Package 'resevol'

August 25, 2026

Type Package

Title Simulate Agricultural Production and Evolution of Pesticide Resistance

Version 0.4.1.1

Imports stats(>= 4.0.0), utils (>= 4.0.0)

Maintainer A. Bradley Duthie <brad.duthie@gmail.com>

Description Simulates individual-based models of agricultural pest management and the evolution of pesticide resistance. Management occurs on a spatially explicit landscape that is divided into an arbitrary number of farms that can grow one of up to 10 crops and apply one of up to 10 pesticides. Pest genomes are modelled in a way that allows for any number of pest traits with an arbitrary covariance structure that is constructed using an evolutionary algorithm in the mine_gmatrix() function. Simulations are then run using the run_farm_sim() function. This package thereby allows for highly mechanistic social-ecological models of the evolution of pesticide resistance under different types of crop rotation and pesticide application regimes.

URL https://bradduthie.github.io/resevol/

BugReports https://github.com/bradduthie/resevol/issues

Depends R (>= 4.0.0)

License GPL (>= 2)

LazyData TRUE

Encoding UTF-8

VignetteBuilder knitr

Suggests knitr, rmarkdown, testthat, markdown

RoxygenNote 7.3.2

NeedsCompilation yes

Author A. Bradley Duthie [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8343-4995>),
Rose McKeon [aut, ctr],
Rosie Mangan [ctr],
Luc Bussiére [ctr],
Matthew Tinsley [ctr]

Repository CRAN

Date/Publication 2026-08-25 21:40:02 UTC

Contents

mine_gmatrix 2
run_farm_sim 4
stress_test 14

Index **15**

`mine_gmatrix` *Mine G-matrices*

Description

Mine networks for establishing the link between genome and g-matrix. The output from this function is required to run individual-based simulations in the rest of the package. The key input to this function, 'gmatrix', is a (square) covariance matrix, with each row and column representing a trait for the individual-based model. This function will run an evolutionary algorithm to try to find a network that produces traits with the covariance structure of gmatrix from a set of random standard normal values. The network from loci values to trait values goes through a number of linked nodes to achieve this, and each generation tests the stress of the resulting network in terms of expected squared deviation of trait covariances from the input gmatrix. Simulations can take minutes to hours or longer, depending on parameters chosen and the number of traits. See vignettes for a more comprehensive explanation for what this function is doing.

Usage

```
mine_gmatrix(
  loci = 18,
  layers = 6,
  indivs = 1000,
  npsize = 2000,
  mu_pr = 0.05,
  mu_sd = 0.01,
  max_gen = 1000,
  pr_cross = 0.05,
  sampleK = 40,
  chooseK = 4,
  term_cri = -5.3,
  sd_ini = 0.1,
  use_cor = FALSE,
  prnt_out = TRUE,
  gmatrix
)
```

Arguments

loci	The number of loci for an individual. Simulations can allow for both haploid and diploid individuals. Allele values at each loci affect trait values through a network of intermediary nodes.
layers	The number of hidden layers in the network linking loci to traits.
indivs	The number of individuals initialised in each generation of the evolutionary algorithm to test among-individual trait correlations. Individuals are initialised with allele values drawn from a standard normal distribution.
npsize	The size of the population of networks in each generation of the evolutionary algorithm. Each network is a discrete individual in the population.
mu_pr	The probability that a value in the network will mutate in a generation. Mutation events change the existing value by adding a new value drawn from a normal distribution with a mean of 0 and standard deviation of mu_sd.
mu_sd	The standard deviation of the random normal value mean centered at 0 that is added to the existing value of the network when a mutation event occurs.
max_gen	The maximum number of generations that the evolutionary algorithm is allowed to run before terminating (regardless of how well the evolved covariance structure matches the pre-specified gmatrix).
pr_cross	The probability that a focal network in the population will initiate a crossover of a subset of its values with a randomly selected second network (note that any given network might therefore be included in more than one crossover event in a generation). The size of the subset is determined randomly.
sampleK	During a round of selection, the number of random networks chosen to compete in a tournament. A single generation will include as many tournaments as necessary to create a new network population of size npsize.
chooseK	During a round of selection tournament, the number of networks within the sampleK random subset of the tournament that have the highest fitness will be selected to populate the next generation of networks
term_cri	The criteria for terminating the evolutionary algorithm. The algorithm will terminate if a network is found in which the mean squared deviation of the covariance matrix elements from gmatrix is less than exp(term_crit).
sd_ini	The standard deviation of initialised network values at the start of the evolutionary algorithm. All network values are initialised by randomly sampling from a normal distribution with a mean of 0 and a standard deviation of sd_ini
use_cor	Should the gmatrix be treated as a correlation matrix rather than a covariance matrix when calculating fitness?
prnt_out	Should the function print out progress showing the stress for each generation
gmatrix	The pre-specified trait covariance matrix. This will define what the covariance will be between each trait when allele values are drawn from a standard normal distribution.

