DETERMINING THE GLOBAL MAXIMUM OF A REAL FUNCTION USING GENETIC ALGORITHMS

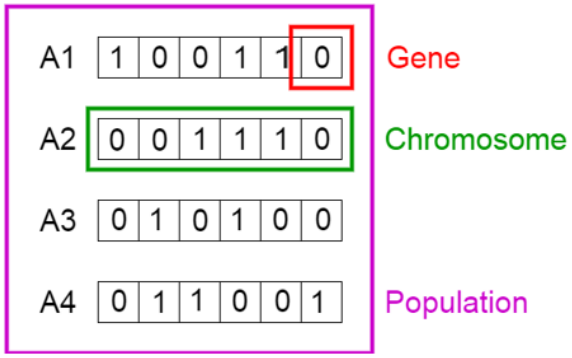
Consider the optimization problem: **with precision ε, determine the true value x for which $\max_{x\in[a,b]\subset\mathbf{R}} g(x)$.**

Genetic algorithms are **probabilistic** algorithms that simulate evolution through a sequence of *generations* (iterative process) of a *population* of potential fixed-size solutions. The possible solutions of the problem (population) are represented as chromosomes. The new population is generated from the previous one by specific genetic methods (crosses between chromosomes and spontaneous mutations). In this way, an attempt is made to improve the chromosome population within the time available in order to get as close as possible to the optimal solution.

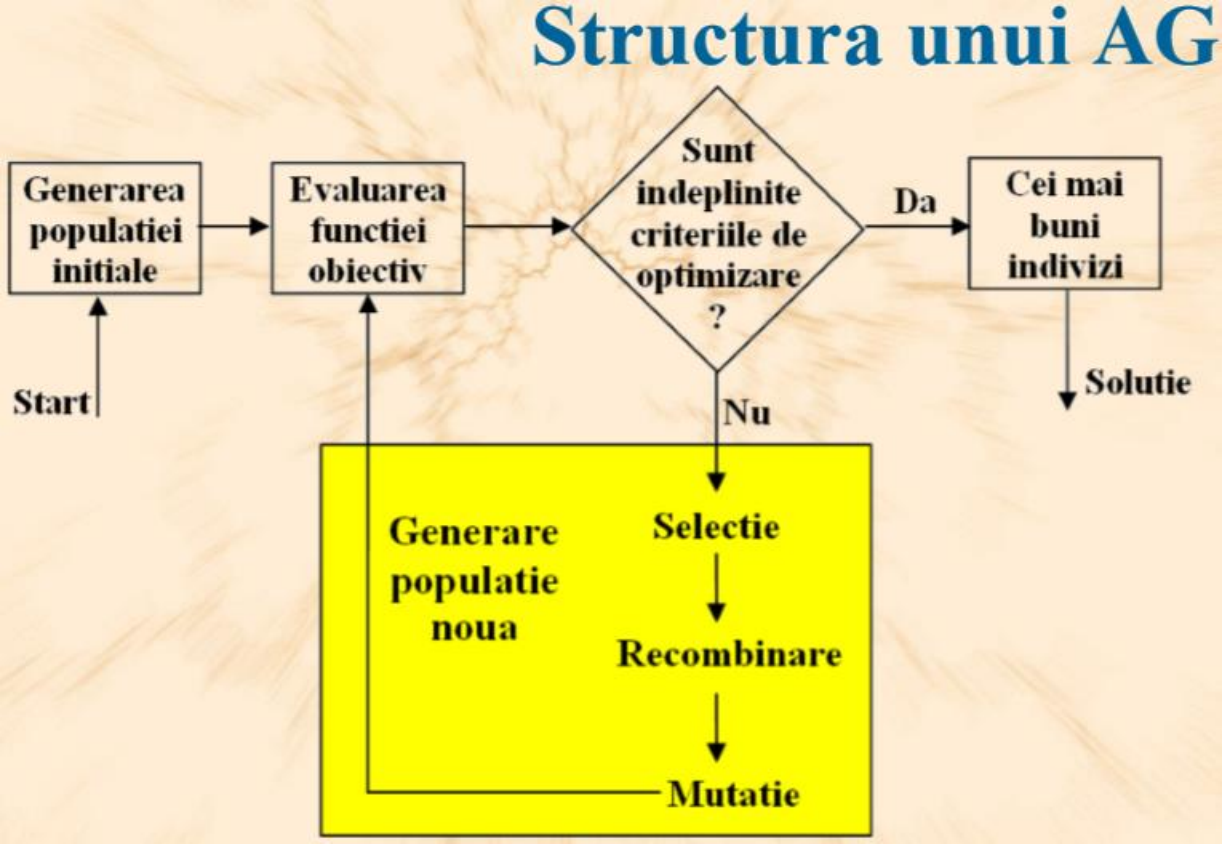
Terminology

Generic/evolutionary algorithms use a vocabulary borrowed from genetics:

- evolution is simulated by a succession of **generations of** a **population** of possible solutions;
- One possible solution is called a **chromosome** and is represented as a string of genes;
- **gene** is the atomic information in a chromosome;



- The position that a gene occupies is called the **locus**;
- all possible values for a gene form the **allele** set of the gene;
- the population evolves through the application of **genetic operators**: **mutation** and **interbreeding**;
- the chromosome to which a genetic operator is applied is called the **parent and the** resulting chromosome is called the **offspring**;
- **selection** is the procedure by which the chromosomes that will survive into the next generation are chosen; better adapted individuals will be given a better chance;
- the degree of adaptation to the environment is measured by the **fitness function**;
- **the solution** returned by a genetic algorithm is **the best individual of the last generation**.



The main steps of genetic algorithms:

Input data:

f	//the fitness function, e.g. $f(x) = 2 + x \cdot sin(10\pi \cdot x)$
a, ***b***	//the range interval in which the optimal solution is sought, $a<b$
d	//precision *eps = 10^(-d)*, e.g. ***d* = 6**, ***a* = -1, b=2**

Initialisations:

$t \leftarrow 0$	//moment of time 0
$m \leftarrow (b-a)*10^d$	//number of individuals in the population
$n \leftarrow \lceil log_2 m \rceil$	//number of bits of each chromosome
tMax	// maximum number of iterations / generations (e.g. 50)
pc	//rate (probability) of crossing (e.g. 0.6)
pm	//rata (probability) of mutation (e.g. 0.01)

[InitializationPopulation] - **Create the initial population** by randomly generating a fixed number *m* of chromosomes with *n* genes from the search space.

[CalculFitness] - **Evaluate** the success of each chromosome x in the initial population using the fitness function $f(x)$. //calculate the fitness of the chromosomes in the initial population

as long as the stop criterion [Stop] is not met repeat
//the algorithm stops if: the best fitness function value is good enough or //if the maximum number of iterations/generations has been reached, etc.

t ← **t** + 1 // increment the time counter

[New population] - Create a new population by applying the following steps until the new population is complete:

a. **[ParentSelection]** - Random **selection** of 2 parents p_1 and p_2 (chromosomes from the current population) according to their fitness f.
b. **[Crossover]** - **Combining** the genes of the 2 parents p_1 and p_2 by crossover to produce 2 children, f_1 and f_2 . The children will have the genes of the parents, with some probability of crossover, certain genes are exchanged between the children.
c. **[Mutation]** - Random **modification of the** genes of children f_1 and f_2 by mutation (with a probability of mutation, certain genes will change).
d. **[Insert]** - **Insert** the two children f_1 and f_2 into the new population.

[Replacement] - We use the newly generated population for a new run of the algorithm, first evaluating the fitness function for this population.

Output data: best chromosome (with highest fitness value).
//for example, x = 1.850773 (c = 11110011010000100000101), and max f = 3.850227.

1. Fitness function

At each step *t*, **the current population** will be **evaluated**, which involves calculating the fitness function for each individual (here real elements ***x*[1],..., *x*[*m*]** in the interval **[*a*,*b*]**). This can be done in an ***eval*** vector:

```
for i=1,m
        eval[i] <- f(x[i])
```

The fitness function *f* is used to measure the quality/chance of survival of chromosomes. It is formulated from the **numerical optimization function** ***g***. It must be **positive** and is constructed for **maximization** (adapted individuals obtain high fitness function values).

