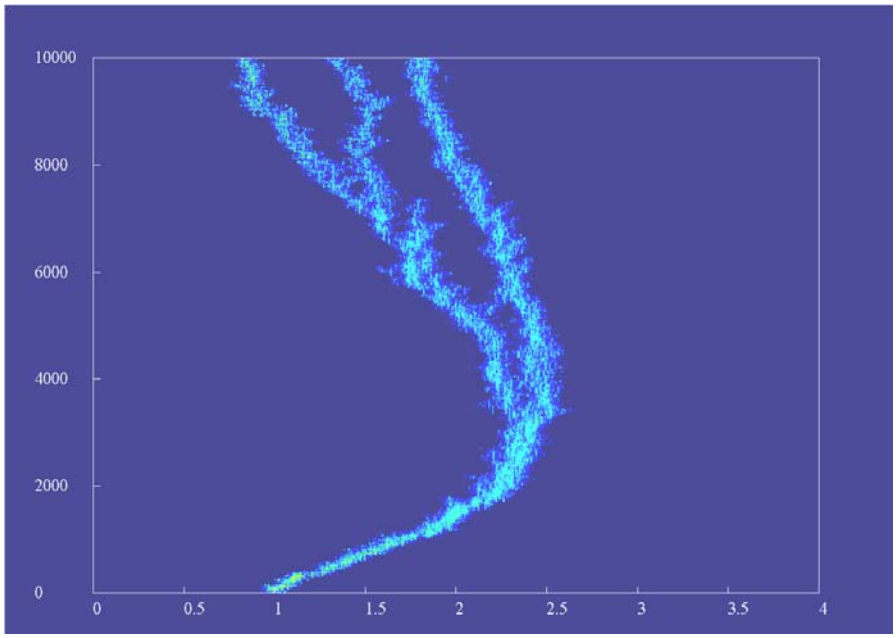
ZEN

ECO-EVOLUTIONARY SOFTWARE

REFERENCE MANUAL



http://www.biologie.ens.fr/ecologie/ecoevolution/legendre/legendre/zen.html

CONTENTS OF ZEN REFERENCE MANUAL

CONTENTS OF ZEN DISTRIBUTION

zen.exe	ZEN program
qtintf.dll	runtime library
zenref.pdf	reference manual
	model files: population dynamics
allee.zen	probability of mating (Møller & Legendre 2001)
arg_esd.zen	extinction dynamics for the spider *Argiope argentata*
met_esd.zen	extinction dynamics for the spider *Metepeira datona*
pass_2s.zen	2-sex model for passerine (Legendre et al. 1999)
regis.zen	chaotic dynamics
	model files: evolutionary dynamics
cicadas8.zen	prime numbers in cicadas
ferriere.zen	mutualistic interactions (Ferrière et al. 2002)
ferriere1.zen	discretization of continuous time version
kisdi.zen	asymmetric competition (Kisdi & Geritz 2001)
nirma.zen	adaptive radiation in bacteria
ravigne.zen	habitat selection (Ravigné 2001)

INSTALL

- Download self-extracting archive *autozen.exe* from website.
- Double-click on *autozen.exe* to expand it inside the *zen* directory.
- Double-click on any model file **.zen* files, open it with the ZEN program *zen.exe*.
- Drag and drop any model file **.zen* to the *zen.exe* icon will run the ZEN program with the file as input.

→ For PC/Linux and MAC/OSX versions, see p. 47.

Cover. Figure 1. Deployment of polymorphism in a population subjected to asymmetric competition. The distribution of phenotypes is shown along evolutionary time in ordinates. ZEN simulation, discrete version of a model from Kisdi & Geritz 2001.

ZEN

ECO-EVOLUTIONARY SOFTWARE

The ZEN computer program is designed to study population dynamics models, with an evolutionary component in the line of the theory of adaptive dynamics. ZEN handles discrete time deterministic or stochastic relations with discrete or continuous variables. Its evolutionary mechanism allows to explore polymorphism in populations from an ecological point of view.

ZEN performs similarly to an individual-based software: it is in fact 'phenotype-based'. The kernel is a symbolic evaluator handling multivalued variables. The ZEN program is compatible with the ULM program: it lacks the matrix formalism of ULM, but relation-type ULM models can be run with ZEN.

The biological system under study is described in a text file, the model file, using a reduced declaration language, and appropriate mathematical functions. When ZEN is run with the model file as input, the model can be studied interactively, producing numerical results and graphical representations.

Author
Stéphane Legendre
Laboratoire d'Ecologie, Ecole Normale Supérieure
46 rue d'Ulm, 75230 Paris Cedex 05
France
legendre@ens.fr

Contribution
Nicolas Champagnat, Guillaume Chapron, Jean Clobert, Régis Ferrière, Mathias Gauduchon
Isabelle Olivieri, Virginie Ravigné, Ophélie Ronce, Minus van Baalen

Funding
Action Incitative Bioinformatique
CNRS

1. THEORETICAL FRAMEWORK

Ecology and evolution

Ecological models with an evolutionary component allow to study several biological phenomena, like sexual selection, host-pathogens interactions, coevolution of plants and pollinators, mimicry, development of the immune system, evolution of cooperation, evolution of life history traits, and more generally biodiversity and speciation.

The first evolutionary models in biology are verbal arguments, like Fisher's proof that the primary sex ratio is 1/2. Theoretical evolutionary models started to develop in the 80's with quantitative genetics (Lande 1982) and game theory (Maynard Smith 1982). In this context originated the important notion of Evolutionary Stable Strategy (ESS), which had premises in the work of Hamilton (1963) on kin selection.

Adaptive dynamics

The recent theory of adaptive dynamics (Metz et al. 1996, Dieckmann & Law 1996) is based on population dynamics. It incorporates implicitly a hereditary mechanism, and allows studying the evolution of continuous phenotypic traits under the process of mutation-selection. Selection operates via ecological interactions. The evolutionary outcome of the system is determined from the persistence of rare mutants with trait s' in a resident population with trait s via the fitness gradient

$$\left.\frac{\partial f(s',s)}{\partial s'}\right|_{s'=s}.$$

The fitness function f is computed from the ecological setup.

A great achievement of the theory is the classification of the possible evolutionary outcomes (Geritz et al. 1998), making a synthesis of previous results. For example the notion of CSS (Continuously Stable Strategy, Eshel and Motro 1981) and ESS could be unified in a single framework. A fascinating eventuality is that of evolutionary branching (Geritz et al. 1998), which stems from the biological nature of the models, and does not appear in physical systems. In this situation, ecological constraints lead to disruptive selection, and possibly to sympatric speciation (Maynard Smith 1966, Benton & Pearson 2001), by means of assortative mating (Dieckmann & Doebeli 1999).

In the context of adaptive dynamics, it has been shown by Dieckmann that, on the way to the evolutionary attractor, evolution can be assumed to proceed by trait substitution, the population remaining monomorphic in each trait. A canonical (deterministic) equation accounting for the evolutionary dynamics of traits can be built. The canonical equation has been placed in a general mathematical framework by Nicolas Champagnat (Champagnat et al. 2001).

However, in the case of evolutionary branching, no equation is known that accounts for the dynamics of the polymorphic population.

Individual-based models

Individual-based models are appealing in biology because they have the potential to simulate biological phenomena finely. However, without a strong theoretical background their results can be difficult to interpret, as a consequence of their inherent stochasticity. Moreover, they can be memory and time consuming. Another drawback is that they are usually specific: developing 'generic' individual-based programs is a difficult task.

Individual-based programs have been used successfully in adaptive dynamics (Dieckmann & Law 1996), using a different method than that of ZEN.

The advantage of using simulations is that limitations in the theory of adaptive dynamics can be relaxed. In the ZEN framework:

- mutations need not be small,
- there are no restrictions on their distributions,
- polymorphism is fully accounted for,
- no fitness function is necessary.

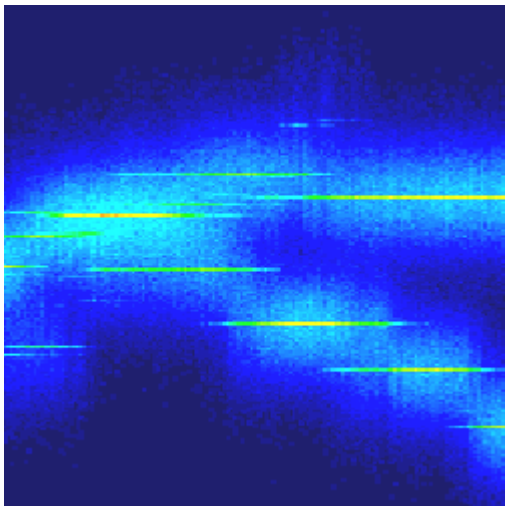


Figure 2. Close up of an evolutionary branching showing the jumps in adaptive trait (evolutionary time in abscissas, model from Ravigné 2000).

The ZEN program

ZEN simulates the evolution of populations under the mutation-selection process.

Ingredients operated by the ZEN program are:
- Stochastic equations in discrete time describing the dynamics of finite populations,
- Adaptive traits and their mutations (mutation rates, distributions, occurrences),
- Ecological interactions between phenotypes.

The last 2 ingredients may be omitted, in which case population dynamics are simulated with no evolutionary component.

Phenotypes

The evolutionary component of ZEN is 'phenotype-based'. A phenotype is a set of individuals sharing the same values of their adaptive traits. Within a phenotype individuals are identical. We say that a population is monomorphic for a trait if the distribution of the trait across phenotypes is unimodal, and polymorphic if the distribution is multimodal.

In ZEN, a set of phenotypes is called a group. At initialization, groups are strictly monomorphic, containing a single phenotype (with several individuals). During the ZEN simulation, mutant phenotypes created by the triggering of mutations interact with resident phenotypes. They persist or go extinct by the play of ecological interactions, possibly leading to evolutionary branching and polymorphism (see cover Fig. 1, and Fig. 2).

How ZEN works

In population dynamics, the population vector N is updated from one time step to the next using discrete relations:

$$N' = F(N)\,.$$

In evolutionary dynamics, the function F depends on a set S of adaptive traits:

$$N' = F_S(N)\,.$$

In the ZEN evolutionary formalism, a group of phenotypes is defined by a set of relations and adaptive traits. Let's assume for simplicity that there is only one adaptive trait s, and a single relation:

$$n' = f_s(n)\,.$$

Within the group, there is a description of the way mutations in the adaptive trait occur. As the ZEN simulation proceeds, phenotypes are created and destroyed. When there are P phenotypes with traits $s^{(1)}$, …, $s^{(P)}$, each phenotype (i) is driven by his own relation:

$$n^{(i)\prime} = f_{s^{(i)}}(n^{(i)})\,,$$

with $n^{(i)}$ the number of individuals in the phenotype, and $s^{(i)}$ the value of the trait in the phenotype. The relation-variable n is multivalued, with values $n^{(1)}$, …, $n^{(P)}$. The above phenotype-specific relations are not independent but linked by ecological interactions. Typically, the population size $n^{(i)}$ in each phenotype depends on the population size in other phenotypes in the same group, or in other groups.

The ZEN process

The mutation procedure has 2 important parameters, the mutation rate μ and the standard deviation σ in the distribution of mutations. These parameters influence the evolutionary time **t**, and also the execution time:

$$\mathbf{t} \text{ is proportional to } \frac{1}{\mu},$$

$$\mathbf{t} \text{ is proportional to } \frac{1}{\sigma^2}.$$

For example, multiplying σ by 2 divides evolutionary time by 4, phenotypes making larger incursions in the evolutionary space. In this case, the execution time should be reduced because the computations are roughly the same, while the simulation needs to be run for a shorter period. On the contrary, multiplying μ by 2 divides evolutionary time by 2, but should not reduce execution time. Indeed, more mutants are created, resulting in more computations. The mutation procedure is described precisely in section 3.

The process operated by ZEN is a form of branching process: it is density-dependent (though this is not requested) in order to avoid population sizes to blow, and it is bifurcating because of the mutations. This process has not been studied mathematically.