Value

A list of eight elements that includes the following: (1) A vector of input parameters, (2) the pre-specified covariance matrix, (3) matrix defining the effects of loci values on the first layer of the network, (4) a three dimensional array link the first network layer to trait values, (5) a matrix of the marginal effect of each locus on each trait, (6) the mined covariance structure, (7) all network values to be inserted into individual genomes, and (8) the log stress of the mined matrix against the pre-specified matrix.

Examples

```
gmt       <- matrix(data = 0, nrow = 4, ncol = 4);
diag(gmt) <- 1;
mg        <- mine_gmatrix(gmatrix = gmt, loci = 4, layers = 3, indivs = 100,
                          npsize = 100, max_gen = 2, prnt_out = FALSE);
```

---

run_farm_sim *Initialise individuals and simulate farming*

Description

Initialises a new set of individuals and then simulates farming over time. This is the main function that runs individual-based simulations of crop and pesticide use and the evolution of pesticide resistance over time. To run this function, output from the mine_gmatrix function is required to specify the covariance structure of individual traits and individual genomes. The arguments to this function are used to initialise a landscape with the make_landscape function and initialise individuals with the initialise_inds function. After initialisation, the simulation continues for up to a set number of time steps (unless extinction occurs), and individuals on the landscape feed, encounter pesticide, move, reproduce, and die depending upon the arguments specified in this function. After a specified number of time steps, the crop or pesticide applied to a landscape cell can also change. The end result is an evolving population of individuals that express traits that can potentially affect fitness (e.g., food consumption, pesticide consumption, movement). Population level statistics are calculated by default and printed to a CSV, but individual level data (which includes all individual characteristics in a large table) need to be turned on because files can become extremely large (use print_inds with extreme caution and print_last with care).

Usage

```
run_farm_sim(
  mine_output,
  N = 1000,
  xdim = 100,
  ydim = 100,
  repro = "sexual",
  neutral_loci = 10,
  max_age = 9,
  min_age_move = 0,
  max_age_move = 9,
```

```
  min_age_reproduce = 0,
  max_age_reproduce = 9,
  min_age_feed = 0,
  max_age_feed = 9,
  food_consume = 0.25,
  pesticide_consume = 0.1,
  rand_age = FALSE,
  move_distance = 1,
  food_needed_surv = 0.25,
  pesticide_tolerated_surv = 0.1,
  food_needed_repr = 0,
  pesticide_tolerated_repr = 0,
  reproduction_type = "lambda",
  mating_distance = 1,
  lambda_value = 1,
  movement_bouts = 1,
  selfing = TRUE,
  feed_while_moving = FALSE,
  pesticide_while_moving = FALSE,
  mortality_type = 0,
  age_food_threshold = 0,
  age_pesticide_threshold = 0,
  farms = 4,
  time_steps = 100,
  mutation_pr = 0,
  crossover_pr = 0,
  mutation_type = 0,
  net_mu_layers = 0,
  net_mu_dir = 0,
  mutation_direction = 0,
  crop_init = "random",
  crop_rotation_type = 2,
  crop_rotation_time = 1,
  pesticide_init = "random",
  pesticide_rotation_type = 2,
  pesticide_rotation_time = 1,
  crop_per_cell = 1,
  pesticide_per_cell = 1,
  crop_sd = 0,
  pesticide_sd = 0,
  crop_min = 0,
  crop_max = 1000,
  pesticide_min = 0,
  pesticide_max = 1000,
  crop_number = 2,
  pesticide_number = 1,
  print_inds = FALSE,
  print_gens = TRUE,
```

```
  print_last = FALSE,
  K_on_birth = 1e+06,
  pesticide_start = 0,
  immigration_rate = 0,
  get_f_coef = FALSE,
  get_stats = TRUE,
  metabolism = 0,
  baseline_metabolism = 0,
  min_age_metabolism = 1,
  max_age_metabolism = 9,
  terrain = NA,
  trait_means = NULL,
  land_edge = "torus",
  crop_growth = 0,
  crop_growth_type = "none",
  pesticide_threshold = "none",
  pesticide_delay = 1,
  population_filename = "population_data.csv",
  last_step_filename = "last_time_step.csv",
  initial_inds = NULL
)
```

