

evo.gen

Quick Reference Guide

version 0.0

Viktor Milkevych

17 April 2024

CONTENTS

CLASS POPULATION.....	4
Population().....	4
set_population	4
set_population	5
aging.....	5
size	6
capacity	6
clear	6
reshape.....	7
id_at.....	7
id_at.....	8
sire_at.....	8
sire_at.....	9
dame_at.....	9
dame_at.....	9
age_at	10
age_at	10
alive_at	11
alive_at	11
isgenotyped_at.....	12
isgenotyped_at.....	12
sex_at	12
sex_at	13
phenotype_at	13
phenotype_at	14
breedingvalue_at.....	14
breedingvalue_at.....	15
get_ld.....	15
get_popid	16
CLASS GROUP.....	17
Group.....	17

size	17
size_at	18
clear	18
remove.....	19
move.....	19
add	20
add	20
add	20
mate	21
mate	21
regroup_newborn	22
aging.....	22
genotype	23
kill	23
make_observation	24
make_observation	24
make_observation	25
make_observation	26
make_observation	27
select.....	27
select_into_group	28
CLASS TRAIT	31
Trait	31
Trait	31
set_trait	33
reset_trait	34
get_observations	36
get_observations	36
get_observations	37
get_observations	38
get_observations	39
clear	40
is_cleared	41

CLASS POPULATION

Population()

Description

Construct an empty Population object. This is a default class constructor with no parameters.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
```

void

set_population (*size_t* *nindividuals*,
 const std::string & *genotype_structure*,
 float *ref_allele_probability*,
 int *n_ploidy*)

Description

Stochastically generates an initial population with no LD. The method simulates a specified number of unrelated individuals with simulated genome of properties determined by the specific parameters of the method. In order to assign an LD structure for such population it should pass some rounds of random mating.

Parameters

nindividuals – a number of individuals in the population; positive integer.

genotype_structure – the file name where the genome structure is described.

ref_allele_probability – the frequency of reference allele at each of loci of the simulated genome; floating point number in the interval [0,1].

n_ploidy – the ploidy of the simulated genome; integer value.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 4)
```

void

```
set_population ( const std::string & haplotypes_fname,  
                 const std::string & genotype_structure,  
                 bool with_pedigree )
```

Description

Generates population using an existing data of haplotypes (simulated or real). The method uses a predefined (provided in a data file) haplotypes, genome structure and pedigree (optional).

Parameters

haplotypes_fname – the file name where the genome's haplotypes is provided.

genotype_structure – the file name where the genome structure is described.

with_pedigree – TRUE or FALSE indicating whether to include or to not include an initial pedigree information (provided in a haplotypes data file) in a generated population.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
```

```
pop.set_population("haplotypes_pop.dat", "gen_struct_pop.dat", TRUE)
```

void

```
aging ( int delta_t )
```

Description

Increase the age of every individual in a population:

$\text{new_age} = \text{current_age} + \text{delta_t}$.

Parameters

delta_t - the amount of time added to the current age of a particular individual; the time unit is arbitrary.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
```

```
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
```

```
pop.aging(3)
```

`size_t`
`size ()`

Description

Get the current size of a population in terms of number of active (alive) individuals.

Parameters

None.

Return value

Positive integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
n_individ = pop.size()
```

`size_t`
`capacity ()`

Description

Get the entire size of a population in terms of number of all individuals (alive + disabled).

Parameters

None.

Return value

Positive integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
n_individ = pop.capacity()
```

`void`
`clear ()`

Description

Removes all active individuals from a population and clears the allocated memory.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.clear()
```

void

reshape ()

Description

Optimizes the memory allocated to a population by clearing the memory allocated for inactive (disabled) individuals and rearranging the list of individuals in a population. After the call of this method the positions of individuals in the population might be changed. Therefore, any existing association with a specific Group (see the methods for the class GROUP) will be disabled.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
for i in range( pop.size ):
    if i%2 == 0:
        pop.alive_at(i, false) # disable every second individual in the population pop
pop.reshape()
```

void

id_at (size_t at, unsigned long id)

Description

Assign a specific identity number (ID) to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.

