Nicolas GRIFFITHS SANCHEZ

Adrien THIBAULT

F21BC-Biollogically Inspired Computation

Coursework 1 – Blackbox Optimization

https://github.com/ndg2/BIO.git

Part A –1. Evolving a MLP

Overall approach

Function choice: In the first place we chose to approximate the sphere function and once the algorithm was working properly, we extended the approximation to other functions, such as the cardinal sine.

Programming language: We made a rapid consideration of how to implement the neural network. Since we would mostly be dealing with weights matrices and we both had experience with this, our chosen language was MATLAB.

Chromosomes: In the first version, we only optimized the weight matrices of the MLP. In the following versions, we included the bias.

MLP implementation

Since we were implementing a neural network for the first time, we made sure everything was working after we coded every part of the program. The main components of the program are:

Activation function: We chose $f = a * tanh(I * p)$, where I is the input and a and p parameters. These parameters are not optimised in the GA.

As the codomain of hyperbolic tangent is only $[-1, 1]$, we tuned the a factor for the last layer of the network. For the sphere function in 2 dimensions with input range $[-5,5]$ in every dimension, the codomain is $[0,50]$. We gave a the value 100 so that the codomain of the MLP would be $[-100,100]$. A lower value would have assumed too much information about the function, which is assumed to be unknown.

Bias: The bias is added to the sum of the outputs of the previous layer factored with the weights. Initially, we did not use it but in later versions we tuned it manually. This parameter plays an important role, so we decided to introduce it in the chromosome to be optimized. At first we used one bias value for the whole MLP and then we introduced one bias per layer, thus improving the results considerably.

Fitness: Using weight matrices, activation functions and biases, we can compute the output of the MLP. In order to train the MLP we need an estimator of the accuracy provided by the MLP. Let the continuous function be f_{cont} and its MLP approximation f_{MLP}. We chose $fitness = \frac{(\sum_i^{i \in inputs}(f_{MLP}(i) - f_{cont}(i))^2)}{nbPairs}$ as a fitness function. We used 50 to 300 input vectors in the domain [-5,5] to approximate the sphere function. The fitness function is based on the squared error instead of its absolute value to penalize single values with great errors rather than a number of values with small errors. In this way, we obtain better approximations over the entire domain.

GA implementation strategy

Firstly, only mutations were used. The crossover was introduced in the following version, creating more complex operators and selection methods.

Indicators for GA tuning

GA optimization tends to produce fitter individuals in each generation, but the goal is to find the set of parameters that produced the fittest individual possible in the shortest time. We developed several indicators in order to follow the GA during the optimization process and analyse the final results.

Here is a typical display while the optimization is running:

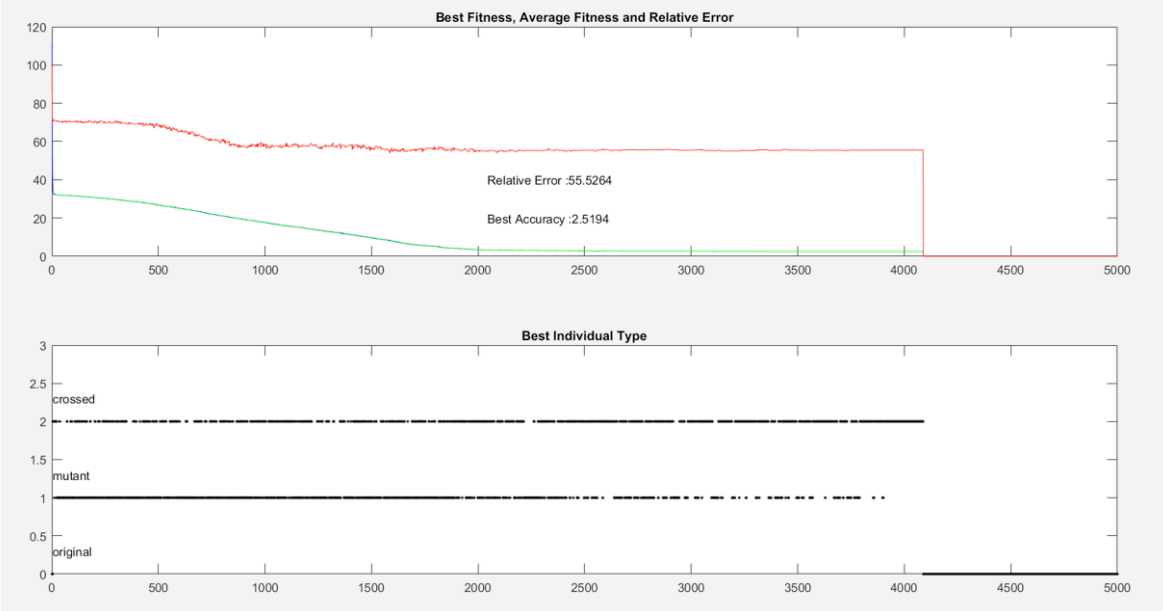


Figure 1. Indicators during the optimization of the cardinal Sine function in 2 dimensions on [-5,5]

The first graph shows the highest accuracy in the population in green, which is calculated as:

$$accuracy = \frac{1}{range_{max}} \times \frac{\sum_i^{i \in inputs} abs(f_{MLP}(i) - f_{cont}(i))}{nbPairs}$$

The relative error is calculated as:

$$relative\ error = 100 \times \sum_i^{i \in inputs} min\left(1, abs\left(\frac{f_{MLP}(i) - f_{cont}(i)}{f_{cont}(i)}\right)\right)$$

The second graph is used to tune the ratio crossover/mutation.