Arguments

mine_output	The output from mine_gmatrix, which will be used to initialise the genomes and traits of pests.
N	The number of individuals that are initialised in a simulation. Individuals are initialised in a random location on the landscape, and at least two individuals are needed.
xdim	The number of cells in the horizontal dimension of the landscape. This value must be an integer greater than two.
ydim	The number of cells in the vertical dimension of the landscape. This value must be an integer greater than two.
repro	The type of reproduction that individuals undergo in the simulation. There are three options: (1) "asexual," in which individuals reproduce clonally and offspring have haploid genomes and traits identical to their mother with the potential for mutation; (2) "sexual," in which individuals are monoecious (both female and male) and offspring have diploid genomes with alleles inherited from both parents with mutation and recombination; (3) "biparental," in which individuals are dioecious (only female or male) and offspring have diploid genomes with alleles inherited from both parents with mutation and recombination.
neutral_loci	The number of loci that are completely neutral (i.e., have no effect on fitness). These loci can be used to monitor genetic drift or calculate inbreeding coefficients.
max_age	This is the maximum number of time steps that an individual can survive. Individuals that are older than this age in a time step will always die.

`min_age_move` This is the minimum age at which an individual can move. Individuals below this age will always remain on their current cell.

`max_age_move` This is the maximum age at which an individual can move. Individuals above this age will always remain on their current cell.

`min_age_reproduce`
This is the minimum age at which an individual can be reproductively active. No individuals below this age will engage in any reproductive activity, nor will they be recognised as potential mates by other individuals.

`max_age_reproduce`
This is the maximum age at which an individual can be reproductively active. No individuals above this age will engage in any reproductive activity, nor will they be recognised as potential mates by other individuals.

`min_age_feed` This is the minimum age at which an individual can eat. No individuals below this age will be able to consume food on the landscape.

`max_age_feed` This is the maximum age at which an individual can eat. No individuals above this age will be able to consume food on the landscape.

`food_consume` This defines how much food an individual will consume from the cell on which it is feeding. Food consumption can take on any positive real value, and an individual will consume up to this amount if possible (if not, they will consume however much food is left within their landscape cell).

`pesticide_consume`
This defines how much pesticide an individual will consume from the cell on which it resides. Pesticide consumption can take on any positive real value, and an individual will consume up to this amount if possible (if not, they will consume however much pesticide has been placed on the landscape cell).

`rand_age` This argument determines whether individuals in the simulation will be initialised with a random age selected uniformly from zero to max_age. If FALSE, then all individuals will be initialised at age zero.

`move_distance` This is the maximum number of cells that an individual can move, in any direction, on the landscape during one bout of movement.

`food_needed_surv`
This is the amount of food that an individual needs to consume to survive. If the individual has not consumed this amount of food before the age of age_food_threshold, then they will die in the time step.

`pesticide_tolerated_surv`
This is the amount of pesticide that an individual can tolerate and still survive. If the individual has consumed more than this amount of pesticide on or after the age of age_pesticide_threshold, then they will die in the time step.

`food_needed_repr`
This is the amount of food that an individual needs to produce one offspring. The total number of offspring that an individual produces in a time step is the floor value of their food consumption divided by this value.

