GenAlg

P. Baillehache

March 26, 2019

Contents

Introduction

GenAlg is a C library providing structures and functions implementing a Genetic Algorithm.

The genes are memorized as a VecFloat and/or VecShort. The user can defined a range of possible values for each gene. The user can define the size of the pool of entities and the size of the breeding pool. Selection, reproduction and mutation are designed to efficiently explore all the possible gene combination, and avoid local optimum. It is also possible to save and load

the GenAlg.

It uses the `PBErr`, `PBMath` and `GSet` libraries.

1 Definitions

A genetic algorithm has 3 steps. In a pool of entities it discards a given number of entities based on their ranking (given by a mean external to the algorithm). Then it replaces each of the discarded entity by a new one created from two selected entities from hte non discarded one. The newly created entity's properties are a mix of these two selected entities, plus a certain amount of random modification. The detail of the implementation in GenAlg of these 3 steps (selection, reproduction and mutation) are given below.

1.1 Selection

The non discarded entities are called 'elite' in GenAlg. The size of the pool of elite is configurable by the user. The selection of two elite entities is simply a random selection in the pool of elites. Selection of the same elite twice is allowed.

1.2 Reproduction

The reproduction step copies the genes of the elite entity into the new entity. Each gene has a probability of 50% to be chosen in one or the other elite.

1.3 Mutation

The mutation occurs as follow. First we calculate the probability of mutation for every gene as follow: $P = \frac{rank}{nbEntity} * (1 - \frac{1}{\sqrt{age+1}})$ where rank is the rank of the discarded entity in the pool of entities, and nbEntity is the number of entities in the pool, and age is the age of the oldest elite entity used during the reproduction step for the entity. A gene affected by a mutation according to this probability is modified as follow. The amplitude of the mutation is equal to $1 - \frac{1}{\sqrt{age+1}}$ where age is the age of the oldest elite entity used during

the reproduction step for the entity. Then the new value of the gene is equals to $gene + range * amp * (rnd + delta)$ where gene is the current value of the gene, range is equal to $max_{gene} - min_{gene}$ (the difference of the maximum allowed value for this gene and its minimum value), amp is the amplitude calculated above, rnd is a random value between -0.5 and 0.5, and delta is the mutation that has been applied to this gene in the corresponding elite entity. Genes' value is kept in bounds by bouncing it on the bounds when necessary ($gene = 2 * bound - gene$)

To counteract inbreeding (the algorithm getting stuck into a local minimum), when the diversity level of the elite pool falls below a threshold, we also reset the adn of all the non entities and all the elite entities (except the best one) having at least one diversity level with another elite entity below the diversity level of the elite pool (set to 0.01 by default). The diversity level of the whole elite pool is calculated as follow $Avg_{i,j} \frac{||\overrightarrow{adn}(elite_i) - \overrightarrow{adn}(elite_j)||}{||\overrightarrow{bound}_{max} - \overrightarrow{bound}_{min}||}$ where $\overrightarrow{adn}(elite_i)$ is the genes vector of the i-th elite entity, and $\overrightarrow{bound}_{max}$ and $\overrightarrow{bound}_{min}$ are the vector of maximum and minimum values of the genes.

Some explanation: delta bias the mutation toward the direction that improved the result at previous step; in the pool of discarded entities high ranked ones tend to have few mutations and low ranked ones tend to have more mutation, this tends to cover any posibilities of evolution; entities newly entered in the elite pool tends to produce new entities near to them (in term of distance in the genes space), while older ones tend to produce more diverse new entities, thus the exploration of solution space occurs from the vicinity of newly better solutions toward larger areas; from the previous point, a good entity tends to create a lot of similar entity, which may lead to an elite pool saturated with very similar entities (inbreeding) from which the algorithm can't escape, this is prevented by the forced mutation of elites when the inbreeding level gets too high.

2 Interface

```
// ============ GENALG.H ================

#ifndef GENALG_H
#define GENALG_H

// ================= Include =================

#include <stdlib.h>
```

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbmath.h"
#include "gset.h"
#include "respublish.h"

// ================= Define ==================

#define GABestAdnF(that) GAAdnAdnF(GABestAdn(that))
#define GABestAdnI(that) GAAdnAdnI(GABestAdn(that))

#define GENALG_NBENTITIES 100
#define GENALG_NBELITES 20

#define GENALG_TXTOMETER_NBADNDISPLAYED 40
#define GENALG_TXTOMETER_LINE1 "Epoch #xxxxxx  KTEvent #xxxxxx  \n"
#define GENALG_TXTOMETER_FORMAT1 "Epoch #%06lu  KTEvent #%06lu\n"
#define GENALG_TXTOMETER_LINE2 "Id        Age      Val\n"
#define GENALG_TXTOMETER_LINE3 "xxxxxxxx  xxxxxx  +xxxxxx.xxxxxx\n"
#define GENALG_TXTOMETER_FORMAT3 "%08lu  %06lu  %+06.6f\n"
#define GENALG_TXTOMETER_LINE4 "--------------------------------\n"
#define GENALG_TXTOMETER_LINE5 "Diversity +xxxxxx.xxxxxx        \n"
#define GENALG_TXTOMETER_FORMAT5 "Diversity %+06.6f  \n"
#define GENALG_TXTOMETER_LINE6 "Size pool xxxxxx                \n"
#define GENALG_TXTOMETER_FORMAT6 "Size pool %06d  \n"

// ------------- GenAlgAdn

// ================= Data structure ===================

typedef struct GenAlg GenAlg;

typedef struct GenAlgAdn {
  // ID
  unsigned long _id;
  // Age
  unsigned long _age;
  // Adn for floating point value
  VecFloat* _adnF;
  // Delta Adn during mutation for floating point value
  VecFloat* _deltaAdnF;
  // Adn for integer value
  VecLong* _adnI;
  // Value
  float _val;
  // Mutability of adn for floating point value
  VecFloat* _mutabilityF;
  // Mutability of adn for integer value
  VecFloat* _mutabilityI;
} GenAlgAdn;

// ================ Functions declaration ====================

// Create a new GenAlgAdn with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgAdn* GenAlgAdnCreate(const unsigned long id, const long lengthAdnF,
  const long lengthAdnI);

// Free memory used by the GenAlgAdn 'that'
```

```
void GenAlgAdnFree(GenAlgAdn** that);

// Return the adn for floating point values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* GAAdnAdnF(const GenAlgAdn* const that);

// Return the delta of adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* GAAdnDeltaAdnF(const GenAlgAdn* const that);

// Return the adn for integer values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
VecLong* GAAdnAdnI(const GenAlgAdn* const that);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga' according to the type of the GenAlg
void GAAdnInit(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga);

// Get the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetGeneF(const GenAlgAdn* const that, const long iGene);

// Get the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetDeltaGeneF(const GenAlgAdn* const that, const long iGene);

// Get the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
int GAAdnGetGeneI(const GenAlgAdn* const that, const long iGene);

// Set the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetGeneF(GenAlgAdn* const that, const long iGene,
  const float gene);

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that' to 'delta'
#if BUILDMODE != 0
inline
```

```
#endif
void GAAdnSetDeltaGeneF(GenAlgAdn* const that, const long iGene,
  const float delta);

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetGeneI(GenAlgAdn* const that, const long iGene,
  const long gene);

// Get the id of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long GAAdnGetId(const GenAlgAdn* const that);

// Get the age of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long GAAdnGetAge(const GenAlgAdn* const that);

// Get the value of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetVal(const GenAlgAdn* const that);

// Print the information about the GenAlgAdn 'that' on the
// stream 'stream'
void GAAdnPrintln(const GenAlgAdn* const that, FILE* const stream);

// Return true if the GenAlgAdn 'that' is new, i.e. is age equals 1
// Return false
#if BUILDMODE != 0
inline
#endif
bool GAAdnIsNew(const GenAlgAdn* const that);

// Copy the GenAlgAdn 'tho' into the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
void GAAdnCopy(GenAlgAdn* const that, const GenAlgAdn* const tho);

// Set the mutability vectors for the GenAlgAdn 'that' to 'mutability'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetMutabilityInt(GenAlgAdn* const that,
  const VecFloat* const mutability);
#if BUILDMODE != 0
inline
#endif
void GAAdnSetMutabilityFloat(GenAlgAdn* const that,
  const VecFloat* const mutability);

// ------------- GenAlg

// ================= Data structure ===================
```

```
typedef enum GenAlgType {
  genAlgTypeDefault,
  genAlgTypeNeuraNet,
  genAlgTypeNeuraNetConv
} GenAlgType;

// Data used when GenAlg is applied to a NeuraNet
typedef struct GANeuraNet {
  // Nb of input, hidden and output of the NeuraNet
  int _nbIn;
  int _nbHid;
  int _nbOut;
  long _nbBaseConv;
  long _nbBaseCellConv;
  long _nbLink;
} GANeuraNet;

typedef struct GenAlg {
  // GSet of GenAlgAdn, sortval == score so the head of the set is the
  // worst adn and the tail of the set is the best
  GSet* _adns;
  // Copy of the best adn
  GenAlgAdn* _bestAdn;
  // Type of the GenAlg
  GenAlgType _type;
  // Current epoch
  unsigned long _curEpoch;
  // Nb elite entities in population
  int _nbElites;
  // Id of the next new GenAlgAdn
  unsigned long _nextId;
  // Length of adn for floating point value
  const long _lengthAdnF;
  // Length of adn for integer value
  const long _lengthAdnI;
  // Bounds (min, max) for floating point values adn
  VecFloat2D* _boundsF;
  // Bounds (min, max) for integer values adn
  VecLong2D* _boundsI;
  // Norm of the range value for adns (optimization for diversity
  // calculation)
  float _normRangeFloat;
  float _normRangeInt;
  // Data used if the GenAlg is applied to a NeuraNet
  GANeuraNet _NNdata;
  // Number of ktevent
  unsigned long _nbKTEvent;
  // Flag to remember if we display info via a TextOMeter
  // about the population
  bool _flagTextOMeter;
  // DiversityThreshold
  float _diversityThreshold;
  // TextOMeter to display information about the population
  // If the TextOMEter is used, its ocntent is refreshed at each call
  // of the function GAStep();
  TextOMeter* _textOMeter;
  // Nb min of adns
  int _nbMinAdn;
  // Nb max of adns
  int _nbMaxAdn;
} GenAlg;
```

```
// ================ Functions declaration ====================

// Create a new GenAlg with 'nbEntities', 'nbElites', 'lengthAdnF'
// and 'lengthAdnI'
// 'nbEntities' must greater than 2
// 'nbElites' must greater than 1
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlg* GenAlgCreate(const int nbEntities, const int nbElites,
  const long lengthAdnF, const long lengthAdnI);

// Free memory used by the GenAlg 'that'
void GenAlgFree(GenAlg** that);

// Get the type of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
GenAlgType GAGetType(const GenAlg* const that);

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNet, the GenAlg
// will be used with a NeuraNet having 'nbIn' inputs, 'nbHid' hidden
// values and 'nbOut' outputs
#if BUILDMODE != 0
inline
#endif
void GASetTypeNeuraNet(GenAlg* const that, const int nbIn,
  const int nbHid, const int nbOut);

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNetConv,
// the GenAlg will be used with a NeuraNet having 'nbIn' inputs,
// 'nbHid' hidden values, 'nbOut' outputs, 'nbBaseConv' bases function,
// 'nbLink' links dedicated to the convolution and 'nbBaseCellConv' bases function per cell of convolution
#if BUILDMODE != 0
inline
#endif
void GASetTypeNeuraNetConv(GenAlg* const that, const int nbIn,
  const int nbHid, const int nbOut, const long nbBaseConv,
  const long nbBaseCellConv, const long nbLink);

// Return the GSet of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
GSet* GAAdns(const GenAlg* const that);

// Return the max nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbMaxAdn(const GenAlg* const that);

// Return the min nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbMinAdn(const GenAlg* const that);

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
inline
#endif
```

```
void GASetNbMaxAdn(GenAlg* const that, const int nb);

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
inline
#endif
void GASetNbMinAdn(GenAlg* const that, const int nb);

// Return the nb of entities of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbAdns(const GenAlg* const that);

// Get the diversity threshold of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
float GAGetDiversityThreshold(const GenAlg* const that);

// Set the diversity threshold of the GenAlg 'that' to 'threshold'
#if BUILDMODE != 0
inline
#endif
void GASetDiversityThreshold(GenAlg* const that, const float threshold);

// Return the nb of elites of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbElites(const GenAlg* const that);

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long GAGetCurEpoch(const GenAlg* const that);

// Return the number of KTEvent of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long GAGetNbKTEvent(const GenAlg* const that);

// Set the nb of entities of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 1, if 'nb' is lower than the current nb
// of elite the number of elite is set to 'nb' - 1
void GASetNbEntities(GenAlg* const that, const int nb);

// Set the nb of elites of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 0, if 'nb' is greater or equal to the
// current nb of entities the number of entities is set to 'nb' + 1
void GASetNbElites(GenAlg* const that, const int nb);

// Get the length of adn for floating point value
#if BUILDMODE != 0
inline
#endif
long GAGetLengthAdnFloat(const GenAlg* const that);

// Get the length of adn for integer value
#if BUILDMODE != 0
```

```
inline
#endif
long GAGetLengthAdnInt(const GenAlg* const that);

// Get the bounds for the 'iGene'-th gene of adn for floating point
// values
#if BUILDMODE != 0
inline
#endif
const VecFloat2D* GABoundsAdnFloat(const GenAlg* const that,
  const long iGene);

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
inline
#endif
const VecLong2D* GABoundsAdnInt(const GenAlg* const that,
  const long iGene);

// Set the bounds for the 'iGene'-th gene of adn for floating point
// values to a copy of 'bounds'
#if BUILDMODE != 0
inline
#endif
void GASetBoundsAdnFloat(GenAlg* const that, const long iGene,
  const VecFloat2D* const bounds);

// Set the bounds for the 'iGene'-th gene of adn for integer values
// to a copy of 'bounds'
#if BUILDMODE != 0
inline
#endif
void GASetBoundsAdnInt(GenAlg* const that, const long iGene,
  const VecLong2D* bounds);

// Get the GenAlgAdn of the GenAlg 'that' currently at rank 'iRank'
#if BUILDMODE != 0
inline
#endif
GenAlgAdn* GAAdn(const GenAlg* const that, const int iRank);

// Init the GenAlg 'that'
// Must be called after the bounds have been set
// The random generator must have been initialised before calling this
// function
void GAInit(GenAlg* const that);

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgAdn
void GAStep(GenAlg* const that);

// Print the information about the GenAlg 'that' on the stream 'stream'
void GAPrintln(const GenAlg* const that, FILE* const stream);

// Print a summary about the elite entities of the GenAlg 'that'
// on the stream 'stream'
void GAEliteSummaryPrintln(const GenAlg* const that,
  FILE* const stream);

// Get the diversity of the GenAlg 'that'
#if BUILDMODE != 0
inline
```

```
#endif
float GAGetDiversity(const GenAlg* const that);

// Function which return the JSON encoding of 'that'
JSONNode* GAEncodeAsJSON(const GenAlg* const that);

// Function which decode from JSON encoding 'json' to 'that'
bool GADecodeAsJSON(GenAlg** that, const JSONNode* const json);

// Load the GenAlg 'that' from the stream 'stream'
// If the GenAlg is already allocated, it is freed before loading
// Return true in case of success, else false
bool GALoad(GenAlg** that, FILE* const stream);

// Save the GenAlg 'that' to the stream 'stream'
// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true in case of success, else false
bool GASave(const GenAlg* const that, FILE* const stream,
  const bool compact);

// Set the value of the GenAlgAdn 'adn' of the GenAlg 'that' to 'val'
#if BUILDMODE != 0
inline
#endif
void GASetAdnValue(GenAlg* const that, GenAlgAdn* const adn,
  const float val);

// Update the norm of the range value for adans of the GenAlg 'that'
void GAUpdateNormRange(GenAlg* const that);

// Reset the GenAlg 'that'
// Randomize all the gene except those of the first adn
void GAKTEvent(GenAlg* const that);

// Return the best adn of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
const GenAlgAdn* GABestAdn(const GenAlg* const that);

// Return the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
bool GAIsTextOMeterActive(const GenAlg* const that);

// Set the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that' to 'flag'
void GASetTextOMeterFlag(GenAlg* const that, bool flag);

// ================= Polymorphism ==================

// ================ Inliner ====================

#if BUILDMODE != 0
#include "genalg-inline.c"
#endif


#endif
```

3 Code

3.1 genalg.c

```
// ============ GENALG.C ================

// ================ Include =================

#include "genalg.h"
#if BUILDMODE == 0
#include "genalg-inline.c"
#endif

// ------------- GenAlgAdn

// ================ Functions declaration ====================

// Get the diversity value of 'adnA' against 'adnB'
// The diversity is equal to
float GAAdnGetDiversity(const GenAlgAdn* const adnA,
  const GenAlgAdn* const adnB, const GenAlg* const ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga'
void GAAdnInitDefault(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
// with convolution
void GAAdnInitNeuraNetConv(const GenAlgAdn* const that,
  const GenAlg* const ga);

// ================ Functions implementation ====================

// Create a new GenAlgAdn with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgAdn* GenAlgAdnCreate(const unsigned long id,
  const long lengthAdnF, const long lengthAdnI) {
#if BUILDMODE == 0
  if (lengthAdnF < 0) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'lengthAdnF' is invalid (%ld>=0)",
      lengthAdnF);
    PBErrCatch(GenAlgErr);
  }
  if (lengthAdnI < 0) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'lengthAdnI' is invalid (%ld>=0)",
      lengthAdnI);
    PBErrCatch(GenAlgErr);
  }
#endif
  // Allocate memory
  GenAlgAdn* that = PBErrMalloc(GenAlgErr, sizeof(GenAlgAdn));
  // Set the properties
  that->_age = 1;
  that->_id = id;
```

```
  that->_val = 0.0;
  if (lengthAdnF > 0) {
    that->_adnF = VecFloatCreate(lengthAdnF);
    that->_deltaAdnF = VecFloatCreate(lengthAdnF);
    that->_mutabilityF = VecFloatCreate(lengthAdnF);
  } else {
    that->_adnF = NULL;
    that->_deltaAdnF = NULL;
    that->_mutabilityF = NULL;
  }
  if (lengthAdnI > 0) {
    that->_adnI = VecLongCreate(lengthAdnI);
    that->_mutabilityI = VecFloatCreate(lengthAdnI);
  } else {
    that->_adnI = NULL;
    that->_mutabilityI = NULL;
  }
  // Return the new GenAlgAdn
  return that;
}

// Free memory used by the GenAlgAdn 'that'
void GenAlgAdnFree(GenAlgAdn** that) {
  // Check the argument
  if (that == NULL || *that == NULL) return;
  // Free memory
  if ((*that)->_adnF != NULL)
    VecFree(&((*that)->_adnF));
  if ((*that)->_deltaAdnF != NULL)
    VecFree(&((*that)->_deltaAdnF));
  if ((*that)->_adnI != NULL)
    VecFree(&((*that)->_adnI));
  if ((*that)->_mutabilityF != NULL)
    VecFree(&((*that)->_mutabilityF));
  if ((*that)->_mutabilityI != NULL)
    VecFree(&((*that)->_mutabilityI));
  free(*that);
  // Set the pointer to null
  *that = NULL;
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga' according to the type of GenAlg
void GAAdnInit(const GenAlgAdn* const that, const GenAlg* const ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  switch (GAGetType(ga)) {
    case genAlgTypeNeuraNet:
      GAAdnInitNeuraNet(that, ga);
      break;
    case genAlgTypeNeuraNetConv:
      GAAdnInitNeuraNetConv(that, ga);
      break;
    case genAlgTypeDefault:
    default:
      GAAdnInitDefault(that, ga);
  }
```