After the optimization is completed, another figure is displayed:

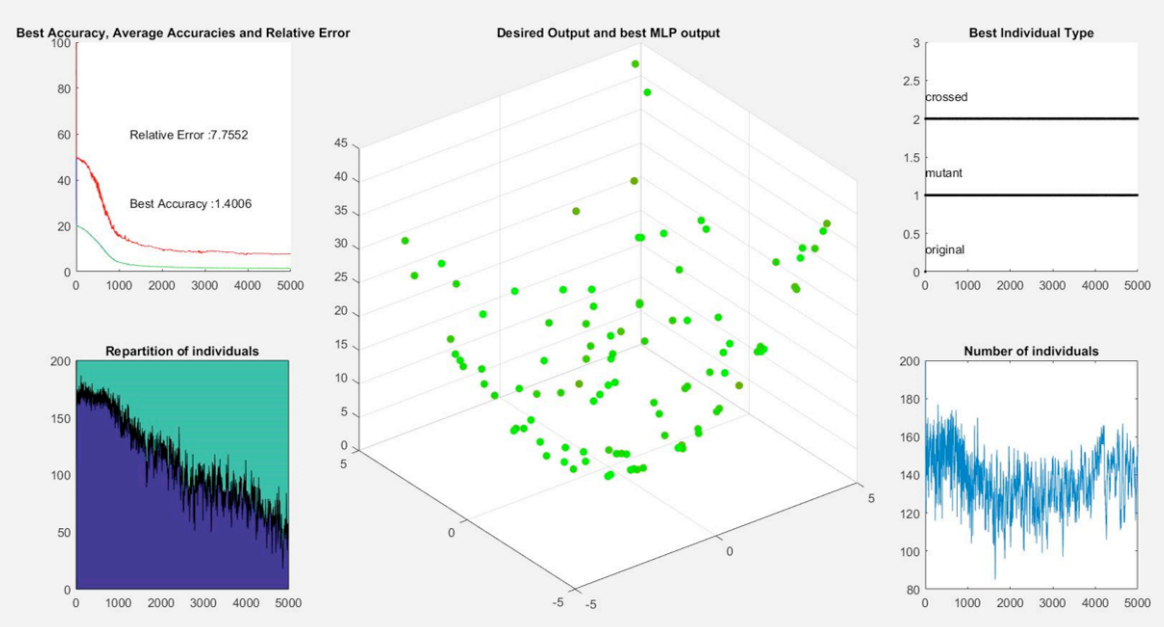


Figure 2. Indicators after the process of optimization of the sphere function in 2 dimensions on [-5,5]

The graph in the bottom left corner is used to tune the mutation/crossover ratio.

The graph in the bottom right corner shows the diversity in the population, serving as a guide when creating operators to avoid cloning individuals.

The graph in the centre represents the sampled continuous function to approximate. The colour of the dots, from **green** to **red**, represents the accuracy of the MLP approximation. As the GA algorithm performed better and this graph became less interesting, another display was used to identify the regions with the lowest accuracy:

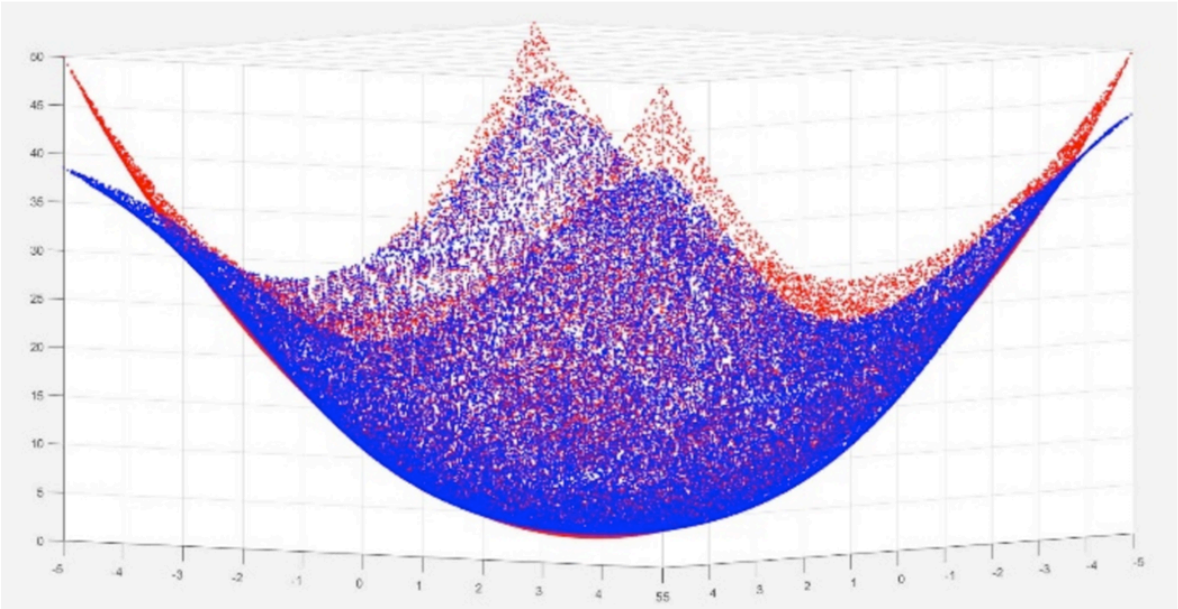


Figure 3. Sphere function (in red) and it MLP approximation (in blue) on [-5,5]

