Appendix B: Code for analysis comparing to historical data

2018-11-23

Section 1: Model estimates from data

Computes model parameter estimates for selected stocks in RAM using NIMBLE.

```
# devtools::install_github("boettiger-lab/sarsop")  ## install package first if necessary.
library(tidyverse)
library(sarsop)
library(nimble)
library(parallel)
library(gridExtra)
library(tictoc)
tic()
```

```
if(!file.exists("ramlegacy.zip")){
download.file(paste0(
  "https://depts.washington.edu/ramlegac/wordpress/databaseVersions/",
  "RLSADB_v3.0_(assessment_data_only)_excel.zip"),
  "ramlegacy.zip")
}
```

```
path <- unzip("ramlegacy.zip")
sheets <- readxl::excel_sheets(path)
ram <- lapply(sheets, readxl::read_excel, path = path)
names(ram) <- sheets
unlink("ramlegacy.zip")
unlink("RLSADB v3.0 (assessment data only).xlsx")
ramlegacy <-
  ram$timeseries_values_views %>%
  select(assessid, stockid, stocklong, year, SSB, TC) %>%
  left_join(ram$stock) %>%
  left_join(ram$area) %>%
  select(assessid, stockid, scientificname,
         commonname, areaname, country, year,
         SSB, TC) %>%
  left_join(ram$timeseries_units_views %>%
              rename(TC_units = TC, SSB_units = SSB)) %>%
  select(scientificname, commonname,
         stockid, areaname, country, year,
         SSB, TC, SSB_units, TC_units)
```

Let's filter out missing data, non-matching units, and obvious reporting errors (catch exceeding total spawning biomass), then we re-scale each series into the 0,1 by appropriate choice of units:

```
df2 <- ramlegacy %>%
  filter(!is.na(SSB), !is.na(TC)) %>%
  filter(SSB_units == "MT", TC_units=="MT") %>%
  filter(SSB > TC) %>%
  select(-SSB_units, -TC_units) %>%
```

```
  group_by(stockid) %>%
  mutate(scaled_catch = TC / max(SSB),
         scaled_biomass = SSB / max(SSB))
```

```
stock_ids <- c("PLAICNS", "ARGHAKENARG")
examples <- df2 %>%
  filter(stockid %in% stock_ids) %>%
  ungroup() %>%
  group_by(commonname)
```

```
## Model does not estimate sigma_m; data is insufficient to do so.
gs_code  <- nimble::nimbleCode({
  r ~ dunif(0, 2)
  K ~ dunif(0, 2)
  sigma ~ dunif(0, 1)

  x[1] <- x0
  for(t in 1:(N-1)){
    mu[t] <- x[t] + x[t] * r * (1 - x[t] / K) - min(a[t],x[t])
    x[t+1] ~ dnorm(mu[t], sd = sigma)
  }
})

fit_models <- function(fish, code){
  # fish <- examples %>% filter(stockid == stock_ids[1])

  ## Rescale data
  N <- dim(fish)[1]
  scaled_data <- data.frame(t = 1:N, y = fish$scaled_biomass, a = fish$scaled_catch)
  data = data.frame(x = scaled_data$y)

  ## Compile  model
  constants <- list(N = N, a = scaled_data$a)
  inits <- list(r = 0.5, K = 0.5, sigma = 0.02, x0 = scaled_data$y[1])
  model <- nimbleModel(code, constants, data, inits)
  C_model <- compileNimble(model)

  mcmcspec <- configureMCMC(model, thin = 1e2)
  mcmc <- buildMCMC(mcmcspec)
  Cmcmc <- compileNimble(mcmc, project = model)
  Cmcmc$run(1e6)


  samples <- as.data.frame(as.matrix(Cmcmc$mvSamples))
  burnin <- 1:(0.05 * dim(samples)[1]) # drop first 5%
  samples <- samples[-burnin,1:(length(inits) - 1)] # drop raised vars, burnin
  #gather(samples) %>% ggplot() + geom_density(aes(value)) + facet_wrap(~key, scale='free')

  ## Return fit
  data.frame(stockid = fish$stockid[1],
             commonname = fish$commonname[1],
             r = mean(samples$r),
             K = mean(samples$K),
             sigma_g = mean(samples$sigma),
```

```
              r_sd = sd(samples$r),
              K_sd = sd(samples$K),
              sigma_g_sd = sd(samples$sigma),
              stringsAsFactors = FALSE)

}
```

```
set.seed(123)
fits <- examples %>% do(fit_models(., code=gs_code))
```

```
|------------|------------|------------|------------|
|------------------------------------------------------|
|------------|------------|------------|------------|
|------------------------------------------------------|
```

```
fits
```

```
# A tibble: 2 x 8
# Groups:   commonname [2]
  stockid     commonname          r     K sigma_g   r_sd  K_sd sigma_g_sd
  <chr>       <chr>           <dbl> <dbl>   <dbl>  <dbl> <dbl>      <dbl>
1 ARGHAKENARG Argentine hake  1.04   1.20   0.112 0.177  0.188     0.0263
2 PLAICNS     European Plaice 0.906  1.78   0.128 0.0734 0.155     0.0130
```

```
pars <- fits %>% ungroup() %>% select(commonname, r, K, sigma_g)
pars
```