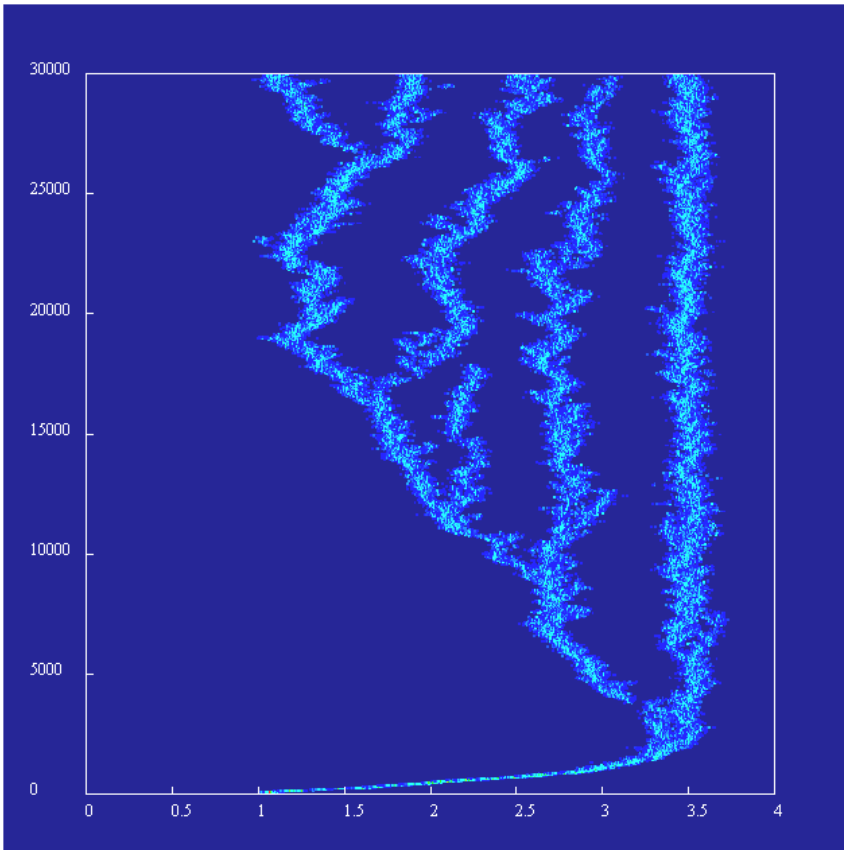
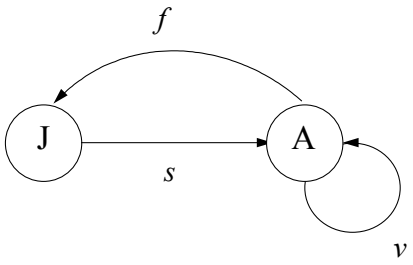


Figure 3. Same model as in Fig. 1, with a different competition function (discrete version of a model from Kisdi & Geritz 2001).

2. FIRST STEPS

Population dynamics

Population dynamics models in discrete time are based on the life cycle graph. The life cycle is a macroscopic description of an organism within a population. It incorporates the genotype and part of the phenotype.



In this (female-based) example, there are 2 age classes, juveniles and adults. Juveniles survive to the adult stade with survival rate s. Adults survive with adult survival rate v, and reproduce with fertility rate f. Let n_1 be the number of juveniles, and n_2 the number of adults. Relations can be written, relating population size from one time step to the next:

(1a) $$n_1' = fn_2$$

(1b) $$n_2' = sn_1 + vn_2 .$$

Relation (1b) means that the number of adults at the next time step is obtained from the number of juveniles surviving with rate s, plus the number of adults surviving with rate v. These relations can be put in matrix form and relevant informations about the dynamics (growth rate, population structure, elasticities) can be retrieved (Caswell 2000), as is done in the ULM program (Legendre & Clobert 1995, Ferrière et al. 1996).

Example: model file *regis.zen*

- Drag and drop model file *regis.zen* to the *zen.exe* icon. The file appears in the *Model file* window:

```
defmod regis(2)
rel: r1,r2

defrel r1
n1 = f*n2

defrel r2
n2 = s*n1 + v*n2
```

These declarations correspond to relations (1ab).
The declaration language is described in section 3.

Some more declarations are commented below:

```
defvar n = n1 + n2                    { total population size n
defvar r = 115                        { basic fertility r
defvar f = r*exp(-0.01*(n1+n2))       { fertility f regulated by density
```

- Click button to 'compile' the file, that is to translate it into internal representation.
- Click button to run the model (100 time steps by default). Population trajectories are displayed in the graphic window #1: *n*1 and *n*2 along time **t**.
- Click button to open graphic window #2. Click button in window #2 to set the graphics. Change *t* to *n*1, *n*1 to *n*2, and remove *n*2. Select *line off* in the *General* panel. Click OK.
- Click button in the main window to initialize the system (**t** = 0). Select option *Run* | *Settings*. Change *Number of time steps* to 10000, change *Dt text interp* to 1000, click OK. Click to run the model for 10000 time steps, with output in the main window every 1000 times steps. A strange attractor appears in window #2 (Fig. 4), with the corresponding population trajectories in window #1.
- Click button to view the variables. Change the *expression* of variable *r* to 50. Type <return>. The system is initialized (**init**). Click the *Run* button . A single point equilibrium is displayed.

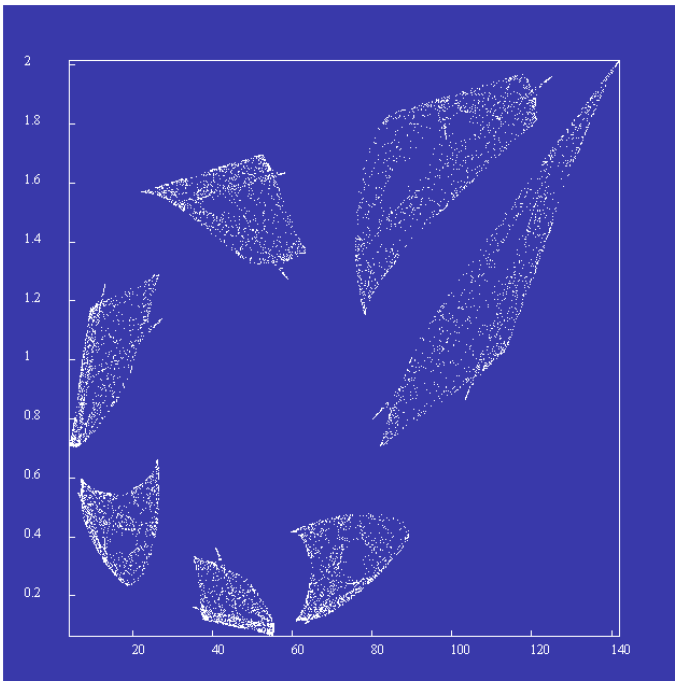


Figure 4. Density-dependent population dynamics produce this strange attractor (model from Ferrière 1992).

Demographic stochasticity

Demographic stochasticity is inherent to the demographic process. Its effects are more important when population size is small. Using the previous example, it is modeled by building a branching process on the basic relations (1ab), using integer-valued distributions:

(2a) $$n_1' = poisson(n_2, f)\,,$$

(2b) $$n_2' = binom(n_1, s) + binom(n_2, v)\,.$$

Relation (2a) means that the number of juveniles at the next time step is obtained by summing f samples of the Poisson distribution. In relation (2b) the number of survivors is computed using binomial distributions.

This modeling of demographic stochasticity gives an individual-based feature to the simulation. Individuals are not distinguished by their demographic parameters, which keep their average values, but the fate of individuals is taken into account, via the chance realization of these average parameters.

Example: model file *pass_2s.zen*

This is a 2-sex model for passerine (Legendre et al. 1999). Demographic stochasticity is modelled in the same way as in relations (2ab).

- Click  . Type ‘graph t n’ and <return> in the *Interp* panel of the main window to parameterize the graphics in window #1 (population size *n* as a function of time **t**). Type ‘yscale 0 400’ to fix the bounds on the Y axis. Type ‘addgraph’ to superimpose graphics.
- Type ‘run 50 10’ to run the model for 50 time steps with output every 10 time steps. The stochastic trajectory appears in graphic window #1.
- Type ‘init 2’ in the *Interp* panel. The system is initialized (**init**), with *seed* = 2. Now type ‘run’. Another realization of the process is run for 50 time steps (the previous trajectory corresponded to *seed* = 1). The trajectory appears superimposed in the graphic window. Type ‘init 3’, ‘init 4’, … to produce other trajectories.
- Type ‘montecarlo 50 1000’ to run the Monte Carlo simulation (50 time steps, 1000 trajectories). The mean trajectory is displayed. The probability of extinction along time appears in the main window.

The previous commands could have been executed by appropriate clicking. See section 4 for the list of available commands.

- Click  in graphics window to set graphics. Unselect option *superimpose* in the *General panel*. Select options *MinMax* and *2 sigma* in the *MonteCarlo* panel. Click OK.
- Click  to run Monte Carlo simulation (50 time steps, 1000 trajectories). Mean trajectory with minima, maxima and 2 sigma confidence intervals, along time and across trajectories, are displayed.
- Click  to open a text window. Click the *Settings* option  in the text window. Replace *nm*1 by *n*, remove *nm*2, *nf*1, *nf*2. Click OK.
- Click  to run Monte Carlo simulation (50 time steps, 1000 trajectories). Mean values of *n* along time, over all trajectories and over non extinct trajectories, appear in the text window with standard errors (SE).

Evolutionary dynamics

The population dynamics can be further refined by distinguishing individuals according to phenotypic traits. These traits can be adaptive, subjected to the mutation-selection process, and this is what is intended in the evolutionary component of ZEN.

Example: model file *kisdi.zen*

The original model (Kisdi & Geritz 2001) is built from a Lotka-Volterra system

(3) $$\frac{1}{n_i}\frac{dn_i}{dt} = r(s_i) - \sum_j \alpha(s_i, s_j) n_j ,$$

with n_i the density of phenotype i, $r(s_i)$ the intrinsic growth rate depending on the adaptive trait s_i, the size of individuals. The sum is an interaction term depending on the difference in size via the competition function α. This model is translated into a discrete one, handling integer population sizes:

(4) $$n(t+1) = poisson(n(t), f),$$

with the growth rate $f = \exp(ra - rb)$ (see below).

Note: The use of the Poisson distribution is not mathematically rigorous, though it produces satisfactory results. See p. 44 for a rigorous approach.

