BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

EVOLUTIONARY DESIGN OF ULTRASOUND TREATMENT PLANS

EVOLUČNÍ NÁVRH ULTRAZVUKOVÝCH OPERAČNÍCH PLÁNŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. MÁRIA MASÁROVÁ

SUPERVISOR
VEDOUCÍ PRÁCE

doc. Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2021

Department of Computer Systems (DCSY) Academic year 2020/2021

Master's Thesis Specification

23930

Student:	**Masárová Mária, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Bioinformatics and Biocomputing
Title:	**Evolutionary Design of Ultrasound Treatment Plans**
Category:	Artificial Intelligence

Assignment:

1. Get acquainted with the evolutionary design of ultrasound treatment plans. Focus mainly on physical models used in the k-Wave toolbox.
2. Familiarize yourself with commonly used evolutionary algorithms and their efficiency in terms of the population size and the number of evaluations needed.
3. Create a set of suitable testing treatment cases inspired by clinical practice.
4. Design a set of criteria for evaluation of the evolutionary algorithm efficiency.
5. Design and implement a suitable encoding and fitness function maximizing the outcome of the treatment plan.
6. Statistically evaluate the efficiency of selected evolutionary algorithms and the quality of developed treatment plans.
7. Discuss about the achieved results and their contribution for clinical practice.

Recommended literature:

- According to supervisor's advice.

Requirements for the semestral defence:

- Items 1 to 4 of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Jaroš Jiří, doc. Ing., Ph.D.**
Head of Department:	Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work:	November 1, 2020
Submission deadline:	May 19, 2021
Approval date:	April 16, 2021

Abstract

The use of focused ultrasound helps save and facilitate human lives, as it can be used to treat epilepsy, destroy cancer cells and stop internal bleeding in a non-invasive way, which is a more acceptable and safer solution for humans. Given that human safety and health is a priority in the treatment of serious diseases, this work deals with the comparison of different evolutionary algorithms and their use in the design of evolutionary ultrasound treatment plans. Two types of media are used in this work, namely homogeneous medium and heterogeneous medium. When evaluating algorithms, we focus on efficiency with respect to the size of the population, number of fitness function evaluations and computational time. In a homogeneous medium, CMA-ES proved to be the best algorithm, which was able to find the optimal solution with a 100% coverage of the target area for a rotated grain of rice within 20 seconds. However, the heterogeneous medium is a much more complex problem, mainly due to the skull, which reflects and absorbs much of the ultrasound. Here, the SA algorithm proved to be the best, finding the result with a 23% coverage of the target area in the first test scenario. The calculation time took about 1 hour and 18 minutes, which means that time is a very prohibitive factor.

Abstrakt

Použitie zameraného ultrazvuku pomáha zachraňovať a uľahčovať ľudské životy, nakoľko práve jeho využitím môžeme liečiť epilepsiu, ničiť rakovinové bunky a zastavovať vnútorné krvácanie neinvazívnou cestou, ktorá predstavuje pre človeka prijateľnejšie a bezpečnejšie riešenie. Vzhľadom na to, že bezpečnosť a zdravie človeka je prioritou pri liečení závažných ochorení, sa táto práca zaoberá porovnaním rôznych evolučných algoritmov a ich použitím pri návrhu evolučných ultrazvukových operačných plánov. V práci sa využívajú dva typy médií, a to homogénne médium a heterogénne médium. Pri vyhodnocovaní algoritmov sa zameriavame na efektivitu s ohľadom na veľkosť populácie, počet evaluácií fitness funkcie a výpočetný čas. V homogénnom médiu sa ako najlepší algoritmus ukázal CMA-ES, ktorý v priebehu 20 sekúnd dokázal nájsť optimálne riešenie so 100% pokrytím cieľovej oblasti pre rotované zrnko ryže. Heterogénne médium je ale oveľa zložitejší problém, predovšetkým kvôli lebke, ktorá odráža a pohlcuje veľkú časť ultrazvuku. Tu sa ako najlepší preukázal algoritmus SA, ktorý našiel výsledok s 23% pokrytím cieľovej oblasti v prvom testovacom scenári. Doba výpočtu trvala približne 1 hodinu a 18 minút, čo značí že je čas veľmi znemožňujúci ("drahý") faktor.

Keywords

genetic algorithm, CMA-ES, simulated annealing, optimization, HIFU, LIFU, transducer

Kľúčové slová

genetický algoritmus, CMA-ES, simulované žíhanie, optimalizácia, HIFU, LIFU, vysielač

Reference

MASÁROVÁ, Mária. *Evolutionary Design of Ultrasound Treatment Plans.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Jiří Jaroš, Ph.D.

Rozšírený abstrakt

Problém, ktorým sa zaoberá táto diplomová práca môžeme formulovať do otázky: *Kam treba umiestniť vysielač tak, aby ultrazvuk, ktorý z neho vychádza zasiahol cieľ, respektíve aby zasiahol čo najväčšiu časť cieľa?* Inými slovami, snažíme sa nájsť čo najlepšiu pozíciu ultrazvukového vysielača.

Tento problém je riešený pomocou optimalizácie, a to konkrétne pomocou troch rôznych evolučných algoritmov. Týmito algoritmami sú: genetický algoritmus, CMA-ES a simulované žíhanie. Algoritmy optimalizujú päť hodnôt, ktoré reprezentujú x-ovú a y-ovú súradnicu pozície vysielača, x-ovú a y-ovú súradnicu smeru lúča a zakrivenie vysielača. Vysielač vysiela ultrazvuk do všetkých smerov. Preto je potrebné, aby bol jemne zakrivený a tým došlo k stretu ultrazvukových vĺn, ktoré z neho boli vyslané. V mieste, v ktorom sa ultrazvukové vlny stretnú dochádza k zvýšeniu tlaku a nazývame ho ohniskom. Vysielač je ďalej definovaný aj šiestou hodnotou, ktorou je priemer clony. Táto hodnota ale nie je vstupným parametrom optimalizácie, nakoľko je predom určená a je rovná 4,1 cm.

Optimalizácia pozície vysielača prebiehala na dvoch rôznych médiach, homogénnom a heterogénnom médiu. Za homogénne médium považujeme objekt/látku/zmes, ktorá má v každej svojej časti tie isté vlastnosti. Pri simulácii ultrazvuku týmito vlastnosťami myslíme rýchlosť zvuku, hustotu a absorbčné koeficienty. Heterogénne médium je médium, ktoré sa skladá z viacerých objektov/látok/zmesí a každá z nich má definované svoje vlastné hodnoty. V tejto práci je heterogénnym médiom hlava, ktorá je ponorená vo vode. Toto heterogénne médium obsahuje vodu, mäkké tkanivo (kožu a mozog) a lebku. Každé z nich má inú rýchlosť zvuku a hustotu.

Na implementáciu bol použitý programovací jazyk MATLAB. MATLAB je programovací jazyk a prostredie so zameraním na numerické výpočty, modelovanie, návrh algoritmov, analýzu údajov a iné. Obsahuje tiež Global Optimization Toolbox, čo je balík, ktorý poskytuje funkcie na nájdenie globálnych riešení. Z thoto balíka sa využíva genetický algoritmus a simulované žíhanie. Algoritmus pre CMA-ES bol prevzatý z internetu[18] a upravený pre použitie na optimalizáciu pozície vysielača. Na simulovanie propagácie ultrazvuku boli použité funkcie z balíčka k-Wave. k-Wave je voľne dostupný balíček na prácu s akustikou pre MATLAB a C++.

Experimenty s homogénnym médiom prebiehali na troch rôznych benchmarkoch, a to zrnko ryže, pootočené zrnko ryže a pootočené zrnko ryže s penalizáciou. Simulácia ultrazvuku bola vykonaná pomocou funkcie `acousticFieldPropagator` z balíčka k-Wave. Najlepšie výsledky dosahoval algoritmus CMA-ES s nastavením parametrov: počet iterácií = 20, veľkosť populácie = 54, počet rodičov = 27, sigma0 = 38.1, ktorý dokázal veľa krát nájsť riešenie s fitness hodnotou 0 (pri minimalizácii najlepšie riešenie).

Experimenty s heterogénnym médiom prebiehali na troch rôznych scenároch, a to scenár, kedy sa snažíme trafiť cieľovú oblasť pomocou jednej sonikácie, scenár kedy sa snažíme zasiahnuť cieľovú oblasť pomocou štyroch sonikácií a scenár, kedy sa snažíme pomocou jednej sonikácie zasiahnuť cieľ ale netrafiť pri tom penalizačnú časť. Ďalšie experimenty, ktoré boli vykonané pozostávali z väčšieho vysielača, penalizačnej masky okolo celej cieľovej oblasti, lepšieho počiatočného jedinca a nižšieho prahu tlaku. Na simuláciu propagácie ultrazvuku bola využitá funkcia `kspaceFirstOrder2D` z balíčka k-Wave. Nakoľko boli výpočty s heterogénnym médiom časovo náročné, vykonávali sa na clusti Barbora. Najlepšie výsledky dosahoval algoritmus SA (simulované žíhanie) s nastavením parametrov: počet iterácií = 270, počiatočná teplota = 1000, chladiaci krok = $teplota * 0.95^k$, kde k značí aktuálnu iteráciu. Najlepšie nájdené riešenie malo hodnotu fitness 58.75. Jeden test pre simulované žíhanie s týmto nastavením parametrov trval približne 1 hodinu a 20 minút.

Evolutionary Design of Ultrasound Treatment Plans

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of doc. Ing. Jiří Jaroš, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Mária Masárová
May 15, 2021

Acknowledgements

I would like to thank doc. Ing. Jiří Jaroš, Ph.D. for his valuable advice and professional help in writing this master thesis. I would also like to thank prof. Bradley Treeby for examples of ultrasound propagation simulation and Ing. Jakub Chlebík, who always answered any question I had. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).

Contents

Chapter 1

Introduction

With the popularization and development of science, it has been easier in recent years to connect knowledge from all scientific, development and research areas. Such an example is the use of computer science in medicine, to solve complex calculations to refine treatment methods. As it is natural for humans that everyone is afraid of surgery as an invasive intervention in the body, we try to invent and use methods that are non-invasive and painless. One such method is the Focused ultrasound, which is an early-stage, non-invasive therapeutic technology that uses an ultrasonic beam to try to remove a tumour, suppress the symptoms of epilepsy, stop internal bleeding, etc.

The aim of this work is to compare a few evolutionary algorithms and find the best one for the evolutionary design of ultrasound treatment plans. No free lunch theorem proves that there is no general solution and that different search algorithms may obtain different results, but overall problems, they are indistinguishable. It means that if an algorithm achieves superior results on some problems, it must pay with inferior results on other problems. In this sense, there is no free lunch in search, so it means there is no free lunch in optimization [35, 36].

In order to find the best algorithm for our problem, we need to define criteria according to which we will evaluate the effectiveness of the algorithms. These criteria are mainly the size of the population, the number of evaluations of the fitness function and computation time.

The evolutionary algorithms that were used are described in Chapter 2. These algorithms are genetic algorithm, CMA-ES and simulated annealing. The solved problem, optimization of the transducer position, is described in Chapter 3. This chapter explains what an optimization problem is and how we can solve it, how focused ultrasound works and what parameters of the transmitter we need to optimize. It further describes the tools used in the simulation of ultrasound propagation, the difference between homogeneous and heterogeneous media and how we calculate fitness function. Chapter 4 then describes the implementation of the algorithms and the fitness function, which differs depending on the used medium. Chapter 5 covers sets of experiments that were performed with a homogeneous medium to determine the efficiency of individual algorithms. Experiments with a heterogeneous medium are described in Chapter 6. Last is Chapter 7, which covers all acquired knowledge and achieved results.

Chapter 2

Evolutionary algorithms

Evolutionary algorithms (EAs) belong to a set of modern heuristics-based search methods which use simulated evolution to explore the solutions for complex problems. EAs use techniques that mimic the evolutionary processes inspired by biology, genetics and evolution to constitute search and optimization procedures. These techniques include inheritance, natural selection, crossover and mutation. Evolutionary algorithms are based on the Darwin's theory of evolution which says: *Individuals who adapt better to the environment have a higher chance to survive and thus to produce more offspring* [3, 13, 20].

Evolution can be described as a process, in which individuals have to adapt to the environment in order to survive. For evolution to take place, it is necessary to have more than one individual. We need the whole population of individuals, so they can evolve and produce better offspring. In order for the population to evolve, it must comply with the following rules:

- Members of the population have to be capable of producing offspring.
- The offspring has to carry forward the parent's properties with some variations.
- The offspring that adapted to the environment has to survive with a higher probability.

If a population meets these rules, it doesn't necessarily mean that the evolution will occur. However, without these rules, it would be very difficult, almost impossible, for a population to evolve [19].

All evolutionary algorithms are based on this principle of evolution, but each provides a different mechanism for performing the search. According to this, the main evolutionary paradigms can be divided into four categories:

- Genetic algorithms (GA)
- Genetic programming (GP)
- Evolutionary strategies (ES)
- Evolutionary programming (EP)

In this thesis, the genetic algorithm, the CMA-ES algorithm and simulated annealing were used, so in the next section, these algorithms are described in more detail.

2.1 Genetic algorithms

The genetic algorithms (GAs) belong to a class of evolutionary algorithms that use techniques mimicking evolutionary processes in biology. GAs were first introduced by John H. Holland in 1975. They are search algorithms based on natural selection and the principles of genetics.

The main advantage of GAs is that they are simple and applicable to a wide range of different tasks. Also, the performance of the algorithm does not depend on the representation compared to other numerical methods. The evaluation of each individual can be calculated in parallel.

There are also some limitations to the use of the genetic algorithm. First of all, GAs tend to converge towards local optima rather than the global optimum. The quality of results depends highly on the initial population and the genetic operators (selection, crossover, mutation) and whether they are well-suited for the solved problem. Moreover, repeated fitness function evaluation for complex problems is often the most prohibitive and limiting aspect. Finding an optimal solution to complex problems often requires very expensive fitness function evaluations, which means that a single function evaluation may require several hours to several days of complete simulation [37].

This chapter is mainly based on my bachelor thesis and materials for the courses EVO (Applied Evolutionary Algorithms) and SFC (Soft Computing) [25, 8, 38].

2.1.1 Terminology and algorithm procedure

Genetic algorithms are based on principles of genetics and therefore it is clear that they have similar terminology as in genetics. However, some terms are simplified from their biology version.

- A **gene** is a section, a sequence, that encodes an individual's parameters. The gene acquires different values called alleles.
- An **allele** is a specific form of a gene. We understand alleles as different values that a gene can acquire. For example in binary representation, a gene can acquire one of two different alleles, value 0 or 1.
- An **individual** is a unique representative of a population. This means that an individual represents one solution found by a genetic algorithm. An individual is defined by a single chromosome which has a fixed, predetermined length.
- In a genetic algorithm, the **population** consists of a set of individuals, where each individual represents a solution to a given problem, whether appropriate or inappropriate.
- The **generation** consists of all individuals whom we work with at the same time. One population of individuals consists of more generations. By crossing individuals of one generation (parents), individuals of a new generation (offspring) are created.
- The **chromosome** is a carrier of genetic information and is divided into individual genes which are arranged linearly. That means the i-th gene of a chromosome of the same type represents the same property. In genetic algorithms, a chromosome may be encoded differently. The method of encoding affects the success or failure of a genetic

algorithm on the given task. According to the type of problem to be solved, we must choose the ideal encoding. The encoding types are:

  - **Binary Encoding**: Every chromosome is represented as a string of bits, 0 or 1. The binary encoding provides many possible chromosomes even with a short chromosome length.
  - **Permutation Encoding**: This encoding can be used in ordering problems. Each chromosome contains numbers or letters that represent the order. For example, in the case of a Travelling salesman problem, the ECDAB chromosome expresses the order of the visited places, which means that he visits city E first, then cities C, D, A and finally, he goes to city B. Individual numbers or letters are in the chromosome only once, because he visits every city just once.
  - **Value Encoding**: In value encoding, each chromosome is a string of some values. Values can be anything related to the problem, for example, real numbers or characters. This type of encoding is very good for some specific problems, but there is also a need to develop new crossover and mutation for the problem.
  - **Tree Encoding**: Every chromosome is a tree of some objects. These object can be functions or commands in the programming language. Tree encoding is used for evolving programs or expressions in genetic programming.

- The **fitness** of an individual is a measure of how "good" the solution represented by the individual is. The better the solution, the higher the fitness. The following requirements should be satisfied by any fitness function [24, 32]:

  1. The fitness function should be clearly defined. The reader should be able to clearly understand how the fitness score is calculated.
  2. The fitness function should be implemented efficiently. If the fitness function becomes the bottleneck of the algorithm, then the overall efficiency of the genetic algorithm will be reduced.
  3. The fitness function should quantify how the solution fits into the problem.
  4. The fitness function should generate intuitive results. The best/worst candidates should have the best/worst score values.

The genetic algorithm consists of several steps. Each step simulates a certain biological process. The first step of the algorithm is to create an initial population. Subsequent steps are performed in a cycle until an optimal solution is found or the maximum number of iterations is reached. These steps are: population evaluation, selection, crossover, mutation and the creation of a new population.

2.1.2 Creating an initial population

The initial population is made up of a predetermined number of individuals. Individuals are usually generated randomly. If we want to generate individuals close to the optimal solution, we can use heuristics that will allow it.

The number of individuals in a population depends on the problem, but typically a population consists of hundreds to thousands of individuals because it is known from biology that small populations can extinct due to a lack of sufficient genetic diversity [5].

2.1.3 Population evaluation

Each individual represents one solution to a given problem. In order to determine which solutions are better and which are worse, it is important to be able to properly evaluate all individuals. As already mentioned, the fitness function is used for this evaluation. The evaluation of an individual consists of comparing how much the solution found by this individual differs from the optimal, correct solution to the given problem.

Fitness function quantitatively evaluates the degree of chromosome adaptation concerning the requirements (criteria, goals) of the evolutionary process.

The *purpose function* evaluates quantitatively the quality of a candidate solution in relation to a given problem. Its result can be directly the fitness value (then it is identical with the fitness function), but it can also serve as an „intermediate result“ for the calculation of fitness.

The value of the fitness function also serves as a termination condition. The algorithm can be set to terminate as soon as the optimal solution is found (if we know it) or if the value of the fitness function has not improved for several generations. The algorithm can be terminated even if the maximum number of iterations is reached. In that case, the individual who came closest to the optimal one is provided as a solution.

Evaluation of individuals is an important part of the next step of the genetic algorithm, selection. While in nature individuals compete with each other and parents become the best of them, in genetic algorithms this competition must be imitated differently. The fitness value is used for this and it replaces natural selection when choosing parents. Individuals with a higher fitness value tend to be preferred when choosing.

2.1.4 Selection

The selection is a process of selecting a pair of individuals from a population which will become parents. The said pair of individuals (parents) will cross each other to create a new individual or individuals (offspring). For the quality of the population to improve with each generation and lead us to the solution we are looking for, the selection must simulate the natural selection of individuals from nature, thus choosing the best ones. Natural selection is achieved by selecting the most capable individuals, which is aided by selective pressure. Selective pressure suppresses below-average individuals and, on the contrary, favours above-average individuals.

From a mathematical point of view, the selection is about determining the probabilities with which individuals can be selected for the process of reproduction. The most well-known ways of choosing parents are roulette, stochastic universal sampling, tournament and elite selection.

Roulette and **stochastic universal sampling** are based on the same principle. The sections of the roulette wheel are proportional to the fitness values of individuals. Individuals with a higher value of the fitness function occupy a larger part of roulette, which increases the probability that they will be chosen. Individuals with a low fitness value occupy a smaller part of roulette, which reduces the probability of their choice. The difference between roulette and stochastic universal sampling is in the number of roulette spins. Roulette has only one selective arrow, so the number of spins is equal to the number of parents. In each spin, one parent is selected. The stochastic universal sampling has as many selective arrows as the number of parents and there is only one spin in which all parents are selected.

A **tournament** is a method of selection in which a population is divided into several groups, where individuals are randomly distributed. The number of groups depends on the total number of individuals in the population and the size of the tournament. After the division, the individual with the highest fitness value is selected from each group and it becomes a parent. The tournament preserves the diversity of the population, as it may happen that only worse individuals get into the group and one of them will have to be chosen as a parent.

The **elite selection** consists of sorting individuals in ascending order according to the value of their fitness function. Only a number of the best-rated individuals are then selected as parents, usually 10% to 50%. This type of selection suppresses the influence of above-average individuals and helps maintain selection pressure.

2.1.5 Crossover

Crossover, also called recombination, produces new individuals from information contained in the parents' genes by exchanging parts of their chromosomes. Depending on how the individual parts of the parents are crossed, we distinguish several types of crossovers.

The **single-point crossover** consists of selecting one crossing point. Then the offspring is created in such a way that everything before this crossing point is from one parent and everything behind this point is from the other parent.

First parent:	a b c	d e f g
Second parent:	a b c	d e f g
First offspring:	a b c	d e f g
Second offspring:	a b c	d e f g

Multi-point crossover consists of selecting multiple crossing points, two at least. The offspring are created by copying everything from the beginning to the first crossing point from the first parent, everything from the first crossing point to the second crossing point is copied from the other parent, everything from the second crossing point to the end of the next crossing point is copied again from the first parent, etc. The maximum number of crossing points for n genes is $n - 1$.

First parent:	a b	c d e	f g
Second parent:	a b	c d e	f g
First offspring:	a b	c d e	f g
Second offspring:	a b	c d e	f g

Uniform crossover is that each binary gene is transferred from either the first or second parent with some probability. This probability is fixed and is usually 0.5. This means that there is a 50% probability that the gene will be transferred from the first parent and a 50% probability that it will be transferred from the second parent.

First parent:	a b c d e f g
Second parent:	a b c d e f g
First choice:	1 2 1 1 2 1 1
Second choice:	2 2 1 2 1 2 1
First offspring:	a b c d e f g
Second offspring:	a b c d e f g

Arithmetic crossover is used for genes with real values. This crossover selects values of individual genes randomly from both parents with a uniform probability distribution of the coefficient a_i according to these formulas:

$$\begin{aligned} g_i^{O_1} &= g_i^{P_1} a_i + g_i^{P_2}(1-a_i) \qquad a_i \in [-0.25, 1.25] \\ g_i^{O_2} &= g_i^{P_1}(1-a_i) + g_i^{P_2} a_i \end{aligned} \tag{2.1}$$

where g_i is a value of gene i, P_j is a parent, $j = 1, 2$ and O_k is offspring, $k = 1, 2$.

Heuristic crossover is also used for genes with real values. It selects values of individual genes from parents according to a randomly selected coefficient $r \in < 0, 1 >$:

$$\begin{aligned} O_1 &= P_{better} + r * (P_{better} - P_{worse}) \\ O_2 &= P_{better} \end{aligned} \tag{2.2}$$

where P_j is a parent, $j = better, worse(f_{better} > f_{worse})$ and O_k is an offspring, $k = 1, 2$.

Crossing in permutation optimization problems is used for problems where we cannot simply take one part of the genes from the first parent and the other part from the second parent, because the solution given by this individual would not make sense. For example, in the case of a traveller salesman problem, it could happen that some cities will not be visited even once and some even twice. Therefore, these problems are handled differently, for example, by using index tables or PMX.

2.1.6 Mutation

A mutation is a change in genetic material. In genetic algorithms, we understand it as a change in the value of a randomly selected gene in a randomly selected offspring. The mutation has a predetermined low probability. It usually ranges from 0.001 to 0.05 and can be calculated as $P = \frac{1}{n}$, where n is the number of individuals in the population.

If the individual consists of binary values 0 and 1, the mutation means inverting the value of one bit. A bit with a value of 0 is inverted to 1 and a bit with a value of 1 is inverted to 0.

Before mutation: 1 0 1 1 0 0 1 1 0 $\rightarrow$ After mutation: 1 0 1 1 0 0 0 1 0

For an individual which consists of real values, a mutation means that a small, random real value is added to the value of the random gene. Its size is a maximum of 10 % and can also be negative.

Before mutation: 1.85 -2.56 6.23 -0.79 $\rightarrow$ After mutation: 1.85 -2.56 5.94 -0.79

A mutation in permutation optimization problems must always preserve any letters or numbers that are used to represent an individual. We cannot just mutate one value, we always have to exchange two (or more) values with each other, or we can invert the order of number in some interval, we can transfer a number from one position to another, etc [31]. An example of mutation, where two genes are randomly selected and their positions exchanged can be seen below.

Before mutation: 4 9 1 5 2 7 6 3 8 $\rightarrow$ After mutation: 4 9 3 5 2 7 6 1 8

2.1.7 Creating a new population

The new population will replace the previous population. According to the number of original individuals who are replaced in the new generation, we are talking about three types of models:

- The **incremental model** means that in the new population, only one individual of the original population is replaced by the offspring.
- In the **generation model**, all individuals of the original population are replaced by offspring in the new population.
- **Models with overlapping generations** mean that in the new population, part of the individuals of the original population is replaced by offspring. The incremental model and the generation model are extreme cases of this model.

2.2 CMA-ES

CMA-ES (Covariance Matrix Adaptation - Evolution Strategy) is a stochastic method for real-parameter optimization of non-linear, non-convex functions. It belongs to the class of evolutionary algorithms and evolutionary computation. The CMA-ES is considered as the state-of-the-art in evolutionary computation. The main features of evolution strategy are the concept of advanced self-adaptation of parameters, mutation as the main variational operator and different strategies for reproduction and survival. CMA-ES then extends this principle by using the construction of a rotation matrix to calculate the optimal mutation [16].

CMA-ES does not require lengthy parameter tuning for its application. The choice of the internal parameters of the strategy is not left to the user (arguably except for the population size λ). Finding good (default) strategy parameters is considered part of the algorithm design and not part of its application - the goal is to have a well-performing algorithm as it is. For the CMA-ES application, the user must set the initial solution, the initial standard deviation (step-size) and possibly also the termination criteria [15].

Although we can use any distribution function, the normal distribution is most often used. This is mainly because the normal distribution is commonly observed in nature, for example as phenotypic traits and is the only stable distribution with finite variance. It is also the most convenient way to generate isotropic (it does not favour any direction) search points. And it has a maximum entropy distribution with finite variance [17].

2.2.1 Covariance and covariance matrix

Covariance is a measure of the joint variability of two random variables. For uncorrelated variables, the covariance is zero. However, if the variables are correlated in some way, then their covariance will be nonzero. A positive covariance arises if greater values of one variable correspond mainly with greater values of the other variable, and the same applies for lesser values. It means that with a positive covariance, the variables tend to show similar behaviour. Otherwise, the variables tend to show the opposite behaviour, and we called it a negative covariance. That means that greater values of one variable correspond mainly to lesser values of the other. It follows that the sign of covariance shows a tendency in a linear relationship between variables.

Definition 1. *Covariance*

Let X_i and X_j be random variables and for their mean values let:

$$E[{X_i}^2] < \infty$$
$$E[{X_j}^2] < \infty$$

Then a real number expressed by the following relation:

$$cov(X_i, X_j) = E[(X_i - E[X_i]) * (X_j - E[X_j])] \tag{2.3}$$

indicates the covariance of these two random variables. In words, the covariance is defined as the expected value (or mean) of the product of their deviations from their individual expected values.

A *covariance matrix* is a square matrix giving the covariance between each pair of elements of a given random vector. It means that the matrix has a covariance between the i-th and j-th elements of the random vector at the intersection of the i-th row and the j-th column. Any covariance matrix is symmetric and positive semi-definite, and its major diagonal contains variances (the covariance of each element with itself).

Definition 2. *Covariance matrix*

If the entries in the column vector

$$X = (X_1, X_2, ..., X_n)^T$$

are random variables, each with finite variance and expected value, then the covariance matrix K_{XX} is the matrix whose (i, j) entry is the covariance

$$K_{X_iX_j} = cov(X_i, X_j) = E[(X_i - E[X_i]) * (X_j - E[X_j])] \tag{2.4}$$

where the operator E denotes the expected value (mean) of its argument [27].