```
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga'
void GAAdnInitDefault(const GenAlgAdn* const that,
  const GenAlg* const ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // For each floating point value gene
  for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
    float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
    float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
    float val = min + (max - min) * rnd();
    VecSet(that->_adnF, iGene, val);
    VecSet(that->_mutabilityF, iGene, 1.0);
  }
  // For each integer value gene
  for (long iGene = GAGetLengthAdnInt(ga); iGene--;) {
    long min = VecGet(GABoundsAdnInt(ga, iGene), 0);
    long max = VecGet(GABoundsAdnInt(ga, iGene), 1);
    long val = (long)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iGene, val);
    VecSet(that->_mutabilityI, iGene, 1.0);
  }
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
// with convolution
void GAAdnInitNeuraNetConv(const GenAlgAdn* const that,
  const GenAlg* const ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // For each floating point value gene
  for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
    float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
    float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
    float val = min + (max - min) * rnd();
    VecSet(that->_adnF, iGene, val);
    VecSet(that->_mutabilityF, iGene, 1.0);
  }
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
```

```
#endif
  // Init the base functions randomly
  // For each floating point value gene
  for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
    float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
    float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
    float val = min + (max - min) * rnd();
    VecSet(that->_adnF, iGene, val);
    VecSet(that->_mutabilityF, iGene, 1.0);
  }
  // Init the links by ensuring there is at least one link reaching
  // each output and use inputs as start of the initial links
  // For each integer value gene
  int shiftOut = ga->_NNdata._nbIn + ga->_NNdata._nbHid;
  for (long iGene = GAGetLengthAdnInt(ga); iGene--;) {
    VecSet(that->_adnI, iGene, -1);
    VecSet(that->_mutabilityI, iGene, 1.0);
  }
  for (int iOut = 0; iOut < ga->_NNdata._nbOut; ++iOut) {
    // The base function is randomly choosen but can't be an
    // inactive link
    long min = 0;
    long max = VecGet(GABoundsAdnInt(ga, iOut * 3), 1);
    long val = (long)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iOut * 3, val);
    // The start of the link is randomly choosen amongst inputs
    min = 0;
    max = ga->_NNdata._nbIn - 1;
    val = (long)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iOut * 3 + 1, val);
    // The end of the link is choosen sequencially amongst outputs
    VecSet(that->_adnI, iOut * 3 + 2, iOut + shiftOut);
  }
}

// Print the information about the GenAlgAdn 'that' on the
// stream 'stream'
void GAAdnPrintln(const GenAlgAdn* const that, FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (stream == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'stream' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  fprintf(stream, "id:%lu age:%lu", GAAdnGetId(that), GAAdnGetAge(that));
  fprintf(stream, "\n");
  fprintf(stream, "  adnF:");
  if (GAAdnAdnF(that) != NULL)
    VecFloatPrint(GAAdnAdnF(that), stream, 6);
  else
    fprintf(stream, "<null>");
  fprintf(stream, "\n");
  fprintf(stream, "  deltaAdnF:");
  if (GAAdnAdnF(that) != NULL)
    VecFloatPrint(GAAdnDeltaAdnF(that), stream, 6);
  else
```

```
    fprintf(stream, "<null>");
  fprintf(stream, "\n");
  fprintf(stream, "  adnI:");
  if (GAAdnAdnI(that) != NULL)
    VecPrint(GAAdnAdnI(that), stream);
  else
    fprintf(stream, "<null>");
  fprintf(stream, "\n");
}

// ------------- GenAlg

// ================ Functions declaration ====================

// Select the rank of two parents for the SRM algorithm
// Return the ranks in 'parents', with parents[0] <= parents[1]
void GASelectParents(const GenAlg* const that, int* const parents);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
void GAReproduction(GenAlg* const that, const int* const parents,
  const int iChild);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
void GAReproductionDefault(GenAlg* const that,
  const int* const parents, const int iChild);

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by inheriting whole bases from parents
void GAReproductionNeuraNetConv(GenAlg* const that,
  const int* const parents, const int iChild);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// by inheriting whole bases and links from parents
void GAReproductionNeuraNet(GenAlg* const that,
  const int* const parents, const int iChild);

// Router toward the appropriate Mute function according to the type
// of GenAlg
void GAMute(GenAlg* const that, const int* const parents,
  const int iChild);

// Mute the genes of the entity at rank 'iChild'
void GAMuteDefault(GenAlg* const that, const int* const parents,
  const int iChild);

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// by ensuring coherence in links: outputs have at least one link
// and there is no dead link
void GAMuteNeuraNet(GenAlg* const that, const int* const parents,
  const int iChild);

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by muting bases function per cell
void GAMuteNeuraNetConv(GenAlg* const that, const int* const parents,
```

```
  const int iChild);

// Refresh the content of the TextOMeter attached to the GenAlg 'that'
void GAUpdateTextOMeter(const GenAlg* const that);

// ================ Functions implementation ====================

// Create a new GenAlg with 'nbEntities', 'nbElites', 'lengthAdnF'
// and 'lengthAdnI'
// 'nbEntities' must greater than 2
// 'nbElites' must greater than 1
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlg* GenAlgCreate(const int nbEntities, const int nbElites,
  const long lengthAdnF, const long lengthAdnI) {
  // Allocate memory
  GenAlg* that = PBErrMalloc(GenAlgErr, sizeof(GenAlg));
  // Set the properties
  that->_type = genAlgTypeDefault;
  that->_adns = GSetCreate();
  that->_curEpoch = 0;
  that->_nbKTEvent = 0;
  that->_flagTextOMeter = false;
  that->_diversityThreshold = 0.01;
  that->_textOMeter = NULL;
  that->_nbMinAdn = nbEntities;
  that->_nbMaxAdn = nbEntities;
  that->_bestAdn = GenAlgAdnCreate(0, lengthAdnF, lengthAdnI);
  *(long*)&(that->_lengthAdnF) = lengthAdnF;
  *(long*)&(that->_lengthAdnI) = lengthAdnI;
  if (lengthAdnF > 0) {
    that->_boundsF =
      PBErrMalloc(GenAlgErr, sizeof(VecFloat2D) * lengthAdnF);
    for (long iGene = lengthAdnF; iGene--;)
      that->_boundsF[iGene] = VecFloatCreateStatic2D();
  } else
    that->_boundsF = NULL;
  if (lengthAdnI > 0) {
    that->_boundsI =
      PBErrMalloc(GenAlgErr, sizeof(VecLong2D) * lengthAdnI);
    for (long iGene = lengthAdnI; iGene--;)
      that->_boundsI[iGene] = VecLongCreateStatic2D();
  } else
    that->_boundsI = NULL;
  that->_normRangeFloat = 1.0;
  that->_normRangeInt = 1.0;
  that->_nbElites = 0;
  that->_nextId = 0;
  GASetNbEntities(that, nbEntities);
  GASetNbElites(that, nbElites);
  // Return the new GenAlg
  return that;
}

// Free memory used by the GenAlg 'that'
void GenAlgFree(GenAlg** that) {
  // Check the argument
  if (that == NULL || *that == NULL) return;
  // Free memory
  GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(*that));
  do {
    GenAlgAdn* gaEnt = GSetIterGet(&iter);
    GenAlgAdnFree(&gaEnt);
```

```
  } while (GSetIterStep(&iter));
  GSetFree(&((*that)->_adns));
  if ((*that)->_boundsF != NULL)
    free((*that)->_boundsF);
  if ((*that)->_boundsI != NULL)
    free((*that)->_boundsI);
  GenAlgAdnFree(&((*that)->_bestAdn));
  if ((*that)->_textOMeter != NULL) {
    TextOMeterFree(&((*that)->_textOMeter));
  }
  free(*that);
  // Set the pointer to null
  *that = NULL;
}

// Set the nb of entities of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 1, if 'nb' is lower than the current nb
// of elite the number of elite is set to 'nb' - 1
void GASetNbEntities(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (nb <= 1) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
    PBErrCatch(GenAlgErr);
  }
#endif
  while (GSetNbElem(GAAdns(that)) > nb) {
    GenAlgAdn* gaEnt = GSetPop(GAAdns(that));
    GenAlgAdnFree(&gaEnt);
  }
  while (GSetNbElem(GAAdns(that)) < nb) {
    GenAlgAdn* ent = GenAlgAdnCreate(that->_nextId++,
      GAGetLengthAdnFloat(that), GAGetLengthAdnInt(that));
    GSetPush(GAAdns(that), ent);
  }
  if (GAGetNbElites(that) >= nb)
    GASetNbElites(that, nb - 1);
}

// Set the nb of elites of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 0, if 'nb' is greater or equal to the
// current nb of entities the number of entities is set to 'nb' + 1
void GASetNbElites(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (nb <= 1) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
    PBErrCatch(GenAlgErr);
  }
#endif
  if (GAGetNbAdns(that) <= nb)
    GASetNbEntities(that, nb + 1);
```

```
  that->_nbElites = nb;
}

// Init the GenAlg 'that'
// Must be called after the bounds have been set
// The random generator must have been initialised before calling this
// function
void GAInit(GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // For each adn
  GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(that));
  do {
    // Get the adn
    GenAlgAdn* adn = GSetIterGet(&iter);
    // Initialise randomly the genes of the adn
    GAAdnInit(adn, that);
  } while (GSetIterStep(&iter));
  GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
}

// Reset the GenAlg 'that'
// Randomize all the gene except those of the best adn
void GAKTEvent(GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // No KTEvent on the first epoch
  if (GAGetCurEpoch(that) == 0)
    return;
  // Get the diversity level
  //float diversity = GAGetDiversity(that);
  // Loop until the diversity of the elites is sufficient
  int nbKTEvent = 0;
  int nbMaxLoop =
    (int)round((float)GAGetNbAdns(that) / (float)GAGetNbElites(that));
  do {
    nbKTEvent = 0;
    // For each pair of adn
    for (int iAdn = GAGetNbElites(that) - 1; iAdn >= 1 ; --iAdn) {
      for (int jAdn = iAdn - 1; jAdn >= 0 ; --jAdn) {
        // Get the diversity of this pair
        float div = fabs(GAAdnGetVal(GAAdn(that, iAdn)) -
          GAAdnGetVal(GAAdn(that, jAdn)));
        // If it's below the diversity threshold
        if (div <= GAGetDiversityThreshold(that)) {
          GenAlgAdn* adn = GAAdn(that, iAdn);
          GAAdnInit(adn, that);
          adn->_age = 1;
          adn->_id = (that->_nextId)++;
          GASetAdnValue(that, adn,
            GAAdnGetVal(GAAdn(that, GAGetNbAdns(that) - 1)));
          jAdn = 0;
```

```
          ++nbKTEvent;
        }
      }
    }
    // If their has been KTEvent
    if (nbKTEvent > 0) {
      // We need to sort the adns
      GSetSort(GAAdns(that));
      // Memorize the total number of KTEvent
      that->_nbKTEvent += nbKTEvent;
    }
    --nbMaxLoop;
  } while (nbKTEvent > 0 && nbMaxLoop > 0);
  // If the best adn is older than (nb elite) epochs
  if (GAAdnGetAge(GAAdn(that, 0)) >= (unsigned int)GAGetNbElites(that)) {
    // Get the diversity relatively to the best of all
    float div = fabs(GAAdnGetVal(GAAdn(that, 0)) -
      GAAdnGetVal(GABestAdn(that)));
    // If it's below the diversity threshold or the it's older than
    // (pool size) epochs
    if (div <= GAGetDiversityThreshold(that) ||
      GAAdnGetAge(GAAdn(that, 0)) >= (unsigned int)GAGetNbAdns(that)) {
      GenAlgAdn* adn = GAAdn(that, 0);
      GAAdnInit(adn, that);
      adn->_age = 1;
      adn->_id = (that->_nextId)++;
      GASetAdnValue(that, adn,
        GAAdnGetVal(GAAdn(that, GAGetNbAdns(that) - 1)));
      // We need to sort the adns
      GSetSort(GAAdns(that));
      // Memorize the total number of KTEvent
      that->_nbKTEvent += 1;
    }
  }

  /*++(that->_nbKTEvent);
  GenAlgAdn* adn = GAAdn(that, 0);
  unsigned long int age = adn->_age;
  GAAdnCopy(adn, GABestAdn(that));
  adn->_age = age;
  int parents[2] = {0};
  GAMute(that, parents, 0);
  adn->_age = 1;
  for (int iEnt = 1; iEnt < GAGetNbAdns(that);
    ++iEnt) {
    GenAlgAdn* adn = GAAdn(that, iEnt);
    GAAdnInit(adn, that);
    adn->_age = 1;
    adn->_id = (that->_nextId)++;
  }*/
}

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgAdn
void GAStep(GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
```

```
// Selection, Reproduction, Mutation
// Ensure the set of adns is sorted
GSetSort(GAAdns(that));
// Variable to memorize if there has been improvement
bool flagImprov = false;
// Update the best adn if necessary
if (that->_curEpoch == 1) {
  GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
  that->_bestAdn->_age = that->_curEpoch + 1;
} else {
  if (GAAdnGetVal(GAAdn(that, 0)) > GAAdnGetVal(GABestAdn(that))) {
    GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
    that->_bestAdn->_age = that->_curEpoch + 1;
    flagImprov = true;
  }
}
// Ensure diversity level
GAKTEvent(that);
// Refresh the TextOMeter if necessary
if (that->_flagTextOMeter) {
  GAUpdateTextOMeter(that);
}
// Resize the population according to the improvement
if (that->_curEpoch > 1) {
  if (flagImprov) {
    GASetNbEntities(that,
      MAX(GAGetNbMinAdn(that), GAGetNbAdns(that) / 2));
  } else {
    GASetNbEntities(that,
      MIN(GAGetNbMaxAdn(that), 2 * GAGetNbAdns(that)));
  }
}

/*// Get the diversity level
float diversity = GAGetDiversity(that);
// Correct the diversity level with the age of the best adn
//diversity *=
  //1.0 - fsquare((float)(GAAdnGetAge(GAAdn(that, 0))) / 1000.0);
// If the diversity level is too low
if (that->_curEpoch > 1 &&
  (diversity < GAGetDiversityThreshold(that) ||
  GAAdnGetAge(GAAdn(that, 0)) > 200)) {
  // Renew diversity by applying a KT event (in memory of
  // chickens' grand pa and grand ma)
  GAKTEvent(that);
// Else, the diversity level is ok
} else { */
  // For each adn which is an elite
  for (int iAdn = 0; iAdn < GAGetNbElites(that); ++iAdn) {
    // Increment age
    ++(GAAdn(that, iAdn)->_age);
  }
  // For each adn which is not an elite
  for (int iAdn = GAGetNbElites(that); iAdn < GAGetNbAdns(that);
    ++iAdn) {
    // Declare a variable to memorize the parents
    int parents[2];
    // Select two parents for this adn
    GASelectParents(that, parents);
    // Set the genes of the adn as a 50/50 mix of parents' genes
    GAReproduction(that, parents, iAdn);
    // Mute the genes of the adn
```

```
      GAMute(that, parents, iAdn);
    }
  //}
  // Increment the number of epochs
  ++(that->_curEpoch);
}

// Select the rank of two parents for the SRM algorithm
// Return the ranks in 'parents', with parents[0] <= parents[1]
void GASelectParents(const GenAlg* const that, int* const parents) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Declare a variable to memorize the parents' rank
  int p[2];
  do {
    for (int i = 2; i--;)
      // p[i] below may be equal to the rank of the highest non elite
      // adn, but it's not a problem so leave it and let's call that
      // the Hawking radiation of this function in memory of this great
      // man.
      p[i] = (int)floor(rnd() * (float)GAGetNbElites(that));
  } while (p[0] == p[1]);
  // Memorize the sorted parents' rank
  if (p[0] < p[1]) {
    parents[0] = p[0];
    parents[1] = p[1];
  } else {
    parents[0] = p[1];
    parents[1] = p[0];
  }
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
void GAReproduction(GenAlg* const that,
  const int* const parents, const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
```

```
  }
#endif
  switch (GAGetType(that)) {
    case genAlgTypeNeuraNet:
      GAReproductionNeuraNet(that, parents, iChild);
      break;
    case genAlgTypeNeuraNetConv:
      GAReproductionNeuraNetConv(that, parents, iChild);
      break;
    case genAlgTypeDefault:
    default:
      GAReproductionDefault(that, parents, iChild);
  }
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// by inheriting whole bases and links from parents
void GAReproductionNeuraNet(GenAlg* const that,
  const int* const parents, const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the parents and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* parentB = GAAdn(that, parents[1]);
  GenAlgAdn* child = GAAdn(that, iChild);
  // For each gene of the adn for floating point value
  for (long iGene = 0; iGene < GAGetLengthAdnFloat(that); iGene += 3) {
    // Get the gene from one parent or the other with equal
    // probabililty
    if (rnd() < 0.5) {
      for (long jGene = 3; jGene--;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentA->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentA->_deltaAdnF, iGene + jGene));
      }
    } else {
      for (long jGene = 3; jGene--;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentB->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentB->_deltaAdnF, iGene + jGene));
      }
    }
```

```
  }
  // For each gene of the adn for int value
  for (long iGene = 0; iGene < GAGetLengthAdnInt(that); iGene += 3) {
    // Get the gene from one parent or the other with equal probabililty
    if (rnd() < 0.5) {
      for (long jGene = 3; jGene--;)
        VecSet(child->_adnI, iGene + jGene,
          VecGet(parentA->_adnI, iGene + jGene));
    } else {
      for (long jGene = 3; jGene--;)
        VecSet(child->_adnI, iGene + jGene,
          VecGet(parentB->_adnI, iGene + jGene));
    }
  }
  // Reset the age of the child
  child->_age = 1;
  // Set the id of the child
  child->_id = (that->_nextId)++;
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by inheriting whole bases from parents
void GAReproductionNeuraNetConv(GenAlg* const that,
  const int* const parents, const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the parents and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* parentB = GAAdn(that, parents[1]);
  GenAlgAdn* child = GAAdn(that, iChild);
  // For each gene of the adn for floating point value of convolution
  // base functions
  for (long iGene = 0;
    iGene < that->_NNdata._nbBaseConv * 3;
    iGene += that->_NNdata._nbBaseCellConv * 3) {
    // Get the gene from one parent or the other with equal probabililty
    if (rnd() < 0.5) {
      for (long jGene = that->_NNdata._nbBaseCellConv * 3;
        jGene--;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentA->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentA->_deltaAdnF, iGene + jGene));
      }
```

```
    } else {
      for (long jGene = that->_NNdata._nbBaseCellConv * 3;
        jGene--;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentB->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentB->_deltaAdnF, iGene + jGene));
      }
    }
  }
  // For each gene of the adn for floating point value of convolution
  // base functions
  for (long iGene = that->_NNdata._nbBaseConv * 3;
    iGene < GAGetLengthAdnFloat(that); iGene += 3) {
    // Get the gene from one parent or the other with equal probabililty
    if (rnd() < 0.5) {
      for (long jGene = 3; --jGene;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentA->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentA->_deltaAdnF, iGene + jGene));
      }
    } else {
      for (long jGene = 3; --jGene;) {
        VecSet(child->_adnF, iGene + jGene,
          VecGet(parentB->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
          VecGet(parentB->_deltaAdnF, iGene + jGene));
      }
    }
  }
  // Reset the age of the child
  child->_age = 1;
  // Set the id of the child
  child->_id = (that->_nextId)++;
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
void GAReproductionDefault(GenAlg* const that,
  const int* const parents, const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the parents and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* parentB = GAAdn(that, parents[1]);
```

```
  GenAlgAdn* child = GAAdn(that, iChild);
  // For each gene of the adn for floating point value
  for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
    // Get the gene from one parent or the other with equal probabililty
    if (rnd() < 0.5) {
      VecSet(child->_adnF, iGene, VecGet(parentA->_adnF, iGene));
      VecSet(child->_deltaAdnF, iGene,
        VecGet(parentA->_deltaAdnF, iGene));
    } else {
      VecSet(child->_adnF, iGene, VecGet(parentB->_adnF, iGene));
      VecSet(child->_deltaAdnF, iGene,
        VecGet(parentB->_deltaAdnF, iGene));
    }
  }
  // For each gene of the adn for int value
  for (long iGene = GAGetLengthAdnInt(that); iGene--;) {
    // Get the gene from one parent or the other with equal probabililty
    if (rnd() < 0.5)
      VecSet(child->_adnI, iGene, VecGet(parentA->_adnI, iGene));
    else
      VecSet(child->_adnI, iGene, VecGet(parentB->_adnI, iGene));
  }
  // Reset the age of the child
  child->_age = 1;
  // Set the id of the child
  child->_id = (that->_nextId)++;
}