- Drag and drop model file *kisdi.zen* to the *zen.exe* icon. The file appears in the *Model file* window. Here is the declaration of the group, to which comments are added:

```
defgroup gg(1)              { declaration of group gg
rel: rel
mut: s

defrel rel
n = poissonf(n,f)           { the relation corresponds to Eq. 4

defvar n = 1000             { initial population size in the group

defmut  s = 1.0             { declaration of adaptive trait s, with initial value 1
trigger: t1                 { and characteristics of mutations
occur: 1
number: binomf(n,mu)
distrib: min(max(gaussf(s,sigma),s_min),s_max)
replace: 0
concern: n

defvar ra = r(s)                                { growth term

defvar rb = groupsumf(gg,alpha(focalf(s),s))    { interaction term

defvar f = exp(ra - rb)     { growth rate

endgroup                    { end of group declaration
```

The group corresponds to the system of equations (3), to which is added the description of mutations in the adaptive trait s (in blue). In the ZEN formalism, population size n and the adaptive trait s are multivalued variables, and so are the group variables ra, rb, f. The

expression for variable *rb* can be read: $rb = rb(s^*) = \sum_s \alpha(s^*, s)n$. It describes the ecological interactions (the competitive environment perceived by the focal phenotype with trait s^*), and corresponds to the sum in Eq. 3. The functions *r* (growth) and *alpha* (competition) appearing in the declaration of *ra* and *rb* are declared elsewhere in the file:

```
deffun r(s) = 4 - s

deffun alpha(s1,s2) = c*(1 - 1/(1 + nu*exp(-k*(s1 - s2)))) + a
```

- Click . Click in the graphic window to set graphics. Replace *nnn* by *s*. In the *Axis* panel, set *Ymax* to 4, select option *Fix Yscale* (this sets the bounds on the Y axis). Click OK. Click to run the model (100 time steps). The diversification of phenotypes appears in the graphic window: the distribution of the trait *s* across phenotypes is displayed along evolutionary time in abscissas. The color indicates the density in each phenotype (red, yellow, green, blue with decreasing density). The main window tells that at time **t** = 100, there are 4390 individuals in the group, dispatched in 128 phenotypes:

```
Model kisdi -> pop = 962.0
        growth rate from [t = 0] -> 0.999613
Group gg -> pop = 4390  nb_pheno = 128
        growth of group from [t = 0] -> 1.014903
        life expectancy of phenotypes from [t = 0] -> 0.8435
        growth rate of phenotypes from [t = 0] -> 1.049634
```

- Select option *Variable* | *All* to get a hierarchical view of ZEN objects. Select *Phenotypes* in group *gg*. All phenotypes present at time 100 appear in the left panel (128 phenotypes in all):

```
Pheno#3942 created from #379 at t = 100   nb = 1 -> 0.023%
        n = 1
        s = 0.9794
        ra = 3.021
        rb = 6.955
        f = 0.01955
Pheno#3941 created from #379 at t = 100   nb = 1 -> 0.023%
        n = 1
        s = 0.9747
        ra = 3.025
        rb = 6.976
        f = 0.01925
(…)
Pheno#1874 created from #793 at t = 48   nb = 692 -> 16%
        n = 692
        s = 1.01
        ra = 2.99
        rb = 6.823
        f = 0.02164
(…)
Pheno#535 created from #442 at t = 14   nb = 131 ->  3%
        n = 131
        s = 1.042
        ra = 2.958
        rb = 6.685
        f = 0.02407
```

```
Pheno#379 created from #1 at t = 10   nb = 80 -> 1.8%
        n = 80
        s = 0.9762
        ra = 3.024
        rb = 6.969
        f = 0.01935
```

In each phenotype, the values of the adaptive trait *s* are given, together with the values of the group variables. Phenotypes #3942 and #3941 are mutants created at time **t** = 100. They contain 1 individual. At the bottom appear the most ancient phenotypes. Phenotype #1874 is the most frequent (692 individuals, 16% of the group population), though its growth rate *f* is less favorable than that of phenotype #535 (131 individuals). The last phenotype #379 (80 individuals) mutated from the initial (now extinct) phenotype #1 at time **t** = 10. The term *nb* is the number of individuals in the phenotype, computed by summing the values of the relation-variables of the group. Here there is only one relation with variable *n*, so that *n* and *nb* agree.

- Select *Phylogeny* in the *Variable* | *All* panel. A phylogenetic tree of all actual phenotypes is displayed:

```
(…)
Pheno#1874 t = 48 -> 16%
s = 1.01
    Pheno#3254 t = 83 -> 3.1%
    s = 0.9736
        Pheno#3845 t = 99 -> 0.14%
        s = 0.9807
        Pheno#3888 t = 100 -> 0.023%
        s = 0.9936
    Pheno#3852 t = 99 -> 0.068%
    s = 1.046
        Pheno#3856 t = 100 -> 0.023%
        s = 1.052
    Pheno#3912 t = 100 -> 0.023%
    s = 1.014
    Pheno#3913 t = 100 -> 0.023%
    s = 1.006
    Pheno#3914 t = 100 -> 0.023%
    s = 1.007
    Pheno#3915 t = 100 -> 0.023%
    s = 1.004
(...)
```

The most abundant phenotype #1874 created at time 48 produced #3254 at time 83, #3852 at time 99, #3912 at time 100, etc… Its descendant #3254 produced #3845 at time 99 and #3888 at time 100.

- Click  to initialize (**init**). Select option *Run* | *Settings*. Change *Number of time steps* to 5000, and *Dt text interp* to 1000. Click OK to close the panel. Click  to run the model (5000 time steps). This takes about 8 mn. The graphics correspond to the bottom of Fig. 1.

- Click  to view the variables. Change expression of variable *c*0 from 1 to 2, type <return>. Change expression of variable *a*0 from 1 to 0, type <return>. This modifies the parameters in the competition function *alpha*. The system is initialized. Click  to run the model (5000 time steps). This takes about 4 mn. The different shape of the evolutionary branching can be appreciated. The graphics correspond to the bottom of Fig. 3.

3. OBJECTS

ZEN models are built from objects related by mathematical functions, and processed along time by the ZEN kernel. The models are described in an input text file (**.zen* file), using a declaration language. The model file is processed by the *compile* command (), and searched for syntax errors. When the syntax is correct, the model can be run ().

There are 6 types of objects handled by the ZEN kernel with corresponding keywords:

General

defmod	declaration of model
defrel	declaration of relation
defvar	declaration of variable
deffun	declaration of function

Evolutionary

defgroup	declaration of group of phenotypes
(…)	
defmut	declaration of adaptive trait within group
(…)	
endgroup	end of group declaration

Each object is referenced by a user chosen name (names begin with any letter ‘a’ to ‘z’). Other keywords specify mathematical operators or functions (see section 5).

General

- Declarations of objects must be separated by blank lines.
- Lines beginning with ‘**{**‘ are comment lines and are not processed.
- The model file must begin with the declaration of a model (**defmod**).
- The declaration of a model must precede the declarations of its associated relations.
- Relations may be declared without any link to a model.
- All variables and functions must be declared explicitly.
- Letters are converted to low case; the interpreter is not case sensitive.

Evolutionary

- Declaration of adaptive trait (**defmut**) is local to a group.
- Declaration of variable (**defvar**) and relation (**defrel**) within a group relate these objects to the group.
- General purpose variables (like constants) should not be declared within a group.

defmod — declaration of model

defmod model_name**(**k**)** declaration of model of size *k*
rel: rel1**, ... ,** relk names of *k* relations

Example: model file *regis.zen*

defmod regis**(**2**)**
rel: r1, r2 2 relations, *r*1 and *r*2

A model describes populations whose dynamics are driven by a set of discrete time relations, *r*1 and *r*2 in the example. A single model file may include several models.

defrel — declaration of relation

defrel relation_name
var_name = expression expression for the relation

Example: model file *regis.zen*

defrel r1
n1 = f*n2

defrel r2
n2 = s*n1 + v*n2

In this example, variables *n*1 and *n*2 are relation-variables constituting a population vector. From one time step to the next, relation-variables are updated in parallel (and not sequentially), as would be the case in matrix form:

$$\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}_{t+1} = \begin{bmatrix} 0 & f \\ s & v \end{bmatrix}_t \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}_t .$$

The population size of the model is the sum of the values of the relation-variables: *n*1 + *n*2.

defvar — declaration of variable

defvar variable_name = expression

There is one and only one predefined variable, whose name is **t** for 'time'. Variable **t** takes values 0, 1, 2, ... as the system is run. Other variables are declared by the user.

If *variable_name* corresponds to a variable pertaining to a relation (such a variable is called a a relation-variable), then *expression* must be a real number, which is the initial value of the variable. Variables declared within a group (group variables) are multivalued (see **defgroup**). Group variables can be used outside of their group within the scope of the functions **groupmeanf**, **groupmaxf**, **groupminf**, **groupsumf**, **groupsum1f** (section 5).

Examples:

defvar s0 = 0.2	constant
defvar n1 = 100	relation-variable with initial value 100
defvar phi = **(**1+**sqrt(**5**))**/2	constant
defvar x = **gaussf(**2,0.1**)**	random variable normal distribution with mean 2 and standard deviation 0.1
defvar w = **if(t >** 10**,** x**,** 0**)**	conditional
defvar n1 n2 = 100	shared declaration

deffun — declaration of function

deffun function_name**(**arg1**,** ... **,** argN**)** = expression

The arguments of the function have the names *arg*1, …, *argN*.

Examples:

deffun som**(**v**,**n**)** = **(**1 - v^**(**n+1**))** / **(**1 - v**)**	sum of a geometric serie
deffun fac**(**n**)** = **if(**n, n*fac**(**n-1**),** 1**)**	recursive definition of the factorial
deffun alpha**(**s1**,**s2**)** = c***(**1 - 1/**(**1 + nu***exp(-**k***(**s1 - s2**))))** + a	(file *kisdi.zen*)

The update procedure, from one time step to the next

At each time step of the ZEN simulation, relations and variables are updated in a specific order given below.

- The right hand side expression of all relations are computed.
- Relation-variables associated with group relations are updated.

  The number *nb* of individuals in each phenotype in each group is known at this stage, as the sum of the values of the relation-variables of the group.
  All phenotypes for which *nb* = 0 are destroyed.

  Relation-variables associated with model relations are updated.
  Relation-variables associated with other relations are updated.

- Time **t** is updated (**t** = **t** + 1).
  All remaining variables are updated according to their dependencies.

  Some variables can trigger mutations (see the mutation procedure forward).
  The mutations are triggered as soon as their triggering variable is updated.

At initialization (**init**), the ZEN program builds a hierarchy of all variables, according to their dependencies. It is checked whether there are circular definitions of variables: if this is the case, the message "cycling definition of variable xxx" warns the user that the computations are not reliable. From the hierarchy an order of evaluation over all variables is established. This order is used throughout the simulation to update the variables consistently (use button  to see the order of evaluation).

The values of the ZEN variables are computed in real numbers arithmetic. The relation-variables in models represent population numbers or densities: they take discrete (integer) values or continuous (real) values. The relation-variables in groups represent finite populations, and should take integer values. In any case, for each phenotype, the sum of their values represents the number *nb* of individuals in the phenotype, and is rounded to the nearest integer value.

defgroup — declaration of group

defgroup group_name**(**k**)** declaration of group of size *k*
rel: rel1**,** ... **,** relk names of *k* relations
mut: mut1, …, mutm names of *m* adaptive traits

declarations of relations *rel1*, …, *relk* (using **defrel**)
declarations of variables associated with these relations (using **defvar**)
declarations of local group variables (using **defvar**)
declarations of adaptive traits *mut1*, …, *mutm* (using **defmut**)

endgroup end of group declaration

A group is sub-object of a model. It represents a set of individuals pertaining to a single phenotype at the beginning of the simulation, and to several phenotypes as evolution proceeds. The relations (**defrel**) account for the population dynamics of the group; the adaptive traits (**defmut**) are subjected to mutations. During the ZEN simulation, mutations are triggered, new phenotypes are created, while others go extinct by the play of ecological interactions. Within a group, individuals are distinguished by their phenotypes, that is by the values of their adaptive traits. Variables declared in a group (called group variables) are multivalued: they have as many values as there are different phenotypes.

Example: model file *kisdi.zen*

defgroup gg**(**1**)** declaration of group *gg*
rel: rel with relation *rel*
mut: s and adaptive trait *s*

defrel rel declaration of group relation *rel*
n = **poissonf(**n**,**f**)**

defvar n = 1000 *n* is a group relation-variable:
number of individuals in the initial single phenotype
number of individuals in each phenotype during the simulation

defmut s = 1.0 declaration of adaptive trait *s* (see **defmut**)
(…)

defvar ra = r**(**s**)** *ra*, *rb*, *r* are group variables
each phenotype has is own values of these variables

defvar rb = **groupsumf(**gg**,**alpha**(focalf(**s**),**s**))**
functions **groupsumf** and **focalf** are described in section 5

defvar f = **exp(**ra – rb**)**

endgroup end of group declaration

defmut — declaration of adaptive trait

defmut mut_name = val	declaration of adaptive trait with initial value *val*
trigger: var_name	name of triggering variable
occur: expr_occur	expression for date of occurrence
number: expr_number	expression for number of mutations
distrib: expr_distrib	expression for distribution of mutations
replace: 0/1	replace option, 0 or 1
concern: var_concern	name of concerned variable

The declaration of adaptive trait appears only within the scope of a group (**defgroup** … **endgroup**), and has a corresponding entry in the group declaration (see **defgroup**).
The **defmut** keyword is followed by 6 keywords (**trigger**, **occur**, **number**, **distrib**, **replace**, **concern**), in this order (no blank line). These keywords describe the way mutations occur. Their meaning is explained below. Adaptive traits are considered as group variables with special features.

