resevol: an R package for spatially explicit models of pesticide resistance given evolving pest genomes

A. Bradley Duthie[12], et al.

[1] Biological and Environmental Sciences, University of Stirling, Stirling, UK [2] alexander.duthie@stir.ac.uk, Biological and Environmental Sciences 3A149 University of Stirling Stirling, FK9 4LA, UK

Abstract

1. Pesticide resistance is a serious problem.
2. We introduce the resevol R package, which runs individual-based models of pests with quantitative and covarying traits and three mating systems.
3. Simulations are on a landscape where crops and pesticides are changed.
4. We give an example of simulating pests with covarying traits
5. The resevol R package is open source under GNU Public License; source code and documents are freely available on GitHub.

Key words: *pest management, food security, ecological modelling*

1 Introduction

- Pest resistance to pesticide is a serious wicked problem that affects food security (I have some references for this, but am probably missing several).
- A quantitative genetic approach to pesticide resistance to maintaining resistance (has this been done at all?).

The resevol package uses individual-based modelling and a quantitative genetics approach to simulate the evolution of a pest population on a changing landscape. A focal goal of the software is to model traits with a pre-specified, but potentially evolving, covariance structure. To achieve this goal, each individual has a genome with L loci that underlie a set of T potentially evolving traits. Pleiotropic loci can vary in their effects on the direction and magnitude of polygenic trait, causing population-wide trait covariance to arise mechanistically from the underlying genetic architecture of individuals. To achieve this, two separate steps are necessary. First, a genetic algorithm is used to find a network of internal nodes that map standard random normal loci values to covarying traits. Values used to map loci to traits are incorporated into individual genomes. Second, a population of asexual or sexual individuals is initialised and simulated on a spatially explicit landscape separated into distinct units (e.g., farms). Land units can apply one of up to 10 pesticides, and one of up to 10 crops; pesticides and crops rotate independently in a pre-specified way over time. The resevol package can thereby model complex and evolving traits in agricultural pests over realistic landscapes that undergo different pesticide use and crop regimes.

2 Covarying pest quantitative traits

The first step of simulation with the resevol package is building individual genomes. This step is separate because building genomes is often time-consuming, and genomes that are built might need to be saved or inspected before actual simulation. High computation time is due to the mechanistic nature of how genomes and covarying traits are modelled. Instead of imposing a trait covariance structure directly, the objective is to use an evolutionary algorithm to find a network that maps standard random normal values (loci) to covarying values (traits). This is useful because it allows genomes to model potentially evolving physiological constraints, and trade-offs among traits, from the bottom up. Since multiple networks can potentially map loci to the same trait covariance structure, it is possible to replicate evolution with differently randomised genetic architectures. This approach to modelling individual genomes and traits thereby increases the complexity of questions that can be addressed for simulating evolution in agricultural pests.
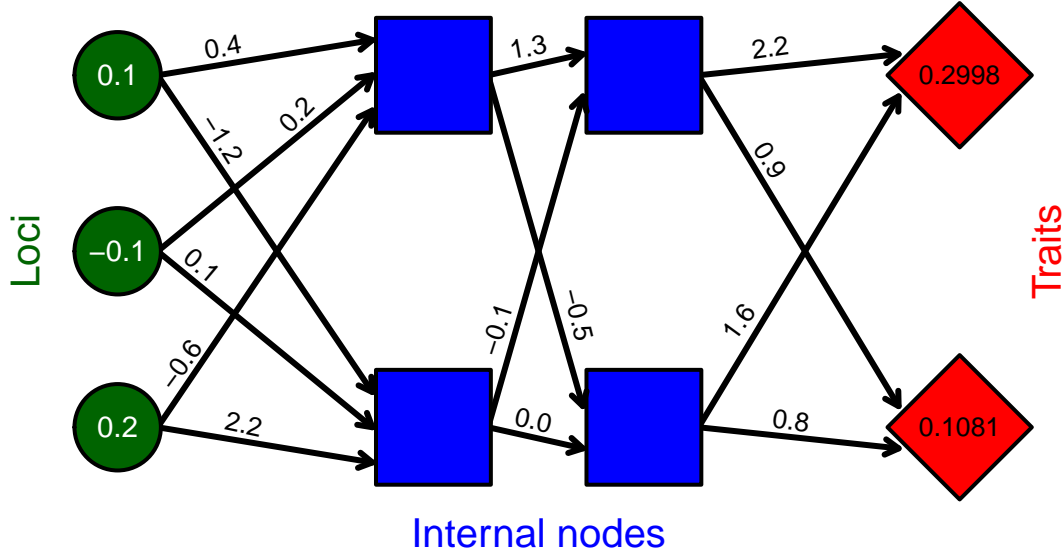