// Router toward the appropriate Mute function according to the type
// of GenAlg
void GAMute(GenAlg* const that, const int* const parents,
  const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  switch (GAGetType(that)) {
    case genAlgTypeNeuraNet:
      GAMuteNeuraNet(that, parents, iChild);
      break;
    case genAlgTypeNeuraNetConv:
      GAMuteNeuraNetConv(that, parents, iChild);
      break;
    case genAlgTypeDefault:
    default:
      GAMuteDefault(that, parents, iChild);
  }
}
```

```
// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// by ensuring coherence in links: outputs have at least one link
// and there is no dead link
void GAMuteNeuraNet(GenAlg* const that, const int* const parents,
  const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the first parent and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* child = GAAdn(that, iChild);
  // Get the proba and amplitude of mutation
  float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
  float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age + 1));
  probMute /= (float)(GAGetLengthAdnInt(that));
  probMute += (float)(parentA->_age) / 10000.0;
  // Ensure the proba is not null
  if (probMute < PBMATH_EPSILON)
    probMute = PBMATH_EPSILON;
  // Declare a variable to memorize if there has been mutation
  bool hasMuted = false;
  // Declare a variable to memorize the used values amongst input and
  // hidden
  long nbMaxUsedVal = that->_NNdata._nbIn + that->_NNdata._nbHid;
  char* isUsed = PBErrMalloc(GenAlgErr, sizeof(char) * nbMaxUsedVal);
  // Loop until there has been at least one mutation
  do {
    // Reset the used values
    memset(isUsed, 0, sizeof(char) * nbMaxUsedVal);
    memset(isUsed, 1, sizeof(char) * that->_NNdata._nbIn);
    // For each gene of the adn for int value (links definitions)
    for (long iGene = 0; iGene < GAGetLengthAdnInt(that); iGene += 3) {
      // If the link mutes
      if (rnd() < probMute) {
        hasMuted= true;
        // If this link is currently inactivated
        if (GAAdnGetGeneI(child, iGene) == -1) {
          // Base function
          long iBase = (int)round((float)iGene / 3.0);
          GAAdnSetGeneI(child, iGene, iBase);
          // Input
          long min =
            VecGet(GABoundsAdnInt(that, iGene + 1), 0);
          long max =
            VecGet(GABoundsAdnInt(that, iGene + 1), 1);
```

```
    long val = min;
    // Ensure the input is a used value
    do {
      val = (long)round((float)min +
        (float)(max - min) * rnd());
    } while (isUsed[val] == 0);
    GAAdnSetGeneI(child, iGene + 1, val);
    // Output
    min = MAX(val, VecGet(GABoundsAdnInt(that, iGene + 2), 0));
    max = VecGet(GABoundsAdnInt(that, iGene + 2), 1);
    val = (long)round((float)min + (float)(max - min) * rnd());
    GAAdnSetGeneI(child, iGene + 2, val);
    if (val < nbMaxUsedVal)
      isUsed[val] = 1;
  // Else, this link is currently activated
  } else {
    // Choose between inactivation or mutation
    if (rnd() < 0.5) {
      // Inactivate the link
      GAAdnSetGeneI(child, iGene, -1);
    } else {
      // Input
      long min =
        VecGet(GABoundsAdnInt(that, iGene + 1), 0);
      long max =
        VecGet(GABoundsAdnInt(that, iGene + 1), 1);
      long val = min;
      // Ensure the input is a used value
      do {
        val = (long)round((float)min +
          (float)(max - min) * rnd());
      } while (isUsed[val] == 0);
      GAAdnSetGeneI(child, iGene + 1, val);
      // Output
      min = MAX(val, VecGet(GABoundsAdnInt(that, iGene + 2), 0));
      max = VecGet(GABoundsAdnInt(that, iGene + 2), 1);
      val = (long)round((float)min + (float)(max - min) * rnd());
      GAAdnSetGeneI(child, iGene + 2, val);
      if (val < nbMaxUsedVal)
        isUsed[val] = 1;
    }
  }
}
// Get the index of the base function
long baseFun = GAAdnGetGeneI(child, iGene);
// If the link is active
if (baseFun != -1) {
  // If the associated base function mutes
  if (rnd() < probMute) {
    long baseFunGene = baseFun * 3;
    for (long jGene = 3; jGene--;) {
      // Get the bounds
      const VecFloat2D* const bounds =
        GABoundsAdnFloat(that, baseFunGene + jGene);
      // Declare a variable to memorize the previous value
      // of the gene
      float prevVal = GAAdnGetGeneF(child, baseFunGene + jGene);
      // Apply the mutation
      GAAdnSetGeneF(child, baseFunGene + jGene,
        GAAdnGetGeneF(child, baseFunGene + jGene) +
        (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
        //VecGet(parentA->_mutabilityF, baseFun * 3) *
```

```
              (rnd() - 0.5) +
              GAAdnGetDeltaGeneF(child, baseFunGene + jGene));
            // Keep the gene value in bounds
            while (GAAdnGetGeneF(child, baseFunGene + jGene) <
              VecGet(bounds, 0) ||
              GAAdnGetGeneF(child, baseFunGene + jGene) >
              VecGet(bounds, 1)) {
              if (GAAdnGetGeneF(child, baseFunGene + jGene) >
                VecGet(bounds, 1))
                GAAdnSetGeneF(child, baseFunGene + jGene,
                  2.0 * VecGet(bounds, 1) -
                  GAAdnGetGeneF(child, baseFunGene + jGene));
              else if (GAAdnGetGeneF(child, baseFunGene + jGene) <
                VecGet(bounds, 0))
                GAAdnSetGeneF(child, baseFunGene + jGene,
                  2.0 * VecGet(bounds, 0) -
                  GAAdnGetGeneF(child, baseFunGene + jGene));
            }
            // Update the deltaAdn
            GAAdnSetDeltaGeneF(child, baseFunGene + jGene,
              GAAdnGetGeneF(child, baseFunGene + jGene) - prevVal);
          }
        }
      }
    }
  } while (hasMuted == false);
  free(isUsed);
}

// Mute the genes of the entity at rank 'iChild'
void GAMuteDefault(GenAlg* const that, const int* const parents,
  const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the first parent and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* child = GAAdn(that, iChild);
  // Get the proba amplitude of mutation
  float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
  float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age));
  probMute /= (float)(MAX(GAGetLengthAdnInt(that),
    GAGetLengthAdnFloat(that)));
  probMute += (float)(parentA->_age) / 10000.0;
  if (probMute < PBMATH_EPSILON)
    probMute = PBMATH_EPSILON;
  bool hasMuted = false;
```

```
  do {
    // For each gene of the adn for floating point value
    for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
      // If this gene mutes
      if (rnd() < probMute) {
        hasMuted = true;
        // Get the bounds
        const VecFloat2D* const bounds = GABoundsAdnFloat(that, iGene);
        // Declare a variable to memorize the previous value of the gene
        float prevVal = GAAdnGetGeneF(child, iGene);
        // Apply the mutation
        GAAdnSetGeneF(child, iGene, GAAdnGetGeneF(child, iGene) +
          (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
          (rnd() - 0.5) + GAAdnGetDeltaGeneF(child, iGene));
        // Keep the gene value in bounds
        while (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0) ||
          GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1)) {
          if (GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1))
            GAAdnSetGeneF(child, iGene,
              2.0 * VecGet(bounds, 1) - GAAdnGetGeneF(child, iGene));
          else if (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0))
            GAAdnSetGeneF(child, iGene,
              2.0 * VecGet(bounds, 0) - GAAdnGetGeneF(child, iGene));
        }
        // Update the deltaAdn
        GAAdnSetDeltaGeneF(child, iGene,
          GAAdnGetGeneF(child, iGene) - prevVal);
      }
    }
    // For each gene of the adn for int value
    for (long iGene = GAGetLengthAdnInt(that); iGene--;) {
      // If this gene mutes
      if (rnd() < probMute) {
        hasMuted = true;
        // Get the bounds
        const VecLong2D* const boundsI = GABoundsAdnInt(that, iGene);
        VecFloat2D bounds = VecLongToFloat2D(boundsI);
        // Apply the mutation (as it is int value, ensure the amplitude
        // is big enough to have an effect
        float ampI = MIN(2.0,
          (float)(VecGet(&bounds, 1) - VecGet(&bounds, 0)) * amp);
        GAAdnSetGeneI(child, iGene, GAAdnGetGeneI(child, iGene) +
          (long)round(ampI * (rnd() - 0.5)));
        // Keep the gene value in bounds
        while (GAAdnGetGeneI(child, iGene) < VecGet(&bounds, 0) ||
          GAAdnGetGeneI(child, iGene) > VecGet(&bounds, 1)) {
          if (GAAdnGetGeneI(child, iGene) > VecGet(&bounds, 1))
            GAAdnSetGeneI(child, iGene,
              2 * VecGet(&bounds, 1) - GAAdnGetGeneI(child, iGene));
          else if (GAAdnGetGeneI(child, iGene) < VecGet(&bounds, 0))
            GAAdnSetGeneI(child, iGene,
              2 * VecGet(&bounds, 0) - GAAdnGetGeneI(child, iGene));
        }
      }
    }
  } while (hasMuted == false);
}

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by muting bases function per cell
void GAMuteNeuraNetConv(GenAlg* const that, const int* const parents,
```

```
  const int iChild) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (parents == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'parents' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the first parent and child
  GenAlgAdn* parentA = GAAdn(that, parents[0]);
  GenAlgAdn* child = GAAdn(that, iChild);
  // Get the proba amplitude of mutation
  float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
  float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age));
  probMute /= (float)(that->_NNdata._nbLink);
  probMute += (float)(parentA->_age) / 10000.0;
  if (probMute < PBMATH_EPSILON)
    probMute = PBMATH_EPSILON;
  bool hasMuted = false;
  int nbTry = 0;
  do {
    // For each gene of the adn for floating point value
    for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
      // If this gene mutes
      if (rnd() < probMute * VecGet(parentA->_mutabilityF, iGene)) {
        hasMuted = true;
        // Get the bounds
        const VecFloat2D* const bounds = GABoundsAdnFloat(that, iGene);
        // Declare a variable to memorize the previous value of the gene
        float prevVal = GAAdnGetGeneF(child, iGene);
        // Apply the mutation
        GAAdnSetGeneF(child, iGene, GAAdnGetGeneF(child, iGene) +
          (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
          (rnd() - 0.5) + GAAdnGetDeltaGeneF(child, iGene));
        // Keep the gene value in bounds
        while (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0) ||
          GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1)) {
          if (GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1))
            GAAdnSetGeneF(child, iGene,
              2.0 * VecGet(bounds, 1) - GAAdnGetGeneF(child, iGene));
          else if (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0))
            GAAdnSetGeneF(child, iGene,
              2.0 * VecGet(bounds, 0) - GAAdnGetGeneF(child, iGene));
        }
        // Update the deltaAdn
        GAAdnSetDeltaGeneF(child, iGene,
          GAAdnGetGeneF(child, iGene) - prevVal);
      }
    }
    ++nbTry;
  } while (hasMuted == false && nbTry < 10);
```

```
}

// Print the information about the GenAlg 'that' on the stream 'stream'
void GAPrintln(const GenAlg* const that, FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (stream == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'stream' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  fprintf(stream, "epoch:%lu\n", GAGetCurEpoch(that));
  fprintf(stream, "%d entities, %d elites\n", GAGetNbAdns(that),
    GAGetNbElites(that));
  GSetIterBackward iter = GSetIterBackwardCreateStatic(GAAdns(that));
  int iEnt = 0;
  do {
    GenAlgAdn* ent = GSetIterGet(&iter);
    fprintf(stream, "#%d value:%f ", iEnt,
      GSetIterGetElem(&iter)->_sortVal);
    if (iEnt < GAGetNbElites(that))
      fprintf(stream, "elite ");
    GAAdnPrintln(ent, stream);
    ++iEnt;
  } while (GSetIterStep(&iter));
}

// Print a summary about the elite entities of the GenAlg 'that'
// on the stream 'stream'
void GAEliteSummaryPrintln(const GenAlg* const that,
  FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (stream == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'stream' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  GSetIterBackward iter = GSetIterBackwardCreateStatic(GAAdns(that));
  int iEnt = 0;
  GenAlgAdn* leader = GSetIterGet(&iter);
  fprintf(stream, "(age,val,div) ");
  do {
    GenAlgAdn* ent = GSetIterGet(&iter);
    fprintf(stream, "(%lu,%.3f,%.3f) ", GAAdnGetAge(ent),
      GSetIterGetElem(&iter)->_sortVal,
      GAAdnGetDiversity(ent, leader, that));
    ++iEnt;
  } while (GSetIterStep(&iter) && iEnt < GAGetNbElites(that));
  fprintf(stream, "\n");
}
```

```
// Update the norm of the range value for adans of the GenAlg 'that'
void GAUpdateNormRange(GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // If there are float adn
  if (GAGetLengthAdnFloat(that) > 0) {
    // Declare a vector to memorize the ranges in float gene values
    VecFloat* range = VecFloatCreate(GAGetLengthAdnFloat(that));
    // Calculate the ranges in gene values
    for (long iGene = GAGetLengthAdnFloat(that); iGene--;)
      VecSet(range, iGene,
        VecGet(GABoundsAdnFloat(that, iGene), 1) -
        VecGet(GABoundsAdnFloat(that, iGene), 0));
    // Calculate the norm of the range
    that->_normRangeFloat = VecNorm(range);
    // Free memory
    VecFree(&range);
  }

  // If there are int adn
  if (GAGetLengthAdnInt(that) > 0) {
    // Declare a vector to memorize the ranges in int gene values
    VecFloat* range = VecFloatCreate(GAGetLengthAdnInt(that));
    // Calculate the ranges in gene values
    for (long iGene = GAGetLengthAdnInt(that); iGene--;)
      VecSet(range, iGene,
        VecGet(GABoundsAdnInt(that, iGene), 1) -
        VecGet(GABoundsAdnInt(that, iGene), 0));
    // Calculate the norm of the range
    that->_normRangeInt = VecNorm(range);
    // Free memory
    VecFree(&range);
  }
}


// Get the diversity value of 'adnA' against 'adnB'
// The diversity is equal to
float GAAdnGetDiversity(const GenAlgAdn* const adnA,
  const GenAlgAdn* const adnB, const GenAlg* const ga) {
#if BUILDMODE == 0
  if (adnA == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'adnA' is null");
    PBErrCatch(GenAlgErr);
  }
  if (adnB == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'adnB' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Declare a variable to memorize the result
  float diversity = 0.0;
  // If there are adn for floating point values
  if (GAAdnAdnF(adnA) != NULL && GAAdnAdnF(adnB) != NULL) {
    // Get the difference in adn with the first entity
```

```
    VecFloat* diff =
      VecGetOp(GAAdnAdnF(adnA), 1.0, GAAdnAdnF(adnB), -1.0);
    // Calculate the diversity
    diversity += VecNorm(diff) / ga->_normRangeFloat;
    // Free memory
    VecFree(&diff);
  }
  // If there are adn for int values
  if (GAAdnAdnI(adnA) != NULL && GAAdnAdnI(adnB) != NULL) {
    // Get the difference in adn with the first entity
    VecLong* diffI =
      VecGetOp(GAAdnAdnI(adnA), 1, GAAdnAdnI(adnB), -1);
    VecFloat* diff = VecLongToFloat(diffI);
    // Calculate the diversity
    diversity += VecNorm(diff) / ga->_normRangeInt;
    // Free memory
    VecFree(&diffI);
    VecFree(&diff);
  }
  // Correct diversity if there was both float and int adns
  if (GAAdnAdnF(adnA) != NULL && GAAdnAdnF(adnB) != NULL &&
    GAAdnAdnI(adnA) != NULL && GAAdnAdnI(adnB) != NULL)
    diversity /= 2.0;
  // Return the result
  return diversity;
}

// Function which return the JSON encoding of 'that'
JSONNode* GAAdnEncodeAsJSON(const GenAlgAdn* const that,
  const float elo) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
#endif
  // Create the JSON structure
  JSONNode* json = JSONCreate();
  // Declare a buffer to convert value into string
  char val[100];
  // Encode the id
  sprintf(val, "%lu", that->_id);
  JSONAddProp(json, "_id", val);
  // Encode the age
  sprintf(val, "%lu", that->_age);
  JSONAddProp(json, "_age", val);
  // Encode the elo
  sprintf(val, "%f", elo);
  JSONAddProp(json, "_elo", val);
  // Encode the value
  sprintf(val, "%f", that->_val);
  JSONAddProp(json, "_val", val);
  // Encode the genes
  if (that->_adnF != NULL) {
    JSONAddProp(json, "_adnF", VecEncodeAsJSON(that->_adnF));
    JSONAddProp(json, "_deltaAdnF", VecEncodeAsJSON(that->_deltaAdnF));
  }
  if (that->_adnI != NULL)
    JSONAddProp(json, "_adnI", VecEncodeAsJSON(that->_adnI));
  // Return the created JSON
  return json;
```

```
}

// Function which return the JSON encoding of 'that'
JSONNode* GAEncodeAsJSON(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
#endif
  // Create the JSON structure
  JSONNode* json = JSONCreate();
  // Declare a buffer to convert value into string
  char val[100];
  // Encode the type
  sprintf(val, "%d", GAGetType(that));
  JSONAddProp(json, "_type", val);
  switch (GAGetType(that)) {
    case genAlgTypeNeuraNet:
      sprintf(val, "%d", that->_NNdata._nbIn);
      JSONAddProp(json, "NN_nbIn", val);
      sprintf(val, "%d", that->_NNdata._nbHid);
      JSONAddProp(json, "NN_nbHid", val);
      sprintf(val, "%d", that->_NNdata._nbOut);
      JSONAddProp(json, "NN_nbOut", val);
      break;
    case genAlgTypeNeuraNetConv:
      sprintf(val, "%d", that->_NNdata._nbIn);
      JSONAddProp(json, "NN_nbIn", val);
      sprintf(val, "%d", that->_NNdata._nbHid);
      JSONAddProp(json, "NN_nbHid", val);
      sprintf(val, "%d", that->_NNdata._nbOut);
      JSONAddProp(json, "NN_nbOut", val);
      sprintf(val, "%ld", that->_NNdata._nbBaseConv);
      JSONAddProp(json, "NN_nbBaseConv", val);
      sprintf(val, "%ld", that->_NNdata._nbBaseCellConv);
      JSONAddProp(json, "NN_nbBaseCellConv", val);
      sprintf(val, "%ld", that->_NNdata._nbLink);
      JSONAddProp(json, "NN_nbLink", val);
      break;
    default:
      break;
  }
  // Encode the nb adns
  sprintf(val, "%d", GAGetNbAdns(that));
  JSONAddProp(json, "_nbAdns", val);
  // Encode the nb elites
  sprintf(val, "%d", GAGetNbElites(that));
  JSONAddProp(json, "_nbElites", val);
  // Encode the length adn float
  sprintf(val, "%ld", GAGetLengthAdnFloat(that));
  JSONAddProp(json, "_lengthAdnF", val);
  // Encode the length adn int
  sprintf(val, "%ld", GAGetLengthAdnInt(that));
  JSONAddProp(json, "_lengthAdnI", val);
  // Encode the epoch
  sprintf(val, "%lu", GAGetCurEpoch(that));
  JSONAddProp(json, "_curEpoch", val);
  // Encode the next id
  sprintf(val, "%lu", that->_nextId);
  JSONAddProp(json, "_nextId", val);
```