Example: model file *kisdi.zen*

defmut s = 1.0
{ adaptive trait *s* has initial value 1
trigger: t1
{ mutations are triggered by variable *t*1
occur: 1
{ they occur at each time step (next date = 1)
number: binomf(n,mu**)**
{ the number *M* of mutants is drawn from phenotype size *n* (number of individuals
{ sharing the same value of *s*), with *mu* the mutation rate
distrib: min(max(gaussf(s,sigma**),**0**),**4**)**
{ the value of the adaptive trait *s* of mutants is drawn according to the normal distribution
{ with standard deviation *sigma* (the distribution is constrained here between 0 and 4)
replace: 0
{ replace mode off
concern: n
{ the concerned variable is phenotype size *n*, to which *M* will be subtracted (*M* mutants).

The mutation procedure

The mutation procedure triggers one or several times within each time step. It applies to all phenotypes in one or several groups. The procedure proceeds sequentially in the following way.

1. trigger. The triggering variable allows to specify when, inside a time step, the corresponding adaptive traits are mutated. In the example above, the triggering variable is *t*1 = *magicf*(**t**) (see function **magicf** in section 5), triggering the mutation of trait *s*. After the update of relation-variables, time **t** is the first variable updated at each time step (see update procedure above): the declaration means in this example that the mutation procedure is triggered just after **t** is updated, and before any other variable is updated (see model file *ravigne.zen* for a more sophisticated example).

2. occur. The date of the next occurrence is specified by an expression (1 in the example above). If the date (computed at the previous time step) does not match the time **t**, the mutation procedure is exited at this stage, and has no effect.

3. number. For each phenotype (*i*), the number $M^{(i)}$ of mutants is computed according to the expression provided. Typically, a fraction of the number of offspring in the phenotype mutate with the mutation rate μ.

4. distrib. For each mutant in each phenotype, the mutated value *s*' of the adaptive trait *s* is computed according to the expression provided. Typically, the value *s*' is computed using a distribution around the trait value *s* of the phenotype, with the standard deviation σ. However, any expression can be used to compute *s*'.

5. replace. The expression is the constant 0 or the constant 1. If the *replace* option is 1, when a mutant appears with trait *s*', it is checked whether a phenotype with the same trait value already exists. If this is the case, the number of individuals in the existing phenotype is increased by 1, and no new phenotype is created. If the *replace* option is 0, the check is not performed. The option is intended for mutation with integer-valued distributions (for distributions with continuous values, the probability to create an already existing value of the trait is theoretically 0). The option *replace* = 1 can be time consuming.

6. concern. The concerned group variable *x* represents, in each phenotype, the individuals subjected to mutations. As it is the case in the example above, the concerned variable is usually the variable from which the number of mutants is computed in the **number** section.

Assume that there are *P* phenotypes at the start of the mutation procedure. During the mutation procedure, each phenotype (*i*) produces $M^{(i)}$ mutants (**number**). A total of $N = M^{(1)} + \cdots + M^{(P)}$ new phenotypes are created, each one containing a single mutant individual, with the mutated value *s*' of the trait *s* computed according to the distribution (**distrib**). The value of the concerned variable *x* (**concern**) on each mutant phenotype is set to 1. The value of *x* on each original phenotype (*i*) of is set to $x - M^{(i)}$. Consequently, the sum $\sum_i x^{(i)}$ of the values of the multivalued variable *x* over phenotypes does not change. The number *nb* of individuals in the created mutant phenotype is set to 1. The number *nb* of

individuals in the original phenotype (i) is changed to $nb - M^{(i)}$. Consequently, the number of individuals in the group does not change. At the end of the mutation procedure there are $P + N$ phenotypes in the group.
When a mutant phenotype is created, the values of the local variables and adaptive traits are set to those of the original phenotype, and the values of the relation-variables are set to 0. Then, the only modifications are the adaptive trait set to the mutated value s', and the concerned variable x set to 1. In particular, variables depending on x in the mutant and in the original phenotype are not updated. After the mutation procedure, the values of variables in the original and in the created phenotypes can be temporarily inconsistent (but should be updated consistently at the next time step). The model must be constructed so that the update of variables and the triggering of mutations are consistent.

Parallel and sequential mutations

The mutation procedure is triggered each time a triggering variable is updated. A single variable can trigger several mutations, possibly in different groups. When mutations m_1, m_2 … are triggered by the same triggering variable, they do not interfere in the sense that m_i-mutants are not subjected to any of the m_j mutations. On the contrary, if mutations m_1, m_2 … have been triggered and then mutations q_1, q_2 … are triggered by another triggering variable, previous m_i-mutants are subjected to them.
At the end of each execution of the mutation procedure, the created mutants belong to the group population. These mutants may influence the update of the variables which were not yet updated at the instant of their creation.

Mutations can also occur in parallel, as a single mutational event. This is the case in the model file *ravigne.zen*:

```
defgroup gg(2)
rel : rel1, rel2
mut : p, a1, a2
(…)

defmut p a1 a2 = 0.9 0.6 0.1
trigger : declench
occur : 1
number : binomf(eggs,mu)
distrib : min(max(gaussf(p,sigma),0),1); min(max(gaussf(a1,sigma),0),1); min(max(gaussf(a2,sigma),0),1)
replace : 0
concern : eggs
```

The adaptive traits p, $a1$, $a2$ have initial values 0.9, 0.6, and 0.1 respectively (separated by blanks). They share mutational characteristics, except for the distributions (declarations separated by a semi-colon ';'). One can consider that the traits p, $a1$, $a2$ are coded by a single locus, with an infinity of alleles. Their 'parallel' mutations are computed simultaneously from a same set of individuals. The mutants have the mutated traits (p', $a1$', $a2$'). If the mutations of p, $a1$, $a2$ were 'sequential', the mutants would have the traits (p', $a1$, $a2$), (p, $a1$', $a2$), (p, $a1$, $a2$'), and would not be created from exactly the same set of individuals.

4. COMMANDS

Once the ZEN model file is compiled (using ), commands allow to study the model interactively (run simulations, set graphics, …). Most commands rely on clicking the appropriate buttons (like  to run the model,  to initialize the model), and can be parameterized using the appropriate *Settings* options (like *Run* | *Settings*). Most commands can also be entered in the small *Interp* panel of the main window. The syntax is

command_name *p*1 *p*2 ...

where *command_name* is the name of the command, and *p*1, *p*2, ... are parameters of the command. For example, after typing

run 100 10

the system is run for 100 time steps with output every 10 time steps in the large panel of the main window. Trajectories are displayed in the graphic windows.

❖ Equivalently, select the *Run* | *Settings* option, set *Number of time steps* to 100, and *Dt text interp* to 10. Then click the *Run* button  .

Parameters of commands are names, integer or real values, or may be empty. Each command can be abbreviated by a single character. For example,

graph t n1 n2 is equivalent to **g** t n1 n2

and sets the trajectories to be displayed in graphic window #1, in this case the values of variables *n*1 and *n*2 along time **t**.

❖ Graphics can also be parameterized using the *Settings* option  in the graphic windows.

In this section, the list of commands is sorted in alphabetical order. The mention 'on/off' means that the command works in an *on/off* manner. For example, typing

addgraph

allows to superimpose graphs in graphic window #1 ('Addgraph ON'), and typing again

addgraph

disables this option ('Addgraph OFF').

The mention 'graph' indicates that the command is related with graphics. Optional command parameters are between < >.

Command file

ZEN simulations can be performed in absence of the user via a command file, a text file containing the commands you would have typed in the *interp* panel.

Example1: command file associated with model file *kisdi.zen*

graph t s	set graphics, adaptive trait *s* along evolutionary time **t**
yscale 0 4	fix bounds on Y axis
run 30000 1000	run 30000 time steps, with results every 1000 time steps
savegraph kisdi.bmp	save graphics in bitmap file *kisdi.bmp*

The procedure is the following:

- Drag and drop the *zen.exe* icon to the desktop, creating a shortcut to *zen.exe*.
- Right click the shortcut to access to its properties, and add to the name of the program (*zen.exe*) the names of the files:
  C:\zen\zen.exe kisdi.zen kisdi.in kisdi.out
  The syntax is:
  C:\zen\zen.exe model_file command_file output_file
  It can be necessary to give the path to the files, like C:\zen\kisdi.zen. The output file is optional: if provided, results displayed in the main window will be stored in it.
- Left click the shortcut to run the ZEN simulation.

Example2: command file associated with model file *pass_2s.zen*

graph t n	set graphics
text t n	set output in the main window
change nm1 2	set population size and structure
change nm2 2	
change nf1 2	
change nf2 2	
montecarlo 100 1000	run Monte Carlo simulation
change nm1 4	set population size and structure
change nm2 4	
change nf1 4	
change nf2 4	
montecarlo 100 1000	run Monte Carlo simulation

Several simulations with different model files can be grouped using a batch file. Simply create a text file containing the relevant commands for executing the ZEN program.

Example3: batch file associated with the 2 previous examples

zen.exe	kisdi.zen	kisdi.in	kisdi.out
zen.exe	pass_2s.zen	pass_2s.in	pass2s.out

The file must have the *.bat* extension, for example *MySimul.bat*. In this example, the batch file is located in the same directory as *zen.exe*, *kisdi.zen*, *kisdi.in*, *pass_2s.zen* and *pass_2s.in*. Double-click the batch file *MySimul.bat* to execute the simulations.

Addgraph **on/off** **graph**

abbreviation: +
other name: **add**
syntax: addgraph
function: Superimpose graphics.
default: *off*

- Use option *Superimpose* in the graphic windows *Settings*  .
- Graphic window #*i* can be selected using the **window** command.

note: When addgraph is *on*, do not resize the graphic window.

example: model file *met_esd.zen*

? **change** nmax 200	change value of population ceiling
? **graph** t n	plot population size *n* as a function of time **t**
? **run** 100	run 100 time steps
? **yscale**	fix bounds in Y axis
Yscale ON [1; 3115]	
? **addgraph**	superimpose graphics
Addgraph ON	
? **change** nmax 50	back to initial value of population ceiling
Init	
? **run** 100	run 100 time steps appreciate how the trajectory separates from the previous one once the population ceiling is reached

Changevar

abbreviation: **c**
other name: **change**
syntax: change *var expr*
parameters: *var* variable name
expr mathematical expression
function: Replace actual expression of variable *var* by new expression *expr*.
System is initialized (**init**).