Parameters

at – the position of an individual in the population's list of individuals.
id – positive integer value will be assigned to the individual's ID.

Return value

None.

Example (use in Python)

```
import evogen
import random

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
for i in range( pop.size ):
    r_num = random.randint(0, 1000)
    pop.id_at(i, r_num)
```

[unsigned long](#)[id_at](#) ([size_t](#) *at*)**Description**

Get (retrieve) an identity number (ID) of a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

Positive integer value, an individual’s ID.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
id = pop.id_at(0) # ID of the individual at the position 0 in the population.
```

[void](#)[sire_at](#) ([size_t](#) *at*, [unsigned long](#) *id*)**Description**

Assign a sire to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.

Parameters

at – the position of an individual in the population’s list of individuals.

id – positive integer value will be assigned to the individual’s property named as sire.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.sire_at(0, 10089) # individual at the position 0 will get new sire with id 10089
```

unsigned long
sire_at (size_t at)

Description

Get (retrieve) a sire's identity number (ID) of a specific individual in a population.

Parameters

at – the position of an individual in the population's list of individuals.

Return value

Positive integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
sire_id = pop.sire_at(0) # sire's ID of the individual at the position 0 in the population.
```

void
dame_at (size_t at, unsigned long id)

Description

Assign a dame to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.

Parameters

at – the position of an individual in the population's list of individuals.

id – positive integer value will be assigned to the individual's property named as dame.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.dame_at(0, 10089) # individual at the position 0 will get new dame with id 10089
```

unsigned long
dame_at (size_t at)

Description

Get (retrieve) a dame's identity number (ID) of a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

Positive integer value.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
dame_id = pop.dame_at(0) # dame's ID of the individual at the position 0 in the population.
```

```
void
```

```
age_at ( size_t at, int age )
```

Description

Assign (modify) an age for a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals,

age – (positive or negative) integer value to be a new age of the individual.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.age_at(0, 23) # individual at the position 0 will get new age of 23.
```

```
int
```

```
age_at ( size_t at )
```

Description

Get (retrieve) an age of a specific individual in the population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

Positive or negative integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
age = pop.age_at(0) # age of the individual at the position 0.
```

void

alive_at (*size_t at*, *bool alive*)

Description

Set (modify) the living status of a specific individual in the population.

Parameters

at – the position of an individual in the population's list of individuals.
alive – TRUE or FALSE indicating the new status.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.alive_at(0, FALSE) # the individual at the position 0 is disabled.
```

bool

alive_at (*size_t at*)

Description

Get (retrieve) the living status of a specific individual in the population.

Parameters

at – the position of an individual in the population's list of individuals.

Return value

Boolean (TRUE or FALSE) value indicating the individual's status.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
is_active = pop.alive_at(0) # checking if the individual at the position 0 is active or disabled.
```

void
isgenotyped_at (*size_t at*, *bool genotyped*)

Description

Sets (modify) the genotyping status of a specific individual in the population.

Parameters

at – the position of an individual in the population’s list of individuals.
genotyped – TRUE or FALSE indicating the new status.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.isgenotyped_at(0, TRUE) # the individual at the position 0 is genotyped.
```

bool
isgenotyped_at (*size_t at*)

Description

Get (retrieve) the genotyping status of a specific individual in the population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

Boolean (TRUE or FALSE) value indicating the individual’s status.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
is_genotyped = pop.isgenotyped_at(0) # checking if the individual at the position 0 is genotyped.
```

void
sex_at (*size_t at*, *int sex*)

Description

Assign (modify) a sex of a specific individual in a population. This method overwrite the existing sex of an individual.

Parameters

at – the position of an individual in the population’s list of individuals,
sex – (positive/negative) integer value to be a new sex of the individual.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.sex_at(0, 1)
```

short

sex_at (*size_t at*)

Description

Get (retrieve) the sex of a specific individual in the population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

