Package 'AMPLE'

November 24, 2022

Title Shiny Apps to Support Capacity Building on Harvest Control Rules

Version 1.0.1

Author Finlay Scott [aut, cre] (<https://orcid.org/0000-0001-9950-9023>), Pacific Community (SPC) [cph]

Maintainer Finlay Scott <finlays@spc.int>

Description Three Shiny apps are provided that introduce Harvest Control Rules (HCR) for fisheries management.
'Introduction to HCRs' provides a simple overview to how HCRs work. Users are able to select their own HCR and
step through its performance, year by year. Biological variability and estimation uncertainty are introduced.
'Measuring performance' builds on the previous app and introduces the idea of using performance indicators
to measure HCR performance.
'Comparing performance' allows multiple HCRs to be created and tested, and their performance compared so that the
preferred HCR can be selected.

License GPL (>= 3)

URL https://github.com/PacificCommunity/ofp-sam-ample

Encoding UTF-8

RoxygenNote 7.2.2

Suggests knitr, rmarkdown, bookdown, testthat (>= 3.1.5)

Config/testthat/edition 3

Depends shiny (>= 1.7.3)

Imports markdown, graphics, grDevices, RColorBrewer, stats, R6 (>= 2.5.1), scales (>= 1.2.1), shinyjs (>= 2.1.0), ggplot2 (>= 3.4.0), shinyscreenshot (>= 0.2.0)

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-11-24 03:40:06 UTC

R topics documented:

`AMPLE` *AMPLE: A package of Shiny apps that introduce Harvest Control Rules (HCR) for fisheries management.*

Description

AMPLE provides three Shiny apps that introduce Harvest Control Rules (HCR) for fisheries management. 'Introduction to HCRs' provides a simple overview to how HCRs work. Users are able to select their own HCR and step through its performance, year by year. Biological variability and estimation uncertainty are introduced. 'Introduction to indicators' builds on the previous app and introduces the idea of using performance indicators to evaluate HCR performance. 'Comparing performance' allows multiple HCRs to be created and tested, and their performance compared so that the preferred HCR can be selected.

Harvest Control Rules are used as part of a fishery harvest strategy. This package was put together as part of capacity building efforts by the Pacific Community (SPC) to develop fishery harvest strategies for tuna stocks in the western and central Pacific Ocean (WCPO), working with the Western and Central Pacific Fisheries Commission (WCPFC). For more information on tuna harvest strategies in the WCPO please see LINKs.

AMPLE functions

Too launch the apps use the functions: `intro_hcr()`, `measuring_performance()` and `comparing_performance()`.

Acknowledgement

With thanks to Andre Punt. Also thanks to Winston Chang for help with the R6 class / Shiny reactivity.

assessment *assessment*

Description

Function used by `get_hcr_ip()` to generate input data for an assessment based HCR. The input to the HCR is depletion (i.e. Biomass / K).

Usage

```
assessment(stock, mp_params, yr, iters = 1:dim(stock$biomass)[1])
```

Arguments

`stock`	The stock object
`mp_params`	A named list of MP parameters (with est_sigma and est_bias elements)
`yr`	The timestep that the biomass is taken from.
`iters`	Numeric vector of iters. Default is all of them.

comparing_performance *'Comparing HCR Performance' app launcher*

Description

Launches the Comparing Performance Shiny app. See the 'Information' tab in the app for more information. Also see the package vignette (`vignette("comparing_performance", package="AMPLE")`) for a tutorial.

Usage

```
comparing_performance(...)
```

Arguments

`...`	Not used

Examples

```
## Not run: comparing_performance()
```

constant *Evaluates a constant harvest control rule*

Description

Evaluates a constant harvest control rule, i.e. one that ignores the stock status and just returns the constant level (catch or effort). Used by the hcr_op function.

Usage

```
constant(mp_params, ...)
```

Arguments

mp_params	The HCR / management procedure parameters used to evaluate the HCR (as a list).
...	Unused

estimation_error *estimation_error*

Description

Estimation error applied to the 'true' stock status to generate an 'observed' stock status used in the HCR. The error is a combination of bias and lognormally distributed noise.

Usage

```
estimation_error(input, sigma, bias)
```

Arguments

input	A vector of the 'true' stock status
sigma	Observation error standard deviation
bias	Observation error bias

get_hcr_ip *Get the input to the HCR*

Description

Run the MP analyses function to generate the input to the HCR i.e. observed stock status. For example, estimated biomass from an assessment.