```
  // Encode the bounds
  JSONArrayStruct setBoundFloat = JSONArrayStructCreateStatic();
  if (GAGetLengthAdnFloat(that) > 0) {
    for (long iBound = 0; iBound < GAGetLengthAdnFloat(that); ++iBound)
      JSONArrayStructAdd(&setBoundFloat,
        VecEncodeAsJSON((VecFloat*)GABoundsAdnFloat(that, iBound)));
    JSONAddProp(json, "_boundFloat", &setBoundFloat);
  }
  JSONArrayStruct setBoundInt = JSONArrayStructCreateStatic();
  if (GAGetLengthAdnInt(that) > 0) {
    for (long iBound = 0; iBound < GAGetLengthAdnInt(that); ++iBound)
      JSONArrayStructAdd(&setBoundInt,
        VecEncodeAsJSON((VecLong*)GABoundsAdnInt(that, iBound)));
    JSONAddProp(json, "_boundInt", &setBoundInt);
  }
  // Save the adns
  JSONArrayStruct setAdn = JSONArrayStructCreateStatic();
  for (int iEnt = 0; iEnt < GAGetNbAdns(that); ++iEnt) {
    GenAlgAdn* ent = GSetElemData(GSetElement(GAAdns(that), iEnt));
    float sortVal = GSetElemGetSortVal(GSetElement(GAAdns(that), iEnt));
    JSONArrayStructAdd(&setAdn, GAAdnEncodeAsJSON(ent, sortVal));
  }
  JSONAddProp(json, "_adns", &setAdn);
  // Save the best adn
  JSONAddProp(json, "_bestAdn",
    GAAdnEncodeAsJSON(GABestAdn(that), 0.0));
  // Free memory
  JSONArrayStructFlush(&setBoundFloat);
  JSONArrayStructFlush(&setBoundInt);
  JSONArrayStructFlush(&setAdn);
  // Return the created JSON
  return json;
}

// Function which decode from JSON encoding 'json' to 'that'
bool GAAdnDecodeAsJSON(GenAlgAdn** that, const JSONNode* const json) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
  if (json == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'json' is null");
    PBErrCatch(PBMathErr);
  }
#endif
  // If 'that' is already allocated
  if (*that != NULL)
    // Free memory
    GenAlgAdnFree(that);
  // Get the id from the JSON
  JSONNode* prop = JSONProperty(json, "_id");
  if (prop == NULL) {
    return false;
  }
  unsigned long id = strtoul(JSONLabel(JSONValue(prop, 0)), NULL, 10);
  // Get the lengthAdnF from the JSON
  long lengthAdnF = 0;
  prop = JSONProperty(json, "_adnF");
  if (prop != NULL) {
```

```
    JSONNode* subprop = JSONProperty(prop, "_dim");
    lengthAdnF = atol(JSONLabel(JSONValue(subprop, 0)));
  }
  // Get the lengthAdnI from the JSON
  long lengthAdnI = 0;
  prop = JSONProperty(json, "_adnI");
  if (prop != NULL) {
    JSONNode* subprop = JSONProperty(prop, "_dim");
    lengthAdnI = atol(JSONLabel(JSONValue(subprop, 0)));
  }
  // Allocate memory
  *that = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
  // Get the age from the JSON
  prop = JSONProperty(json, "_age");
  if (prop == NULL) {
    return false;
  }
  (*that)->_age = strtoul(JSONLabel(JSONValue(prop, 0)), NULL, 10);
  // Get the adnF from the JSON
  prop = JSONProperty(json, "_adnF");
  if (prop != NULL) {
    if (!VecDecodeAsJSON(&((*that)->_adnF), prop)) {
      return false;
    }
    prop = JSONProperty(json, "_deltaAdnF");
    if (prop == NULL) {
      return false;
    }
    if (!VecDecodeAsJSON(&((*that)->_deltaAdnF), prop)) {
      return false;
    }
  }
  // Get the adnI from the JSON
  prop = JSONProperty(json, "_adnI");
  if (prop != NULL)
    if (!VecDecodeAsJSON(&((*that)->_adnI), prop)) {
      return false;
    }
  // Get the value
  prop = JSONProperty(json, "_val");
  if (prop == NULL) {
    return false;
  }
  (*that)->_val = atof(JSONLabel(JSONValue(prop, 0)));
  // Return the success code
  return true;
}

// Function which decode from JSON encoding 'json' to 'that'
bool GADecodeAsJSON(GenAlg** that, const JSONNode* const json) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
  if (json == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'json' is null");
    PBErrCatch(PBMathErr);
  }
#endif
```

```
// If 'that' is already allocated
if (*that != NULL)
  // Free memory
  GenAlgFree(that);
// Decode the nb adns
JSONNode* prop = JSONProperty(json, "_nbAdns");
if (prop == NULL) {
  return false;
}
int nbAdns = atoi(JSONLabel(JSONValue(prop, 0)));
// Decode the nb elites
prop = JSONProperty(json, "_nbElites");
if (prop == NULL) {
  return false;
}
int nbElites = atoi(JSONLabel(JSONValue(prop, 0)));
// Decode the length adn float
prop = JSONProperty(json, "_lengthAdnF");
if (prop == NULL) {
  return false;
}
long lengthAdnF = atol(JSONLabel(JSONValue(prop, 0)));
// Decode the length adn int
prop = JSONProperty(json, "_lengthAdnI");
if (prop == NULL) {
  return false;
}
long lengthAdnI = atol(JSONLabel(JSONValue(prop, 0)));
// Allocate memory
*that = GenAlgCreate(nbAdns, nbElites, lengthAdnF, lengthAdnI);
// Decode the type
prop = JSONProperty(json, "_type");
if (prop == NULL) {
  return false;
}
int type = atoi(JSONLabel(JSONValue(prop, 0)));
int nbIn = 0;
int nbOut = 0;
int nbHid = 0;
switch (type) {
  case genAlgTypeNeuraNet:
    prop = JSONProperty(json, "NN_nbIn");
    if (prop == NULL) {
      return false;
    }
    nbIn = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbOut");
    if (prop == NULL) {
      return false;
    }
    nbOut = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbHid");
    if (prop == NULL) {
      return false;
    }
    nbHid = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbBaseConv");
    if (prop == NULL) {
      return false;
    }
    GASetTypeNeuraNet(*that, nbIn, nbHid, nbOut);
    break;
```

```
  case genAlgTypeNeuraNetConv:
    prop = JSONProperty(json, "NN_nbIn");
    if (prop == NULL) {
      return false;
    }
    nbIn = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbOut");
    if (prop == NULL) {
      return false;
    }
    nbOut = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbHid");
    if (prop == NULL) {
      return false;
    }
    nbHid = atoi(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbBaseConv");
    if (prop == NULL) {
      return false;
    }
    long nbBaseConv = atol(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbBaseCellConv");
    if (prop == NULL) {
      return false;
    }
    long nbBaseCellConv = atol(JSONLabel(JSONValue(prop, 0)));
    prop = JSONProperty(json, "NN_nbLink");
    if (prop == NULL) {
      return false;
    }
    long nbLink = atol(JSONLabel(JSONValue(prop, 0)));
    GASetTypeNeuraNetConv(*that, nbIn, nbHid, nbOut, nbBaseConv,
      nbBaseCellConv, nbLink);
    break;
  default:
    break;
}
// Decode the epoch
prop = JSONProperty(json, "_curEpoch");
if (prop == NULL) {
  return false;
}
(*that)->_curEpoch =
  strtoul(JSONLabel(JSONValue(prop, 0)), NULL, 10);
// Decode the next id
prop = JSONProperty(json, "_nextId");
if (prop == NULL) {
  return false;
}
(*that)->_nextId = strtoul(JSONLabel(JSONValue(prop, 0)), NULL, 10);
// Decode the bounds
prop = JSONProperty(json, "_boundFloat");
if (prop != NULL) {
  if (JSONGetNbValue(prop) != GAGetLengthAdnFloat(*that))
    return false;
  for (long iBound = 0; iBound < GAGetLengthAdnFloat(*that); ++iBound) {
    JSONNode* val = JSONValue(prop, iBound);
    VecFloat2D* b = NULL;
    if (!VecDecodeAsJSON((VecFloat**)&b, val)) {
      return false;
    }
    GASetBoundsAdnFloat(*that, iBound, b);
```

```
      VecFree((VecFloat**)&b);
    }
  }
  prop = JSONProperty(json, "_boundInt");
  if (prop != NULL) {
    if (JSONGetNbValue(prop) != GAGetLengthAdnInt(*that))
      return false;
    for (long iBound = 0; iBound < GAGetLengthAdnInt(*that); ++iBound) {
      JSONNode* val = JSONValue(prop, iBound);
      VecLong2D* b = NULL;
      if (!VecDecodeAsJSON((VecLong**)&b, val)) {
        return false;
      }
      GASetBoundsAdnInt(*that, iBound, b);
      VecFree((VecLong**)&b);
    }
  }
  // Upadte the norm of the range values
  GAUpdateNormRange(*that);
  // Decode the adns
  prop = JSONProperty(json, "_adns");
  if (prop == NULL) {
    return false;
  }
  if (JSONGetNbValue(prop) != GAGetNbAdns(*that))
    return false;
  for (int iEnt = 0; iEnt < GAGetNbAdns(*that); ++iEnt) {
    JSONNode* val = JSONValue(prop, iEnt);
    if (!GAAdnDecodeAsJSON(
      (GenAlgAdn**)&(GSetElement(GAAdns(*that), iEnt)->_data), val)) {
      return false;
    }
  }
  // Decode the best adn
  prop = JSONProperty(json, "_bestAdn");
  if (prop == NULL) {
    return false;
  }
  if (!GAAdnDecodeAsJSON((GenAlgAdn**)&((*that)->_bestAdn), prop)) {
    return false;
  }

  // Return the success code
  return true;
}

// Load the GenAlg 'that' from the stream 'stream'
// If the GenAlg is already allocated, it is freed before loading
// Return true in case of success, else false
bool GALoad(GenAlg** that, FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (stream == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'stream' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
```

```
  // Declare a json to load the encoded data
  JSONNode* json = JSONCreate();
  // Load the whole encoded data
  if (!JSONLoad(json, stream)) {
    return false;
  }
  // Decode the data from the JSON
  if (!GADecodeAsJSON(that, json)) {
    return false;
  }
  // Free the memory used by the JSON
  JSONFree(&json);
  // Return the success code
  return true;
}

// Save the GenAlg 'that' to the stream 'stream'
// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true in case of success, else false
bool GASave(const GenAlg* const that, FILE* const stream,
  const bool compact) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (stream == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'stream' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the JSON encoding
  JSONNode* json = GAEncodeAsJSON(that);
  // Save the JSON
  if (!JSONSave(json, stream, compact)) {
    return false;
  }
  // Free memory
  JSONFree(&json);
  // Return success code
  return true;
}

// Set the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that' to 'flag'
void GASetTextOMeterFlag(GenAlg* const that, bool flag) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // If the requested flag is different from the current flag;
  if (that->_flagTextOMeter != flag) {
    if (flag && that->_textOMeter == NULL) {
      char title[] = "GenAlg";
      int width = strlen(GENALG_TXTOMETER_LINE1) + 1;
      int height = 10 +
```

```
        MIN(GENALG_TXTOMETER_NBADNDISPLAYED, GAGetNbMaxAdn(that));
      that->_textOMeter = TextOMeterCreate(title, width, height);
    }
    if (!flag && that->_textOMeter != NULL) {
      TextOMeterFree(&(that->_textOMeter));
    }
    that->_flagTextOMeter = flag;
  }
}

// Refresh the content of the TextOMeter attached to the GenAlg 'that'
void GAUpdateTextOMeter(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (that->_textOMeter == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that->_textOMeter' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Clear the TextOMeter
  TextOMeterClear(that->_textOMeter);
  // Declare a variable to print the content of the TextOMeter
  char str[50];
  // Print the content of the TextOMeter
  // Epoch #xxxxxx  KTEvent #xxxxxx
  sprintf(str, GENALG_TXTOMETER_FORMAT1,
    GAGetCurEpoch(that), GAGetNbKTEvent(that));
  TextOMeterPrint(that->_textOMeter, str);
  // Diversity +xxxxxx.xxxxxx
  sprintf(str, GENALG_TXTOMETER_FORMAT5, GAGetDiversity(that));
  TextOMeterPrint(that->_textOMeter, str);
  // Nb adns xxxxxx
  sprintf(str, GENALG_TXTOMETER_FORMAT6, GAGetNbAdns(that));
  TextOMeterPrint(that->_textOMeter, str);
  //
  sprintf(str, "\n");
  TextOMeterPrint(that->_textOMeter, str);
  // Id      Age     Val
  sprintf(str, GENALG_TXTOMETER_LINE2);
  TextOMeterPrint(that->_textOMeter, str);
  // xxxxxx  xxxxxx  +xxxxxx.xxxx
  sprintf(str, GENALG_TXTOMETER_FORMAT3,
    GAAdnGetId(GABestAdn(that)), GAAdnGetAge(GABestAdn(that)),
    GAAdnGetVal(GABestAdn(that)));
  TextOMeterPrint(that->_textOMeter, str);
  // ........................
  sprintf(str, GENALG_TXTOMETER_LINE4);
  TextOMeterPrint(that->_textOMeter, str);
  // xxxxxx  xxxxxx  +xxxxxx.xxxx
  for (int iRank = 0; iRank < GAGetNbElites(that); ++iRank) {
    sprintf(str, GENALG_TXTOMETER_FORMAT3,
      GAAdnGetId(GAAdn(that, iRank)), GAAdnGetAge(GAAdn(that, iRank)),
      GAAdnGetVal(GAAdn(that, iRank)));
    TextOMeterPrint(that->_textOMeter, str);
  }
  // ........................
  sprintf(str, GENALG_TXTOMETER_LINE4);
```

```
  TextOMeterPrint(that->_textOMeter, str);
  // xxxxxx  xxxxxx  +xxxxxx.xxxx
  int maxRank = MIN(GENALG_TXTOMETER_NBADNDISPLAYED, GAGetNbAdns(that));
  for (int iRank = GAGetNbElites(that); iRank < maxRank; ++iRank) {
    sprintf(str, GENALG_TXTOMETER_FORMAT3,
      GAAdnGetId(GAAdn(that, iRank)), GAAdnGetAge(GAAdn(that, iRank)),
      GAAdnGetVal(GAAdn(that, iRank)));
    TextOMeterPrint(that->_textOMeter, str);
  }
  // Fill in with blank lines if necessary
  sprintf(str, "\n");
  for (int iBlank = GAGetNbAdns(that);
    iBlank < GENALG_TXTOMETER_NBADNDISPLAYED; ++iBlank) {
    TextOMeterPrint(that->_textOMeter, str);
  }
  // If there are more adns than availalbe space in the TextOMeter
  if (GAGetNbAdns(that)> GENALG_TXTOMETER_NBADNDISPLAYED) {
    sprintf(str, "...");
    TextOMeterPrint(that->_textOMeter, str);
  }
  // Flush the content of the TextOMeter
  TextOMeterFlush(that->_textOMeter);
}
```

3.2 genalg-inline.c

```
// ============ GENALG-INLINE.C ================

// ------------- GenAlgAdn

// ================ Functions implementation ====================

// Return the adn for floating point values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* GAAdnAdnF(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_adnF;
}

// Return the delta of adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* GAAdnDeltaAdnF(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
```

```
#endif
  return that->_deltaAdnF;
}

// Return the adn for integer values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
VecLong* GAAdnAdnI(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_adnI;
}

// Get the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetGeneF(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return VecGet(that->_adnF, iGene);
}

// Get the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetDeltaGeneF(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return VecGet(that->_deltaAdnF, iGene);
}

// Get the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
int GAAdnGetGeneI(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
```

```
  }
#endif
  return VecGet(that->_adnI, iGene);
}

// Set the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetGeneF(GenAlgAdn* const that, const long iGene,
  const float gene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  VecSet(that->_adnF, iGene, gene);
}

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that' to 'delta'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetDeltaGeneF(GenAlgAdn* const that, const long iGene,
  const float delta) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  VecSet(that->_deltaAdnF, iGene, delta);
}

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetGeneI(GenAlgAdn* const that, const long iGene,
  const long gene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  VecSet(that->_adnI, iGene, gene);
}

// Get the id of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long int GAAdnGetId(const GenAlgAdn* const that) {
#if BUILDMODE == 0
```

```
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_id;
}

// Get the age of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long int GAAdnGetAge(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_age;
}

// Get the value of the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
float GAAdnGetVal(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_val;
}

// Return true if the GenAlgAdn 'that' is new, i.e. is age equals 1
// Return false
#if BUILDMODE != 0
inline
#endif
bool GAAdnIsNew(const GenAlgAdn* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return (that->_age == 1);
}

// Copy the GenAlgAdn 'tho' into the GenAlgAdn 'that'
#if BUILDMODE != 0
inline
#endif
void GAAdnCopy(GenAlgAdn* const that, const GenAlgAdn* const tho) {
#if BUILDMODE == 0
  if (that == NULL) {
```

```
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (tho == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'tho' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_id = tho->_id;
  that->_age = tho->_age;
  that->_val = tho->_val;
  if (tho->_adnF != NULL)
    VecCopy(that->_adnF, tho->_adnF);
  else
    VecFree(&(that->_adnF));
  if (tho->_deltaAdnF != NULL)
    VecCopy(that->_deltaAdnF, tho->_deltaAdnF);
  else
    VecFree(&(that->_deltaAdnF));
  if (tho->_adnI != NULL)
    VecCopy(that->_adnI, tho->_adnI);
  else
    VecFree(&(that->_adnI));
}

// Set the mutability vectors for the GenAlgAdn 'that' to 'mutability'
#if BUILDMODE != 0
inline
#endif
void GAAdnSetMutabilityInt(GenAlgAdn* const that,
  const VecFloat* const mutability) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (that->_mutabilityI == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that->_mutabilityI' is null");
    PBErrCatch(GenAlgErr);
  }
  if (mutability == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'mutability' is null");
    PBErrCatch(GenAlgErr);
  }
  if (VecGetDim(mutability) != VecGetDim(GAAdnAdnF(that))) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'mutability''s dim is invalid (%ld==%ld)",
      VecGetDim(mutability), VecGetDim(GAAdnAdnI(that)));
    PBErrCatch(GenAlgErr);
  }
#endif
  VecCopy(that->_mutabilityI, mutability);
}

#if BUILDMODE != 0
inline
#endif
```

```
void GAAdnSetMutabilityFloat(GenAlgAdn* const that,
  const VecFloat* const mutability) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (that->_mutabilityF == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that->_mutabilityF' is null");
    PBErrCatch(GenAlgErr);
  }
  if (mutability == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'mutability' is null");
    PBErrCatch(GenAlgErr);
  }
  if (VecGetDim(mutability) != VecGetDim(GAAdnAdnF(that))) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'mutability''s dim is invalid (%ld==%ld)",
      VecGetDim(mutability), VecGetDim(GAAdnAdnF(that)));
    PBErrCatch(GenAlgErr);
  }
#endif
  VecCopy(that->_mutabilityF, mutability);
}

// ------------- GenAlg

// ================ Functions implementation ====================

// Get the type of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
GenAlgType GAGetType(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_type;
}

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNet, the GenAlg
// will be used with a NeuraNet having 'nbIn' inputs, 'nbHid' hidden
// values and 'nbOut' outputs
#if BUILDMODE != 0
inline
#endif
void GASetTypeNeuraNet(GenAlg* const that, const int nbIn,
  const int nbHid, const int nbOut) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
```

```
  if (GAGetLengthAdnFloat(that) != GAGetLengthAdnInt(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "Must have the same nb of bases and links");
    PBErrCatch(GenAlgErr);
  }
  that->_type = genAlgTypeNeuraNet;
  that->_NNdata._nbIn = nbIn;
  that->_NNdata._nbHid = nbHid;
  that->_NNdata._nbOut = nbOut;
  that->_NNdata._nbBaseConv = 0;
}

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNetConv,
// the GenAlg will be used with a NeuraNet having 'nbIn' inputs,
// 'nbHid' hidden values, 'nbOut' outputs, 'nbBaseConv' bases function,
// 'nbLink' links dedicated to the convolution and 'nbBaseCellConv' bases function per cell of convolution
#if BUILDMODE != 0
inline
#endif
void GASetTypeNeuraNetConv(GenAlg* const that, const int nbIn,
  const int nbHid, const int nbOut, const long nbBaseConv,
  const long nbBaseCellConv, const long nbLink) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_type = genAlgTypeNeuraNetConv;
  that->_NNdata._nbIn = nbIn;
  that->_NNdata._nbHid = nbHid;
  that->_NNdata._nbOut = nbOut;
  that->_NNdata._nbBaseConv = nbBaseConv;
  that->_NNdata._nbBaseCellConv = nbBaseCellConv;
  that->_NNdata._nbLink = nbLink;
}

// Return the GSet of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
GSet* GAAdns(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_adns;
}

// Return the nb of entities of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbAdns(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
```