If the optimized *g-function*

- is minimum, the fitness function f is obtained by multiplying by -1:
  $f(x) = -g(x)$ $min\{g(x)\} = -max\{-g(x)\} = -max\{f(x)\}$
- for positive functions requiring minimization it is preferable to use the fitness function obtained by inverting the function value:
  $f(x) = 1/g(x)$.
- is not positive, the fitness function f is constructed by adding a constant large enough so that the result is positive

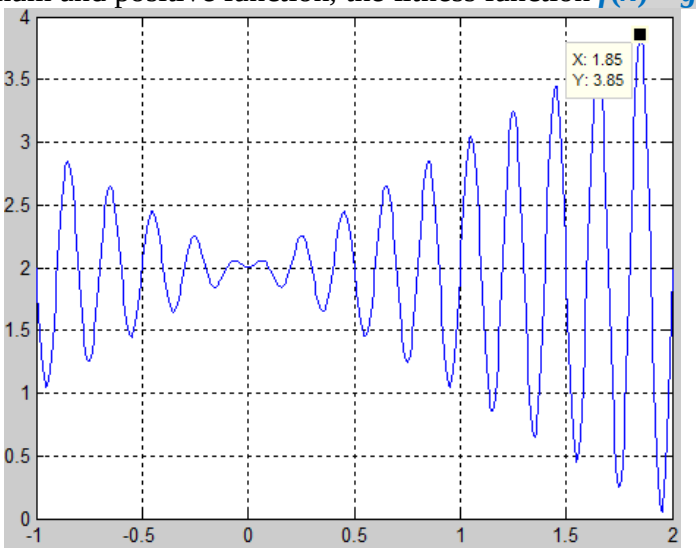
$f(x) = g(x) + C$, $C >= \text{abs}(\min(f(x)))$

The higher the constant, the more the fitness variance in the population decreases, and thus the selection pressure for fitness-proportional selection schemes decreases.

For example, this paper will consider the optimization problem

$max\ g(x) = 2 + x \cdot sin(10\pi \cdot x)$, **for** $x \in [-1, 2]$

Being a maximum and positive function, the fitness function $f(x) = g(x)$ can be chosen.



2. Representation of candidate solution points

In the case of the optimization problem considered here, the **search space of the solution of the** optimization problem is given by the interval **[*a, b*]**. Obviously we cannot consider all the points (candidates for the solution of the problem) in this interval, and then compare the values of the functions at these points, but we will work with **populations of *m*** equidistant **points** in this interval.

Each of the ***m* points *x* in the interval [*a,b*]** will be encoded by a **binary string *c*** (chromosome) **of length *n***. This length of chromosomes is chosen so as to ensure the accuracy of the solution ***ε***. For this, the search space [*a,b*] will be discretized up to a certain **precision $\varepsilon = 10^{-d}$** into subintervals of the same length. Therefore, the points in the interval [*a*, *b*] will be

$$m = (b-a)*10^d .$$

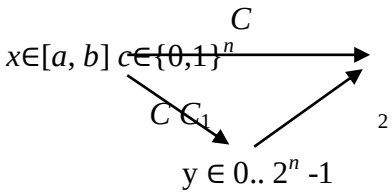
In order to represent the $(b-a)*10^d$ values, a number of

$$n = \text{whole\_part\_by\_adaos}(\log_2 (m)) \text{ bits}.$$

The length of the bit string representing a possible solution will be the sum of the lengths of the representations for each interval in the definition domain of the function to be optimized.

2.1 Binary coding

The coding function is an application $\boldsymbol{C : [a, b] \rightarrow \{0, 1\}^n}$. A **real element *x* in the interval [*a, b*]** will be encoded by a **binary string *c*** (chromosome) **of length *n***. For this we can compose two functions, one that transforms the real number interval [*a*, *b*] into the integer interval 0.. 2^n -1, respectively the second function transforms this interval (integers) into their binary representation.



The linear transformation C_1 is done starting from the relation $\frac{x-a}{b-a} = \frac{y-0}{2^n-1}$, which gives $y = \frac{(x-a)(2^n-1)}{b-a}$,
respectively $x = a + \frac{y(b-a)}{2^n-1}$.

Input parameters: a, b, n, x (e.g. a=-1, b=2, n=22). *Output parameter: c*
Thus, **encoding an element *x* in the range [*a,b*] into a string *c*** (chromosome) of **length *n*** is done in two steps:
1. determine the corresponding integer from the range 0.. 2^n -1: $\boldsymbol{y = \frac{(x-a)(2^n-1)}{b-a}}$ 2. then pass this decimal number *y* **in base 2** (by repeated divisions to 2 as it is the representation of a natural number in base 2), representing it on ***n* bits** and the result is the chromosome *c*.