Positive or negative integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
sex = pop.sex_at(0) # gets a sex value of the individual at the position 0.
```

void

phenotype_at (*size_t at*, *pybind11::array_t<float> phen*)

Description

Assign (modify) phenotypic observations (trait values) for a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals,
phen – a python array (list or numpy array) of a floating point numbers representing phenotypic observations. The size of *phen* should be equal to the number of traits assigned to the population.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
obs = [ 40.0, 5.0, 0.5 ] # three traits observation
pop.phenotype_at(0, obs) # modify observations for the individual at the position 0.
```

```
pybind11::array_t<float>
phenotype_at ( size_t at )
```

Description

Get (retrieve) phenotypic observations (trait values) for a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals.

Return value

A numpy array of an individuals phenotype of floating point numbes.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
in_obs = [ 40.0, 5.0, 0.5 ] # three traits observation
pop.phenotype_at(0, in_obs) # modify observations for the individual at the position 0
out_obs = pop.phenotype_at(0) # get observations of the individual at the position 0.
```

```
void
breedingvalue_at ( size_t at, pybind11::array_t<float> bv )
```

Description

Assign (modify) breeding values for a specific individual in a population.

Parameters

at – the position of an individual in the population’s list of individuals.
bv – a python array (list or numpy array) of a floating point numbes of breeding values. The size of *bv* should be equal to the number of traits assigned to the population.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
bvs = [ 0.01, 0.05, -0.5 ] # three traits observation
pop.breedingvalue_at(0, bvs) # modify BVs for the individual at the position 0.
```

```
pybind11::array_t<float>
breedingvalue_at ( size_t at )
```

Description

Get (retrieve) breeding values for a specific individual in a population.

Parameters

at – the position of an individual in the population's list of individuals.

Return value

A numpy array of a breeding values of an individual. Floating point numbers.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
in_bvs = [ 0.01, 0.05, -0.5 ] # three traits observation
pop.breedingvalue_at(0, in_bvs) # modify BVs for the individual at the position 0.
out_bvs = pop.breedingvalue_at(0) # get BVs of the individual at the position 0.
```

```
void
get_ld (const std::string &out_file, bool full_info, int which_chr, unsigned int snp_step)
```

Description

Get (retrieve) linkage disequilibrium (LD) data for a population.

Parameters

out_file – the name of ASCII file where LD data will be written.
full_info – logical variable indicating if only the averaged LD data will be provided (*full_info* = *False*) or, besides the averaged LD, variants frequencies and LD matrix will be provided as well (*full_info* = *True*).
which_chr – specific chromosome number for which the LD should be calculated; if *which_chr* = -1, the LD data will be provided for all available chromosomes.
snp_step – SNP interval for calculating paired correlations; if *snp_step* = 1, the correlations will be calculated for each pair of snp in a chromosome; if *snp_step* = 2, the correlation will be calculated for each second pair of snp, etc.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop.get_ld("ld.txt", False, 0, 10) # prints only averaged LD to ld.txt for 1st chromosome with
correlation between every 10th snp
```

size_t

get_popid ()

Description

Get (retrieve) unique population ID number.

Parameters

None.

Return value

Positive integer.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
pop_id = pop.get_popid()
```

CLASS GROUP

Group()

Description

Construct an empty Group object. This is a default class constructor with no parameters.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
grp = evogen.Group() # create the class instance
```

[size_t](#)

size()

Description

Return the number of distinct populations consisting the group.

Parameters

None.

Return value

Positive integer value.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
sz = grp.size() # will return 1
```

`size_t`
`size_at(size_t at)`

Description

Returns the number of individuals in the group belonging to a population at the position `at` in the group.

Parameters

`at` – position of a specific population in the group.

Return value

Positive integer value.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
sz = grp.size_at(0) # will return 5
```

`void`
`clear()`

Description

Clears the group and releases the occupied memory.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.clear()
```

void

remove ()

Description

For those individuals assigned to a calling group removes them from their original populations and from the group. This method also clears the calling group.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.remove( ) # the group will be empty as well as the population pop
```

void

move (*Population & pop*)**Description**

Relocate all individuals assigned to the calling group to the population *pop*. The relocated individuals will be removed from their original populations.

