Advanced case study options

GMSE: an R package for generalised management strategy evaluation (Supporting Information 4)

A. Bradley Duthie[1,3], Jeremy J. Cusack[1], Isabel L. Jones[1], Jeroen Minderman[1], Erlend B. Nilsen[2], Rocío A. Pozo[1], O. Sarobidy Rakotonarivo[1], Bram Van Moorter[2], and Nils Bunnefeld[1]

[1] Biological and Environmental Sciences, University of Stirling, Stirling, UK [2] Norwegian Institute for Nature Research, Trondheim, Norway [3] alexander.duthie@stir.ac.uk

Fine-tuning simulation conditions using `gmse_apply`

Here we demonstrate how simulations in GMSE can be more fine-tuned to specific empirical situations through the use of `gmse_apply`. To do this, we use the same scenario described in Example case study in GMSE; we first recreate the basic scenario run in `gmse` using `gmse_apply`, and then build in additional modelling details including (1) custom placement of user land, (2) parameterisation of individual user budgets, and (3) density-dependent movement of resources. We emphasise that these simulations are provided only to demonstrate the use of GMSE, and specifically to show the flexibility of the `gmse_apply` function, not to accurately recreate the dynamics of a specific system or make management recommendations.

We reconsider the case of a protected waterfowl population that exploits agricultural land (e.g., Fox and Madsen, 2017; Mason et al., 2017; Tulloch et al., 2017; Cusack et al., 2018). The manager attempts to keep the watefowl at a target abundance, while users (farmers) attempt to maximise agricultural yield on the land that they own. We again parameterise our model using demographic information from the Taiga Bean Goose (*Anser fabalis fabalis*), as reported by Johnson et al. (2018) and AEWA (2016). Relevant parameter values are listed in the table below.

Table 1: GMSE simulation parameter values inspired by Johnson et al. (2018) and AEWA (2016)

Parameter	Value	Description
`remove_pr`	0.122	Goose density-independent mortality probability
`lambda`	0.275	Expected offspring production per time step
`res_death_K`	93870	Goose carrying capacity (on adult mortality)
`RESOURCE_ini`	35000	Initial goose abundance
`manage_target`	70000	Manager's target goose abundance
`res_death_type`	3	Mortality (density and density-independent sources)

Additionally, we continue to use the following values for consistency, except in the case of `stakeholders`, where we reduce the number of farmers to `stakeholders = 8`. This is done to for two reasons. First, it speeds up simulations for the purpose of demonstration; second, it makes the presentation of our custom landscape ownership easier to visualise (see below).

Table 2: Non-default GMSE parameter values chosen by authors

Parameter	Value	Description
`manager_budget`	10000	Manager's budget for setting policy options
`user_budget`	10000	Users' budgets for actions
`public_land`	0.4	Proportion of the landscape that is public
`stakeholders`	8	Number of stakeholders
`land_ownership`	TRUE	Users own landscape cells
`res_consume`	0.02	Landscape cell output consumed by a resource
`observe_type`	3	Observation model type (survey)
`agent_view`	1	Cells managers can see when conducting a survey

All other values are set to GMSE defaults, except where specifically noted otherwise.

Re-creating gmse simulations using gmse_apply

We now recreate the simulations in Example case study in GMSE, which were run using the `gmse` function, in `gmse_apply`. Doing so requires us to first initialise simulations using one call of `gmse_apply`, then loop through multiple time steps that again call `gmse_apply`; results of interest are recorded in a data frame (`sim_sum_1`). Following the protocol introduced in Use of the gmse_apply function, we can call the initialising simulation `sim_old`, and use the code below to read in the relevant parameter values.

```
sim_old  <- gmse_apply(get_res = "Full", remove_pr = 0.122, lambda = 0.275,
                       res_death_K = 93870, RESOURCE_ini = 35000,
                       manage_target = 70000, res_death_type = 3,
                       manager_budget = 10000, user_budget = 100000,
                       public_land = 0.4, stakeholders = 8, res_consume = 0.02,
                       res_birth_K = 200000, land_ownership = TRUE,
                       observe_type = 3, agent_view = 1, converge_crit = 0.01,
                       ga_mingen = 200);
```