`pesticide_tolerated_repr`
This is the amount of pesticide tolerated below which an individual can reproduce. Note that individuals above the threshold can still mate and sire offspring,

`reproduction_type`
: This determines how individuals reproduce; the two options are "lambda" and "food_based." If "lambda," then the number of offspring an individual produces is sampled from a Poisson distribution with a fixed rate parameter lambda_value (potentially adjusted by other factors in the simulation). If "food_based," then the number of offspring produced is based on the amount of food consumed by the individual.

`mating_distance`
: This is the distance in cells (any direction) away from a focal individual from which they can successfully find and identify a mate (e.g., if 0, then only individuals on the same cell are potential mates).

`lambda_value`
: This is the rate parameter for the Poisson sampling of offspring number; it only applies when reproduction_type is set to "lambda."

`movement_bouts`
: This is the number of times an individual can move in a single time step (i.e., the number of cells that it can potentially visit). Each time an individual visits a new cell, it can potentially feed or consume pesticide.

`selfing`
: This determines whether or not self-fertilisation is allowed when repro is set to "sexual."

`feed_while_moving`
: If TRUE, then individuals will feed in each movement bout when they arrive to a new landscape cell.

`pesticide_while_moving`
: If TRUE, then individuals will consume pesticide in each movement bout when they arrive to a new landscape cell.

`mortality_type`
: This determines how mortality is enacted in the simulation. Currently there is only one mortality type possible; mortality occurs if individuals exceed their maximum age, do not consume enough food, or consume too much pesticide.

`age_food_threshold`
: This is the age at which mortality associated with feeding is enacted, so an individual younger than this age will not die if they have not yet consumed sufficient food to satisfy food_needed_surv.

`age_pesticide_threshold`
: This is the age at which mortality associated with pesticide consumption is enacted, so an individual younger than this age will not die even if they have exceeded their pesticide threshold.

`farms`
: This is the number of farms to be placed on the landscape. Farms are placed in blocks of roughly equal sizes using a shortest splitline algorithm. Farms operate independently in terms of what crops they grow and pesticides they apply.

`time_steps`
: This is the number of time steps that a simulation will run. Simulations will be terminated before this number if extinction occurs.

`mutation_pr`
: This is the probability of mutation occurring at any locus of a newly produced offspring.

`crossover_pr`
: This is the probability of crossover between two homologous loci. This only applies for diploid genomes.

`mutation_type` This determines how mutation is modelled. If 0, then a completely new allele value is drawn from a normal distribution with a mean of mutation_direction and a standard deviation of 1 (or 1 / sqrt(2) for diploids, so that the expected standard devation of the sum of both allele values is 1). If 1, then a new value is drawn from a normal distribution with mean mutation_direction and standard deviation of 1, and this new value is then added to the existing allele value.

`net_mu_layers` This is the proportion of the genome that can evolve. If 0, then only loci values (green circles in Figure 1) can mutate. If 1, then loci and the first column of arrows (green circles to first column of blue squares in Figure 1) can mutate. If 2, then the first two columns of arrows in Figure 1 can mutate, and so forth. Fewer mutation layers will constrain the covariance among traits, while more mutation layers will allow the covariance structure to evolve more readily.

`net_mu_dir` The direction along the network in which net_mu_layers applies (not loci, green circles in Figure 1, can always mutate). If 1, then net_mu_layers applies in the direction from loci to traits. If 0, then the direction applies from traits to loci (i.e., net_mu_dir = 0 and net_mu_layers = 1 would mean that only the arrow values between the last hidden layer and traits in Figure 1 could mutate).

`mutation_direction`
This allows mutations to be biased in one direction. A default value of 0 makes positive or negative allele values equally likely.

`crop_init` Initial crop type for each farm. This can be set in one of two ways. First, the default value "random" will randomly assign each farm to an initial crop to produce. Second, a vector can be used to specify the crop initialised on each farm. The vector must be the same length as the number of farms, and the value of each element 'i' of the vector defines which crop is initialised for each farm i. Hence, a crop_init vector must have as many elements as there are farms, and vector elements must include natural numbers from 1 to the total number of crops.