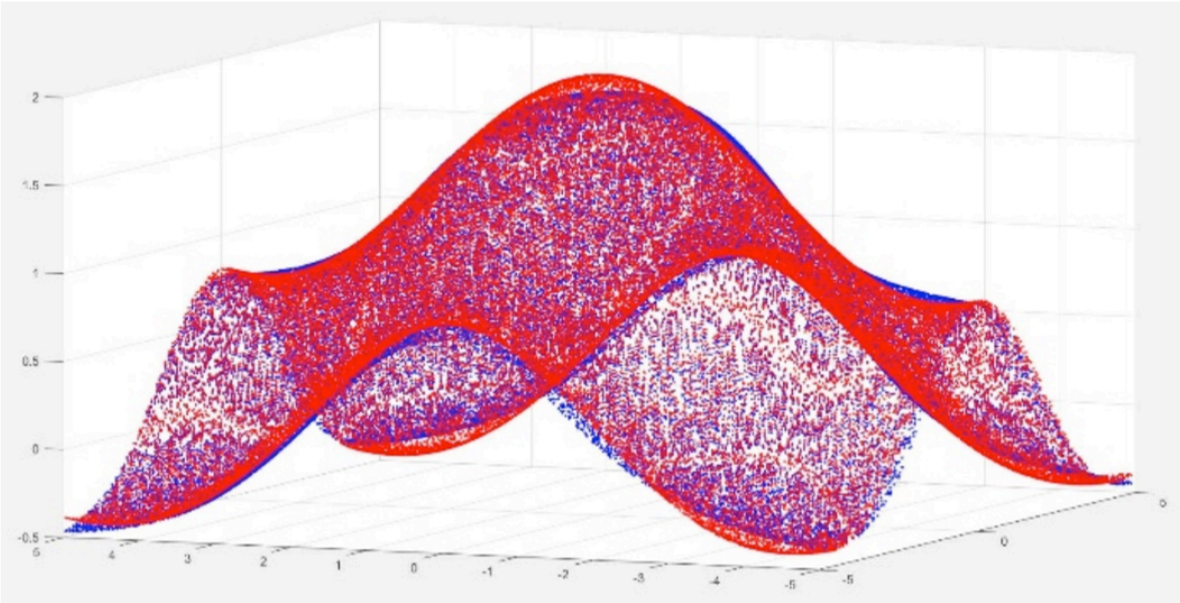


Figure 4. Cardinal Sine function (in red) and it MLP approximation (in blue) on [-5,5]

NOTE: For this graph, the number of samples (~50.000) is greater than the number of input-output pairs used.

Selection methods

At first, we selected the individuals that would lead to the next generation according to their fitness (the best 30%). The next version ruled out the very best ones to keep diversity (selecting the best 5% to 35%). The final version uses rank-based tournament selection, leading to the best results.

Crossover

Firstly, the children were copies of one of the parents, except for one of the weight matrices, which was taken from the other parent. This method created clones, so we added low-value random noise to the matrix from the second parent. The chosen matrix was selected randomly, so all matrices had the same chance to be selected. Since we observed that some matrices were more important than others, we added probabilities of being chosen for each matrix, initially with the same value, which were also optimized throughout the optimization.

Mutating only the weights was not sufficient, so we introduced the biases in the chromosome, one per layer. During the crossover, only the biases or the weight are optimized, with the same probability.

Mutation

For mutations, only the bias or the weights is modified. This is performed by taking an individual and adding random noise to one of the biases or to one of the columns of the weight matrices. The range of the noise depends on the rank of the individual (good, medium or bad). The worse the parent is, the greater the range of noise will be.

Results

We now present averaged results for several tests with the same parameter configurations:

Function	Sphere	Sphere	Sphere	Sphere	Sphere	Sphere	Sphere
Dimension	1	1	1	1	2	2	2
Range	[-1,1]	[-1,1]	[-5,5]	[-5,5]	[-1,1]	[-1,1]	[-1,1]
Layers	3	3	3	3	5	5	5
Total number of neurons	50	50	40	40	100	150	150
Weight Optimization	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bias Optimization	No	No	No	Yes	No	No	Yes
Number of pairs	50	50	50	50	100	200	100
Generations	400	700	500	500	1000	1000	1000
Population	100	100	100	100	150	150	150
Type Selection	Best	Best	Best	Good	Good	Good	Good
Crossover/Mutation	50/50	50/50	50/50	50/50	40/60	30/70	40/60
Type Crossover	Matrix switch	Matrix switch	Matrix switch	Column switch	Matrix switch	Matrix switch	Matrix switch
Type Mutation	Random matrix	Random column	Random column	Random column	Random column	Random column	Random column
Best Accuracy	6%	2.5%	8%	1%	23%	30%	13%

Function	Sphere	Sphere	Sphere	Sphere	Sinc	Sinc	Sinc
Dimension	2	2	2	2	2	2	2
Range	[-1,1]	[-5,5]	[-5,5]	[-5,5]	[-5,5]	[-5,5]	[-5,5]
Layers	3	5	4	4	7	10	15
Total number of neurons	90	100	100	150	150	150	150
Weight Optimization	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bias Optimization	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Number of pairs	100	200	150	150	300	300	300
Generations	1000	5000	5000	5000	5000	5000	5000
Population	150	150	200	200	100	100	100
Type Selection	Good	Rank-based	Rank-based	Rank-based	Rank-based	Rank-based	Rank-based
Crossover/Mutation	40/60	10/90	10/90	5/95	5/95	5/95	5/95
Type Crossover	Matrix switch	Matrix switch	Matrix switch	Matrix switch	Matrix switch	Matrix switch	Matrix switch
Type Mutation	Random column	Random column	Random column	Random column	Random column	Random column	Random column
Best Accuracy	13%	8%	6%	2%	12%	7%	3%

The results illustrate important lessons for the optimization strategy:

- Bias optimization is necessary. The lack of bias produces symmetry-related issues.
- Rank-based selection obtains the best results for the tested functions.
- For the tested functions and operators, the best results were obtained relying more on mutation than crossover.
- Good accuracy is achieved faster by mutating an entire matrix, but in order to obtain very good accuracy, column mutation is needed.
- For the same number of neurons, architectures with more layers perform better. The training takes longer, but it is more accurate.