Figure 1: Example network mapping loci (green circles) to traits (red diamonds) through an intermediate set of hidden layers (blue squares) in the mine_gmatrix function. Individual genomes in the resevol R package consist of standard random normal values for at loci, real values for black arrows linking nodes, and real values of traits. Values shown for loci and arrows are an example for illustration.

Conceptually, the relationship between individual genotypes and traits is defined by a network connecting loci to traits through a set of hidden internal nodes (Figure 1). Values for loci are randomly drawn from a standard normal distribution $\mathcal{N}(0, 1)$. Links between loci, internal nodes, and traits, can take any real value and are represented by black arrows in Figure 1. Traits can take any real value, and are calculated as the summed effects of all preceding nodes (i.e., the blue squares immediately to the left of the traits in Figure 1). Mathematically, loci are represented by row a vector of length L. Effects of loci on the first layer of internal nodes (black arrows emanating from loci in Figure 1) are represented by an $L \times T$ matrix, and transitions between internal nodes, and between the last set of internal nodes and the final traits, are represented by $T \times T$ matrices. For the example in Figure 1,

$$\left(0.1, \quad -0.1, \quad 0.2\right) \begin{pmatrix} 0.4, & 0.1 \\ -1.2, & -0.6 \\ 0.2, & 2.2 \end{pmatrix} \begin{pmatrix} 1.3, & -0.5 \\ -0.1 & 0.0 \end{pmatrix} \begin{pmatrix} 2.2, & 0.9 \\ 1.6 & 0.8 \end{pmatrix} = \left(0.2998, \quad 0.1081\right).$$

Values mapping loci to traits become part of an individual's genome, so the genome for the individual represented by Figure 1 is stored in the model as below:

```
0.1, -0.1, -0.2, 0.4, 0.1, -1.2, -0.6, 0.2, 2.2, 1.3, -0.5, -0.1, 0.0, 2.2, 0.9, 1.6, 0.8
```

Individuals with different loci can therefore have different covarying traits that are constrained by the network structure encoded in each genome. Individuals can be haploid (as in Figure 1) or diploid (in which case, allele values are summed at a locus).

An evolutionary algorithm is used to find appropriate values that produce covarying traits from loci (see Supporting Information 1 for details). Evolutionary algorithms are heuristic tools can simulate adaptive evolution to find solutions for a broad range of problems (Hamblin, 2013; Duthie et al., 2018). In the resevol package, the `mine_gmatrix` function runs an evolutionary algorithm, and requires the argument `gmatrix`, which specifies the desired trait covariance matrix. The function runs a genetic algorithm that initialises a population of `npsize` networks (Figure 1), and this population evolves until some maximum generation number (`max_gen`) or minimum expected network stress (`term_cri`) is met. In a single generation of the algorithm, network values crossover and mutate. Next, trait covariances produced for each network are estimated by initalising `indivs` individuals with loci sampled from a standard normal distribution. Network stress is calculated as the logged mean squared deviation between estimated covariances and those in `gmatrix`. Tournament selection (Hamblin, 2013) is then used to determine the networks for the next generation of the algorithm. Networks with the lowest estimated stress have the highest fitness, so these networks are disproportionately represented in the next generation. Throughout the evolutionary algorithm, the lowest stress network is saved and output upon network termination. The robustness of this network's stress to sets of individuals with different loci values can be tested using the `stress_test` function.

