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 Supporting Information 3; 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 landscape ownership easier (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

Parameter	Value	Description
`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 Supporting Information 3, 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 Supporting Information 2, 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` 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] 34281
##
## $observation_results
## [1] 34281
##
## $manager_results
##          resource_type scaring culling castration feeding help_offspring
## policy_1             1      NA     515         NA      NA             NA
##
## $user_results
##         resource_type scaring culling castration feeding help_offspring
## Manager             1      NA       0         NA      NA             NA
## user_1              1      NA     189         NA      NA             NA
## user_2              1      NA     188         NA      NA             NA
## user_3              1      NA     189         NA      NA             NA
## user_4              1      NA     188         NA      NA             NA
## user_5              1      NA     189         NA      NA             NA
## user_6              1      NA     189         NA      NA             NA
## user_7              1      NA     188         NA      NA             NA
## user_8              1      NA     188         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 Supporting Information 3. 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`. 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    32690   32690       842        930
##  [2,]    2    32119   32119       957        818
##  [3,]    3    32242   32242       985        796
##  [4,]    4    33087   33087      1001        785
##  [5,]    5    37318   37318       989        794
##  [6,]    6    38590   38590      1006        782
##  [7,]    7    40054   40054      1000        786
##  [8,]    8    41853   41853       995        790
##  [9,]    9    43863   43863      1003        785
## [10,]   10    45793   45793       999        786
## [11,]   11    48318   48318       986        797
## [12,]   12    50664   50664       995        791
## [13,]   13    53132   53132       995        789
## [14,]   14    55813   55813      1002        785
## [15,]   15    58850   58850       991        793
## [16,]   16    62289   62289       997        786
## [17,]   17    65855   65855       996        788
## [18,]   18    69412   69412       998        786
## [19,]   19    73371   73371        10      29268
## [20,]   20    47258   47258      1010        778
## [21,]   21    49768   49768       999        786
## [22,]   22    52332   52332       993        793
## [23,]   23    55254   55254      1004        785
## [24,]   24    58236   58236      1002        785
```

```
## [25,]   25    61620   61620        996         787
## [26,]   26    64819   64819       1001         785
## [27,]   27    68333   68333        997         787
## [28,]   28    71910   71910         10       29205
## [29,]   29    45677   45677       1010         778
## [30,]   30    48148   48148        995         788
```

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 submodels. Custom submodels could produce different outputs in `sim_new` (see Supporting Information 2 for examples). For default submodels, there are some list elements that might be especially useful. These elements can potentially be edited *within the above loop* to dynamically adjust simulations.

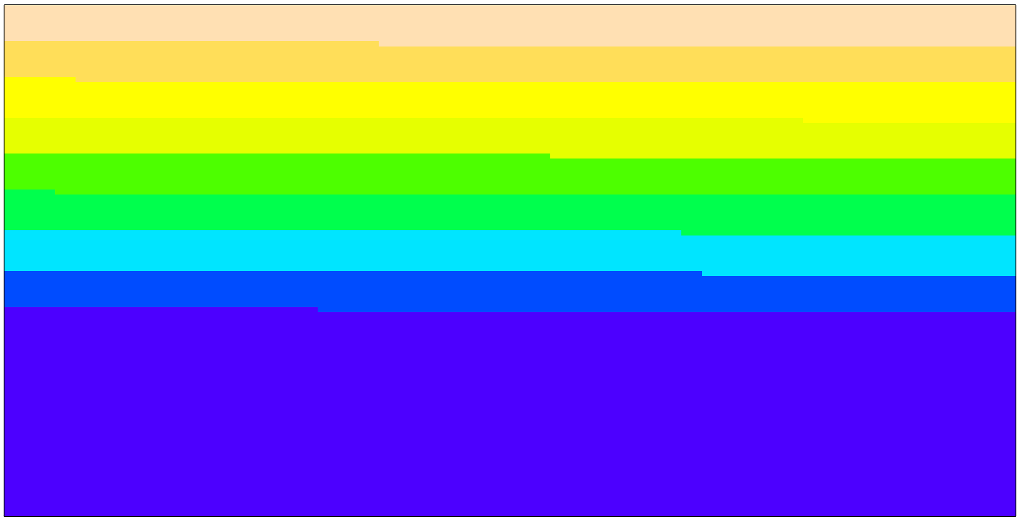
- `sim_new$resource_array`: A table holding all information on resources. Rows correspond to discrete resources, and columns correspond to resource properties: (1) ID, (2-4) types (not currently in use), (5) x-location, (6) y-location, (7) movement parameter, (8) time, (9) density independent mortality parameter (`remove_pr`), (10) reproduction parameter (`lambda`), (11) offspring number, (12) age, (13-14) observation columns, (15) consumption rate (`res_consume`), and (16-20) recorded experiences of user actions (e.g., was the resource culled or scared?).
- `sim_new$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) budget (`manager_budget` and `user_budget`).
- `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$LAND[,,1]`) is not currently used in default submodels, but could be used to store values that affect resources and agents. Layer 2 (`sim_new$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 the same number of landscape cells, which are simply placed in order of the farmer'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`.

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

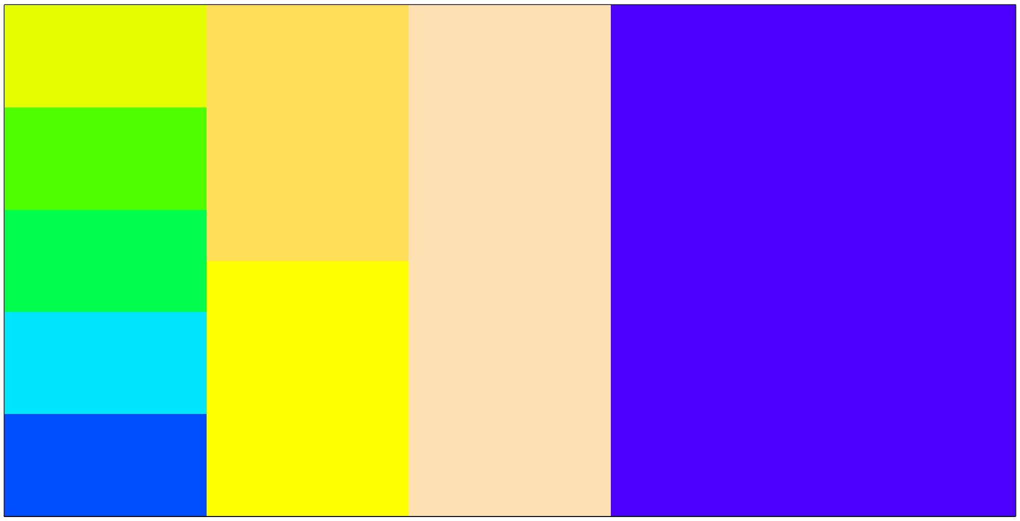


We can change the ownership of cells by manipulating `sim_old$LAND[,,3]`. First we initialise a new `sim_old` below.