2.2.2 Algorithm overview

In CMA-ES in each generation K points $\mathbf{x}_1, ..., \mathbf{x}_K$ with dimension n are sampled using a normal distribution $\mathcal{N}(\mathbf{m}, \mathbf{C})$, where the normal distribution is completely specified by the mean vector $\mathbf{m}$ and the covariance matrix $\mathbf{C}$. From the K points, Z of them that have the fittest f-values are selected and then they are used to calculate a weighted mean for the normal distribution of the next generation. Calculation of the covariance matrix is performed in such a way that both the variance between the selected points in one generation and the correlation between generations is fully exploited. The objective of CMA-ES is to fit the search distribution to the contour lines of the objective function to be minimized. In Figure 2.1, we can see that with each generation, the parameters of the search distribution (the mean vector and covariance matrix) are adapted and at the end all the f-evaluations are concentrated in the neighbourhood of the minimum [22]. CMA-ES follows these steps:

- Sampling (Generating new population)
- Updating the mean (Selection and Recombination)
- Estimating the covariance matrix
- Step size control

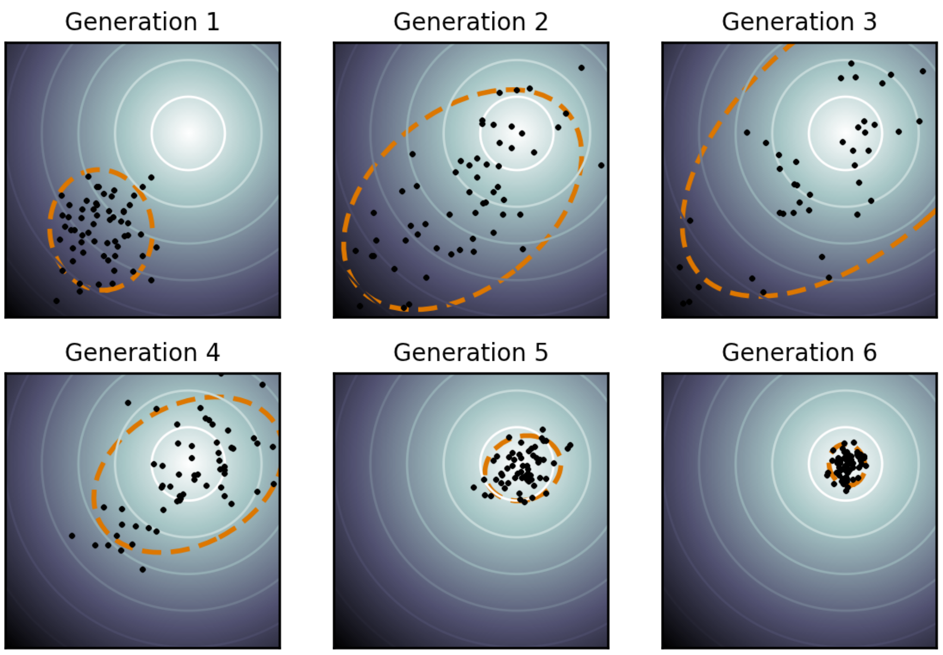


Figure 2.1: Illustration of an optimization run with covariance matrix adaptation on a simple two-dimensional problem [30].

Sampling (Generating new population)

In each generation g, K points $\mathbf{x}_1, ..., \mathbf{x}_K$ are sampled using a normal distribution as follows

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)}\mathcal{N}(0, \mathbf{C}^{(g)}) \text{ for } k = 1, ..., K \tag{2.5}$$

where $\mathbf{m}^{(g)}$ is the mean vector and $({\sigma^{(g)}}^2\mathbf{C}^{(g)})$ is covariance matrix of normal distribution $\mathcal{N}$ at generation g. $\sim$ denotes the same probability distribution on the left and right side. $\sigma^{(g)}$ is the step size and is described in *Step size control* section [22].

Updating the mean (Selection and Crossover)

The objective function is evaluated at K points, which were generated in the previous step. Then, Z best points are selected and their weighted average is calculated as:

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{Z} w_i \mathbf{x}_{i:K} \tag{2.6}$$

where $\mathbf{x}_{i:K}$ is the i-th fittest point among $\mathbf{x}_1, ..., \mathbf{x}^K$ and w_i are the recombination weights. Their sum equals one and $w_1 \geq w_2 \geq ... \geq w_Z > 0$. A parameter variance effective mass is defined as

$$\mu_{\text{eff}} = \left(\sum_{i=1}^{Z} w_i^2\right)^{-1} \tag{2.7}$$

If equal recombination weights are used, i.e. $w_i = 1/Z$, then $\mu_{\text{eff}} = Z$, more generally $1 \leq \mu_{\text{eff}} \leq Z$ [22].

Estimating the covariance matrix

There are various approaches to the estimation of the covariance matrix. If we have a small population, some approaches can result in unreliable estimates. To deal with this, we can use information from previous generations. For example, the next generation covariance matrix can be estimated as the average of all the covariance matrices of the previous generations. A better way would be to give higher weight to recent generations and lower weight to older generations. This can be accomplished by exponential smoothing, with a learning rate $0 < c_Z \leq 1$. This approach is known as the **rank-Z update** and equation is:

$$\mathbf{C}^{(g+1)} = (1 - c_Z)\mathbf{C}^{(g)} + c_Z \sum_{i=1}^{Z} w_i \mathbf{y}_{i:K}{}^{(g+1)} \mathbf{y}_{i:K}{}^{(g+1)^T} \tag{2.8}$$

where $\mathbf{y}_{i:K}^{(g+1)} = (\mathbf{x}_{i:K}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^{(g)}$.

Apart from the rank-Z update, CMA-ES also uses a **rank-1 update** in which the covariance matrix is repeatedly updated in the generation sequence using a single selected step only. If we set Z of previous equation to 1, we get equation for rank-1 update:

$$\mathbf{C}^{(g+1)} = (1 - c_1)\mathbf{C}^{(g)} + c_1 \mathbf{y}^{(g+1)} \mathbf{y}^{(g+1)^T} \tag{2.9}$$

One thing to notice about this equation is that the sign of the selected steps does not count, as $\mathbf{y}\mathbf{y}^T = (-\mathbf{y})(-\mathbf{y})^T$. Because of that, to utilize the sign information, the concept of an evolution path is introduced. An **evolution path** is a sequence of successive steps, that the strategy performs over several generations. An evolution path can be expressed by a sum of consecutive steps. A better strategy could be to use exponential smoothing like last time, to get the evolution path $\mathbf{p}_c^{(g)}$ given by:

$$\mathbf{p}_c^{(g+1)} = (1 - c_c)\mathbf{p}_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{eff}} \frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}} \tag{2.10}$$

where c_c is the backward time horizon for the evolution path. The task of the normalization constant $\sqrt{c_c(2 - c_c)\mu_{eff}}$ is to ensure that $\mathbf{p}_c^{(g+1)}$ and $\mathbf{p}_c^{(g)}$ have the same probability distribution.

Combining all three equation we get the CMA update equation:

$$\mathbf{C}^{(g+1)} = (1 - c_1 - c_Z)\mathbf{C}^{(g)} + c_1 \mathbf{p}_c^{(g+1)} \mathbf{p}_c^{(g+1)^T} + c_Z \sum_{i=1}^{Z} w_i \mathbf{y}_{i:K}{}^{(g+1)} \mathbf{y}_{i:K}{}^{(g+1)^T} \tag{2.11}$$

where the second part represents **rank-1 update** and the third part represents **rank-Z update** [22].

Step size control

Consider three different selection scenarios shown in Figure 2.2. Although the length of the individual steps is the same, the length of the evolution path varies. On the left side (a), the evolution path is short, because the individual steps cancel out each other. They are anti-correlated and to fix it, the step size should be decreased. On the right side (c), the evolution path is long, because the individual steps are almost in the same direction. They are correlated and larger step size can be used with fewer steps to cover the same distance.

In the middle (b), the desired length of the evolution path is shown. The individual steps are approximately uncorrelated and hence there is no need to change the step size.

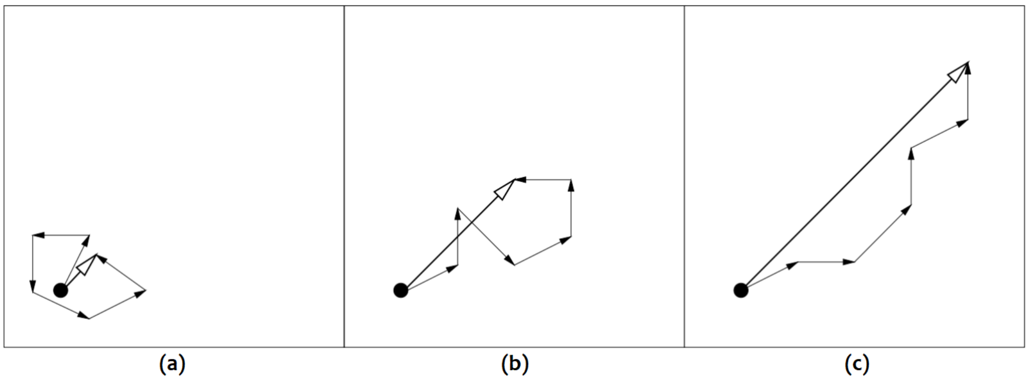


Figure 2.2: Three evolution paths of six (equal length) steps from different selection situations [22].

To determinate whether the evolution path is short or long, we compare the length of the evolution path with the expected length of the path under random selection. In the path under random selection, the subsequent steps are independent and thus uncorrelated and this should be the preferred path length. Thus, we increase the step size if the evolution path is longer than the random selection path and decrease it if the evolution path is shorter than the random selection path [22].

2.3 Simulated annealing

Simulated annealing belongs to memetic algorithms, which are a branch of evolutionary algorithms. Simulated annealing algorithm is based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimization problems. SA is a good example of how introducing ideas from a different and at first sight, an unrelated field may bring unexpected benefits.

In condensed matter physics, annealing refers to the physical process by which a solid is placed in a furnace heated to maximum temperature. At this value, all particles of the solid randomly arrange themselves in the liquid phase. Then it is followed by cooling where through slowly lowering the temperature the internal defects of the solid are removed. In this way, all particles arrange themselves in the low energy ground state of a corresponding lattice, provided the maximum temperature is sufficiently high and the cooling is carried out sufficiently slowly [23].

The SA algorithm starts at the maximum value of the temperature. Then, the cooling phase follows. At each temperature value T, the solid is allowed to reach thermal equilibrium. The thermal equilibrium is characterized by a probability of being in a state with energy E_i given by the Boltzmann distribution:

$$W_T(E_i) = \frac{1}{\mathbf{Z}(T)} \cdot exp\left(-\frac{E_i}{k_B T}\right), \tag{2.12}$$

where $\mathbf{Z}(T)$ is a normalization factor, known as the partition function, depending on the temperature T and k_B is the Boltzmann constant. The factor $exp(-\frac{E}{k_B T})$ is known as the

Boltzmann factor. As the temperature decreases, the Boltzmann distribution concentrates on the states with the lowest energy and finally, when the temperature approaches zero, only the minimum energy states have a non-zero probability of occurrence.

If the cooling is too rapid, i.e. if the solid is not allowed to reach thermal equilibrium for each temperature value, defects can be 'frozen' into the solid, and thus metastable structures are formed that are far from the lowest energy lattice structure. In SA, these 'frozen' defects mean local minima that the algorithm cannot overcome.

For the simulation of Boltzmann distribution in SA, the Metropolis algorithm is used. The algorithm is an implementation of Monte Carlo method and simulates the evolution of a system by generating a sequence of states as follows:

- Let be given the current state of the system, which is determined by the position of the body particles. Then, a small random fault is generated by gently shifting the particles. This fault must be symmetrical, ie. the probability that with a small fault, state A will change to state B must be the same as changing state B to A.

- If the energy difference between the disturbed state and the current state is negative, then the process continues with the new disturbed state. Otherwise, the probability of acceptance of the violated condition is determined by the relationship $exp(-\frac{\Delta E}{k_B T})$, which is called the Metropolis criterion.

- According to this criterion, by applying a large number of perturbations and accepting them into another process with probability P, we obtain a system of thermal equilibrium, while the probability distribution of the state distribution is asymptotically close to the Boltzmann distribution.

The whole algorithm is controlled by *Cooling Schedule*. The most used schedule is the linear cooling schedule, which can be defined as $T(i) = T_0 * \alpha^i$. The α parameter ranges from 0 to 1, inclusive. If we choose a suitably small α, the cooling schedule is slow enough to allow global convergence [4, 11, 21, 23].

Chapter 3

Optimization problem

Consider a system **S** ($\sim$ problem **P**) whose operation (solution quality) can be evaluated by the function *f*. Following the above, we call the process of finding the characteristics $\vec{x}$, that meet the requirements for the operation of the system **S** (solution of the problem **P**), as the optimization of the function f (finding a solution to the problem **P**) [9].

We can approach this in various ways using either precise methods (A*, backtracking, ...) or approximate methods (approximation algorithms, heuristics, ...). *Precise methods* guarantee to find the optimal solution, but they do not guarantee to find the solution (even suboptimal) in an acceptable time. *Approximation algorithms* guarantee to find a solution within a given quality limit, but in terms of time, they are often insufficient for complex problems. The goal of *heuristics* is to provide a satisfactory (quality) solution in a reasonable time, although the optimum cannot be generally guaranteed. Heuristics are also problem-dependent, i.e, we can define a heuristic for a specific problem. *Metaheuristics* represent strategies, templates for creating heuristic algorithms applicable to various problems. Metaheuristics are problem-independent techniques, so they can provide a solution for a range of problems.

When designing heuristic methods, we must focus on two criteria (exploration and exploitation) that go against each other, so we have to find a compromise. By *exploration* we mean the discovery of new areas of the state space, we try to uncover as much of it as possible. *Exploitation* is the utilization of information in promising parts of the state space, means refinement of the solution [9].

We understand the optimization problem as finding the (global) optimum of the function f from the set of potential solutions M of a given problem domain.

Definition 3. *Global minimum*

Given a function $f : M \subseteq \mathbb{R}^n \to \mathbb{R}, M \neq \emptyset$, for $\vec{x}^* \in M$ the value $f^* := f(\vec{x}^*) > -\infty$ is called a global minimum, if and only if

$$\forall \vec{x} \in M : f(\vec{x}^*) \leq f(\vec{x}). \tag{3.1}$$

Then, $\vec{x}^*$ is the global minimum point, f is called the objective function, and the set M is called the feasible region. The problem of determining global minimum point is called the global optimization problem [12]. Thanks to a relationship

$$max\{f(\vec{x})|\vec{x} \in M\} = -min\{-f(\vec{x})|\vec{x} \in M\} \tag{3.2}$$

we can always assume that the optimization problem is a minimization problem and we do not have to deal with maximization.

3.1 Focused ultrasound

Focused ultrasound (FU) is an early-stage, non-invasive therapeutic technology that can interact with the body in several different ways. FU can produce both mechanical and thermal energy and can supply it through high-pressure or low-pressure waves. These various focused ultrasound applications induce a wide range of biological effects on the treated tissue. Some effects, such as tissue destruction and clot lysis, are permanent, others, such as blood-brain barrier opening and neuromodulation, are transient and reversible [2].

Several types of ultrasound applications can produce the same biological effect, which further increases the versatility of focused ultrasound in the treatment of diseases. These ultrasound applications can be:

- **Nonthermal** - Low-intensity ultrasound is applied. It produces a temperature rise of less than 2°C and the biological effects are mediated by mechanical forces.
- **Histotripsy** - High-pressure focused ultrasound capable of destroying tissue through mechanical rather than thermal effects.
- **Hyperthermia** - A slight increase in temperature to 42°C performed continuously for several hours to induce hyperperfusion, a physiological response that increases the supply of blood and drugs in the bloodstream to the target area.
- **Thermal Ablation** - Tissue heating denatures proteins and leads to the death of all cells. The heat dose required to induce irreversible damage and coagulation necrosis depends on the cell type, temperature and exposure time [2].

3.1.1 High-intensity focused ultrasound

High-intensity focused ultrasound (HIFU) is a non-invasive method to treat certain primary solid tumours and metastatic diseases, to ablate foci of ectopic electrical activity in the heart, and to achieve hemostasis in acute traumatic injuries to the limbs and internal organs. Unlike invasive methods such as radiofrequency or cryoablation, which are also used to ablate tumours, ultrasound is completely non-invasive and can be used to reach tumours deep in the body [14].

The main effect of HIFU ablation is coagulation thermal necrosis due to acoustic energy absorption. The basic principle of thermal treatment of HIFU is to supply sufficient ultrasound energy to increase the temperature by several tens of degrees to destroy the tissue through coagulation necrosis. The HIFU beam is focused only on a specific location within a small volume, which minimizes the possibility of tissue damage outside the focal region, in other words, it produces cell death in the target area with minimal damage to the surrounding tissue [34].

Figure 3.1 shows how HIFU can pass through the skin, fat and muscles and destroy the tumour. One lesion is almost never enough to destroy the entire tumour. To treat larger structures, such as tumours, multiple lesions can be combined to cover the entire volume. Each lesion removes a certain part of the tumour, which means that after all lesions, the whole tumour is destroyed. Cooling is often required between applications to prevent unwanted heating of the surrounding tissue. Therefore, the treatment of very large structures can be time consuming. However, there are techniques to reduce treatment time. These include optimized scanning algorithms, microbubble injection to increase acoustic energy absorption, and the use of helical sonications [28].

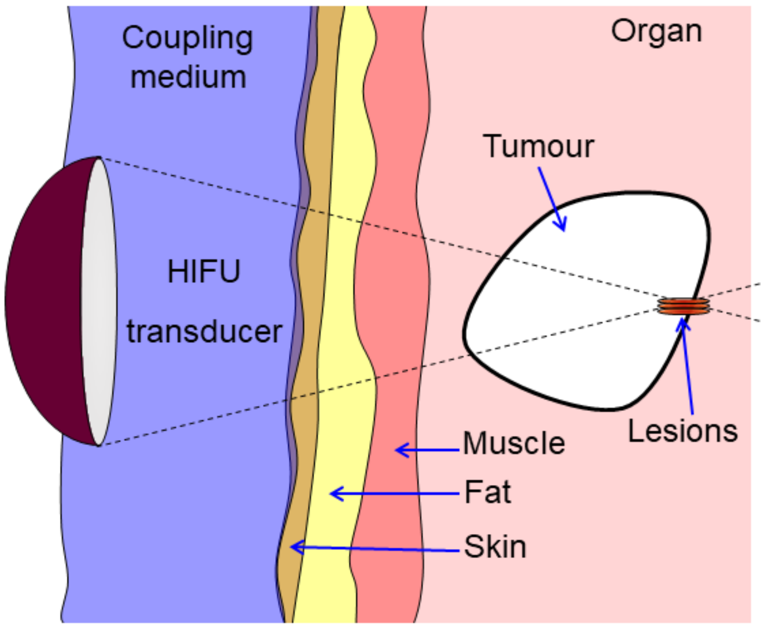


Figure 3.1: Diagram showing how HIFU can be used to destroy a soft tissue tumour [26].

3.1.2 Low-intensity focused ultrasound

Low-intensity focused ultrasound (LIFU) induces reversible biological effect. It has bi-modal capability, which means both excitatory and suppressive capability, with excellent spatial specificity and depth penetration. LIFU can be used to stimulate or suppress (increase or block) neural activity without damaging the nerves in the brain. It uses both mechanical and thermal mechanisms of ultrasound for this. Thus, there is a rapidly growing interest in the application of LIFU-mediated neuromodulation for the treatment of neurological or psychiatric disorders [7]. The effect of neuromodulation is shown in Figure 3.2.

Neuromodulation mediated by LIFU has excellent advantages over conventional neuromodulation techniques such as deep brain stimulation (DBS), transcranial magnetic stimulation (TMS), vagus nerve stimulation (VNS), and transcranial current stimulation (tCS), which have a lot of drawbacks. For example, LIFU is non-invasive, has a very good depth of stimulation (10-15 cm or more) and can be used in conjunction with fMRI brain mapping. On the other hand, the DBS technique is invasive, which means that it requires surgery that can cause an infection or immune response and has also cumbersome maintenance. Furthermore, both TMS and tCS are limited by the fact that stimulation fields cannot be highly controlled due to the lack of spatial specificity and depth. It is also very difficult to use any of these methods with fMRI brain mapping [29].

Deep brain stimulation is used to treat Parkinson's Disease, dystonia and obsessive-compulsive disorder (OCD), vagus nerve stimulation is used for epilepsy and depression, and repetitive transcranial magnetic stimulation for the treatment of depression. It can be expected that all these conventional neuromodulatory techniques will be replaced in the future by the non-invasive LIFU method, with which we will be able to treat all the above-mentioned diseases [10].

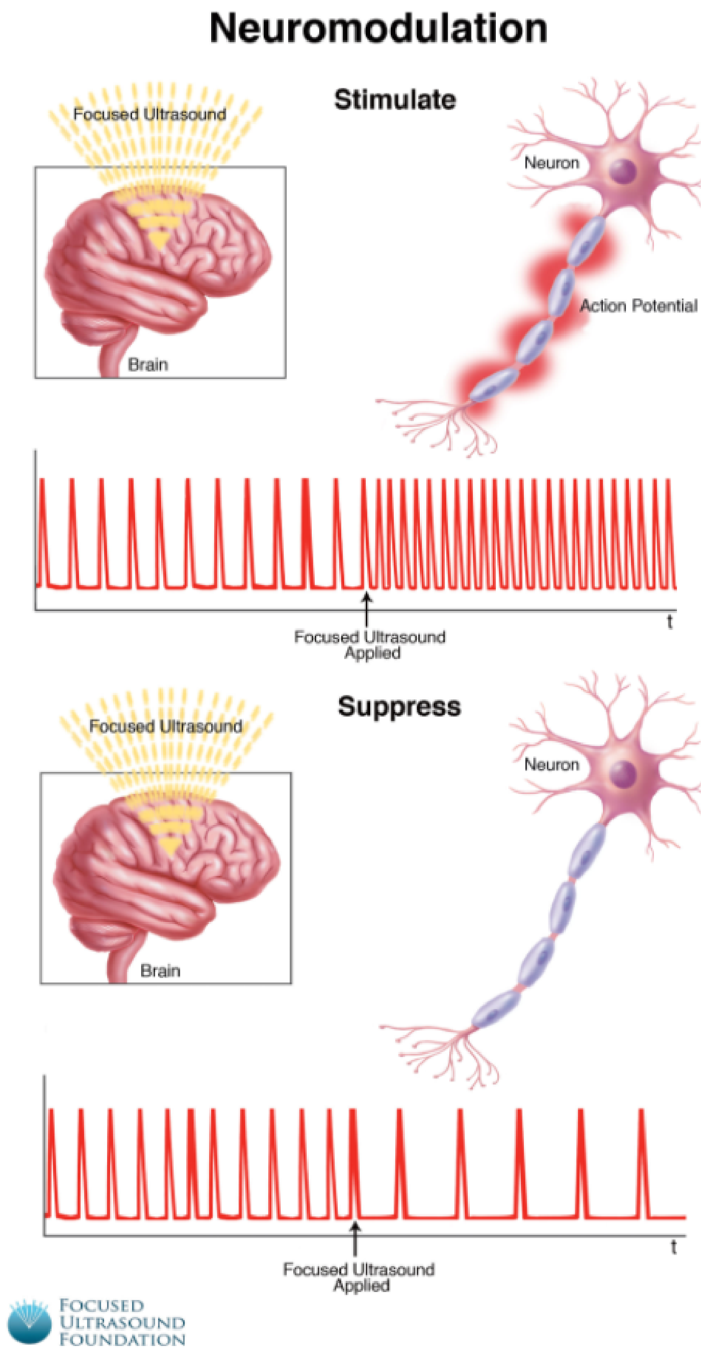


Figure 3.2: Diagram showing how LIFU can be used to stimulate (increase) or suppress (block) neural activity [1].

3.2 Transducer position optimization

Transducer position optimization is a problem, which we can described as follows: *Where should we place the transducer so that the ultrasound beam hits the target, respectively, hits* *as much of the target as possible?* In other words, we are trying to find the best possible position of the transducer.

When we want to simulate the transducer, we have to know its position, but also the direction of the beam, aperture diameter and curvature of arc. The transducer transmits ultrasound in all directions. In order to increase the pressure in a certain area (point), it is necessary for the transducer to be slightly curved. As a result, the transmitted waves meet at a certain point and this is where the pressure increases.

In MATLAB, the transducer can be defined by four values, namely its arc position, focus point, radius and aperture diameter. Arc position and focus point consist of two values, x and y coordinates. This means that we need to know a total of six values, the midpoint of the arc given as x and y coordinates, focus point, which is any point on the beam axis of the arc given as x and y coordinates, radius of curvature of the arc and aperture diameter. The aperture diameter is the length of the line connecting the endpoints of the arc and is the only value known in advance ($41e^{-3}$ m = 4.1 cm). The values of the remaining five variables are not given but are searched using optimization algorithms.

3.3 Simulation of ultrasound

Sound is a mechanical wave in the form of molecular vibrations that transfers energy from one position to another. Ultrasound is a term used for sound waves with frequencies higher than the upper audible limit of human hearing. To simulate the propagation of ultrasound, the functions from the k-Wave package[1] are used. k-Wave is an open source acoustics toolbox for MATLAB and C++. Every simulation function in k-Wave requires four input structures. These structures define:

- the properties of the computational grid
- the material properties of the medium
- the properties and locations of any acoustic sources
- the properties and locations of the sensor points used to record the evolution of the pressure and velocity fields over time

The computation grid is a regular Cartesian mesh. The medium is an object through which ultrasound passes. The medium can be homogeneous or heterogeneous. The main property which has to be define in medium is sound speed. Additional properties which can be defined are absorption coefficients, density, etc. For a homogeneous medium, sound speed and other properties are defined as scalar values. For a heterogeneous medium, properties are defined as an array of scalar values. The source is defined as initial pressure distribution for each grid points. The sensor is specified by a mask which defines the positions within the computational domain where the pressure field is recorded at each time-step.

3.3.1 Medium

In the first half of the academic year, only a homogeneous medium was considered. All experiments were performed on a medium with a sound speed of 1500 m/s, which is equal to a sound speed in seawater. In the second half of the academic year, the solution was extended to a more complex heterogeneous medium. This medium consists of seawater, skin, skull and brain, where speed of sound and density are different in every one of them.

[1] http://www.k-wave.org/

Homogeneous medium

A homogeneous medium is a medium whose all parameters (sound speed, density, etc.) are scalar values. In our case, a medium with a sound speed of 1500 m/s was used, which is equal to the speed of sound in seawater.

To simulate the propagation of ultrasound in a homogeneous medium, an acoustic field propagator is used. Acoustic field propagator (AFP) is a function written in MATLAB by Bradley Treeby and Ben Cox. This function is part of the k-Wave Toolbox and calculates the steady-state field pattern (complex pressure or amplitude and phase) from an arbitrary phased array (or another acoustic source) driven by a single frequency continuous wave sinusoid in a homogeneous and lossless medium. There is also a compiled variant in C++ `acousticFieldPropagatorC`, but only the MATLAB version was used in this work. As inputs, AFP function takes five parameters. The first parameter is `amp_in` which represents a matrix of the source amplitude at each grid point. Next AFP input is `phase_in` which is a matrix of the source phase at each grid point and is always 0. Next input, `dx`, is a grid spacing. The other two parameters, `f0` and `c0`, represent source frequency in Hz and medium sound speed in m/s. There can also be a few optional inputs. The preview of usage with only required inputs can be seen below.

```
pressure = acousticFieldPropagator(amp_in, phase_in, dx, f0, c0)
[amp_out, phase_out] = acousticFieldPropagator(amp_in, phase_in, ...
                                              dx, f0, c0)
[pressure, amp_out, phase_out] = acousticFieldPropagator(amp_in, ...
                                              phase_in, dx, f0, c0)
```

As we see from usage examples, as an output we can have `pressure` or `amp_out` and `phase_out` or all of three. Value of `pressure` is a matrix of the complex pressure field at each grid point in steady-state, where the real part corresponds to the pressure field for a cosine excitation, and the imaginary part corresponds to the pressure field for a sine excitation, in pascals. Value of `amp_out` is a matrix of the output amplitude at each grid point in steady-state, in pascals. And value of `phase_out` is a matrix of the output phase at each grid point in steady-state, in radians [33].

Heterogeneous medium

A heterogeneous medium is a medium whose parameters are arrays of different scalar values. In our case, a human head immersed in water is used as a heterogeneous medium. The heterogeneous medium then consists of water, soft tissue (skin and brain) and the skull. The sound speed of water is 1509 m/s, sound speed of soft tissue is 1550 m/s and the sound speed of skull varies from 1500 to 3100 m/s (these ranges are according [6]). This medium was created thanks to function `skull2medium` which converts a CT image of a head and skull to the acoustic and thermal properties used by the k-Wave simulation functions. Function `skull2medium` was created by Bradley E. Treeby.