Parameters

pop – the instance of a Population class.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop1 = evogen.Population() # create the class instance
pop2 = evogen.Population() # create the class instance
pop1.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add(pop1) # add individuals from the population pop1 to the group grp.
grp.move(pop2) # relocate all individuals from the population pop1 to the population pop2.
```

void
add (*Population* & *pop*)

Description

Adds (assign) all individuals in the population *pop* to the group. The assigned individuals will not change their location and population status (they will still belong to the population *pop*).

Parameters

pop – the instance of a Population class.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp.
```

void
add (*Population* & *pop*, *size_t* *which_one*)

Description

Adds the specific individual (determined by its position *which_one* in the population) from the population *pop* to the group.

Parameters

pop – the instance of a Population class.

which_one – position of an individual in the population *pop*; positive integer value.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop, 1 ) # add individual at position 1 in the population pop to the group grp.
```

void
add (*Group* & *grp*)

Description

Adds all individuals assigned to the group *grp* to the calling group. The assigned individuals will not change their memory location and population status (belonging).

Parameters

grp – the instance of a Group class.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop, 1 ) # add individual at position 1 in the population pop to the group grp.
grp2 = evogen.Group() # create the class instance
grp2.add( grp )
```

void

mate ()

Description

Initiates random mating for all individuals assigned to the group. The method uses sexual reproduction with two offspring per dame and the success rate of having exactly two offsprings of 0.8.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp.
grp.mate()
```

void

mate (*bool sexual_reproduction*, *int max_offspring*, *float success_rate*)

Description

Initiates random mating for all individuals in the group.

Parameters

sexual_reproduction – TRUE or FALSE indicating the use of sexual reproduction.
max_offspring – maximal number of offsprings per dame. Integer value.
success_rate – the success rate of having exactly two offsprings. Floating point number in the range [0,1].

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp.
grp.mate(TRUE, 2, 0.8)
```

void

regroup_newborn (*Group* & *grp*)

Description

Move (relocate) all new-born individuals assigned to the calling group to another group *grp*. The individuals will be cleared from the calling group but will retain the connections to their original populations.

Parameters

grp – the instance of a Group class.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp.
grp.mate(TRUE, 2, 0.8)
grp2 = evogen.Group() # create the class instance
grp.regroup_newborn( grp2 ) # move these new-born to the new group grp2.
```

void

aging (*int* *delta_t*)

Description

Adds *delta_t* to existing values of age for every individual assigned to the calling group.

Parameters

delta_t – Integer value by which a specific age will be changed: `new_age = current_age + delta_t`.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.aging(2)
```

void

genotype()

Description

Change a genotyped status (makes it TRUE) for each individual assigned to the calling group.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.genotype()
```

void

kill()

Description

Disables all individuals assigned to the calling group. This method changes the alive status of an individual to FALSE in their original populations.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.kill()
```

void

make_observation (Trait & trt, pybind11::array_t<float> env)
Description

Collect phenotypic observations (calculate trait values) for all individuals assigned to the calling group.

Parameters

trt – the instance of the Trait class.

env – the array (list or numpy array) determining environmental conditions for an observing trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
trt = evogen.Trait( ... some parameters ... ) # create the trait object
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
env = [ 0.5, 0.0, 0.1 ] # three correlated traits
grp.make_observation(trt, env)
```

void

make_observation (Trait & trt,
 pybind11::array_t<float> env,
 const std::string & trvalues)
Description

Collect phenotypic observations (calculate trait values) for all individuals assigned to the calling group and write out the results to a file.

Parameters

trt – the instance of the Trait class.

env – the array (list or numpy array) determining environmental conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.

trvalues – file name for output of trait values.

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
trt = evogen.Trait( ... some parameters ... ) # create the trait object
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
env = [ 0.5, 0.0, 0.1 ] # three correlated traits
grp.make_observation(trt, env, "observations.txt")
```

void

```
make_observation ( Trait & trt,
                  pybind11::array_t<float> env,
                  const std::string & trvalues,
                  const std::string & genotypes )
```

Description

Collect phenotypic observations (calculate trait values) for all individuals assigned to the calling group and write out the results in terms of calculated trait values to a file as well as the genotypes of measured individuals to the separate file.