Figures:

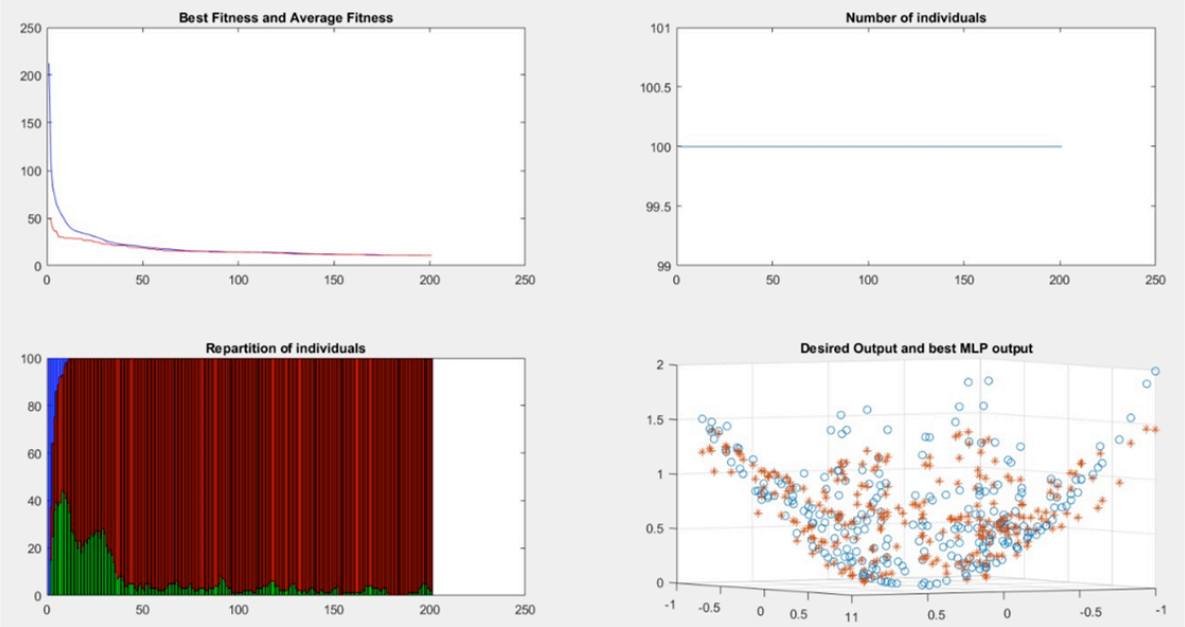


Figure 5. Optimization of the sphere function on [-1,1] with good individual selection

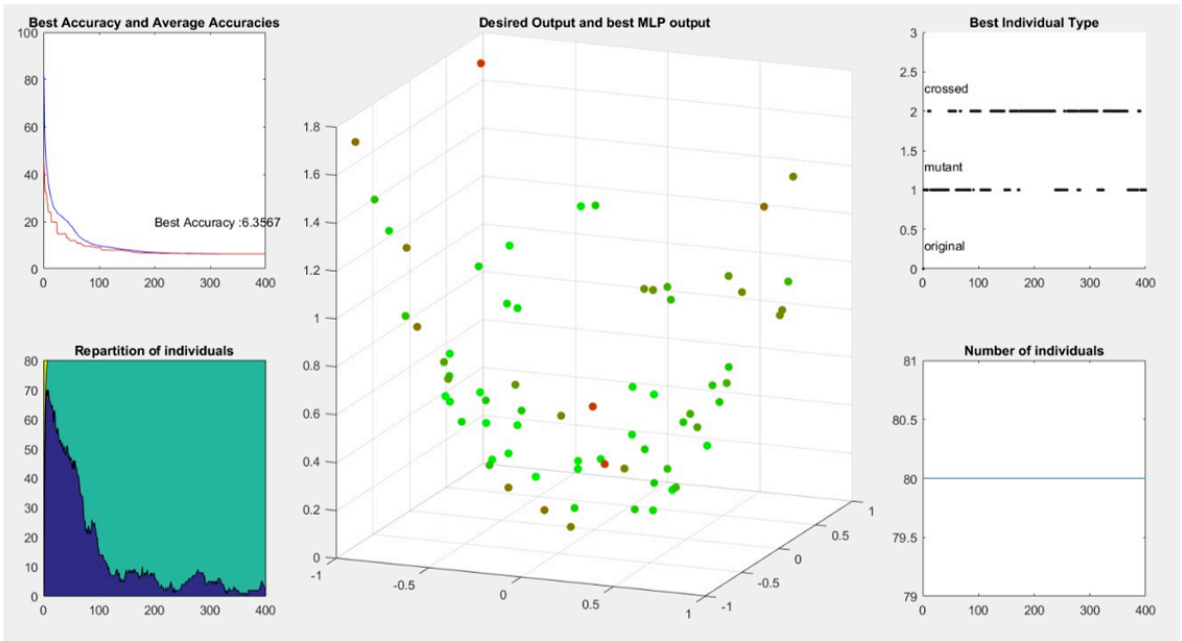


Figure 6. Optimization of the sphere function on [-1,1] with rank-based selection and matrix mutation