To simulate the propagation of ultrasound in a heterogeneous medium, the function `kspaceFirstOrder2D`[2] is used. This function simulates the time-domain propagation of compressional waves through a two-dimensional homogeneous or heterogeneous acoustic medium given four input structures: kgrid, medium, source, and sensor. The computation is based on a first-order k-space model which accounts for power law absorption and a heterogeneous sound speed and density. At each time-step (defined by `kgrid.dt` and

[2] http://www.k-wave.org/documentation/kspaceFirstOrder2D.php

kgrid.Nt or kgrid.t_array), the acoustic field parameters at the positions defined by sensor.mask are recorded and stored. An anisotropic absorbing boundary layer called a perfectly matched layer is implemented to prevent waves that leave one side of the domain being reintroduced from the opposite side (a consequence of using the FFT to compute the spatial derivatives in the wave equation). This allows infinite domain simulations to be computed using small computational grids. The preview of usage with required parameters:

```
sensor_data = kspaceFirstOrder2D(kgrid, medium, source, sensor)
```

The output sensor_data returned from the simulation is defined as a structure, with the recorded acoustic variables appended as structure fields. The structure contains data based on the setting of the sensor.record variable. The variable sensor.record is a cell array of the acoustic parameters to record. They are written in the form sensor.record = {'p', 'p_max', ...} and their corresponding results, which are saved in structure sensor_data, can be obtain as sensor_data.p, sensor_data.p_max, etc. If sensor.record variable is not defined by the user, sensor_data contains only one result which represents time-varying pressure recorded at the sensor positions given by sensor.mask. This is equivalent to option 'p' in sensor.record. In this thesis, the variable sensor.record was not defined, so variable sensor_data contains only one result.

After obtaining sensor_data, we need to extract amplitude from them. This can be done using extractAmpPhase[3] function, which extracts the amplitude and phase information at a specified frequency from a vector or matrix of time series data. Example of usage with three required inputs:

```
[amp, phase] = extractAmpPhase(sensor_data, Fs, source_freq)
```

Value of amp is a matrix of the pressure field at each grid point. Value of phase contains phase between 0 and $2 * \pi$ in radians.

3.4 Fitness function

To optimize the position of the transducer, a suitable fitness function had to be selected. As mention in Section 2.1.1: The fitness of an individual is a measure of how "good" the solution represented by the individual is. The better the solution, the higher the fitness. However, since minimization is used in this thesis, the fitness function works in such a way that the smaller its value, the better the solution found by the individual. The best fitness value is 0. Negative fitness does not exist.

Two types of masks were created, target mask and penalty mask. The target mask represents the points we want to hit (for example a tumour). The penalty mask represents points we do not want to hit (internal organs, veins, etc.). The fitness function is generally calculated as the number of missed points from the target mask plus the number of incorrectly hit points from the penalty mask. The fitness function for homogeneous and heterogeneous medium slightly differ. Explanatory notes for symbols used in equations for fitness function:

t	number of points in target mask
t_{hit}	number of hit points in target mask
p	number of points in penalty mask
p_{hit}	number of hit points in penalty mask
$head_{hit}$	number of points representing the head and transducer overlap

[3] http://www.k-wave.org/documentation/extractAmpPhase.php

Homogeneous medium

In a homogeneous medium, both target mask and penalty mask are simply defined as two-dimensional arrays containing values 0 and 1. Target mask is saved in files `rice.txt` and `rotated-rice.txt`. The value 0 represents blank space and value 1 represents our target. The penalty mask is saved in file `rotated-rice-penalisation.txt` and also as in target mask, value 0 represents blank space and value 1 represents our penalty area.

Fitness function is calculated as a number of points in target mask t minus number of hit points t_{hit} from target mask plus the number of points in penalty mask p times number of hit points p_{hit} from penalty mask. In the penalty part, multiplication is used because we want solutions, where many points from the penalty mask are hit to be rated much worse than solutions where the penalty mask is not hit. Equation for calculating the fitness function for a homogeneous medium:

$$fitness = \left(\sum t - \sum t_{hit}\right) + \left(\sum p * \sum p_{hit}\right) \tag{3.3}$$

Heterogeneous medium

In a heterogeneous medium, target mask and penalty mask are obtained from Harvard-Oxford cortical and subcortical structural atlases, which are probabilistic atlases covering 48 cortical and 21 subcortical structural areas, derived from structural data and segmentations. In this thesis `HarvardOxford-cort-maxprob-thr25-1mm.nii`[4] was used. Since the simulation takes place in 2D space, only one slice of the brain is needed for the simulation. The used slice of the brain is shown in Figure 3.3. The target area, which is indicated in green with the number 21 in the figure, is the angular gyrus. The penalty area is indicated in red with the number 22 and represents lateral occipital cortex, superior division.

Unlike a homogeneous medium in a heterogeneous medium we must also take into account the minimum distance of the transducer from the head. The transducer must not be placed inside the head or touch it in any way. Therefore, the number of points representing the head and transducer overlap is added to the fitness function.

Also, to make it easier for optimization algorithms to find better solutions, a Gaussian curve, which can be seen in Figure 3.4, was applied to the target mask. The Gaussian curve has a size 27x27 pixels and a value of sigma equal to 5. The curve consists of values in the range (0, 1) and causes the values in the middle to have more weight than the values at the edges of the target area. This means that if two cases occur:

1. the ultrasound beam hits the edge of the target but hits more points,
2. the ultrasound beam hits the center of the target but hits fewer points,

the second case is evaluated better than the first one. This Gaussian curve was created for this particular target mask and all values were multiplied by 100. If the target mask contained multiple targets, it would be necessary to overlay each target separately with a Gaussian curve. Equation for calculating the fitness function for a heterogeneous medium:

$$fitness = \left(\sum t - \sum t_{hit}\right) + \left(\sum p * \sum p_{hit}\right) + \sum head_{hit} \tag{3.4}$$

The penalty area has the greatest weight on the value of the fitness function, then the number of hit points and only then the situation that the transducer overlays the head. It would be possible to add some constant that most penalizes the solution where the transducer overlays the head.

[4] https://neurovault.org/images/1705/

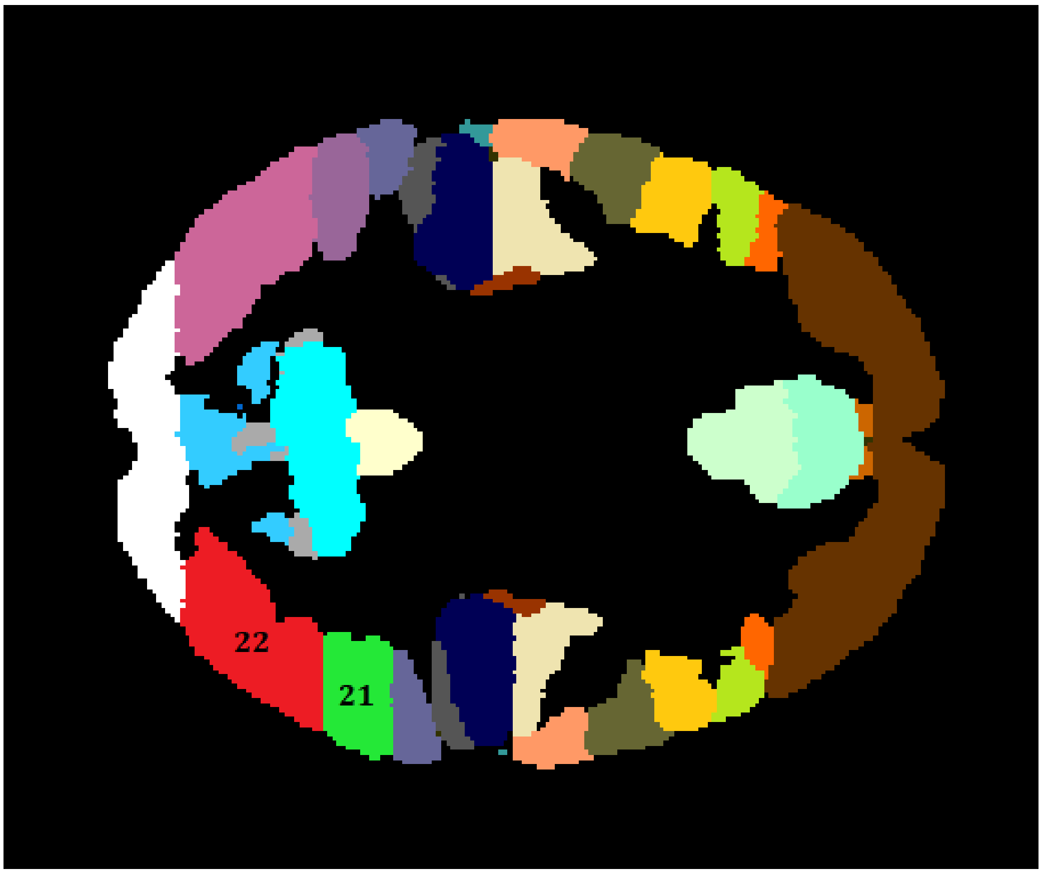


Figure 3.3: Diagram showing slice of the brain used in simulations.

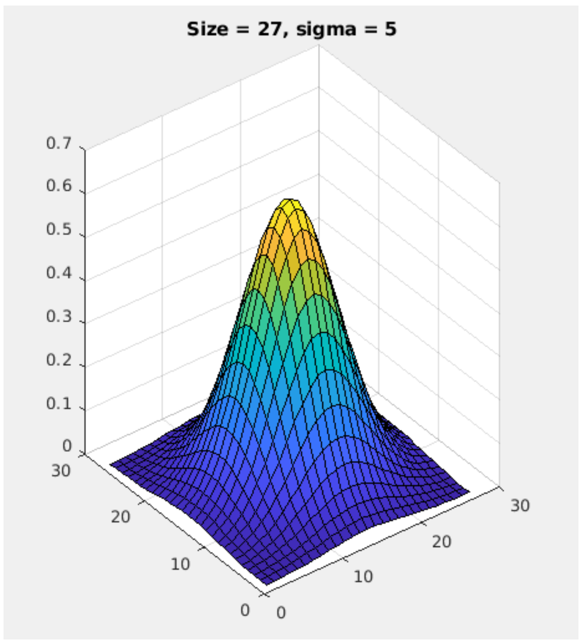


Figure 3.4: Gaussian curve used on the target mask.

Chapter 4

Implementation

This chapter describes the implementation of a program that uses optimization algorithms such as genetic algorithm, CMA-ES and simulated annealing to find the best solution for the position of the ultrasound transducer. The division and structure of individual files are also explained here. Next, the individual parts of the code are described together with an explanation of their functionality.

4.1 Language and tools

All source codes are written in MATLAB, which is a programming environment focusing on numerical calculations, modelling, algorithm design, data analysis and more. A version of MATLAB R2020b was used to create scripts in this thesis.

MATLAB also contains Global Optimization Toolbox that provides functions for finding global solutions allowing to perform design optimization tasks. The toolbox includes pattern search, genetic algorithm, particle swarm, simulated annealing and others. For this work, a genetic algorithm[1] and simulated annealing[2] were used from this toolbox. Along with these two algorithms, the CMA-ES algorithm is also used. However, the implementation of CMA-ES was taken over from the Internet[18] and adapted for use to optimize the position of the transducer.

Another tools which were used are Image Processing Toolbox, MATLAB Compiler, k-Wave Package and NIfTI Toolbox.

4.2 Structure

The implementation is split into two parts. The first part contains source codes for working with a homogeneous medium, which were used in the first half of the academic year. The second part contains source codes for working with a heterogeneous medium, which were used in the second half of the academic year. The scripts are saved in the `Sources` folder, which is divided into two subfolders, `Homogeneous` and `Heterogeneous`. The scripts in `Homogeneous` folder are divided into three subfolders named after the algorithm that is used for the optimization. The full hierarchy of `Homogeneous` folder is shown below.

[1] https://www.mathworks.com/help/gads/genetic-algorithm.html
[2] https://www.mathworks.com/help/gads/simulated-annealing.html

```
Homogeneous
├── GA
│   ├── fitness_func.m
│   ├── GA_main.m
│   └── gaoutputfcn.m
├── SA
│   ├── sa_fitness_func.m
│   ├── SA_main.m
│   └── saoutputfcn.m
└── CMA-ES
    ├── cmaes_fitness_func.m
    └── CMAES_main.m
```

The scripts in the `Heterogeneous` folder were not divided into more subfolders, because each file with the optimization algorithm shares the same fitness function implemented in file `fitness_func_heterogeneous.m` (case with one sonication) or `fitness.m` (case with four sonications). Fitness function files use both `.nii` files and `skull2medium.m` file.

```
Heterogeneous
├── CMAES_main_hetero.m
├── CT.nii
├── fitness.m
├── fitness_func_heterogeneous.m
├── GA_main_hetero.m
├── gaoutputfcn_hetero.m
├── HarvardOxford-cort-maxprob-thr25-1mm.nii
├── SA_main_hetero.m
├── saoutputfcn_hetero.m
└── skull2medium.m
```

4.3 Optimization with GA

Optimization with genetic algorithm consists of 3 MATLAB files. The files used for optimization in a homogeneous medium are: `GA_main.m`, `fitness_func.m` and `outputfcn.m`.

The files used for optimization of the position of the transducer in a heterogeneous medium are: `GA_main_hetero.m`, `fitness_func_heterogeneous.m` and `gaoutputfcn_hetero.m`.

The main file is `GA_main.m` or `GA_main_hetero.m` from which the genetic algorithm can be executed. The main file for a homogeneous and heterogeneous medium is only slightly different. In both cases, we must first set the lower and upper boundaries for optimized values and then set all GA parameters, like selection and crossover operator, mutation probability, the size of the population, the number of generation, etc.

We simply define the lower and upper bounds as a vector of five values, as we optimize the five values that are: x and y values of the position of the arc, x and y values of the beam focus point, and radius. The grid size for a homogeneous medium is 128x128 (from 1 to 128), so the lower limit is 1 and upper limit is 128.

```
% grid size is 128x128, dx = 5e-4 [m], domain size = 64x64 [mm]
% arcPosX arcPosY focusPosX focusPosY radius
lb = [2 2 1 1 21]; % vector of lower bounds
ub = [127 127 128 128 128]; % vector of upper bounds
```

As we can see, the fifth value in lower bounds is different from the others. This is because the size of the radius of curvature cannot be lower than 21, because the value of radius $* 2$ has to be greater than the aperture diameter. The value of aperture diameter is always 41, so the value of the radius has to be minimally 21 because $21 * 2 = 42$ which is greater than 41. The first and second value in lower and upper bounds is different too because we can't allow the centre of the transducer to be at the end of the grid.

For heterogeneous medium, the grid size is 257x257 (from -128 to 128), so the lowest limit is -128 and upper limit is 128. Function `addArcElement`, which is used to add arc-shaped element to the grid, takes input parameters in meters. This is why the values in `lb` and `ub` are decimal.

```
% grid size is 257x257, dx = 1e-3 [m], domain size = 257x257 [mm]
% arcPosX arcPosY focusPosX focusPosY radius
lb = [-0.127 -0.127 -0.128 -0.128 0.021]; % vector of lower bounds
ub = [0.127 0.127 0.128 0.128 0.128]; % vector of upper bounds
```

An example of what the initial parameter settings in GA look like and what we set up is shown below.

```
options = optimoptions('ga', ...
    'PopulationSize', 60, ...
    'Generations', 30, ...
    'FitnessLimit', 0, ...
    'CrossoverFcn', {@crossoverintermediate, 0.8}, ...
    'SelectionFcn', {@selectiontournament, 10}, ...
    'MutationFcn', {@mutationadaptfeasible, 0.5}, ...
    'Display', 'diagnose', ...
    'OutputFcn', @gaoutputfcn);
```

Once we have set the individual parameters, we can call the function `ga` from the Global Optimization Toolbox. An example of a function call is shown below.

```
[x,fval] = ga(@(x) fitness_func(x(1),x(2),x(3),x(4),x(5)),5, ...
             [],[],[],[],lb,ub,[],options);
```

The `gaoutputfcn.m` and `gaoutputfcn_hetero.m` files are used to plot the best solution found after completing the GA calculation, exceeding the maximum number of iteration, or finding the optimal solution with a fitness value of 0.

4.4 Optimization with CMA-ES

Optimization with CMA-ES contains two files. When working with a homogeneous medium, these files are called `cmaes_main.m` and `cmaes_fitness_func.m`. On the other hand, when we are working with a heterogeneous medium, these files are called `cmaes_main_hetero.m` and `fitness_func_heterogeneous.m`. The main file, in which the whole logic of the algorithm is implemented, is `cmaes_main.m` or `cmaes_main_hetero.m`. The two files are almost identical as the medium only affects the calculation of the fitness function. The `cmaes_fitness_func.m` and `fitness_func_heterogeneous.m` files are used to calculate the fitness function, otherwise called the cost function.

At the beginning of the main file, we need to define problem settings and CMA-ES settings. The preview of the problem settings we need to define when working with a homogeneous medium is shown below.

```
CostFunction = @(x)cmaes_fitness_func(x(1),x(2),x(3),x(4),x(5));

nVar = 5; % Number of Unknown (Decision) Variables
VarSize = [1 nVar]; % Decision Variables Matrix Size

VarMin = 1; % Lower Bound of Decision Variables
VarMax = 128; % Upper Bound of Decision Variables
```

The preview of problem settings for a heterogeneous medium differs in the name of the fitness function and in the lower limit.

```
CostFunction = @(x)fitness_func_heterogeneous(x(1),x(2),x(3),x(4),x(5));

nVar = 5; % Number of Unknown (Decision) Variables
VarSize = [1 nVar]; % Decision Variables Matrix Size

VarMin = -128; % Lower Bound of Decision Variables
VarMax = 128; % Upper Bound of Decision Variables
```

The preview of CMA-ES settings is shown below. These settings were defined by the author of the algorithm and I did not change them directly. However, they depend on some of the problem settings such as `nVar`, `VarMin` and `VarMax`, which were defined according to the needs of the problem.

```
% Maximum Number of Iterations
MaxIt = 20;

% Population Size (and Number of Offsprings)
lambda = (4+round(3*log(nVar))*10);

% Number of Parents
mu = round(lambda/2);
```

```
% Parent Weights
w = log(mu+0.5)-log(1:mu);
w = w/sum(w);

% Number of Effective Solutions
mu_eff = 1/sum(w.^2);

% Step Size Control Parameters (c_sigma and d_sigma);
sigma0 = 0.3*(VarMax-VarMin);
cs = (mu_eff+2)/(nVar+mu_eff+5);
ds = 1+cs+2*max(sqrt((mu_eff-1)/(nVar+1))-1,0);
ENN = sqrt(nVar)*(1-1/(4*nVar)+1/(21*nVar^2));

% Covariance Update Parameters
cc = (4+mu_eff/nVar)/(4+nVar+2*mu_eff/nVar);
c1 = 2/((nVar+1.3)^2+mu_eff);
alpha_mu = 2;
cmu = min(1-c1,alpha_mu*(mu_eff-2+1/mu_eff)/((nVar+2)^2+alpha_mu*
       mu_eff/2));
hth = (1.4+2/(nVar+1))*ENN;
```

This is followed by initialization and then the algorithm loop itself, which is executed as many times as a maximum number of iterations was defined or until an optimal solution with a fitness value of 0 is found. After the loop, the best solution is shown.

4.5 Optimization with SA

Optimization with simulated annealing consists of 3 files. For a homogeneous medium, these are `SA_main.m`, `sa_fitness_func.m` and `saoutputfcn.m` files. When working with a heterogeneous medium, these are `SA_main_hetero.m`, `fitness_func_heterogeneous.m` and `saoutputfcn_hetero.m` files. The main file, from which the simulated annealing can be executed, is `SA_main.m` or `SA_main_hetero.m`. These two files are only slightly different.

First of all, we need to set the lower and upper boundaries, an initial individual and than set all SA parameters, like the number of iterations, annealing function, initial temperature, etc. We simply define the lower and upper bounds as a vector of five values, as we optimize the five values. The setting of the upper limit and the lower limit in SA is the same as shown in the GA implementation. The initial individual is defined by the user randomly as a vector of five values and may look as follows:

```
% initial individual (a homogeneous medium)
init = [20 40 100 60 100];

% initial individual (a heterogeneous medium)
init = [-0.075 0.075 0 0 0.100];
```

An example of what the initial parameter settings in SA look like and what we set up is shown below.

```
options = optimoptions('simulannealbnd', ...
    'MaxIterations', 100, ...
    'InitialTemperature', 200, ...
    'ObjectiveLimit', 1, ...
    'AnnealingFcn', 'annealingboltz', ...
    'OutputFcn', @saoutputfcn);
```

Once we have set the individual parameters, we can call the function `simulannealbnd` from the Global Optimization Toolbox. An example of a function call is shown below.

```
[x,fval] = simulannealbnd(@(x) sa_fitness_func(x(1),x(2),x(3), ...
                           x(4),x(5)),init,lb,ub,options);
```

The `saoutputfcn.m` and `saoutput_hetero.m` files are used to plot the best solution found after the SA calculation is completed, the maximum number of iteration was exceeded or the optimal solution with a fitness value of 0 was found.

4.6 Implementation of fitness function

The implementation of the fitness function for each algorithm is almost the same. Since different functions are used to simulate ultrasound propagation in the medium, the implementation of the fitness function for a homogeneous medium differs from the implementation for a heterogeneous medium.

4.6.1 Homogeneous medium

The fitness function for CMA-ES contains only one difference from the implementation of the fitness function for SA and GA. Since the lower and upper boundary for CMA-ES was set as only one number and not as an array of five values, because the CMA-ES implementation taken from the Internet uses this one value to calculate other variables and therefore the boundaries could not be adjusted to the array of values as in the fitness function for GA and SA, it could happen that the radius will be less than 21. Therefore, in the fitness function for CMA-ES, a condition is added which, if the radius is smaller than 21, sets the radius to 21.

For the algorithm to know what should hit and what must not hit, we need to load the target mask and the penalty mask. Then, we count all target points and all penalty points, so we can use them later to calculate fitness.

```
targetMask = readmatrix('rotated-rice.txt');
penaltyMask = readmatrix('rotated-rice-penalisation.txt');

pointsTarget = sum(sum(targetMask,2));
pointsPenalty = sum(sum(penaltyMask,2));
```

To create transducer, which is represented as a matrix of the source amplitude at each grid point, we can use the function `makeArc([grid size x, grid size y], arcPos, radius, diameter, focusPos)`. Grid size is always 128x128 and diameter is 41. The remaining parameters (`arcPos`, `focusPos` and `radius`) are our optimized variables. Then, AFP is used to simulate ultrasound propagation and calculate the pressure in the grid.

```
% generate the source geometry
amp = makeArc([128, 128], arcPos, radius, diameter, focusPos);
pressure = acousticFieldPropagator(amp, 0, 0.5e-3, 500e3, 1500);
```

When we get a matrix of the complex pressure field at each grid point in steady-state, we need to convert it from pascals to decibels. Then we create a threshold value, which is the maximum value minus 6 dB, leaving only the values that are above the threshold value.

```
pRef = 2.0 * 1e-5; % the pressure of the smallest sound we can hear

% formula to determine the sound pressure level (SPL) in decibels
y = 20 .* log10(abs(pressure) ./ pRef);

% find maximum
maxDb = max(y(:));

% threshold = maximum - 6dB
thresholdDb = maxDb - 6;

finalMatrix = y > thresholdDb;
```

Now, when we have a matrix of boolean values, true or false, which shows whether the point of the grid was hit or not, we can combine it with both, target and penalty mask, to find out how many points of the target mask was hit and how many points of the penalty was hit.

```
% combine targetMask and pressure values
resTarget = targetMask .* finalMatrix;

% combine penaltyMask and pressure values
resPenalty = penaltyMask .* finalMatrix;

sumTarget = sum(sum(resTarget,2));
sumPenalty = sum(sum(resPenalty,2));
```

Then the value of the fitness function, called the score in the code, is calculated as addition of the difference between the points we wanted to hit and did not hit them, and the multiplication of the points we did not want to hit and hit them.

```
% compute score
score = (pointsTarget - sumTarget) + (pointsPenalty * sumPenalty);
```

4.6.2 Heterogeneous medium

The fitness function for working with a heterogeneous medium is the same for all three algorithms. In the beginning, it is necessary to define several parameters, such as transducer (source) properties, parameters of simulation and image. The `source_f0` value indicates the ultrasound frequency and `source_mag` is the ultrasound magnitude. The simulation properties are used to determine the simulation time and the number of periods to be recorded.

```
% source properties
source_f0 = 600e3; % [Hz]
source_mag = 40e3; % [Pa]
bowl_diameter = 41e-3; % bowl aperture diameter [m]
```

```
% simulation properties
round_trips = 1;
cfl = 0.25;
record_periods = 1;

% image properties
img_slice_idx = 92;
img_pad = [38 38 20 20];
```

At the beginning we read data from the `CT.nii` file which contain a CT scan of the head. From the `HarvardOxford-cort-maxprob-thr25-1mm.nii` file the target and penalty mask is obtained. After loading the `.nii` files, we have a scan of the whole head, but since we work in 2D, we only need one slice. From the slice of CT scan of the head, we create a medium, a skull mask and a head mask.

```
% load the CT data (requires the nifti toolbox to be on the path)
nii = load_nii('CT.nii');
nii2 = load_nii('HarvardOxford-cort-maxprob-thr25-1mm.nii');

% choose a slice
img_slice = single(nii.img(:, :, img_slice_idx));
img_slice2 = single(nii2.img(:, :, img_slice_idx));

% convert to acoustic properties
[medium, skull_mask, head_mask] = skull2medium(img_slice, [], source_f0,
              'IncludeAir', false, 'ReturnValues', 'acoustic-linear');
```

The used slice of the brain shown in Figure 3.3 contains a lot of segments. We need to extract only the wanted parts. The target area is indicated with the number 21 and the penalty area is indicated with the number 22. Subsequently, a Gaussian curve of size 27x27 with sigma 5 is created, which is overlaid through the target mask. If the target mask was different, a different Gaussian curve would also have to be created.

```
% target mask (segment 21), penalty mask (segment 22)
targetMask = (img_slice2 == 21);
penaltyMask = (img_slice2 == 22);

% create gaussian curve
g1 = Gaussian_filter(27,5);
zeroMatrix2 = zeros(257,257);
zeroMatrix2(170:196,80:106) = g1;
```

Next, we have to create a grid, define a source and a sensor mask. The source is defined by four values, the position of the transducer `source_pos`, direction of the beam `focus_pos`, aperture diameter `bowl_diameter` and curvature of arc `bowl_roc`.

```
% add arc shaped element
karray.addArcElement(source_pos, bowl_roc, bowl_diameter, focus_pos);
```

Since it is obvious that the transducer cannot be in the head, we calculate the number of points in which the transducer and the head overlap. We will later add this amount to the fitness function to force evolution to avoid such solutions. In img_slice everything that is not a head has a value of -1024 so we need to get only values that represent a head.

```
restricted_area = img_slice > -1024;
overlaping = restricted_area .* source.p_mask;
sumOverlap = sum(sum(overlaping, 2));
```

Then we can run a simulation of ultrasound propagation with which we get time varying pressure recorded at the sensor positions given by sensor.mask. From this data the amplitude has to be extracted.

```
sensor_data = kspaceFirstOrder2D(kgrid, medium, source, sensor, ...
              'PMLInside', false, ...
              'PMLSize', 'auto', ...
              'DataCast', 'single', ...
              'PlotScale', [-1, 1] * source_mag * 2, ...
              'DisplayMask', skull_mask);

% extract amplitude from the sensor data
[p_amp, p_phase] = extractAmpPhase(sensor_data, 1/kgrid.dt, source_f0,
              'Dim', 2, 'Window', 'Rectangular', 'FFTPadding', 1);
```

After obtaining the amplitude, similarly to a homogeneous medium, the pressure amplitude is combined with a target mask and a penalty mask. At the end, the score of the individual is calculated, to which, in the case of a heterogeneous medium, the sum of the points where the transmitter and the head overlap is added.

```
% compute score
score = (pointsTarget2 - sumTarget) + (pointsPenalty * sumPenalty) +
       sumOverlap;
```

4.7 Usage

The scripts can be run from the command line or in the MATLAB IDE. Each optimization algorithm has its own main file with the name *_main.m (for homogeneous medium) or *_main_hetero.m (for heterogeneous medium), where * represents the name of the optimization algorithm (GA, SA, CMA-ES). To run the optimization using GA, run the GA_main.m file.