```
    PBErrCatch(GenAlgErr);
  }
#endif
  return GSetNbElem(that->_adns);
}

// Return the nb of elites of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbElites(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_nbElites;
}

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long int GAGetCurEpoch(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_curEpoch;
}

// Return the number of KTEvent of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
unsigned long int GAGetNbKTEvent(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_nbKTEvent;
}

// Return the min nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbMinAdn(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
```

```
#endif
  return that->_nbMinAdn;
}

// Return the max nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbMaxAdn(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_nbMaxAdn;
}

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
inline
#endif
void GASetNbMaxAdn(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_nbMaxAdn = MAX(nb, GAGetNbElites(that) + 1);
  if (GAGetNbMinAdn(that) > that->_nbMaxAdn)
    GASetNbMinAdn(that, nb);
}

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
inline
#endif
void GASetNbMinAdn(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_nbMinAdn = MAX(nb, GAGetNbElites(that) + 1);
  if (GAGetNbMaxAdn(that) < that->_nbMinAdn)
    GASetNbMaxAdn(that, nb);
}

// Get the length of adn for floating point value
#if BUILDMODE != 0
inline
#endif
long GAGetLengthAdnFloat(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
```

```
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_lengthAdnF;
}

// Get the length of adn for integer value
#if BUILDMODE != 0
inline
#endif
long GAGetLengthAdnInt(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_lengthAdnI;
}

// Set the bounds for the 'iGene'-th gene of adn for floating point
// values to a copy of 'bounds'
#if BUILDMODE != 0
inline
#endif
void GASetBoundsAdnFloat(GenAlg* const that, const long iGene,
  const VecFloat2D* const bounds) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (bounds == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'bounds' is null");
    PBErrCatch(GenAlgErr);
  }
  if (VecGet(bounds, 0) >= VecGet(bounds, 1)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'bounds' is invalid (%f<%f)",
      VecGet(bounds, 0), VecGet(bounds, 1));
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnF) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
      iGene, that->_lengthAdnF);
    PBErrCatch(GenAlgErr);
  }
#endif
  VecCopy(that->_boundsF + iGene, bounds);
  GAUpdateNormRange(that);
}

// Set the bounds for the 'iGene'-th gene of adn for integer values
// to a copy of 'bounds'
#if BUILDMODE != 0
inline
#endif
void GASetBoundsAdnInt(GenAlg* const that, const long iGene,
```

```
  const VecLong2D* const bounds) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (bounds == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'bounds' is null");
    PBErrCatch(GenAlgErr);
  }
  if (VecGet(bounds, 0) >= VecGet(bounds, 1)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'bounds' is invalid (%ld<%ld)",
      VecGet(bounds, 0), VecGet(bounds, 1));
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnI) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
      iGene, that->_lengthAdnI);
    PBErrCatch(GenAlgErr);
  }
#endif
  VecCopy(that->_boundsI + iGene, bounds);
  GAUpdateNormRange(that);
}

// Get the bounds for the 'iGene'-th gene of adn for floating point
// values
#if BUILDMODE != 0
inline
#endif
const VecFloat2D* GABoundsAdnFloat(const GenAlg* const that,
  const long iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnF) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
      iGene, that->_lengthAdnF);
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_boundsF + iGene;
}

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
inline
#endif
const VecLong2D* GABoundsAdnInt(const GenAlg* const that,
  const long iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
```

```
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnI) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
      iGene, that->_lengthAdnI);
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_boundsI + iGene;
}

// Get the GenAlgAdn of the GenAlg 'that' currently at rank 'iRank'
// (0 is the best adn)
#if BUILDMODE != 0
inline
#endif
GenAlgAdn* GAAdn(const GenAlg* const that, const int iRank) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iRank < 0 || iRank >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iRank' is invalid (0<=%d<%d)",
      iRank, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
  }
#endif
  return (GenAlgAdn*)GSetGet(that->_adns,
    GSetNbElem(that->_adns) - iRank - 1);
}

// Set the value of the GenAlgAdn 'adn' of the GenAlg 'that' to 'val'
#if BUILDMODE != 0
inline
#endif
void GASetAdnValue(GenAlg* const that, GenAlgAdn* const adn,
  const float val) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (adn == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'adn' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Set the value
  adn->_val = val;
  GSetElemSetSortVal((GSetElem*)GSetFirstElem(GAAdns(that), adn), val);
}

// Get the diversity of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
```

```
float GAGetDiversity(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  float diversity = GAGetDiversityThreshold(that) + 1.0;
  for (int iAdn = 0; iAdn < GAGetNbElites(that) - 1; ++iAdn) {
    for (int jAdn = iAdn + 1; jAdn < GAGetNbElites(that); ++jAdn) {
      diversity = fabs(MIN(diversity,
        GAAdn(that, iAdn)->_val - GAAdn(that, jAdn)->_val));
    }
  }
  return diversity;
}

// Get the diversity threshold of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
float GAGetDiversityThreshold(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_diversityThreshold;
}

// Set the diversity threshold of the GenAlg 'that' to 'threshold'
#if BUILDMODE != 0
inline
#endif
void GASetDiversityThreshold(GenAlg* const that, const float threshold) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_diversityThreshold = threshold;
}

// Return the best adn of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
const GenAlgAdn* GABestAdn(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_bestAdn;
}
```

```
// Return the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
bool GAIsTextOMeterActive(const GenAlg* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_flagTextOMeter;
}
```

4 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=genalg
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
```

```
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "genalg.h"

#define RANDOMSEED 2

void UnitTestGenAlgAdnCreateFree() {
  unsigned long int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
  if (ent->_age != 1 ||
    ent->_id != id ||
    VecGetDim(ent->_adnF) != lengthAdnF ||
    VecGetDim(ent->_deltaAdnF) != lengthAdnF ||
    VecGetDim(ent->_adnI) != lengthAdnI) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgAdnCreate failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgAdnFree(&ent);
  if (ent != NULL) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgAdnFree failed");
    PBErrCatch(GenAlgErr);
  }
  printf("UnitTestGenAlgAdnCreateFree OK\n");
}

void UnitTestGenAlgAdnGetSet() {
  unsigned long int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
  if (GAAdnAdnF(ent) != ent->_adnF) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnAdnF failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnDeltaAdnF(ent) != ent->_deltaAdnF) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnDeltaAdnF failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnAdnI(ent) != ent->_adnI) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnAdnI failed");
    PBErrCatch(GenAlgErr);
  }
  GAAdnSetGeneF(ent, 0, 1.0);
  if (ISEQUALF(VecGet(ent->_adnF, 0), 1.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnSetGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  if (ISEQUALF(GAAdnGetGeneF(ent, 0), 1.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  GAAdnSetDeltaGeneF(ent, 0, 2.0);
```

```
  if (ISEQUALF(VecGet(ent->_deltaAdnF, 0), 2.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnSetDeltaGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  if (ISEQUALF(GAAdnGetDeltaGeneF(ent, 0), 2.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetDeltaGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  GAAdnSetGeneI(ent, 0, 3);
  if (VecGet(ent->_adnI, 0) != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnSetGeneI failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnGetGeneI(ent, 0) != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetGeneI failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnGetAge(ent) != 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetAge failed");
    PBErrCatch(GenAlgErr);
  }
  ent->_val = 2.0;
  if (ISEQUALF(GAAdnGetVal(ent), 2.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetVal failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnGetId(ent) != id) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnGetId failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAAdnIsNew(ent) != true) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnIsNew failed");
    PBErrCatch(GenAlgErr);
  }
  ent->_age = 2;
  if (GAAdnIsNew(ent) != false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnIsNew failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgAdnFree(&ent);
  printf("UnitTestGenAlgAdnGetSet OK\n");
}

void UnitTestGenAlgAdnInit() {
  srandom(5);
  unsigned long int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
```

```
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GAAdnInit(ent, ga);
  if (ISEQUALF(VecGet(ent->_adnF, 0), -0.907064) == false ||
    ISEQUALF(VecGet(ent->_adnF, 1), -0.450509) == false ||
    VecGet(ent->_adnI, 0) != 2 ||
    VecGet(ent->_adnI, 1) != 10) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAAdnInit failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  GenAlgAdnFree(&ent);
  printf("UnitTestGenAlgAdnInit OK\n");
}

void UnitTestGenAlgAdn() {
  UnitTestGenAlgAdnCreateFree();
  UnitTestGenAlgAdnGetSet();
  UnitTestGenAlgAdnInit();
  printf("UnitTestGenAlgAdn OK\n");
}

void UnitTestGenAlgCreateFree() {
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  if (ga->_type != genAlgTypeDefault ||
    ga->_curEpoch != 0 ||
    ga->_nbKTEvent != 0 ||
    ga->_nextId != GENALG_NBENTITIES ||
    ga->_nbElites != GENALG_NBELITES ||
    ga->_lengthAdnF != lengthAdnF ||
    ga->_lengthAdnI != lengthAdnI ||
    ga->_flagTextOMeter != false ||
    ga->_nbMinAdn != GENALG_NBENTITIES ||
    ga->_nbMaxAdn != GENALG_NBENTITIES ||
    ISEQUALF(ga->_diversityThreshold, 0.01) != true ||
    ga->_textOMeter != NULL ||
    GSetNbElem(GAAdns(ga)) != GENALG_NBENTITIES) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgCreate failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  if (ga != NULL) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgFree failed");
    PBErrCatch(GenAlgErr);
  }
  printf("UnitTestGenAlgCreateFree OK\n");
}

void UnitTestGenAlgGetSet() {
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
```

```
  lengthAdnF, lengthAdnI);
if (GAGetType(ga) != ga->_type) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetType failed");
  PBErrCatch(GenAlgErr);
}
if (GAAdns(ga) != ga->_adns) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAEloRank failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetNbAdns(ga) != GENALG_NBENTITIES) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetNbAdns failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetNbElites(ga) != GENALG_NBELITES) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetNbElites failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetCurEpoch(ga) != 0) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetCurEpoch failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetNbKTEvent(ga) != 0) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetNbKTEvent failed");
  PBErrCatch(GenAlgErr);
}
if (ISEQUALF(GAGetDiversityThreshold(ga),
  ga->_diversityThreshold) != true) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetDiversityThreshold failed");
  PBErrCatch(GenAlgErr);
}
GASetDiversityThreshold(ga, 2.0);
if (ISEQUALF(GAGetDiversityThreshold(ga), 2.0) != true) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetDiversityThrehsold failed");
  PBErrCatch(GenAlgErr);
}
GASetNbEntities(ga, 10);
if (GAGetNbAdns(ga) != 10 ||
  GAGetNbElites(ga) != 9 ||
  GSetNbElem(GAAdns(ga)) != 10) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetNbEntities failed");
  PBErrCatch(GenAlgErr);
}
GASetNbElites(ga, 20);
if (GAGetNbAdns(ga) != 21 ||
  GAGetNbElites(ga) != 20 ||
  GSetNbElem(GAAdns(ga)) != 21) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetNbElites failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetLengthAdnFloat(ga) != lengthAdnF) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetLengthAdnFloat failed");
```

```
  PBErrCatch(GenAlgErr);
}
if (GAGetLengthAdnInt(ga) != lengthAdnI) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetLengthAdnInt failed");
  PBErrCatch(GenAlgErr);
}
if (GABoundsAdnFloat(ga, 1) != ga->_boundsF + 1) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GABoundsAdnFloat failed");
  PBErrCatch(GenAlgErr);
}
VecFloat2D boundsF = VecFloatCreateStatic2D();
VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
GASetBoundsAdnFloat(ga, 1, &boundsF);
if (VecIsEqual(GABoundsAdnFloat(ga, 1), &boundsF) == false) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetBoundsAdnFloat failed");
  PBErrCatch(GenAlgErr);
}
VecLong2D boundsS = VecLongCreateStatic2D();
VecSet(&boundsS, 0, -1); VecSet(&boundsS, 1, 1);
GASetBoundsAdnInt(ga, 1, &boundsS);
if (VecIsEqual(GABoundsAdnInt(ga, 1), &boundsS) == false) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetBoundsAdnInt failed");
  PBErrCatch(GenAlgErr);
}
if (GABoundsAdnInt(ga, 1) != ga->_boundsI + 1) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GABoundsAdnInt failed");
  PBErrCatch(GenAlgErr);
}
GASetAdnValue(ga, GAAdn(ga, 0), 1.0);
if (ISEQUALF(GAAdn(ga, 0)->_val, 1.0) == false ||
  ISEQUALF(ga->_adns->_tail->_sortVal, 1.0) == false) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetAdnValue failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetNbMaxAdn(ga) != ga->_nbMaxAdn) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetNbMaxAdn failed");
  PBErrCatch(GenAlgErr);
}
if (GAGetNbMinAdn(ga) != ga->_nbMinAdn) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAGetNbMinAdn failed");
  PBErrCatch(GenAlgErr);
}
GASetNbMaxAdn(ga, 100);
if (GAGetNbMaxAdn(ga) != 100) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetNbMaxAdn failed");
  PBErrCatch(GenAlgErr);
}
GASetNbMinAdn(ga, 100);
if (GAGetNbMinAdn(ga) != 100) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GASetNbMinAdn failed");
  PBErrCatch(GenAlgErr);
}
```

```
  GenAlgFree(&ga);
  ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES, 3, 3);
  GASetTypeNeuraNet(ga, 1, 2, 3);
  if (GAGetType(ga) != genAlgTypeNeuraNet ||
    ga->_NNdata._nbIn != 1 ||
    ga->_NNdata._nbHid != 2 ||
    ga->_NNdata._nbOut != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetTypeNeuraNet failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  printf("UnitTestGenAlgGetSet OK\n");
}

void UnitTestGenAlgInit() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GAInit(ga);
  GenAlgAdn* ent = (GenAlgAdn*)(GAAdns(ga)->_head->_data);
  if (ISEQUALF(VecGet(ent->_adnF, 0), -0.907064) == false ||
    ISEQUALF(VecGet(ent->_adnF, 1), -0.450509) == false ||
    VecGet(ent->_adnI, 0) != 2 ||
    VecGet(ent->_adnI, 1) != 10) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAInit failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  printf("UnitTestGenAlgInit OK\n");
}

void UnitTestGenAlgPrint() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GAInit(ga);
  GAPrintln(ga, stdout);
  GAEliteSummaryPrintln(ga, stdout);
  GenAlgFree(&ga);
  printf("UnitTestGenAlgInit OK\n");
}
```

```
void UnitTestGenAlgGetDiversity() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GASetNbElites(ga, 2);
  GASetNbEntities(ga, 3);
  GAInit(ga);
  if (ISEQUALF(GAGetDiversity(ga), 0.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetDiversity failed");
    PBErrCatch(GenAlgErr);
  }
  VecCopy(GAAdn(ga, 1)->_adnF, GAAdn(ga, 0)->_adnF);
  VecCopy(GAAdn(ga, 1)->_adnI, GAAdn(ga, 0)->_adnI);
  if (ISEQUALF(GAGetDiversity(ga), 0.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetDiversity failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  printf("UnitTestGenAlgGetDiversity OK\n");
}

void UnitTestGenAlgStep() {
  srandom(2);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GAInit(ga);
  for (int i = 3; i--;)
    GASetAdnValue(ga, GAAdn(ga, i), 3.0 - (float)i);
  printf("Before Step:\n");
  GAPrintln(ga, stdout);
  GenAlgAdn* child = GAAdn(ga, 2);
  GAStep(ga);
  printf("After Step:\n");
  GAPrintln(ga, stdout);
  if (ga->_nextId != 4 || GAAdnGetId(child) != 3 ||
    GAAdnGetAge(child) != 1 ||
    ISEQUALF(GAAdnGetGeneF(child, 0), -0.156076) == false ||
    ISEQUALF(GAAdnGetGeneF(child, 1), 0.174965) == false ||
    ISEQUALF(GAAdnGetDeltaGeneF(child, 0), 0.0) == false ||
    ISEQUALF(GAAdnGetDeltaGeneF(child, 1), 0.0) == false ||
```

```
    GAAdnGetGeneI(child, 0) != 4 ||
    GAAdnGetGeneI(child, 1) != 7 ||
    GAAdn(ga, 2) != child ||
    GAAdnGetAge(GAAdn(ga, 0)) != 2 ||
    GAAdnGetAge(GAAdn(ga, 1)) != 2 ||
    GAAdnGetId(GAAdn(ga, 0)) != 0 ||
    GAAdnGetId(GAAdn(ga, 1)) != 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAStep failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  printf("UnitTestGenAlgStep OK\n");
}

void UnitTestGenAlgLoadSave() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  GASetBoundsAdnFloat(ga, 0, &boundsF);
  GASetBoundsAdnFloat(ga, 1, &boundsF);
  GASetBoundsAdnInt(ga, 0, &boundsI);
  GASetBoundsAdnInt(ga, 1, &boundsI);
  GAInit(ga);
  GAStep(ga);
  GSet* rank = GSetCreate();
  for (int i = 3; i--;)
    GSetAddSort(rank, GAAdn(ga, i), 3.0 - (float)i);
  FILE* stream = fopen("./UnitTestGenAlgLoadSave.txt", "w");
  if (GASave(ga, stream, false) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASave failed");
    PBErrCatch(GenAlgErr);
  }
  fclose(stream);
  stream = fopen("./UnitTestGenAlgLoadSave.txt", "r");
  GenAlg* gaLoad = NULL;
  if (GALoad(&gaLoad, stream) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GALoad failed");
    PBErrCatch(GenAlgErr);
  }
  fclose(stream);
  if (ga->_nextId != gaLoad->_nextId ||
    ga->_curEpoch != gaLoad->_curEpoch ||
    ga->_nbElites != gaLoad->_nbElites ||
    ga->_type != genAlgTypeDefault ||
    ga->_lengthAdnF != gaLoad->_lengthAdnF ||
    ga->_lengthAdnI != gaLoad->_lengthAdnI ||
    VecIsEqual(ga->_boundsF, gaLoad->_boundsF) == false ||
    VecIsEqual(ga->_boundsF + 1, gaLoad->_boundsF + 1) == false ||
    VecIsEqual(ga->_boundsI, gaLoad->_boundsI) == false ||
    VecIsEqual(ga->_boundsI + 1, gaLoad->_boundsI + 1) == false ||
    GAAdnGetId(GAAdn(ga, 0)) != GAAdnGetId(GAAdn(gaLoad, 0)) ||
    GAAdnGetId(GAAdn(ga, 1)) != GAAdnGetId(GAAdn(gaLoad, 1)) ||
    GAAdnGetId(GAAdn(ga, 2)) != GAAdnGetId(GAAdn(gaLoad, 2)) ||
    GAAdnGetAge(GAAdn(ga, 0)) != GAAdnGetAge(GAAdn(gaLoad, 0)) ||
```

```
    GAAdnGetAge(GAAdn(ga, 1)) != GAAdnGetAge(GAAdn(gaLoad, 1)) ||
    GAAdnGetAge(GAAdn(ga, 2)) != GAAdnGetAge(GAAdn(gaLoad, 2)) ||
    VecIsEqual(GAAdn(ga, 0)->_adnF,
      GAAdn(gaLoad, 0)->_adnF) == false ||
    VecIsEqual(GAAdn(ga, 0)->_deltaAdnF,
      GAAdn(gaLoad, 0)->_deltaAdnF) == false ||
    VecIsEqual(GAAdn(ga, 0)->_adnI,
      GAAdn(gaLoad, 0)->_adnI) == false ||
    VecIsEqual(GAAdn(ga, 1)->_adnF,
      GAAdn(gaLoad, 1)->_adnF) == false ||
    VecIsEqual(GAAdn(ga, 1)->_deltaAdnF,
      GAAdn(gaLoad, 1)->_deltaAdnF) == false ||
    VecIsEqual(GAAdn(ga, 1)->_adnI,
      GAAdn(gaLoad, 1)->_adnI) == false ||
    VecIsEqual(GAAdn(ga, 2)->_adnF,
      GAAdn(gaLoad, 2)->_adnF) == false ||
    VecIsEqual(GAAdn(ga, 2)->_deltaAdnF,
      GAAdn(gaLoad, 2)->_deltaAdnF) == false ||
    VecIsEqual(GAAdn(ga, 2)->_adnI,
      GAAdn(gaLoad, 2)->_adnI) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "UnitTestGenAlgLoadSave failed");
    PBErrCatch(GenAlgErr);
  }
  GSetFree(&rank);
  GenAlgFree(&ga);
  GenAlgFree(&gaLoad);
  printf("UnitTestGenAlgLoadSave OK\n");
}

float ftarget(float x) {
  return -0.5 * fastpow(x, 3) + 0.314 * fastpow(x, 2) - 0.7777 * x + 0.1;
}

float evaluate(const VecFloat* adnF, const VecLong* adnI) {
  float delta = 0.02;
  int nb = (int)round(4.0 / delta);
  float res = 0.0;
  float x = -2.0;
  for (int i = 0; i < nb; ++i, x += delta) {
    float y = 0.0;
    for (int j = 4; j--;)
      y += VecGet(adnF, j) * fastpow(x, VecGet(adnI, j));
    res += fabs(ftarget(x) - y);
  }
  return res / (float)nb;
}

void UnitTestGenAlgTest() {
  srandom(0);
  int lengthAdnF = 4;
  int lengthAdnI = lengthAdnF;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecLong2D boundsI = VecLongCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 0); VecSet(&boundsI, 1, 4);
  for (int i = lengthAdnF; i--;) {
    GASetBoundsAdnFloat(ga, i, &boundsF);
    GASetBoundsAdnInt(ga, i, &boundsI);
  }
```

```
  GAInit(ga);
  GASetTextOMeterFlag(ga, true);
  GASetDiversityThreshold(ga, 0.0001);
  //GASetDiversityThreshold(ga, 0.0);
float best = 1.0;
//int step = 0;
  do {
//float ev = evaluate(GABestAdnF(ga), GABestAdnI(ga));
//printf("%lu %f %f\n",GAGetCurEpoch(ga), ev, GAGetDiversity(ga));
    for (int iEnt = GAGetNbAdns(ga); iEnt--;)
      if (GAAdnIsNew(GAAdn(ga, iEnt)))
        GASetAdnValue(ga, GAAdn(ga, iEnt),
          -1.0 * evaluate(GAAdnAdnF(GAAdn(ga, iEnt)),
          GAAdnAdnI(GAAdn(ga, iEnt))));
    GAStep(ga);
    // Slow down the process to have time to read the TextOMeter
    unsigned int microseconds = 10000;
    usleep(microseconds);
    //sleep(1);
    // Display info if there is improvment
    float ev = evaluate(GABestAdnF(ga), GABestAdnI(ga));
    if (best - ev > PBMATH_EPSILON) {
      best = ev;
      printf("%lu %f ", GAGetCurEpoch(ga), best);
      VecFloatPrint(GABestAdnF(ga), stdout, 6);
      printf(" ");
      VecPrint(GABestAdnI(ga), stdout);
      printf("\n");
    }
  } while (GAGetCurEpoch(ga) < 20000 ||
    evaluate(GABestAdnF(ga), GABestAdnI(ga)) < PBMATH_EPSILON);
  printf("target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1\n");
  printf("approx: \n");
  GAAdnPrintln(GABestAdn(ga), stdout);
  printf("error: %f\n", evaluate(GABestAdnF(ga), GABestAdnI(ga)));
  GenAlgFree(&ga);
  printf("UnitTestGenAlgTest OK\n");
}

void UnitTestGenAlgPerf() {
  int nbRun = 500;
  unsigned long int nbMaxEpoch = 2000;
  float maxEv = 0.0;
  float bestEv = 0.0;
  float sumEv = 0.0;
  float avgEv = 0.0;
  for (int iRun = 0; iRun < nbRun; ++iRun) {
    srandom(time(NULL));
    int lengthAdnF = 4;
    int lengthAdnI = lengthAdnF;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
      lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 0); VecSet(&boundsI, 1, 4);
    for (int i = lengthAdnF; i--;) {
      GASetBoundsAdnFloat(ga, i, &boundsF);
      GASetBoundsAdnInt(ga, i, &boundsI);
    }
    GAInit(ga);
    GASetDiversityThreshold(ga, 0.001);
```

```
    float ev = 0.0;
    do {
      for (int iEnt = GAGetNbAdns(ga); iEnt--;)
        if (GAAdnIsNew(GAAdn(ga, iEnt)))
          GASetAdnValue(ga, GAAdn(ga, iEnt),
            -1.0 * evaluate(GAAdnAdnF(GAAdn(ga, iEnt)),
            GAAdnAdnI(GAAdn(ga, iEnt))));
      GAStep(ga);
      ev = evaluate(GABestAdnF(ga), GABestAdnI(ga));
    } while (GAGetCurEpoch(ga) < nbMaxEpoch || ev < PBMATH_EPSILON);
    sumEv += ev;
    if (iRun == 0 || bestEv > ev)
      bestEv = ev;
    if (iRun == 0 || maxEv < ev)
      maxEv = ev;
    avgEv = sumEv / (float)iRun;
    printf("best: %f, worst: %f, avg: %f, ktevent: %lu\n",
      bestEv, maxEv, avgEv, ga->_nbKTEvent);
    GenAlgFree(&ga);
  }
  avgEv = sumEv / (float)nbRun;
  printf("in %d runs, %lu epochs, best: %f, worst: %f, avg: %f\n",
    nbRun, nbMaxEpoch, bestEv, maxEv, avgEv);
  printf("UnitTestGenAlgPerf OK\n");
}

void UnitTestGenAlg() {
  UnitTestGenAlgCreateFree();
  UnitTestGenAlgGetSet();
  UnitTestGenAlgInit();
  UnitTestGenAlgPrint();
  UnitTestGenAlgGetDiversity();
  UnitTestGenAlgStep();
  UnitTestGenAlgLoadSave();
  UnitTestGenAlgTest();
  UnitTestGenAlgPerf();
  printf("UnitTestGenAlg OK\n");
}

void UnitTestAll() {
  UnitTestGenAlgAdn();
  UnitTestGenAlg();
  printf("UnitTestAll OK\n");
}

int main() {
  UnitTestAll();
  // Return success code
  return 0;
}
```