Note that the argument `get_res = "Full"` causes `sim_old` to retain all of the relevant data structures for simulating a new time step and recording simulation results. This includes the key simulation output, which is located in `sim_old$basic_output`, which is printed below.

```
## $resource_results
## [1] 34079
##
## $observation_results
## [1] 34079
##
## $manager_results
##          resource_type scaring culling castration feeding help_offspring
## policy_1             1      NA     512         NA      NA             NA
##
## $user_results
##         resource_type scaring culling castration feeding help_offspring
## Manager             1      NA       0         NA      NA             NA
## user_1              1      NA     195         NA      NA             NA
## user_2              1      NA     195         NA      NA             NA
## user_3              1      NA     195         NA      NA             NA
## user_4              1      NA     195         NA      NA             NA
```

```
## user_5              1       NA     195          NA       NA             NA
## user_6              1       NA     195          NA       NA             NA
## user_7              1       NA     195          NA       NA             NA
## user_8              1       NA     195          NA       NA             NA
##         tend_crops kill_crops
## Manager         NA         NA
## user_1          NA         NA
## user_2          NA         NA
## user_3          NA         NA
## user_4          NA         NA
## user_5          NA         NA
## user_6          NA         NA
## user_7          NA         NA
## user_8          NA         NA
```

We can then loop over 30 time steps to recreate the simulations from Example case study in GMSE. In these simulations, we are specifically interested in the resource and observation outputs, as well as the manager policy and user actions for culling, which we record below in the data frame `sim_sum_1`. The inclusion of the argument `old_list` tells `gmse_apply` to use parameters and values from the list `sim_old` in the new time step.

```
sim_sum_1 <- matrix(data = NA, nrow = 30, ncol = 5);
for(time_step in 1:30){
    sim_new                 <- gmse_apply(get_res = "Full", old_list = sim_old);
    sim_sum_1[time_step, 1] <- time_step;
    sim_sum_1[time_step, 2] <- sim_new$basic_output$resource_results[1];
    sim_sum_1[time_step, 3] <- sim_new$basic_output$observation_results[1];
    sim_sum_1[time_step, 4] <- sim_new$basic_output$manager_results[3];
    sim_sum_1[time_step, 5] <- sum(sim_new$basic_output$user_results[,3]);
    sim_old                 <- sim_new;
}
colnames(sim_sum_1) <- c("Time", "Pop_size", "Pop_est", "Cull_cost",
                         "Cull_count");
print(sim_sum_1);
```

```
##       Time Pop_size Pop_est Cull_cost Cull_count
##  [1,]    1    32207   32207      1010        792
##  [2,]    2    31912   31912      1010        791
##  [3,]    3    32145   32145      1010        792
##  [4,]    4    32892   32892      1010        792
##  [5,]    5    37100   37100      1010        791
##  [6,]    6    38135   38135      1010        792
##  [7,]    7    39494   39494      1009        792
##  [8,]    8    40993   40993      1010        791
##  [9,]    9    43135   43135      1010        792
## [10,]   10    45408   45408      1009        792
## [11,]   11    48090   48090      1010        792
## [12,]   12    50401   50401      1010        792
## [13,]   13    53055   53055      1009        791
## [14,]   14    55973   55973      1010        792
## [15,]   15    58985   58985      1010        792
## [16,]   16    62366   62366      1010        791
## [17,]   17    66267   66267      1010        792
## [18,]   18    69840   69840      1009        792
## [19,]   19    73995   73995       230       3472
```

```
## [20,]   20   75220   75220         176         4544
## [21,]   21   75816   75816         158         5056
## [22,]   22   75563   75563         165         4848
## [23,]   23   75411   75411         170         4704
## [24,]   24   75604   75604         164         4872
## [25,]   25   75601   75601         164         4872
## [26,]   26   75939   75939         154         5192
## [27,]   27   75718   75718         160         5000
## [28,]   28   75590   75590         164         4872
## [29,]   29   75525   75525         166         4816
## [30,]   30   75470   75470         168         4760
```