When performing the experiments with a heterogeneous medium, it was necessary to run scripts on the Barbora cluster, due to their time complexity. However, the cluster did not contain the required version of MATLAB R2020b. The highest version of MATLAB, located in Barbora, is the R2015b version. However, this version is not compatible with scripts written in version R2020b, as some functions did not exist in version R2015b or were called otherwise. Since it was not possible to install a newer version of MATLAB on the cluster, it was necessary to use the MATLAB Runtime. The MATLAB Runtime is a standalone set of shared libraries that enables the execution of compiled MATLAB applications or components. In other words, it is possible to run compiled MATLAB applications or components without installing MATLAB. To install MATLAB Runtime on Barbora cluster, it is necessary to download MATLAB Runtime installer[3] based on the

[3]https://www.mathworks.com/products/compiler/matlab-runtime.html

version of MATLAB (R2020b in this case). Then it can be installed on Barbora cluster by unziping and running the installer:

```
unzip MATLAB_Runtime_R2020b_glnxa64.zip
cd MATLAB_Runtime_R2020b_glnxa64
./install
```

Detailed description how to install and configure MATLAB Runtime can be found at their web page[4]. When MATLAB Runtime is installed, we can run any executable MATLAB file on Barbora cluster.

For this purpose, three scripts were created, for compiling MATLAB scripts to executable files, which can be executed on cluster. Each optimization algorithm has its own compiling script, with name `create_binary_*.m`, where * represents the name of the algorithm. For example, after running script `create_binary_SA.m`, executable file `SA_main_hetero` is created. To run the executable file, just type in the terminal:

```
./SA_main_hetero
```

However, it is necessary to have the `HarvardOxford-cort-maxprob-thr25-1mm.nii` and `CT.nii` files in the same folder as the executable file, because the executable file uses these files and will not run without them. For example the final folder for running SA optimization looks like this:

```
SA_binaries
 ├─ CT.nii
 │
 ├─ HarvardOxford-cort-maxprob-thr25-1mm.nii
 │
 └─ SA_main_hetero
```

It may happen that even though the MATLAB Runtime is installed, it will not be possible to run the executable file. One reason may be that the executable file does not know the path to the shared libraries. Then, a similar error can occur:

```
./SA_main_hetero: error while loading shared libraries:
                  libmwlaunchermain.so: cannot open shared object file:
                  No such file or directory
```

To fix this, it is necessary to link some shared libraries. This is done using the commands below. Instead of `/path/to/`, there must be entered the actual path to the MATLAB Runtime, depending on where it was installed.

```
export LD_LIBRARY_PATH=/path/to/MATLAB_Runtime/v99/runtime/glnxa64:
                      /path/to/MATLAB_Runtime/v99/bin/glnxa64:
                      /path/to/MATLAB_Runtime/v99/sys/os/glnxa64:
                      /path/to/MATLAB_Runtime/v99/extern/bin/glnxa64/

export LD_PRELOAD=/path/to/MATLAB_Runtime/v99/bin/glnxa64/glibc-2.17_
shim.so
```

An example of a simulation run for GA using a homogeneous medium is shown in Appendix A. An example of a simulation run for SA using a heterogeneous medium is shown in Appendix B.

[4]https://www.mathworks.com/help/compiler/install-the-matlab-runtime.html

Chapter 5

Experiments with a homogeneous medium

This chapter describes the performed experiments with a homogeneous medium. The size of the computational grid for the experiments was 128 x 128. The sound speed of the medium was 1500 m/s, which is equal to the sound speed in seawater. Experiments were performed on a laptop with Windows 10, processor Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and graphical card NVIDIA GeForce GTX 1050 Ti.

For the experiments, three test benchmarks, which can be seen in Figure 5.1, were created: a rice grain (on the left side), a rotated rice grain (in the middle) and a rotated rice grain with a penalty (on the right side).

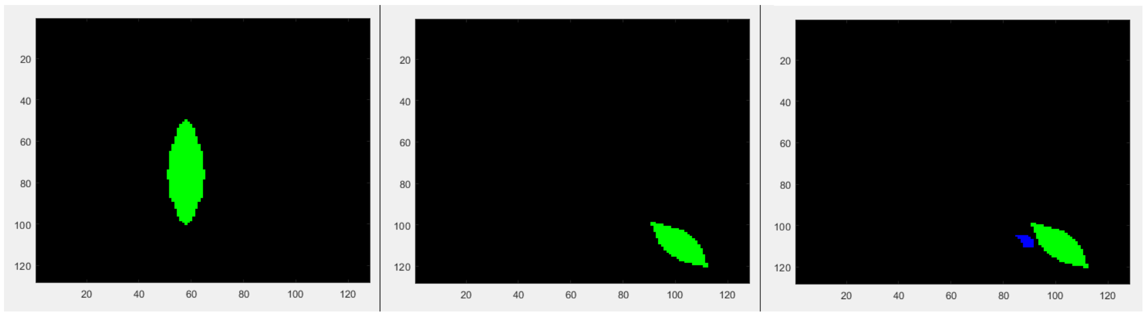


Figure 5.1: Three benchmarks

The experiments were performed with various algorithms, namely genetic algorithms, CMA-ES and simulated annealing. Each algorithm was tested on all three benchmarks. Different parameters were tested for each algorithm. Each test consists of 20 runs.

5.1 Parameters

Each one of the algorithms used for optimization must have some initial parameters set.

Genetic algorithms

To experiment with GA, we need to know the population size, the number of generations, elitism, selection and crossover operator and the probability of mutation. The genetic algorithm was started with two different parameter sets. These sets are shown in Table 5.1:

Table 5.1: Parameters of Genetic Algorithm

	1st parameter set	2nd parameter set
Population	50	60
Generations	20	30
Elitism	2	2
Selection	Tournament, 4	Tournament, 10
Crossover	Single-point	Intermediate 0.8
Mutation	0.4	0.5

CMA-ES

To experiment with CMA-ES, there was no need to set initial parameters. CMA-ES calculate everything on its own, according to a solved problem. Initial parameters, which algorithm already has, were: number of iterations - 20, population size - 54, number of parents - 27 and sigma0 - 38.1 (approximately 1/3 of grid size).

Simulated annealing

To experiment with SA, we need to know the initial temperature, the initial individual, the cooling step and the number of iterations. The simulated annealing was started with three different parameter sets shown in Table 5.2:

Table 5.2: Parameters of Simulated Annealing

	1st parameter set	2nd parameter set	3rd parameter set
Iterations	100	100	100
Temperature	100	100	200
Initial individual	20, 20, 100, 100, 100	20, 40, 100, 60, 100	50, 50, 70, 70, 70
Cooling step	$InitialTemperature * 0.95^k$ where k means iteration		

5.2 Experiments with a rice grain

The first testing benchmark is a rice grain. This rice grain consists of 511 points, which we need to hit. In each experiment, the value of the fitness function, the time of calculation and the number of evaluations of the fitness function were examined. In some cases, also, the generation of the found solution and the position of the transducer was examined. The results obtained are statistically given below.

5.2.1 Genetic algorithms

Experiments with GA contained two different sets of parameters as shown in Table 5.1. The results of the first scenario can be seen in Figure 5.2. The results of the second scenario can be seen in Figure 5.3. In the first scenario, no solution was found even once. In the second scenario, only one solution out of twenty runs was found. For this reason, it was not necessary to make graphs for the position of the transducer and the generation of the found solution.

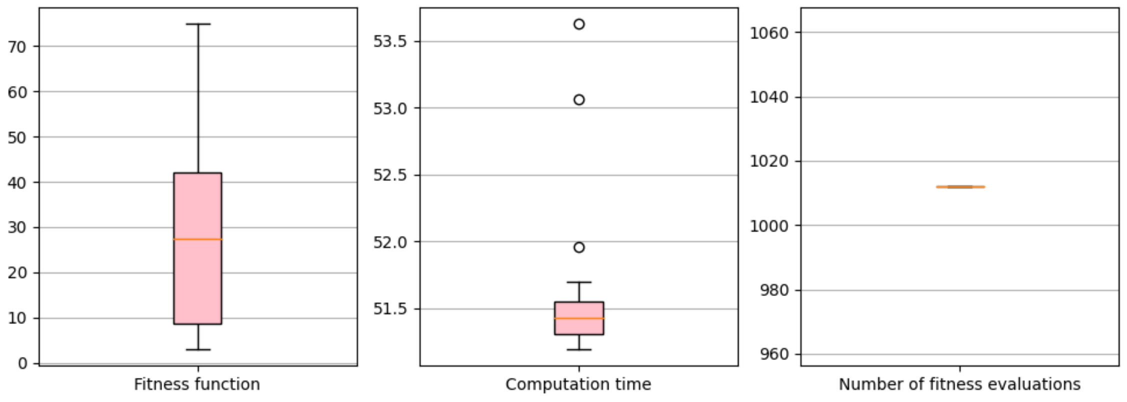


Figure 5.2: Results of GA for a rice grain in the first scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

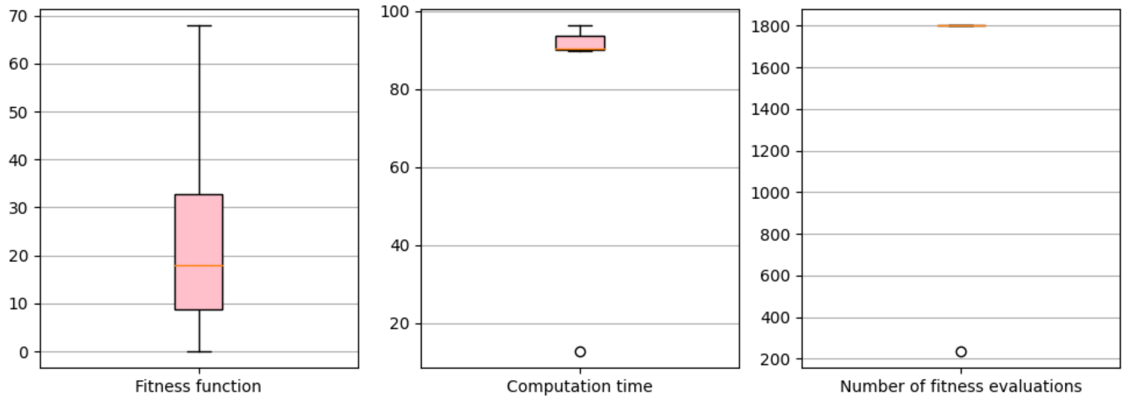


Figure 5.3: Results of GA for a rice grain in the second scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

The worst possible value of the fitness function is 511 because we need to hit 511 points of a rice grain. In the first scenario, a median of the fitness function is 27.5, which belong to the 5.4% of the best solutions. In the second scenario, a median of the fitness function is 18, which belong to the 3.5% of the best solutions.

The average computation time of one run in the first scenario is approximately 52 seconds. In the second scenario it is 88 seconds. The average number of fitness evaluations is 1012 in the first scenario and 1724 in the second scenario (maximum is 1802).

We can see that the second scenario was more successful than the first one because it had a lower average value of fitness function. However, computation time and the number of fitness evaluations were higher in this second scenario, due to more individuals in the population as well as the number of iterations.

5.2.2 CMA-ES

The CMA-ES algorithm was run with only one set of parameters. The results, for the fitness function, computation time and the number of fitness evaluations, obtained during the twenty runs are shown in Figure 5.4.

The algorithm was more successful than GA, as it found a solution fourteen times out of twenty. We can see that the median value of the fitness function is much lower than with genetic algorithms. The median is 0 because the optimal solution was found in 70% of cases. The average computation time of one run is approximately 40 seconds. The average number of fitness evaluations to find a solution is 787.

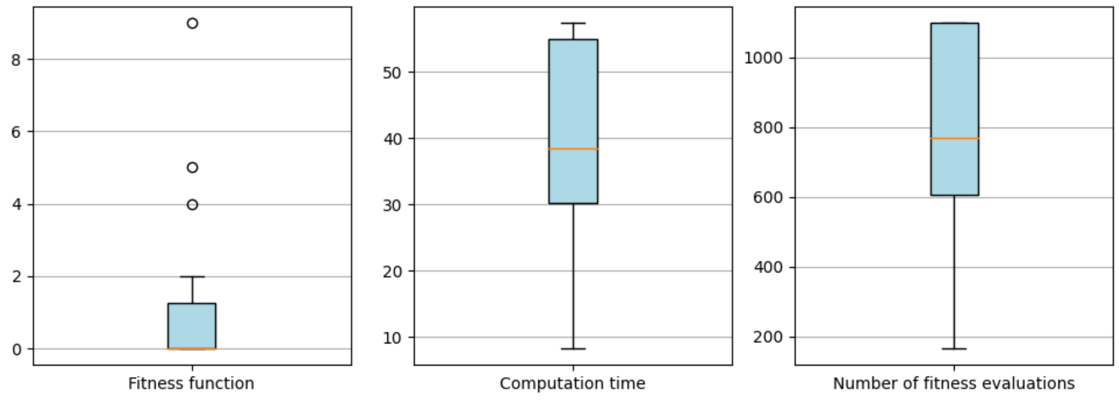


Figure 5.4: Results of CMA-ES for a rice grain (value of fitness function, computation time (in seconds) and number of fitness evaluations)

Since the algorithm has found a solution multiple times, we can also display graphs for the generation of found solution and the position of the transducer. They are shown in Figure 5.5.

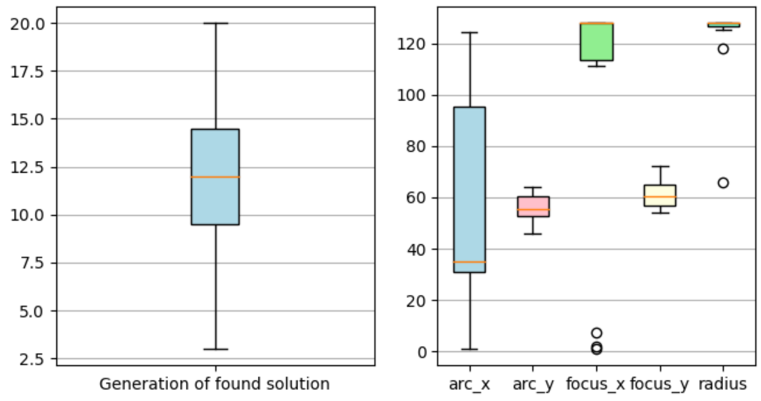


Figure 5.5: Results of CMA-ES for a rice grain (generation of found solution and transducer position)

From the graph for the position of the transducer, we can see that the solutions found by CMA-ES have variability in x axe of arc, but there is almost no variability in the radius size. This is because since there is no penalty anywhere, the radius can be large and hit as much as possible from the target. However, if there was a penalty near the target, the radius would have to be smaller so that the beam did not hit an unwanted area.

5.2.3 Simulated annealing

As we can see from the initial parameters for SA in Table 5.2, the main difference between the three scenarios is their initial individuals. For better understanding of how initial individuals for this scenario (a rice grain) look like, they are shown in Figure 5.6. These different initial individuals were chosen in this way so that we could observe how the algorithm can deal with different starting situations. The results are displayed below.

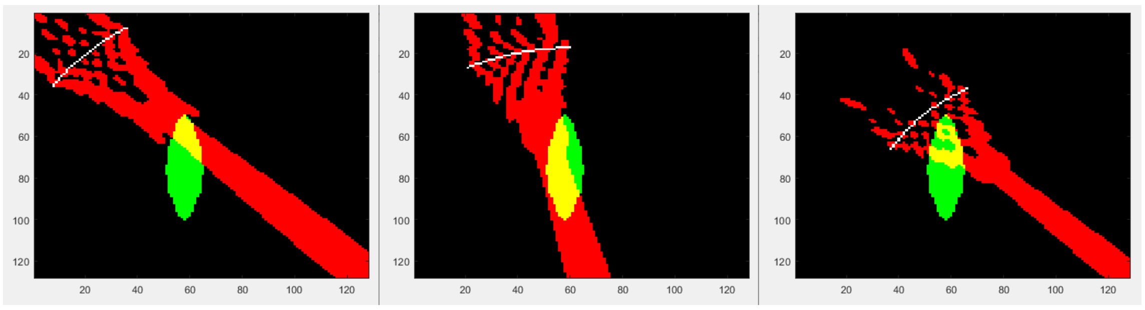


Figure 5.6: Three initial options for rice grain

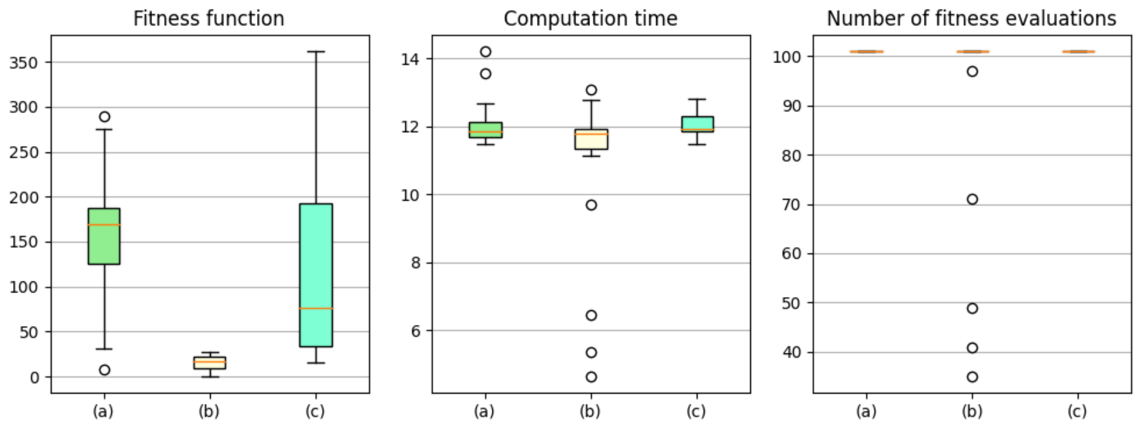


Figure 5.7: Results of SA for a rice grain (value of fitness function, computation time (in seconds) and number of fitness evaluations)

In Figure 5.7, the results of all three scenarios are displayed. The first scenario is in graphs shown as (a), second scenario as (b) and third as (c). The median value of fitness function in the first scenario is 168.5 which belong to 33% of the best solutions, in second scenario only 17, which belong to the 3.3% of the best solutions and in third scenario 75.5 which belong to 14.8% of the best solutions. That means that the second scenario has the best results. In this scenario, also the global minimum was found a few times. For this reason, only the results from this scenario are displayed in the next box plot in Figure 5.8.

The average computation time of one run in the first and the third scenario is 12 seconds. In the second scenario the average computation time was lower by one second, because the global minimum was a few times found sooner than in the last iteration. The average number of fitness evaluations in the first and third scenario is 101, which is the maximum. In the second scenario it is only 91.

From the graph for the position of the transducer, we can see that the solutions found by SA are quite similar, unlike CMA-ES where solutions more variate.

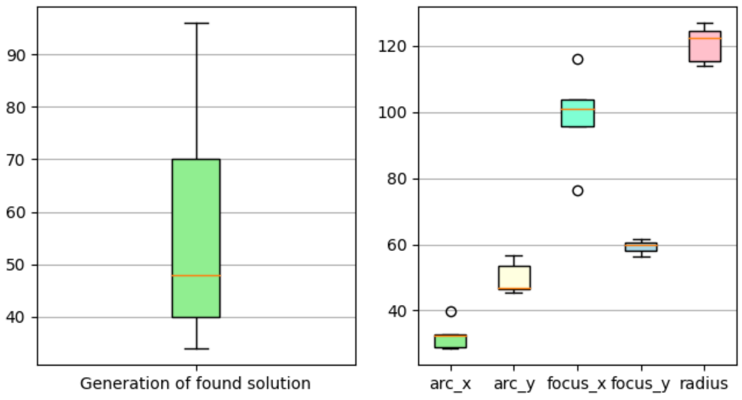


Figure 5.8: Results of SA for a rice grain (generation of found solution and transducer positions)

5.3 Experiments with a rotated rice grain

The second testing scenario is a rotated rice grain. This rotated rice grain consists of 240 points that we need to hit. It means, that the worst value of the fitness function can be 240. In each experiment, the value of fitness, the calculation time and the number of fitness evaluations were examined. Obtained results are statistically shown below.

5.3.1 Genetic algorithms

The results of the first scenario can be seen in Figure 5.9. In the first scenario, the solution was found fourteen times out of twenty, which means that the median of the value of fitness function is 0. For this reason, also the generation of a found solution and transducer position were display to the graph and can be seen in Figure 5.10. The average computation time of one run in the first scenario is approximately 28 seconds. The average number of fitness evaluations to found the solution in the first scenario is 549 (maximum is 1012).

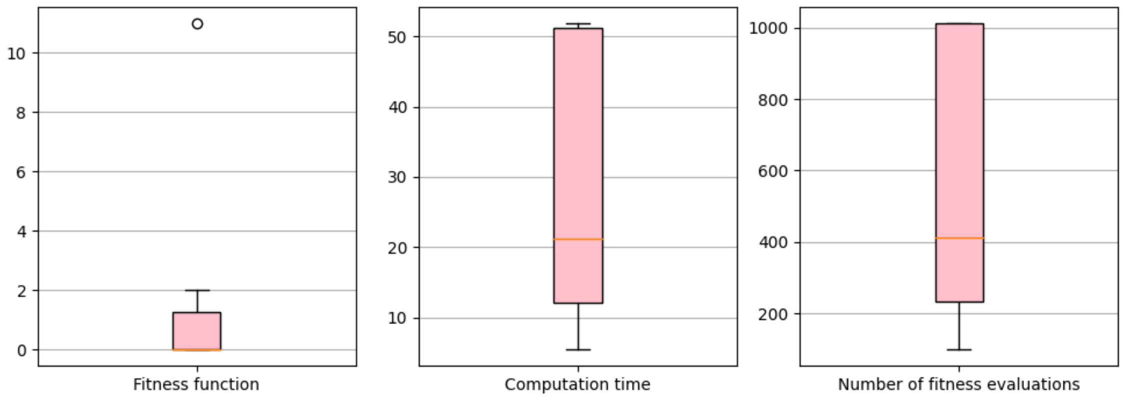


Figure 5.9: Results of GA for a rotated rice grain in the first scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

The results of the second scenario can be seen in Figure 5.11. The second scenario was more successful and found solution fifteen times out of twenty, i.e., with a success rate of 75%. The median of the value of the fitness function is 0. Additional information about the generation of found solution and transducer position are shown in Figure 5.12. The

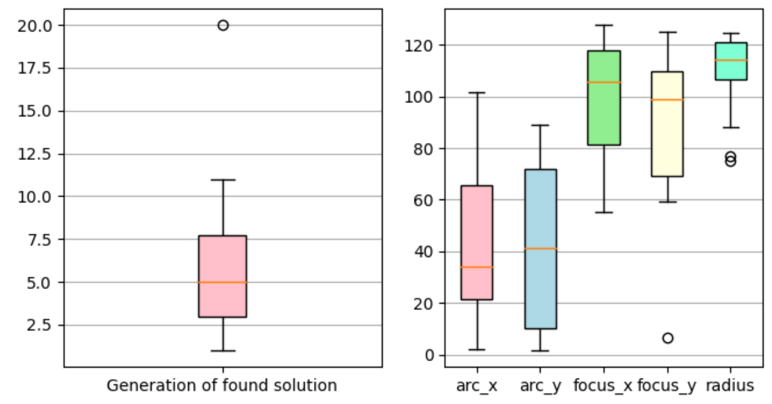


Figure 5.10: Results of GA for a rotated rice grain in the first scenario (generation of found solution and transducer position)

average computation time of one run in the second scenario is approximately 44 seconds. The average number of fitness evaluations to found the solution is 860 (maximum is 1802).

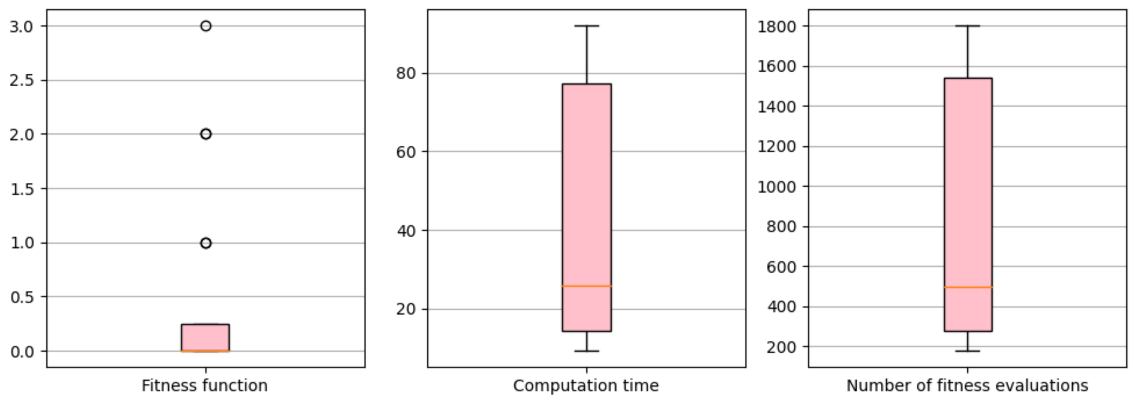


Figure 5.11: Results of GA for a rotated rice grain in the second scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

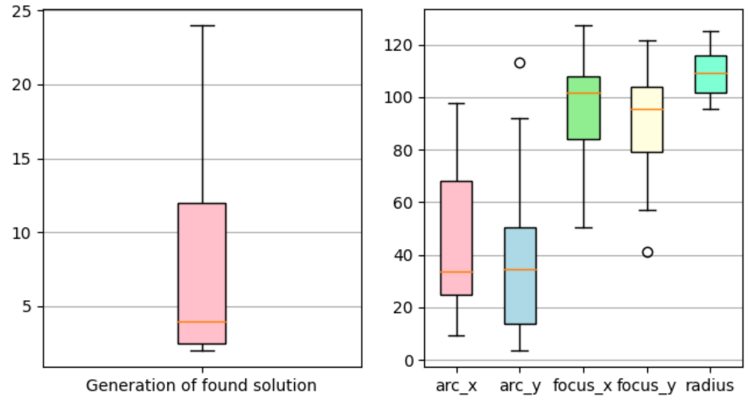


Figure 5.12: Results of GA for a rotated rice grain in the second scenario (generation of found solution and transducer position)

From both graphs for the transducer position, we see that there is not only one solution, but there are many of them that differ from each other and GA was able to find them.

5.3.2 CMA-ES

CMA-ES was again started with only one parameter set. The results, for the value of fitness function, computation time (in seconds) and the number of fitness evaluations, obtained during the twenty runs are shown in Figure 5.13.

The algorithm was successful in nineteen runs out of twenty, i.e. with a success rate of 95% and again, we can see that the CMA-ES algorithm was more successful than GA. The worst value of the fitness function found by CMA-ES was 2, which belongs to 0.8% of best solutions, so we can say CMA-ES was quite successful in this scenario. The average computation time of one run is approximately 22 seconds. The average number of fitness evaluations to found the solution is 435 (maximum is 1100).

The results of the generation of found solution and transducer position are shown in Figure 5.14. We can see that the solutions found by CMA-ES differ from each other, which means that CMA-ES did not always find similar solutions but was able to examine the whole domain of a function and find many different solutions.