Parameters

trt – the instance of the Trait class.

env – the array (list or numpy array) determining environmental conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.

trvalues – file name for output of trait values.

genotypes – file name for output of individual genotypes.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
trt = evogen.Trait( ... some parameters ... ) # create the trait object
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
env = [ 0.5, 0.0, 0.1 ] # three correlated traits
grp.make_observation(trt, env, "observations.txt", "genotypes.txt")
```

void

make_observation (*Trait* & *trt*,
 pybind11::array_t<float> *env*,
 pybind11::array_t<float> *trvalues*)

Description

Collect phenotypic observations (calculate trait values) for all individuals assigned to the calling group and write out the results to a python array.

Parameters

trt – the instance of the Trait class.

env – the array (list or numpy array) determining environmental conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.

trvalues – numpy array of type float32 and arbitrary shape for output of trait values..

Return value

None.

Example (use in Python)

```
import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
trt = evogen.Trait( ... some parameters ... ) # create the trait object
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
env = [ 0.5, 0.0, 0.1 ] # three correlated traits
obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importance of the correct type!
grp.make_observation(trt, env, obs)
```

void

```
make_observation ( Trait & trt,
                  pybind11::array_t<float> env,
                  pybind11::array_t<float> trvalues,
                  pybind11::array_t<int> genotypes )
```

Description

Collect phenotypic observations (calculate trait values) for all individuals in the calling group and write out the results in terms of calculated trait values to a python array as well as the genotypes of measured individuals to the separate python array.

Parameters

trt – the instance of the Trait class.

env – the array (list or numpy array) determining environmental conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.

trvalues – numpy array of type *float32* and arbitrary shape for output of trait values.

genotypes – numpy array of type *int32* and arbitrary shape for output of individual genotypes.

Return value

None.

Example (use in Python)

```
import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
trt = evogen.Trait( ... some parameters ... ) # create the trait object
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
env = [ 0.5, 0.0, 0.1 ] # three correlated traits
obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importance of correct type!
gen = np.zeros( shape=(1), dtype=np.int32 ) # note, the importance of correct type!
grp.make_observation (trt, env, obs, gen)
```

void

```
select (const std::string category, const std::string mode, int num, float val)
```

Description

Select individuals in the calling group by specified category and mode. After the call of this method a calling group will consist only of individuals that satisfy conditions within a specified category. Consecutive calls of this method for the same group but with different selection options (changes in specified category for instance) will gradually reduce a number of individuals in a calling group until it will consist only of individuals satisfied all applied selection requirements.

Parameters

category – parameter specifying a selection category; a string variable. The parameter defines the category (based on specific individual's property) by which the selection should be based. Available options for the *category* parameter:

phen (select by phenotype);

bv (select by breeding value);
 age (select by age);
 sex (select by sex);
 alive (select by live status);
 active (select by active status);
 genot (select by genotyping status);
 id (select by individual's identifier)

mode –the mode of selection within the specified category; a string variable. Available options for the *mode* parameter:

rand (random selection);
 best (selection of individuals having highest values of individual's property specified by the *category* parameter);
 worst (selection of individuals having smallest values of individual's property specified by the *category* parameter).

num –parameter specifying an upper bound of a number of candidates to be selected; an integer variable. A number of actually selected is within the range [0, num].

val –auxiliary parameter; an floating point number. Specific input options for the *val* parameter depends on specified category. If a category parameter points to a logical property (sex, alive, active, genot), *val* = {0, 1}. For a category parameter pointing to a vector-type continuous numerical property (phen, bv):

val = -1 determines the use of averaged (among vector elements) value of specified property;

val = [0, n] determines which element of selected properties vector to be used for selection, for example if multiple correlated traits is used phen and bv are vectors, so setting *val* = 0 means a very first element of a specific property will be used; here n is a size of vector.

If the selection category is *id*, *val* can be arbitrary integer number.

For a category parameter pointing to a single-valued continuous numerical property (age, agr):

if *val* = -1 or *mode* = *rand* the *val* parameter has no effect;

if *val* get values other than -1: the parameter determines the lower bound of selected values when *mode* = *best*, or upper bound of selected values if the *mode* = *worst*; for instance if individuals above the age of 10 need to be selected, *val* = 10 and *mode* = *best*; if to select all available below 5, *val* = 5 and *mode* = *worst*.

Return value

None. A calling group will be updated.

Example (use in Python)