`crop_rotation_type`
This determines how crop types are rotated across the landscape. This can be set in one of two ways. First, a natural number can specify a rotation type: (1) crops will never rotate, (2) a new crop type will be randomly chosen every crop_rotation_time time steps for each farm, or (3) farms will cycle through crop types in order, with a change from one crop type to another every crop_rotation_time time step. Second, a square matrix can specify the probability of transition from a focal crop type (rows) to the next crop type (columns). Matrix rows must therefore sum to 1. For example, an identity matrix (1s in the diagonal and 0s in the off-diagonal) would specify crops that never rotate (i.e., crop i always rotates to itself).

`crop_rotation_time`
This determines how many time steps a crop is left before being refreshed and potentially changed. Note that even if the crop type does not change, this value still has the effect of determining how often crops are replenished (if some have been eaten since the last time they were replenished).

`pesticide_init` Initial pesticide type for each farm. This can be set in one of two ways. First, the default value "random" will randomly assign each farm to an initial pesticide to apply. Second, a vector can be used to specify the pesticide initialised on each

farm. The vector must be the same length as the number of farms, and the value of each element 'i' of the vector defines which pesticide is initialised for each farm i. Hence, a pesticide_init vector must have as many elements as there are farms, and vector elements must include natural numbers from 1 to the total number of pesticides.

`pesticide_rotation_type`
: This determines how pesticide types are rotated across the landscape. This can be set in one of two ways. First, a natural number can specify a rotation type: (1) pesticides will never rotate, (2) a new pesticide type will be randomly chosen every pesticide_rotation_time time steps for each farm, or (3) farms will cycle through pesticide types in order, with a change from one pesticide type to another every pesticide_rotation_time time step. Second, a square matrix can specify the probability of transition from a focal pesticide type (rows) to the next pesticide type (columns). Matrix rows must therefore sum to 1. For example, an identity matrix (1s in the diagonal and 0s in the off-diagonal) would specify pesticides that never rotate (i.e., pesticide i always rotates to itself).

`pesticide_rotation_time`
: This determines how many time steps a pesticide is left before being replenished and potentially changed. Note that unlike crops, pesticide levels do not decrease on the landscape over time (e.g., with consumption).

`crop_per_cell`
: This determines the expected amount of crop that is placed on a single landscape cell. The more crop on a cell, the more that can be potentially consumed by individuals.

`pesticide_per_cell`
: This determines how much pesticide is placed on a single landscape cell. The higher concentration of pesticide per cell, the more that individuals on the cell will imbibe and potentially be affected by.

`crop_sd`
: This is the standard deviation of crop number placed on landscape cells. A default value of 0 assumes that all cells have the same amount of crop.

`pesticide_sd`
: This is the standard deviation of pesticide applied to each landscape cell. A default value of 0 assumes that each cell has the same concentration of pesticide applied.

`crop_min`
: This is the minimum amount of crop that is possible to have on a single cell (i.e., crop values will never be initialised to be lower than this value).

`crop_max`
: This is the maximum amount of crop that is possible to have on a single cell (i.e., crop values will never be initialised to be higher than this value).

`pesticide_min`
: This is the minimum concentration of pesticide that is possible to have on a single cell (i.e., pesticide values will never be initialised to be lower than this value).

`pesticide_max`
: This is the maximum concentration of pesticide that is possible to have on a single cell (i.e., pesticide values will never be initialised to be higher than this value).

`crop_number`
: This is the number of unique crops that can exist on the landscape during the course of a simulation. The maximum number of possible crops is 10.

`pesticide_number`
: This is the number of unique pesticides that can exist on the landscape during the course of a simulation. The maximum number of possible pesticides is 10.

print_inds If TRUE, a CSV file will print in the working directory with every individual and all of their characteristics (i.e., locations, traits, genomes) in every time step. By default, this is set to FALSE and should only be set to TRUE with extreme caution, as large populations persisting over long periods of time can produce extremely large CSV files.

print_gens If TRUE, the time step and the population size will be printed to the R console as the simulation is running.

print_last If TRUE, a CSV file will print in the working directory with every individual and all of their characteristics (i.e., locations, traits, genomes) in only the last time step. Note that for large populations, the file size generated can be very large (10s to 100s of GBs).