commonname	r	K	sigma_g
Argentine hake	1.0387783	1.196112	0.1118370
European Plaice	0.9055933	1.778186	0.1275455

Calculations of the Decision Policies for Historical Data

```
options(mc.cores = parallel::detectCores()) # Reserve ~ 10 GB per core
log_dir <- "../data/ram_ex"
```

```
## Classic Gordon-Schaefer. Note that recruitment occurs *before* harvest
gs <- function(r,K){
  function(x, h){
    x + x * r * (1 - x / K) - pmin(x,h)
  }
}
reward_fn <- function(x,h) pmin(x,h)
discount <- .95
```

Discretize space

Note that the large values of K require we carry the numerical grid out further.

```
states <- seq(0,4, length=150)
actions <- states
observations <- states
```

Consider all parameter values combinations for which we want solutions (both species at each of three possible levels of measurement uncertainty; though we will focus on the `0.1` level for simplicity as overall pattern is the same at `0.15`):

```
meta <- expand.grid(commonname = pars$commonname,
                    sigma_m = c(0, 0.1, 0.15),
                    stringsAsFactors = FALSE) %>%
  left_join(pars) %>%
  mutate(scenario  = as.character(1:length(sigma_m)))

meta
```

commonname	sigma_m	r	K	sigma_g	scenario
Argentine hake	0.00	1.0387783	1.196112	0.1118370	1
European Plaice	0.00	0.9055933	1.778186	0.1275455	2
Argentine hake	0.10	1.0387783	1.196112	0.1118370	3
European Plaice	0.10	0.9055933	1.778186	0.1275455	4
Argentine hake	0.15	1.0387783	1.196112	0.1118370	5
European Plaice	0.15	0.9055933	1.778186	0.1275455	6

Create the model matrices (transition, observation, and reward matrix):

```
models <-
  parallel::mclapply(1:dim(meta)[1],
                     function(i){
                       fisheries_matrices(
                         states = states,
                         actions = actions,
                         observed_states = observations,
                         reward_fn = reward_fn,
                         f = gs(meta[i, "r"][[1]], meta[i, "K"][[1]]),
                         sigma_g = meta[i,"sigma_g"][[1]],
                         sigma_m = meta[i,"sigma_m"][[1]],
                         noise = "normal")
                     })
```

Here's the slowest part: computing POMDP alpha vectors.

```
dir.create(log_dir)
options(mc.cores = 1)

## POMDP solution (slow, >20,000 seconds per loop memory intensive)
system.time(
  alphas <-
    parallel::mclapply(1:length(models),
    function(i){
      log_data <- data.frame(model = "gs",
                             r = meta[i, "r"][[1]],
                             K = meta[i, "K"][[1]],
                             sigma_g = meta[i,"sigma_g"][[1]],
```

```
                              sigma_m = meta[i,"sigma_m"][[1]],
                              noise = "normal",
                              commonname = meta[i, "commonname"][[1]],
                              scenario = meta[i, "scenario"][[1]])

      sarsop(models[[i]]$transition,
             models[[i]]$observation,
             models[[i]]$reward,
             discount = discount,
             precision = 2e-6,
             timeout = 25000,
             log_dir = log_dir,
             log_data = log_data)
    })
)
```

```
meta <- meta_from_log(data.frame(model="gs"), log_dir)  %>%
  mutate(scenario = as.character(scenario)) %>%
  left_join(
    select(meta, sigma_m, commonname, scenario),
    by = c("scenario", "sigma_m", "commonname")) %>%
  arrange(scenario)
alphas <- alphas_from_log(meta, log_dir)
#models <- models_from_log(meta)
```

Comparison to the static models

```
pars <- examples %>%
  group_by(commonname) %>%
  summarise(N = max(SSB)) %>%
  right_join(
    meta %>%
      select(commonname, r, K) %>%
      distinct())
```