Usage

```
get_hcr_ip(stock, mp_params, yr, ...)
```

Arguments

stock	The stock object
mp_params	The HCR / management procedure parameters used to evaluate the HCR (as a list).
yr	The time step of the true stock status used to generate the HCR IP .
...	Other arguments, including iters

get_hcr_op *Evaluates the harvest control rule.*

Description

Evaluates the harvest control rule in a single year (timestep).

Usage

```
get_hcr_op(stock, mp_params, yr, iters = 1:dim(stock$biomass)[1])
```

Arguments

stock	The stock object
mp_params	The HCR / management procedure parameters used to evaluate the HCR (as a list).
yr	The timestep.
iters	A numeric vector of iters.

Value

A vector of outputs from the HCR.

intro_hcr *Introduction to HCRs app launcher*

Description

Launches the introduction to HCRs Shiny app. See the 'Information' tab in the app for more information. Also see the package vignette (`vignette("intro_hcr", package="AMPLE")`) for a tutorial.

Usage

```
intro_hcr(...)
```

Arguments

`...`	Not used

Examples

```
## Not run: intro_hcr()
```

measuring_performance *Measuring performance app launcher*

Description

Launches the 'Measuring Performance' Shiny app. See the 'Information' tab in the app for more information. Also see the package vignette (`vignette("measuring_performance", package="AMPLE")`) for a tutorial.

Usage

```
measuring_performance(...)
```

Arguments

`...`	Not used

Examples

```
## Not run: measuring_performance()
```

MP modules *mpParamsSetterUI*

Description

The interface for the HCR options. The parameter selection inputs shown in the app are conditional on the selected type of HCR. Some of the inputs have initial values that can be set using the function arguments.

Does the setting part of the MP params module. Returns a list of MP params based on the MP inputs.

Creates the MP params list based on the MP selection from the Shiny UI. Defined outside of a reactive environment above so we can use it non-reactively (helpful for testing).

Usage

```
mpParamsSetterUI(
  id,
  mp_visible = NULL,
  title = "Select the type of HCR you want to test.",
  init_thresh_max_catch = 140,
  init_thresh_belbow = 0.5,
  init_constant_catch = 50,
  init_constant_effort = 1
)

mpParamsSetterServer(id, get_stoch_params = NULL)

mp_params_switcheroo(input, est_sigma = 0, est_bias = 0)
```

Arguments

id	The id (shiny magic)
mp_visible	Which HCR types to show.
title	The title.
init_thresh_max_catch	Initial value of the maximum catch for the catch threshold HCR.
init_thresh_belbow	Initial value of the belbow for the catch threshold HCR.
init_constant_catch	Initial value of constant catch for the constant catch HCR.
init_constant_effort	Initial value of constant effort for the constant effort HCR.
get_stoch_params	Reactive expression that gets the parameters from the stochasticity setter. Otherwise est_sigma and est_bias are set to 0.

input	List of information taken from the Shiny UI (mpParamsSetterUI)
est_sigma	Standard deviation of the estimation variability (default = 0).
est_bias	Estimation bias as a proportion. Can be negative (default = 0).

Value

A taglist

A list of HCR options.

`Stochasticity module` *stochParamsSetterUI*

Description

stochParamsSetterUI() is the UI part for the stochasticity options. Stochasticity is included in the projections in two areas: biological variability (e.g. recruitment) and estimation error (to represent the difference between the 'true' status of the stock and the estimated status that is used by the HCR). Estimation error includes bias and variability. The arguments to this function allow only some of these elements to be shown.

stochParamSetterServer() does the server side stuff for the stochasticity options.

set_stoch_params() sets up default values for the stochasticity parameters. Defined as a separate function so it can be used for testing outside of a reactive environment.

Usage

```
stochParamsSetterUI(
  id,
  show_var = FALSE,
  show_biol_sigma = TRUE,
  show_est_sigma = TRUE,
  show_est_bias = TRUE,
  init_biol_sigma = 0,
  init_est_sigma = 0,
  init_est_bias = 0
)

stochParamsSetterServer(id)

set_stoch_params(input)
```

Arguments

id	The id (shiny magic)
show_var	Show the variability options when app opens (default is FALSE).
show_biol_sigma	Show the biological productivity variability option (default is TRUE).

show_est_sigma	Show the estimation variability option (default is TRUE).
show_est_bias	Show the estimation bias option (default is TRUE).
init_biol_sigma	Default value for biological productivity variability (ignored if not shown).
init_est_sigma	Default value for estimation variability (ignored if not shown).
init_est_bias	Default value for estimation bias (ignored if not shown).
input	A list of stochasticity parameters.