```

import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.select("sex", "rand", 25, 1) # selecting 25 males from pop randomly
grp.select("phen", "best", 10, -1) # then among selected males selecting 10 that have best phenotypes
  
```

void

select_into_group (Group &grp, const std::string category, const std::string mode, int num, float val)

Description

Select individuals from the calling group into another group *grp* by specified category and mode. After the call of this method a calling group will consist only those individuals that do not satisfy

conditions within a specified category; in the same time the group *grp* will consist individuals that do satisfy conditions within a specified category. This is the only difference between this method and the other selection method named *select(...)*.

Parameters

grp – the object of class Group where all selected individuals will be relocated;

category – parameter specifying a selection category; a string variable. The parameter defines the category (based on specific individual's property) by which the selection should be based. Available options for the *category* parameter:

- phen (select by phenotype);
- bv (select by breeding value);
- age (select by age);
- sex (select by sex);
- alive (select by live status);
- active (select by active status);
- genot (select by genotyping status);
- id (select by individual's identifier)

mode – the mode of selection within the specified category; a string variable. Available options for the *mode* parameter:

- rand (random selection);
- best (selection of individuals having highest values of individual's property specified by the *category* parameter);
- worst (selection of individuals having smallest values of individual's property specified by the *category* parameter).

num – parameter specifying an upper bound of a number of candidates to be selected; an integer variable. A number of actually selected is within the range [0, num].

val – auxiliary parameter; an floating point number. Specific input options for the *val* parameter depends on specified category. If a category parameter points to a logical property (sex, alive, active, genot), *val* = {0,1}. For a category parameter pointing to a vector-type continuous numerical property (phen, bv):

val = -1 determines the use of averaged (among vector elements) value of specified property;

val = [0,n] determines which element of selected properties vector to be used for selection, for example if multiple correlated traits is used phen and bv are vectors, so setting *val* = 0 means a very first element of a specific property will be used; here n is a size of vector.

If the selection category is *id*, *val* can be arbitrary integer number.

For a category parameter pointing to a single-valued continuous numerical property (age, agr):

if *val* = -1 or *mode* = *rand* the *val* parameter has no effect;

if *val* get values other than -1: the parameter determines the lower bound of selected values when *mode* = *best*, or upper bound of selected values if the *mode* = *worst*; for instance if individuals above the age of 10 need to be selected, *val* = 10 and *mode* = *best*; if to select all available below 5, *val* = 5 and *mode* = *worst*.

Return value

None. A calling group will be updated.

Example (use in Python)

```
import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
grp = evogen.Group() # create the class instance
grp2 = evogen.Group() # create the class instance
grp.add( pop ) # add population pop to the group grp
grp.select into_group(grp2,"sex", "rand", 25, 1) # selecting 25 mails from pop randomly into grp2
```

CLASS TRAIT

Trait()

Description

Construct an empty Trait object. This is a default class constructor with no parameters.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen
```

```
T = evogen.Trait()
```

```
Trait ( Population & pop,
        pybind11::array_t<float> trmean,
        pybind11::array_t<float> qtl_prop_chrom,
        pybind11::array_t<float> corr_g,
        pybind11::array_t<float> varr_g,
        pybind11::array_t<float> corr_e,
        pybind11::array_t<float> varr_e,
        pybind11::array_t<float> envr,
        size_t dist_model,
        pybind11::array_t<float> dist_par )
```

Description

Construct an object of class Trait (create the class instance) and determines the specific configuraton of correlated traits. None-default class constructor. Note, the default constructor should not be called in this case.