Add corresponding static policy levels on:

```
statics <- function(P){
  f <- gs(P$r, P$K)
  S_star <- optimize(function(x) -f(x,0) + x / discount, c(0, 2* P$K))$minimum
  B_MSY <- S_star
  MSY <- f(B_MSY,0) - B_MSY

  tibble(S_star, F_MSY = MSY / B_MSY, F_TAC = 0.8 * F_MSY,
         commonname = P$commonname, N = P$N)
}

policy_pars <-
  pars %>%
  rowwise() %>%
  do(statics(.))
```

Convert example data into discrete index space.

```
index <- function(x, grid) map_int(x, ~ which.min(abs(.x - grid)))
## repeats each series for each static model
ex <- examples %>%
  mutate(biomass = index(scaled_biomass, states),
         catch = index(scaled_catch, actions))  %>%
  left_join(policy_pars) %>%
  left_join(pars) %>%
  ungroup()
```

Static policy calculations:

```
CE_f <- function(S_star, r, K, i)
  index(pmax(gs(r[[1]],K[[1]])(states,0) - S_star[[1]],0), actions)[i]
MSY_f <- function(F_MSY, i) index(states * F_MSY[[1]], actions)[i]
TAC_f <- function(F_TAC, i) index(states * F_TAC[[1]], actions)[i]
rescale <- function(x, N) states[x]*N

historical <- ex %>%
  group_by(commonname) %>%
  mutate(CE =  CE_f(S_star, r, K, biomass),
         MSY =  MSY_f(F_MSY, biomass),
         TAC =  TAC_f(F_TAC, biomass)) %>%
  select(year, biomass, catch, CE, MSY, TAC, commonname, N) %>%
  gather(model, stock, -year, -commonname, -N) %>%
  mutate(stock = states[stock] * N) %>%
  select(-N)
```

Compute POMDP policy for historical data:

```
set.seed(123456)
pomdp_sims <-
  pmap_dfr(list(models, alphas, 1:dim(meta)[[1]]),
           function(.x, .y, .z){

             ## avoid NSE
             who <- (ex$commonname == meta[.z,"commonname"])
             df <- ex[who,]

             hindcast_pomdp(.x$transition, .x$observation, .x$reward, discount,
                            obs = index(df$scaled_biomass, states),
                            action = index(df$scaled_catch,states),
                            alpha = .y)$df %>%
              mutate(method = "pomdp") %>% # include a column labeling method
              mutate(year = ex[who, "year"][[1]])
           },
           .id = "scenario")
```

Join records:

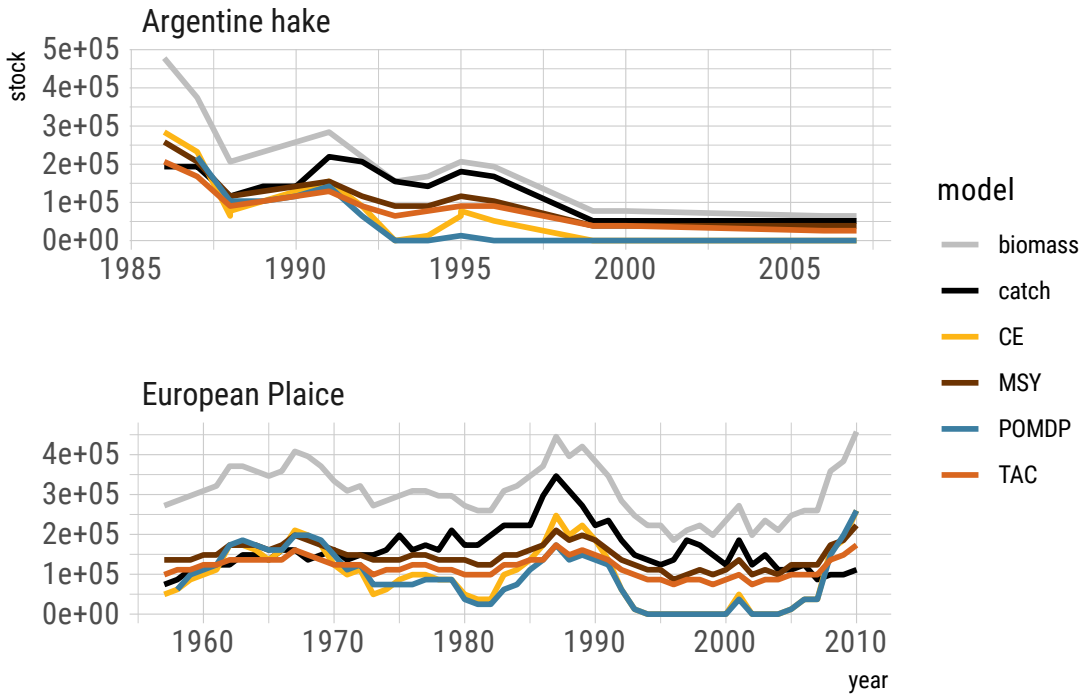
```
pomdp_sims <-
  meta %>%
  select(scenario, commonname,sigma_m) %>%
  left_join(pars) %>%
  right_join(pomdp_sims)
```

```
sims <- pomdp_sims %>%
  mutate(optimal = states[optimal] * N) %>% # original scale
  select(year, optimal, commonname, sigma_m) %>%
  rename(stock = optimal) %>%
  ## treat each sigma_m value as separate 'model'
  mutate(sigma_m = as.factor(sigma_m)) %>%
  mutate(model = recode(sigma_m,
                        "0" = "CE",
                        "0.1" = "POMDP",
                        "0.15" = "POMDP_0.15")) %>%
  select(-sigma_m) %>%
  bind_rows(historical)

write_csv(sims,file.path(log_dir, "ram_ex.csv"))
```