Value

A taglist

A list of stochasticity options.

`Stock` *R6 Class representing a stock*

Description

A stock object has life history parameters, fields and methods for a biomass dynamic model.

Details

A stock has biomass, effort, catch and hcr_ip and hcr_op fields as well as the life history parameters. The population dynamics are a simple biomass dynamic model. The Stock class is used for the Shiny apps in the AMPLE package.

Public fields

`biomass` Array of biomass

`catch` Array of catches

`effort` Array of fishing effort

`hcr_ip` Array of HCR input signals

`hcr_op` Array of HCR output signals

`msy` MSY (default = 100).

`r` Growth rate (default = 0.6). Set by the user in the app.

`k` Carrying capacity (default = NULL - set by msy and r when object is initialised).

`p` Shape of the production curve (default = 1).

`q` Catchability (default = 1).

`lrp` Limit reference point, expressed as depletion (default = 0.2).

`trp` Target reference point, expressed as depletion (default = 0.5).

`b0` Virgin biomass (default = NULL - set by msy and r when object is initialised).

`current_corrnoise` Stores the current values of the correlated noise (by iteration).

`biol_sigma` Standard deviation of biological variability (default = 0).

`last_historical_timestep` The last historical timestep of catch and effort data.

Methods

Public methods:

- Stock$new()
- Stock$reset()
- Stock$reactive()
- Stock$fill_history()
- Stock$fill_catch_history()
- Stock$fill_biomass()
- Stock$as_data_frame()
- Stock$project()
- Stock$relative_cpue()
- Stock$relative_effort()
- Stock$replicate_table()
- Stock$time_periods()
- Stock$performance_indicators()
- Stock$pi_table()
- Stock$clone()

Method `new()`**:** Create a new stock object, with fields of the right dimension and NA values (by calling the `reset()` method. See the `reset()` method for more details.

Usage:

```
Stock$new(stock_params, mp_params, niters = 1)
```

Arguments:

`stock_params` A list of stock parameters with essential elements: r (growth rate, numeric), stock_history (string: "fully", "over", "under") initial_year (integer), last_historical_timestep (integer), nyears (integer), biol_sigma (numeric).

`mp_params` A list of the MP parameters. Used to fill HCR ip and op.

`niters` The number of iters in the stock (default = 1).

Returns: A new Stock object.

Method `reset()`**:** Resets an existing stock object, by remaking all fields (possibly with different dimensions for the array fields) . Fills up the catch, effort and biomass fields in the historical period based on the stock history and life history parameters in the `stock_params` argument. This is a reactive method which invalidates a reactive instance of this class after it is called.

Usage:

```
Stock$reset(stock_params, mp_params, niters)
```

Arguments:

`stock_params` A list with essential elements: r (growth rate, numeric, default=6), stock_history (string: "fully", "over", "under", default="fully") initial_year (integer, default=2000), last_historical_timestep (integer, default=10), nyears (integer, default=30), biol_sigma (numeric, default = 0).

`mp_params` A list of the MP parameters. Used to fill HCR ip and op.

`niters` The number of iters in the stock (default = 1).

Returns: A new Stock object.

Method `reactive()`**:** Method to create a reactive instance of a Stock.

Usage:

```
Stock$reactive()
```

Returns: a reactiveExpr.

Method `fill_history()`**:** Fills the historical period of the stock

Usage:

```
Stock$fill_history(stock_params, mp_params)
```

Arguments:

`stock_params` Named list with last_historical_timestep and stock_history elements.

`mp_params` A list of the MP parameters. Used to fill HCR ip and op.

Method `fill_catch_history()`**:** Fill up the historical period of catches with random values to simulate a catch history

Usage:

```
Stock$fill_catch_history(stock_params)
```

Arguments:

`stock_params` A list with essential elements: r (growth rate, numeric), stock_history (string: "fully", "over", "under") initial_year (integer), last_historical_timestep (integer), nyears (integer).

`stock_history` Character string of the exploitation history (default = "fully", alternatives are "under" or "over").