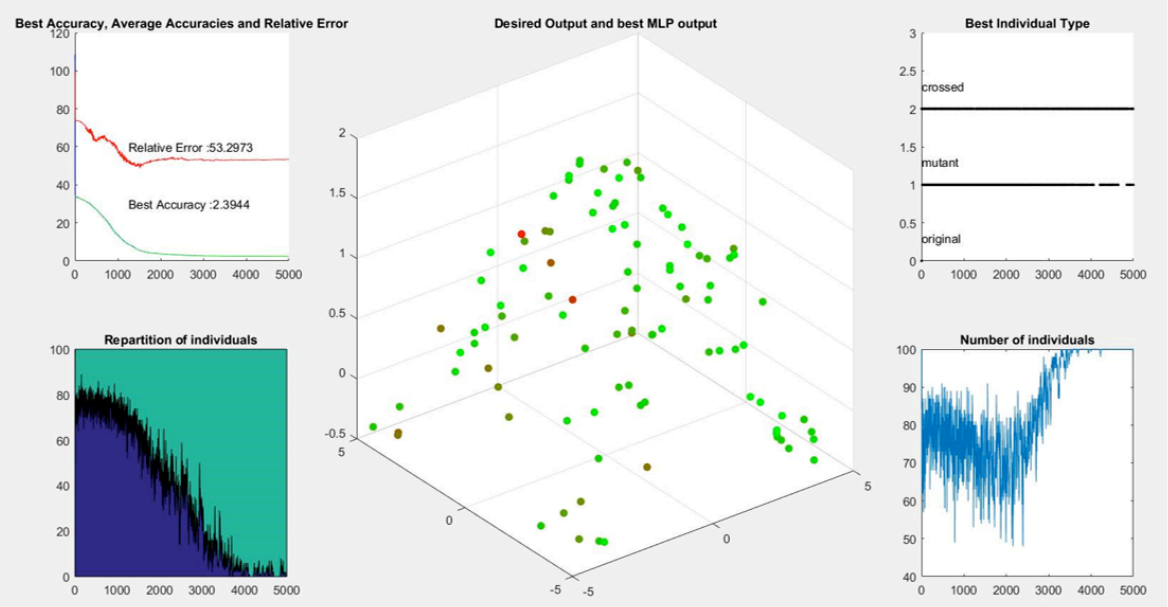


Figure 7. Optimization of the Cardinal Sine function on [-5,5] with best settings

Part A –2. Optimizing the MLP approximation

The optimization algorithm utilised will be further explained in part B.

In this case, the chromosome is formed by inputs to the network, which are trying to be optimized in order to minimize the output.

Assuming that the network approximates the function correctly, optimizing the approximation should be equivalent to optimizing the real function. This strategy is used when evaluating the real function is more computationally expensive than the approximation or when the function is unknown and only input-output pairs are available.

Mutation and crossovers operators are more intuitive in this case. Crossover is performed by calculating the average of the inputs of the parents and mutation by adding noise to the existing inputs. As we are working on noiseless *continuous* functions, these simple operators lead to good accuracy.

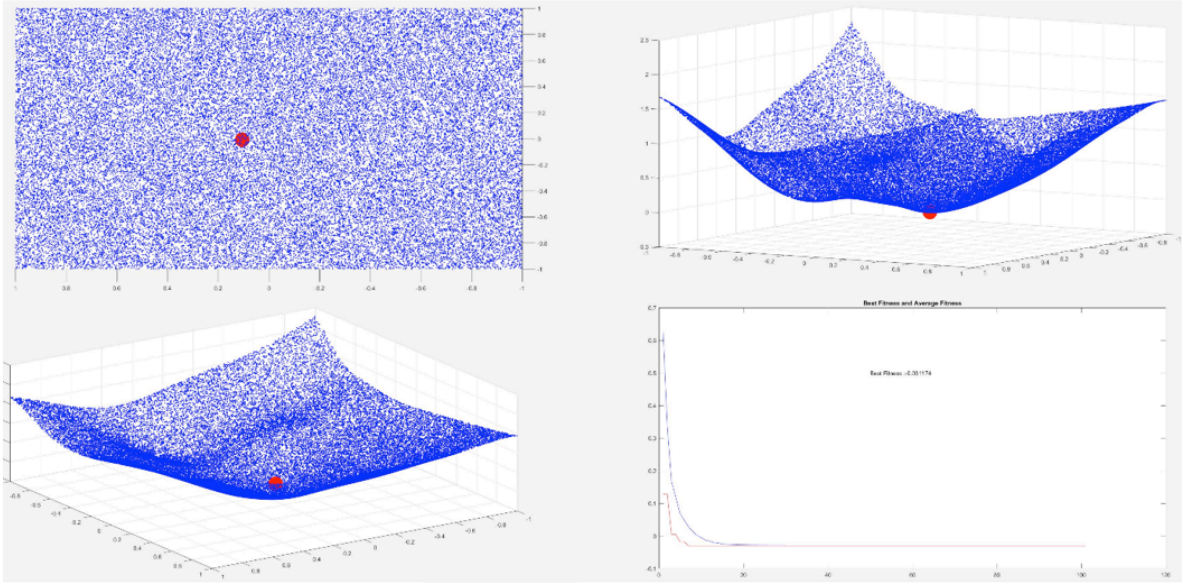


Figure 5. Optimum search on sphere function approximation

As we can see in the graph above, the optimization algorithm finds a good solution in a short time. Nevertheless, given that the approximation is not very accurate, the solution is not exactly (0,0) as it would be for the real function.

Part B – EP optimization algorithm

Overview

The algorithm tries to minimize a function from which only the dimension and the range of the input are known. It performs the steps described below:

1. **Initial population creation**: Each individual is represented by an input vector, which is created randomly in the range [-5,5].

2. **Population fitness**: As the goal is to minimize a function, the fitness value is calculated evaluating the function for each individual. The lower the value returned by the function evaluation, the better the fitness value is. The population is then ranked in order of fitness.

3. **Selection of individuals**: A subset of the population is chosen, from which new possible members of the population will be created. The selection method can be different in each case.

4. **Mutation**: The individuals in the subset are modified according to the mutation operator. No crossover is performed, following the coursework guidelines.

5. **Fitness analysis and ranking of mutated individuals**: The fitness of each mutated individual is analysed and ranked.

6. **New generation**: The mutated individuals are compared to the current members of the population. If there are members in the population with worse fitness values, these are replaced.

7. **Loop/End**: The best individual in the population is selected. The fitness value of this individual is compared to the target value and if it has been reached, the algorithm ends with this individual as a solution. If the condition is not fulfilled, steps 4-8 are repeated until this occurs or until the maximum number of generations is reached.

Parameters

Population size: Number of individuals in the population. If it is too small, the search space will not be explored appropriately; if too large, the algorithm will be slow.