The above output from `sim_sum_1` shows the data frame that holds the information we were interested in pulling out of our simulation results. All of this information was available under the list element `sim_new$basic_output`, but other list elements of `sim_new` might also be useful to record. It is important to remember that this example of `gmse_apply` is using the default resource, observation, manager, and user sub-models. Custom sub-models could produce different outputs in `sim_new` (see Use of the gmse_apply function for examples). For default sub-models, there are some list elements that might be especially useful. These elements can potentially be edited *within the above loop* to dynamically adjust simulations. For more explanation of built-in GMSE data arrays, see Default GMSE data structures.

- `sim_new$resource_array`: A table holding all information on resources. Rows correspond to discrete resources, and columns correspond to resource properties: (1) ID, (2-4) types (not currently in use), (5) x-location, (6) y-location, (7) movement parameter, (8) time, (9) density independent mortality parameter (`remove_pr`), (10) reproduction parameter (`lambda`), (11) offspring number, (12) age, (13-14) observation columns, (15) consumption rate (`res_consume`), (16-20) recorded experiences of user actions (e.g., was the resource culled or scared?), (21) how much yield has the resource consumed, and (22) how many times the resource can consume yield in one time step.
- `sim_new$AGENTS`: A table holding basic information on agents (manager and users). Rows correspond to a unique agent, and columns correspond to agent properties: (1) ID, (2) type (0 for the manager, 1 for users), (3-4) additional type options not currently in use, (5-6), x and y locations (usually ignored), (7) movement parameter (usually ignored), (8) time, (9) agent's viewing ability in cells (`agent_view`), (10) error parameter, (11-12) values for holding marks and tallies of resources, (13-15) values for holding observations, (16) yield from landscape cells, (17) baseline budget (`manager_budget` and `user_budget`), (18-24) agent's perception of the efficacy of scaring, culling, castrating, feeding, helping, tending crops, and killing crops, (25-26) increments to budget, (27) unused.
- `sim_new$observation_vector`: Estimate of total resource number from the observation model (`observation_array` also holds this information in a different way depending on `observe_type`)
- `sim_new$LAND`: The landscape on which interactions occur, which is stored as a 3D array with `land_dim_1` rows, `land_dim_2` columns, and 3 layers. Layer 1 (`sim_new$LAND[,,1]`) is not currently used in default sub-models, but could be used to store values that affect resources and agents. Layer 2 (`sim_new$LAND[,,2]`) stores crop yield from a cell, and layer 3 (`sim_new$LAND[,,3]`) stores the owner of the cell (value corresponds to the agent's ID).
- `sim_new$manage_vector`: The cost of each action as set by the manager. For even more fine-tuning, individual costs for the actions of each agent can be set for each user in `sim_new$manager_array`.
- `sim_new$user_vector`: The total number of actions performed by each user. A more detailed breakdown of actions by individual users is held in `sim_new$user_array`.

Next, we show how to adjust the landscape to manually set land ownership in `gmse_apply`.

1. Custom placement of user land

By default, all farmers in GMSE are allocated roughly the same number of landscape cells, which are placed on the landscape using a shortest-splitline algorithm that makes similar size rectangles. In the `LAND` array, ownership is designated by the agent's ID. Public land is produced by placing landscape cells that are technically owned by the manager, and therefore have landscape cell values of 1. The image below shows this landscape for the eight farmers from `sim_old`.

```
image(x = sim_old$LAND[,,3], col = topo.colors(9), xaxt = "n", yaxt = "n");
```