6 Unit tests output

```
UnitTestGenAlgAdnCreateFree OK
UnitTestGenAlgAdnGetSet OK
UnitTestGenAlgAdnInit OK
UnitTestGenAlgAdn OK
UnitTestGenAlgCreateFree OK
```

```
UnitTestGenAlgGetSet OK
UnitTestGenAlgInit OK
epoch:0
3 entities, 2 elites
#0 value:0.000000 elite id:0 age:1
  adnF:<0.788004,-0.003504>
  deltaAdnF:<0.000000,0.000000>
  adnI:<3,1>
#1 value:0.000000 elite id:1 age:1
  adnF:<-0.840711,-0.704622>
  deltaAdnF:<0.000000,0.000000>
  adnI:<5,4>
#2 value:0.000000 id:2 age:1
  adnF:<-0.907064,-0.450509>
  deltaAdnF:<0.000000,0.000000>
  adnI:<2,10>
(age,val,div) (1,0.000,0.000) (1,0.000,0.455)
UnitTestGenAlgInit OK
UnitTestGenAlgGetDiversity OK
Before Step:
epoch:0
3 entities, 2 elites
#0 value:3.000000 elite id:0 age:1
  adnF:<0.285933,0.174965>
  deltaAdnF:<0.000000,0.000000>
  adnI:<4,10>
#1 value:2.000000 elite id:1 age:1
  adnF:<-0.156076,-0.303386>
  deltaAdnF:<0.000000,0.000000>
  adnI:<2,7>
#2 value:1.000000 id:2 age:1
  adnF:<0.619353,0.401953>
  deltaAdnF:<0.000000,0.000000>
  adnI:<2,2>
After Step:
epoch:1
3 entities, 2 elites
#0 value:3.000000 elite id:0 age:2
  adnF:<0.285933,0.174965>
  deltaAdnF:<0.000000,0.000000>
  adnI:<4,10>
#1 value:2.000000 elite id:1 age:2
  adnF:<-0.156076,-0.303386>
  deltaAdnF:<0.000000,0.000000>
  adnI:<2,7>
#2 value:1.000000 id:3 age:1
  adnF:<-0.156076,0.174965>
  deltaAdnF:<0.000000,0.000000>
  adnI:<4,7>
UnitTestGenAlgStep OK
UnitTestGenAlgLoadSave OK
2 0.203473 <-0.535477,-0.411679,-0.262673,0.499542> <3,1,1,2>
3 0.195333 <-0.509326,0.560061,-0.216620,-0.673737> <1,2,2,3>
4 0.143161 <-0.993537,-0.411679,0.495386,-0.211346> <1,3,2,2>
6 0.121755 <-0.595302,-0.596549,0.130730,0.279675> <3,1,2,2>
10 0.112001 <-0.553979,-0.130973,0.495386,-0.577177> <3,2,2,1>
21 0.109586 <-0.595302,-0.130973,0.477931,-0.577177> <3,2,2,1>
24 0.101654 <-0.554550,-0.596549,0.130730,0.239866> <3,1,2,2>
25 0.092341 <-0.535477,-0.640766,0.130730,0.239866> <3,1,2,2>
28 0.084221 <-0.535477,-0.640766,0.477931,-0.133433> <3,1,2,2>
29 0.073412 <-0.535477,-0.554858,0.382368,-0.159050> <3,1,2,1>
32 0.073340 <-0.540608,-0.649452,0.477931,-0.133433> <3,1,2,2>
```

```
33 0.066256 <-0.509709,-0.777019,0.130730,0.239866> <3,1,2,2>
38 0.065540 <-0.509709,-0.777019,0.471051,-0.102282> <3,1,2,2>
42 0.063982 <-0.509709,-0.733643,0.477931,-0.133433> <3,1,2,2>
50 0.062623 <-0.509709,-0.733643,0.477931,-0.127537> <3,1,2,2>
55 0.060356 <-0.509709,-0.740938,0.477931,-0.127537> <3,1,2,2>
59 0.059140 <-0.509709,-0.757635,0.471051,-0.115363> <3,1,2,2>
65 0.058751 <-0.500331,-0.777019,0.469435,-0.115363> <3,1,2,2>
498 0.058717 <-0.496327,-0.785919,0.463821,-0.108970> <3,1,2,2>
597 0.058647 <-0.492355,-0.795860,0.473360,-0.120206> <3,1,2,2>
756 0.058616 <-0.487897,-0.809940,0.461756,-0.108970> <3,1,2,2>
810 0.058601 <-0.487713,-0.809940,0.461756,-0.108970> <3,1,2,2>
960 0.058522 <-0.485299,-0.814151,0.461756,-0.107747> <3,1,2,2>
1196 0.058496 <-0.481871,-0.824379,0.461756,-0.108754> <3,1,2,2>
1211 0.058474 <-0.481871,-0.822442,0.461756,-0.107747> <3,1,2,2>
12220 0.058461 <-0.480317,-0.825928,0.633169,-0.278616> <3,1,2,2>
16382 0.058448 <-0.480381,-0.826693,0.627395,-0.273327> <3,1,2,2>
17222 0.057908 <-0.488069,-0.805422,0.358266,0.002294> <3,1,2,0>
17223 0.057787 <-0.490046,-0.810895,0.358266,0.005386> <3,1,2,0>
17225 0.047051 <-0.488069,-0.794229,0.358266,0.031503> <3,1,2,0>
17231 0.045808 <-0.490046,-0.790413,0.358266,0.037339> <3,1,2,0>
17232 0.043866 <-0.490046,-0.790413,0.355622,0.037339> <3,1,2,0>
17234 0.031842 <-0.490046,-0.810895,0.338240,0.091660> <3,1,2,0>
17237 0.021098 <-0.490046,-0.797411,0.332244,0.066114> <3,1,2,0>
17241 0.016551 <-0.491319,-0.810895,0.315186,0.091660> <3,1,2,0>
17244 0.016187 <-0.490046,-0.790413,0.328999,0.091660> <3,1,2,0>
17246 0.015080 <-0.490046,-0.810895,0.315186,0.091660> <3,1,2,0>
17248 0.012024 <-0.491319,-0.790413,0.315186,0.091660> <3,1,2,0>
17251 0.011718 <-0.490046,-0.805422,0.315186,0.095194> <3,1,2,0>
17252 0.010192 <-0.491319,-0.797411,0.315186,0.091660> <3,1,2,0>
17260 0.010082 <-0.490046,-0.797411,0.315186,0.096854> <3,1,2,0>
17261 0.008961 <-0.491319,-0.797411,0.315186,0.096854> <3,1,2,0>
17266 0.008846 <-0.491319,-0.794414,0.315186,0.096854> <3,1,2,0>
17309 0.008715 <-0.491319,-0.794414,0.315186,0.098981> <3,1,2,0>
17468 0.008641 <-0.500694,-0.767911,0.315186,0.096854> <3,1,2,0>
17479 0.008181 <-0.500694,-0.784437,0.315186,0.098981> <3,1,2,0>
17494 0.006908 <-0.500694,-0.770133,0.315186,0.095194> <3,1,2,0>
17504 0.006194 <-0.500694,-0.770133,0.315186,0.098981> <3,1,2,0>
17506 0.006164 <-0.500694,-0.770133,0.316654,0.098981> <3,1,2,0>
17515 0.003027 <-0.500902,-0.778147,0.316654,0.098981> <3,1,2,0>
17517 0.002932 <-0.500694,-0.778147,0.316654,0.096854> <3,1,2,0>
17531 0.001565 <-0.499321,-0.778147,0.315540,0.098981> <3,1,2,0>
17568 0.001539 <-0.499321,-0.777831,0.314871,0.098981> <3,1,2,0>
17569 0.001510 <-0.499321,-0.777831,0.313672,0.099487> <3,1,2,0>
17573 0.001138 <-0.500902,-0.775650,0.313672,0.099487> <3,1,2,0>
18116 0.000959 <-0.500902,-0.775650,0.313672,0.100159> <3,1,2,0>
19015 0.000878 <-0.499343,-0.779654,0.314188,0.100159> <3,1,2,0>
19418 0.000842 <-0.500555,-0.776924,0.313698,0.099666> <3,1,2,0>
19424 0.000632 <-0.500555,-0.776924,0.313698,0.100159> <3,1,2,0>
target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1
approx:
id:1742927 age:19424
  adnF:<-0.500555,-0.776924,0.313698,0.100159>
  deltaAdnF:<0.000347,0.003782,0.000026,0.003305>
  adnI:<3,1,2,0>
error: 0.000632
UnitTestGenAlgTest OK
best: 0.000708, worst: 0.000708, avg: inf, ktevent: 23968
best: 0.000708, worst: 0.002076, avg: 0.002785, ktevent: 23908
best: 0.000708, worst: 0.002076, avg: 0.001956, ktevent: 23576
best: 0.000708, worst: 0.002164, avg: 0.002025, ktevent: 24669
best: 0.000708, worst: 0.002164, avg: 0.001831, ktevent: 24071
best: 0.000708, worst: 0.002164, avg: 0.001688, ktevent: 24199
```

```
best: 0.000708, worst: 0.058459, avg: 0.011150, ktevent: 24720
best: 0.000708, worst: 0.058459, avg: 0.009810, ktevent: 24464
best: 0.000708, worst: 0.058473, avg: 0.015893, ktevent: 23820
best: 0.000708, worst: 0.058473, avg: 0.014428, ktevent: 24243
best: 0.000708, worst: 0.058473, avg: 0.013071, ktevent: 23604
best: 0.000708, worst: 0.058495, avg: 0.017201, ktevent: 23759
best: 0.000708, worst: 0.058495, avg: 0.015903, ktevent: 23565
best: 0.000708, worst: 0.058495, avg: 0.014863, ktevent: 24293
best: 0.000708, worst: 0.058506, avg: 0.017980, ktevent: 28463
best: 0.000708, worst: 0.058506, avg: 0.016990, ktevent: 24967
best: 0.000708, worst: 0.058506, avg: 0.016023, ktevent: 24290
best: 0.000708, worst: 0.058506, avg: 0.015217, ktevent: 24080
best: 0.000708, worst: 0.058506, avg: 0.014436, ktevent: 23902
best: 0.000708, worst: 0.058506, avg: 0.013744, ktevent: 23948
best: 0.000708, worst: 0.058521, avg: 0.015983, ktevent: 23411
best: 0.000708, worst: 0.058521, avg: 0.015277, ktevent: 23535
best: 0.000708, worst: 0.058521, avg: 0.014642, ktevent: 24120
best: 0.000367, worst: 0.058521, avg: 0.014021, ktevent: 23852
best: 0.000367, worst: 0.058521, avg: 0.013496, ktevent: 24077
best: 0.000367, worst: 0.058521, avg: 0.013033, ktevent: 24332
best: 0.000367, worst: 0.058521, avg: 0.012573, ktevent: 24000
best: 0.000367, worst: 0.058521, avg: 0.012143, ktevent: 23395
best: 0.000367, worst: 0.058521, avg: 0.013204, ktevent: 24467
best: 0.000367, worst: 0.058521, avg: 0.014765, ktevent: 26547
best: 0.000367, worst: 0.058521, avg: 0.014303, ktevent: 24032
best: 0.000367, worst: 0.058521, avg: 0.013866, ktevent: 23756
best: 0.000367, worst: 0.058521, avg: 0.013478, ktevent: 24203
best: 0.000367, worst: 0.058521, avg: 0.013101, ktevent: 23687
best: 0.000367, worst: 0.058521, avg: 0.012748, ktevent: 24001
best: 0.000367, worst: 0.058521, avg: 0.012401, ktevent: 24546
best: 0.000367, worst: 0.058521, avg: 0.012083, ktevent: 23988
best: 0.000367, worst: 0.058521, avg: 0.011776, ktevent: 24031
best: 0.000367, worst: 0.058521, avg: 0.011510, ktevent: 23540
best: 0.000367, worst: 0.058521, avg: 0.011266, ktevent: 23875
best: 0.000367, worst: 0.058521, avg: 0.012035, ktevent: 24956
best: 0.000367, worst: 0.058521, avg: 0.012763, ktevent: 25802
best: 0.000367, worst: 0.058521, avg: 0.012471, ktevent: 24197
best: 0.000367, worst: 0.058521, avg: 0.012224, ktevent: 24088
best: 0.000367, worst: 0.058521, avg: 0.011973, ktevent: 23597
best: 0.000367, worst: 0.058521, avg: 0.011753, ktevent: 23835
best: 0.000367, worst: 0.058521, avg: 0.011521, ktevent: 23694
best: 0.000367, worst: 0.058521, avg: 0.012520, ktevent: 25254
best: 0.000367, worst: 0.058521, avg: 0.012301, ktevent: 23792
best: 0.000367, worst: 0.058521, avg: 0.012062, ktevent: 24325
best: 0.000367, worst: 0.058525, avg: 0.012991, ktevent: 24737
best: 0.000367, worst: 0.058525, avg: 0.012749, ktevent: 24142
best: 0.000367, worst: 0.058525, avg: 0.012521, ktevent: 24086
best: 0.000367, worst: 0.058525, avg: 0.012313, ktevent: 23878
best: 0.000367, worst: 0.058525, avg: 0.013168, ktevent: 27452
best: 0.000367, worst: 0.058525, avg: 0.012951, ktevent: 24602
best: 0.000367, worst: 0.058525, avg: 0.012743, ktevent: 23837
best: 0.000367, worst: 0.058525, avg: 0.012530, ktevent: 24250
best: 0.000367, worst: 0.058525, avg: 0.012329, ktevent: 23408
best: 0.000367, worst: 0.058525, avg: 0.012164, ktevent: 24200
best: 0.000367, worst: 0.058525, avg: 0.011970, ktevent: 23725
best: 0.000367, worst: 0.058525, avg: 0.011782, ktevent: 24324
best: 0.000367, worst: 0.058525, avg: 0.012536, ktevent: 24525
best: 0.000367, worst: 0.058525, avg: 0.012356, ktevent: 24269
best: 0.000367, worst: 0.058525, avg: 0.012178, ktevent: 23892
best: 0.000367, worst: 0.058525, avg: 0.012013, ktevent: 23920
best: 0.000367, worst: 0.058525, avg: 0.011842, ktevent: 23857
best: 0.000367, worst: 0.058525, avg: 0.011694, ktevent: 23828
```

```
best: 0.000367, worst: 0.058525, avg: 0.011533, ktevent: 24546
best: 0.000367, worst: 0.058525, avg: 0.011379, ktevent: 24217
best: 0.000367, worst: 0.058525, avg: 0.011226, ktevent: 23988
best: 0.000367, worst: 0.058525, avg: 0.011078, ktevent: 24006
best: 0.000367, worst: 0.058525, avg: 0.010939, ktevent: 23937
best: 0.000367, worst: 0.058525, avg: 0.010811, ktevent: 24078
best: 0.000367, worst: 0.058525, avg: 0.010673, ktevent: 23732
best: 0.000318, worst: 0.058525, avg: 0.010534, ktevent: 23520
best: 0.000318, worst: 0.058525, avg: 0.010413, ktevent: 23923
best: 0.000081, worst: 0.058525, avg: 0.010279, ktevent: 23767
best: 0.000081, worst: 0.058525, avg: 0.010155, ktevent: 24293
best: 0.000081, worst: 0.058525, avg: 0.010767, ktevent: 24906
best: 0.000081, worst: 0.058525, avg: 0.010646, ktevent: 23668
best: 0.000081, worst: 0.058525, avg: 0.010528, ktevent: 23850
best: 0.000081, worst: 0.058525, avg: 0.010418, ktevent: 23489
best: 0.000081, worst: 0.058525, avg: 0.010997, ktevent: 26558
best: 0.000081, worst: 0.058525, avg: 0.010887, ktevent: 24103
best: 0.000081, worst: 0.058525, avg: 0.010773, ktevent: 24456
best: 0.000081, worst: 0.058525, avg: 0.010661, ktevent: 24052
best: 0.000081, worst: 0.058525, avg: 0.010557, ktevent: 23843
best: 0.000081, worst: 0.058525, avg: 0.011101, ktevent: 27745
best: 0.000081, worst: 0.058525, avg: 0.010986, ktevent: 24231
best: 0.000081, worst: 0.058525, avg: 0.010879, ktevent: 24192
best: 0.000081, worst: 0.058525, avg: 0.010769, ktevent: 24217
best: 0.000081, worst: 0.058525, avg: 0.011106, ktevent: 24551
best: 0.000081, worst: 0.058525, avg: 0.011011, ktevent: 24653
best: 0.000081, worst: 0.058525, avg: 0.010907, ktevent: 24181
best: 0.000081, worst: 0.058525, avg: 0.010802, ktevent: 23872
best: 0.000081, worst: 0.058525, avg: 0.010702, ktevent: 24200
best: 0.000081, worst: 0.058525, avg: 0.011023, ktevent: 23827
best: 0.000081, worst: 0.058525, avg: 0.010927, ktevent: 24351
best: 0.000081, worst: 0.058525, avg: 0.010833, ktevent: 23734
best: 0.000081, worst: 0.058525, avg: 0.010737, ktevent: 23602
best: 0.000081, worst: 0.058525, avg: 0.010641, ktevent: 23930
best: 0.000081, worst: 0.058525, avg: 0.010551, ktevent: 24126
best: 0.000081, worst: 0.058525, avg: 0.010478, ktevent: 23809
best: 0.000081, worst: 0.058525, avg: 0.010383, ktevent: 24566
best: 0.000081, worst: 0.058525, avg: 0.010295, ktevent: 23826
best: 0.000081, worst: 0.058525, avg: 0.010230, ktevent: 24312
best: 0.000081, worst: 0.058525, avg: 0.010143, ktevent: 24491
best: 0.000081, worst: 0.058525, avg: 0.010070, ktevent: 24017
best: 0.000081, worst: 0.058525, avg: 0.009994, ktevent: 24244
best: 0.000081, worst: 0.058525, avg: 0.009911, ktevent: 23479
best: 0.000081, worst: 0.058525, avg: 0.010200, ktevent: 24311
best: 0.000081, worst: 0.058525, avg: 0.010112, ktevent: 23755
best: 0.000081, worst: 0.058525, avg: 0.010032, ktevent: 24083
best: 0.000081, worst: 0.058525, avg: 0.009947, ktevent: 24050
best: 0.000081, worst: 0.058525, avg: 0.009874, ktevent: 23647
best: 0.000081, worst: 0.058525, avg: 0.009794, ktevent: 23616
best: 0.000081, worst: 0.058525, avg: 0.010210, ktevent: 24948
best: 0.000081, worst: 0.058525, avg: 0.010130, ktevent: 24107
best: 0.000081, worst: 0.058525, avg: 0.010061, ktevent: 23994
best: 0.000081, worst: 0.058525, avg: 0.009987, ktevent: 24127
best: 0.000081, worst: 0.058525, avg: 0.009917, ktevent: 24338
best: 0.000081, worst: 0.058525, avg: 0.009843, ktevent: 23974
best: 0.000081, worst: 0.058525, avg: 0.009788, ktevent: 24215
best: 0.000081, worst: 0.058525, avg: 0.009729, ktevent: 24685
best: 0.000081, worst: 0.058525, avg: 0.009669, ktevent: 24245
best: 0.000081, worst: 0.058525, avg: 0.009607, ktevent: 24674
best: 0.000081, worst: 0.058525, avg: 0.009542, ktevent: 23875
best: 0.000081, worst: 0.058525, avg: 0.009477, ktevent: 24022
best: 0.000081, worst: 0.058525, avg: 0.009415, ktevent: 24468
```

```
best: 0.000081, worst: 0.058525, avg: 0.009347, ktevent: 23919
best: 0.000081, worst: 0.058525, avg: 0.009722, ktevent: 26286
best: 0.000081, worst: 0.058525, avg: 0.009660, ktevent: 23713
best: 0.000081, worst: 0.058525, avg: 0.009902, ktevent: 24021
best: 0.000081, worst: 0.058525, avg: 0.010140, ktevent: 24059
best: 0.000081, worst: 0.058525, avg: 0.010070, ktevent: 23414
best: 0.000081, worst: 0.058525, avg: 0.010001, ktevent: 24172
best: 0.000081, worst: 0.058525, avg: 0.009931, ktevent: 24122
best: 0.000081, worst: 0.058525, avg: 0.010162, ktevent: 25142
best: 0.000081, worst: 0.058525, avg: 0.010094, ktevent: 23871
best: 0.000081, worst: 0.058525, avg: 0.010030, ktevent: 24056
best: 0.000081, worst: 0.058525, avg: 0.009964, ktevent: 24201
best: 0.000081, worst: 0.058525, avg: 0.009907, ktevent: 23854
best: 0.000081, worst: 0.058525, avg: 0.009847, ktevent: 25282
best: 0.000081, worst: 0.058525, avg: 0.009793, ktevent: 23848
best: 0.000081, worst: 0.058525, avg: 0.009739, ktevent: 24733
best: 0.000081, worst: 0.058525, avg: 0.010073, ktevent: 23962
best: 0.000081, worst: 0.058525, avg: 0.010010, ktevent: 23275
best: 0.000081, worst: 0.058525, avg: 0.009949, ktevent: 23685
best: 0.000081, worst: 0.058525, avg: 0.009888, ktevent: 24001
best: 0.000081, worst: 0.058525, avg: 0.009836, ktevent: 23401
best: 0.000081, worst: 0.058525, avg: 0.009785, ktevent: 24174
best: 0.000081, worst: 0.058525, avg: 0.010105, ktevent: 24306
best: 0.000081, worst: 0.058525, avg: 0.010314, ktevent: 25468
best: 0.000081, worst: 0.058525, avg: 0.010249, ktevent: 23089
best: 0.000081, worst: 0.058525, avg: 0.010192, ktevent: 23804
best: 0.000081, worst: 0.058525, avg: 0.010142, ktevent: 24153
best: 0.000081, worst: 0.058525, avg: 0.010085, ktevent: 24196
best: 0.000081, worst: 0.058525, avg: 0.010029, ktevent: 23989
best: 0.000081, worst: 0.058525, avg: 0.009972, ktevent: 24355
best: 0.000081, worst: 0.058525, avg: 0.009913, ktevent: 24185
best: 0.000081, worst: 0.058525, avg: 0.010215, ktevent: 24991
best: 0.000081, worst: 0.058525, avg: 0.010160, ktevent: 24133
best: 0.000081, worst: 0.058525, avg: 0.010102, ktevent: 24098
best: 0.000081, worst: 0.058525, avg: 0.010046, ktevent: 23800
best: 0.000081, worst: 0.058525, avg: 0.009993, ktevent: 23821
best: 0.000081, worst: 0.058525, avg: 0.009938, ktevent: 23859
best: 0.000081, worst: 0.058525, avg: 0.009889, ktevent: 23657
best: 0.000081, worst: 0.058525, avg: 0.010082, ktevent: 24783
best: 0.000081, worst: 0.058525, avg: 0.010368, ktevent: 24402
best: 0.000081, worst: 0.058525, avg: 0.010316, ktevent: 23693
best: 0.000081, worst: 0.058604, avg: 0.010599, ktevent: 24613
best: 0.000081, worst: 0.058604, avg: 0.010542, ktevent: 24517
best: 0.000081, worst: 0.058604, avg: 0.010490, ktevent: 23987
best: 0.000081, worst: 0.058604, avg: 0.010434, ktevent: 23548
best: 0.000081, worst: 0.058604, avg: 0.010382, ktevent: 23922
best: 0.000081, worst: 0.058604, avg: 0.010326, ktevent: 23310
best: 0.000081, worst: 0.058604, avg: 0.010272, ktevent: 24360
best: 0.000081, worst: 0.058604, avg: 0.010221, ktevent: 24482
best: 0.000081, worst: 0.058604, avg: 0.010490, ktevent: 26692
best: 0.000081, worst: 0.058604, avg: 0.010437, ktevent: 24009
best: 0.000081, worst: 0.058604, avg: 0.010385, ktevent: 23733
best: 0.000081, worst: 0.058604, avg: 0.010650, ktevent: 24149
best: 0.000081, worst: 0.058604, avg: 0.010595, ktevent: 23965
best: 0.000081, worst: 0.058604, avg: 0.010856, ktevent: 26742
best: 0.000081, worst: 0.058604, avg: 0.011113, ktevent: 23043
best: 0.000081, worst: 0.058604, avg: 0.011058, ktevent: 23413
best: 0.000081, worst: 0.058604, avg: 0.011007, ktevent: 24475
best: 0.000081, worst: 0.058604, avg: 0.010957, ktevent: 23941
best: 0.000081, worst: 0.058604, avg: 0.010909, ktevent: 23829
best: 0.000081, worst: 0.058604, avg: 0.010858, ktevent: 23758
best: 0.000081, worst: 0.058604, avg: 0.010805, ktevent: 24124
```

```
best: 0.000081, worst: 0.058604, avg: 0.010752, ktevent: 23972
best: 0.000081, worst: 0.058604, avg: 0.010699, ktevent: 24394
best: 0.000081, worst: 0.058604, avg: 0.010649, ktevent: 23843
best: 0.000081, worst: 0.058604, avg: 0.010607, ktevent: 24171
best: 0.000081, worst: 0.058604, avg: 0.010559, ktevent: 23464
best: 0.000081, worst: 0.058604, avg: 0.010802, ktevent: 26429
best: 0.000081, worst: 0.058604, avg: 0.010756, ktevent: 23243
best: 0.000081, worst: 0.058604, avg: 0.010710, ktevent: 23810
best: 0.000081, worst: 0.058604, avg: 0.010949, ktevent: 27541
best: 0.000081, worst: 0.058604, avg: 0.010897, ktevent: 23444
best: 0.000081, worst: 0.058604, avg: 0.010844, ktevent: 24041
best: 0.000081, worst: 0.058604, avg: 0.010793, ktevent: 23852
best: 0.000081, worst: 0.058604, avg: 0.010747, ktevent: 24276
best: 0.000081, worst: 0.058604, avg: 0.010980, ktevent: 24292
best: 0.000081, worst: 0.058604, avg: 0.010934, ktevent: 24544