An example run of `mine_gmatrix` with the same number of loci and internal nodes is shown below with traits that do not covary:

```
trait_covs  <- matrix(data = c(1, -0.4, -0.4, 1), nrow = 2, ncol = 2);
new_network <- mine_gmatrix(loci = 3, layers = 2, gmatrix = trait_covs,
                            max_gen = 1000, term_cri = -6.0, prnt_out = FALSE);
```

The code above found a genome that produced the following expected trait covariances:

```
##              [,1]       [,2]
## [1,]  1.0008181 -0.3925799
## [2,] -0.3925799  1.0271340
```

The mean deviation between elements of the above matrix and the identity matrix provided by `trait_covs` is 2.1176024×10^{-4}. Lower values of `term_cri` and higher values of `max_gen` will result in a lower stress, but this will require more computation time, especially if the number of traits is high. Similarly, higher values of `indivs` will result in more accurate estimations of true stress, but this also requires more computation time. Additional arguments to `mine_gmatrix` can also be used to improve the performance of the evolutionary algorithm (see Supporting Information 1).

3 Simulating landscape-level pesticide resistance

The full output of `mine_gmatrix` is passed to the `run_farm_sim` function, which initialises and simulates an evolving population of pests on a changing landscape for a natural number of time steps (`time_steps`). In this section, we explain the landscape, pest ecology, and evolving pest traits.

3.1 Landscape

Landscapes are spatially explicit and initialised in `run_farm_sim` in one of two ways. First, a landscapes can be built from the arguments `xdim`, `ydim`, and `farms`. These arguments specify the dimensions of the landscape and the number of farms on it. Contiguous rectangular farms of roughly equal size are generated on the landscape using a splitline algorithm. Second, a custom landscapes can be input using the `terrain`

argument, which takes a matrix with elements that includes integers 1 to `farms`. Each value defines a
unique farm, but values do not need to be contiguous. These 'farms' could even model non-farmland (e.g.,
water, roads), if pesticides and crops on them are invisible to pests (see below). This `terrain` customisation
therefore allows for a high degree of landscape detail, and offers the potential for modelling real-world
landscapes from raster images. Landscapes are assumed to be a torus, so pests that move off of one edge
return on the opposite side of the landscape.

Each farm can hold one pesticide and one crop type at any time step. The `crop_init` and `pesticide_init`
arguments initialise one of `crop_number` crops and one of `pesticide_number` pesticides for each farm, respec-
tively (maximum of 10 each). Initialisation can be random for each farm with equal probability, or can be set
using a vector of length `farms` in which vector elements define the initialised crop or pesticide number. After
initialisation, crops and pesticides rotate once ever `crop_rotation_time` and `pesticide_rotation_time`
time steps, respectively. Each of these rotation time arguments can take either an integer value from 1-3, or a
matrix. Integer values specify (1) no rotation, (2) random transition from one type to another, or (3) cycling
through each available crop or pesticide in order. Square matrices can be used to define the probability that
a given crop or pesticide in row i transitions to that in column j. Hence, any possible Markov chain can be
used to transition between crop or pesticide types on farms.

3.2   Pest ecology

Individual pests can be modelled to have several reproductive systems and life histories. Pest reproductive
system can be specified in the `run_farm_sim` function using the `repro` argument, which accepts arguments
`"asexual"` (haploid), `"sexual"` (monoecious), and `"biparental"` (dioecious). For `"sexual"` pests, the
`selfing` argument specifies if self-fertilisation is (`TRUE`) or is not (`FALSE`) allowed. At the start of a simulation,
pests are initialised in a random location. Initialised pests are age of zero if `rand_age = FALSE` or a random
age from zero to `max_age` if `rand_age = TRUE`. Following initialisation, a single time step proceeds with land-
scape change (see above), pest aging and metabolism, feeding, pesticide consumption, movement, reproduc-
tion, mortality, and immigration (Figure 2). Feeding, pesticide consumption, movement, and reproduction all
depend on pest age. Pests feed and consume pesticide from ages `min_age_feed` to `max_age_feed`, move from
ages `min_age_move` to `max_age_move`, and reproduce from ages `min_age_reproduce` to `max_age_reproduce`
(all inclusive). Food accumulated is lost during aging if `baseline_metabolism > 0` and pest age is within
`min_age_metabolism` and `max_age_metabolism`. The option to set minimum and maximum ages for events
makes it possible to model pests with much different life histories (e.g., Sudo et al., 2018).