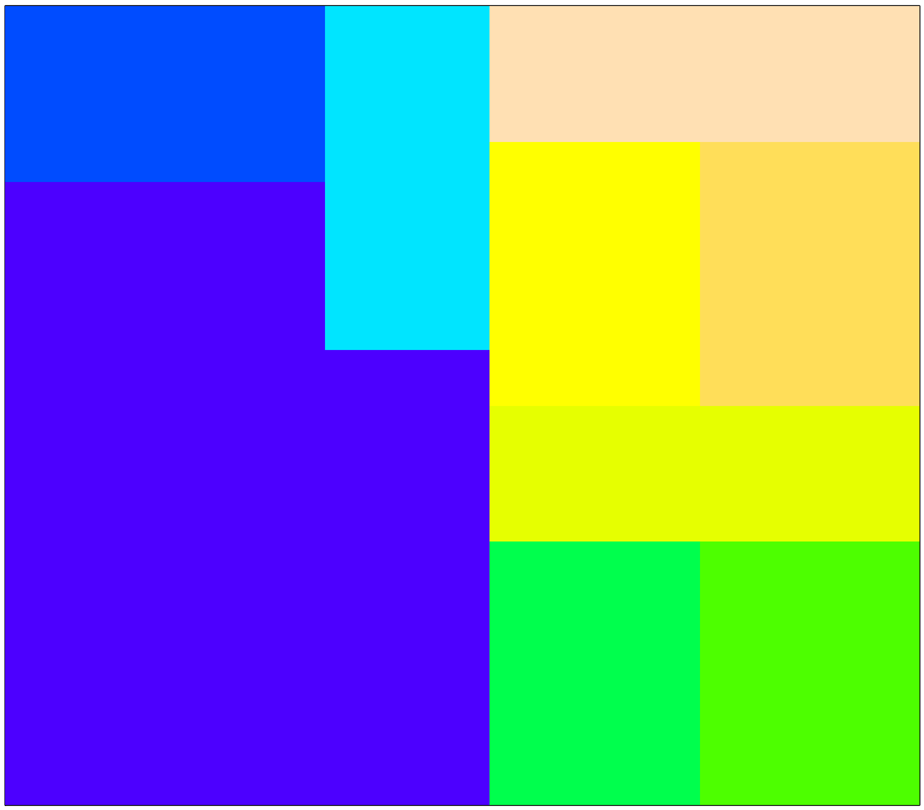

Figure 1: Default position of land ownership by farmers.

We can change the ownership of cells by manipulating `sim_old$LAND[,,3]`. First we initialise a new `sim_old` below.

```
sim_old  <- gmse_apply(get_res = "Full", remove_pr = 0.122, lambda = 0.275,
                       res_death_K = 93870, RESOURCE_ini = 35000,
                       manage_target = 70000, res_death_type = 3,
                       manager_budget = 10000, user_budget = 10000,
                       public_land = 0.4, stakeholders = 8, res_consume = 0.02,
                       res_birth_K = 200000, land_ownership = TRUE,
                       observe_type = 3, agent_view = 1, converge_crit = 0.01,
```

```
                         ga_mingen = 200);
```

Because we have not specified landscape dimensions in the above, the landscape reverts to the default size of 100 by 100 cells. We can then manually assign landscape cells to the eight farmers, whose IDs range from 2-9 (ID value 1 is the manager). Below we do this to make eight different sized farms.

```
sim_old$LAND[1:20,   1:20,  3] <- 2;
sim_old$LAND[1:20,  21:40,  3] <- 3;
sim_old$LAND[1:20,  41:60,  3] <- 4;
sim_old$LAND[1:20,  61:80,  3] <- 5;
sim_old$LAND[1:20,  81:100, 3] <- 6;
sim_old$LAND[21:40,  1:50,  3] <- 7;
sim_old$LAND[21:40, 51:100, 3] <- 8;
sim_old$LAND[41:60,  1:100, 3] <- 9;
sim_old$LAND[61:100, 1:100, 3] <- 1; # Public land
image(x = sim_old$LAND[,,3], col = topo.colors(9), xaxt = "n", yaxt = "n");
```

The above image shows the modified landscape stored in `sim_old`, which can now be incorporated into simulations using `gmse_apply`. We can think of all the plots on the left side of the landscape as farms of various sizes, while the blue area of the landscape on the right is public land.

