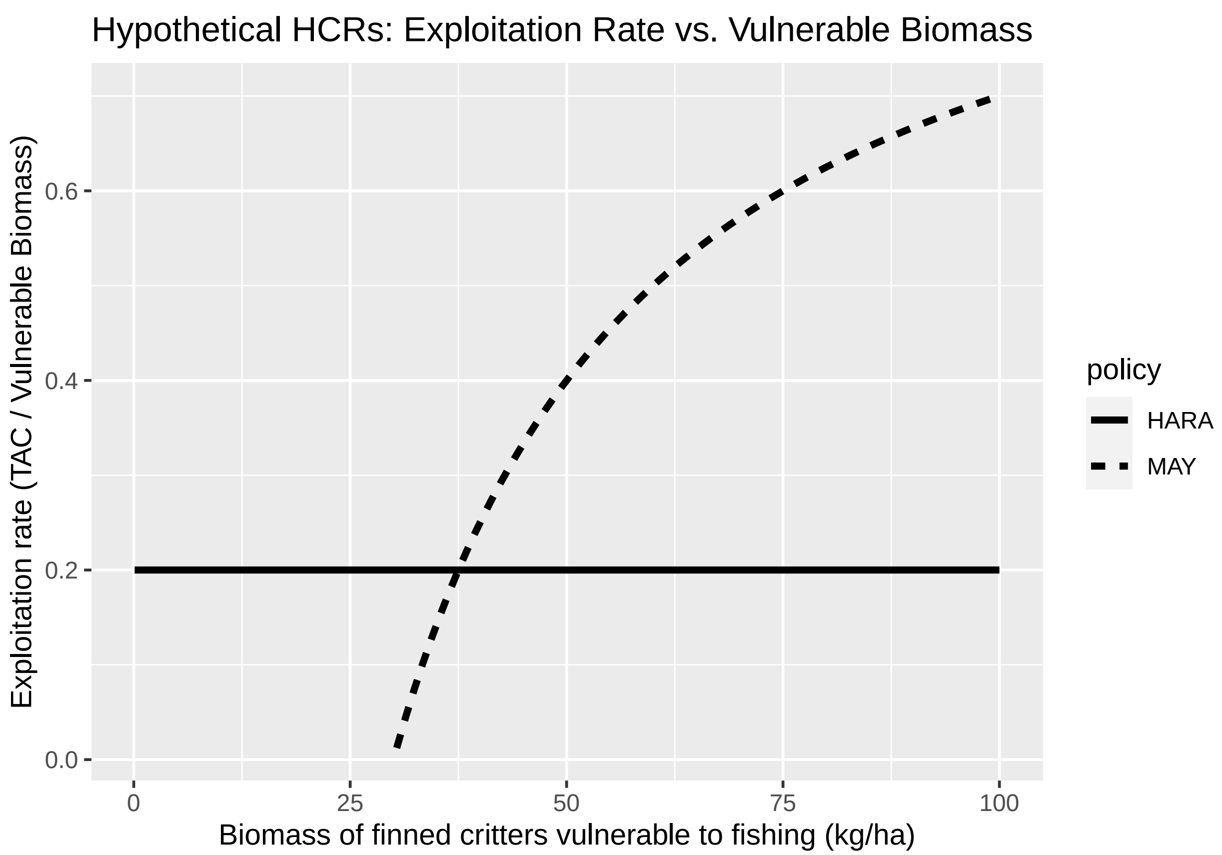
And then simulate and plot some simple harvest control rules (I made these up):

```r
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:

```r
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 rules that we will go through with our simulation simply shift the intercepts and slopes of these lines in Figure 1, and then test the performance of those lines or harvest control rule policies.

Thinking through one more harvest control rule

While the polices above arise from explicit objectives and dynamic programming solutions (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 will encounter in fisheries. This is DFO's so-called precautionary harvest control rule (DFO 2006), which appears to be loosely based on work from Restrepo and Powers (1999). 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 specifically is precautionary about this rule we will simulate it anyway to show you what it is doing.

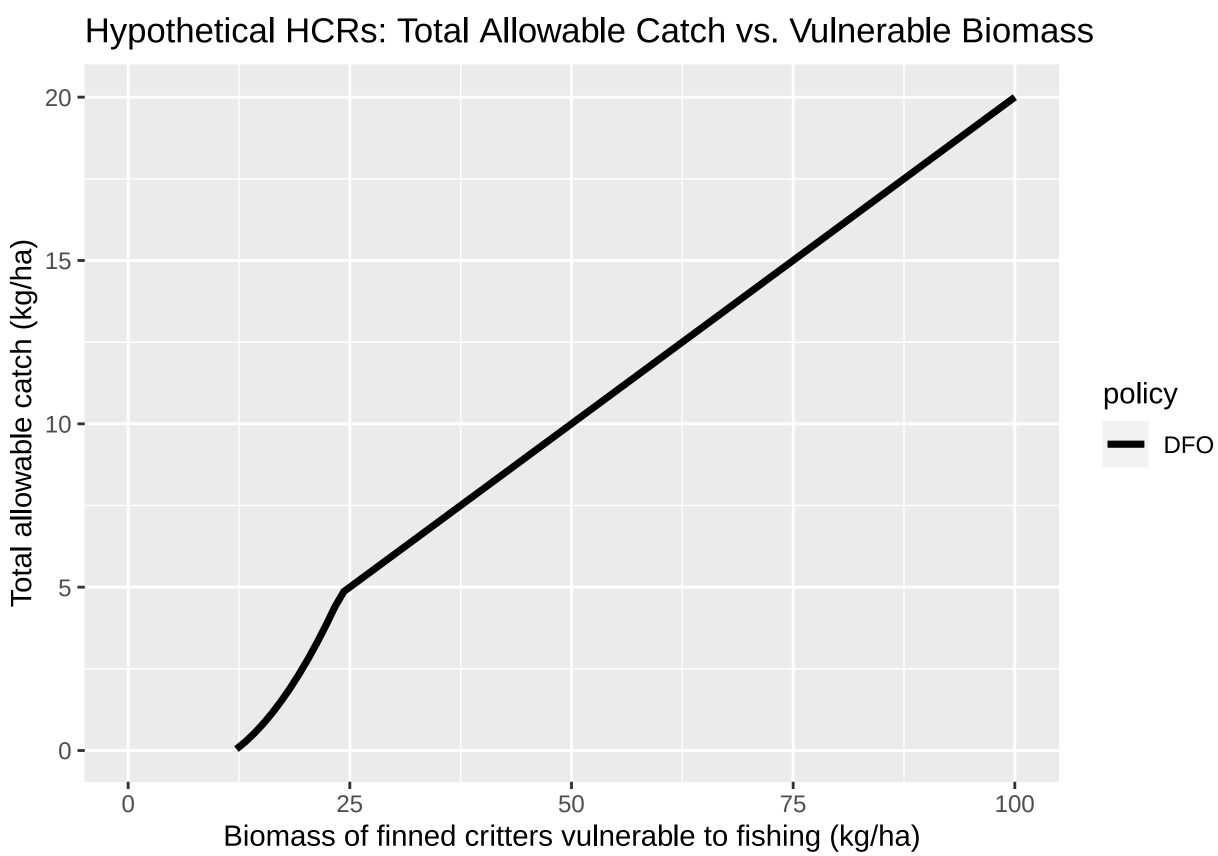
The DFO policy increases exploitation rate from zero to U_{MSY} as biomass increases from a lower limit reference point to an upper limit reference point. These reference points are typically set at $0.4 \cdot B_{MSY}$ and $0.8 \cdot B_{MSY}$. Note that U_{MSY} and B_{MSY} typically need to be estimated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

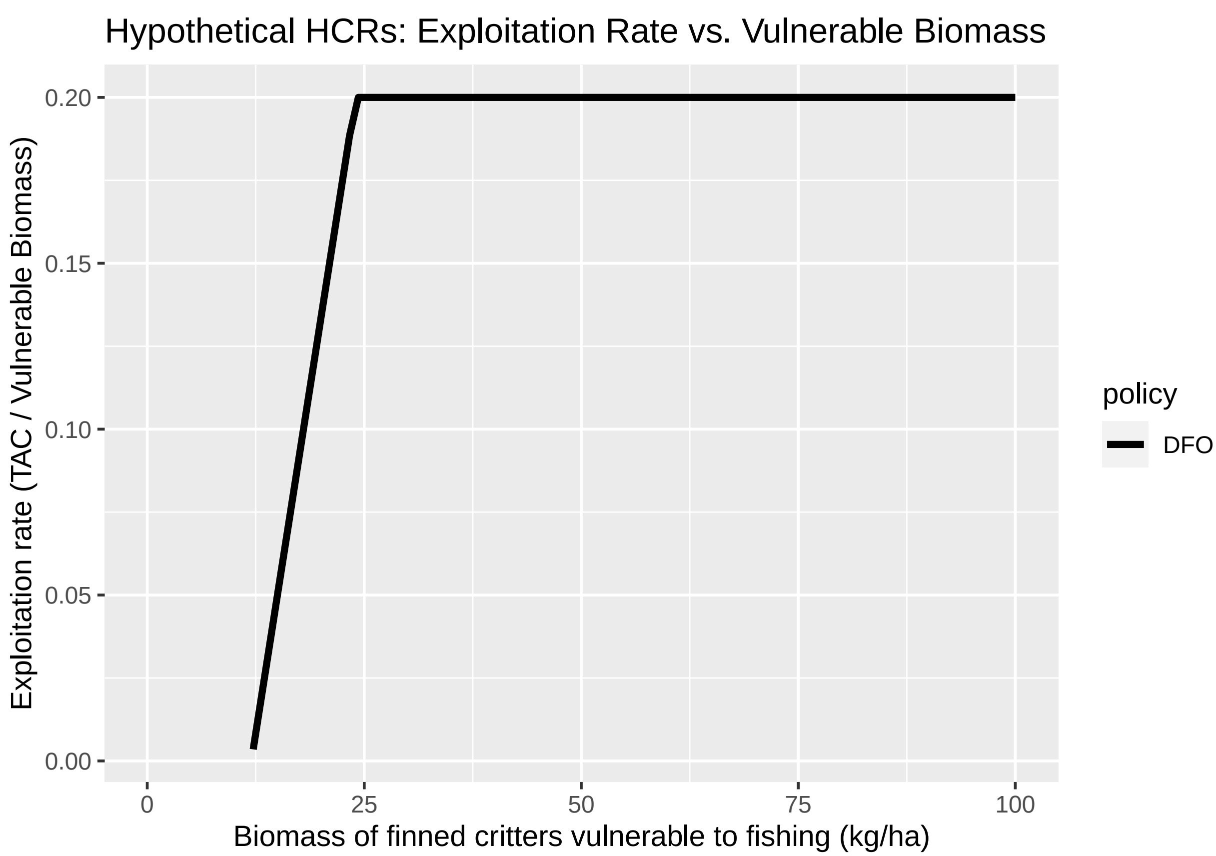
We can then plot it like we did before:

```
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 is 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?

We will not talk about the rectilinear harvest control rule any more in this tutorial, but it can be run using the simulation script. This is a thing that we should talk about in person and I can show you how to do it quickly. As we are going to ignore this rectilinear harvest control rule for the remainder of the tutorial, we'll just ignore the bits of code that only pertain to this rule. These are any bits of code that are wrapped in if-statements with `rule == rectilinear`. FYI.

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 also 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 saved the posteriors from these model fits for each lake with 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. 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. 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 the simulation and number of draws. 3. Once we are done, look back across all policies we tested, and look up the policy that maximized average yield or HARA utility, respectively. 4. Go for a long run and drink tasty beer, never think about designing controllers for nonlinear dynamical systems again.

This will become 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 of MAY and HARA, it is straightforward and trivial exercise 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 will ignore the rectilinear rule entirely and just focus on the simple linear rules. As stated above, we can discuss running the simulation routine for rectilinear harvest control rules in person if folks want.

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 any 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 (simulation control parameters and our Bayesian model fits in list form) that `get_hcr()` 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 do 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 and play around with easily. The arguments that you can play around with are:

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

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`, which is declared outside the function `hcr_pars()` 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 will notice that I have not put these variables 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, which I needed to do a lot while Carl and I were developing these simulation programs. This was the best way I could think to do that, though I am sure there are other (and perhaps more elegant) solutions.

I don't want you to worry too much about these values in `hcr_pars` 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 takes the `which_lake` string, and uses it to grab the appropriate Bayesian model fit from our R environment:

```
#-------------------------------------------------------------------
# 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 declare a control variable called `rec_ctl`. This `rec_ctl` term indicates whether the Bayesian model was fitted with a Beverton-Holt (i.e., "bh") or Ricker stock-recruit model (i.e., "ricker").

The next ten or so lines are used for the rectilinear harvest control rule, and so we will ignore this. Note this code is only triggered when rule = "rectilinear", and you can look at it here if you want:

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

The next 80 or so lines are pretty annoying–I am referring to lines 64-144 in the script. All these lines of code are doing is extracting relevant values from our Bayesian model fits and setting them up so that we can use them in our simulation. Some of the stuff we need for our simulation was read into our Bayesian model as data, while other stuff was 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 a function called `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 to grabbing three parameters in this case–R_0, $ln(ar)$, and br. These values are then 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.

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 might 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.

We are also sub-setting 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 conservative 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 for our age-structured simulations. We'll discuss explicitly how we set up these recruitment anomaly sequences in a later section.

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

```
# 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.

We then extract some leading parameters from the Bayesian model fit. These are terms that were read in as data, and thus these did not changing on each draw of the posterior. Because they don't vary per draw, we can 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*
```

And so we basically we are just extracting 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 vulnerability to the survey gear at each age.
- `v_fish` is a vector describing vulnerability to fishing at each age.
- `Lo` is survivorship at age in the unfished condition.
- `M_a` is instantaneous natural morality 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.

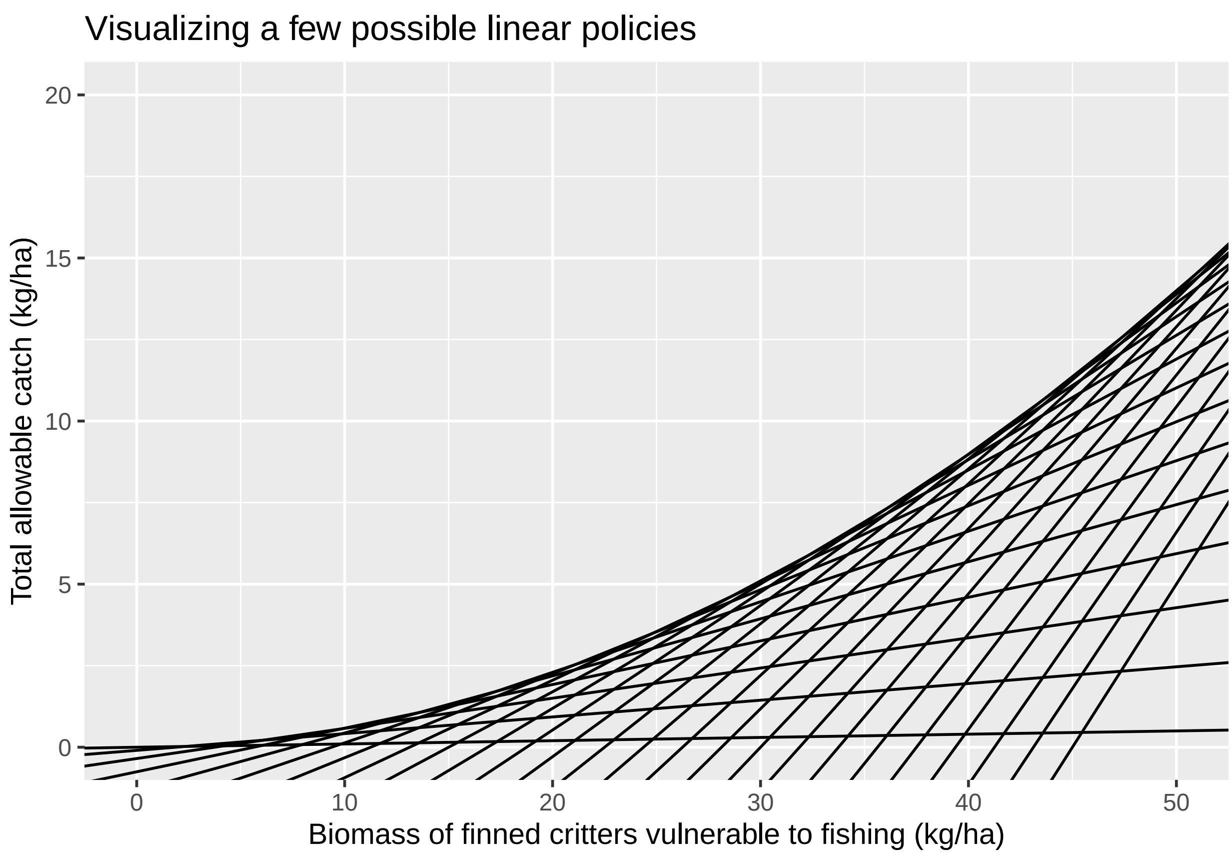
and we then put the stuff we need into a dataframe called `post` for use later on. Remember: this is just us reading in relevant information, parameters, and data, from the BERTA model fits and setting it up to use with our simulations.

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 harvest policies for which we want to run our simulation for.

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 age-structured model some number of years into the future. The simple linear policies described above have two parameters: b_{lrp} and $cslope$, 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.

Let's look at a few simple linear policies in R to remind you of 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")
```

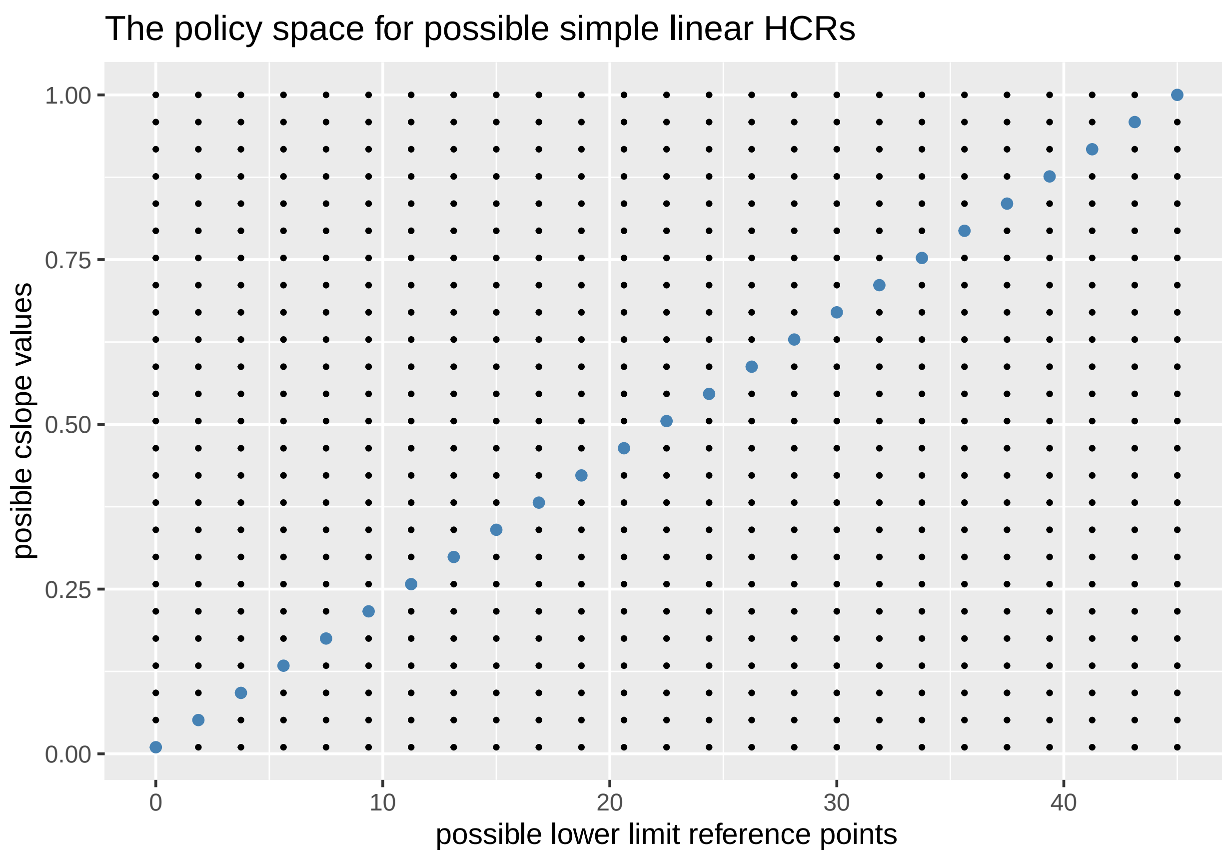


Each of these lines has a specific lower limit reference point b_{lrp} and a slope to that particular line c_{slope}. What we need to do then, is test these policies (and additional linear policies–not shown) against our simulation and see which one wins in terms of achieving our objective(s).

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.

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