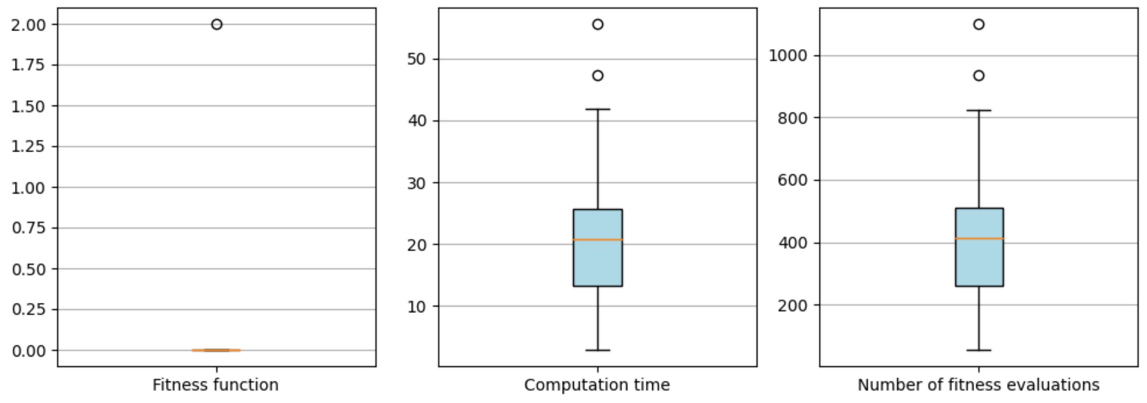


Figure 5.13: Results of CMA-ES for a rotated rice grain (value of fitness function, computation time (in seconds) and number of fitness evaluations)

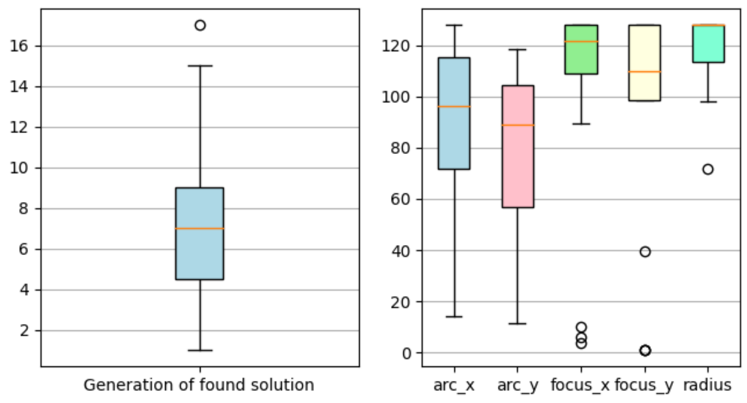


Figure 5.14: Results of CMA-ES for a rotated rice grain (generation of found solution and transducer position)

5.3.3 Simulated annealing

Simulated annealing for rotated rice grain benchmark has again three different initial individuals. For better understanding, how initial individuals look like, we can see all of them in Figure 5.15.

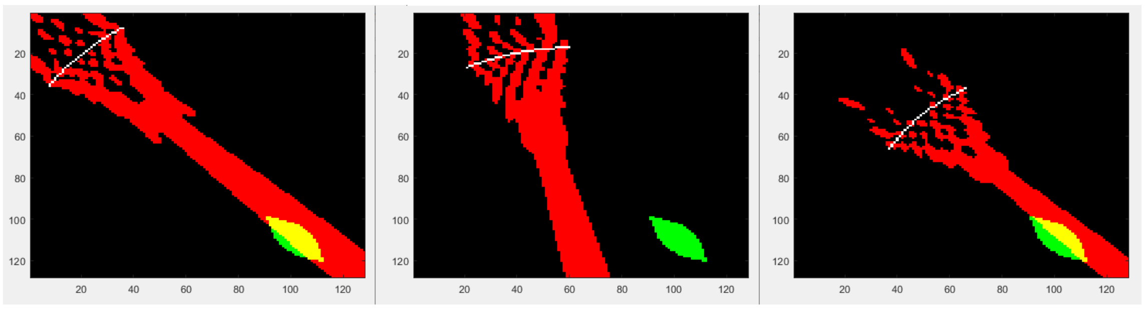


Figure 5.15: Three initial options for rotated rice grain

In Figure 5.16 results of all three scenarios are displayed. The first scenario is in graphs shown as (a), second scenarios as (b) and third as (c). As we can see, the first scenario has the best results out of all three. In this scenario, the global minimum was found every time. In the second scenario solution was found only two times out of twenty (success rate 10%) and in the third scenario twelve times out of twenty (success rate 60%). The median value of fitness function for the first and third scenario was 0, for the second scenario it was 240. The second scenario was difficult for the algorithm because the beam of an initial individual did not hit any part of the target. This means that the algorithm guessed completely blindly and it was difficult for it to decide which side to go, whether to the right or the left.

The average computation time of one run in the first scenario is 1 second, in the second scenario 11 seconds and in the third scenario 7 seconds. The average number of fitness evaluations to found the solution is 6 in the first scenario, 94 in the second scenario and 56 in the third scenario (maximum is 101).

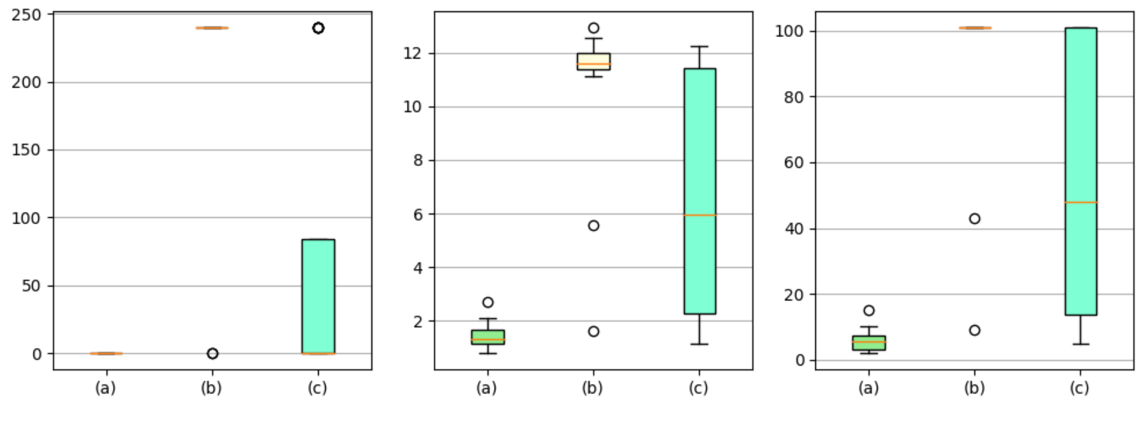


Figure 5.16: Results of SA for a rotated rice grain (value of fitness function, computation time (in seconds) and number of fitness evaluations)

Next graph shows the generation of found solution and the transducer position for the first scenario, because it was the best scenario out of all three, and is shown in Figure 5.17.

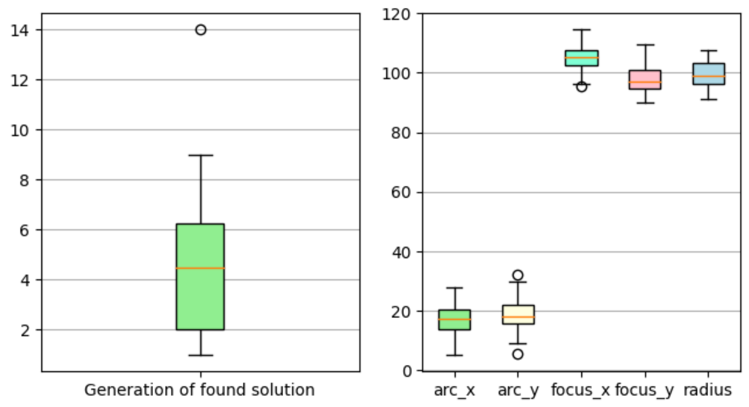


Figure 5.17: Results of SA for a rotated rice grain in the first scenario (generation of found solution and transducer position)

We can see, that solutions found by SA in the first scenario are quite similar. It is because the initial individual has good starting condition as it hits a big part of the target, so there was no need for the algorithm to explore, just exploit.

Figure 5.18 shows the generation of found solution and the position of the transducer for the third scenario and we can see, that unlike the first scenario, the solutions found by the third scenario more differ from each other. It is because the initial individual did not have as good starting conditions as an individual in the first scenario, so the algorithm had to explore more of the domain.

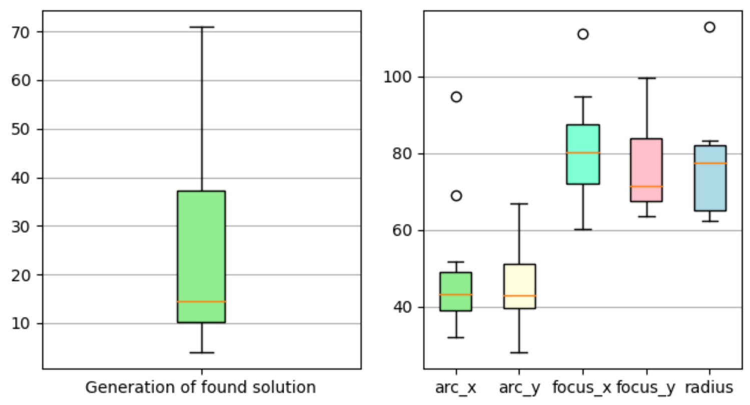


Figure 5.18: Results of SA for a rotated rice grain in the third scenario (generation of found solution and transducer position)

5.4 Experiments with a rotated rice grain with a penalty

The third testing scenario is a rotated rice grain with a penalty. This rotated rice grain with penalty consists of 240 points, which we need to hit and 29 points which we do not want to hit. The worst possible value of fitness function is 1081 and the individual can have this fitness function, if it does not hit any target point (value of fitness, in this case, is 240) and hits all penalty points (value of fitness, in this case, is 29*29 = 841).

In each experiment, the value of fitness, calculation time and the number of fitness evaluations were examined. Also, the generation of the found solution and the position of the transducer was examined in some cases. Obtained results are statistically shown below.

5.4.1 Genetic algorithms

A genetic algorithm for this third benchmark was again started with two different parameter sets. Results of first scenario can be seen in Figure 5.19. As we can see from the graph, the algorithm found a solution four times out of twenty, i.e. with a success rate of 20%. The second scenario was a little bit better, with five successful runs, i.e. with a success rate of 25%. The results of the second scenario can be seen in Figure 5.20.

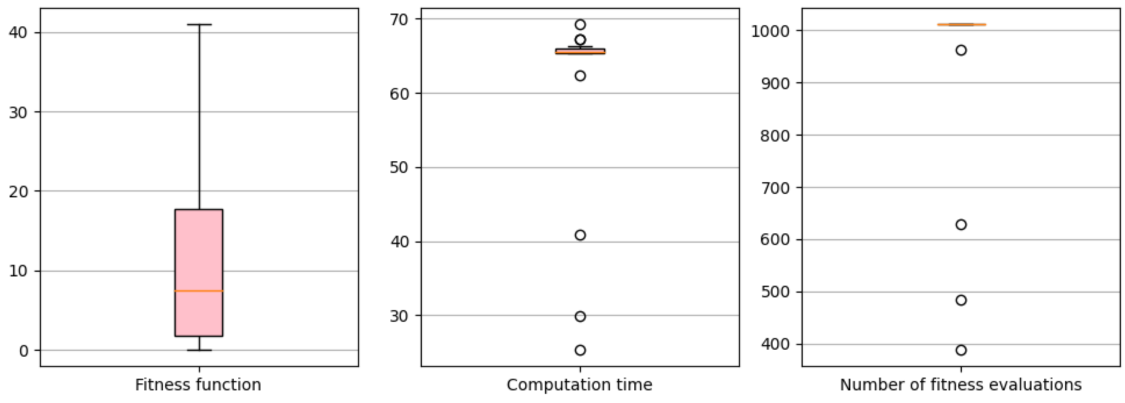


Figure 5.19: Results of GA for a rotated rice grain with a penalty in the first scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

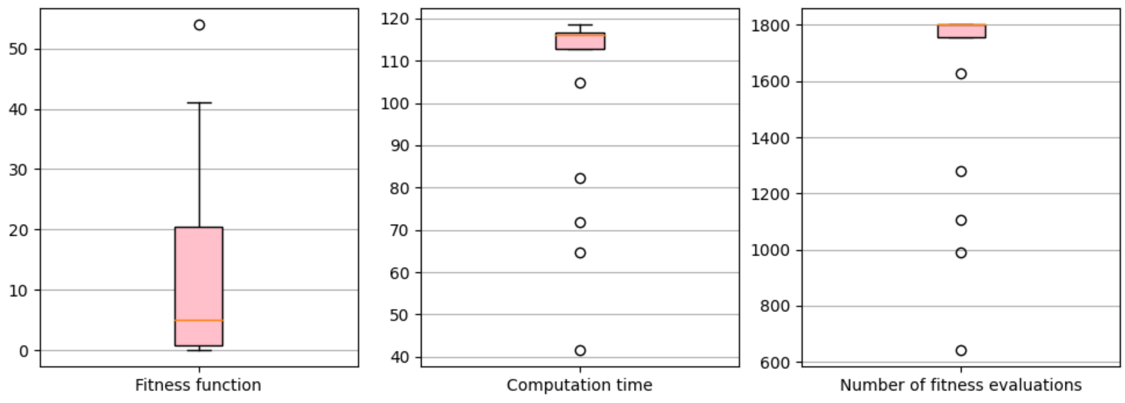


Figure 5.20: Results of GA for a rotated rice grain with a penalty in the second scenario (value of fitness function, computation time (in seconds) and number of fitness evaluations)

The median value of fitness function in the first scenario was 7.5 and 5 in the second scenario, which belongs to 0.7% and 0.46% of the best solutions. The algorithm was in both scenarios quite successful. The average computation time of one run in the first scenario is 61 seconds. In the second scenario, it is 106 seconds. The average number of fitness evaluations to found the solution is 933 in the first scenario (maximum is 1012) and 1634 in the second scenario (maximum is 1802).

5.4.2 CMA-ES

The results, for the value of fitness function, computation time and the number of fitness evaluations, obtained during the twenty runs are shown in Figure 5.21. The algorithm was successful in seven runs out of twenty, ie with a success rate of 35%. The median value of the fitness function was 1, which belong to 0.09% of the best solutions. CMA-ES was more successful than GA, but both of them achieved really good results.

The average computation time of one run is approximately 95 seconds. The average number of fitness evaluations to found the solution is 877 (maximum is 1100).

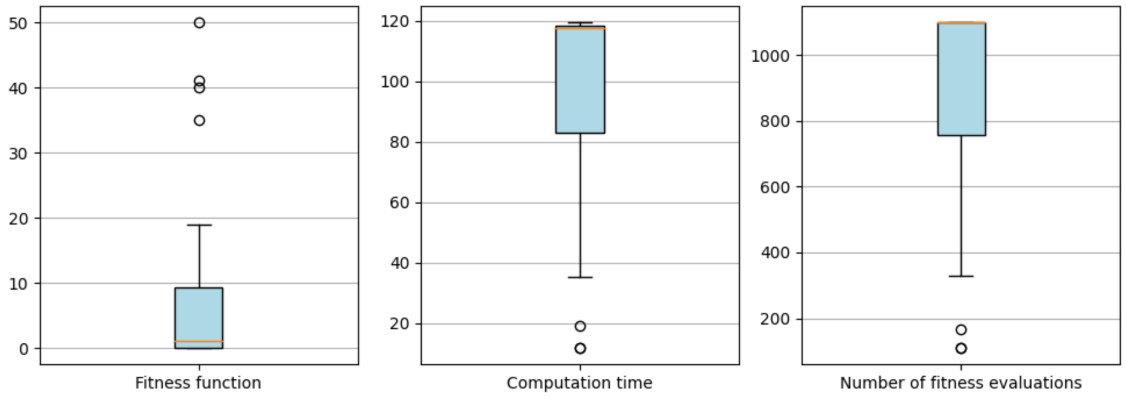


Figure 5.21: Results of CMA-ES for a rotated rice grain with a penalty (value of fitness function, computation time (in seconds) and number of fitness evaluations)

The results of the generation of found solution and transducer position are shown in Figure 5.22. The median value of the generation of the found solution is 6, which means that in 50% of cases, the solution was found in the 6-th generation or sooner. From the transducer position graph, we can again see, that CMA-ES found solutions with a wide range of transducer positions.

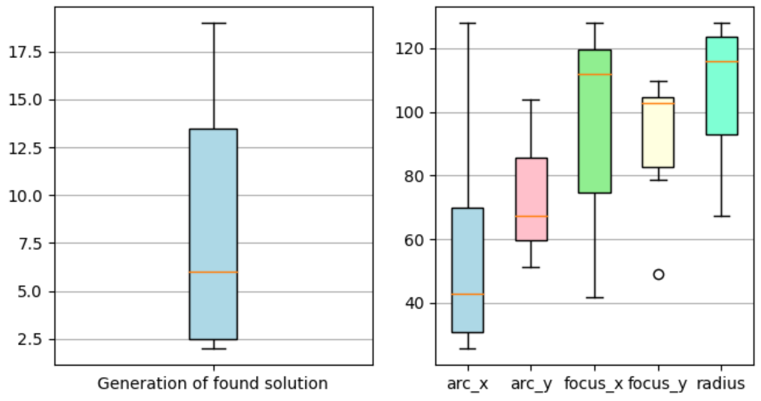


Figure 5.22: Results of CMA-ES for a rotated rice grain with a penalty (the generation of found solution and transducer position)

5.4.3 Simulated annealing

For the third benchmark, three scenarios with different parameter sets were again used. This time, the parameters were a little different than the previous two times. Parameters are shown in Table 5.3. The difference between parameter sets for this benchmark and parameter sets for previous two benchmarks is in the number of iterations for second and third scenario (100 -> 200), an initial temperature for third scenario (200 -> 100) and in initial individual for the second scenario.

Table 5.3: Parameters of Simulated Annealing

	1st parameter set	**2nd parameter set**	**3rd parameter set**
Iterations	100	200	200
Temperature	100	100	100
Initial individual	20, 20, 100, 100, 100	20, 20, 100, 100, 100	50, 50, 70, 70, 70
Cooling step	$InitialTemperature * 0.95^k$ where k means iteration		

This change in the initial individual was necessary because, as we have seen with rotated rice, if the beam does not hit the target at all, it is very difficult for the algorithm to determine in which direction it should go. In this case, it was even more difficult because even if it went in the right direction, it had to overcome the penalty part first and only then would it get to the target part. Simulated annealing algorithm did not know how to overcome this.

For better understanding, how initial individuals for this scenario look like, we can see all of them in Figure 5.23. The second initial individual is not used as we mention above, because the algorithm couldn't solve this. The second scenario has the same initial individual as the first scenario.

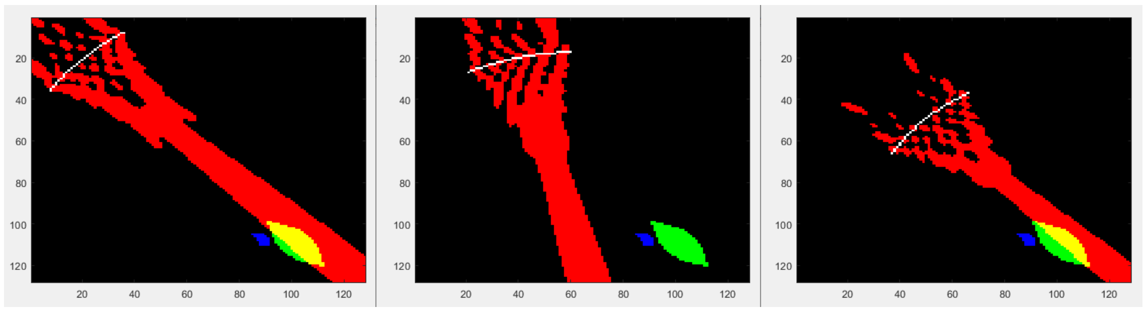


Figure 5.23: Three initial options for rotated rice grain with a penalty (initial individual shown in the middle is not used)

In Figure 5.24, results of all three scenarios are displayed. The first scenario is in graphs shown as (a), second scenarios as (b) and third as (c). We can see that each one out of the three scenarios found solution multiple times. The first one found solution nine times out of twenty, i.e. with a success rate of 45%, the second scenario found solution eight times out of twenty, i.e. with a success rate of 40% and the third scenario six times out of twenty, i.e. with a success rate of 30%. The median values of a fitness function for scenarios were 3, 4 and 5, which belongs from 0.28% to 0.46% of the best solutions. The average computation time of one run is approximately 10 seconds in the first scenario, 17 seconds in the second scenario and 20 seconds in the third scenario. The average number of fitness evaluations to found the solution is 70 in the first scenario (maximum is 101), 136 in the second scenario (maximum is 201) and 155 in the third scenario (maximum is 201).

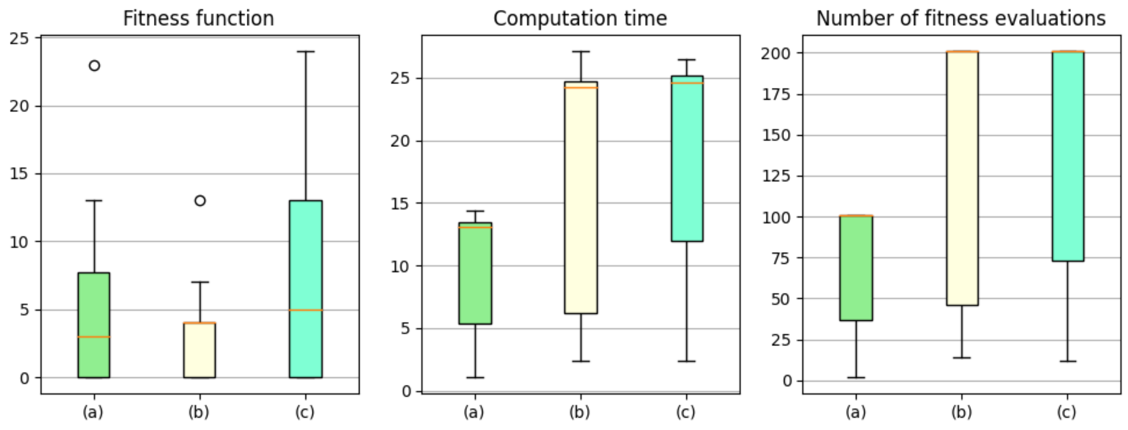


Figure 5.24: Results of SA for a rotated rice grain with a penalty (value of fitness function, computation time (in seconds) and number of fitness evaluations)

5.5 Summary

This chapter described all the experiments that were performed on the three benchmarks with a homogeneous medium. Based on the measured values and observations, we can call CMA-ES the most successful algorithm among all three. Simulated annealing was also very successful, but unlike CMA-ES, its success depended greatly on the initial individual. The worst out of all three algorithms was a genetic algorithm. However, this does not mean that the genetic algorithm cannot solve such an optimization problem. This means that CMA-ES and SA were more successful than GA, although GA was also able to find some good solutions. For genetic algorithm to work better, it could help to use more individuals, but this would increase optimization time. Examples of found solutions can be seen in Appendix C.

Chapter 6

Experiments with a heterogeneous medium

This chapter describes the performed experiments with a heterogeneous medium. Experiments were performed on Barbora supercomputer cluster, which consists of 201 compute nodes, totaling 7232 compute cores with 44544 GB RAM, giving over 848 TFLOP/s theoretical peak performance. Nodes are interconnected through a fully non-blocking fat-tree InfiniBand network, and are equipped with Intel Cascade Lake processors. A few nodes are also equipped with NVIDIA Tesla V100-SXM2[1]. The number of burned normalized core-hours is 6216.

The size of the computational grid for the experiments was 257 x 257 (from -128 to 128). Heterogeneous medium which consists of water, soft tissue (skin and brain) and skull, can be seen in Figure 6.1. Blue colour represents the water, yellow colour represents the skull and turquoise colour represents skin and brain.

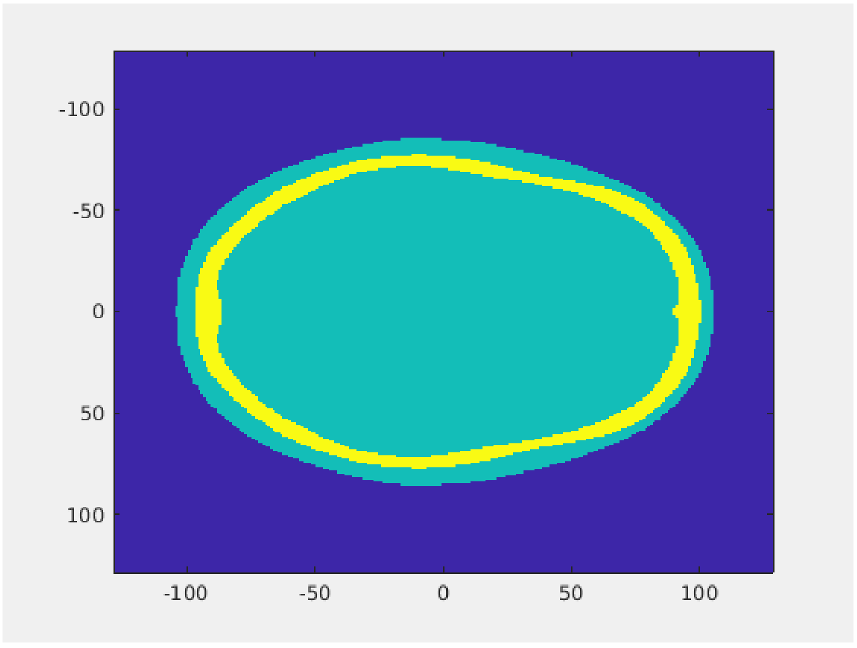
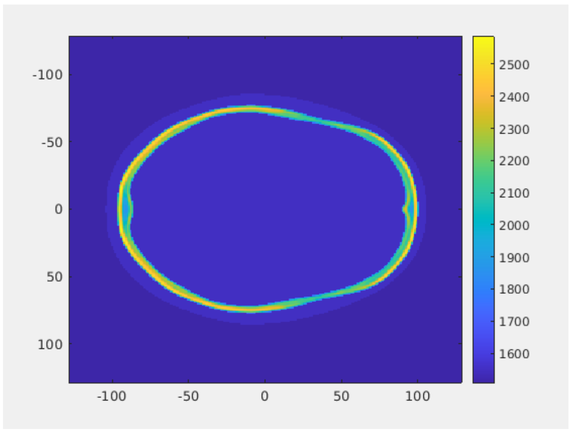


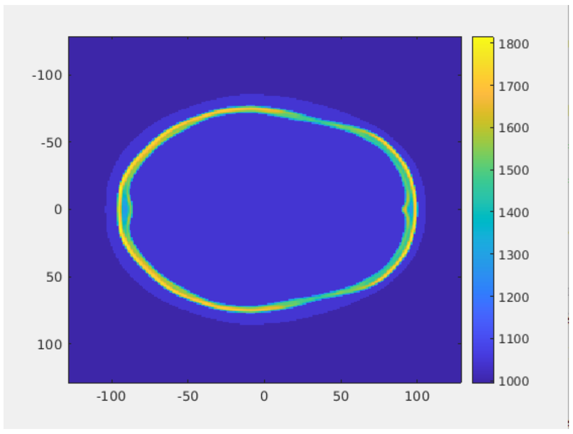
Figure 6.1: Heterogeneous medium formed by water, skin, skull and brain.

[1]https://docs.it4i.cz/barbora/introduction/

The sound speed of the medium consists of values 1509 m/s (for water), 1550 m/s (for soft tissue - skin and brain) and in range 1500-3100 m/s (for skull). The density is also different, 996 kg/m^3 for water, 1045 kg/m^3 for soft tissue and in range 1000-2200 kg/m^3 for skull. Sound speed and density are shown in Figure 6.2.



(a) Sound speed



(b) Density

Figure 6.2: Diagrams showing sound speed and density.

The experiments were performed with various algorithms, namely genetic algorithms, CMA-ES and simulated annealing. Each algorithm was tested on three scenarios and each test consists of several runs. Three test scenarios were:

1. hitting the target area with a single sonication
2. hitting the target area without hitting the penalty area with four sonications
3. hitting the target area without hitting the penalty area with a single sonication

In the following Figure 6.3, we can see the skull (white) and then the areas we want to hit (target area - green) and which we want to avoid (penalty area - blue).