In each time step, pests feed in a random order, consuming the crop on their landscape cell. Pests consume a
maximum amount of the crop as specified by the `food_consume` argument, which takes a vector with as many
elements as their are crops (i.e., if `crop_number = 2`, then `food_consume` has two elements, the first and
second defining consumption of crops 1 and 2, respectively). If crop amount on the landscape cell exceeds
pest consumption ability, then pests consume their maximum amount, and this amount is removed from
the landscape cell. If crop amount is less than pest consumption ability, then pests consume whatever crop
is left and crop amount is reduced to zero. Pesticide consumption works identically to crop consumption,
except that the amount of pesticide on a landscape cell is not decreased. Pests simply consume an amount of
pesticide as specified by the `pesticide_consume` argument, which also takes a vector with as many elements
as their are pesticides. Hence, each pest can potentially feed and be affected by the pesticide of their focal
landscape cell.

After interacting with their landscape cell, pests can move. Each pest visits a number of cells during
movement, as is specified by the parameter `movement_bouts`. Individual movement bouts occur in a
random order across pests. During a movement bout, a pest can travel to any cell within a value de-
fined by `movement_distance` from their current location, which could include their current location (i.e.,
moving zero distance). Upon arrival to a cell, a pest can feed if they are of an appropriate feeding age
and `feed_while_moving = TRUE`. The pest also consumes pesticide if they are of the appropriate age and
`pesticide_while_moving = TRUE`. Having pests feed and consume pesticide in a random order while mov-
ing among landscape cells can model a population competing for food and encountering pesticides on a
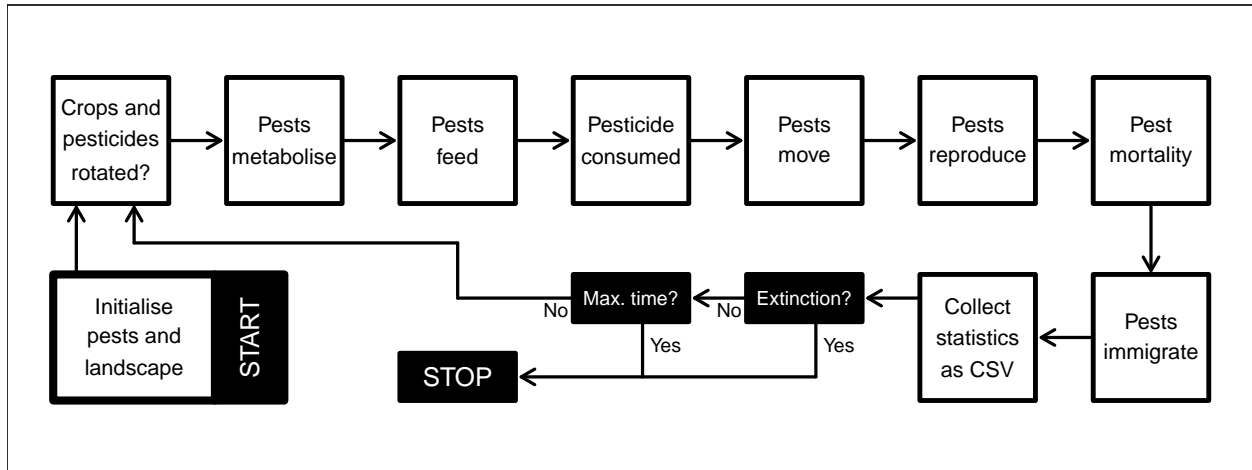shorter time scale than an individual time step.

Figure 2: Overview of simulated events in the resevol R package. Note that metabolism, feeding, pesticide consumption, movement, and reproduction are all subject to a minimum and maximum pest age. Consequently, simulation order might not reflect the order of events from the perspective of a focal pest (e.g., pests might move from ages 1-2, but only feed from ages 2-4). Crops and pesticides are also not necessarily rotated in each time step (see Landscape). Statistics collected within a time step are printed in a CSV file.