K_on_birth This is a carrying capacity applied to new individuals across the entire landscape. If the total number of offspring in a time step exceeds this value, then offspring are removed at random until the total number of new offspring equals K_on_birth. In practice, this can help speed up simulations by avoiding the unnecessary production of individuals when most will perish.

pesticide_start
This is the time step at which pesticide begins to be applied. No pesticide will be applied prior to this start time, so individuals will not experience any effects of pesticide. This can be useful as a tool to burn in the population prior to introducing pesticide.

immigration_rate
This is the number of immigrant individuals arriving in the landscape in each time step. Immigrants are initialised in random locations with the same network structure (Figure 1) as individuals initialised at the start of the simulation, and with allele values randomly drawn from a standard normal distribution.

get_f_coef This determines whether or not inbreeding coefficients will be calculated for sexual populations and printed off in CSV files. Because this can add some computation time, it is best to set to FALSE unless it is needed.

get_stats If TRUE, a CSV file will print in the working directory with summary statistics for each time step. This is set to TRUE by default.

metabolism This determines the rate at which food consumed in previous time steps is lost in subsequent time steps, which can be especially relevant if food consumed determines survival or reproductive output. Values of 0 mean that stored gains will always persist throughout an individual's lifetime, while very high values will model the gains of one time step being wiped out in subsequent time steps (if, e.g., the objective is to model individuals needing to consume food successfully in each time step to survive or reproduce, as opposed to having a feeding life history stage followed by a mating and reproduction stage).

baseline_metabolism
This fixes a baseline metabolic rate at which food consumed in previous time steps is lost in subsequent steps. This fixed value is always added to metabolism for each individual. By default, this value is 0.

min_age_metabolism
This determines the minimum age at which losses of food consumed in previous time steps enacted by metabolism and baseline_metabolism can occur.

max_age_metabolism
: This determines the maximum age at which losses of food consumed in previous time steps enacted by metabolism and baseline_metabolism can occur.

terrain
: Insert a custom terrain of different farms, which takes the form of a matrix that includes a sequence of natural numbers in all matrix elements. For example, if there are 4 farms, then all matrix elements must be 1, 2, 3, or 4. Beyond this requirement, there is no restriction on where different farms are placed; the do not even need to be contiguous on the landscape. Note that a custom terrain will override the arguments farms, xdim, and ydim. For example, if the matrix given to the terrain argument has 10 rows and 10 columns, then the simulation will automatically set xdim and ydim equal to 10 without any warnings. Also note that these terrain values do not necessarily need to be farms. Through the use of a custom landscape and pesticide rotation option, these cells could represent something like diversionary feeding sites or even buildings or rivers. See vignettes and other documentation for details.

trait_means
: This provides the mean values of the evolving pest traits in the initialised pest population, which by default are zero. To change mean trait values in the initialised population, the trait_means argument requires a vector of the same length as the number of evolving pest traits defined within the function (i.e., if there are two evolving traits in the simulation, "T1" and "T2", then trait_means should be a vector of length 2). Note that mean trait values may change as the population evolves, so the values in this vector define only the means of the initial population.

land_edge
: This sets what happens at the edge of the landscape. Three options are possible, including "torus", "leaky", and "reflect". A torus results in no edge, such that pests that leave one side of the landscape end up on the other side. A leaky edge causes pests to leave the landscape and the simulation entirely, so they are effectively no longer recorded. A reflective edge causes pests to bounce at the edge back in the direction from which they came. The default and recommended edge is a torus.

crop_growth
: This sets the amount by which landscape cell values increase for each crop from one time step to the next. This might be used to model the increase in biomass caused by plant growth on the landscape cell. The way in which growth is enacted is set by crop_growth_type. Values for crop_growth can either be a single number (in which case, the growth amount applies to all crop types) or a vector of the same length as crop_types (in which case, vector indices give the amount of growth for each crop number).