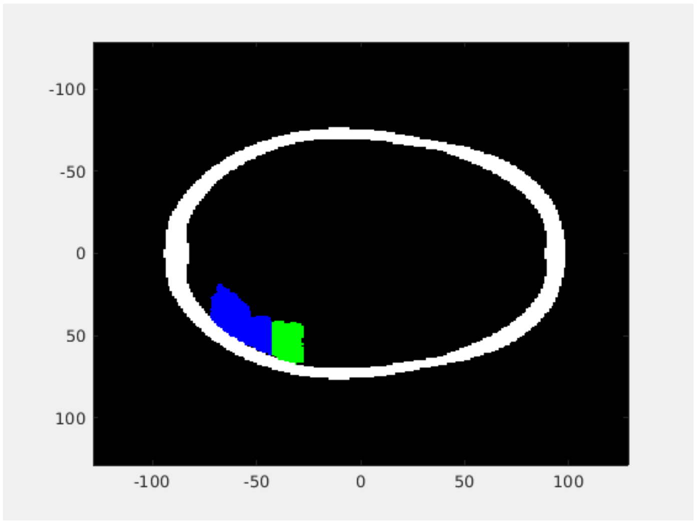


Figure 6.3: Diagram showing target and penalty area.

6.1 First test scenario

The first test scenario is hitting the target area with a single sonication. In this scenario, only the mask of the area we want to hit is used, and the penalty mask is not used. Target mask consists of 350 points, which we need to hit. After overlapping the target mask with a Gaussian curve, the sum of the points we want to hit is equal to 84.42. A solution that does not hit any part of the target or the penalty area and also does not contain a transducer that overlaps with the head, has a fitness value of 84.42. So we are looking for solutions that have a fitness value lower than 84.42 because it means that some part of the target has been hit. Solutions with a value higher than 84.42 had to hit the penalty area or the transducer overlapped with the head.

Each optimization algorithm was run ten times. In each experiment, the value of the fitness function, the time of calculation and the number of evaluations of the fitness function were examined. The results obtained are statistically given below.

Genetic algorithms

The genetic algorithm for the first scenario was run with the parameters listed in Table 6.1:

Table 6.1: Parameters of Genetic Algorithm

	Parameter set
Population	27
Generations	10
Selection	Tournament, 5
Crossover	Intermediate 0.8
Mutation	0.5

The results of the experiments with GA for the first scenario are shown below. In Figure 6.4 we can see the boxplot of the best fitness function found in each run. Figure 6.5 shows boxplots of fitness functions of the last generation individuals in each run.

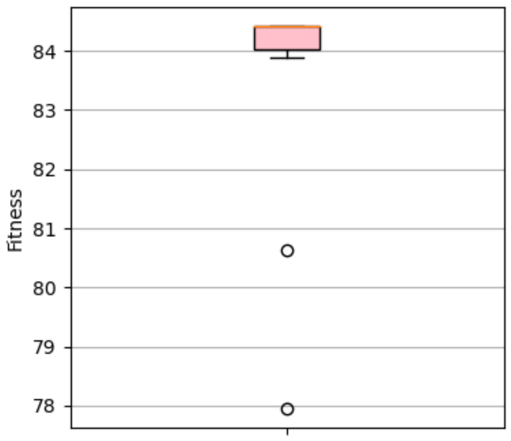


Figure 6.4: Boxplot of the best fitness results from all ten GA runs.

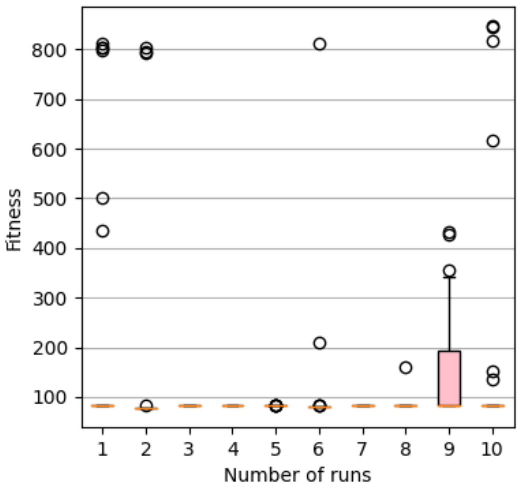


Figure 6.5: Boxplots of the status of the last generation from all ten GA runs.

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.6 and Figure 6.7. In the first figure, the value of the fitness function is the mean of the average of all individuals in a particular generation. The fitness function in the second figure is the average of the best individuals over a particular generation.

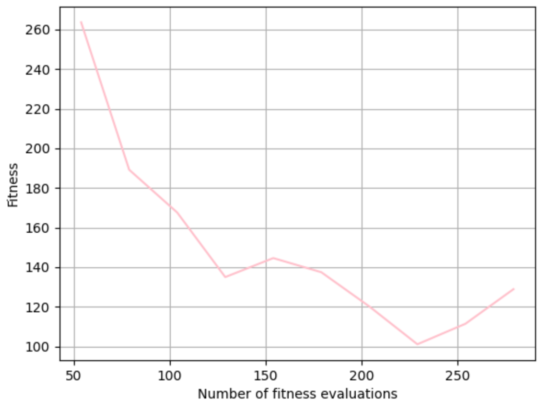


Figure 6.6: The influence of the mean of average solutions found in each generation to the number of fitness function evaluations.

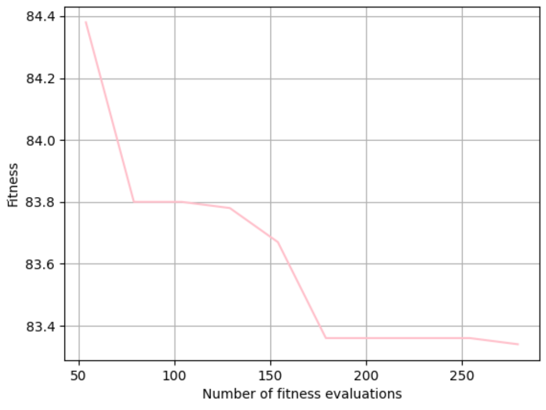


Figure 6.7: The influence of the mean of the best solutions found in each generation to the number of fitness function evaluations.

Figure 6.8 shows a boxplot of the computation time of all runs. The duration of all ten runs is 13 hours 49 minutes and 6 seconds, which is equal to 49746 seconds.

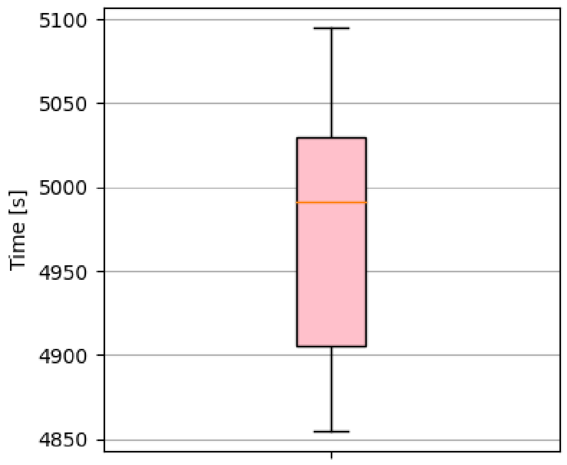


Figure 6.8: Boxplot of time required to complete the optimization, in seconds. ($4800s = 1h20m$)

CMA-ES

The CMA-ES algorithm for the first scenario was run with a population size equal to 54, number of iterations 4 (originally it was supposed to be 5 iterations so that the number of fitness function evaluations was about the same as it was for GA and SA, unfortunately, I entered a value of 4 in the tests and didn't notice it) and number of parents 27.

The results of the experiments with CMA-ES for the first scenario are shown below. In Figure 6.9 we can see the boxplot of the best fitness function found in each run. Figure 6.10 shows boxplots of the average population of each iteration in each run.

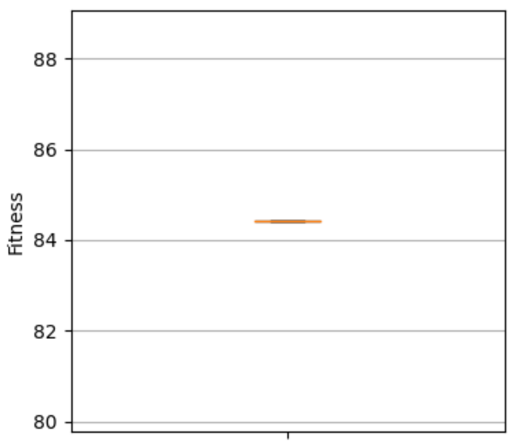


Figure 6.9: Boxplot of the best fitness results from all ten CMA-ES runs.

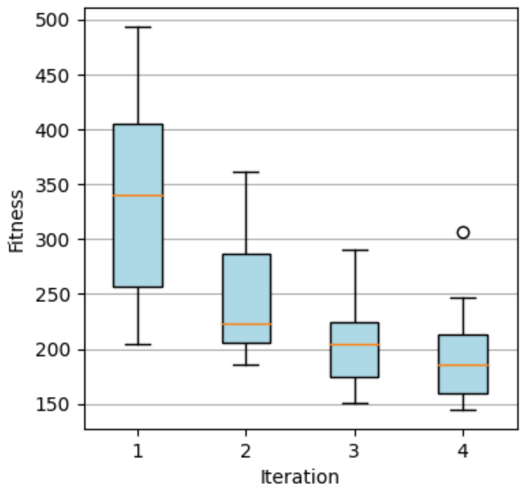


Figure 6.10: Boxplots of the mean of the population of each iteration from all ten CMA-ES runs.

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.11. The value of the fitness function is the mean of the average of all individuals in a particular iteration for all runs. Figure 6.12 shows the boxplot of the computation time of all runs. The duration of all ten runs is 10 hours 30 minutes and 58 seconds, which is equal to 37858 seconds.

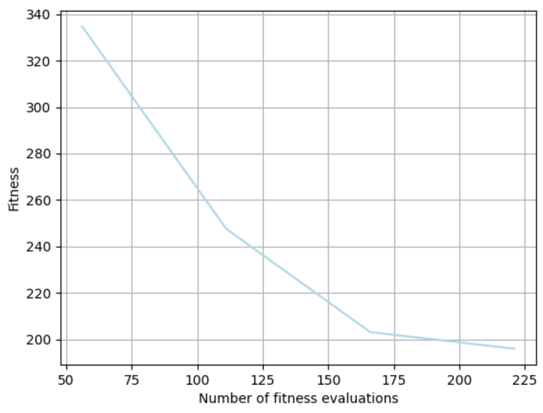


Figure 6.11: The influence of the mean of average solutions found in each generation to the number of fitness function evaluations.

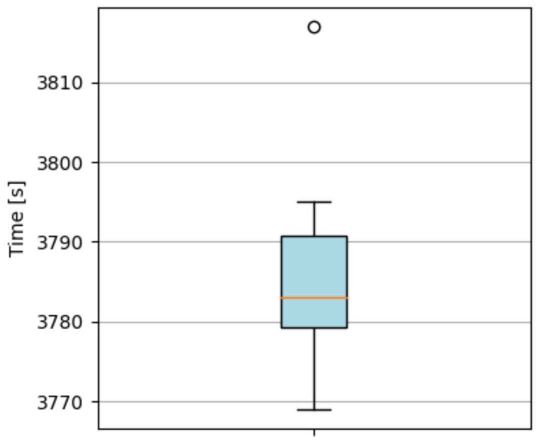


Figure 6.12: Boxplot of time required to complete the optimization, in seconds. ($3600s = 1h$)

Simulated annealing

The simulated annealing for the first scenario was started with one parameter set shown in Table 6.2:

Table 6.2: Parameters of Simulated Annealing

	Parameter set
Iterations	270
Temperature	1000
Initial individual	[-75, -75, 0, 0, 100]
Cooling step	$InitialTemperature * 0.95^k$ where k means iteration

The initial individual was selected at random. Its values represent the transducer located in the upper left corner, so in order to hit the target, the optimization must move the transducer to the lower left.

The results of the experiments with SA for the first scenario are shown below. In Figure 6.13 we can see the boxplot of the best fitness function found in each run. Figure 6.14 shows the boxplot of the computation time of all runs. The duration of all ten runs is 13 hours 15 minutes and 14 seconds, which is equal to 47714 seconds.

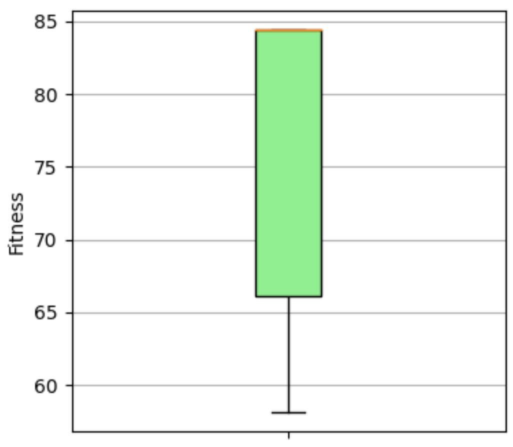


Figure 6.13: Boxplot of the best fitness results from all ten SA runs.

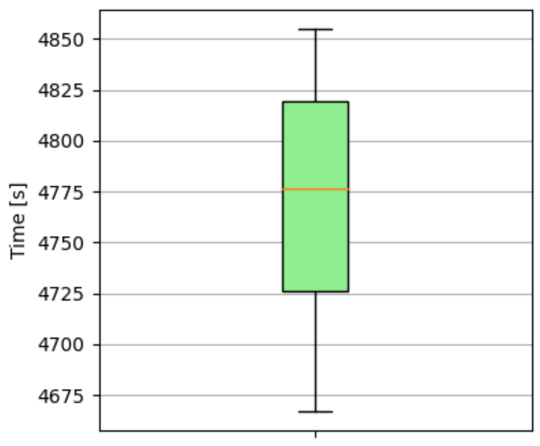


Figure 6.14: Boxplot of time required to complete the optimization, in seconds. $(4800s = 1h20m)$

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.15 and 6.16. In the first figure, the value of the fitness function is the mean of ten individuals (one from each run) in a particular iteration. The second figure shows the second half of the impact of fitness function evaluation on its value at run, in which the best solution was found.

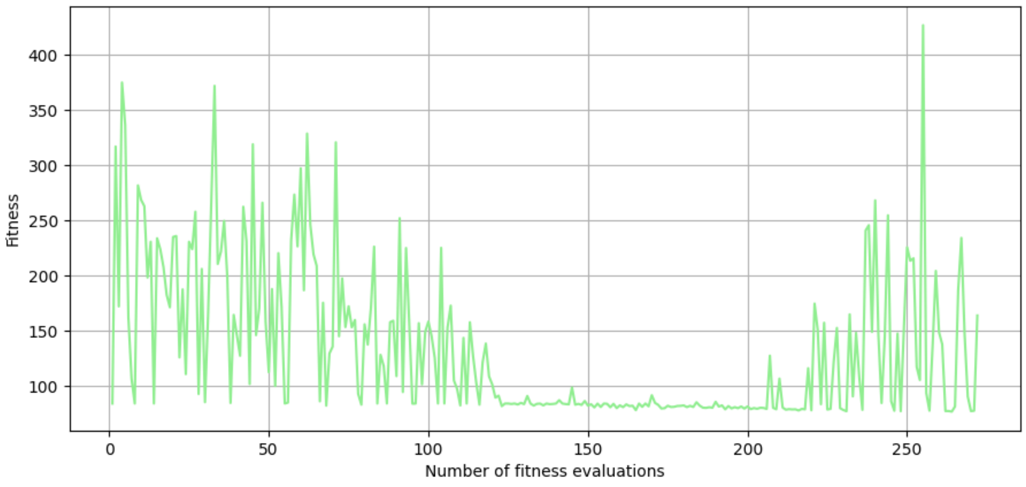


Figure 6.15: The influence of the mean of ten individuals (one from each run) to the number of fitness function evaluations.

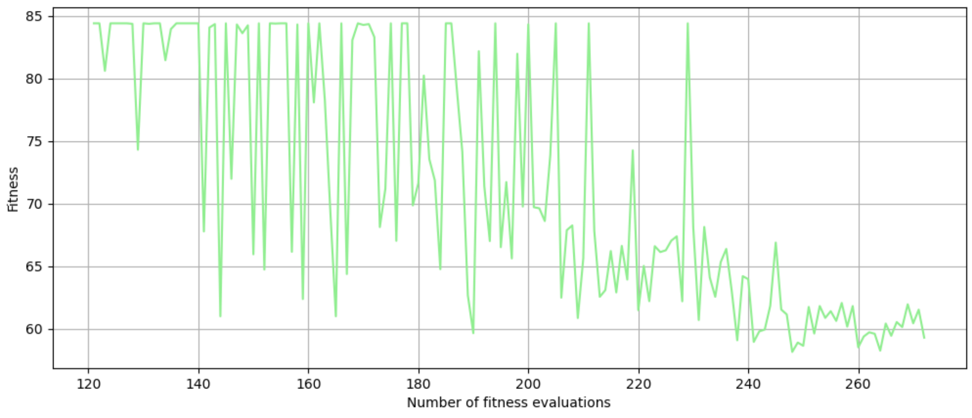


Figure 6.16: The influence of the fitness function of the individual from the best run to the number of fitness function evaluations.

6.2 Second test scenario

The second test scenario is hitting the target area with four sonications. In this scenario, only the mask of the area we want to hit is used, and the penalty mask is not used. Target mask consists of 350 points, which we need to hit. After overlapping the target mask with a Gaussian curve, the sum of the points we want to hit is equal to 84.42. Due to the time complexity, only five runs of individual optimization algorithms were run and not ten as in the first scenario.

The calculation of the fitness function is a little bit different here. After one sonication, we obtain a matrix of pressure from which we find out which points were hit. However, we do not calculate the fitness value for this one sonication, but we run another and find out which points it hit. We will repeat this for the third and fourth sonications. In the end, we have one matrix of pressure, which was created by the sum of all four matrices that we obtained during the individual sonications. From this resulting matrix of pressure, we calculate how many points of the target area and the penalty area were hit. This approach prevents individual sonications from hitting the same part of the target and favors finding solutions that cover a larger part of the target area.

Genetic algorithms

The genetic algorithm for the second scenario was run with the same parameters as in the first scenario, only the number of individuals was reduced by half (to 13 individuals) due to time complexity.

The results of the experiments with GA for the second scenario are shown below. In Figure 6.17 we can see the boxplot of the best fitness function found in each run. Figure 6.18 shows boxplots of fitness functions of the last generation individuals in each run.

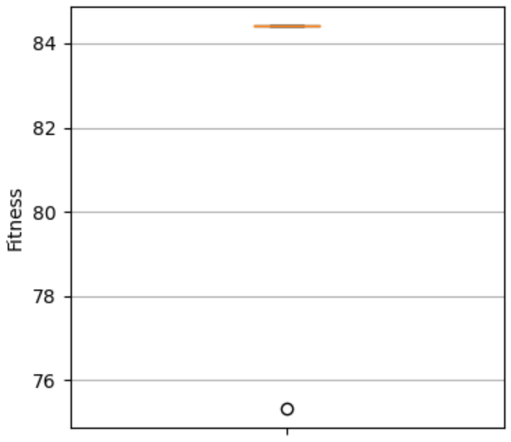


Figure 6.17: Boxplot of the best fitness results from all five GA runs.

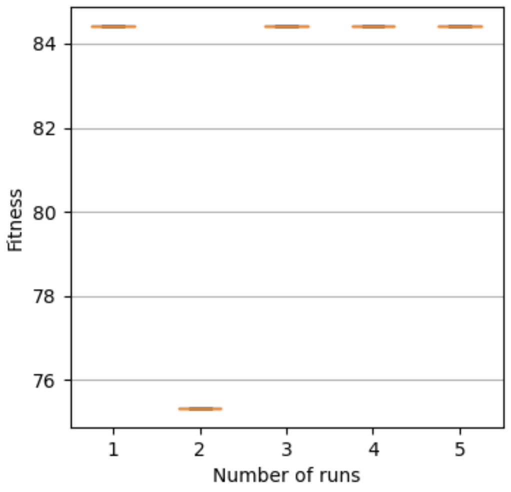


Figure 6.18: Boxplot of the status of the last generation from all five GA runs.

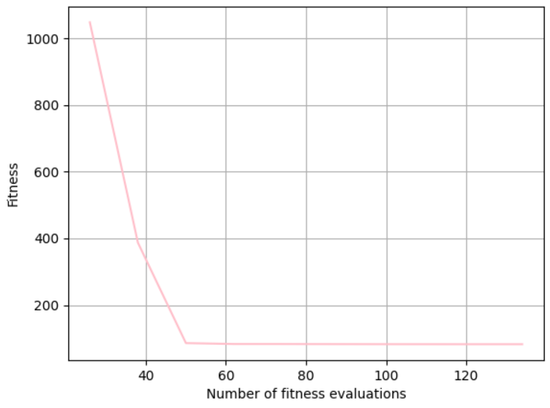


Figure 6.19: The influence of the mean of average solutions found in each generation to the number of fitness function evaluations.

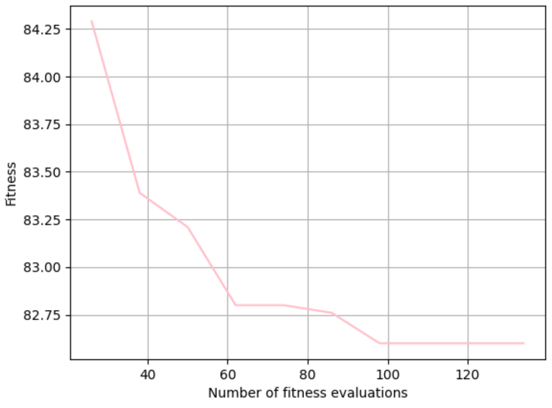


Figure 6.20: The influence of the mean of the best solutions found in each generation to the number of fitness function evaluations.

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.19 and Figure 6.20. In the first figure, the value of the fitness function is the mean of the average of all individuals in a particular generation. The fitness function in the second figure is the average of the best individuals over a particular generation.

Figure 6.21 shows a boxplot of the computation time of all runs. The duration of all five runs is 12 hours 51 minutes and 34 seconds, which is equal to 46294 seconds. The average time of one run is approximately 9259s.

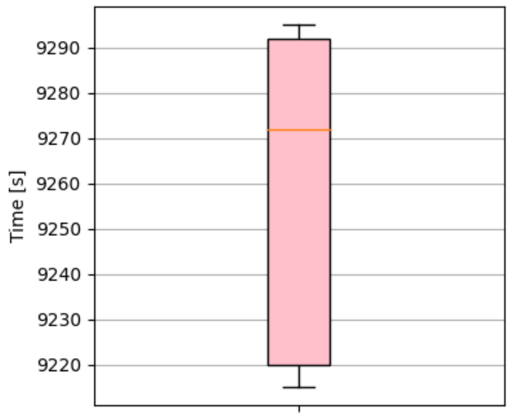
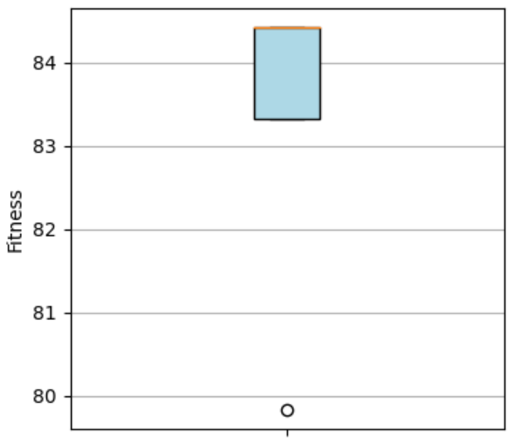


Figure 6.21: Boxplot of time required to complete the optimization, in seconds. ($9000s = 2h30m$)

CMA-ES

The CMA-ES for the second scenario was run with a population size equal to 26, number of iterations 5 and number of parents 13. The results of the experiments with CMA-ES for the second scenario are shown below. In Figure 6.22 we can see the boxplot of the best fitness function found in each run. Figure 6.23 shows boxplots of the average population of each iteration in each run.



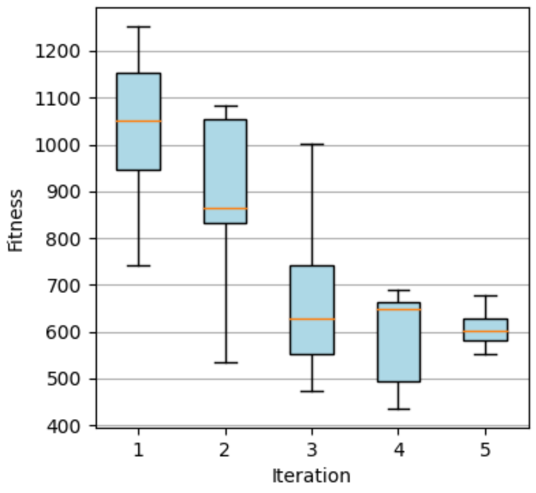


Figure 6.22: Boxplot of the best fitness results from all five CMA-ES runs.

Figure 6.23: Boxplots of the mean of the population of each iteration from five runs.

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.24. The value of the fitness function is the mean of the average of all individuals in a particular iteration for all runs. Figure 6.25 shows the boxplot of the computation time of all runs. The duration of all five runs is 12 hours 59 minutes and 1 second, which is equal to 46741 seconds.

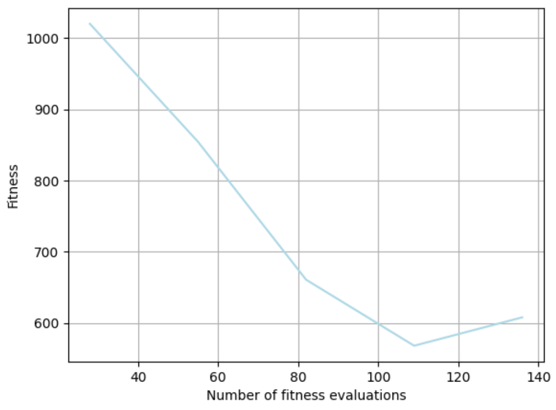


Figure 6.24: The influence of the mean of average solutions found in each generation to the number of fitness function evaluations.

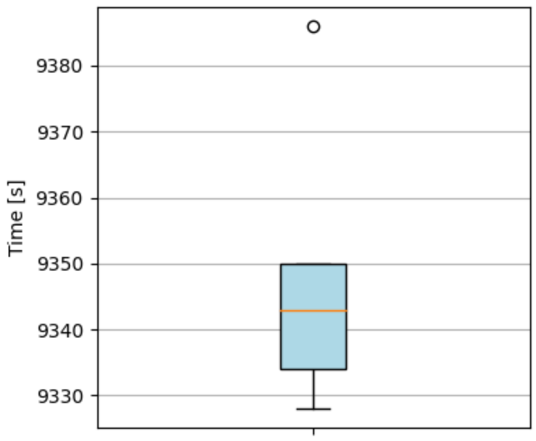


Figure 6.25: Boxplot of time required to complete the optimization, in seconds. ($9300s = 2h35m$)

Simulated annealing

The simulated annealing for the second scenario was started with one parameter set shown in Table 6.3:

Table 6.3: Parameters of Simulated Annealing

	Parameter set
Iterations	130
Temperature	1000
Initial individual	[-75, -75, 0, 0, 100], [75, 75, 0, 0, 100], [-105, -105, 0, 0, 100], [105, 105, 0, 0, 100]
Cooling step	$InitialTemperature * 0.95^k$ where k means iteration

The results of the experiments with SA for the second scenario are shown below. In Figure 6.26 we can see the boxplot of the best fitness function found in each run. Figure 6.27 shows the boxplot of the computation time of all runs. The duration of all five runs is 12 hours 48 minutes and 21 seconds, which is equal to 46101 seconds.

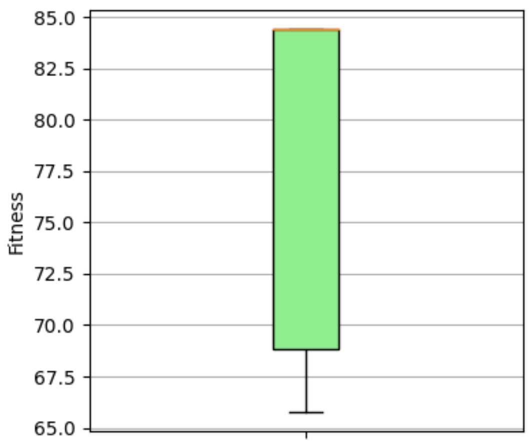


Figure 6.26: Boxplot of the best fitness results from all ten SA runs.