After all pests are done moving, pests reproduce. Offspring production is possible for asexual, monoecious, or female pests. Pest expected offspring number is defined by a fixed parameter if `reproduction_type = "lambda"`, or is calculated from the amount of food consumed if `reproduction_type = "food_based"`. The former requires specifying `lambda_value`, which becomes the rate parameter for sampling offspring number from a Poisson distribution. The latter requires specifying a real value for `food_needed_repr`, which is the amount of food needed to produce one offspring. For food-based reproduction, the total amount of food consumed is divided by `food_needed_repr`, then floored to determine offspring number. Sexual reproduction requires a mate of reproductive age that is either monoecious or male, and within range of the reproducing focal pest (potential including the focal pest, if `selfing = TRUE`). A potential mate is within range if it is within an integer number of cells from the focal pest, as defined by `mating_distance` (e.g., if `mating_distance = 0`, then mates must share a landscape cell). All available potential mates sire offspring with equal probability, and reproducing pests are assumed to mate multiply (i.e., paternity is a fair raffle for all offspring). If a carrying capacity on birth is set (`K_on_birth > 0`) and total offspring number in the population exceeds this capacity, then offspring are removed at random until they are within carrying capacity. An real value `immigration_rate` specifies the rate parameter for Poisson random sampling of the number of immigrants added to the population at the end of a time step. Immigrants are initialised in the same way as pests were at the start of the simulation. Hence, any spatial structure or evolution that occurs during the simulation does not affect immigrant locations, genomes, or traits.

3.3 Pest evolution

Pest genomes evolve in a complex and highly mechanistic way. Offspring inherit genome values from their parent(s) with the possibility for mutation and recombination; offspring traits are then calculated from their newly initialised genomes. For asexually and sexually reproducing pests, genomes are haploid and diploid, respectively. Asexually reproducing pests receive the full genomes of their parent, while sexually reproducing pests receive half of their alleles from each parent. Each diploid parent contributes one half of their genome, effectively modelling a genome with a single paired chromosome. Crossover occurs at each position of the genome with a probability of `crossover_pr`, so complete recombination is also possible if `crossover_pr = 0.5`. For both haploids and diploids, each genome value then mutates independently with a probability of `mutation_pr`, which can be set to any real number between 0 and 1. If a genome value mutates, then a new value is randomly sampled from a standard normal distribution. If `mutation_type = 0`, then this new

176 value replaces the old value, and if `mutation_type = 1`, then the new value is added to the old value. After mutation, genome values are used to calculate trait values.

178 Evolution of the genetic architecture linking loci to traits can be constrained by disabling mutation genome values linking loci, internal nodes, and traits (i.e., 'network values' represented by arrows in Figure 1).
180 While mutation at loci (green circles in Figure 1) is always possible as long as `mutation_pr > 0`, the number of intermediary layers for which network values can mutate is constrained by `net_mu_layers`. If
182 `net_mu_layers = 0`, then no network values can mutate, but higher integer values cause mutation to occur at network value layers from loci to traits (if `net_mu_dir = 1`) or traits to loci (if `net_mu_dir = 0`). For
184 example, if `net_mu_layers = 2` and `net_mu_dir = 1`, then the network values linking loci to the first internal node, and the first internal node to the second, can mutate (i.e., first two columns of arrows in
186 1, but not those linking the second internal node to traits). This allows pest traits to evolve with varying degrees of constraint on the covariance between traits. Low `net_mu_layers` values model strong genetic
188 constraints, while high values model high evolvability of trait covariances.

Finally, evolving and covarying traits can be used in place of fixed parameters described in pest ecology.
190 This is done by substituting `"Tj"` as argument input in place of a numeric value, where j represents the trait number. For example, the argument `move_distance = "T1"` will make Trait 1 the movement
192 distance for individuals. The argument `food_consume = c("T2", "T3")` will set Traits 2 and 3 to define the amount of food of types 1 and 2 that can be consumed by a pest, respectively. Param-
194 eters that can be replaced include `move_distance`, `food_needed_surv`, `pesticide_tolerated_surv`, `food_needed_repr`, `pesticide_tolerated_repr`, `mating_distance`, `lambda_value`, `movement_bouts`,
196 `metabolism`, `food_consume`, and `pesticide_consume`. The resevol package can thereby simulate agricultural pests with complex and co-evolving traits, and potentially evolving trait covariances, under a range of
198 possible pest life histories.

4 Simulation output

200 Simulation output is large, so output is printed in two CSV files, both of which are created in the working directory. The first file "population_data.csv" prints population level data over time, including population
202 size, mean age, sex ratio, mean food and pesticide consumed of each type, mortality rate, and mean trait values. The second file "individuals.csv" prints all information, including full genomes and traits (columns),
204 for every individual (rows) in the population. The printing of individual level data is disabled by default. It can be turned on for all time steps by setting `print_inds = TRUE`, but this should be done with caution
206 because it can create extremely large files. Instead, individual level data can be printed for only the final time step by setting `print_last = TRUE`. Output produced by `run_farm_sim` is a list of two elements, which
208 includes a vector of parameter values used in the simulation and the final state of the landscape as an array.