Maximum number of generations: When it is reached, the algorithm ends. This happens when it is able to achieve the target value. If the target value is very ambitious, this parameter can affect how close the provided solution becomes to the target.

Number of individuals in subset: Number of individuals that form the population subset that will be mutated.

Selection Method: In the first version, the individuals with better fitness are selected. This is modified in the subsequent versions and will be later explained.

Mutation operator: This is a very important parameter in order to obtain good diversity. In the first version, a random component of the individual is selected and then substituted by a random number. This is modified in the following versions and will be later explained.

Mutation rate: In this case, as there is no crossover operator, every individual in the selected subset is mutated. The size of the subset acts as the mutation rate parameter.

Local tests

The algorithm is tuned using COCO's functions and independent locally developed tests. The local test returns two graphs, showing the error between the fitness of the best individual and the target, and the relative error. It also outputs the average of the total and relative error.

The functions used for all tests are 1, 14, 16, 19 and 21 in COCO.

Results are averaged for 10 tests and 15 function instances for a maximum of 100 generations.

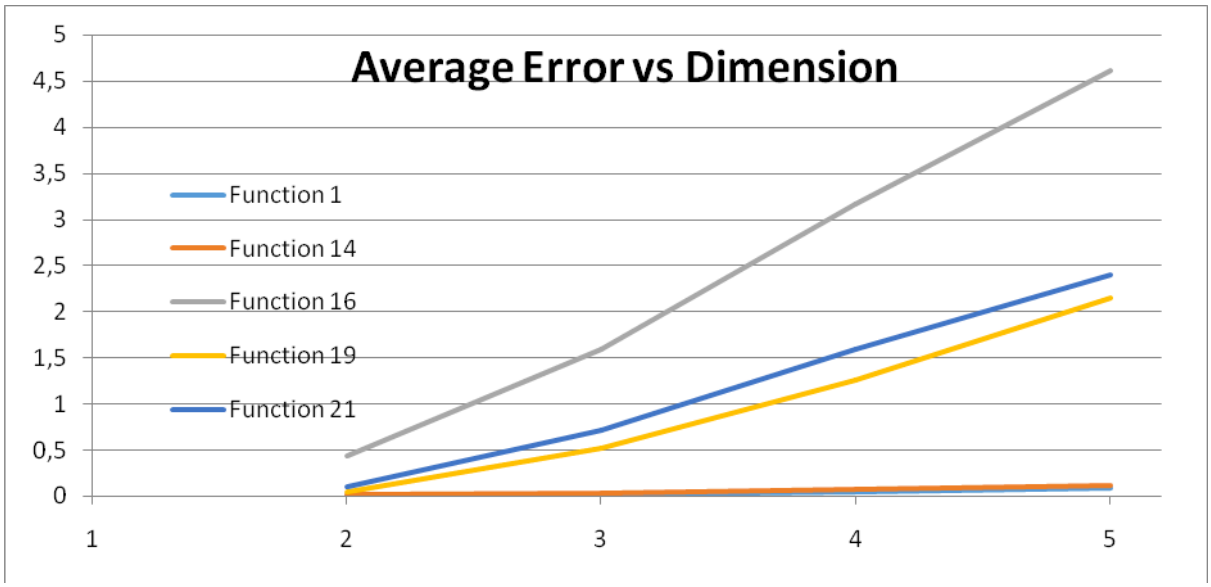
Version 1

Population size: 50

Selected subset size: 5

Selection method: Best fitness

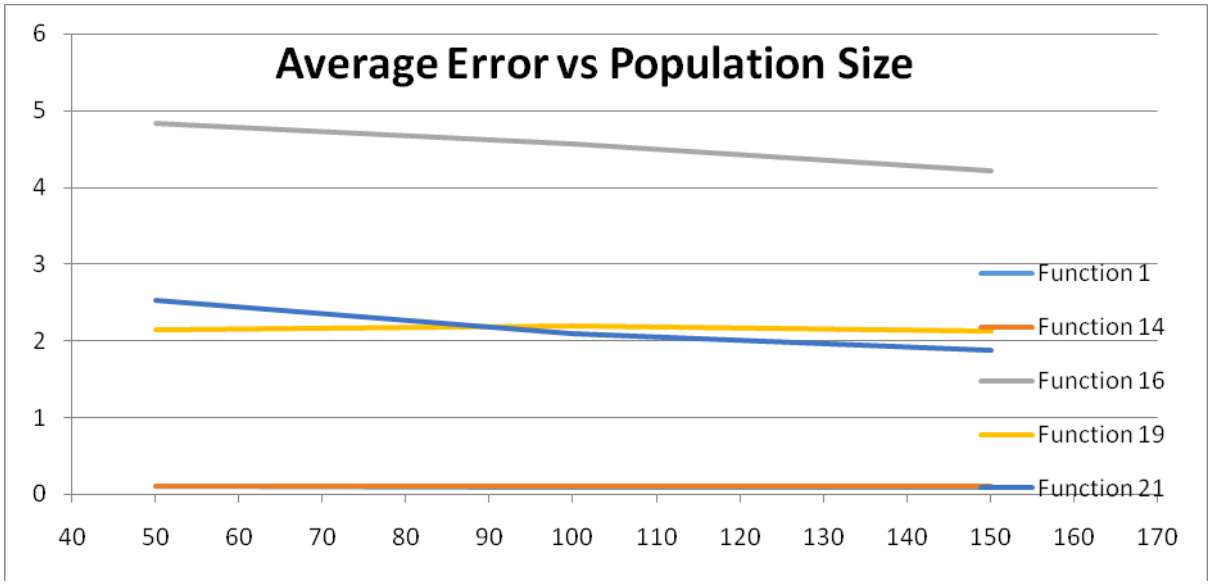
Mutation: Random number in random component (RCRV)



The graph shows that the error increases rapidly with the number of dimensions. This makes sense, since the search space also increases. For the following versions, 5 dimensions will be used to show the effects of the parameters more clearly.