- Use button to display the model variables. The expression of any variable can be changed by clicking in the corresponding field.

note: Relation-variables must be set to a real number, which is their initial value.
Adaptive traits (declared using **defmut**) are considered as variables,
their initial value can be changed; it must be a real number.

example1: model file *pass_2s.zen*

? **graph** t n	plot population size *n* as a function of time **t**
? **montecarlo** 50 1000	run Monte Carlo simulation probability of extinction *pe* = 0.616
? **change** s0 beta1f(0.2,0.15)	make juvenile survival rate *s*0 stochastic
? **montecarlo** 50 1000	run Monte Carlo simulation probability of extinction *pe* = 0.911

example2: model file *kisdi.zen*

? **graph** t s	plot adaptive trait *s* as a function of (evolutionary) time **t**
? **yscale** 0 4	fix bounds in Y
? **run** 5000 100	run 5000 time steps, with results every 100 time steps (wait about 6 minutes) display evolutionary branching at time **t** = 3200
? **view** mu sigma	display parameters of mutations
mu = 0.02	*mu* = mutation rate μ
sigma = 0.02	*sigma* = standard deviation σ
? **change** sigma 0.04	multiply standard deviation by 2
Init	
? **run** 2000	run 2000 time steps (wait about 2 minutes) display evolutionary branching at time **t** = 800 = 3200/4 shows that evolutionary time **t** is proportional to σ^2
? **change** mu 0.04	multiply mutation rate by 2
Init	
? **run** 1000	run 1000 time steps (wait about 4 minutes) display evolutionary branching at time **t** = 400 = 800/2 shows that evolutionary time **t** is proportional to μ

Distribution on/off graph

abbreviation: **u**
other name: **distrib**
syntax: distrib <*delta*>
parameter: *delta* real number > 0 (default *delta* = 1)
function: Display distributions of variables specified by the **graph** command.
For variable *x*: number of values of *x* such that
*delta***j* <= *x* < *delta**(*j*+1) for *j* = 1, ..., 10000.
For the **run** command, distribution along time.
For the **montecarlo** command, distribution across trajectories at time horizon.
default: *off*

- Use option *distrib* in graphic windows *Settings*  .
- Select option *include 0* to include the value 0 in the distribution.
- Graphic window #*i* can be selected using the **window** command.

note: The distrib command does not affect the representation of group variables (group variables are represented using their distribution across phenotypes).

example1: model file *pass_2s.zen*

? **graph** t n — note: **t** is dummy for **distrib**
? **distrib** 100 — set distribution mode with *delta* = 100
Distribution mode ON
? **run** 100 — display distribution of *n*(*t*) along time
? **montecarlo** 100 1000
display distribution of *n*(*t*) at time *t* = 100, over 1000 trajectories

example2: model file *regis.zen*

? **graph** t n — note: **t** is dummy for **distrib**
? **distrib** 0.1 — set distribution mode with *delta* = 0.1
Distribution mode ON
? **change** r 50 — lead to point-equilibrium
? **run** 1000 1000 — display distribution of *n* over 1000 time steps
? **change** r 60 — lead to quasi-cycle
? **run** 10000 — display distribution of *n* over 10000 time steps
? **change** r 110 — lead to chaos
? **run** 10000 — display distribution of *n* over 10000 time steps

Erase · graph

abbreviation: **e**
other name: **clear**
syntax: erase
function: Clear graphics (window #1 or graphic window selected by command **window**).

- Use alternatively button · in graphic windows.

File **on/off**

abbreviation: **f**
syntax: file file_name *x*1 ... *xN*
parameters: *file_name* name of file
*x*1 ... *xN* names of variables
function: Create text file *file_name* and store values of variables *x*1, ... , *xN* in the file as the model is run (**run** or **montecarlo** command).
When the variable names are not given, the file *file_name* is closed.
Variables *x*1, ... , *xN* must either all belong to the same group or not belong to any group.
Storage differ according to the **run** or **montecarlo** command.

1) For the **run** command:
For variables that are not group variables the format of each line in the file is:

t *v*1 *v*2 … *vN*

where *v*1, …, *vN* are the values of variables *x*1, …, *xN* at time *t*.
There is a new line in the file at each time step that is a multiple of the second parameter Δ of the **run** command (see **run** command).
For example, with the command

? **run** 1000 10

values are stored every Δ = 10 time steps.
For group variables the format is:

t *n*1 *v*11 *v*21 … *vN*1
t *n*2 *v*12 *v*22 … *vN*2
…
t *nP* *v*1*P* *v*2*P* … *vNP*

where *n*1, …, *nP* is the number of individuals in *P* phenotypes in the group at time *t*, and *v*1*i*, …, *vNi* the values of the group variables in phenotype *i*.
At each time step that is a multiple of Δ there are as many lines in the file as phenotypes in the group.

2) For the **montecarlo** command, **montecarlo** *T* *M*, with *T* the number of time steps and *M* the number of trajectories:
For variables that are not group variables the format of each line in the file is:

j *v*1 *v*2 … *vN*

where *v*1, …, *vN* are the values of variables *x*1, …, *xN* at time *T* in the *j*th trajectory (*j* = 1, …, *M*).
There is a new line in the file at each trajectory.
For group variables the format is:

j *n*1 *v*11 *v*21 … *vN*1
j *n*2 *v*12 *v*22 … *vN*2
…
j *nP* *v*1*P* *v*2*P* … *vNP*

where *n*1, …, *nP* is the number of individuals in *P* phenotypes in the group at time *T* in the *j*th trajectory, and *v*1*i*, …, *vNi* the values of the group variables in phenotype *i*.
For each trajectory there are as many lines as phenotypes in the group.

- Up to 5 files can be created in a session.
- Up to 10 variables can be stored simultaneously in a file.
- The path of the file can be specified (the default path is where the ZEN program is located, usually c:/zen). Example: ? **file** c:/myfolder/myfile.txt x1 x2.
- The number of digits after the decimal point can be specified using the separator ':' (the default precision is 4). For example, after the command

  ? **file** myfile.txt x1:10 x2

  variable *x*1 will be stored with 10 digits after the decimal point, and variable *x*2 with 4 digits after the decimal point.

example1: model file *ferriere1.zen*

? **file** toto.txt ux	open file *toto.txt* to store group variable *ux*
File toto.txt opened	
? **file** titi.txt tt ux_moy	open file *titi.txt* to store variables *tt* and *ux_moy*
? **run** 100 1	run model (100 time steps, $\Delta = 1$)
(…)	values are stored in the files every time step ($\Delta = 1$)
? **file** toto.txt	close file *toto.txt*
File toto.txt closed	
? **file** titi.txt	close file *titi.txt*
File toto.txt closed	

File *toto.txt* looks:

0	4000	50.0000
1	1	49.7584
1	1	51.7309
1	1	49.7585
1	4137	50.0000
2	1	49.7584
2	1	51.7309
2	1	49.7585
2	3896	50.0000
(…)		

File *titi.txt* looks:

0	0.0000	50.0000
1	0.0689	50.0003
2	0.1171	50.0003
3	0.1660	49.9997
(…)		

example2: model file *nirma.zen*

? **file** nirma.txt di_simp di_moy — open file *nirma.txt* to store variables *di_simp* and *di_moy*

File nirma.txt opened

? **montecarlo** 1000 10 — run model (1000 time steps, 10 trajectories)

(…) — values are stored in the file for each trajectory

? **file** nirma.txt — close file *nirma.txt*

File nirma.txt closed

File *nirma.txt* looks:

```
1    1    49.7584
2    1    51.7309
3    1    49.7585
(…)
```

Graph  graph

abbreviation: **g**
syntax: graph *x y*1 ... *yN*
parameters: *x y*1 ... *yN* names of variables
function: Display variables *y*1, ... , *yN* as a function of variable *x* in graphic window.
If distribution mode is *on*, distributions of *y*1, ..., *yN* are displayed.
Group variables are represented by their distribution across phenotypes.
For group variables, fix scales using commands **xscale**, **yscale**

see also: **addgraph**, **distribution**, **erase**, **line**, **savegraph**, **window**, **xscale**, **yscale**

- Up to 6 graphic windows, numbered #1 to #6, can be created using button  .
- Each graphic window can be parameterized using the *Settings* option  .
- The **graph** command operates on window #1 unless window #*i* has been selected using the **window** command. It is useful in command files.

note: When the number of time steps is larger that 10000, a sampling of the trajectories is performed (see *Dt* in the graphic window status bar).
For example, for 100000 time steps a point is taken every *Dt* = 10 time steps.
For group variables the resolution is 600x600.

example1: model file *regis.zen*

? **graph** n1 n2	set graphics for phase portrait
? **change** r 110	change bifurcation parameter
? **run** 10000 1000	display strange attractor
? **change** r 60	change bifurcation parameter
? **run** 10000	display limit cycle

example2: model file *kisdi.zen*

? **graph** s t	set graphics for adaptive trait *s* with time in ordinates
? **xscale** 0 4	fix bounds in X (0 <= *s* <= 4)
? **run** 10000 1000	produce cover figure (in about 10 mn)

Help

abbreviation: **h** or **?**
syntax: help <*xxx*>
parameter: *xxx* name of command or mathematical function
function: Give succinct on line information about commands and mathematical functions.

help
list of commands and mathematical functions.

help *xxx*
short description of command *xxx* or mathematical function *xxx*.

example: ? **help** groupsum1f

groupsum1f(G,expr)
sum of values of expression expr
over all phenotypes in group G
groupsum1f(G,x) = sum(i=1,..,groupcard;v_i)

Init

abbreviation: **i**
syntax: init <*j*>
parameter: *j* integer >= 0, random generator seed
function: **init**
initialize:
t = 0,
variables are reset to their initial values,
groups are reset to their initial monomorphic state,
random generator is reset to its initial value (called seed).
init *j*
init + seed initialized to *j*,
corresponding to the *j*-th trajectory of the Monte Carlo procedure.
init 1
init + back to the default seed (*j* = 1).
remark: **init** is performed automatically after the following commands:
changevar, **montecarlo**, **newvar**.

- Command **init** can be performed using button in the main window.
- The random generator seed can be set using the *Run* | *Settings* option.

example: model file *pass_2s.zen*

```
? graph  t  n
? init
Init
? montecarlo  50  100      give probability of extinction estimate
( ... )                    pe = 0.68 (at time 50)
? init  500
random generator seed -> 500
Init
? montecarlo  50  100      another simulation of the process
( ... )                    pe = 0.60
? init  1
random generator seed -> 1
Init
? montecarlo  50  1000     back to first simulation, better estimate
( ... )                    pe = 0.616
? init  500
random generator seed -> 500
Init
? montecarlo  50  1000     back to 2nd simulation, better estimate
( ... )                    pe = 0.615
```

abbreviation: **l**
syntax: line *<col>*
parameter: *col* integer in [1, …, 16], line color
function: If *on* lines are drawn between consecutive points in graphic window #1 (or window #*i* specified by the **window** command), using color *col*. Useful in command files.
default: *on*

- Use alternatively option *line off* in graphic windows *Settings*  .
- Use color specification in graphic windows *Settings*  , by clicking on the colored button next to the graphic variables panels Y1 Y2 Y3 Y4.

example: model file *pass_2s.zen*

```
? graph  t  n
? line  1
? yscale  0  400
? run  50              display red trajectory
? addgraph
? line  2
? init  2
? run  50              superimpose green trajectory
```

Montecarlo

abbreviation: **m**
other name: **monte**
syntax: montecarlo *T* *M* <*Ext*> <*Esc*>
parameters: *T* integer > 0, number of time steps
M integer > 0, number of trajectories
Ext real number > 0, extinction threshold (default *Ext* = 1)
Esc real number > 0, escape threshold (default *Esc* = 10^7)
function: Monte Carlo simulation:
M trajectories are run over a time horizon of *T* time steps.
System is initialized at the end (**init**).

- Monte Carlo simulation is parameterized using the *MonteCarlo* | *Settings* option.
- Monte Carlo graphics are parameterized using the *Settings* option in the graphic windows.
- Monte Carlo outputs are parameterized using the text windows *Settings* option.
- Press Ctrl-Alt simultaneously to break simulation (with the main window selected).

- Mean trajectories over *M* runs are displayed in the graphic windows (with min, max and $\pm 2\sigma$ intervals if requested).
- Mean values along time with standard errors are displayed in the text windows (including or excluding extinct trajectories).
- For group variables, mean distribution over phenotypes and across trajectories are displayed.
- If distribution mode is *on*, distributions of trajectories at time *T* are displayed.
- *j*-th trajectory whose population size $n_j(t)$ is < *Ext* is declared *extinct* (at time *t*), but computed to the end (default $n_j(t)<1$).
- *j*-th trajectory whose population size $n_j(t)$ is > *Esc* is declared *escaped* (at time *t*), but computed to the end.
- Population size is computed as the sum of the values of the relation-variables of the group (or model when there are no groups).
- For each group (or model): probability of extinction along time, mean time to extinction (computed over extinct trajectories), probability of escape, mean escape time (computed over escape trajectories), growth rates, non extinct population size values, mean population structure.
- Stochastic growth rate = $\exp(a)$ where *a* is the average of logarithmic growth rates of all *M* trajectories, computed as $a = \frac{1}{M}\sum_{j=1}^{M}\left[\frac{\ln(n_j(T)) - \ln(n_j(0))}{T}\right]$; relevant estimator for pure environmental stochasticity.
- Growth rate of the mean pop = growth rate of the average trajectory, computed as $\exp\left[\frac{\ln(\bar{n}(T)) - \ln(\bar{n}(0))}{T}\right]$ with $\bar{n}(t) = \frac{1}{M}\sum_{j=1}^{M} n_j(t)$, the average trajectory.