Method `fill_biomass()`**:** Fills the biomass in the next timestep based on current biomass and catches The surplus production model has the general form: `Bt+1 = Bt + f(Bt) - Ct` Where the production function f() is a Pella & Tomlinson model with shape `f(Bt) = r/p Bt * (1 - (Bt/k)^p)` Here p is fixed at 1 to give a Schaefer model `cpue = Ct / Et = qBt`

Usage:

```
Stock$fill_biomass(ts, iters = 1:dim(self$biomass)[1])
```

Arguments:

`ts` The biomass time step to be filled (required catch etc in ts - 1).

`iters` The iterations to calculate the biomass for (optional - default is all of them).

Method `as_data_frame()`**:** Produces a data.frame of some of the array-based fields, like biomass. Just used for testing purposes.

Usage:

```
Stock$as_data_frame()
```

Method `project()`**:** Projects the stock over the time steps given and updates the biomass, HCR ip / op and catches It uses a simple biomass dynamic model where the catches or fishing effort are set every time step by the harvest control rule.

Usage:

```
Stock$project(timesteps, mp_params, iters = 1:dim(self$biomass)[1])
```

Arguments:

`timesteps` The timesteps to project over. A vector of length 2 (start and end).

`mp_params` A vector of management procedure parameters.

`iters` A vector of iterations to be projected. Default is all the iterations in the stock

Returns: A stock object (a reactiveValues object with bits for the stock)

Method `relative_cpue()`**:** The catch per unit effort (CPUE, or catch rate) relative to the CPUE in the last historical period.

Usage:

```
Stock$relative_cpue()
```

Returns: An array of same dims as the catch and effort fields.

Method `relative_effort()`**:** The effort relative to the effort in the last historical period.

Usage:

```
Stock$relative_effort()
```

Returns: An array of same dims as the effort field.

Method `replicate_table()`**:** Summarises the final year of each iteration. Only used for the Measuring Performance app.

Usage:

```
Stock$replicate_table(iters = 1, quantiles = c(0.05, 0.95))
```

Arguments:

`iters` The iterations to calculate the table values for (default is iteration 1).

`quantiles` Numeric vector of the quantile range. Default values are 0.05 and 0.95.

Method `time_periods()`**:** Calculates the short, medium and long term periods to calculate the performance indicators over, based on the last historic year of data and the number of years in the projection.

Usage:

```
Stock$time_periods()
```

Method `performance_indicators()`**:** Gets the performance indicators across all indicators, for three time periods. Used in the Measuring Performance and Comparing Performance apps.

Usage:

```
Stock$performance_indicators(
  iters = 1:dim(self$biomass)[1],
  quantiles = c(0.05, 0.95)
)
```

Arguments:

`iters` The iterations to calculate the table values for (default is all of them).

`quantiles` Numeric vector of the quantile range. Default values are 0.05 and 0.95.

Returns: A data.frame

Method `pi_table()`**:** Makes a table of the performance indicators.

Usage:

```
Stock$pi_table(iters = 1:dim(self$biomass)[1], quantiles = c(0.05, 0.95))
```

Arguments:

`iters` The iterations to calculate the table values for (default is all of them).

`quantiles` Numeric vector, length 2, of the low and high quantiles.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Stock$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Stock module` *stockParamsSetterUI*

Description

stockParamsSetterUI() is the interface for the stock options (e.g. life history and exploitation status).

stockParamsSetterServer() does the setting of the stock parameters in the server.

get_stock_params() Sets up default values for the stock, including year range. It's a separate function so it can be used and tested outside of a reactive environment.

Usage

```
stockParamsSetterUI(id)

stockParamsSetterServer(id, get_stoch_params = NULL)

get_stock_params(input, biol_sigma = 0)
```

Arguments

`id`	Shiny magic
`get_stoch_params`	Reactive expression that accesses the stochasticity module server.
`input`	List of stock parameters taken from the shiny UI (stockParamsSetterUI()).
`biol_sigma`	Standard deviation of the biological variability (default = 0).

Value

A taglist

A list of stock options.

`threshold` *Evaluates a threshold harvest control rule*

Description

Evaluates a threshold (i.e. hockey stick) harvest control rule. Used by the `hcr_op` function.

Usage

```
threshold(input, mp_params, ...)
```

Arguments

`input`	A vector of the 'true' stock status
`mp_params`	The HCR / management procedure parameters used to evaluate the HCR (as a list).
`...`	Unused

Value

A vector of the same dimension as the input.

Index