```r
sim_old  <- gmse_apply(get_res = "Full", remove_pr = 0.122, lambda = 0.275,
                       res_d4eath_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.

```r
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 budgets, which are correlated in some way to the number of landscape cells that they own. Custom user budgets can be set by manipulating `sim_old$AGENTS`, the last column of which (column 17) holds the budget for each user. Agent IDs (as stored on the landscape above) correspond to rows of `sim_old$AGENTS`, so individual 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] <- 100 * cells_owned;
}
```

The number of cells owned by each farmer is therefore listed in the table below.

ID	1	2	3	4	5	6	7	8	9
Budget	10000	40000	40000	40000	40000	40000	1e+05	1e+05	2e+05

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 simulations, movement is based on a torus landscape.

```
avoid_aggregation <- function(goose_table, land_dim_1 = 100, land_dim_2 = 100){
    goose_number <- dim(goose_table)[1]    # How many geese are there?
    for(goose in 1:goose_number){          # Loop through all rows of geese
        x_loc  <- goose_table[goose, 5];
        y_loc  <- goose_table[goose, 6];
        shared <- sum(goose_table[,5] == x_loc & goose_table[,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;
            }
            goose_table[goose, 5] <- new_x;
            goose_table[goose, 6] <- new_y;
        }
    }
    return(goose_table);
}
```

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;
```

```r
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_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    33540   33540       803         68
##  [2,]    2    33850   33850       907         61
##  [3,]    3    34721   34721       963         60
##  [4,]    4    36212   36212       982         60
##  [5,]    5    41647   41647      1010         52
##  [6,]    6    43663   43663       996         57
##  [7,]    7    46223   46223       976         60
##  [8,]    8    49013   49013       987         60
##  [9,]    9    52160   52160       995         57
## [10,]   10    55547   55547       989         60
## [11,]   11    59070   59070      1003         52
## [12,]   12    62683   62683       994         58
## [13,]   13    66605   66605       983         60
## [14,]   14    70734   70734       455        124
## [15,]   15    75227   75227       398        149
## [16,]   16    80078   80078       343        170
## [17,]   17    85126   85126       399        149
## [18,]   18    90496   90496       349        167
## [19,]   19    96170   96170       389        150
## [20,]   20   100865  100865       421        138
## [21,]   21   102243  102243       415        139
## [22,]   22   102901  102901       392        150
## [23,]   23   102708  102708       397        146
## [24,]   24   103526  103526       384        153
## [25,]   25   103785  103785       349        168
```

```
## [26,]   26   103845  103845        385         151
## [27,]   27   103778  103778        415         139
## [28,]   28   103534  103534        408         141
## [29,]   29   103557  103557        436         134
## [30,]   30   103769  103769        424         137
```

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. Software users seeking even greater flexibility (e.g., replacing an entire built-in submodel with a custom submodel) should refer to the Supporting Information 2 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.