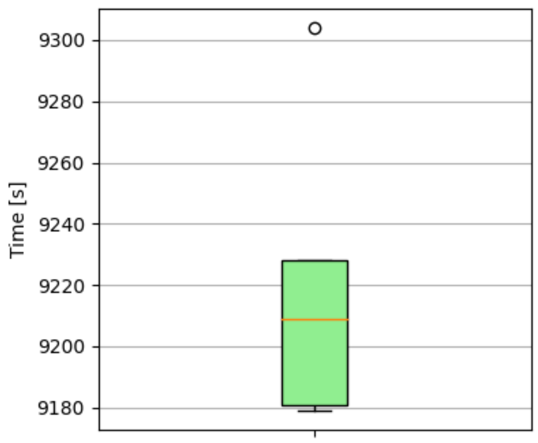


Figure 6.27: Boxplot of time required to complete the optimization, in seconds. ($9000s = 2h30m$)

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.28. In the figure, the value of the fitness function is the mean of five individuals (one from each run) in a particular iteration.

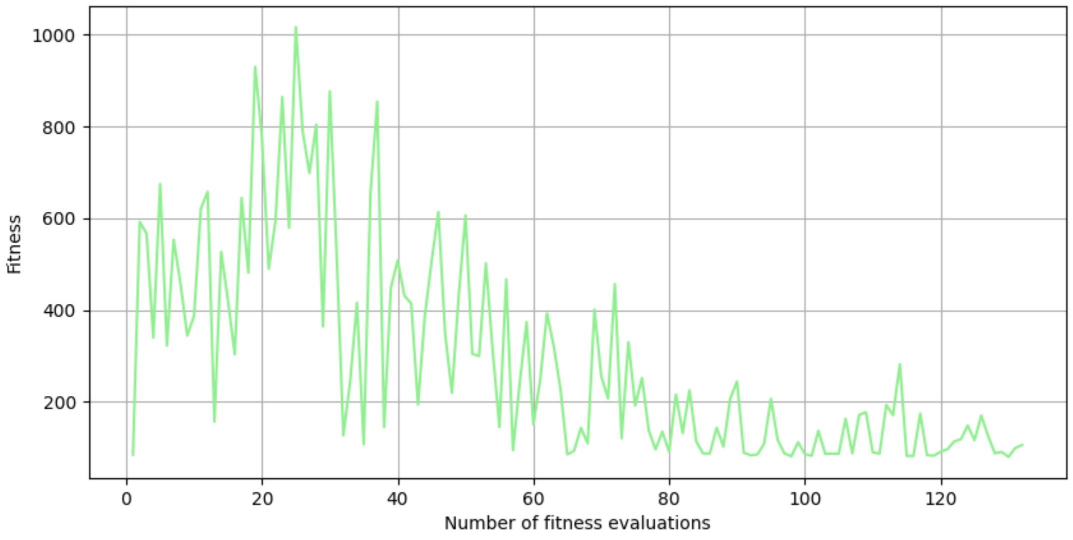


Figure 6.28: The influence of the mean of five individuals (one from each run) to the number of fitness function evaluations.

6.3 Third test scenario

The third test scenario is hitting the target area without hitting the penalty area with one sonication. The target mask of the area we want to hit consists of 350 points. After overlapping the target mask with a Gaussian curve, the sum of the points we want to hit is equal to 84.42. The penalty mask consists of 693 points, which we do not want to hit.

Each optimization algorithm was run five times. In each experiment, the value of the fitness function, the time of calculation and the number of evaluations of the fitness function were examined. The results obtained are statistically given below.

Genetic algorithms

The genetic algorithm for the third scenario was run with the same parameters as in the first scenario. The results of the experiments with GA for the third scenario are shown below. In Figure 6.29 we can see the boxplot of the best fitness function found in each run. Figure 6.30 shows boxplots of fitness functions of the last generation individuals in each run.

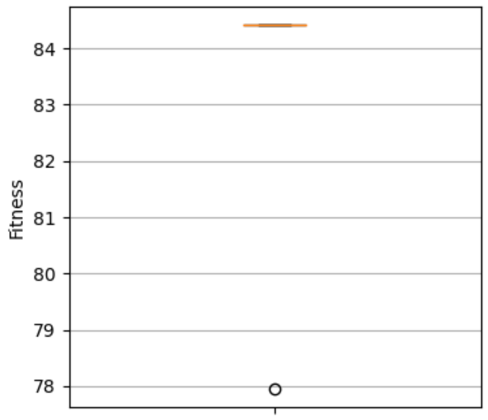


Figure 6.29: Boxplot of the best fitness results from all five GA runs.

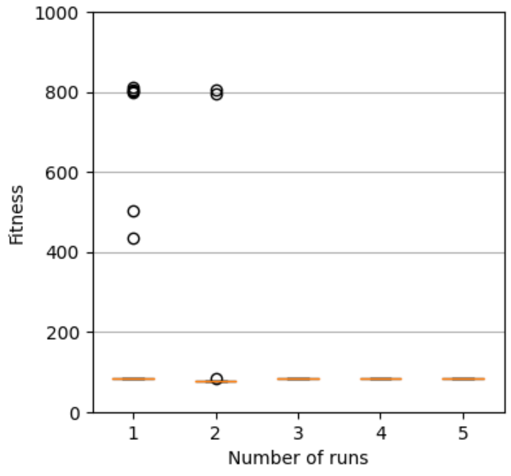


Figure 6.30: Boxplot of the status of the last generation from all five GA runs.

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.31 and Figure 6.32. In the first figure, the value of the fitness function is the mean of the average of all individuals in a particular generation. The fitness function in the second figure is the average of the best individuals over a particular generation.

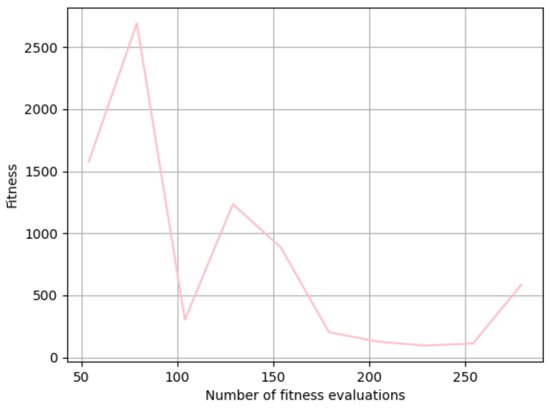


Figure 6.31: The influence of the mean of average solutions found in each generation to the number of fitness evaluations.

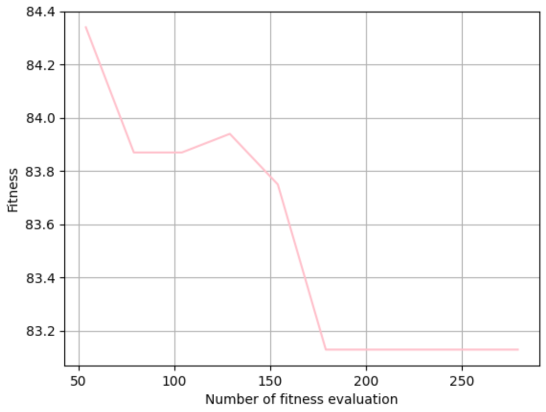


Figure 6.32: The influence of the mean of the best solutions found in each generation to the number of fitness evaluations.

Figure 6.33 shows a boxplot of the computation time of all runs. The duration of all five runs is 6 hours 47 minutes and 50 seconds, which is equal to 24470 seconds.

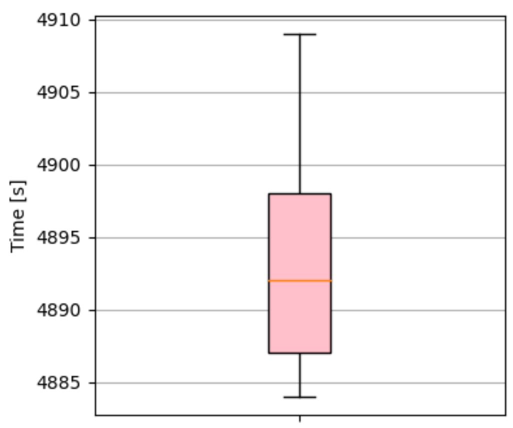


Figure 6.33: Boxplot of time required to complete the optimization, in seconds. ($4800s = 1h20m$)

CMA-ES

The CMA-ES for the third scenario was run with a population size equal to 54, number of iterations 5 and number of parents 27. The results of the experiments with CMA-ES for the third scenario are shown below. In Figure 6.34 we can see the boxplot of the best fitness function found in each run. Figure 6.35 shows boxplots of the average population of each iteration in each run.

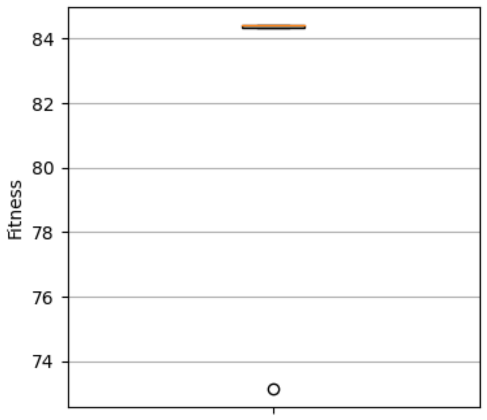


Figure 6.34: Boxplot of the best fitness results from all five CMA-ES runs.

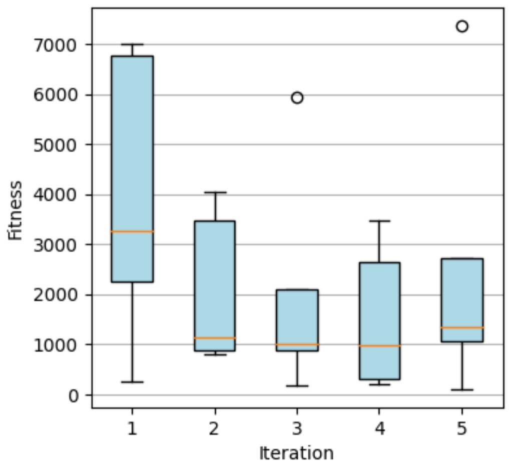


Figure 6.35: Boxplots of the mean of the population of each iteration from all five CMA-ES runs.

The influence of the number of fitness function evaluations on fitness value can be seen in Figure 6.36. The fitness value is the mean of the average of all individuals in a particular iteration for all runs. Figure 6.37 shows the boxplot of the computation time of all runs. The duration of all ten runs is 6 hours and 31 minutes, which is equal to 23460 seconds.

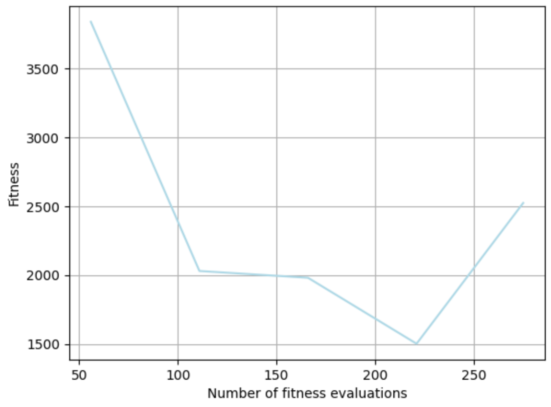


Figure 6.36: The influence of the mean of average solutions found in each generation to the number of fitness function evaluations.

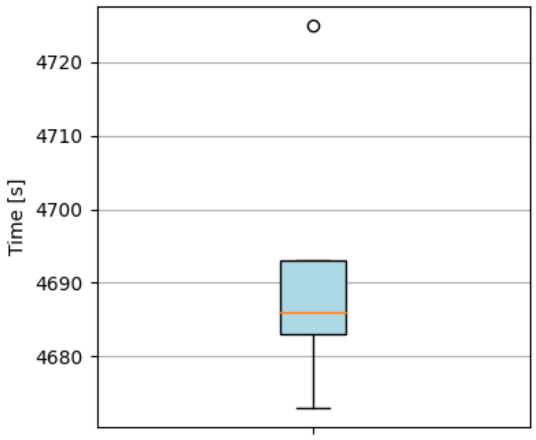


Figure 6.37: Boxplot of time required to complete the optimization, in seconds. ($4800s = 1h20m$)

Simulated annealing

The simulated annealing for the third scenario was started with the same parameters as in the first scenario, with a change in initial individual to [-75, 75, 0, 0, 100].

The results of the experiments with SA for the third scenario are shown below. In Figure 6.38 we can see the boxplot of the best fitness function found in each run. Figure 6.39 shows the boxplot of the computation time of all runs. The duration of all five runs is 6 hours 43 minutes and 37 seconds, which is equal to 24217 seconds.

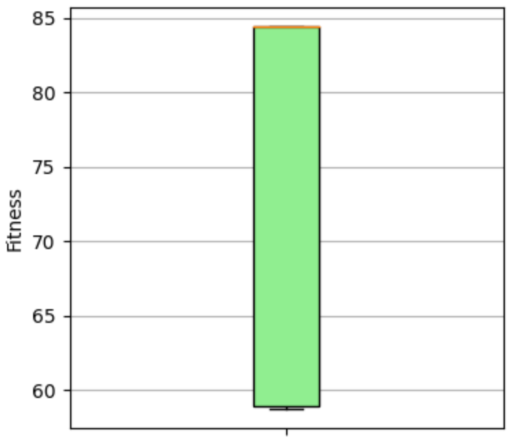


Figure 6.38: Boxplot of the best fitness results from all five SA runs.

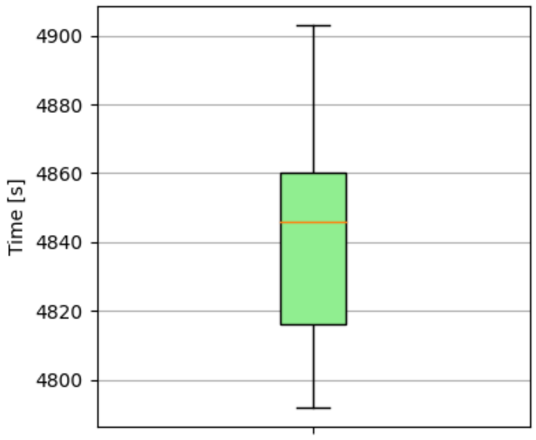


Figure 6.39: Boxplot of time required to complete the optimization, in seconds. ($4800s = 1h20m$)

The influence of the number of fitness function evaluations on fitness function value can be seen in Figure 6.40 and 6.41. In the first figure, the value of the fitness function is the mean of five individuals (one from each run) in a particular iteration. The second figure shows the last third of the impact of fitness function evaluation on its value at run, in which the best solution was found.

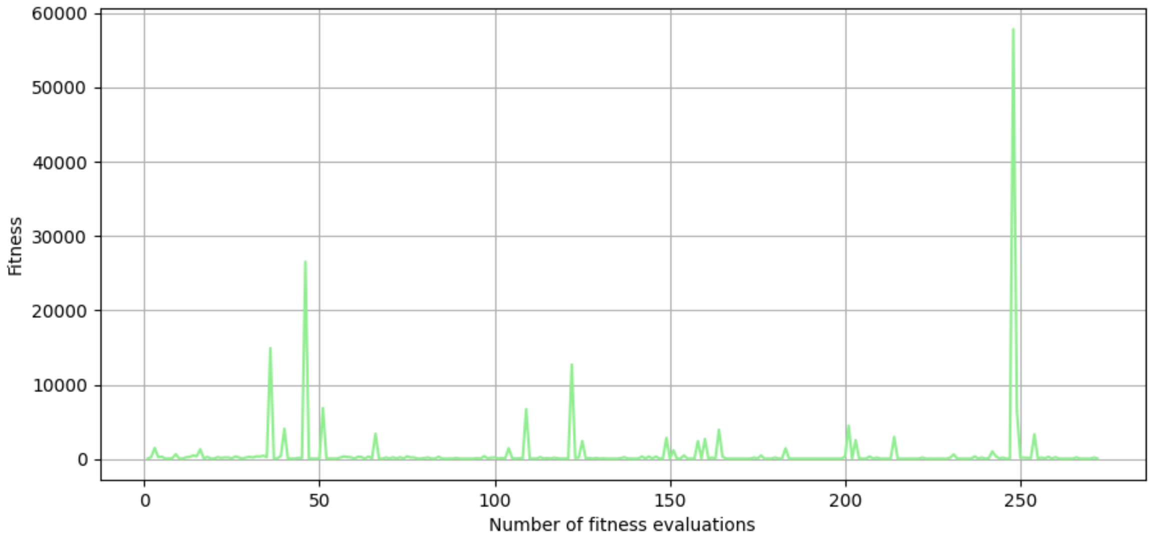


Figure 6.40: The influence of the mean of five individuals (one from each run) to the number of fitness function evaluations.

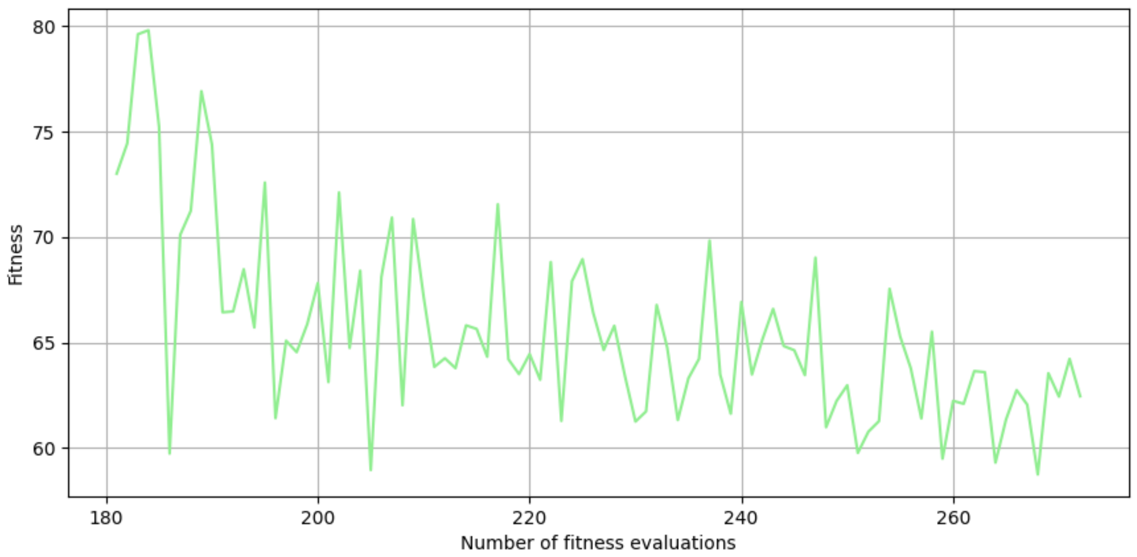


Figure 6.41: The influence of the fitness function of the individual from the best run to the number of fitness function evaluations.

6.4 Additional experiments

Experiment with a larger transducer

A 6.1 centimeter transducer was used instead of a 4.1 centimeter transducer. Because the skull absorbs and reflects much of the ultrasound, the focus behind the skull is smaller and it is difficult to hit a larger part of the target. The aim of the experiment was to see if a larger transducer would achieve better results. Simulated annealing with 200 iterations was used as an optimization algorithm and three runs of the algorithm were performed. Both target and penalty masks were used in this experiment.

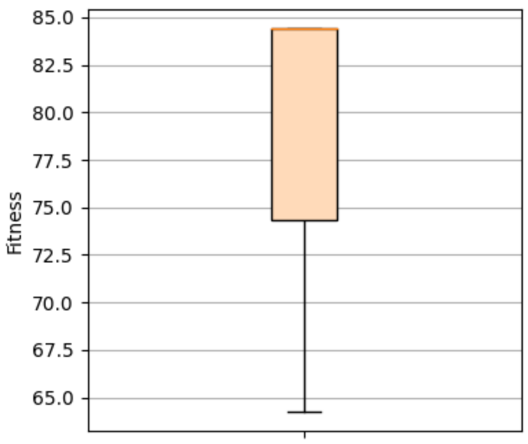


Figure 6.42: Boxplot of the best fitness results from all three SA runs.

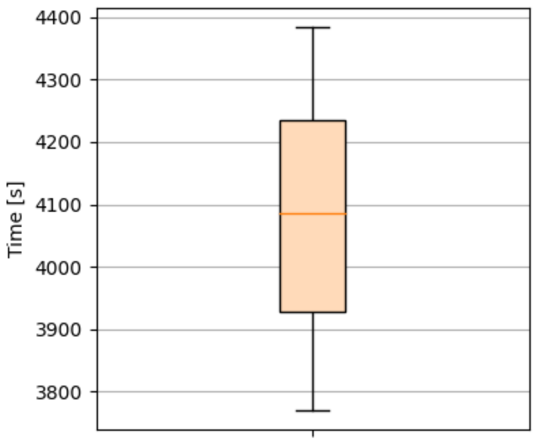


Figure 6.43: Boxplot of time required to complete the optimization, in seconds. $(4200s = 1h10m)$

Figure 6.44 displays the best found result. For comparison, in Figure 6.45 there is a smaller transducer that is in the same position. However, the smaller transducer was unable to exert sufficient pressure in the target area because much of the ultrasound was deflected or absorbed by the skull.

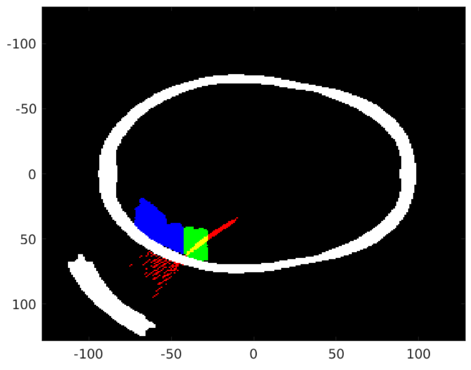


Figure 6.44: The best result with a score of 64.25 using a 6.1 cm transducer.

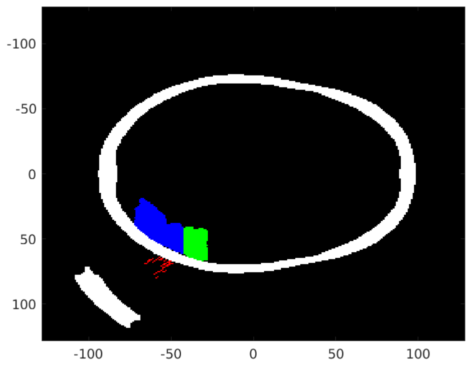


Figure 6.45: Result of a 4.1 cm transducer in the same position.

Experiment with a penalty around the entire target

This experiment aimed to see if the optimization could cope and move the transmitter away from the head so that no part of the penalty mask would be hit. The penalty mask consists of 1028 points, instead of the original 693. An illustration of the penalty mask (blue part) in this experiment is shown in Figure 6.46.

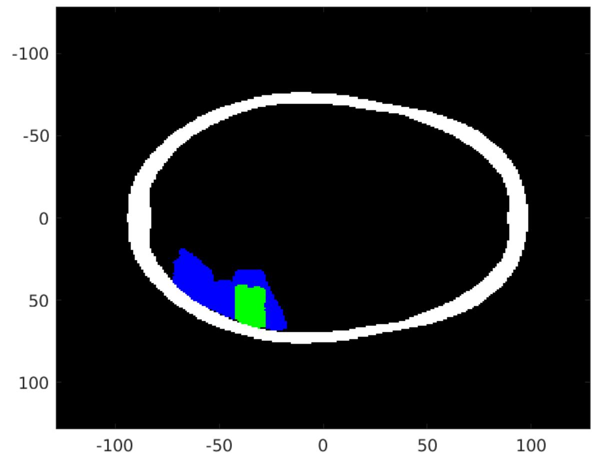


Figure 6.46: Demonstration of a penalty mask around a target.

From the Figure 6.47 we can see that the best solution found with all three experiments was a solution with fitness function 82.42. It means that the target was not hit even once. Because it was difficult to hit the target without hitting the penalty area around, the optimization ended with solutions that send ultrasound in a different direction than the skull.

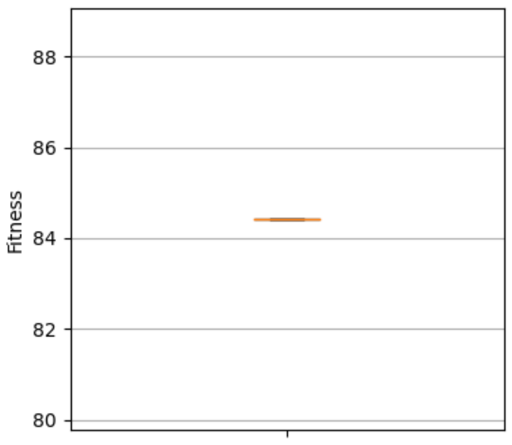


Figure 6.47: Boxplot of the best fitness results from all three SA runs.

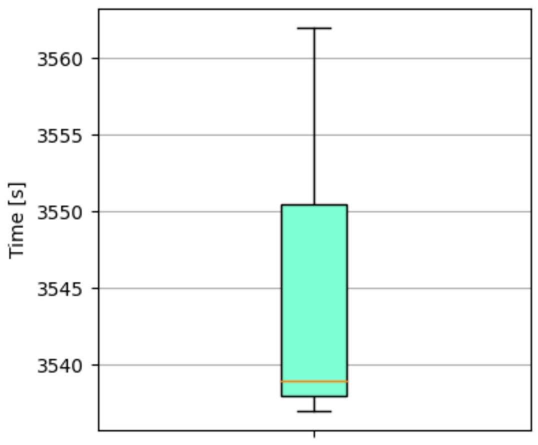


Figure 6.48: Boxplot of time required to complete the optimization, in seconds. ($3600s = 1h$)

Experiment with better initial individual

This experiment aimed to see if a better initial individual would help the optimization to find a better solution. Illustration of the initial individual in this experiment is shown in Figure 6.49. This experiment was also performed with a larger transducer (6.1 cm). Figure 6.50 shows the best found solution with a fitness value of 53.65.

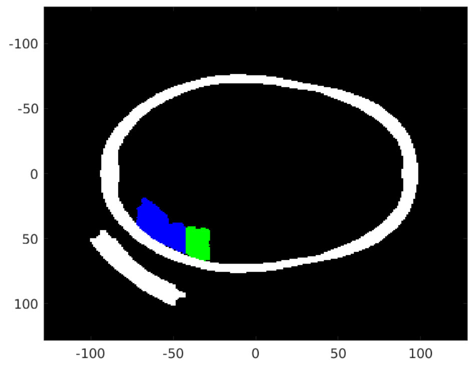


Figure 6.49: Initial individual with values [75, -75, 0, 0, 100].

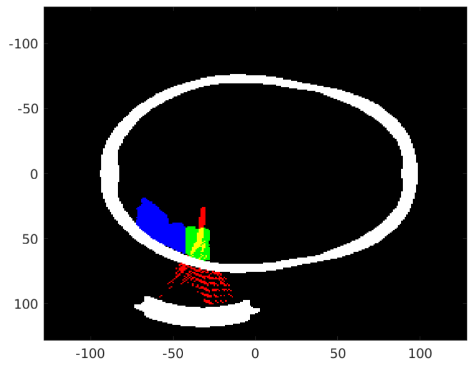


Figure 6.50: The best solution found with a score of 53.65.

Figure 6.51 shows a boxplot of the best fitness found. In this experiment, the best solution found (score = 53.65) was better than the best solution found by SA in the third scenario (score = 58.75). However, only three experiments were performed, so we can say only that it looks like, the better initial individual with a larger transducer helps to find better solutions. To confirm this assumption, more experiments would be needed.

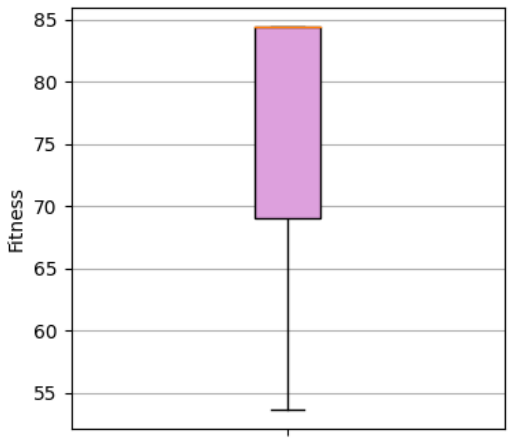


Figure 6.51: Boxplot of the best fitness results from all three SA runs.