Version 2: Population size

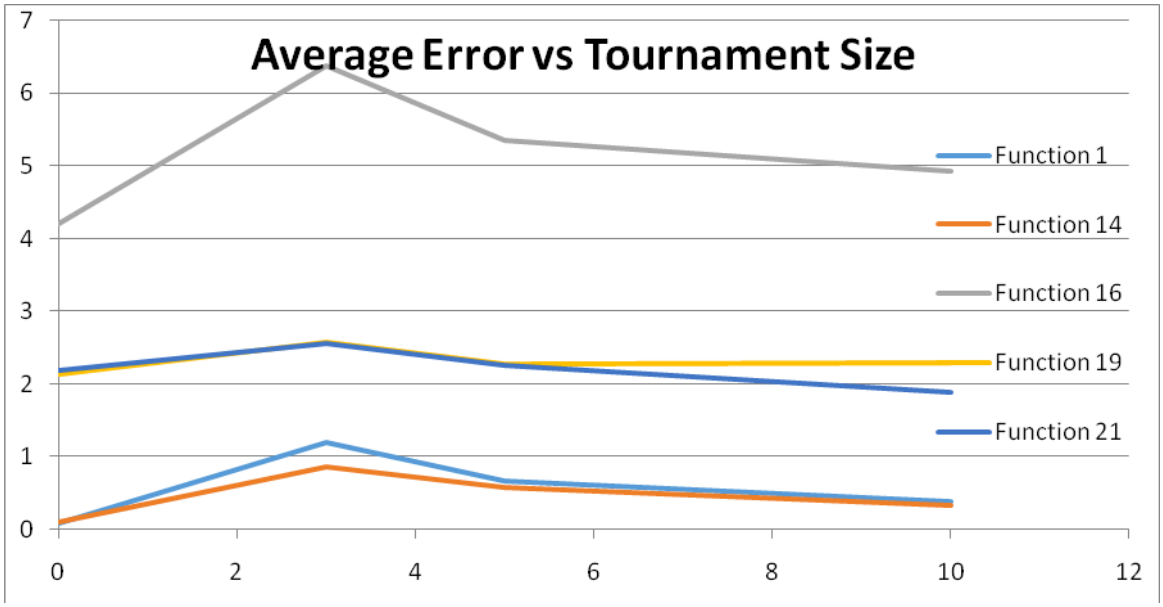
Larger populations allow the search space to be explored in greater depth. If the population is too big, the computation time might be too long.



This graph shows that increasing the population size increases the accuracy. Although the improvement is subtle, a larger population is useful in order to have diversity from the beginning of the search. For the following versions, the population size is 150.

Version 3: Selection method

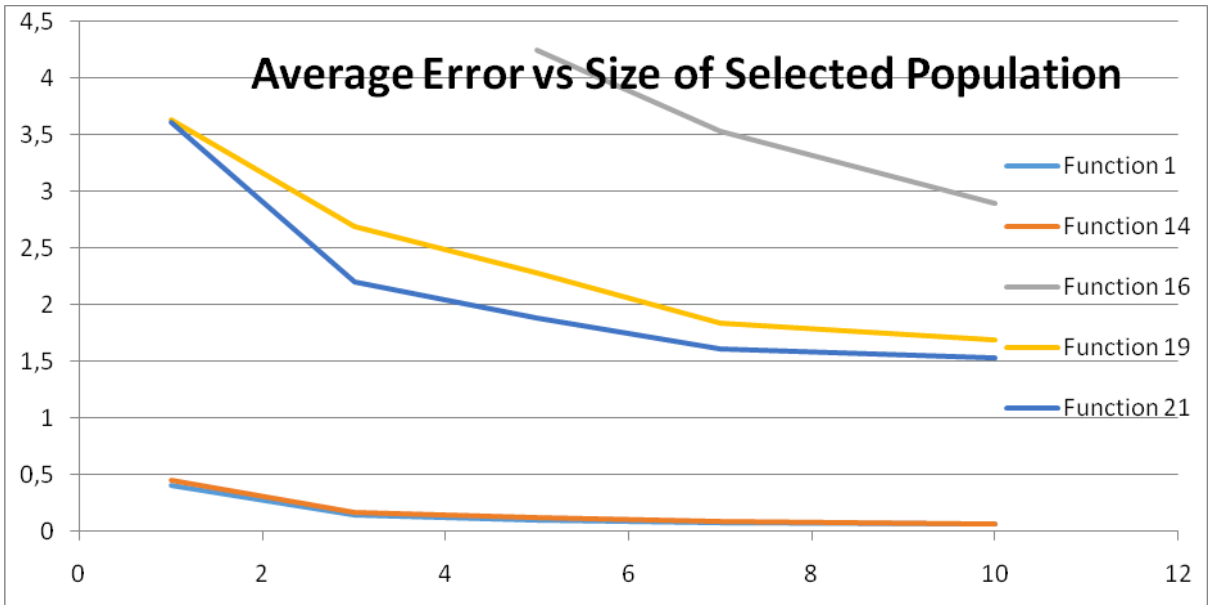
The initial selection method is not always effective because it can reduce the diversity. Another approach is tournament selection, which consists of selecting *t* random individuals from the population and keeping only the best. This is repeated until the number of kept individuals matches the desired number of selected individuals. This method requires more time but sometimes can provide better performance.



In the above graph, the previous selection method is represented with *t*=0 to be able to compare it to tournament selection. The results show that tournament selection does not improve the accuracy. It is worthy of note that this depends on the function that is optimised. Some functions require more diversity and others converge faster with less diversity. This makes the optimal

selection method different for each function. As the tournament size increases, the accuracy improves, but the computation time becomes too long. For the following versions, the selection method will remain unchanged.