2. Parameterisation of individual user budgets

Perhaps we want to assume that farmers have different baseline budgets, which are correlated in some way to the number of landscape cells that they own. Custom user baseline budgets can be set by manipulating `sim_old$AGENTS`, column 17 of which holds the budget for each user. Agent IDs (as stored on the landscape above) correspond to rows of `sim_old$AGENTS`, so individual baseline budgets can be directly input as desired. We can do this manually (e.g., `sim_old$AGENTS[2, 17] <- 4000`), or, alternatively, if farmer budget positively correlates to landscape owned, we can use a loop to input values as below.

```
for(ID in 2:9){
    cells_owned            <- sum(sim_old$LAND[,,3] == ID);
    sim_old$AGENTS[ID, 17] <- 10 * cells_owned;
}
```

The number of cells owned by the manager (1) and each farmer (2-8) is therefore listed in the table below.

ID	1	2	3	4	5	6	7	8	9
Budget	10000	4000	4000	4000	4000	4000	10000	10000	20000

As with `sim_old$LAND` values, changes to `sim_old$AGENTS` will be retained in simulations looped through `gmse_apply`.

3. Density-dependent movement of resources

Lastly, we consider a more nuanced change to simulations, in which the rules for movement of resources are modified to account for density-dependence. Assume that geese tend to avoid aggregating, such that if a goose is located on the same cell as too many other geese, then it will move at the start of a time step. Programming this movement rule can be accomplished by creating a new function to apply to the resource data array `sim_old$resource_array`. Below, a custom function is defined that causes a goose to move up to 5 cells in any direction if it finds itself on a cell with more than 10 other geese. As with default GMSE

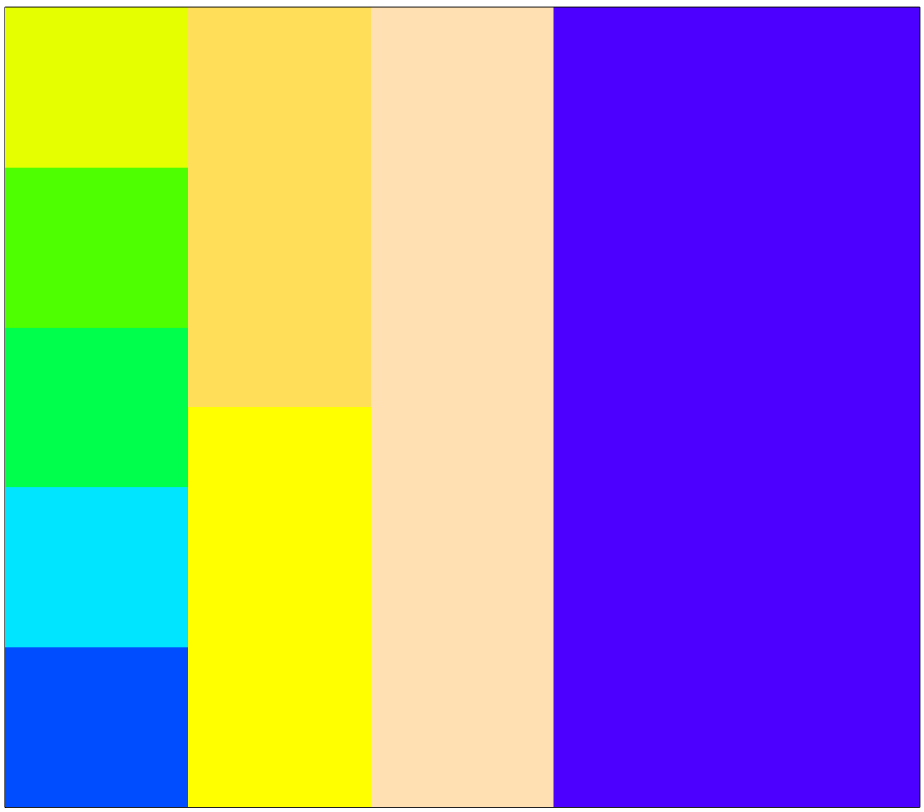

Figure 2: Land ownership by farmers as customised in gmse_apply.

