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], 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-indepent 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` and increase `user_budget` to 100000. 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`	100000	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 run of `gmse_apply`, then loop through multiple time steps that again call `gmse_apply` and saving the results of interest. Following instructions in Supporting Information 1, 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,
                       land_ownership = TRUE, res_consume = 0.02,
                       observe_type = 3, agent_view = 1);
```

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] 34493
##
## $observation_results
## [1] 34493
##
## $manager_results
##          resource_type scaring culling castration feeding help_offspring
## policy_1             1      NA     469         NA      NA             NA
##
## $user_results
##         resource_type scaring culling castration feeding help_offspring
## Manager             1      NA       0         NA      NA             NA
## user_1              1      NA     206         NA      NA             NA
## user_2              1      NA     205         NA      NA             NA
## user_3              1      NA     206         NA      NA             NA
## user_4              1      NA     206         NA      NA             NA
## user_5              1      NA     206         NA      NA             NA
## user_6              1      NA     206         NA      NA             NA
## user_7              1      NA     206         NA      NA             NA
## user_8              1      NA     205         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.

```r
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    32672   32672       469       1645
##  [2,]    2    31366   31366       458       1683
##  [3,]    3    30512   30512       450       1712
##  [4,]    4    30404   30404       462       1671
##  [5,]    5    32994   32994       457       1684
##  [6,]    6    32653   32653       444       1737
##  [7,]    7    32573   32573       457       1688
##  [8,]    8    33014   33014       455       1697
##  [9,]    9    33476   33476       463       1666
## [10,]   10    33935   33935       447       1726
## [11,]   11    34534   34534       447       1725
## [12,]   12    34942   34942       438       1757
## [13,]   13    35608   35608       473       1633
## [14,]   14    36447   36447       460       1676
## [15,]   15    37484   37484       447       1725
## [16,]   16    38221   38221       472       1637
## [17,]   17    38833   38833       446       1728
## [18,]   18    39379   39379       456       1691
## [19,]   19    39900   39900       463       1666
## [20,]   20    40118   40118       456       1692
## [21,]   21    40367   40367       620       1255
## [22,]   22    40907   40907       468       1648
## [23,]   23    40849   40849       448       1720
## [24,]   24    40834   40834       450       1712
## [25,]   25    40671   40671       454       1699
```

```
## [26,]   26    40547   40547        445        1732
## [27,]   27    40547   40547        451        1712
## [28,]   28    40721   40721        456        1691
## [29,]   29    40682   40682        467        1651
## [30,]   30    40825   40825        445        1731
```

The above output from `sim_sum_1` shows the table 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 also 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 1 for examples). For default options, there are some list elements that might be especially useful. All of these elements can be edited *within the above loop* to dynamically adjust simulations.

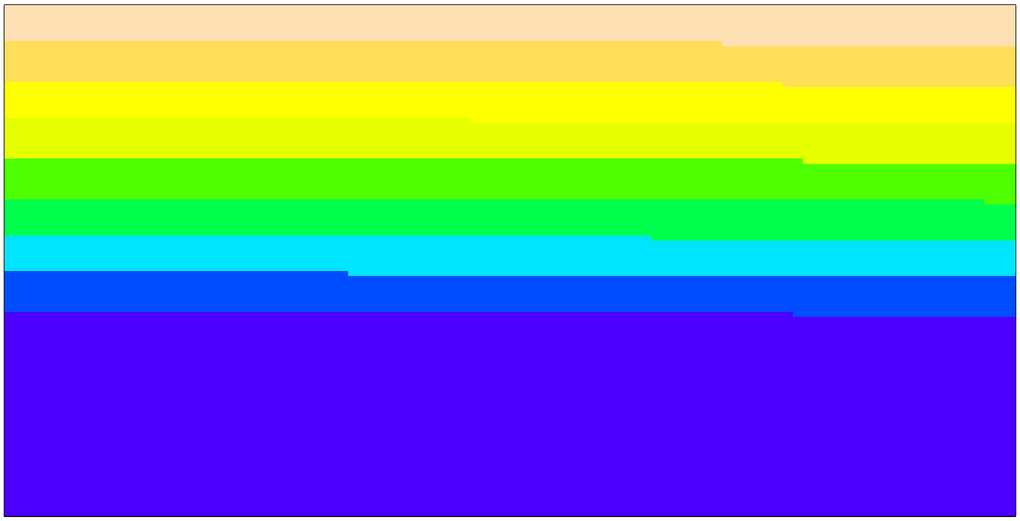
- `sim_new$resource_array`: A table holding all information on resources. Rows correspond to a unique resource, and columns correspond to resource properties: (1) ID, (2-4) resource types (not currently in use), (5) x-location, (6) y-location, (7) movement parameter, (8) time, (9) density independent mortality parameter (`remove_pr`), (10) reproduction parameter (`lambda`), (11) offspring number, (12) age, (13-14) observation columns, (15) consumption rate (`res_consume`), and (16-20) recorded experiences of user actions.
- `sim_new$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$LAND[,,1]`) is not used, but could be used to store values that affect resources and agents. Layer 2 (`sim_new$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 actions 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`.

We now show how to adjust the landscape to manually adjust 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 landscape is produced by placing landscape cells that are technically owned by the manager, and therefore have landscape cell values of 0. 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");
```