```
# get all combinations of blrp, cslope
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 = "steelblue") +
  xlab("possible lower limit reference points") +
  ylab("posible cslope values") +
  ggtitle("The policy space for possible simple linear HCRs")
```




```
# blue points identify the subset of policies from the previous line plot
# black points are additional policies that weren't plotted on the previous 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 few chunks of simulation code sets up the linear policies to test by declaring 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 unfished recruitment R_0 times the sum-product of the survivorship at age, fishing vulnerability at age, and relative weight at age vectors:

$$B_0 = R_0 \cdot \sum_{age=2}^{age=20} L_{o_a} \cdot v_{f_a} \cdot w_a \tag{3}$$

It should make sense that we would say that the lower limit reference point shouldn't be higher than B_0. The next chunk of code in this beast of a function is:

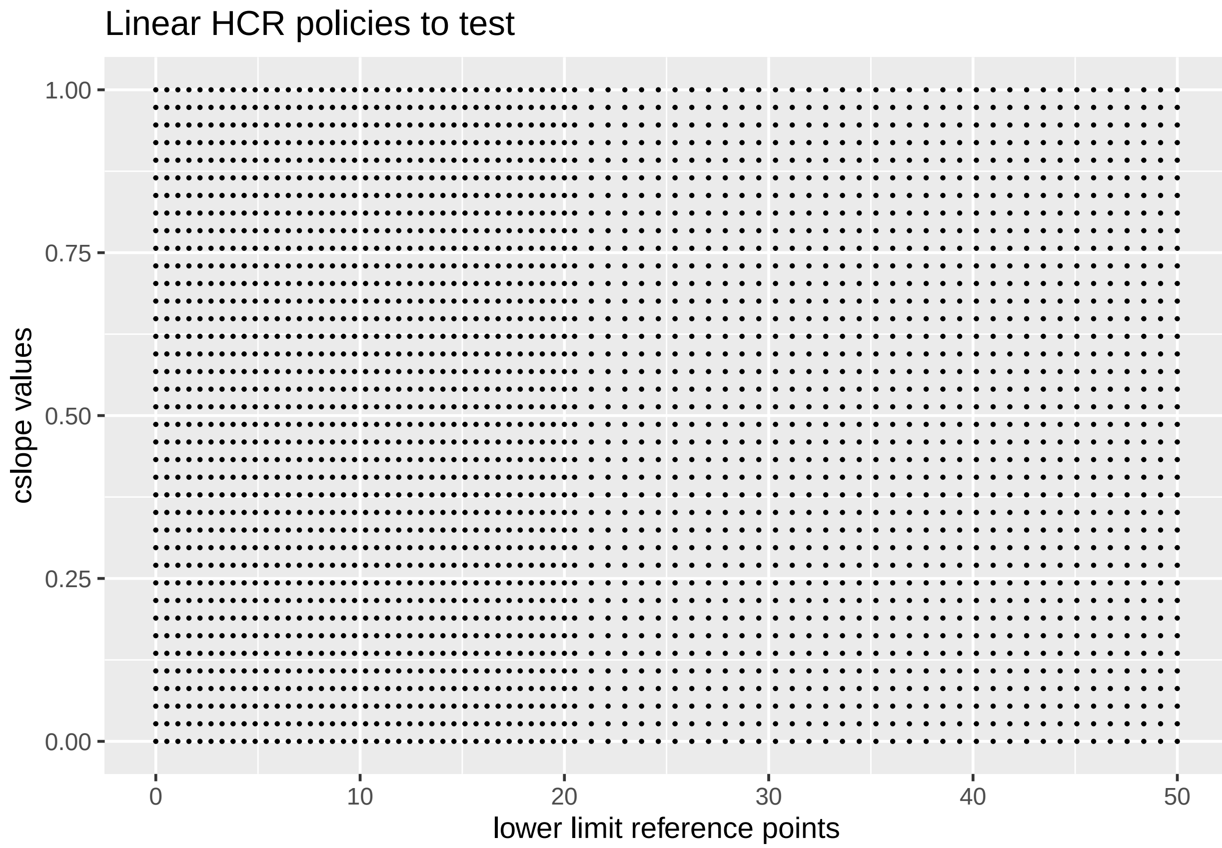
```r
if (which_lake == "calling lake") {
    bmin_max_value <- 100
    } else {
    bmin_max_value <- ifelse(bmin_max_value > 75, 75, bmin_max_value)
}
c_slope_seq <- seq(from = 0.01, to = 1.0, length.out = round(grid_size / 2))
bmin_seq_low <- seq(from = 0, to = 20, length.out = round(grid_size / 2))
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 just smushing them together.

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

```r
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 values") +
  ggtitle("Linear HCR policies to test")
```



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, and because we thought more of the 'action' would be happening at these lower vulnerable biomass values. We wanted more resolution at low stock sizes for the harvest control rule simulation, but we needed to maintain some sort of computational efficiency. Thus, we chose more policies to test in the lower portion of the above 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. This plot is just giving you a general idea of what that code is doing.

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)
```

First we have to set up 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 represents a matrix with rows and columns. We needed 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

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

The next chunk of code is:

```
#----------------------------------------------------------------------
# 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, pick one c_{slope}.
- For each b_{lrp} j, pick one b_{lrp}.
- 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 (we will use this below to generate our recruitment anomaly sequences).
- For each draw of the posterior k, subset the posterior for that specific draw.
- Set up a bunch of stuff based on this specific draw to use in your simulation (not shown – code below).

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 α and β 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 (in words)

The next thirty lines of code are where a bunch of critical stuff happens. In particular, this is where we set the unpredictable recruitment anomaly sequences that we test our policies against. Recall that we have w_t values for 1990-2015. This means we have 26 years of estimated recruitment anomalies.

Given these estimates, there are many ways one could proceed. We could, for example, simply repeat the estimated w_t sequence some number of times into the future. In fact, we did this for a pink shrimp problem we were working on over the fall. For the pink shrimp problem, we had longer time series of trustworthy recruitment estimates (e.g., 40 + years) and the shrimp only live for about four years vs. walleye which may live to 20+ in Alberta. For this pink shrimp management strategy simulation tool, we just put the recruitment anomaly sequences end-to-end-to-end many times, and then tested performance of the various policies we were interested in.

At first we tried to do this with Alberta walleye. We repeated the recruitment series that we estimated with the Bayesian models, but it turned out this didn't work well because there were not enough recruitment estimates to make the harvest control rule simulation reliable. Thus, we had to figure out a workaround for this issue.

While we could not simply repeat the historical sequences for the walleye problem, it seemed there was still useful information in the historical w_t sequences that could be exploited to inform our policy simulation. For example the historical 26 year estimated w_t sequence provides at least some information on the relative frequency of strong and weak year classes, year class magnitude, and on the autocorrelation structure in recruitment for a given lake.

Because we only have 26 years of recruitment estimates for a fish that lives 15-20 years, about the only thing we are sure of is that the next 26 years will almost certainly not be an exact repeat of the last 26 years. Given these ideas, we came up with the following rules to generate plausible future recruitment patterns for Alberta walleye:

1. We start by generating a repeating historical sequence of w_t as $w_{t_{historical}}$, exactly how we did for shrimp problem above. So we just repeat the estimated w_t values `n_repeats` or 8 times x 26 years of reliable recruitment estimates = 208 total simulation years.
2. Next, we create an entirely new auto-correlated and noisy series of simulated recruitment anomalies $w_{t_{sim}}$ where the simulation parameters for this sequence are assumed similar among lakes.
3. For the first 26 years of simulation, set w_t to the first 26 estimated values in $w_{t_{historical}}$.
4. For year 27-208, calculate the remaining w_t values for the simulation as

$$w_t = \psi_{w_t} \cdot (1 - w_{t_{historical}}) \cdot w_{t_{sim}}, \text{ where } \psi_{w_t} = 0.5 \tag{4}$$

You need to be careful here–these w_t values in point 4 are now different than the originally estimated values from our Bayesian models (the w_t values referenced in point 1).

These rules resulted in qualitatively similar patterns to the estimated recruitment anomalies for Alberta walleye. All this jiggery-pokery does is set the first 26 years to the estimated w_t values from the assessment models, and then sets the remaining w_t values (i.e., years 27-208) as a weighted average of the repeated historical sequence and our simulated sequence $w_{t_{sim}}$. The weighting between the estimated and simulated recruitment anomalies is given by the parameter ψ_{w_t}.

We set the $w_{t_{sim}}$ values as follows:

$$w_{t_{sim}} \sim \text{Normal}(0, \sigma_{wt}) \text{ if t = 1} \tag{5}$$

$$w_{t_{sim}} = \rho \cdot w_{t-1_{sim}} + wt_{re}, \text{ where } wt_{re} \sim \text{Normal}(0, \sigma_{wt}) \text{ if t > 1} \tag{6}$$

Recall that t = 1 corresponds to year 1990 in our estimation model. Again, this recipe with $w_{t_{historical}}$ and $w_{t_{sim}}$ gave us recruitment anomaly sequences that were qualitatively similar to the values we estimated (we will plot some of them below to show you in a bit).

Setting up the recruitment anomaly sequences (for real this time–in code)

First we extract the estimated w_t values from the posterior and set up $w_{t_{historical}}$:

```
# repeat the historical recruitment series
wt_historical <- sub_post$w
wt_bar <- mean(wt_historical)
wt_historical <- wt_historical - wt_bar # correct nonzero wt over initialization period
wt_historical <- rep(wt_historical, n_repeats) # 8 x 26 year sequence of values

# set wt = BERTA estimated values for yrs 1-26
wt[1:n_historical_yrs] <- wt_historical[1:n_historical_yrs]
```

We are mean-centering the historical recruitment series by calculating and then subtracting off `wt_bar` from each historical w_t sequence during 1990-2015. We do this so that we do not do anything silly like assume recruitment was trending upward or downward in the initialization period. This avoids a hidden or implicit assumption that productivity was shifting through time (i.e., we need to make sure that when we select our 1990-2015 w_t values that they were not larger or smaller on average than our whole model fitting period (from 1980-2018).

Once we correct the `wt_historical` sequence by subtracting `wt_bar`, we repeat the sequence `n_repeats` times. Note that we are limited to the historical 26 year reference period for our management strategy simulations. In all our simulations we set this at 8 repeats, and so simulations are 8 x 26 = 208 years in length. A simple rule of thumb here is to run the simulation at least ten times the generation length of the fish you are working on, which for Walleye around 200+ years. This preserves the historical frequency and relative strength of strong and weak year classes that we estimated in the assessment model.

Also, the reason we simulate this long into the future is to ensure the initial condition or state that we start the simulation at does not influence the best harvest control rule choice. Phrased differently, we are trying to find a stationary and approximately stationary harvest control rule that works reasonably well regardless of the initial conditions.

Up to now we have specified the w_t sequences identically to how we did for the shrimp problem, but here is where things change. Next we set up the autoregressive simulated $w_{t_{sim}}$ deviates:

```
# set the random effect vector
wt_re <- wt_re_mat[, k]

# generate auto-correlated w(t)'s
wt_sim <- wt <- rep(NA, length(wt_historical))
wt_sim[1] <- wt_re[1] # initialize the process for t = 1

# create autoregressive wt_sim[t]
for (t in seq_len(n_sim_yrs)[-n_sim_yrs]) { # t = 1 to 207
  wt_sim[t + 1] <- rho * wt_sim[t] + wt_re[t + 1]
}
```

Recall here that we filled the whole `wt_re_mat` in line 179 above, and we did it this way for two reasons. First, we wanted to use the same deviates (we need to test the harvest control rules against the same unpredictable recruitment sequences). The other reason we did this is because it is faster to do it this way.

We first initialize $w_{t=1_{sim}}$ using the first value of `wt_re`, and then we use a for loop to set up the rest of our autoregressive $w_{t_{sim}}$ deviates. Fun stuff.

We then calculate the new w_t values that we want to use for this specific draw of the posterior–i.e., these are the recruitment anomaly values we will use for one run of our simulation.

```r
# calculate wt differently for yrs 26 +
for (t in n_historical_yrs:n_sim_yrs) { # t = 26 to 208
  wt[t] <- psi_wt * wt_historical[t] + (1 - psi_wt) * wt_sim[t]
}
```

And so this code represents the technical implementation of equation 4 above. You can work through the math on your own sometime and play around with the parameters, but the key point is this helps us generate reasonable recruitment time-series into the future. These distributions appear to have statistical properties similar to the original estimated recruitment anomaly values For example, if we randomly select 30 draws of the recruitment estimates from Baptiste lake, and then simulate future recruitment using these rules and steps, we get the following plot:

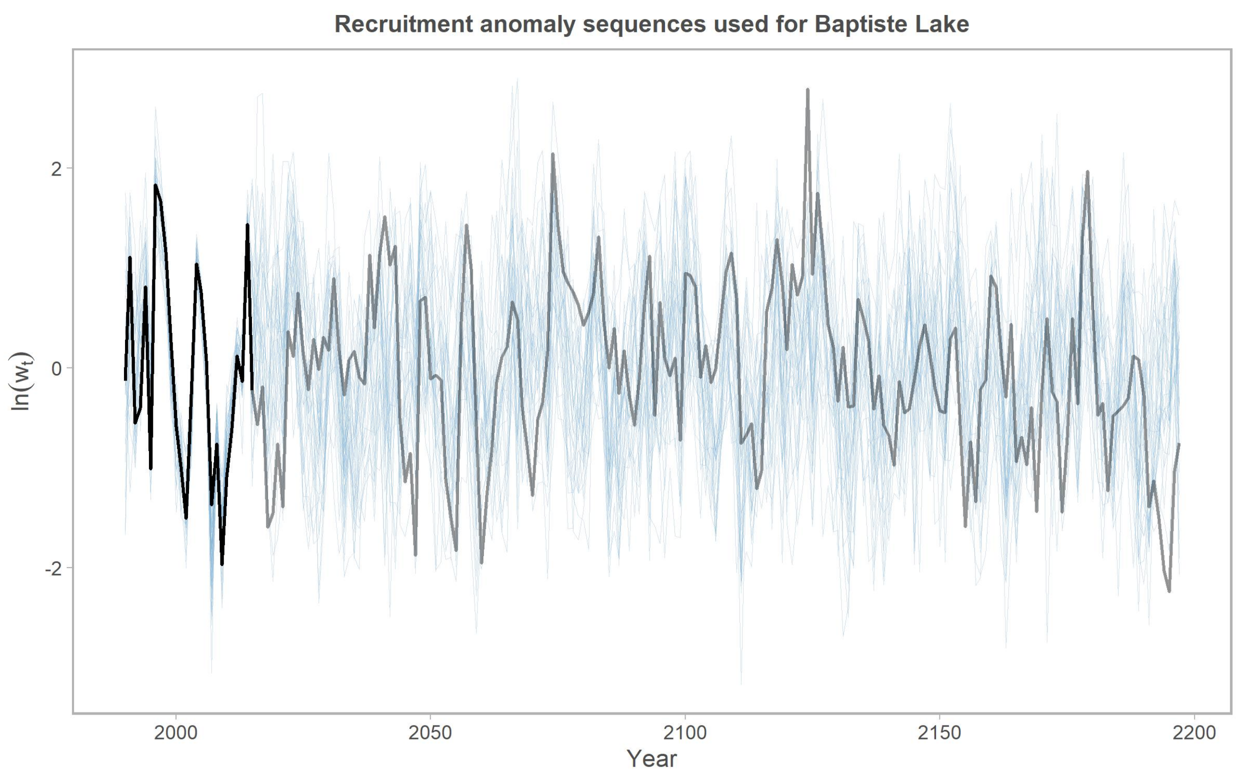


Figure 1: Estimated and simulated recruitment anomalies w_t that we use for harvest control rule development. Thick black line represents one estimated recruitment anomaly sequence from 1990-2015, while the light grey line shows the simulated recruitment sequence into the future using the recruitment deviation rules. Note the simulated values are modeled as a weighted average of the estimated sequence during 1990-2015 that is repeated through time and an autocorrelated stochastic process. Light blue lines show different recruitment anomaly trajectories.

Now that we have set up our recruitment anomaly sequences, we have even more bookkeeping to take care of:

```r
  # extract the initial age structure
  Ninit <- sub_post[
    which(sub_post$year == retro_initial_yr),
    which(colnames(sub_post) %in% ages)
  ] %>%
    slice() %>%
```

```r
    unlist(., use.names = FALSE)

  Rinit_yr_2 <- sub_post[
    which(sub_post$year == retro_initial_yr + 1),
    which(colnames(sub_post) == 2)
  ] %>%
    slice() %>%
    unlist(., use.names = FALSE)

  # nta matrix
  nta <- matrix(NA, nrow = length(wt) + 2, ncol = length(ages)) # recruit @ age 2

  # SSB, Rpred, vulnerable biomass vectors
  SSB <- Rpred <- vB_fish <- vB_survey <- rep(0, length(wt))
  nta[1, ] <- Ninit # initialize from posterior for retro_initial_yr
  nta[2, 1] <- Rinit_yr_2
  t_last_ass <- 1 - ass_int # when was last survey / assessment (initialize)
```

In words, this is extracting the initial numbers at age from the posterior in the year corresponding to 1990, and then saving it as a vector named `Ninit`. Next, we do the same thing for age-2 recruits in 1991, and call this `Rinit_yr_2`. We do this because we actually have to initialize our recruitment models for two years because we are assuming critters recruit at age two.

Next up, we make a numbers at age matrix called `nta`, with dimensions 208 (years or rows) by 19 (ages 2-20, columns. We then declare some vectors `SSB`, `Rpred`, `vB_fish`, and `vB_survey`, and fill in or initialize the first couple values of `nta`. Lastly, we declare a term called `t_last_ass`, which is keeping track of when the last assessment occurred in a given fishery. We will use this to figure out how often to assess a walleye fishery with some clever if-statements in a bit.

Loops of misery, continued

Now we will begin simulating our age-structured population dynamics model into the future. Finally. Everything up to now was setting up the estimates from our Bayesian model, organizing and figuring out how to deal with the recruitment anomalies, and a bunch of bookkeeping hogwash just so that we can actually run our simulation models.

Here we finally begin to loop across years t:

```r
 # run age-structured model for sim years
for (t in seq_len(n_sim_yrs)[-n_sim_yrs]) { # years 1 to (n_sim_year-1)
  SSB[t] <- sum(nta[t, ] * f_a * w_a)
  vB_fish[t] <- sum(nta[t, ] * v_fish * w_a * ret_a) # without ret_a, overestimate yield
  vB_survey[t] <- sum(nta[t, ] * v_survey * w_a)
  # more code below...
```

These lines of code are looping through t = 1 to year 207 of the simulation, and then setting the values of spawning stock biomass `SSB`, vulnerable biomass to fishing `vB_fish`, and vulnerable biomass to surveys `vB_survey`. The math for these terms can be found in the previous tutorial if you would like to see it. We will talk about the `ret_a` term below, when we discuss the catch and release mortality component. For now you can just ignore it.

The next if statement determines whether an assessment is occurring in that year, which depends on the simulation parameter `ass_int`. Let's assume now this value is set at `ass_int` = 1, so surveys occur every year:

```r
if (t - t_last_ass == ass_int) { # assess every ass_int yrs
    t_last_ass <- t
    # note -0.5*(0.1)^2 corrects exponential effect on mean observation:
    vB_obs <- vB_fish[t] * exp(sd_survey * (rnorm(1)) - 0.5 * (sd_survey)^2)
    # more code below...
```

When a survey is triggered, we first reset the value t_last_ass, to make sure that we are triggering assessments at the desired `ass_int` interval. We then randomly sample the population to get an estimate of the biomass vulnerable to fishing–we assume this alue is lognormally distributed with error term σ_{survey}. This is the value of the x-axis for vulnerable biomass that we will use set up our specific control rule.

We set the TAC for the linear harvest control rules as follows:

```r
if (rule == "linear") {
  TAC <- c_slope * (vB_obs - b_lrp)
}
```

which is exactly how we calculated this total allowable catch or `TAC` up in the first plot of the tutorial, with the exception that we are now assuming uncertainty in the x-axis measurement.

There is a chunk of code below this that only applies when the harvest control rule is rectilinear in form, and so we jump down to the next relevant line:

```r
if (TAC < 0) {
  TAC <- 0
}
Ut <- ifelse((TAC / vB_fish[t]) < Ut_overall, (TAC / vB_fish[t]), Ut_overall)
```

The first chunk of this is handling the `TAC` variable to ensure it does not go below zero. Anglers cannot harvest fewer than zero fish, so this should make some sense.

The next line calculates the exploitation rate in year t using an ifelse() statement. It says that exploitation rate U_t is equal to the `TAC` divided by the biomass vulnerable to fishing in year t if this calculation is less than some term `Ut_overall`. If this rate gets higher than `Ut_overall`, we set U_t to `Ut_overall`. We did this because we will use `Ut_overall` to help us model discard mortality in a bit.

We set `Ut_overall` near the value we used for our F_{early} prior in the Bayesian model. Our rationale for this was that it was unlikely that angling alone would result in exploitation rates higher than values observed during the invisible collapse (see Post et al. 2002, Sullivan 2003), when angling was virtually unrestricted.

Also, the creel survey data that we examined showed no clear trend in overall fishing effort, and when we set the `Ut_overall` value near 0.5 the simulation model matches the qualitative observation from biologists on the ground that fish disappear from the fishery shortly after being exposed to fishing (i.e., few fish beyond minimum size limits in some lakes).

Given this information, we are going to assume `Ut_overall` has been stable over time at a value near our F_{early} prior. It would, however, be relatively straightforward to model growth in fishing demand through time (we don't do this, but you could).

Next up, we calculate a second term for our discard mortality calculations, which we call `rett`. This term is calculated as:

```r
rett <- ifelse(Ut / Ut_overall <= 1.0, Ut / Ut_overall, 1.0)
# cap rett annual retention proportion at 1.0
```

The `rett` term is the annual retention proportion of fish caught that are kept by anglers. Anglers cannot retain more than 100% of the fish they catch, and so we used this line of code to ensure that this was the case. Again, this is just another term we needed to model catch and release mortality. We will deal with these catch and release mortality terms when we actually update the age structure and expose our numbers at age matrix to fishing.

Updating recruitment and the numbers at age matrix

```
# stock-recruitment
if (rec_ctl == "ricker") {
  Rpred[t] <- rec_a * SSB[t] * exp(-rec_b * SSB[t] + wt[t])
}
if (rec_ctl == "bh") {
  Rpred[t] <- rec_a * SSB[t] * exp(wt[t]) / (1 + rec_b * SSB[t])
}
```

This chunk of code is similar to the BERTA stan code that we walked through in an earlier tutorial. We are just setting the predicted recruits for year t according to either Ricker or Beverton-Holt stock-recruitment dynamics.

Now we actually expose the simulated critters to both natural and fishing mortality:

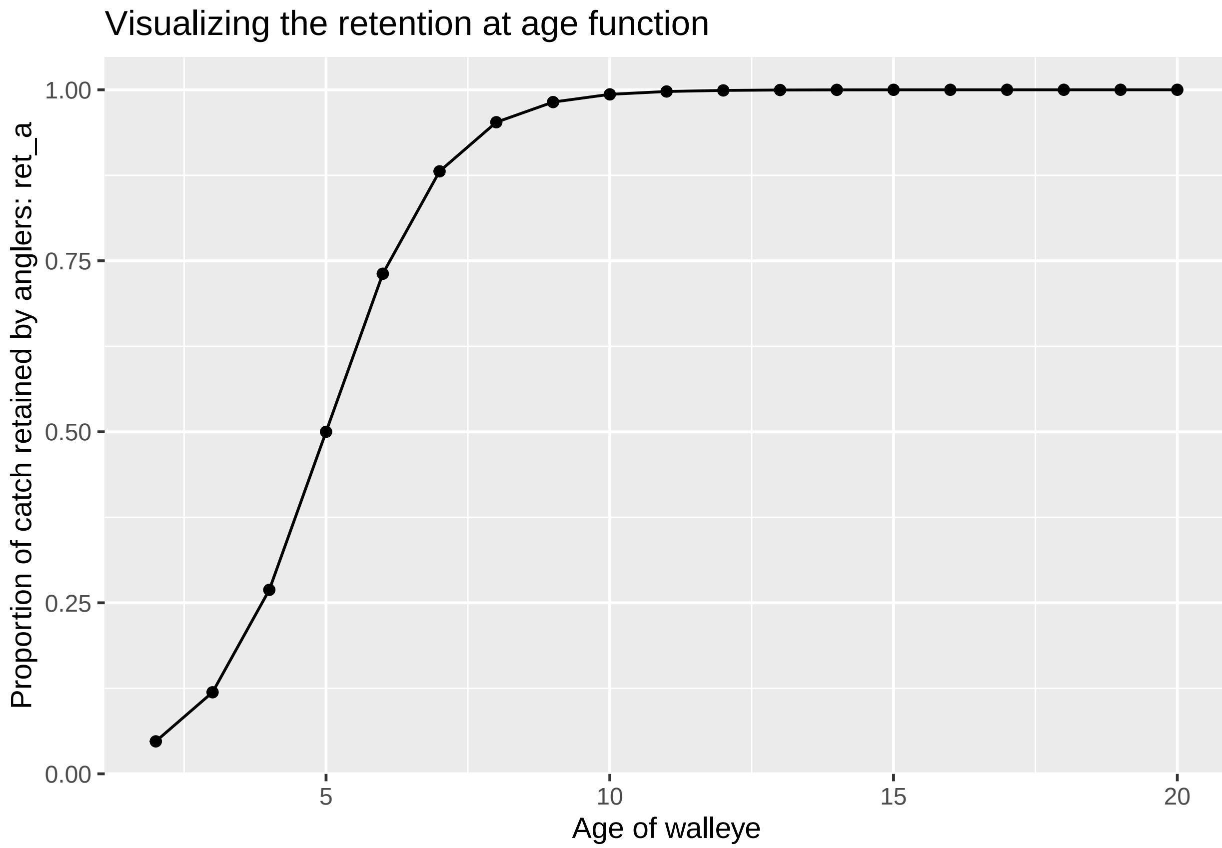
```
# update the age structure
for (a in seq_len(n_ages)[-n_ages]) { # ages 2-19
  nta[t + 1, a + 1] <- nta[t, a] * exp(-M_a[a]) *
  (1 - Ut_overall * v_fish[a] * (ret_a[a] * rett + (1 - ret_a[a] * rett) * d_mort))
}
```

The first line inside the loop is straightforward, and we calculate the number of fish at each age in the next time step as a function of the critters there previously reduced by some natural mortality rate `M_a`.

The second line within the loop is less intuitive and quite nasty. It is where we decrement the numbers at age by the finite rate at which fish survive both fishing and discard mortality processes. This is where all those pesky catch and release modeling terms finally get used. So let's walk through it bit by bit.

Let's start by defining two parameters that we have not yet defined. The first new term is ret_a, which is the proportion of fish retained at each age by anglers. We assumed ret_a was specified by the following relationship:

```
ages <- 2:20
ah_ret <- 5
sd_ret <- 1
ret_a <- 1 / (1 + exp(-(ages - ah_ret) / sd_ret))

data <- data.frame(ages, ret_a)

data %>%
  ggplot(aes(x=ages, y=ret_a))+
  geom_point() +
  geom_line() +
  ylab("Proportion of catch retained by anglers: ret_a")+
  xlab("Age of walleye") +
  ggtitle("Visualizing the retention at age function")
```



Note there are many other ways you could set this ret_a term. This seemed intuitive to us and so we will run with it for now. You might come back and play with this in the future.

The other new term we need to define is just the finite discard mortality rate U_d. We will eventually simulate across several U_d values and so the exact number does not matter that much for now. Given these two new terms we can think about the tail end of the miserable line of code above, which again was:

```
# understanding the misery line:
 (1 - Ut_overall * v_fish[a] * (ret_a[a] * rett + (1 - ret_a[a] * rett) * d_mort))
```

This is saying that the fraction of fish that survive fishing related mortality at each age is 1 minus the number of fish that die due to fishing via `Ut_overall` times v_{f_a} multiplied by some additional term contained in parentheses. This big honking term in parentheses first specifies the number of fish that die because they were kept by anglers as ret_a times $rett$ (the retention proportion we set above).

We then use some algebraic trickery to specify the number of fish that died due to discard mortality as 1 minus the proportion that were kept (if we didn't keep them we must have released them), and we multiply this released proportion by U_d. There is a lot of detail packed into that single line, and so it might behoove you to play around with the math on your own if you have the time.

Once we have updated the numbers at age and dealt with natural and fishing mortality, we move along in the code and set the number of recruits in year t + 2:

```
# set rec value for t + 2
nta[t + 2, 1] <- Rpred[t]
```

We then "just" need to run all those calculations for all years t, all c_{slope} and b_{lrp} values, and all random draws from the posterior. We also record performance metrics and close off all the loops of misery and move on with our lives.

Life after the loops of misery

Everything that happens in the `get_hcr()` function below the previous line of code is actually just processing the output from our simulation or closing our loops. Then there's a bunch of code that just creates a big list and saves it in the `sims/` folder with a unique file name.

All of this code is processing our simulation run output:

https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L296-L358

Really, the only important things to note in that code are that we calculate and store average yield and HARA utility for each c_{slope}, b_{lrp}, and simulation year t in these lines:

https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L297-L301

and where $pp = 0.3$ as per our section on HARA utility objectives.

I recorded some additional performance measures in this section in case folks were interested. I recorded the proportion of times that the fishery fell below some threshold in the `prop_below` matrix, and the number of times the total allowable catch was actually zero in the `TAC_zero` matrix. I have yet to use these but you should know they exist, if only so that you see how simple it is to kick out whatever performance metric your heart desires in this part of the code.

You'll also note that we correct the calculated yields and utilities in these lines so as to get annual values here:

https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L312-L313

and we put a ton of stuff in a big list and save it here:

https://github.com/ChrisFishCahill/managing-irf-complexity/blob/main/r-code/hcr.R#L335-L357

That's it. We made it through `get_hcr()`.

Running the harvest control rule simulations using {furrr}

We set the `get_hcr()` function up in much the same way we set up the `get_fit()` function that we used to fit the stan models in an earlier tutorial. This means that we can run all of our harvest control rule simulations across many different simulation parameter inputs in parallel using {furrr}.

To do this, we just declare some run parameters for our simulation:

```
#----------------------------------------------------------------------
# declare some values for the simulations
#----------------------------------------------------------------------

n_draws <- 30
n_repeats <- 8 # recruitment repeats
retro_initial_yr <- 1990 # initial year for retrospective analysis
retro_terminal_yr <- 2015
n_sim_yrs <- length(retro_initial_yr:retro_terminal_yr) * n_repeats
ages <- 2:20
t <- 2000 # first survey year
max_a <- max(ages) # max age
recruit_a <- min(ages) # age at recruitment
```

```r
initial_yr <- t - max_a + recruit_a - 2 # 1980 = BERTA initial yr
initial_yr_minus_one <- initial_yr - 1
ah_ret <- 5
sd_ret <- 1
ret_a <- 1 / (1 + exp(-(ages - ah_ret) / sd_ret)) # retention by age vector
Ut_overall <- 0.5 # annual capture rate of fully vulnerable fish
sbo_prop <- 0.1 # performance measure value to see if SSB falls below sbo_prop*sbo
rho <- 0.6 # correlation for recruitment terms
sd_wt <- 1.1 # std. dev w(t)'s
psi_wt <- 0.5 # weighting multiplier for wt_historical vs. wt_sim, aka "wthistory" in spreadsheets
grid_size <- 75 # how many bmins, round(grid_size/2) gives how many cslope
```

And then we put it all in a tagged list that we will reference when we actually call the get_hcr() function:

```r
# put it all in a tagged list
hcr_pars <- list(
  "n_draws" = n_draws,
  "n_repeats" = n_repeats,
  "retro_initial_yr" = retro_initial_yr,
  "retro_terminal_yr" = retro_terminal_yr,
  "n_sim_yrs" = n_sim_yrs,
  "ages" = ages,
  "initial_yr" = initial_yr,
  "initial_yr_minus_one" = initial_yr_minus_one,
  "ret_a" = ret_a,
  "Ut_overall" = Ut_overall,
  "sbo_prop" = sbo_prop,
  "rho" = rho,
  "sd_wt" = sd_wt,
  "psi_wt" = psi_wt,
  "grid_size" = grid_size
)
```

We then read in the fits that we want to pass to the harvest control rule simulator. In this case, we choose the Beverton-Holt fits where the informative prior for compensation ratio $\phi = 6$ and read those specific fits in. Then we set up a vector of lake names (spaces separated by underscores):

```r
#----------------------------------------------------------------------
# read in the stan fits and run retrospective mse
#----------------------------------------------------------------------

# extract some saved .stan fit names
paths <- dir("fits/", pattern = "\\.rds$")
paths <- paths[grep("bh_cr_6", paths)]
paths <- paste0(getwd(), "/fits/", paths)

fits <- map(paths, readRDS) %>%
  set_names(paths)

which_lakes <- str_extract(
  string = paths,
  "(?<=fits/).*(?=_bh|_ricker)"
)
```

Let's say for the sake of argument that we then want to run these simulations across a suite of different assessment intervals, survey error terms, and discard mortalities. To do this, we set up a dataframe called `to_sim` that has columns corresponding to the inputs for our get_hcr() function. We do some manipulations on `to_sim` to ensure we have all possible combinations of `which_lake`, `ass_int`, `sd_survey`, `d_mort`, and in this case we set the `rule` = "linear":

```
# things to simulate hcrs across
ass_ints <- c(1, 3, 5, 10) # how often to assess / run FWIN
sd_surveys <- c(0.05, 0.4) # survey sd
d_morts <- c(0.03, 0.15, 0.3) # discard mortalities

# use expand.grid() and distinct() to get all possible combinations
to_sim <- expand.grid(which_lakes, ass_ints, sd_surveys, d_morts)
names(to_sim) <- c("which_lake", "ass_int", "sd_survey", "d_mort")
to_sim <- to_sim %>% distinct()
glimpse(to_sim)
to_sim$rule <- "linear"
```

And finally, after all of that we get to run this beast. This code runs all your simulations and saves them with a convenient file name in the `sims/` folder:

```
# if you run all these your gov computer may explode
if (FALSE) {
  options(future.globals.maxSize = 8000 * 1024^2) # 8 GB
  future::plan(multisession)
  system.time({
    future_pwalk(to_sim, get_hcr,
      .options = furrr_options(seed = TRUE)
    )
  })
}
```

The future_pwalk() call is inside an if-loop set to FALSE. I did this so it does not just start running and accidentally crash your computer. If you do indeed run the inside of this loop it would run 144 different simulation experiments given the Beverton-Holt $\phi = 6$ BERTA fit.

You could use this simulation framework to run a harvest control rule simulator for the Ricker fits or or for different $\phi = 12$, too. You could play around with different assessment intervals, survey error, or discard mortality values. The point of going through all that miserable coding was so that we have some freedom now with what we actually simulate.

Results for MAY and HARA objectives for some of the simulation runs

I ran the harvest control rule simulator for the Beverton-Holt $\phi = 6$ Bayesian model fits across a range of σ_{survey}, different assessment intervals, and a set of U_d values. These specfic settings were shown above in the code. With those results, I can plot the recruitment anomaly w_t sequences used for the the simulations to get an idea of the recruitment variation we are testing our policies against (see Fig. 2 below).

We can also plot the performance of our policies in terms of how well they achieve either MAY or HARA objectives. Early on in the tutorial we said that optimizing MAY and HARA objectives in many theoretical studies has shown that when MAY was the objective to be maximized you typically get an answer with a non-zero fixed escapement b_{lrp} point and a c_{slope} value near one. When HARA is the objective, we said that the b_{lrp} value tended toward zero and the c_{slope} term was closer to F_{msy}.

Because we simulated across an entire policy space, we can visualize these solutions using isopleths. When we make isopleths of the simulation output, we can see that on average the best b_{lrp} values are greater than zero and the c_{slope} values are near one when our objective was to maximize MAY (Fig. 3). Calling lake is a bit different, but in general harvesting theory appears to hold.

Next up, we can compare the Fig. 3 result with the isopleths for HARA utility (Fig. 4). What you see is that the b_{lrp} values do indeed tend to slide toward the x-y origin, and that in general the c_{slope} values are less than they were in the previous plot. Figs 3-4 are showing you there is a tradeoff between policies that seek to maximize MAY vs. those that maximize consistency in catch via HARA utility.

In addition to the isopleths, we can start to do some neat stuff because we simulated across many axes of uncertainty. For example, we can look at the best linear harvest control policies for MAY while varying the interval at which walleye fisheries are assessed. This could be useful as it is unlikely there is enough money or personnel to do surveys each year. We show the result of increasing the assessment intervals on the best harvest control rules for MAY in Fig. 5, and we make the analogous plot to visualize the harvest control rules across assessment intervals for the HARA utility objective in Fig. 6.

We can play other management games and explore all sorts of relevant questions, too. For example, does our performance (in terms of getting MAY or HARA utility) degrade when surveys are conducted more infrequently? We plot this for the MAY objective in Fig. 7, and it looks about the same as the plot for the HARA objective (Fig. 8). In this case performance doesn't appear degrade much through time, probably because there's high uncertainty in our surveys (i.e., $\sigma_{survey} = 0.4$).

These are all the plots we will look at for now, but I hope you realize that the world is really your oyster here and there are lots of interesting things you could kick out from a simulation analysis like this.

Conclusion

Just to wrap up, let's think about what we have acheived in this tutorial. We started out by discussing the importance of explicit objectives for resource management, particularly if we are to evaluate those objectives via simulation. We talked about MAY and HARA objectives extensively. We then jumped into the `hcr_r()` simulation script, and talked through all the intricacies of reading data into this function from a particular Bayesian model fit for use in our simulation script.