simulations, movement is based on a torus landscape (where no landscape edge exists, so that if resources move off of one side of the landscape they appear on the opposite side). We will use this custom function to modify `sim_old$resource_array` prior to running `gmse_apply`, thereby modelling a custom-built process affecting resource distribution that is integrated into GMSE.

```
avoid_aggregation <- function(sim_resource_array, land_dim_1 = 100,
                              land_dim_2 = 100){
    goose_number <- dim(sim_resource_array)[1]  # How many geese are there?
    for(goose in 1:goose_number){               # Loop through all rows of geese
        x_loc  <- sim_resource_array[goose, 5];
        y_loc  <- sim_resource_array[goose, 6];
        shared <- sum( sim_resource_array[,5] == x_loc &
                       sim_resource_array[,6] == y_loc);
        if(shared > 10){
            new_x <- x_loc + sample(x = -5:5, size = 1);
            new_y <- y_loc + sample(x = -5:5, size = 1);
            if(new_x < 0){ # The 'if' statements below apply the torus
                new_x <- land_dim_1 + new_x;
            }
            if(new_x >= land_dim_1){
                new_x <- new_x - land_dim_1;
            }
            if(new_y < 0){
                new_y <- land_dim_2 + new_x;
            }
            if(new_y >= land_dim_2){
                new_y <- new_y - land_dim_2;
            }
            sim_resource_array[goose, 5] <- new_x;
            sim_resource_array[goose, 6] <- new_y;
        }
    }
    return(sim_resource_array);
}
```

With the above function written, we can apply the new movement rule along with our custom farm placement and custom farmer budgets to the simulation of goose population dynamics.

Simulation with custom farms, budgets, and goose movement

Below shows an example of `gmse_apply` with custom landscapes, farmer budgets, and density-dependent goose movement rules.

```
# First initialise a simulation
sim_old  <- gmse_apply(get_res = "Full", remove_pr = 0.122, lambda = 0.275,
                       res_death_K = 93870, RESOURCE_ini = 35000,
                       manage_target = 70000, res_death_type = 3,
                       manager_budget = 10000, user_budget = 10000,
                       public_land = 0.4, stakeholders = 8, res_consume = 0.02,
                       res_birth_K = 200000, land_ownership = TRUE,
                       observe_type = 3, agent_view = 1, converge_crit = 0.01,
                       ga_mingen = 200, res_move_type = 0);
# By setting `res_move_type = 0`, no resource movement will occur in gmse_apply
# Adjust the landscape ownership below
```

```
sim_old$LAND[1:20,   1:20,  3] <- 2;
sim_old$LAND[1:20,  21:40,  3] <- 3;
sim_old$LAND[1:20,  41:60,  3] <- 4;
sim_old$LAND[1:20,  61:80,  3] <- 5;
sim_old$LAND[1:20,  81:100, 3] <- 6;
sim_old$LAND[21:40,  1:50,  3] <- 7;
sim_old$LAND[21:40, 51:100, 3] <- 8;
sim_old$LAND[41:60,  1:100, 3] <- 9;
sim_old$LAND[61:100, 1:100, 3] <- 1;
# Change the budgets of each farmer based on the land they own
for(ID in 2:9){
    cells_owned            <- sum(sim_old$LAND[,,3] == ID);
    sim_old$AGENTS[ID, 17] <- 10 * cells_owned;
}
# Begin simulating time steps for the system
sim_sum_2 <- matrix(data = NA, nrow = 30, ncol = 5);
for(time_step in 1:30){
    # Apply the new movement rules at the beginning of the loop
    sim_old$resource_array  <- avoid_aggregation(sim_resource_array =
                                                 sim_old$resource_array);
    # Next, move on to simulate (old_list remembers that res_move_type = 0)
    sim_new                 <- gmse_apply(get_res = "Full", old_list = sim_old);
    sim_sum_2[time_step, 1] <- time_step;
    sim_sum_2[time_step, 2] <- sim_new$basic_output$resource_results[1];
    sim_sum_2[time_step, 3] <- sim_new$basic_output$observation_results[1];
    sim_sum_2[time_step, 4] <- sim_new$basic_output$manager_results[3];
    sim_sum_2[time_step, 5] <- sum(sim_new$basic_output$user_results[,3]);
    sim_old                 <- sim_new;
}
colnames(sim_sum_2) <- c("Time", "Pop_size", "Pop_est", "Cull_cost",
                         "Cull_count");
print(sim_sum_2);
```

```
##       Time Pop_size Pop_est Cull_cost Cull_count
##  [1,]    1    34284   34284      1007         52
##  [2,]    2    34828   34828      1010         52
##  [3,]    3    36104   36104      1001         52
##  [4,]    4    38119   38119      1009         52
##  [5,]    5    44011   44011      1010         52
##  [6,]    6    46361   46361       999         60
##  [7,]    7    48979   48979      1006         52
##  [8,]    8    52152   52152      1009         52
##  [9,]    9    55500   55500      1010         52
## [10,]   10    59165   59165      1001         52
## [11,]   11    62982   62982      1004         52
## [12,]   12    66878   66878      1010         52
## [13,]   13    71197   71197        51       1174
## [14,]   14    74990   74990        14       4105
## [15,]   15    75766   75766        11       5017
## [16,]   16    75640   75640        11       5030
## [17,]   17    75467   75467        12       4626
## [18,]   18    75785   75785        11       4970
## [19,]   19    75867   75867        11       5079
## [20,]   20    75534   75534        12       4687
```

```
## [21,]   21   75560   75560        12        4709
## [22,]   22   75494   75494        11        4973
## [23,]   23   75392   75392        12        4688
## [24,]   24   75366   75366        12        4709
## [25,]   25   75425   75425        12        4668
## [26,]   26   75246   75246        12        4696
## [27,]   27   75038   75038        13        4358
## [28,]   28   75310   75310        13        4415
## [29,]   29   75835   75835        11        5025
## [30,]   30   75686   75686        11        5035
```