crop_growth_type
: This clarifies whether crop growth should be "none" (default), "geometric", or "linear". If geometric, then the value of a landscape cell increases to cell_value * (1 + crop_growth). If linear, then a fixed value crop_growth is added to the existing cell value. Note that for linear growth, this fixed value is only added if the existing cell value is greater than zero (else it is assumed that there is no crop to grow).

pesticide_threshold
: This specifies a pest density per cell above which farmers will apply pesticide. All other simulation conditions remain unaffected; farmers will just not apply

pesticide unless pest density exceeds the threshold. This parameter is 'none' (no threshold applied) by default. The argument also accepts a scalar value for densities applying to all farms, or a vector of length 'farms' to specify a unique threshold for each farm. If pest density falls below the threshold, pesticide application immediately stops.

`pesticide_delay`
: This specifies a delay between the time step at which pest threshold density is exceeded and pesticide is applied. For example if pest density exceeds the threshold set by 'pesticide_threshold' in time step 10, and 'pesticide_delay = 2', then pesticide will not be applied in response to the threshold being crossed until time step 12. If pest density falls below the threshold, pesticide application immediately stops (i.e., the delay is only in application, not ceasing use, to simulate the time needed to acquire and apply pesticide).

`population_filename`
: The name of the population results CSV file to be saved.

`last_step_filename`
: The name of the CSV file showing data for the last time step of the simulation.

`initial_inds`
: A set of individuals from the output of a previous run_farm_sim call to continue a simulation from rather than initialising a new population. Individuals are passed through exactly as they are. The NULL default builds individuals from scratch.

Value

The output in the R console is a list with three elements; the first element is a vector of parameter values used by the model, the second element is the landscape in the simulation, and the third element is a data frame of individuals and all of their traits in the last time step of the simulation run. Output will also be produced as CSV files within the working directory. When get_stats = TRUE, a file named 'population_data.csv' is produced in the working directory. When print_last = TRUE, a complete array of all individuals and their characteristics is printed for the last time step in the working directory in a file named 'last_time_step.csv' (for large simulations, this file can be > 1GB in size). When print_inds = TRUE, a complete array of all individuals in all time steps is produced in the working directory in a file named 'individuals.csv' (use this option with extreme caution for all but the smallest simulations).

Examples

```
gmt       <- matrix(data = 0, nrow = 4, ncol = 4);
diag(gmt) <- 1;
mg        <- mine_gmatrix(gmatrix = gmt, loci = 4, layers = 3, indivs = 100,
                          npsize = 100, max_gen = 2, prnt_out = FALSE);
sim       <- run_farm_sim(mine_output = mg, N = 100, xdim = 40, ydim = 40,
                          repro = "asexual", time_steps = 1,
                          print_inds = FALSE, print_gens = FALSE,
                          print_last = FALSE, get_stats = FALSE);
```

stress_test *Mine_gmatrix output stress*

Description

Run a diagnoistic test on the stress output of mine_gmatrix. The mine_gmatrix function produces a network from loci to traits for a pre-specified trait covariance structure. This covariance structure is estimated from network values, but can vary due to error in the loci values (standard random normal numbers). This function will test the stress of a network (mean deviation between the estimated covariance matrix and the pre-specified one) a given number of times to produce a distribution of stress estimates.

Usage

```
stress_test(mine_output, indivs = 1000, reps = 10)
```

Arguments

mine_output	The output from mine_gmatrix, which will be used to initialise the genomes and traits of pests.
indivs	The number of individuals to use in the stress test. Higher values produce a larger sample size for more accurate estimates of the true covariance structure produced by the network, and therefore the actual stress expected from it.
reps	The number of times a new set of individuals will be initialised for estimating the covariance between traits and calculating its stress.

Value

A vector of stress values the same length as 'reps'.

Examples

```
gmt       <- matrix(data = 0, nrow = 4, ncol = 4);
diag(gmt) <- 1;
mg        <- mine_gmatrix(gmatrix = gmt, loci = 4, layers = 3, indivs = 100,
                          npsize = 100, max_gen = 2, prnt_out = FALSE);
stresses  <- stress_test(mine_output = mg);
```

Index

mine_gmatrix, 2

run_farm_sim, 4

stress_test, 14