- Mean growth rate2 = average of growth rates of *non extinct* trajectories, computed as $\frac{1}{M^*}\sum_{j=1}^{M^*}\frac{n_j^*(1)+\cdots+n_j^*(T)}{n_j^*(0)+\cdots+n_j^*(T-1)}$, where $n_j^*(t)$ is a non extinct trajectory.
- Growth rate2 of the mean pop = growth rate of average *non extinct* trajectory, computed as $\frac{\bar{n}^*(1)+\cdots+\bar{n}^*(T)}{\bar{n}^*(0)+\cdots+\bar{n}^*(T-1)}$, with $\bar{n}^*(t)=\frac{1}{M^*}\sum_{j=1}^{M^*}n_j^*(t)$ the average non extinct trajectory; relevant estimator for pure demographic stochasticity.

example1: model file *pass_2s.zen*

? **graph** t n
? **text** t n
? **montecarlo** 50 1000 run Monte Carlo simulation
(...) 50 time steps, 1000 trajectories
growth rate2 of the mean pop = 1. 0254
growth rate estimator for demographic stochasticity

t	pe(t)	pop(t)	SE	pop*(t)	SE
10	0.0020	47.1	0.8	47.2	0.8
20	0.1410	42.1	1.3	49.0	1.4
30	0.3690	42.5	2.0	67.4	2.7
40	0.5040	48.8	3.0	98.4	5.2
50	0.6160	63.9	4.8	166.3	10.7

{ probability of extinction, mean pop size, mean pop size over non extinct trajectories

? **view** cc coefficient of reduction in the number of matings
cc = 0.95
? **change** cc 1 no reduction in number of matings
? **montecarlo** 50 1000 run Monte Carlo simulation
(...) 50 time steps, 1000 trajectories
growth rate2 of the mean pop = 1.0812
probability of extinction at time 50 = 0.084 (much lower than 0.616)

example2: model file *pass_2s.zen*

{ initial population size is 48 individuals
? **graph** t n
? **text** t n
? **montecarlo** 50 1000 30 run Monte Carlo simulation
(...) 50 time steps, 1000 trajectories,
extinction threshold = 30
probability of extinction = 0.760
{ probability to get less than 30 individual by time 50
mean population size at time 50 [SE] = 64 [5]
mean population size at time 50 over non extinct trajectories [SE] = 247 [15]

? **change** nm1 24 change initial population size
? **change** nm2 24 to 96 individuals
? **change** nf1 24
? **change** nf2 24
? **montecarlo** 50 1000 30
probability of extinction = 0.121
{ probability to get less than 30 individual by time 50
mean population size at time 50 [SE] = 564 [16]
mean population size at time 50 over non extinct trajectories [SE] = 640 [17]

example3: model file *pass_2s.ulm*

{ initial population size is 48 individuals
? **graph** t n
? **montecarlo** 50 1000 1 100 run Monte Carlo simulation
(...) 50 time steps, 1000 trajectories
extinction threshold = 1
escape threshold = 100

probability of escape = 0.212
{ probability to get more than 100 individuals by time 50
mean time to escape = 21
probability of extinction = 0.616
{ probability to get less than 1 individual by time 50
mean time to extinction = 29

example4: model file *met_esd.zen*

- Click option  in graphic window. Change *n*1 to *n*, remove *n*2, *n*3, *n*4. Select options *MinMax* and *2 sigma*. Click OK.
- Click button  to run Monte Carlo simulation (**montecarlo** 100 10 by default). Mean trajectories appear with confidence intervals.
- Select option *Montecarlo* | *Settings*. Change *Number of trajectories* to 1000. Click OK. Click button  to run Monte Carlo simulation (now **montecarlo** 100 1000).

example5: model file *kisdi.zen*

? **graph** s t set graphics
? **xscale** 0 4
? **montecarlo** 1000 10 display average evolutionary trajectory in graphic window #1
(takes about 10 min)

Newvar

abbreviation: **n**
other name: **new**
syntax: newvar *var* *expr*
parameters: *var* name of a variable
expr mathematical expression
function: Creation of a new variable with name *var*, and expression *expr*.
System is initialized (**init**).

example: model file *pass_2s.zen*

```
? newvar  pe  n < 1          create variable pe = if n < 1 then 1 else 0
Init
? graph  t  pe               set graphics
? yscale  0  1               fix bounds in Y
Yscale ON [0;1]
? montecarlo 100  1000
                             display pe = probability of extinction along time
                             (as average trajectory)
```

example: model file *kisdi.zen*

```
? newvar  di1  groupsumf(gg,n/(ntot*ntot))
                             create variable di1 = Simpson's diversity index of group gg
                             (n is phenotype size, ntot is group size)
? newvar  di2  groupsumf(gg,ln0(n/ntot)/ntot))
                             create variable di2 = Shannon's diversity index of group gg
? graph  t  di1  di2         set graphics
? run  100  10               display diversity indexes along time
```

Run 

abbreviation: **r**
syntax: run T <Δ>
parameters: T integer > 0, number of time steps (default T = 100)
Δ integer > 0, number of steps for text display (default Δ = 10)
function: Run the models for T time steps with output every Δ time steps.

see also: command **init**

- ❖ The **run** command is parameterized using the *Run* | *Settings* option.
- ❖ Graphics are parameterized using the graphic windows *Settings* option  .
- ❖ Results are parameterized using the text windows *Settings* option  .
- ❖ Press Ctrl-Alt simultaneously to break simulation (with the main window selected).

- Trajectories are displayed in graphic windows, numerical values are displayed in text windows. For group variables, distributions across phenotypes are displayed along time.
- Growth rates of the models from time **t** = T_0 : $\lambda = \exp\left[\frac{\ln(n(T+T_0)) - \ln(n(T_0))}{T}\right]$, with n the number (or density) of individuals along time (sum of relation-variables values).
- Growth rates of groups from time **t** = T_0 : $\lambda = \exp\left[\frac{\ln(g(T+T_0)) - \ln(g(T_0))}{T}\right]$, with g the number of individuals in the group along time (sum of group relation-variables values).
- Growth rates of phenotypes in groups: $L = \exp\left[\frac{\ln(C-D)}{T}\right]$, with C the number of phenotypes created (born by mutation) and D the number of phenotypes destroyed (extinct by selection) in the T time steps.
- Life expectancies of phenotypes in groups: mean value of $t_D - t_C$ over phenotypes created at time t_C which were destroyed at time t_D.

example1: model file *regis.zen*

```
? change  r  110
? graph  n1  n2          set graphics
? xscale  0  140         fix bounds in X
? yscale  0  2           fix bounds in Y
? addgraph               Addgraph ON
? run  10000  1000       run 10000 time steps (Δ = 1000)
? run                    run 10000 more time steps
                         display strange attractor
```

example2: model file *kisdi.zen*

? **run** 100 run 100 time steps from **t** = 0

```
Model kisdi -> pop = 962.0
    growth rate from [t = 0] -> 0.999613
Group gg -> pop = 4390  nb_pheno = 128
    growth of group from [t = 0] -> 1.014903
    life expectancy of phenotypes from [t = 0] -> 0.8435
    growth rate of phenotypes from [t = 0] -> 1.049634
```

? **run** 100 run 100 more time steps

```
Model kisdi -> pop = 533.0
    growth rate from [t = 0] -> 0.996859
    growth rate from [t = 100] -> 0.994112
Group gg -> pop = 4258  nb_pheno = 159
    growth of group from [t = 0] -> 1.007270
    growth of group from [t = 100] -> 0.999695
    life expectancy of phenotypes from [t = 0] -> 1.0745
    life expectancy of phenotypes from [t = 100] -> 1.3087
    growth rate of phenotypes from [t = 100] -> 1.034936
```

Savegraph graph

abbreviation: **!**
other name: **save**
syntax: save xxx.bmp
parameter: *xxx.bmp* name of graphic file
function: Store graphic window in bitmap file *xxx.bmp*.
The index of the graphic window to be stored can be specified using the **window** command.
The *bmp* file can be later modified, converted to *jpg* or printed.

- The **savegraph** command is useful in command files.
- Use alternatively the *File* | *Save* option in each graphic window.

note: In case of superimposed graphics (**addgraph** command), graphics are saved using a fixed window size.

Text **on/off**

abbreviation: **t**
syntax: text *var*1 ... *varN*
parameters: *var*1 ... *varN* names of variables (*not* group-variables)
function: If *on*, display values of variables *var*1, ..., *varN* in the main window, as the **run** or **montecarlo** command is executed.
default: *on*

note: Values of group variables are not displayed, since they are multivalued (see command **view**, option *Variables* | *All*) .

- Button allows to open up to 6 text windows, numbered #1 to #6, which can be parameterized using the *Settings* option.
- For the Monte Carlo simulation, no more than 1000 rows can be displayed. The *Sampling interval* should be adjusted in accordance with the number of time steps.
- The text windows can be saved using the *File* | *Save* option.
- Command **text** is totally independent of the text windows, though the purpose is similar.

example1: model file *regis.zen*

- Click to open text window #1, then to run the model 100 time steps. Values of time **t** and variables *n*1, *n*2 are displayed in the text window (every 10 time steps).
- Click the *Settings* option, change *n*1 to *n*, remove *n*2. Change *Sampling interval* to 20. Click OK.
- Click to run the model 100 more time steps. Values of time **t** and variable *n* are displayed every 20 time steps.

example2: model file *pass_2s.zen*

- Select option *Montecarlo* | *Settings*. Change *Number of trajectories* to 1000 (*Number of time steps* is 100 by default), click OK.
- Click to open text window #1. Click the *Settings* option, change *nm*1 to *n*, remove *nm*2, *nf*1, *nf*2. Click OK.
- Click to run the Monte Carlo simulation. Values of time **t** , mean population size (*n*) with standard error (SE), mean population size over non extinct trajectories (*n**) with standard error (SE) are displayed in the text window.

View 

abbreviation: **v**
syntax: view *o*1 ... *oN*
parameters: *o*1 ... *oN* names of ZEN objects
(variable, relation, function, group, adaptive trait)
function: Display initial and actual values of objects *o*1, ..., *oN*.
syntax: view
function: Display all ZEN objects.