5 Example of individual-based simulations

210 In resevol v0.3, the `run_farm_sim` function includes 68 arguments, which specify a wide range of possible simulation conditions affecting landscape and pest characteristics. These arguments are explained the
212 package documentation, and Supporting Information 2, which demonstrates an advanced case study with a custom landscape and complex pest genomes and life history. Here we focus on a simple simulation with
214 asexually reproducing pests that have three loci and two traits (Figure 1). We use the the genome generated in `new_network` from Section 2, in which traits 1 and 2 have variances of 1.0008181 and 1.027134,
216 respectively, with a covariance of -0.3925799.

Traits 1 and 2 will define the rate at which pests consume two separate pesticides, so we model a system in
218 which there is a trade-off for pesticide susceptibility. We use a simple 64×64 cell landscape with nine farms. Each farm grows the same crop and uses one of two randomly intialised pesticides, which are rotated every
220 16 time steps. Hence, we can conceptualise 16 time steps as a single growing season.

In our example, each cell holds four crop units, all of which can potentially be consumed. Pests consume up
to one unit of crop on their landscape cell per time step. Pest survival and reproduction is food-based, and
pests need to consume one unit of crop by age two to survive and reproduce. Pests initialised at the start
of the simulation are randomly assigned an age from 0-4 with equal probabilities. Pests have a maximum
age of four, and they feed in ages 0-2, move up to two cells in age three and four, and reproduce in age four.
Pests can consume pesticides in ages 0-2, and if they consume any pesticide, then they will die. In each time
step, 10 immigrants arrive. We simulate 160 time steps (10 growing seasons) using the function below.

```
sim <- run_farm_sim(mine_output = new_network, repro = "asexual",
                    pesticide_number = 2, pesticide_init = "random",
                    pesticide_consume = c("T1", "T2"), farms = 9,
                    pesticide_rotation_time = 16, pesticide_rotation_type = 3,
                    pesticide_tolerated_surv = 0, pesticide_per_cell = 1,
                    crop_rotation_time = 16, crop_number = 1, crop_per_cell = 4,
                    food_consume = 1, reproduction_type = "food_based",
                    food_needed_surv = 1, food_needed_repr = 1, max_age = 4,
                    min_age_feed = 0, max_age_feed = 2, min_age_move = 3,
                    max_age_move = 4, min_age_reproduce = 4, print_gens = FALSE,
                    max_age_reproduce = 4, age_pesticide_threshold = 2,
                    rand_age = TRUE, move_distance = 2, immigration_rate = 10,
                    time_steps = 160, print_last = TRUE, xdim = 64, ydim = 64);
```

Any arguments to `run_farm_sim` not included above are set to default values. Output files can be used
to plot ecological and evolutionary dynamics of pests. Figure 3a shows the pest population change both
over and within seasons. Due to the specific parameter values chosen, clear patterns reflecting pest cohort
emerge. At the start of a season when the most crop is available, pests eat and abundance increases. As less
crop becomes available in a season, pest abundance decreases. Mean food consumed varies over the course
of a season caused by the varying frequency of pests in different life history stages (Figure 3c). In the first
50 time steps, there is some consumption of pesticides 1 and 2 (Figure 3b), but both traits rapidly evolve to
negative values making both pesticides ineffective (Figure 3d). In this case, the trade-off between pesticide
consumption is not strong enough to maintain susceptibility to either pesticide in the population. Figure 4
shows the location of pests on the landscape in the last time of the simulation (left panel), and how different
pesticides are being applied on the landscape (right panel). Because the number of crops and pesticides,
the number and nature of traits, and the size of pest genomes all vary depending on simulation parameters,
no plotting functions are introduced in the resevol package. Instead, methods and code for producing plots
such as those in Figures 3 and 4 are explained in Supporting Information 2.

6   Availability

The resevol R package can be downloaded from CRAN (https://cran.r-project.org/package=resevol) or
GitHub (https://bradduthie.github.io/resevol/). The package is open source under GNU Public License.