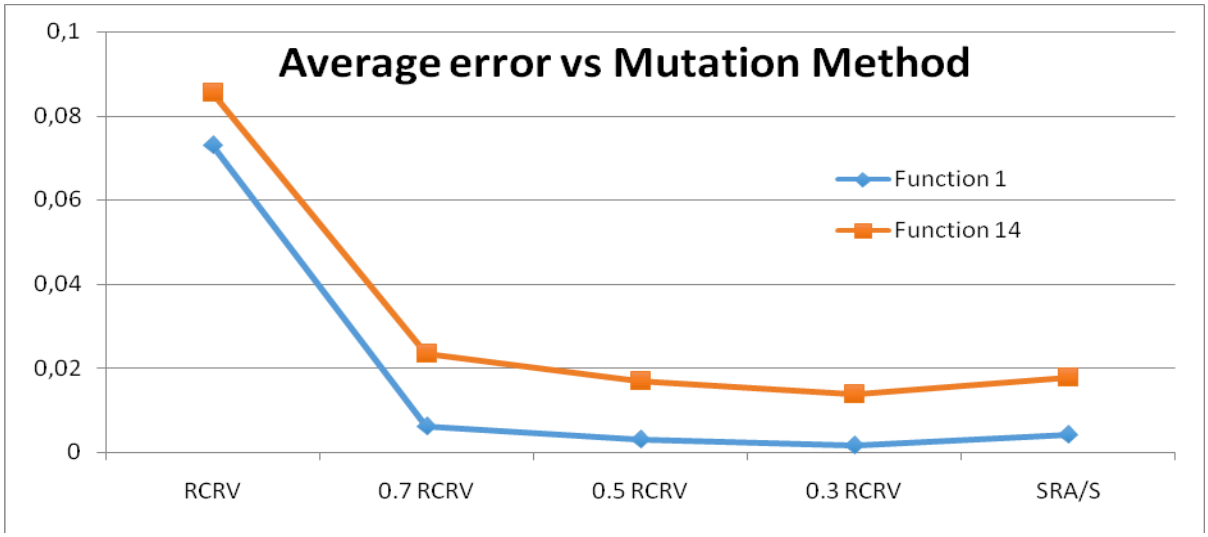
Version 4: Selected population size

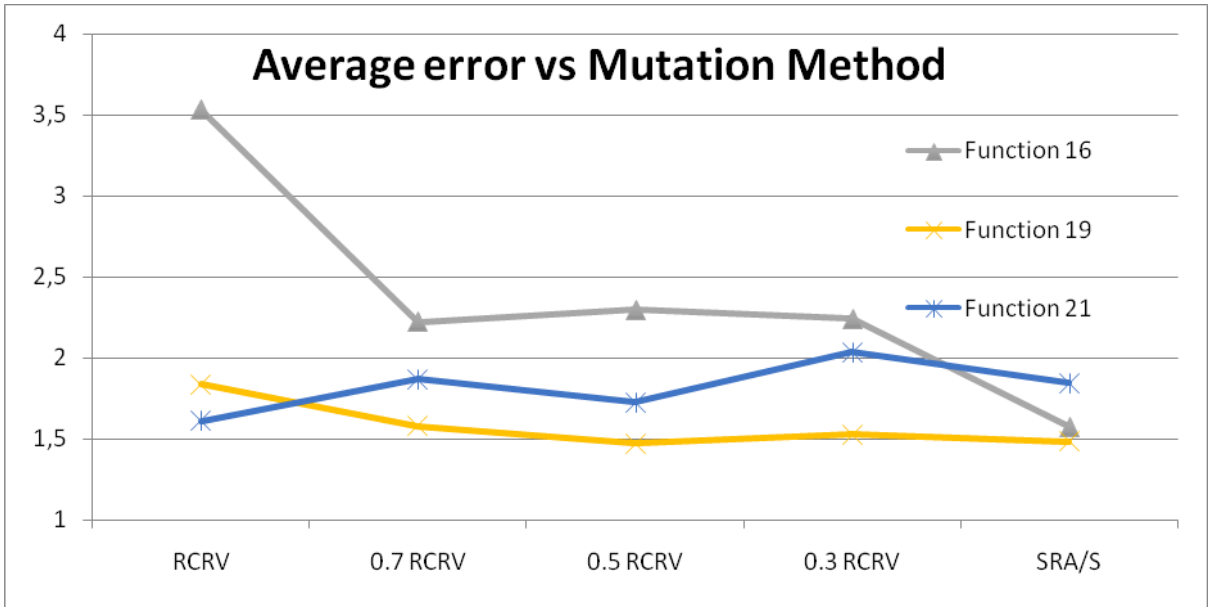


This graph shows that the error decreases with the size of the selected population. This is caused by a faster exploration of the search space, due to the higher number of mutations performed per generation. On the other hand, since after each mutation the fitness must be re-evaluated, the number of function evaluations and the computation time increase (trade-off accuracy vs. speed). For the following versions the selected population size is 7.

Version 5: Mutation operator

The new mutation operator (SRA/S) adds/subtracts a small number from a random component of the individual. This method is combined with the previous one with different probabilities.





These graphs show the accuracy of the algorithm for RCRV, SRA/S and combinations of both of them. The best mutation is highly dependent on the function being optimized. A good choice for obtaining good performance across all the functions is to use 50% RCRV and 50% SRA/S.

Final Version Parameters

Population Size: 150

Selected Population Size: 7

Selection: Best fitness

Mutation: 50% RCRV - 50% SRA/S (0.2 maximum)

References

[1] P. Vargas and M. Vallejo, *Lecture Notes for Bio-inspired Computation*, 2016.

[2] S. Haykin, *Neural Networks and Learning Machines,* 3rd ed., Prentice Hall 2009.

[3] M. Obitko, *Introduction to Genetic Algorithms.* Available: http://www.obitko.com/tutorials/genetic-algorithms/dna-pictures.php