Parameters

pop – instance of the Population class.

trmean – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.

qtl_prop_chrom - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

corr_g - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_g - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

corr_e - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_e - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

envr - python array (list or numpy array) of floating point numbers representing environmental conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

dist_mode – positive integer value {1,2,3} indicating the mode of sampling dominance effects;

dist_mode = 1 – sampling from a Uniform distribution,

dist_mode = 2 – sampling from a Normal distribution,

dist_mode = 3 – sampling from a Gamma distribution.

dist_par – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;

if *dist_mode* = 1, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distribution;

if *dist_mode* = 2, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;

if *dist_mode* = 3, *dist_par* = [*a*], where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
```

void

```

set_trait ( Population & pop,
            pybind11::array_t<float> trmean,
            pybind11::array_t<float> qtl_prop_chrom,
            pybind11::array_t<float> corr_g,
            pybind11::array_t<float> varr_g,
            pybind11::array_t<float> corr_e,
            pybind11::array_t<float> varr_e,
            pybind11::array_t<float> envr,
            size_t dist_model,
            pybind11::array_t<float> dist_par )

```

Description

Determines the specific configuraton of correlated traits. The default Trait class constructor should be called before this method can be invoked.

Parameters

pop – instance of the Population class.

trmean – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.

qtl_prop_chrom - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

corr_g - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_g - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

corr_e - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_e - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

envr - python array (list or numpy array) of floating point numbers representing enviroental conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

dist_mode – positive integer value {1,2,3} indicating the mode of sampling dominance effects;

dist_mode = 1 – sampling from a Uniform distribution,

dist_mode = 2 – sampling from a Normal distribution,

dist_mode = 3 – sampling from a Gamma distribution.

dist_par – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;

if *dist_mode* = 1, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distributon;

if *dist_mode* = 2, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;

if *dist_mode* = 3, *dist_par* = [*a*], where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [ 1.0, 0.5, 0.7 ], [ 0.5, 1.0, 0.2 ], [ 0.7, 0.2, 1.0 ] ] # (3) genomic correlations
cor_e = [ [ 1.0, 0.3, 0.5 ], [ 0.3, 1.0, 0.4 ], [ 0.5, 0.4, 1.0 ] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor, create class instance
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
```

void

```
reset_trait ( Population & pop,
              pybind11::array_t<float> trmean,
              pybind11::array_t<float> qtl_prop_chrom,
              pybind11::array_t<float> corr_g,
              pybind11::array_t<float> varr_g,
              pybind11::array_t<float> corr_e,
              pybind11::array_t<float> varr_e,
              pybind11::array_t<float> envr,
              size_t dist_model,
              pybind11::array_t<float> dist_par )
```

Description

Redetermines the specific configuraton of correlated traits. This method should be applied to already existant and defined instance of the Trait class.

Parameters

pop – instance of the Population class.

trmean – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.

qtl_prop_chrom - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

corr_g - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_g - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

corr_e - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].

varr_e - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.

envr - python array (list or numpy array) of floating point numbers representing environmental conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].

dist_mode – positive integer value {1,2,3} indicating the mode of sampling dominance effects;

dist_mode = 1 – sampling from a Uniform distribution,

dist_mode = 2 – sampling from a Normal distribution,

dist_mode = 3 – sampling from a Gamma distribution.

dist_par – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;

if *dist_mode* = 1, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distribution;

if *dist_mode* = 2, *dist_par* = [*a*, *b*], where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;

if *dist_mode* = 3, *dist_par* = [*a*], where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
qtl_prop = [ 0.5, 0.6, 0.1, 0.9 ] # make changes to the parameter, then reset trait
T.reset_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
```

void

get_observations (*Population* & *pop*, pybind11::array_t<float> *env*)

Description

Collects observations (calculates traits values) for every individual in a specific population.

Parameters

pop – instance of Population class for which observations should be collected.

env – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].

Return value

None.

Example (use in Python)