best: 0.000081, worst: 0.058604, avg: 0.011164, ktevent: 24464
best: 0.000081, worst: 0.058604, avg: 0.011115, ktevent: 24146
best: 0.000081, worst: 0.058604, avg: 0.011342, ktevent: 27970
best: 0.000081, worst: 0.058604, avg: 0.011290, ktevent: 23633
best: 0.000081, worst: 0.058604, avg: 0.011514, ktevent: 26688
best: 0.000081, worst: 0.058604, avg: 0.011736, ktevent: 26769
best: 0.000081, worst: 0.058604, avg: 0.011687, ktevent: 24025
best: 0.000081, worst: 0.058604, avg: 0.011640, ktevent: 24113
best: 0.000081, worst: 0.058604, avg: 0.011596, ktevent: 23918
best: 0.000081, worst: 0.058604, avg: 0.011547, ktevent: 23398
best: 0.000081, worst: 0.058604, avg: 0.011497, ktevent: 23316
best: 0.000081, worst: 0.058604, avg: 0.011713, ktevent: 23345
best: 0.000081, worst: 0.058604, avg: 0.011664, ktevent: 23914
best: 0.000081, worst: 0.058604, avg: 0.011877, ktevent: 23931
best: 0.000081, worst: 0.058604, avg: 0.012088, ktevent: 23935
best: 0.000081, worst: 0.058604, avg: 0.012297, ktevent: 23239
best: 0.000081, worst: 0.058604, avg: 0.012253, ktevent: 25077
best: 0.000081, worst: 0.058604, avg: 0.012207, ktevent: 24166
best: 0.000081, worst: 0.058604, avg: 0.012161, ktevent: 23774
best: 0.000081, worst: 0.058604, avg: 0.012366, ktevent: 24869
best: 0.000081, worst: 0.058604, avg: 0.012314, ktevent: 23829
best: 0.000081, worst: 0.058604, avg: 0.012268, ktevent: 24422
best: 0.000081, worst: 0.058604, avg: 0.012469, ktevent: 26709
best: 0.000081, worst: 0.058604, avg: 0.012418, ktevent: 23660
best: 0.000081, worst: 0.058604, avg: 0.012371, ktevent: 24089
best: 0.000081, worst: 0.058604, avg: 0.012325, ktevent: 23847
best: 0.000081, worst: 0.058604, avg: 0.012273, ktevent: 23725
best: 0.000081, worst: 0.058604, avg: 0.012227, ktevent: 24553
best: 0.000081, worst: 0.058604, avg: 0.012179, ktevent: 23713
best: 0.000081, worst: 0.058604, avg: 0.012131, ktevent: 24344
best: 0.000081, worst: 0.058604, avg: 0.012082, ktevent: 23962
best: 0.000081, worst: 0.058604, avg: 0.012033, ktevent: 23881
best: 0.000081, worst: 0.058604, avg: 0.011984, ktevent: 24054
best: 0.000081, worst: 0.058604, avg: 0.011942, ktevent: 24263
best: 0.000081, worst: 0.058604, avg: 0.011917, ktevent: 24294
best: 0.000081, worst: 0.058604, avg: 0.011873, ktevent: 23973
best: 0.000081, worst: 0.058604, avg: 0.012064, ktevent: 24873
best: 0.000081, worst: 0.058604, avg: 0.012017, ktevent: 24164
best: 0.000081, worst: 0.058604, avg: 0.011973, ktevent: 24336
best: 0.000081, worst: 0.058604, avg: 0.011931, ktevent: 24192
best: 0.000081, worst: 0.058604, avg: 0.011885, ktevent: 24153
best: 0.000081, worst: 0.058604, avg: 0.011846, ktevent: 23929
best: 0.000081, worst: 0.058604, avg: 0.011805, ktevent: 24237
best: 0.000081, worst: 0.058604, avg: 0.011762, ktevent: 23708
best: 0.000081, worst: 0.058604, avg: 0.011719, ktevent: 23772
best: 0.000081, worst: 0.058604, avg: 0.011675, ktevent: 23848
best: 0.000081, worst: 0.058604, avg: 0.011634, ktevent: 24267
```

```
best: 0.000081, worst: 0.058604, avg: 0.011818, ktevent: 27198
best: 0.000081, worst: 0.058604, avg: 0.011773, ktevent: 24070
best: 0.000081, worst: 0.058604, avg: 0.011736, ktevent: 23919
best: 0.000081, worst: 0.058604, avg: 0.011693, ktevent: 24539
best: 0.000081, worst: 0.058604, avg: 0.011653, ktevent: 24233
best: 0.000081, worst: 0.058604, avg: 0.011611, ktevent: 23710
best: 0.000081, worst: 0.058604, avg: 0.011571, ktevent: 24127
best: 0.000081, worst: 0.058604, avg: 0.011531, ktevent: 24122
best: 0.000081, worst: 0.095106, avg: 0.011850, ktevent: 31189
best: 0.000081, worst: 0.095106, avg: 0.011808, ktevent: 24039
best: 0.000081, worst: 0.095106, avg: 0.011770, ktevent: 24043
best: 0.000081, worst: 0.095106, avg: 0.011745, ktevent: 23794
best: 0.000081, worst: 0.095106, avg: 0.011712, ktevent: 24300
best: 0.000081, worst: 0.095106, avg: 0.011669, ktevent: 23940
best: 0.000081, worst: 0.095106, avg: 0.011629, ktevent: 24603
best: 0.000081, worst: 0.095106, avg: 0.011803, ktevent: 25963
best: 0.000081, worst: 0.095106, avg: 0.011763, ktevent: 24402
best: 0.000081, worst: 0.095106, avg: 0.011720, ktevent: 24162
best: 0.000081, worst: 0.095106, avg: 0.011681, ktevent: 24091
best: 0.000081, worst: 0.095106, avg: 0.011643, ktevent: 24337
best: 0.000081, worst: 0.095106, avg: 0.011605, ktevent: 23740
best: 0.000081, worst: 0.095106, avg: 0.011566, ktevent: 23951
best: 0.000081, worst: 0.095106, avg: 0.011526, ktevent: 24121
best: 0.000081, worst: 0.095106, avg: 0.011490, ktevent: 24060
best: 0.000081, worst: 0.095106, avg: 0.011790, ktevent: 31469
best: 0.000081, worst: 0.095106, avg: 0.011753, ktevent: 24251
best: 0.000081, worst: 0.095106, avg: 0.011920, ktevent: 25354
best: 0.000081, worst: 0.095106, avg: 0.011881, ktevent: 24513
best: 0.000081, worst: 0.095106, avg: 0.011844, ktevent: 24361
best: 0.000081, worst: 0.095106, avg: 0.011804, ktevent: 23883
best: 0.000081, worst: 0.095106, avg: 0.011763, ktevent: 23582
best: 0.000081, worst: 0.095106, avg: 0.011728, ktevent: 24312
best: 0.000081, worst: 0.095106, avg: 0.011691, ktevent: 23762
best: 0.000081, worst: 0.095106, avg: 0.011653, ktevent: 23992
best: 0.000081, worst: 0.095106, avg: 0.011613, ktevent: 23781
best: 0.000081, worst: 0.095106, avg: 0.011575, ktevent: 24187
best: 0.000081, worst: 0.095106, avg: 0.011539, ktevent: 24217
best: 0.000081, worst: 0.095106, avg: 0.011505, ktevent: 24096
best: 0.000081, worst: 0.095106, avg: 0.011468, ktevent: 23798
best: 0.000081, worst: 0.095106, avg: 0.011430, ktevent: 24372
best: 0.000081, worst: 0.095106, avg: 0.011395, ktevent: 24191
best: 0.000081, worst: 0.095106, avg: 0.011395, ktevent: 24400
best: 0.000081, worst: 0.095106, avg: 0.011361, ktevent: 24093
best: 0.000081, worst: 0.095106, avg: 0.011520, ktevent: 25226
best: 0.000081, worst: 0.095106, avg: 0.011486, ktevent: 24179
best: 0.000081, worst: 0.095106, avg: 0.011452, ktevent: 23846
best: 0.000081, worst: 0.095106, avg: 0.011423, ktevent: 23487
best: 0.000081, worst: 0.095106, avg: 0.011389, ktevent: 23542
best: 0.000081, worst: 0.095106, avg: 0.011356, ktevent: 24163
best: 0.000081, worst: 0.095106, avg: 0.011320, ktevent: 24275
best: 0.000081, worst: 0.095106, avg: 0.011289, ktevent: 23501
best: 0.000081, worst: 0.095106, avg: 0.011255, ktevent: 23419
best: 0.000081, worst: 0.095106, avg: 0.011220, ktevent: 24062
best: 0.000081, worst: 0.095106, avg: 0.011374, ktevent: 23738
best: 0.000081, worst: 0.095106, avg: 0.011342, ktevent: 24042
best: 0.000081, worst: 0.095135, avg: 0.011613, ktevent: 31065
best: 0.000081, worst: 0.095135, avg: 0.011578, ktevent: 23919
best: 0.000081, worst: 0.095135, avg: 0.011729, ktevent: 26736
best: 0.000081, worst: 0.095135, avg: 0.011695, ktevent: 23260
best: 0.000081, worst: 0.095135, avg: 0.011662, ktevent: 24146
best: 0.000081, worst: 0.095135, avg: 0.011628, ktevent: 23615
best: 0.000081, worst: 0.095135, avg: 0.011597, ktevent: 23966
```

```
best: 0.000081, worst: 0.095135, avg: 0.011563, ktevent: 23989
best: 0.000081, worst: 0.095135, avg: 0.011529, ktevent: 24184
best: 0.000081, worst: 0.095135, avg: 0.011495, ktevent: 23716
best: 0.000081, worst: 0.095135, avg: 0.011462, ktevent: 23531
best: 0.000081, worst: 0.095135, avg: 0.011431, ktevent: 24580
best: 0.000081, worst: 0.095135, avg: 0.011398, ktevent: 23780
best: 0.000081, worst: 0.095135, avg: 0.011366, ktevent: 24095
best: 0.000081, worst: 0.095135, avg: 0.011334, ktevent: 23867
best: 0.000081, worst: 0.095135, avg: 0.011305, ktevent: 23629
best: 0.000081, worst: 0.095135, avg: 0.011276, ktevent: 23740
best: 0.000081, worst: 0.095135, avg: 0.011249, ktevent: 24514
best: 0.000081, worst: 0.095135, avg: 0.011216, ktevent: 24245
best: 0.000081, worst: 0.095135, avg: 0.011185, ktevent: 24464
best: 0.000081, worst: 0.095135, avg: 0.011155, ktevent: 24328
best: 0.000081, worst: 0.095135, avg: 0.011128, ktevent: 24123
best: 0.000081, worst: 0.095135, avg: 0.011095, ktevent: 23591
best: 0.000081, worst: 0.095135, avg: 0.011066, ktevent: 23443
best: 0.000081, worst: 0.095135, avg: 0.011034, ktevent: 23818
best: 0.000081, worst: 0.095135, avg: 0.011003, ktevent: 24232
best: 0.000081, worst: 0.095135, avg: 0.010980, ktevent: 23840
best: 0.000081, worst: 0.095135, avg: 0.010953, ktevent: 23857
best: 0.000081, worst: 0.095135, avg: 0.010923, ktevent: 23776
best: 0.000081, worst: 0.095135, avg: 0.010893, ktevent: 24390
best: 0.000081, worst: 0.095135, avg: 0.010866, ktevent: 24326
best: 0.000081, worst: 0.095135, avg: 0.010838, ktevent: 24077
best: 0.000081, worst: 0.095135, avg: 0.011085, ktevent: 30423
best: 0.000081, worst: 0.095135, avg: 0.011058, ktevent: 23678
best: 0.000081, worst: 0.095135, avg: 0.011031, ktevent: 24566
best: 0.000081, worst: 0.095135, avg: 0.011169, ktevent: 27632
best: 0.000081, worst: 0.095135, avg: 0.011306, ktevent: 24667
best: 0.000081, worst: 0.095135, avg: 0.011277, ktevent: 23903
best: 0.000081, worst: 0.095135, avg: 0.011413, ktevent: 24462
best: 0.000081, worst: 0.095135, avg: 0.011383, ktevent: 24046
best: 0.000081, worst: 0.095135, avg: 0.011353, ktevent: 23824
best: 0.000081, worst: 0.095135, avg: 0.011326, ktevent: 24033
best: 0.000081, worst: 0.095135, avg: 0.011297, ktevent: 23633
best: 0.000081, worst: 0.095135, avg: 0.011269, ktevent: 23924
best: 0.000081, worst: 0.095135, avg: 0.011238, ktevent: 24077
best: 0.000081, worst: 0.095135, avg: 0.011209, ktevent: 23757
best: 0.000081, worst: 0.095135, avg: 0.011180, ktevent: 24534
best: 0.000081, worst: 0.095135, avg: 0.011151, ktevent: 24196
best: 0.000081, worst: 0.095135, avg: 0.011123, ktevent: 23530
best: 0.000081, worst: 0.095135, avg: 0.011094, ktevent: 24177
best: 0.000081, worst: 0.095135, avg: 0.011067, ktevent: 23885
best: 0.000081, worst: 0.095135, avg: 0.011154, ktevent: 24159
best: 0.000081, worst: 0.095135, avg: 0.011126, ktevent: 23971
best: 0.000081, worst: 0.095135, avg: 0.011257, ktevent: 26733
best: 0.000081, worst: 0.095135, avg: 0.011373, ktevent: 26827
best: 0.000081, worst: 0.095135, avg: 0.011347, ktevent: 24376
best: 0.000081, worst: 0.095135, avg: 0.011319, ktevent: 23544
best: 0.000081, worst: 0.095135, avg: 0.011293, ktevent: 23784
best: 0.000081, worst: 0.095135, avg: 0.011265, ktevent: 24101
best: 0.000081, worst: 0.095135, avg: 0.011238, ktevent: 23858
best: 0.000081, worst: 0.095135, avg: 0.011211, ktevent: 23872
best: 0.000081, worst: 0.095135, avg: 0.011338, ktevent: 24689
best: 0.000081, worst: 0.095135, avg: 0.011311, ktevent: 24436
best: 0.000081, worst: 0.095135, avg: 0.011286, ktevent: 24020
best: 0.000081, worst: 0.095135, avg: 0.011258, ktevent: 23875
best: 0.000081, worst: 0.095135, avg: 0.011229, ktevent: 23732
best: 0.000081, worst: 0.095135, avg: 0.011201, ktevent: 24091
best: 0.000081, worst: 0.095135, avg: 0.011175, ktevent: 24194
best: 0.000081, worst: 0.095135, avg: 0.011300, ktevent: 25693
```

```
best: 0.000081, worst: 0.095135, avg: 0.011272, ktevent: 23781
best: 0.000081, worst: 0.095135, avg: 0.011247, ktevent: 23677
best: 0.000081, worst: 0.095135, avg: 0.011371, ktevent: 26343
best: 0.000081, worst: 0.095135, avg: 0.011351, ktevent: 24248
best: 0.000081, worst: 0.095135, avg: 0.011322, ktevent: 24074
best: 0.000081, worst: 0.095135, avg: 0.011296, ktevent: 23894
best: 0.000081, worst: 0.095135, avg: 0.011273, ktevent: 24420
best: 0.000081, worst: 0.095135, avg: 0.011247, ktevent: 24379
best: 0.000081, worst: 0.095135, avg: 0.011220, ktevent: 23544
best: 0.000081, worst: 0.095135, avg: 0.011196, ktevent: 25016
best: 0.000081, worst: 0.095135, avg: 0.011168, ktevent: 23846
best: 0.000081, worst: 0.095135, avg: 0.011143, ktevent: 24165
best: 0.000081, worst: 0.095135, avg: 0.011115, ktevent: 23796
best: 0.000081, worst: 0.095135, avg: 0.011091, ktevent: 23770
best: 0.000081, worst: 0.095135, avg: 0.011066, ktevent: 24011
best: 0.000081, worst: 0.095135, avg: 0.011040, ktevent: 24059
best: 0.000081, worst: 0.095135, avg: 0.011015, ktevent: 24570
best: 0.000081, worst: 0.095135, avg: 0.011135, ktevent: 24458
best: 0.000081, worst: 0.095135, avg: 0.011255, ktevent: 24714
best: 0.000081, worst: 0.095135, avg: 0.011233, ktevent: 24098
best: 0.000081, worst: 0.095135, avg: 0.011210, ktevent: 24420
best: 0.000081, worst: 0.095135, avg: 0.011185, ktevent: 24046
best: 0.000081, worst: 0.095135, avg: 0.011162, ktevent: 23584
best: 0.000081, worst: 0.095135, avg: 0.011139, ktevent: 24166
best: 0.000081, worst: 0.095135, avg: 0.011116, ktevent: 24200
best: 0.000081, worst: 0.095135, avg: 0.011092, ktevent: 23774
best: 0.000081, worst: 0.095135, avg: 0.011068, ktevent: 24039
best: 0.000081, worst: 0.095135, avg: 0.011044, ktevent: 23449
best: 0.000081, worst: 0.095135, avg: 0.011022, ktevent: 24768
best: 0.000081, worst: 0.095135, avg: 0.011123, ktevent: 26759
best: 0.000081, worst: 0.095135, avg: 0.011098, ktevent: 23957
best: 0.000081, worst: 0.095135, avg: 0.011073, ktevent: 24222
best: 0.000081, worst: 0.095135, avg: 0.011188, ktevent: 22887
best: 0.000081, worst: 0.095135, avg: 0.011163, ktevent: 23987
best: 0.000081, worst: 0.095135, avg: 0.011140, ktevent: 24266
best: 0.000081, worst: 0.095135, avg: 0.011119, ktevent: 24458
best: 0.000081, worst: 0.095135, avg: 0.011093, ktevent: 24410
best: 0.000081, worst: 0.095135, avg: 0.011070, ktevent: 24074
best: 0.000081, worst: 0.095135, avg: 0.011184, ktevent: 27677
best: 0.000081, worst: 0.095135, avg: 0.011160, ktevent: 23254
best: 0.000081, worst: 0.095135, avg: 0.011361, ktevent: 30789
best: 0.000081, worst: 0.095135, avg: 0.011337, ktevent: 24462
best: 0.000081, worst: 0.095135, avg: 0.011312, ktevent: 24214
best: 0.000081, worst: 0.095135, avg: 0.011287, ktevent: 23675
best: 0.000081, worst: 0.095135, avg: 0.011263, ktevent: 24086
best: 0.000081, worst: 0.095135, avg: 0.011240, ktevent: 23673
best: 0.000081, worst: 0.095135, avg: 0.011217, ktevent: 24015
best: 0.000081, worst: 0.095135, avg: 0.011193, ktevent: 24359
best: 0.000081, worst: 0.095135, avg: 0.011167, ktevent: 24073
best: 0.000081, worst: 0.095135, avg: 0.011278, ktevent: 24684
best: 0.000081, worst: 0.095135, avg: 0.011350, ktevent: 24072
best: 0.000081, worst: 0.095135, avg: 0.011327, ktevent: 23569
best: 0.000081, worst: 0.095135, avg: 0.011302, ktevent: 23415
best: 0.000081, worst: 0.095135, avg: 0.011278, ktevent: 23807
best: 0.000081, worst: 0.095135, avg: 0.011387, ktevent: 22803
best: 0.000081, worst: 0.095135, avg: 0.011366, ktevent: 24029
best: 0.000081, worst: 0.095135, avg: 0.011343, ktevent: 24283
best: 0.000081, worst: 0.095135, avg: 0.011320, ktevent: 23581
best: 0.000081, worst: 0.095135, avg: 0.011297, ktevent: 23924
best: 0.000081, worst: 0.095135, avg: 0.011276, ktevent: 23559
best: 0.000081, worst: 0.095135, avg: 0.011253, ktevent: 24329
best: 0.000081, worst: 0.095135, avg: 0.011232, ktevent: 23527
```

```
best: 0.000081, worst: 0.095135, avg: 0.011208, ktevent: 23862
best: 0.000081, worst: 0.095135, avg: 0.011185, ktevent: 23788
best: 0.000081, worst: 0.095135, avg: 0.011162, ktevent: 23867
best: 0.000081, worst: 0.095135, avg: 0.011139, ktevent: 23614
best: 0.000081, worst: 0.095135, avg: 0.011208, ktevent: 24924
best: 0.000081, worst: 0.095135, avg: 0.011184, ktevent: 24056
best: 0.000081, worst: 0.095135, avg: 0.011162, ktevent: 24033
best: 0.000081, worst: 0.095135, avg: 0.011142, ktevent: 24438
best: 0.000081, worst: 0.095135, avg: 0.011247, ktevent: 24240
best: 0.000081, worst: 0.095135, avg: 0.011225, ktevent: 23543
best: 0.000081, worst: 0.095135, avg: 0.011201, ktevent: 24165
best: 0.000081, worst: 0.095135, avg: 0.011180, ktevent: 23816
best: 0.000081, worst: 0.095135, avg: 0.011161, ktevent: 24526
best: 0.000081, worst: 0.095135, avg: 0.011142, ktevent: 23784
best: 0.000081, worst: 0.095135, avg: 0.011120, ktevent: 23824
best: 0.000081, worst: 0.095135, avg: 0.011098, ktevent: 23531
best: 0.000081, worst: 0.095135, avg: 0.011074, ktevent: 24363
best: 0.000081, worst: 0.095135, avg: 0.011053, ktevent: 24063
best: 0.000081, worst: 0.095135, avg: 0.011035, ktevent: 24120
best: 0.000081, worst: 0.095135, avg: 0.011138, ktevent: 26998
best: 0.000081, worst: 0.095135, avg: 0.011117, ktevent: 23802
best: 0.000081, worst: 0.095135, avg: 0.011094, ktevent: 23879
best: 0.000081, worst: 0.095135, avg: 0.011073, ktevent: 24365
best: 0.000081, worst: 0.095135, avg: 0.011175, ktevent: 24834
best: 0.000081, worst: 0.095135, avg: 0.011153, ktevent: 23948
best: 0.000081, worst: 0.095135, avg: 0.011132, ktevent: 23856
best: 0.000081, worst: 0.095135, avg: 0.011109, ktevent: 23685
best: 0.000081, worst: 0.095135, avg: 0.011090, ktevent: 24308
best: 0.000081, worst: 0.095135, avg: 0.011068, ktevent: 23407
best: 0.000081, worst: 0.095135, avg: 0.011045, ktevent: 23911
best: 0.000081, worst: 0.095135, avg: 0.011024, ktevent: 23829
best: 0.000081, worst: 0.095135, avg: 0.011004, ktevent: 23661
best: 0.000081, worst: 0.095135, avg: 0.011105, ktevent: 27573
best: 0.000081, worst: 0.095135, avg: 0.011083, ktevent: 23707
best: 0.000081, worst: 0.095135, avg: 0.011061, ktevent: 24291
best: 0.000081, worst: 0.095135, avg: 0.011161, ktevent: 26111
best: 0.000081, worst: 0.095135, avg: 0.011139, ktevent: 24021
best: 0.000081, worst: 0.095135, avg: 0.011122, ktevent: 24381
best: 0.000081, worst: 0.095135, avg: 0.011100, ktevent: 23655
best: 0.000081, worst: 0.095135, avg: 0.011078, ktevent: 24184
best: 0.000081, worst: 0.095135, avg: 0.011061, ktevent: 23345
best: 0.000081, worst: 0.095135, avg: 0.011040, ktevent: 24831
best: 0.000081, worst: 0.095135, avg: 0.011020, ktevent: 23858
best: 0.000081, worst: 0.095135, avg: 0.011118, ktevent: 24167
best: 0.000081, worst: 0.095135, avg: 0.011098, ktevent: 24321
best: 0.000081, worst: 0.095135, avg: 0.011077, ktevent: 24162
best: 0.000081, worst: 0.095135, avg: 0.011175, ktevent: 26619
best: 0.000081, worst: 0.095135, avg: 0.011153, ktevent: 23998
best: 0.000081, worst: 0.095135, avg: 0.011132, ktevent: 24112
best: 0.000081, worst: 0.095135, avg: 0.011110, ktevent: 24229
best: 0.000081, worst: 0.095135, avg: 0.011207, ktevent: 26564
best: 0.000081, worst: 0.095135, avg: 0.011187, ktevent: 23866
best: 0.000081, worst: 0.095135, avg: 0.011165, ktevent: 23378
best: 0.000081, worst: 0.095135, avg: 0.011261, ktevent: 24991
best: 0.000081, worst: 0.095135, avg: 0.011240, ktevent: 24229
best: 0.000081, worst: 0.095135, avg: 0.011221, ktevent: 24421
best: 0.000081, worst: 0.095135, avg: 0.011200, ktevent: 23435
best: 0.000081, worst: 0.095135, avg: 0.011295, ktevent: 25129
best: 0.000081, worst: 0.095135, avg: 0.011273, ktevent: 23652
best: 0.000081, worst: 0.095135, avg: 0.011253, ktevent: 24062
in 500 runs, 2000 epochs, best: 0.000081, worst: 0.095135, avg: 0.011230
UnitTestGenAlgPerf OK
```