- Option  provides a window with initial values, actual values and expressions of all variables. Expressions can be modified by selecting the corresponding field, modifying the expression, and typing <return>. Click in the status bar to list variables according to *Evaluation order* or *Alphabetic order*.
- Option *Variables* | *All* gives all ZEN objects as a hierarchical tree, with values of group variables and adaptive traits across phenotypes. Click in the status bar to sort objects according to date of creation, number of individuals in phenotype, and values.
- Option *Variables* | *Calculator* is a desk calculator allowing the computation of mathematical expressions possibly involving ZEN variables.

example: model file *kisdi.zen*

- Click  . Click  to run the model 100 time steps. Select option *Variables* | *All*. Select *Mutations*, then select *s*. Values of the adaptive trait *s* over all phenotypes present at time **t** = 100 are displayed:

```
Mutation s = 1
    trigger : t1
    occur   : 1
    number  : binomf(n,mu)
    distrib : min(max(gaussf(s,sigma),s_min),s_max)
    replace : 0
    concern : n

Pheno#3942 created from #379 at t = 100   nb = 1 -> 0.023%
    s = 0.9794
Pheno#3941 created from #379 at t = 100   nb = 1 -> 0.023%
    s = 0.9747
(…)
```

- Click  to run the model 100 more time steps. Click into the *Variable* | *All* panel to update. Values of the adaptive trait *s* over all phenotypes present at time **t** = 200 are displayed:

```
Pheno#7829 created from #2002 at t = 200   nb = 1 -> 0.023%
    s = 1.047
Pheno#7828 created from #2002 at t = 200   nb = 1 -> 0.023%
    s = 1.017
(…)
```

Window on/off graph

abbreviation: **w**
syntax: window *i*
parameter: *i* integer in [1, …, 6], refer to graphic window #i
function: Select or create graphic window #i, to which will apply all subsequent graphic commands:

addgraph
distribution
erase
line
graph
savegraph
xscale
yscale

default: *i* = 1

- Useful to store several graphic windows using command files.

example: model file *ferriere1.zen*

```
? graph  tt  ux                    Parameterize graphics (for window #1 by default)
? xscale  0  4000
Xscale ON  [0, 4000] (graphic window #1)
? yscale  0  200
Yscale ON  [0, 200] (graphic window #1)
? window  2                        Create and select graphic window #2
Graphic window #2 selected
? graph  tt  uy                    Parameterize graphics for window #2
? xscale  0  4000
Xscale ON  [0, 4000] (graphic window #2)
? yscale  0  200
Yscale ON  [0, 200] (graphic window #2)
? run  200000  1000                Run model
? savegraph  uy.bmp                Save graphics in file uy.bmp for window #2
? window  1                        Select graphic window #1
Graphic window #1 selected
? savegraph  ux.bmp                Save graphics in file ux.bmp for window #1
```

Xscale — on/off — graph

abbreviation: **x**
syntax: xscale <*xmin*> <*xmax*>
parameters: *xmin xmax* real numbers, bounds of graphics on the X axis
function: Fix bounds *xmin* and *xmax* for abscissas (default: actual values).
default: *off* (automatic scaling on the X axis)
see also: **yscale**, **addgraph**

- Use alternatively option *Fix Xscale* in graphic windows *Settings*  .
- Graphic window #*i* can be selected using the **window** command.

Yscale — on/off — graph

abbreviation: **y**
syntax: yscale <*ymin*> <*ymax*>
parameters: *ymin ymax* real numbers, bounds of graphics on the Y axis
function: Fix bounds *ymin* and *ymax* for ordinates (default: actual values).
default: *off* (automatic scaling on the Y axis)
see also: **xscale**, **addgraph**

- Use alternatively option *Fix Yscale* in graphic windows *Settings*  .
- Graphic window #*i* can be selected using the **window** command.

5. MATHEMATICAL FUNCTIONS

Binary operators

\+ * / - ^ (power)

\ real modulo: $a \setminus b = a - b*\text{trunc}(a/b)$
examples: 7.4 \ 2 = 1.4, 7 \ 2 = 1

@ convolution operator: $F @ n$ = sum of n samples of distribution F
examples: *ber*(*p*) @ *n* = *binomf*(*n*,*p*), *poisson*(*f*) @ *n* = *poissonf*(*n*,*f*)

< $a < b$ is 1 if a is strictly less than b, 0 otherwise
> $a > b$ is 1 if a is strictly greater than b, 0 otherwise

Unary operators

-	minus
sqrt	square root
abs	absolute value
trunc	integer part examples: *trunc*(3.5) = 3, *trunc*(3.8) = 3, *trunc*(-3.5) = -4, *trunc*(-3.8) = -4
round	nearest integer examples: *round*(3.2) = 3, *round*(3.6) = 4, *round*(-3.2) = -3, *round*(-3.6) = -4
ln	neperian logarithm
ln0	neperian logarithm defined in 0 by *ln*0(0) = 0
log	decimal logarithm
exp	exponential
fact	factorial

cos	cosinus	**acos**	inverse cosinus
sin	sinus	**asin**	inverse sinus
tan	tangent	**atan**	inverse tangent

Other operators

min	$min(a_1, ..., a_n)$ = minimum of the a_i
max	$max(a_1, ..., a_n)$ = maximum of the a_i
if	conditional: *if*(*A*,*B*,*C*) = if $A \neq 0$ then B else C examples: *if*(2<3,1,2) = 1, *if*(*trunc*(2.5)-2,1,2) = 2
gratef	*gratef*(*x*) = growth rate of variable x at time $T = \exp\left[\frac{\ln(x(T)) - \ln(x(0))}{T}\right]$
bicof	*bicof*(*n*,*p*) = binomial coefficient $C(n,p)$

Random functions: continuous distributions

rand *rand*(*a*) uniform distribution over [0, *a*]
domain: $a > 0$
range: [0, *a*]
mean: $a/2$
variance: $a^2/12$
density: $(1/a)$*characteristic function of [0, *a*]

gaussf *gaussf*(*m*,*s*) gaussian distribution with mean *m* and standard deviation *s*
domain: $s > 0$
range: **R**
density: $\frac{1}{s\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2 \right]$

gauss *gauss*(*s*) gaussian distribution with mean 0 and standard deviation *s*
gauss(*s*) = *gaussf*(0,*s*)

gamm *gamm*(*a*) gamma distribution with parameter *a*
domain: $a > 0$
range: $\mathbf{R}_+^*$
mean: a
variance: a
density: $(1/\Gamma(a))\, x^{a-1}e^{-x}$

betaf *betaf*(*a*,*b*) beta distribution with parameters *a* and *b*
domain: $a > 0$, $b > 0$
range: [0, 1]
mean: $a/(a+b)$
variance: $ab/((a+b+1)(a+b)^2)$
density: $(\Gamma(a+b)/\Gamma(a)\Gamma(b))\, x^{a-1}(1 - x)^{b-1}e^{-x}$

beta1f *beta1f*(*m*,*s*) variant of beta distribution with mean *m* and standard deviation *s*
domain: $m > 0$, $0 < s^2 < m(1 - m)$
range: [0, 1]
mean: m
variance: s^2
remark: the distribution is bell-shaped for small *s* and U-shaped for large *s*

expo *expo*(*a*) exponential distribution with parameter *a*
domain: $a > 0$
range: $\mathbf{R}_+^*$
mean: $1/a$
variance: $1/a^2$
density: $a\exp(-ax)$

lognormf *lognormf*(*m*,*s*) lognormal distribution with mean *m* and standard deviation *s*
domain: $m > 0$, $s > 0$
range: $\mathbf{R}_+^*$

Random functions: integer distributions

ber *ber*(*p*) Bernoulli samples: $P(X = 0) = 1 - p$, $P(X = 1) = p$
domain: $0 \le p \le 1$
range: {0, 1}
mean: p
variance: $p(1 - p)$
generating function: $f(s) = (1-p) + ps$

binomf *binomf*(*n*,*p*) binomial distribution: $P(X = k) = C(n,k)p^k(1 - p)^{n-k}$
domain: $n \ge 0$, $0 \le p \le 1$
range: {0, 1, ..., *n*}
mean: np
variance: $np(1 - p)$
generating function: $f(s) = ((1-p) + ps)^n$

nbinomf *nbinomf*(*r*,*p*) negative binomial distribution: $P(X = k) = C(k+r-1,r-1)p^r(1 - p)^k$
domain: r real > 0, $0 \le p \le 1$
range: **N**
mean: $r(1 - p)/p$
variance: $r(1 - p)/p^2$

nbinom1f *nbinom1f*(*m*,*s*) negative binomial distribution, mean *m*, standard deviation *s*
domain: $0 < m < s^2$
range: **N**
mean: m
variance: s^2

poisson *poisson*(*m*) Poisson distribution with mean *m*: $P(X = k) = e^{-m} m^k/k!$
domain: $m \ge 0$
range: **N**
mean: m
variance: m
generating function: $f(s) = \exp(m(s-1))$
poissonf *poissonf*(*n*,*m*) give the sum of *n* samples of *poisson*(*m*)

geom *geom*(*p*) geometric distribution with parameter *p*: $P(X = k) = p(1 - p)^k$
domain: $0 \le p \le 1$
range: **N**
mean: $(1 - p)/p$
variance: $(1 - p)/p^2$
generating function: $f(s) = p + p(1 - p)s/(1 - (1 - p)s)$

tabf *tabf*(p_0, ..., p_n) tabulated distribution: $P(X = k)$ = if $k \le n$ then p_k else 0
domain: $0 \le p_k \le 1$, $p_0 + ... + p_n = 1$
range: {0, 1, ..., *n*}
mean: $m = f'(1) = p_1 + 2p_2 + ... + np_n$
variance: $f''(1) + m - m^2$
generating function: $f(s) = p_0 + p_1 s + ... + p_n s^n$

From continuous time to discrete time

Models of adaptive dynamics are often given as birth-death processes in continuous time, while ZEN works in discrete time. A special function *bdf* is provided to handle the discrete time under which a continuous birth-death process is observed.

bdf *bdf*(*n*,*b*,*d*,*Δ*)
n = number of individuals, *b* = birth rate, *d* = death rate, *Δ* = observation interval.

Example: model file *kisdi.zen* (see p. 11)

```
defvar delta = 0.1              { observation interval

defgroup gg(1)                  { declaration of group gg
rel: rel
mut: s

defrel rel
n = bdf(n,ra,rb,delta)          { the relation replaces n = poissonf(n,f)

( … )

defvar ra = r(s)                    { growth term, birth rate

defvar rb = groupsumf(gg,alpha(focalf(s),s))    { interaction term, death rate
```

The *bdf* distribution is used in the same way as the *poissonf* distribution: the number n' of individuals in the next time step is computed as a sum of n trials. The formulas given below have been developed by Amaury Lambert.

Let Z_t denote the stochastic population size at time t, sampled from the continuous time birth-death process, with Δ the observation interval. Let $r = b - d$ denote the rate of increase, $p_0 = P(Z_t = 0)$ the probability of reaching 0 individuals at time t, $p_k = P(Z_t = k | Z_t \neq 0)$, the probability of having k individuals at time t, conditional upon non extinction. We compute p_0 and $p_k = (1-\rho)^{k-1}\rho$ according to

1) $r < 0$: $p_0 = \frac{d(1-e^{r\Delta})}{d-be^{r\Delta}}$, $\rho = \frac{d-b}{d-be^{r\Delta}}$,

2) $r = 0$: $p_0 = \frac{b\Delta}{1+b\Delta}$, $\rho = \frac{1}{1+b\Delta}$,

3) $r > 0$: $p_0 = \frac{d(1-e^{-r\Delta})}{b-de^{-r\Delta}}$, $\rho = \frac{(b-d)e^{-r\Delta}}{b-de^{-r\Delta}}$.

The sum $S = bdf(n,b,d,\Delta)$ is then computed as:

$S := 0$
For $i := 1$ to n do
  If $Ber(p_0) = 0$ then $S := S + Geom(\rho) + 1$
Return S.

Ber is the Bernoulli distribution, *Geom* the geometric distribution as defined p. 47.

For constant birth and death rates, this computation is exact: whatever Δ, the population size computed from function *bdf* will have the same distribution as the continuous process. But in most models b and d vary along time, and the value of the observation interval Δ has to be chosen accordingly. A possibility is to compute an adaptive Δ so as to correctly track the continuous process (see model *ferriere1.zen* for an example).

How to chose the observation interval Δ?
The birth-death process in continuous time is simulated by drawing iteratively a single birth or death event affecting a randomly chosen individual: the date of the next event is computed according to an exponential distribution with mean $\frac{1}{n} \times \frac{1}{b+d}$, and when the event has occurred, population size goes from n to $n' = n \pm 1$. When observing the continuous time birth-death process at discrete time intervals, several events can occur in a single time step Δ, driving population size from n to n'. During the time step Δ an average of $n[1 - \exp(-(b+d)\Delta)]$ events occur. For small Δ, about $n(b+d)\Delta$ events occur in average in a time step. As a result, the value $\Delta = \frac{k}{n} \times \frac{1}{b+d}$ with $k \leq n$ can track variations in population size of the order $n' \approx n \pm k$. This value of Δ can however be time consuming for small k, while the value $\Delta = \frac{1}{b+d}$ can lead to numerical instability. A compromise has to be found between efficiency and accuracy.