```
import evogen
```

```
pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) enviironment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, mdodel 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
env = [ 0.1, 0.5, 0.0 ]
T.get_observations(pop, env)
```

void

get_observations (*Population* & *pop*,
pybind11::array_t<float> *env*,
pybind11::array_t<float> *t*)

Description

Collects observations (calculates traits values) for every individual in a specific population.

Parameters

pop – instance of Population class for which observations should be collected.
env – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].
t – numpy array of type float32 and arbitrary shape for output of trait values.

Return value

None.

Example (use in Python)

```
import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importance of correct type!
env = [ 0.1, 0.5, 0.0 ]
T.get_observations(pop, env, obs)
```

void

get_observations (*Population* & *pop*,
 pybind11::array_t<float> *env*,
 pybind11::array_t<float> *t*,
 pybind11::array_t<int> *g*)

Description

Collects observations (calculates traits values) for every individual in a specific population.

Parameters

pop – instance of Population class for which observations should be collected.
env – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].
t – numpy array of type float32 and arbitrary shape for output of trait values.
g – numpy array of type int32 and arbitrary shape for output of individual genotypes.

Return value

None.

Example (use in Python)

```

import evogen
import numpy as np

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) enviironment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, mdodel 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typepe!
env = [ 0.1, 0.5, 0.0 ]
obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typepe!
gen = np.zeros( shape=(1), dtype=np.int32 ) # note, the importans of correct typepe!
T.get_observations(pop, env, obs, gen)

```

void

```

get_observations ( Population & pop,
                  pybind11::array_t<float> env,
                  const std::string & t )

```

Description

Collects observations (calculates traits values) for every individual in a specific population.

Parameters

pop – instance of Population class for which observations should be collected.
env – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].
t – file name for output of trait values.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
env = [ 0.1, 0.5, 0.0 ]
obs = "observations.txt"
T.get_observations(pop, env, obs)
```

void

```
get_observations ( Population & pop,
                  pybind11::array_t<float> env,
                  const std::string & t,
                  const std::string & g )
```

Description

Collects observations (calculates traits values) for every individual in a specific population.

Parameters

pop – instance of Population class for which observations should be collected.
env – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].
t – file name for output of trait values.
g – file name for output of individual genotypes.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) residual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) environment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, model 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, mode, k_range_U)
env = [ 0.1, 0.5, 0.0 ]
obs = "observations.txt"
gen = "genotypes.txt"
T.get_observations(pop, env, obs, gen)
```

void

clear()

Description

Clears a specific trait object's set-up, and release the occupied memory.
In order to reuse the cleared trait object the method `reset_trait(...)` should be called.

Parameters

None.

Return value

None.

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) enviironment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, mdodel 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, which_model, k_range_U)
T.clear()
```

bool

is_cleared ()

Description

Checks if the trait object is cleared.

Parameters

None.

Return value

Boolean value (TRUE or FALSE).

Example (use in Python)

```
import evogen

pop = evogen.Population() # create the class instance
pop.set_population(5, "gen_struct_pop.dat", 0.4, 2) # simulate population
tr_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means
qtl_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls
qtl_prop = [ 1.0, 1.0, 1.0, 1.0 ]
cor_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations
cor_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations
var_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances
var_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances
env = [ 0.0, 0.0, 0.0 ] # (7) enviironment
# (i) For uniform distribution, model 1
k_range_U = [ -1.0, 1.0 ] # range of k parameter
# (ii) For normal distribution, model 2
```

```
k_range_N = [ 0.0, 0.5 ] # mean & std
# (iii) For gamma distribution, mdodel 3
k_range_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution
mode = 1
T = evogen.Trait() # default constructor
T.set_trait(pop, tr_mean, qtl_prop, cor_g, var_g, cor_e, var_e, env, which_model, k_range_U)
T.clear()
status = T.is_cleared() # will return TRUE
```

Description

Parameters

Return value

Example (use in Python)