We spent a chunk of time discussing recruitment anomalies, including how to set them up to represent some sort of weighted average between our estimated values and an auto-correlated process. From there, we jumped into the age-structured model portion of the simulation script. This was very similiar in form to the model we fitted in our Stan tutorials, with the exception that we introduced some new terms to deal with catch and release mortality. We then stored performance metrics and saved a bunch of simulation output with a nice file name in the `sims/` folder.

Lastly, we ran our simulation models using {furrr}, using methods and techniques we learned in earlier tutorials. Upon running and saving those simulation runs, we plotted our findings to visualize some of the results.

References:

Bertsekas 2019. Reinforcement Learning and Optimal Control.

Cahill et al. 2021. Unveiling the recovery dynamics of walleye after the invisible collapse. Canadian Journal of Fisheries and Aquatic Sciences.

Clark, C.W. 1974. Mathematical bioeconomics. In Mathematical Problems in Biology. Springer. pp. 29-45.

Clark, C.W., and De Pree, J.D. 1979. A simple linear model for the optimal exploitation of renewable resources. Applied Mathematics and Optimization 5(1): 181-196

Conroy and Peterson. 2013. Decision making in natural resource management: a structured, adaptive approach.

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

Deroba and Bence 2008. A review of harvest policies: Understanding relative performance of control rules. Fisheries Research.

Edwards and Dankel 2016. Management science in fisheries: an introduction to simulation-based methods. Routledge.

Gelman et al. 2013. Bayesian data analyis.

Larkin. 1977. Epitaph on maximum sustainable yield. Transactions of the American Fisheries Society.

Mangel, M. 1984. Decision and control in uncertain resource systems. Academic Press, Inc.

Moxnes 2002. Policy sensitivity analysis: simple vs. complex models. Centre for Fisheries Economics Discussion Paper No. 13/2002.

Post et al. 2002. Canada's recreational fisheries: the invisible collapse? Fisheries.

Restrepo and Powers 1999. Precautionary control rules in US fisheries management: specification and performance. ICES Journal of Marine Sci.

Sullivan 2003. Active management of walleye fisheries in Alberta: dilemmas of managing recovering fisheries. North American Journal of Fisheries Management. 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.

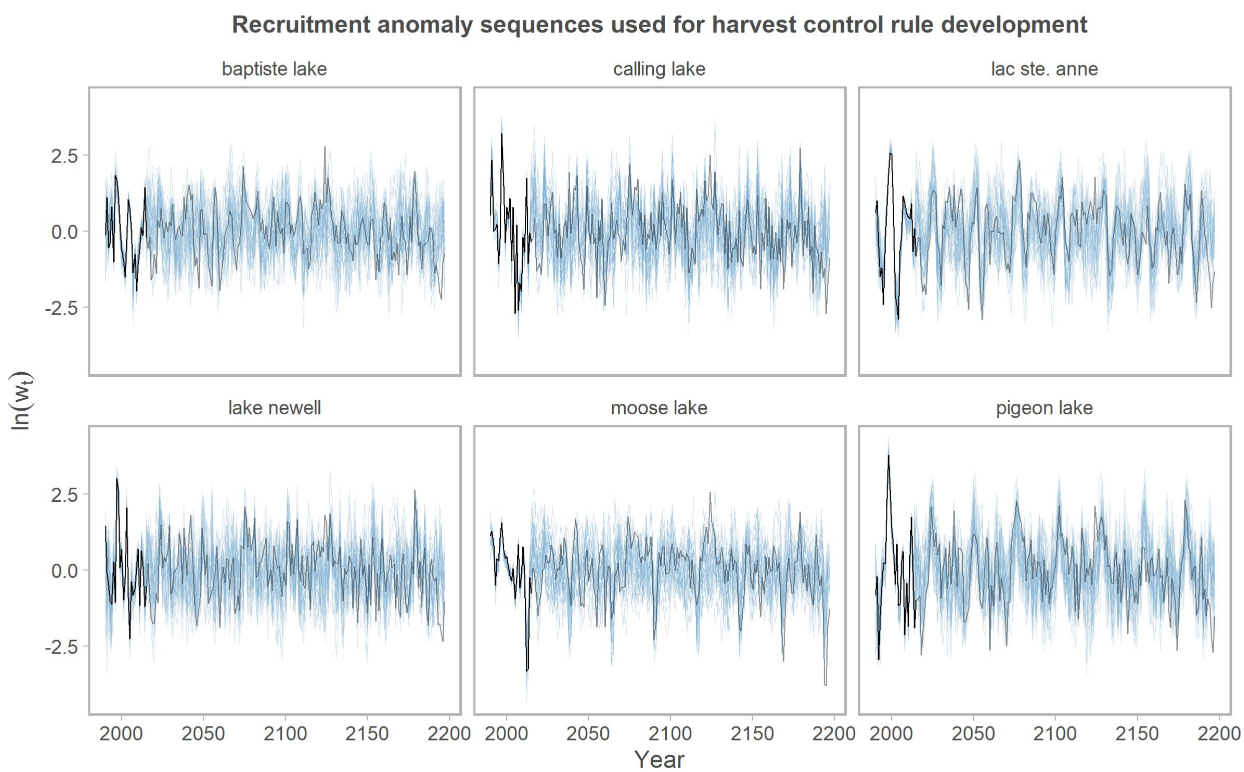


Figure 2: Recruitment anomalies w_t that we use for harvest control rule development. Black line represents one estimated recruitment anomaly sequence, while the light grey line shows the evolution of the simulated recruitment sequence into the future. Light blue lines show individual recruitment anomaly trajectories.

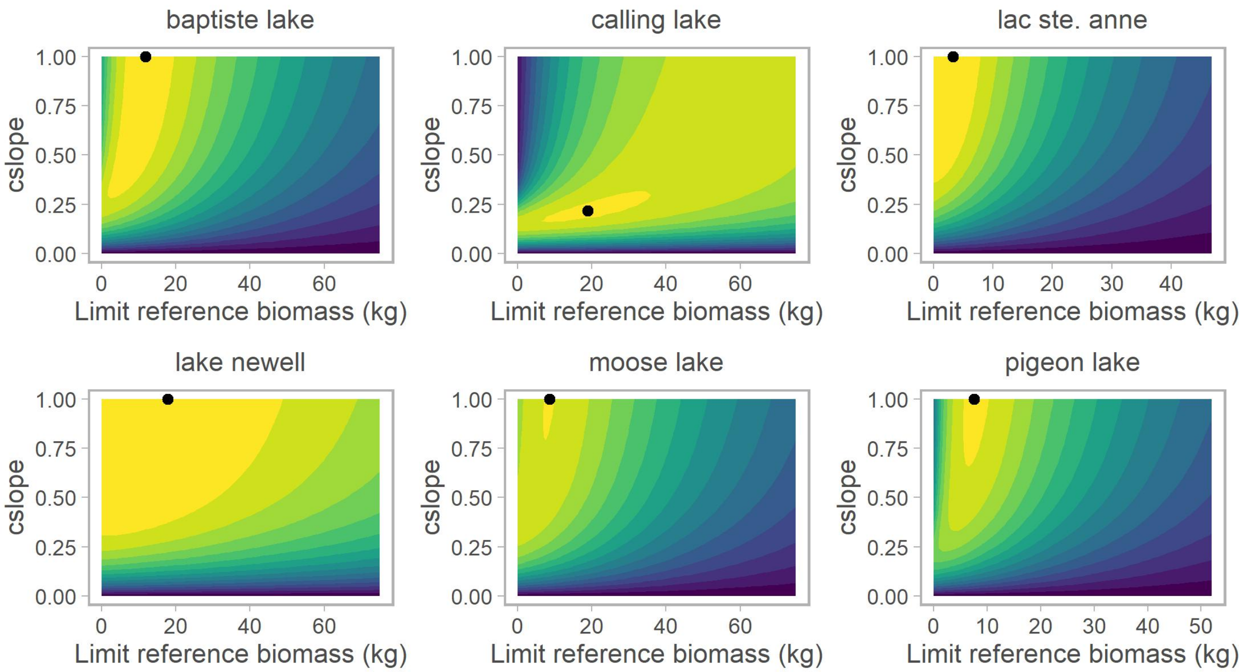


Figure 3: Approximation in policy space results for maximum average yield (MAY) objective. Black dot represents the best linear policy given the MAY objective. This isopleth was created using an assessment interval of 1 year, $\sigma_{survey} = 0.4$, and $U_d = 0.3$.

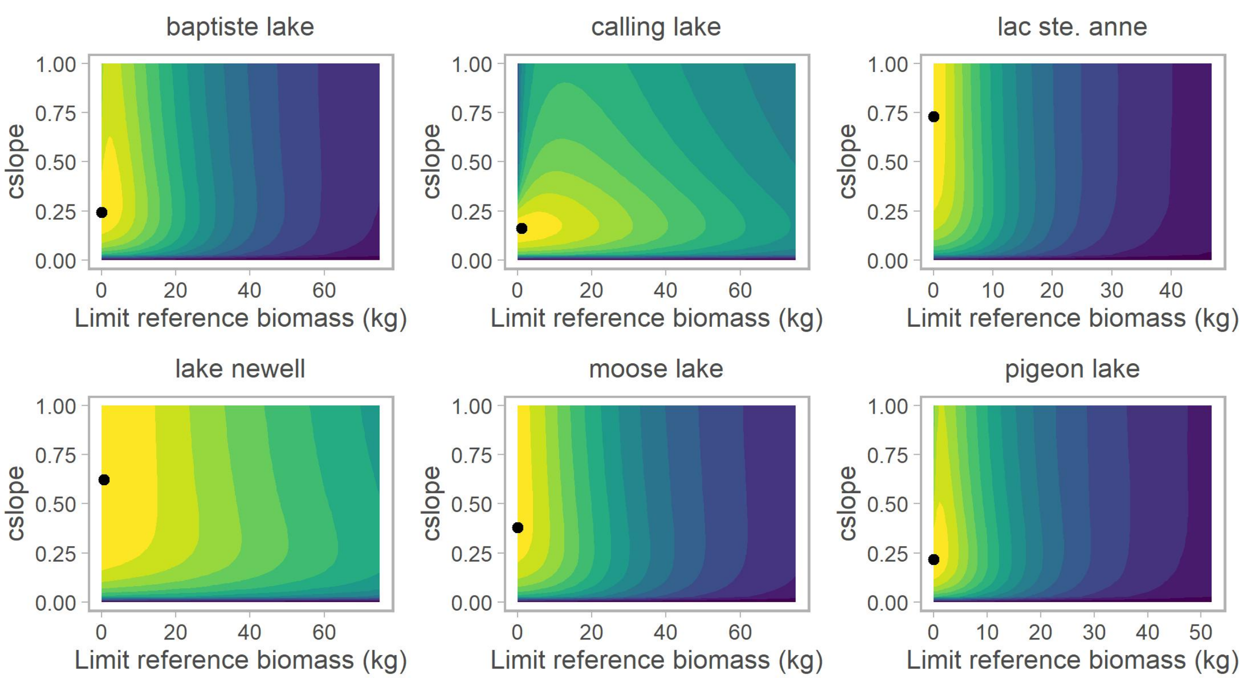


Figure 4: Approximation in policy space results for the HARA utility objective. Black dot represents the best linear policy given the HARA objective. This isopleth was created using an assessment interval of 1 year, $\sigma_{survey} = 0.4$, and $U_d = 0.3$.

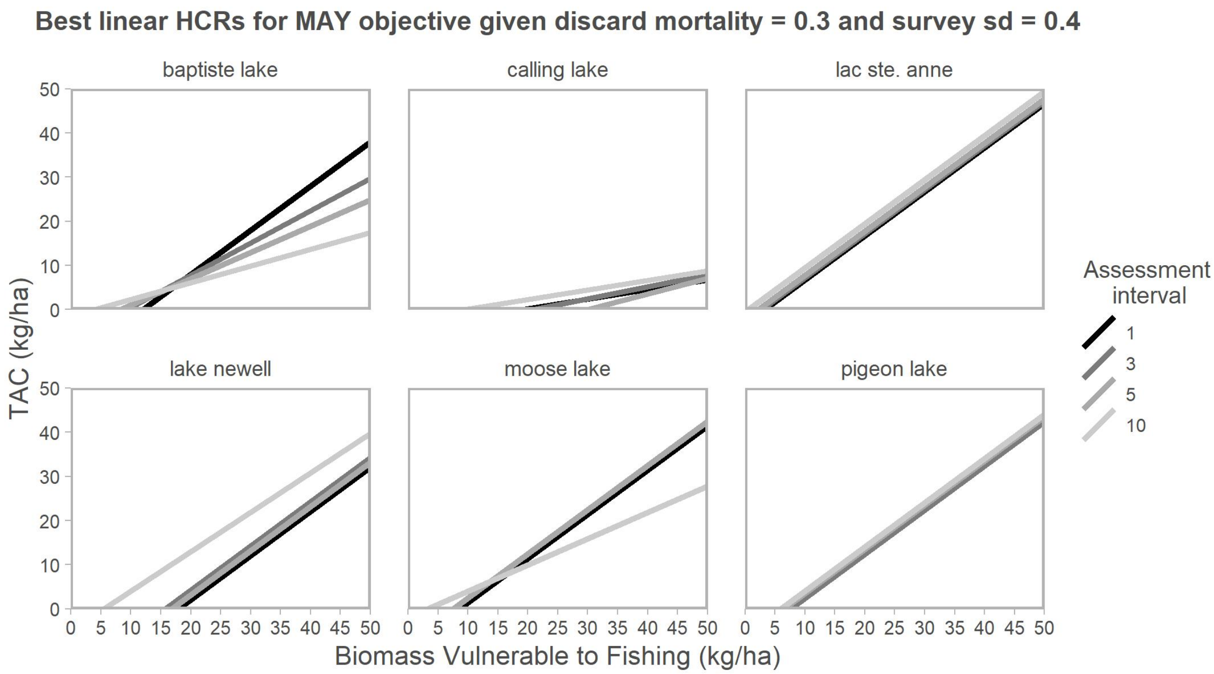


Figure 5: Best linear harvest control rules for MAY objective, where line color indicates the interval at which the fishery is assessed. This plot was created using $\sigma_{survey} = 0.4$, and $U_d = 0.3$.

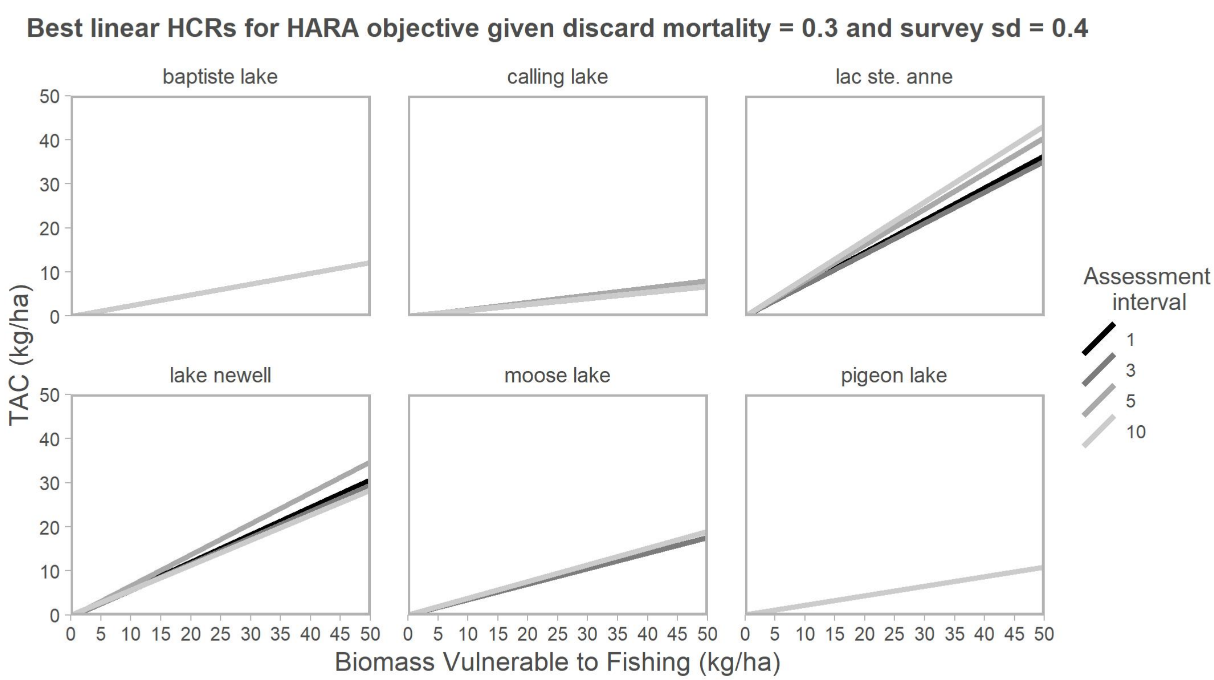


Figure 6: Best linear harvest control rules for HARA utility objective, where line color indicates the interval at which the fishery is assessed. This plot was created using $\sigma_{survey} = 0.4$, and $U_d = 0.3$.

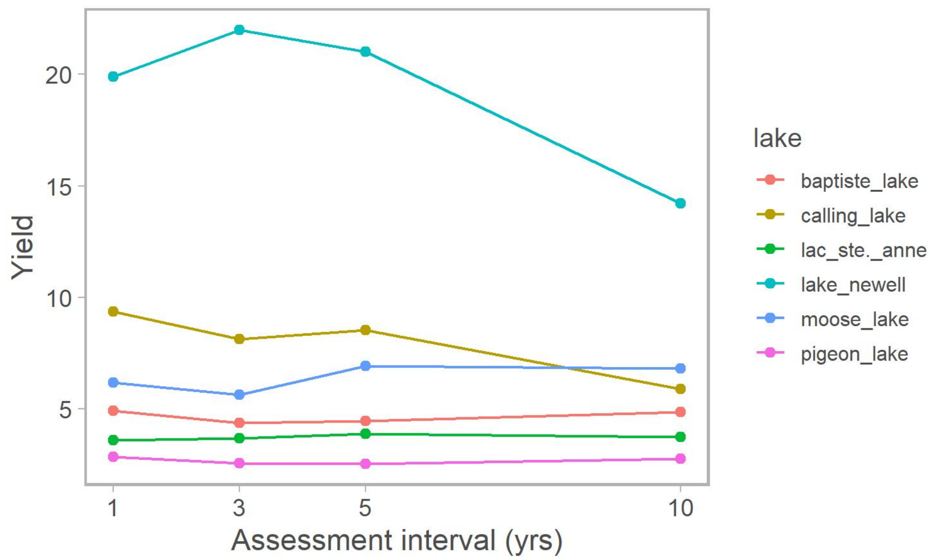


Figure 7: Performance (MAY) of best polcies vs. assessment interval for each lake in the contract. This plot was created using $\sigma_{survey} = 0.4$, and $U_d = 0.3$.

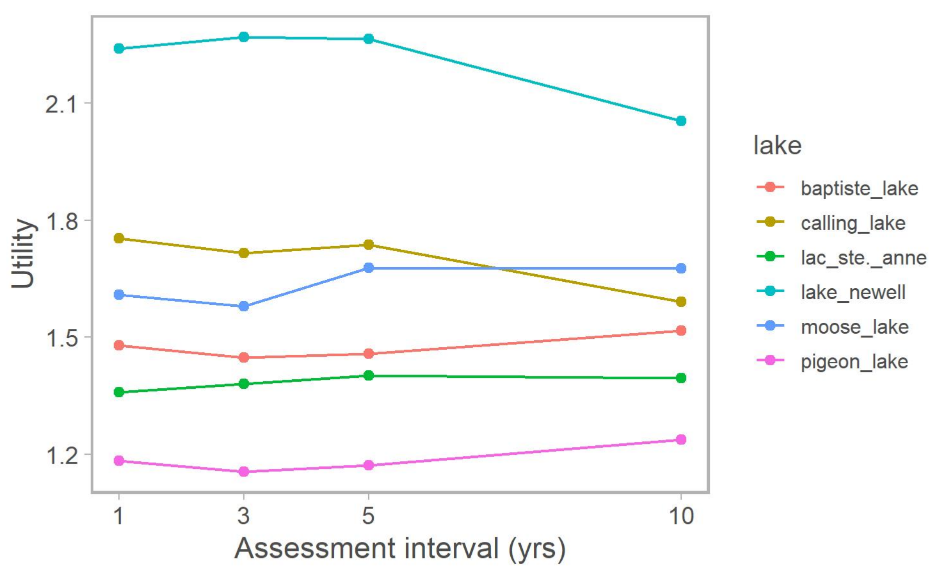


Figure 8: Performance (HARA) of best polcies vs. assessment interval for each lake in the contract. This plot was created using $\sigma_{survey} = 0.4$, and $U_d = 0.3$.