Conclusions

In this example, we showed how the built-in resource, observation, manager, and user sub-models can be customised by manipulating the data within the data structures that they use. The goal was to show how software users can work with these existing sub-models and data structures to customise GMSE simulations. Readers seeking even greater flexibility (e.g., replacing an entire built-in sub-model with a custom sub-model) should refer to Use of the gmse_apply function that introduces `gmse_apply` more generally. Future versions of GMSE are likely to expand on the built-in options available for simulation; requests for such expansions, or contributions, can be submitted to GitHub.

References

AEWA (2016). International single species action plan for the conservation of the Taiga Bean Goose (Anser fabalis fabalis).

Cusack, J. J., Duthie, A. B., Rakotonarivo, S., Pozo, R. A., Mason, T. H. E., Månsson, J., Nilsson, L., Tombre, I. M., Eythórsson, E., Madsen, J., Tulloch, A., Hearn, R. D., Redpath, S., and Bunnefeld, N. (2018). Time series analysis reveals synchrony and asynchrony between conflict management effort and increasing large grazing bird populations in northern Europe. *Conservation Letters*, page e12450.

Fox, A. D. and Madsen, J. (2017). Threatened species to super-abundance: The unexpected international implications of successful goose conservation. *Ambio*, 46(s2):179–187.

Johnson, F. A., Alhainen, M., Fox, A. D., Madsen, J., and Guillemain, M. (2018). Making do with less: Must sparse data preclude informed harvest strategies for European waterbirds. *Ecological Applications*, 28(2):427–441.

Mason, T. H., Keane, A., Redpath, S. M., and Bunnefeld, N. (2017). The changing environment of conservation conflict: geese and farming in Scotland. *Journal of Applied Ecology*, pages 1–12.

Tulloch, A. I. T., Nicol, S., and Bunnefeld, N. (2017). Quantifying the expected value of uncertain management choices for over-abundant Greylag Geese. *Biological Conservation*, 214:147–155.