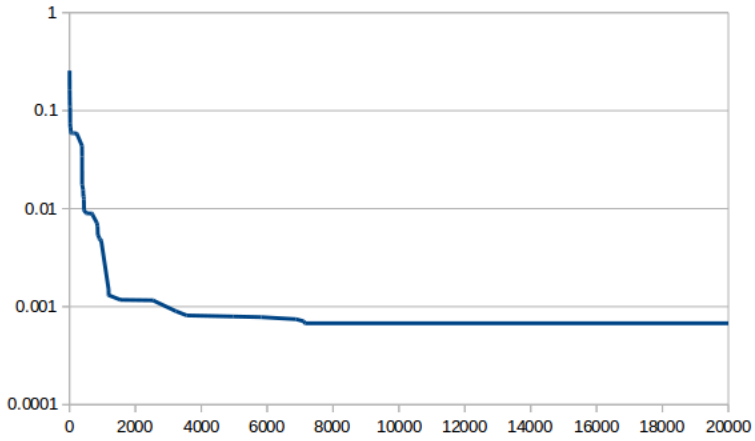
```
UnitTestGenAlg OK
UnitTestAll OK
```

UnitTestGenAlgLoadSave.txt:


```
{
  "_type":"0",
  "_nbAdns":"3",
  "_nbElites":"2",
  "_lengthAdnF":"2",
  "_lengthAdnI":"2",
  "_curEpoch":"1",
  "_nextId":"4",
  "_boundFloat":[
    {
      "_dim":"2",
      "_val":["-1.000000","1.000000"]
    },
    {
      "_dim":"2",
      "_val":["-1.000000","1.000000"]
    }
  ],
  "_boundInt":[
    {
      "_dim":"2",
      "_val":["1","10"]
    },
    {
      "_dim":"2",
      "_val":["1","10"]
    }
  ],
  "_adns":[
    {
      "_id":"3",
      "_age":"1",
      "_elo":"0.000000",
      "_val":"0.000000",
      "_adnF":{
        "_dim":"2",
        "_val":["0.672516","-0.003504"]
      },
      "_deltaAdnF":{
        "_dim":"2",
        "_val":["-0.115488","0.000000"]
      },
      "_adnI":{
        "_dim":"2",
        "_val":["3","1"]
      }
    },
    {
      "_id":"1",
      "_age":"2",
      "_elo":"0.000000",
      "_val":"0.000000",
      "_adnF":{
        "_dim":"2",
        "_val":["-0.840711","-0.704622"]
      },
```

```
      "_deltaAdnF":{
        "_dim":"2",
        "_val":["0.000000","0.000000"]
      },
      "_adnI":{
        "_dim":"2",
        "_val":["5","4"]
      }
    },
    {
      "_id":"0",
      "_age":"2",
      "_elo":"0.000000",
      "_val":"0.000000",
      "_adnF":{
        "_dim":"2",
        "_val":["0.788004","-0.003504"]
      },
      "_deltaAdnF":{
        "_dim":"2",
        "_val":["0.000000","0.000000"]
      },
      "_adnI":{
        "_dim":"2",
        "_val":["3","1"]
      }
    }
  ],
  "_bestAdn":{
    "_id":"0",
    "_age":"1",
    "_elo":"0.000000",
    "_val":"0.000000",
    "_adnF":{
      "_dim":"2",
      "_val":["0.788004","-0.003504"]
    },
    "_deltaAdnF":{
      "_dim":"2",
      "_val":["0.000000","0.000000"]
    },
    "_adnI":{
      "_dim":"2",
      "_val":["3","1"]
    }
  }
}
```


eval() of best genes over epoch:


1
0.1
0.01
0.001
0.0001
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000