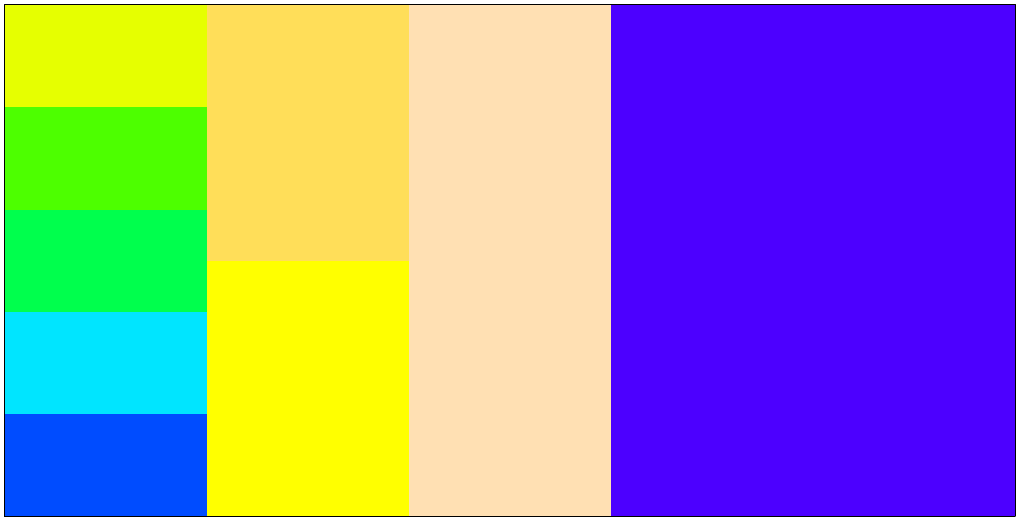


We can change the ownership of cells by manipulating `sim_old$LAND[,,3]`. First we initialise a new `sim_old` below.

```
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 = 100000,
                       public_land = 0.4, stakeholders = 8,
                       land_ownership = TRUE, res_consume = 0.02,
                       observe_type = 3, agent_view = 1);
```

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`.

2. Parameterisation of individual user budgets

Perhaps we want to assume that farmers have different budgets, which are perhaps correlated in some way to the number of landscape cells that they own. Custom user budgets can be set by manipulating `sim_old$AGENTS`, the last column of which holds the budget for each user. Agent IDs (as stored on the landscape above) correspond to rows of `sim_old$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.

```
##         [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
## ID         1     2     3     4     5     6 7e+00 8e+00 9e+00
## Budget 10000 40000 40000 40000 40000 40000 1e+05 1e+05 2e+05
```

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.