Final plot, as in paper but including MSY:

```
ram_ex <- read_csv(file.path(log_dir, "ram_ex.csv"))
ram_ex %>%
  filter(model %in% c("biomass", "catch", "POMDP", "CE", "TAC", "MSY")) %>%
  ggplot(aes(year, stock, col=model)) +
  geom_line(lwd=1) +
  scale_color_manual(values = colors) +
  facet_wrap(~commonname, scales = "free", ncol=1)
```



Total runtime:

```
toc()
```

```
283.786 sec elapsed
```

Hardware:

```
system2("grep", c("MemTotal", "/proc/meminfo"), stdout = TRUE)
```

```
[1] "MemTotal:       16432028 kB"
```

```
system2('grep', '-m 1 "model name" /proc/cpuinfo', stdout = TRUE)
```

```
[1] "model name\t: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz"
```

Software:

```
sessionInfo()
```

```
R version 3.5.1 (2018-07-02)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Debian GNU/Linux 9 (stretch)

Matrix products: default
BLAS: /usr/lib/openblas-base/libblas.so.3
LAPACK: /usr/lib/libopenblasp-r0.2.19.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] bindrcpp_0.2.2     hrbrthemes_0.5.0.1 extrafont_0.17
 [4] Cairo_1.5-9        ggthemes_4.0.1     printr_0.1
 [7] tictoc_1.0         gridExtra_2.3      nimble_0.6-12
[10] sarsop_0.5.0       forcats_0.3.0      stringr_1.3.1
[13] dplyr_0.7.8        purrr_0.2.5        readr_1.1.1
[16] tidyr_0.8.2        tibble_1.4.2       ggplot2_3.1.0
[19] tidyverse_1.2.1

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.0       lubridate_1.7.4  lattice_0.20-35  assertthat_0.2.0
 [5] rprojroot_1.3-2  digest_0.6.18    utf8_1.1.4       R6_2.3.0
 [9] cellranger_1.1.0 plyr_1.8.4       backports_1.1.2  evaluate_0.12
[13] coda_0.19-2      httr_1.3.1       highr_0.7        pillar_1.3.0
[17] rlang_0.3.0.1    lazyeval_0.2.1   readxl_1.1.0     rstudioapi_0.8
[21] extrafontdb_1.0  rmarkdown_1.10   labeling_0.3     igraph_1.2.2
[25] munsell_0.5.0    broom_0.5.0      compiler_3.5.1   modelr_0.1.2
[29] pkgconfig_2.0.2  htmltools_0.3.6  tidyselect_0.2.5 codetools_0.2-15
```

```
[33] fansi_0.4.0      crayon_1.3.4     withr_2.1.2      grid_3.5.1
[37] nlme_3.1-137     jsonlite_1.5     Rttf2pt1_1.3.7   gtable_0.2.0
[41] magrittr_1.5     scales_1.0.0     cli_1.0.1        stringi_1.2.4
[45] xml2_1.2.0       tools_3.5.1      glue_1.3.0       hms_0.4.2
[49] yaml_2.2.0       colorspace_1.3-2 rvest_0.3.2      knitr_1.20
[53] bindr_0.1.1      haven_2.0.0
```

References