For example, for the function $f(x) = 2 + x \cdot sin(10\pi \cdot x)$, $x \in [-1, 2]$ ($a = -1$, $b = 2$), $d = 6$ decimal places precision, we obtain $n = 22$ bits for the representation of each chromosome. In the table below are some examples of representations (chromosomes) corresponding to real values in the range [*a*,*b*], respectively the fitness values of these points.

x ∈ [-1, 2]	$c \in \{0,1\}^n$	$f(x) = 2 + x \cdot sin(10\pi \cdot x)$
- 1	0000000000000000000000	2
+ 0,637197	1000101110110101000111	2,586345
- 0,958973	0000001110000000010000	1,078878
+ 1,627888	1110000000111111000101	3,250650
2	1111111111111111111111	2

2.2 Binary decoding

Decoding an individual represented **by a bit string of length *n*** into a **real element *x* in the range [*a*,*b*]**:

1. write the number in base 10 corresponding to this bit string (e.g. for the binary string $c=101101_2$ (r=6) $y = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 45_{10} \in 0...2^n - 1$),
2. then determine the real number based on the translation of the integer interval $0...2^n - 1$ into the real number interval $[a, b]$: $z = a + (y - 0)(b-a)/(2^n - 1)$.

The components of the previous table can be considered as test data.

3. Initial population

The initial population (generation 0) can be created by randomly choosing a number *m* of points from the range [*a*, *b*], then each point will be encoded by a binary string (chromosome) of length *n*.

Or chromosomes can be randomly generated directly = *m* strings of *n* randomly chosen bits are generated.

In the programme we will need:

- generation of **integer random numbers between 0 and k-1** and
- generation of **real random numbers in the interval [0, 1)**.

Depending on the programming language used, there may or may not be predefined functions for both types of generation. If not, you can proceed as below.

If there is a predefined function for generating integers between 0 and *n-1*, then we can determine a real random number in the interval [0, 1] such that ($0 \le x < k \Rightarrow 0 \le x/k < 1$):

1. Generate *x* a random integer between 0 and k-1
2. *y = (float) x / k* (real division => ***y* will be real random number in the range [0, 1)**)

Conversely, if there is a predefined function for generating real random numbers in the interval [0, 1), then we can determine an integer random number between 0 and k-1 as follows ($0 \le x < 1 \Rightarrow 0 \le kx < k$, then the integers in this interval are considered):

1. Generate *x* a real random number in the interval [0, 1)
2. *y = (int) (kx)* => ***y* will be integer random number between 0 and** k-1

Then, if an integer random number between *min* and *max is* needed, proceed as follows:

1. Generate *x* a random number between 0 and *max-min* (instead of *k* here we have *max - min* + 1)
2. *y = x + min* => ***y* will be integer random number between *min* and** max

For generating the *m* strings of *n* randomly chosen bits:

```
for i=1,m repeat *for each chromosome
    for j=1,n repeat *pt. each gene of the current chromosome
          generates an integer random number between 0 and 1
          gene j of chromosome i <- u
```

4. Parent selection

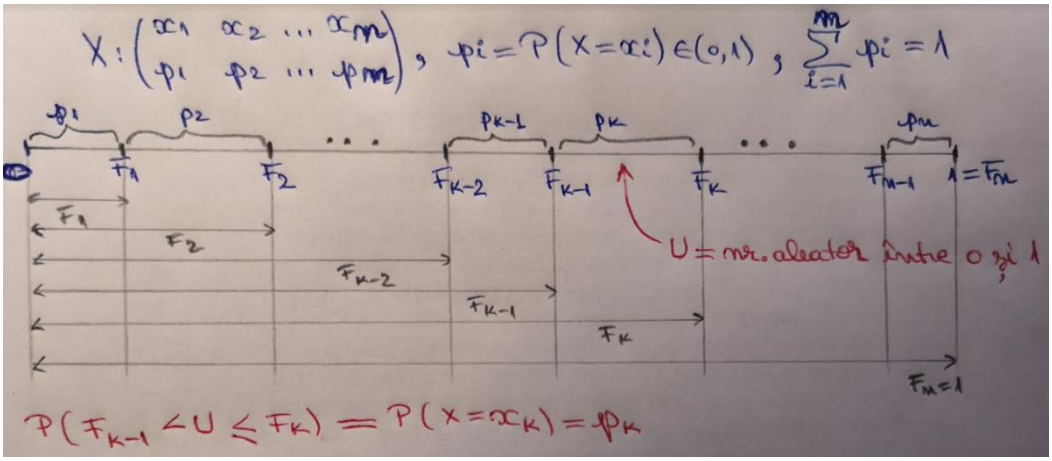
The purpose of selection is to **select** for survival **the fittest individuals** in the population in the hope that their offspring will have **higher fitness**. In combination with variations in the operators of crossing and mutation, selection must maintain a balance between exploration and exploitation. High selection pressure leads to low diversity in the population created from well-adapted but suboptimal individuals, which leads to a limitation of change and thus progress. On the other hand, low selection pressure slows down evolution.