7   Conclusions
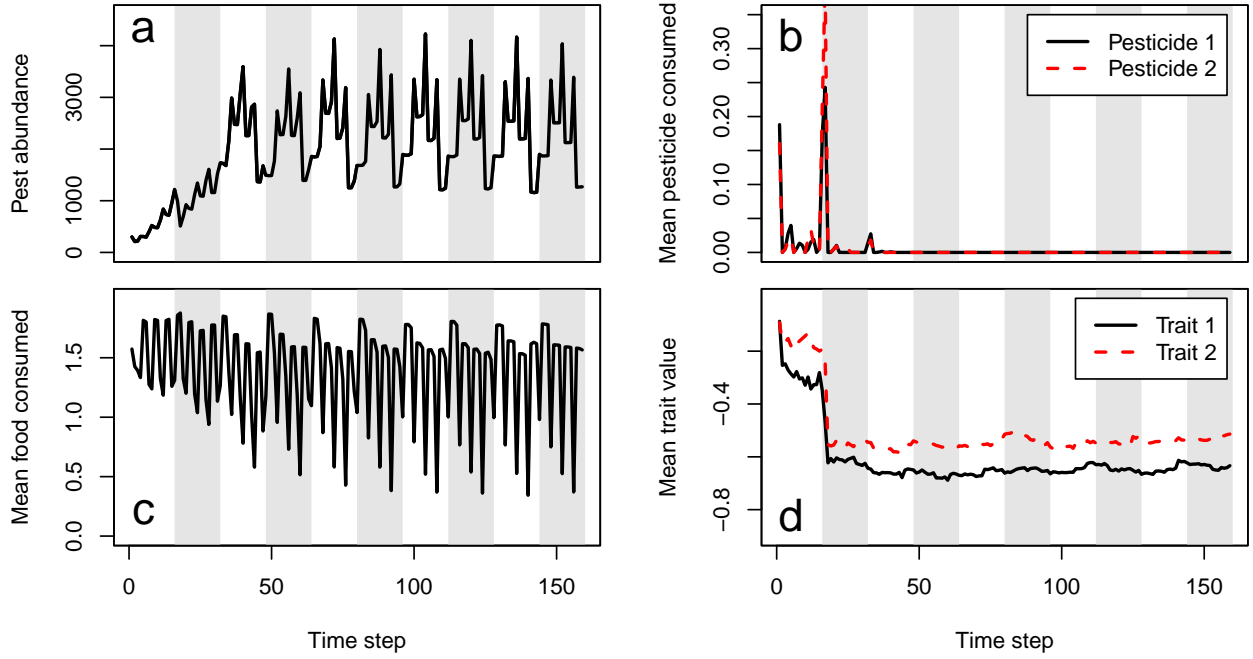
8   Acknowledgements

Figure 3: Agricultural pest ecological and evolutionary dynamics over 160 time steps from an individual-based simulation using the resevol R package. Panels show (a) pesticide abundance change, (b) mean of pesticides 1 and 2 consumed per pest, (c) mean food consumed per pest, and (d) mean values of evolving traits 1 and 2 over time. Traits 1 and 2 define how much of pesticides 1 and 2, respectively, a pest will consume. White and grey vertical stripes indicate seasons of a single crop and pesticide application.

Figure 4: Locations of pests (black) across a landscape that includes nine farms (coloured blocks) in the last time step of a simulation using the resevol R package (left panel). The right panel shows which farms apply pesticide 1 (dark grey) and 2 (light grey).

References

Duthie, A. B., Cusack, J. J., Jones, I. L., Nilsen, E. B., Pozo, R. A., Rakotonarivo, O. S., Moorter, B. V., and Bunnefeld, N. (2018). GMSE: an R package for generalised management strategy evaluation. *Methods in Ecology and Evolution*, 9:2396–2401.

Hamblin, S. (2013). On the practical usage of genetic algorithms in ecology and evolution. *Methods in Ecology and Evolution*, 4(2):184–194.

Sudo, M., Takahashi, D., Andow, D. A., Suzuki, Y., and Yamanaka, T. (2018). Optimal management strategy of insecticide resistance under various insect life histories: Heterogeneous timing of selection and interpatch dispersal. *Evolutionary Applications*, 11(2):271–283.