Functions for evolutionary dynamics

Phenotypes

The following functions allow to retrieve information from a group G containing P phenotypes. They are to be used outside the declaration of the concerned group.

x is a local group variable declared in group G

groupcardf(G) number P of phenotypes in group G.

grouppopf(G) total number N of individuals in group G, with P phenotypes and $n^{(i)}$ individuals in each phenotype (i): $N = \sum_{i=1}^{P} n^{(i)}$.

groupgrowthf(G) growth rate of phenotypes in group G, computed at time t as $\exp\left[\frac{Ln|B-D|}{t}\right]$, where B is the number of phenotypes created from $t = 1$ and D is the number of destroyed phenotypes.

grouplifetimef(G) average life duration of phenotypes in group G.

groupmeanf(x) mean value m of variable x over all individuals in group G, with $n^{(i)}$ individuals in phenotype (i) and $x^{(i)}$ the value of x in

phenotype (i)**:** $m = \frac{1}{N}\sum_{i=1}^{P} x^{(i)} n^{(i)}$

groupmaxf(x) maximum value a of variable x over all phenotypes (i) in group G: $a = \max_i(x^{(i)})$

groupminf(x) minimum value b of variable x over all phenotypes (i) in group G: $b = \min_i(x^{(i)})$

Ecological interactions

The following functions allow to compute ecological interactions between phenotypes within a group or across groups.

groupsumf(G,*exp*) value S of the sum of expression *exp* over all individuals in group G, with $n^{(i)}$ individuals in phenotype (i) and $a^{(i)}$ the value of *exp* in phenotype (i): $S = \sum_{i=1}^{P} a^{(i)} n^{(i)}$

groupsum1f(G,*exp*) value S_1 of the sum of expression *exp* over all phenotypes in group G, with $a^{(i)}$ the value of *exp* in phenotype (i): $S_1 = \sum_{i=1}^{P} a^{(i)}$

focalf(s) value of group variable s, or adaptive trait s, from the focal phenotype (i) in group G

note: function *focalf* is not to be used outside of a group.

For example, assume that ecological interactions between 2 phenotypes in group G, with adaptive traits $s1$ and $s2$, are given by a function $\alpha(s1,s2)$. Then the environment perceived (the selection pressure experienced) by a focal phenotype (i) with trait $s^{(i)}$ is computed as

$$A^{(i)} = \sum_{j=1}^{P} \alpha(s^{(i)}, s^{(j)}) n^{(i)} .$$

The corresponding declaration within group G with adaptive trait s is

defvar a = **groupsumf(**g, alpha**(focalf(**s**)**,s**)**

Function *alpha* is defined outside group G as

deffun alpha**(**s1**,**s2**)** = (…)

Triggering of mutations

magicf(*xa, ..., xz*) *xa*, …, *xz* variables

Any variable that is not a constant or a relation-variable can trigger mutations, and this is used to control for the triggering of mutations inside a time step (see **defmut** and the **trigger** keyword in section 3). Let *u* be a triggering variable. The hierarchy of ZEN variables (see the update procedure in section 3) is constructed in such a way that the mutations corresponding to *u* are triggered as soon as *u* is updated, and before any variable depending on *u* is updated. The aim of the **magicf** function is to help with this feature. When *u* is used as a triggering variable, declaring

defvar u = **magicf(**xa**,** …**,** xz**)**

ensures that *u* is updated as soon as variables *xa*, …, *xz* have been updated (as well as all variables lower than *xa*, …, *xz* in the hierarchy), and not before. Any variable depending on *xa*, …, *xz* (as well as any variable above *xa*, …, *xz* in the hierarchy) will be updated after *u*.

Date of creation of phenotype

datef(*x*) date of creation of focal phenotype, *x* any group variable
the date is the one given by option *Phenotypes* in the *Variable* | *All* panel

note: function *datef* is not to be used outside of the group scope. Any variable pertaining to the group may be used.

TECHNICAL NOTICE

Computer	PC, MAC
System	Windows, Linux, MAC OSX
Minimal memory required	128 M
Programming language	Object Pascal – Borland Delphi 6 & Kylix
Source code size	~ 13000 lines
Exec code size	~ 800 K + 4000 K runtime library *qtintf.dll*

Program bounds

General

maximum number of models in the same model file	5
maximum size of models (number of relations)	100
maximum number of relations (total)	500
maximum number of variables	5000

Evolutionary

maximum number of groups	10
maximum size of groups (number of relations)	30
maximum number of adaptive traits per group	20
maximum number of variables per group	200
maximum number of coexisting phenotypes	20000

Graphics

maximum number of graphics windows	6
maximum number of trajectories per window	4
best graphic resolution in number of time steps	≤ 10000
resolution for the distribution of phenotypes	600×600

Text

maximum number of text windows	6
maximum number of variables per window	16 (4 for Monte Carlo)
maximum number of lines per window	10000 for Monte Carlo

File

maximum number of output text files	5
maximum number of variables per file	10

ZEN DISTRIBUTIONS

Web site http://www.biologie.ens.fr/~legendre/zen/zen.html

Computer / System	DOWNLOAD	Comments
PC Windows	Self-extracting file autozen.exe	Program file zen.exe The Windows distribution also contains a console (no graphics) version zenc.exe
PC Linux	Compressed archive zen.tar.gz expanded using command tar -xzf zen.tar.gz	Program file zen For installation, consult file ZenLinuxInstall.txt The Linux version is identical to the Windows version. The Linux distribution also contains a console (no graphics) version zenc
MAC OS X	Compressed archive zenc_mac.zip	Program file zenc Console version (no graphics)

BIBLIOGRAPHY

- Benton MJ & PN Pearson. 2001. Speciation in the fossil record. Trends in Ecology and Evolution 16:405-411.
- Caswell H. 2000. Matrix population models. Second edition. Sinauer, Mass., USA.
- Champagnat N, R Ferrière & G Ben Arous. 2001. The canonical equation of adaptive dynamics: a mathematical view. Selection 2:73-83.
- Dieckmann U & R Law. 1996. The dynamical theory of coevolution: a derivation from stochastic ecological processes. Journal of Mathematical Biology 34:579-612.
- Dieckmann U, P Marrow & R Law. 1995. Evolutionary cycling in predator prey interactions: Population dynamics and the Red Queen. Journal of Theoretical Biology 176: 91-102.
- Dieckmann U & M Doebeli. 1999. On the origin of species by sympatric speciation. Nature 400:354-357.
- Doebeli M & U Dieckmann. 2003. Speciation along environmental gradients. Nature 421:259-264.
- Eshel I & U Motro. 1981. Kin selection and strong stability of mutual help. Theoretical Population Biology 19:420-433.
- Ferrière R & M Gatto. 1995. Lyapunov exponents and the mathematics of invasion in oscillatory or chaotic populations. Theoretical Population Biology 48:126-171.
- Ferrière R, F Sarrazin, S Legendre & J-P Baron. 1996. Matrix population models applied to viability analysis and conservation: Theory and practice with ULM software. Acta Œcologica 17:629-656.
- Ferrière R, JL Bronstein, S Rinaldi, R Law & M Gauduchon. 2002. Cheating and the evolutionary stability of mutualisms. Proceedings of the Royal Society of London B: 269, 773-780.
- Ferrière R, U Dieckmann & D Couvet eds. 2004. Evolutionary Conservation Biology. Cambridge University Press.
- Geritz SAH, E Kisdi, G Meszéna & JAJ Metz. 1998. Evolutionarily singular strategies and the adaptive growth and branching of the evolutionary tree. Evolutionary Ecology 12:35-57.
- Hamilton WD. 1963. The evolution of altruistic behaviour. The American Naturalist 97:354-356.
- Hamilton WD. 1964. The genetical evolution of social behaviour. I & II. Journal of Theoretical Biology 7:1-16, 17-52.
- Kisdi E. 1999. Evolutionary branching under asymmetric competition. Journal of Theoretical Biology 197:149-162
- Kisdi E & SAH Geritz. 1999. Adaptive dynamics in allele space: evolution of genetic polymorphism by small mutations in a heterogeneous environment. Evolution 53:993-1008.
- Kisdi E & SAH Geritz. 2001. Evolutionary disarmament in interspecific competition. Proceedings of the Royal Society 268 B: 2589-2594.
- Lande R. 1982. A quantitative genetic theory of life history evolution. Ecology 63:607-615.
- Law R, JL Bronstein & R Ferrière. 2001. On mutualists and exploiters: Plant-insect coevolution in pollinating seed-parasite systems. Journal of Theoretical Biology 212, 373-379.

- Legendre S. 2004. Influence of age structure and mating system on population viability. *In* Evolutionary Conservation Biology, Ferrière R, U Dieckmann & D Couvet eds., Cambridge University Press, pp. 41-58.
- Massin N & A Gonzales. 2005. Adaptive radiation in a fluctuating environment: disturbance affects the evolution of diversity in a bacterial microcosm.
- Maynard Smith J. 1966. Sympatric speciation. American Naturalist 100:637-650.
- Maynard Smith J. 1982. Evolution and the theory of games. Cambridge University Press.
- Metz JAJ, RM Nisbet & SAH Geritz. 1992. How should we define fitness for general ecological scenarios? Trends in Ecology and Evolution 7:198-202.
- Metz JAJ, SAH Geritz, G Meszéna, FJA Jacobs & JS van Heerwaarden. 1996. Adaptive dynamics, a geometrical study of the consequences of nearly faithful reproduction. *In* Stochastic and spatial structures of dynamical systems, J van Strien & SM Verduyn Lunel eds., KNAW Verhandelingen, North Holland, Amsterdam, pp 183-231.
- Meszéna G, E Kisdi, U Dieckmann, SAH Geritz & JAJ Metz. 2001. Evolutionary optimisation models and matrix games in the unified perspective of adaptive dynamics. Selection 2:193-210.
- Møller AP & S Legendre. 2001. Allee effect, sexual selection and demographic stochasticity. Oikos 92:27-34.
- Morlon H & S Legendre. *Unpublished manuscript*. A model of sexual selection using adaptive dynamics: The Fisher's runaway and its reversibility.
- Page K & M Novak. 2002. Unifying evolutionary dynamics. Journal of Theoretical Biology 219:93-98.
- Rainey PB & M Travisano. 1998. Adaptive radiation in a heterogeneous environment. Nature 394:69-72.
- Ravigné V. 2000. Spéciation et sélection de l'habitat. Rapport de DEA, Institut des Sciences de l'Evolution, Université de Montpellier II.
- Ronce O & I Olivieri. 1997. Evolution of reproductive effort in a metapopulation with local extinctions and ecological succession. American Naturalist 150:220-249.
- Rozen DE & RE Lenski. 2000. Long-term experimental evolution in *Escherichia coli*. VIII. Dynamics of a balanced polymorphism. American Naturalist 155:24-35.
- van Baalen M. 1998. Coevolution of recovery ability and virulence. Proceedings of the Royal Society of London B 265:317-325.
- van Baalen M & D Rand. 1998. The unit of selection in viscuous populations and the evolution of altruism. Journal of Theoretical Biology 193:631-648.
- van Baalen M. 2000. Pair approximations for different spatial geometries. *In* The geometry of ecological interactions: simplifying spatial complexity, JAJ Metz & U Dieckmann, eds. Cambridge University Press, pp 359-387.
- van Doorn GS & FJ Weissing. 2001. Ecological versus sexual selection models of sympatric speciation: A synthesis. Selection 2:1-2, 17-40.
- Wilke CO & C Adami. 2002. The biology of digital organism. Trends in Ecology and Evolution 17:528-532.