Selection examples:

Wheel of fortune

This is the selection scheme introduced by Holland in the original genetic algorithm. The **probability of selection** of each individual in the current population is equal to **its fitness divided by the total fitness of the population**.

$$p_i = \frac{f(x[i]))}{\sum_{i=1}^{m} f(x[i]))}$$




```
EVAL. CURRENT POPULATION    for i=1,m
                              eval[i] <- f(x[i])
TOTAL FITNESS               T <- 0
                            for i=1,m
                              T <- T + eval[i]
PROB.SEL.INDIVIDUAL         for i=1,m
                              p[i] <- eval[i] / T
CUMULATIVE SALT PROB.       F[1] = p[1]
                            for i=2,m
                              F[i] = F[i-1] + p[i]
SELECTION                   generates U real random number between 0 and 1
                            k = 1
```

```
as long as U > F(k) repeats
    k = k+1
  *select for survival individual k
  *where F[k] < U <= F[k+1]
```

- **Rank-based selection** prevents the premature convergence that occurs when using fitness-proportional selection schemes. Individuals are ranked according to fitness value, and the probability of selection is proportional to the rank occupied. The selection pressure is in this way decreased if the fitness variance is high and is increased if the fitness variance is low.
- **The tournament selection** randomly chooses *k* individuals, and of these only the best *j* are selected for survival. The procedure is repeated until the desired number of individuals is obtained. It is the most efficient in terms of time complexity. In terms of selection pressure it is similar to rank-based selection.
- **Elitism** is an additional scheme to any selection mechanism whose purpose is to retain **the best individuals in each generation**. This can be achieved by a **descending sorting of individuals according to their fitness**.

5. Genetic operators

5.1 Crossover is the exchange of genetic information between two or more parent chromosomes. Crossover of parents occurs with a certain probability; if crossover does not occur, sons will be like their parents. The probability of inbreeding indicates how often inbreeding should occur in a population. It usually has a high value, between 60% and 90%.

5.1.1. Crossing **with a** randomly chosen **cutting point**:

```
01 001011  _\ 01 111100
10 111100  / 10 001011
```

For crossing with probability *pc* of two parent chromosomes p1 and *p2* to give two child chromosomes *f* 1 and *f* 2, one can proceed as follows:

```
f1 <- p1
f2 <- p2
generates a real random number between 0 and 1
if u<pc then
      *the position k of the cutting point is chosen randomly
      generates k integer random number between 1 and n-1
    for i=k,n repeat *gene after position k
          aux <- gene i of chromosome f1
          f1 chromosome i gene <- f2 chromosome i gene
          f2 chromosome i gene <- aux
```

5.1.2. Crossing **with *n*** randomly generated **cut points.** Example for 3 cut points:

```
01 0 010 11  _\ 01 1 010 00
10 1 111 00  / 10 0 111 11
```

5.1.3. **Uniform** crossover: for each locus, the gene of one parent is probabilistically selected.

11001011+11011101⇒11011111

5.1.4. **Arithmetic** crossover: bit-level arithmetic operations are performed to create new descendants

11001011+11011111⇒11001001 (AND)

5.2 Mutation = change (change of bits from 1 to 0 or from 0 to 1) of a randomly chosen gene(s).
For example: 001011000 -> 001011010 or

$$x_1x_2x_3x_4x_5x_6x_7x_8 \longrightarrow x_1x_2x_3x_4x_5x_6\overline{x_7}x_8$$

Mutation occurs mainly to avoid solutions falling into a local optimum. Performing this operation too many times will turn the algorithm into a random search in the space of possible solutions. Therefore, the mutation probability must be small, about 0,5-1%.

For mutation of a *c-chromosome* with probability *pm* this can be done:

```
for i=1,n repeat     *pt. each gene of chromosome c
    *random choice of gene position
    generates k integer random number between 1 and n
    generates a real random number between 0 and 1
    if u<pm then
      c-chromosome k gene <- 1 - c-chromosome k gene
```

6. Stopping the algorithm

The algorithm can stop when a maximum number of iterations is reached or when the population converges (no more significant changes are made in the current generation (or solution) compared to the previous one).