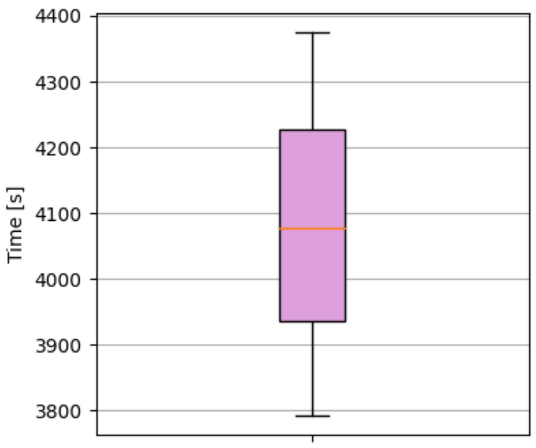


Figure 6.52: Boxplot of time required to complete the optimization, in seconds. ($4200s = 1h10m$)

Experiment with a lower pressure threshold

This experiment aimed to see if a lower pressure threshold would help the optimization to find better solutions. Only two runs of the experiment were performed and only a target mask was used.

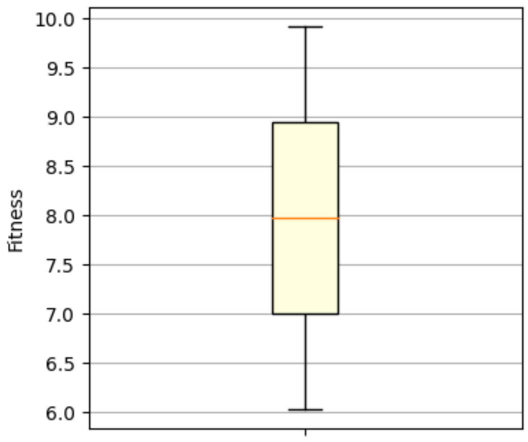


Figure 6.53: Boxplot of the best fitness results from both SA runs.

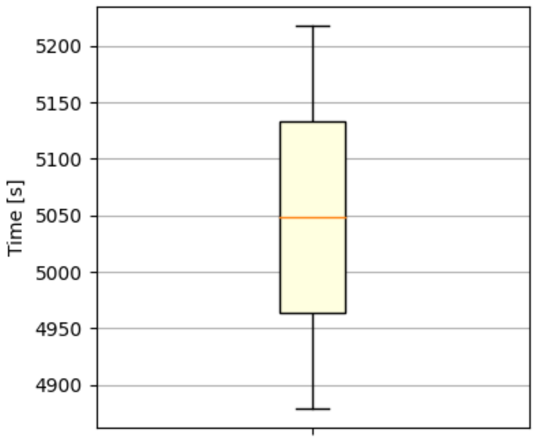


Figure 6.54: Boxplot of time required to complete the optimization, in seconds. ($4800s = 1h20m$)

Illustration of the best found solution in this experiment is shown in Figure 6.55. We can see, that a large part of the target area was hit, but also a large part behind the target area. In this experiment, no penalty was taken into account, so we are satisfied with the solutions found. In reality, however, such a solution would not make much sense, as we would burn half the brain.

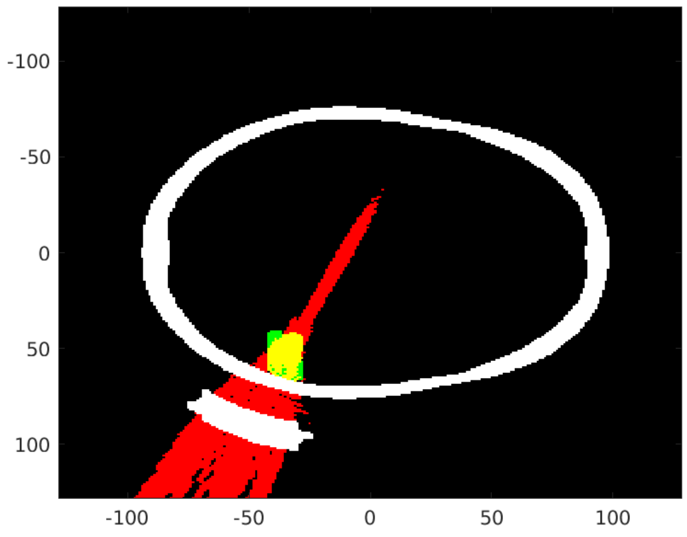


Figure 6.55: The best result with a score of 6.03.

6.5 Summary

This chapter described all the experiments that were performed with a heterogeneous medium. Unlike a homogeneous medium, where nothing prevents the ultrasonic waves, a large part of the waves in the heterogeneous medium is reflected or absorbed by the skull. For this reason, it is difficult to hit the entire target area.

From the results of experiments from the first test scenario described in Section 6.1 which was about hitting the target with one sonication, we can see that all algorithms have a problem with finding better solutions. Many results have a score of 84.42, which means that no point from the target area was hit. Of all three algorithms, the SA algorithm found the best results.

Experiments on the second test scenario, which are described in Section 6.2 achieved similar results as in the first scenario. For example, the genetic algorithm did not find any solution that was better than 84.42. The CMA-ES algorithm was a bit better, as it managed to find a few solutions whose fitness value was lower than 84. The best of all three algorithms was again the SA algorithm, which found a solution with a value of 65.77. This solution, along with the individual sonications, can be seen in Appendix D.

The results of experiments on the third scenario, which is about hitting the target area without hitting the penalty area with one sonication and are described in Section 6.3, again proved that the best algorithm is simulated annealing.

The results of additional experiments described in Section 6.4 serve rather to demonstrate what can be taken into account when optimizing the position of the transducer. Because only a few runs were performed, we cannot infer any conclusions from the results with 100% confidence. From the results of the experiment with a larger transmitter, we can state that it looks like a larger transmitter helped to find better results. This is because a larger transducer creates a larger focus and the skull does not absorb as many ultrasonic waves. From the results of the experiment with the penalty around the entire target area, we can say that the optimization could not cope with moving the transmitter away from the head. An experiment with an initial individual that is closer to the target area achieved better results than SA for the third scenario. Thus, it could be said that a better initial individual contributes to obtaining better solutions. From the results of the last additional experiment, the experiment with a lower pressure threshold, we can see that a larger part of the target area was hit. From the results of the last additional experiment, an experiment with a lower pressure threshold, we can see that most of the target area was hit. Unfortunately, in actual treatment, such results would not be good, as a large part of the skull outside the target is also hit.

In the future, it would be possible to add another optimization parameter, namely the power of the ultrasound transducer. Another possibility could be a heuristic for SA, where the initial individual would be generated so that the transducer axis always hits the target (the focus point would be selected as some point from the target area). Also, to make the results more meaningful and applicable, the problem can be solved in 3D and not just in 2D. However, the biggest problem is the speed of the calculation, because it takes several hours. The solution would be to use a larger number of nodes, use compiled C++ files of k-Wave package (only MATLAB files were used) or parallelize everything that can be parallelized.

Since the results from the simulated annealing are the best, a practical application in the treatment plan could look like the doctor enters some initial solution (as he approximately knows where the target area is) and SA would only improve this solution.

Chapter 7

Conclusion

Focused ultrasound is a very interesting and promising non-invasive method that can replace most invasive treatments in the future. This applies, for example, to the treatment of epilepsy, cancer, OCD, etc. However, as focused ultrasound must be very accurate, very difficult calculations precede the treatment itself. My task was to find out whether optimization using evolutionary algorithms could be used for these calculations.

The theoretical part of this work describes the solved problem of finding the correct position of the transducer and also describes the algorithms used to optimize the position of this transducer. These algorithms are Genetic Algorithms (GAs), Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) and Simulated Annealing (SA) and they are described in Chapter 2. Optimization problem is described in Chapter 3.

The practical part describes the implementation of individual scripts in MATLAB, experimentation with various algorithms and their parameters, as well as the evaluation of the results of efficiency and usability of these algorithms. Experiments were performed with homogeneous and heterogeneous medium. When homogeneous medium was use, the algorithms were tested on three benchmarks created for this purpose, with different sets of parameters. Each test was run twenty times and then the results were plotted. The experiments performed on homogeneous medium are shown in Chapter 5. The experiments for a heterogeneous medium are shown in Chapter 6.

As we can see in the previous two chapters, these algorithms proved to be able to solve the problem of finding the optimal position of the transducer, some better, some worse. This was much easier for a homogeneous medium than for a heterogeneous medium. Although the results of the experiments with the heterogeneous medium were not as good as I had hoped, I believe that better results can be achieved, for example by tuning the parameters. A better chosen fitness function could also lead to better results, as in my case the penalty area has the greatest weight, then the hit of the target area and finally the overlap of the head with the transducer.

Thanks to my work, I was able to contribute to the research of this non-invasive technique, and I hope that my work will serve as a starting point for further research as it is a very complex and time-consuming topic. In the future, it will be possible to examine in more detail all the experiments described in Section 6.4. Also the experiments were performed only in 2D, for the actual treatment it would be necessary to perform all experiments in 3D.

Bibliography

[1] Focused Ultrasound Foundation. Accessed: 2021-04-29. Available at: https://www.fusfoundation.org/mechanisms-of-action/neuromodulation.

[2] *Mechanisms of Action*. Focused Ultrasound Foundation. Accessed: 2021-04-29. Available at: https://www.fusfoundation.org/mechanisms-of-action/.

[3] Ajith, A. *Evolutionary computation, in Handbook of Measuring System Design*. John Wiley and Sons, 2005. ISBN 9780470021439.

[4] Arora, T. and Gigras, Y. A survey of comparison between various meta-heuristic techniques for path planning problem. *International Journal of Computer Engineering & Science*. 2013, vol. 3, p. 62–66. ISSN 22316590.

[5] Ashlock, D. *Evolutionary Computation for Modeling and Optimization*. Springer Science and Business Media, 2006. ISBN 9780387319094.

[6] Aubry, J.-F., Tanter, M., Pernot, M., Thomas, J.-L. and Fink, M. Experimental demonstration of noninvasive transskull adaptive focusing based on prior computed tomography scans. *The Journal of the Acoustical Society of America*. 2003, vol. 113, no. 1, p. 84–93. DOI: 10.1121/1.1529663.

[7] Baek, H., Pahk, K. J. and Kim, H. A review of low-intensity focused ultrasound for neuromodulation. *Biomedical Engineering Letters*. 2017, vol. 7, no. 2, p. 135–142. DOI: 10.1007/s13534-016-0007-y. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6208465/.

[8] Bidlo, M. *Lecture materials for course Evolutionary algorithms*.

[9] Bidlo, M. *Natural computing*. Lecture material for course Evolutionary algorithms. Available at: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FEVO-IT%2Flectures%2F01-NaturalComp.pdf&cid=12681.

[10] Bystritsky, A., Korb, A., Douglas, P., Cohen, M., Melega, W. et al. A review of low-intensity focused ultrasound. *Brain stimulation*. 2011, vol. 4, p. 125–36. DOI: 10.1016/j.brs.2011.03.007.

[11] Chlebík, J. *Evoluční návrh ultrazvukových operačních plánů*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/21776/.

[12] Coello, C., Veldhuizen, D. and Lamont, G. *Evolutionary Algorithms for Solving Multi-Objective Problems*. January 2007. ISBN 978-0-387-33254-3.

[13] Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001. Wiley Interscience Series in Systems and Optimization. ISBN 9780471873396. Available at: https://books.google.sk/books?id=OSTn4GSy2uQC.

[14] Dubinsky, T. J., Cuevas, C., Dighe, M. K., Kolokythas, O. and Hwang, J. H. High-intensity focused ultrasound: current potential and oncologic applications. *American Journal of Roentgenology*. 2008. DOI: 10.2214/AJR.07.2671.

[15] Hansen, N. *The CMA Evolution Strategy*. Available at: http://cma.gforge.inria.fr/.

[16] Hansen, N. The CMA Evolution Strategy: A Tutorial. *CoRR*. 2016, abs/1604.00772. Available at: http://arxiv.org/abs/1604.00772.

[17] Hansen, N. and Auger, A. *Tutorial CMA-ES - Evolution Strategies and Covariance Matrix Adaptation*. 2013. Available at: http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco2013-CMA-ES-tutorial.pdf.

[18] Heris, M. K. *CMA-ES in MATLAB*. Yarpiz. Available at: https://yarpiz.com/235/ypea108-cma-es.

[19] Iba, H. and Noman, N. *New Frontier in Evolutionary Algorithms*. Imperial College Press, 2012. ISBN 9781848166813.

[20] Jangid, S., Puri, R. and Kumar, T. Evolutionary Algorithms: A Critical Review and its Future Prospects. In:. International Journal of Trend in Research and Development (IJTRD), 2019. ISSN 2394-9333. Available at: http://www.ijtrd.com/papers/IJTRD20409.pdf.

[21] Jaroš, J. *Simulované žíhání*. Available at: http://www.fit.vutbr.cz/~jarosjir/groups/eva/sa.html.cz.

[22] Khan, N. A parallel implementation of the covariance matrix adaptation evolution strategy. *CoRR*. 2018, abs/1805.11201. Available at: http://arxiv.org/abs/1805.11201.

[23] Laarhoven, P. J. M. van and Aarts, E. H. L. Simulated annealing. In: *Simulated Annealing: Theory and Applications*. Springer Netherlands, 1987. ISBN 978-94-015-7744-1. Available at: https://doi.org/10.1007/978-94-015-7744-1_2.

[24] Mallawaarachchi, V. *How to define a Fitness Function in a Genetic Algorithm?* Available at: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4.

[25] Masárová, M. *Genetické algoritmy*. Brno, CZ, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/3454/.

[26] McLaughlan, J. R. *Diagram showing liver lesioning using a HIFU transducer*. 2008. Available at: https://en.wikipedia.org/wiki/High-intensity_focused_ultrasound#/media/File:Diagram_showing_liver_lesioning_using_a_HIFU_transducer_2.png.

[27] Park, K. *Fundamentals of Probability and Stochastic Processes with Applications to Communications.* January 2018. ISBN 978-3-319-68074-3.

[28] Park, M. J., Kim, Y.-s., Keserci, B., Rhim, H. and Lim, H. K. Volumetric MR-guided high-intensity focused ultrasound ablation of uterine fibroids: treatment speed and factors influencing speed. *European Radiology.* 2012, vol. 23, no. 4, p. 943–950. DOI: 10.1007/s00330-012-2665-1.

[29] Rezayat, E. and Ghodrati Toostani, I. Review Paper: A Review on Brain Stimulation Using Low Intensity Focused Ultrasound. *Basic and Clinical Neuroscience Journal.* 2016, vol. 7, no. 3. DOI: 10.15412/j.bcn.03070303. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4981830/.

[30] Sentewolf. *Concept of directional optimization in CMA-ES algorithm.* 2008. Available at: https://en.wikipedia.org/wiki/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png.

[31] Soni, N. and Kumar, T. *Study of Various Mutation Operators in Genetic Algorithms.* 2014, vol. 5, p. 4519–4521. Available at: http://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503404.pdf.

[32] Thede, S. An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges.* 2004, vol. 20.

[33] Treeby, E. B., Budiský, J., Wise, S. E., Jaroš, J. and Cox, T. B. Rapid calculation of acoustic fields from arbitrary continuous-wave sources. *Journal of the Acoustical Society of America.* 2018, vol. 143, no. 1, p. 529–537. DOI: 10.1121/1.5021245. ISSN 1520-8524. Available at: https://www.fit.vut.cz/research/publication/11411.

[34] Čudová, M., Treeby, E. B. and Jaroš, J. Design of HIFU treatment plans using an evolutionary strategy. In: *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion.* Association for Computing Machinery, 2018, p. 1568–1575. DOI: 10.1145/3205651.3208268. ISBN 978-1-4503-5764-7. Available at: https://www.fit.vut.cz/research/publication/11696.

[35] Wolpert, D. and Macready, W. No Free Lunch Theorems for Search. In:. 1995.

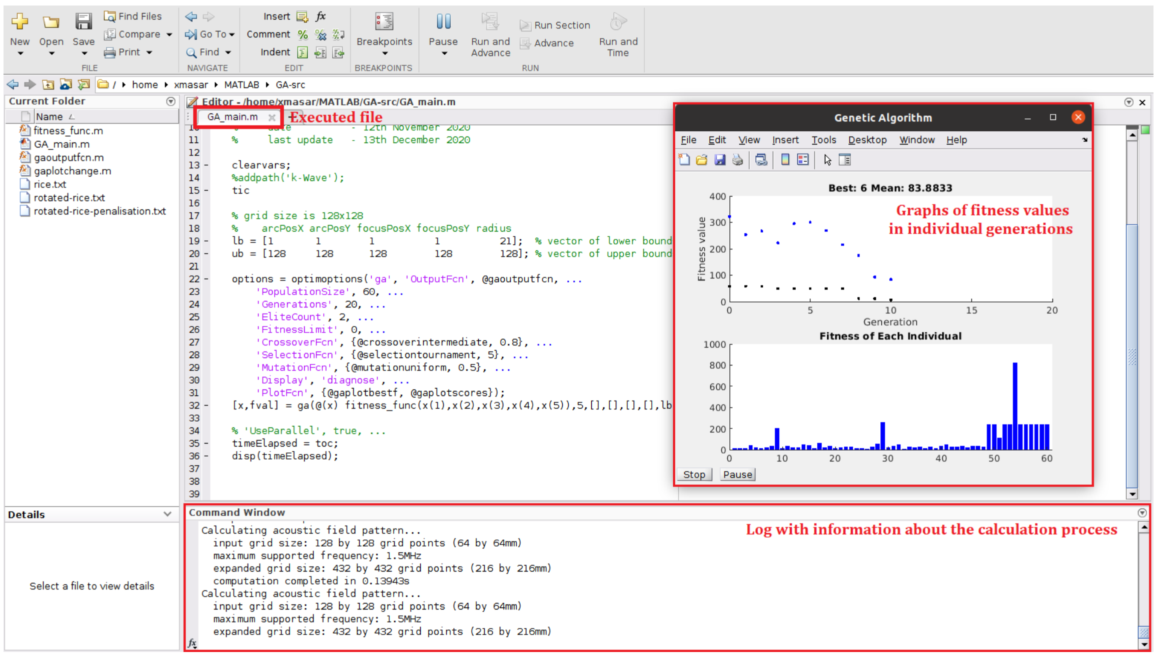
[36] Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation.* 1997, vol. 1, no. 1, p. 67–82. DOI: 10.1109/4235.585893.

[37] Yadav, P. and Prajapati, D. N. L. An Overview of Genetic Algorithm and Modeling. In: *International Journal of Scientific and Research Publications.* 2012, vol. 2. ISSN 2250-3153. Available at: http://www.ijsrp.org/research-paper-0912/ijsrp-p09107.pdf.

[38] Zbořil, F. V. *Lecture materials for course Soft computing.* Available at: https://www.fit.vutbr.cz/study/courses/SFC/private/.cs.

Appendix A

Example of GA run for a homogeneous medium


Editor - /home/xmasar/MATLAB/GA-src/GA_main.m
GA_main.m
Executed file
Current Folder
fitness_func.m
GA_main.m
gaoutputfcn.m
gaplotchange.m
rice.txt
rotated-rice.txt
rotated-rice-penalisation.txt
Details
Select a file to view details
Genetic Algorithm
Best: 6 Mean: 83.8833
Graphs of fitness values in individual generations
Fitness value
Generation
Fitness of Each Individual
Stop
Pause
Command Window
Log with information about the calculation process

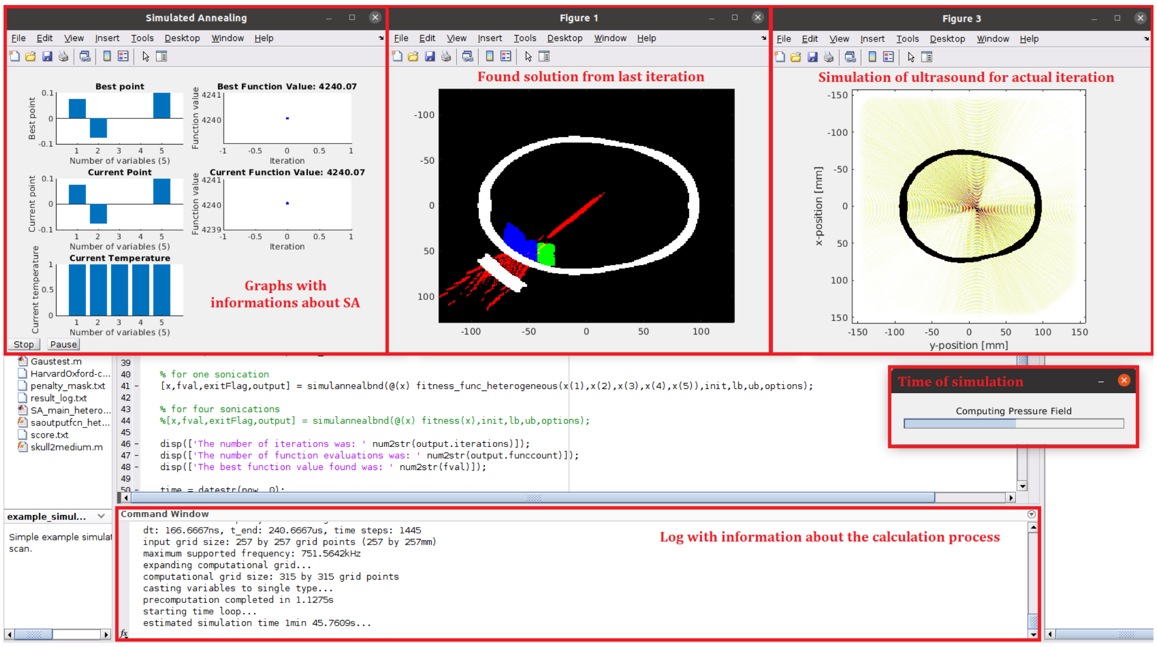
```
    %     date            - 12th November 2020
    %     last update     - 13th December 2020

    clearvars;
    %addpath('k-Wave');
    tic

    % grid size is 128x128
    %    arcPosX arcPosY focusPosX focusPosY radius
    lb = [1       1         1          1          21];  % vector of lower bound
    ub = [128     128       128        128        128]; % vector of upper bound

    options = optimoptions('ga', 'OutputFcn', @gaoutputfcn, ...
        'PopulationSize', 60, ...
        'Generations', 20, ...
        'EliteCount', 2, ...
        'FitnessLimit', 0, ...
        'CrossoverFcn', {@crossoverintermediate, 0.8}, ...
        'SelectionFcn', {@selectiontournament, 5}, ...
        'MutationFcn', {@mutationuniform, 0.5}, ...
        'Display', 'diagnose', ...
        'PlotFcn', {@gaplotbestf, @gaplotscores});
    [x,fval] = ga(@(x) fitness_func(x(1),x(2),x(3),x(4),x(5)),5,[],[],[],[],lb

    % 'UseParallel', true, ...
    timeElapsed = toc;
    disp(timeElapsed);
```

```
Calculating acoustic field pattern...
  input grid size: 128 by 128 grid points (64 by 64mm)
  maximum supported frequency: 1.5MHz
  expanded grid size: 432 by 432 grid points (216 by 216mm)
  computation completed in 0.13943s
Calculating acoustic field pattern...
  input grid size: 128 by 128 grid points (64 by 64mm)
  maximum supported frequency: 1.5MHz
  expanded grid size: 432 by 432 grid points (216 by 216mm)
```

Figure A.1: Demonstration of the appearance of the screen while running GA using a homogeneous medium.

Appendix B

Example of SA run for a heterogeneous medium



```
% for one sonication
[x,fval,exitFlag,output] = simulannealbnd(@(x) fitness_func_heterogeneous(x(1),x(2),x(3),x(4),x(5)),init,lb,ub,options);

% for four sonications
%[x,fval,exitFlag,output] = simulannealbnd(@(x) fitness(x),init,lb,ub,options);

disp(['The number of iterations was: ' num2str(output.iterations)]);
disp(['The number of function evaluations was: ' num2str(output.funccount)]);
disp(['The best function value found was: ' num2str(fval)]);

time = datestr(now, 0);
```

```
dt: 166.6667ns, t_end: 240.6667us, time steps: 1445
input grid size: 257 by 257 grid points (257 by 257mm)
maximum supported frequency: 751.5642kHz
expanding computational grid...
computational grid size: 315 by 315 grid points
casting variables to single type...
precomputation completed in 1.1275s
starting time loop...
estimated simulation time 1min 45.7609s...
```

Figure B.1: Demonstration of the appearance of the screen while running SA using a heterogeneous medium.

Appendix C

Examples of found solutions for a homogeneous medium

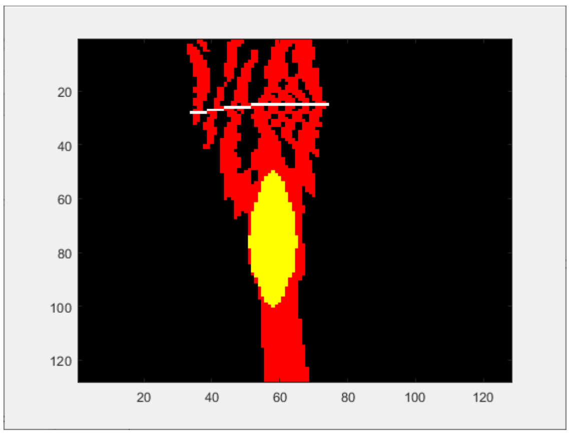


Figure C.1: Solution for a rice grain found by GA

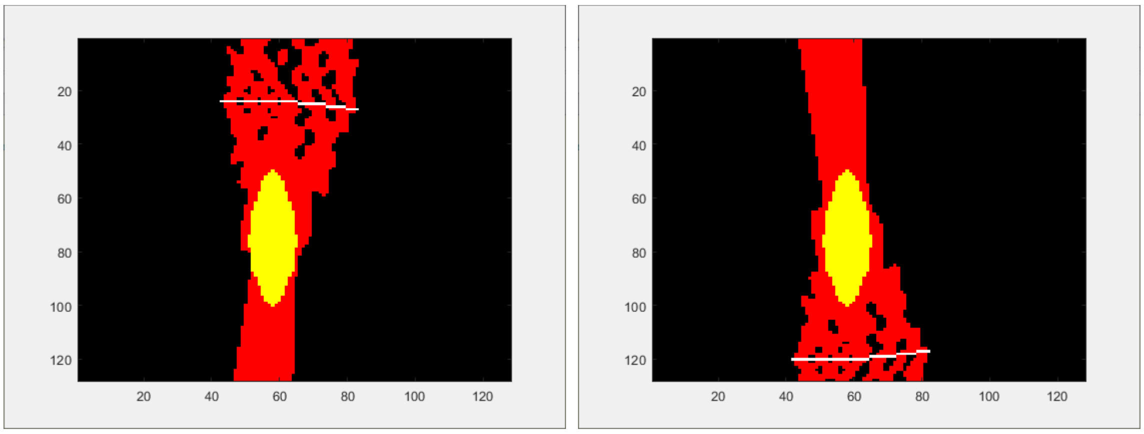


Figure C.2: Solutions for a rice grain found by CMA-ES

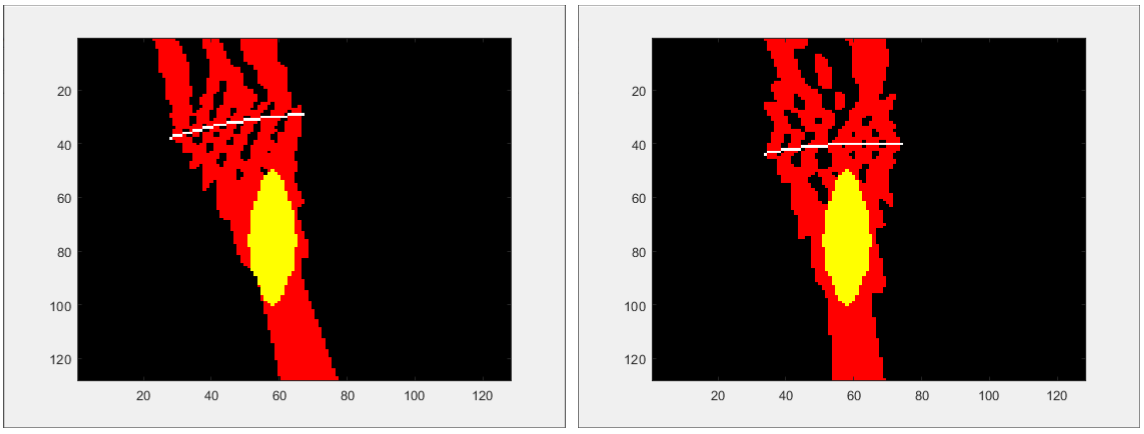


Figure C.3: Solutions for a rice grain found by SA

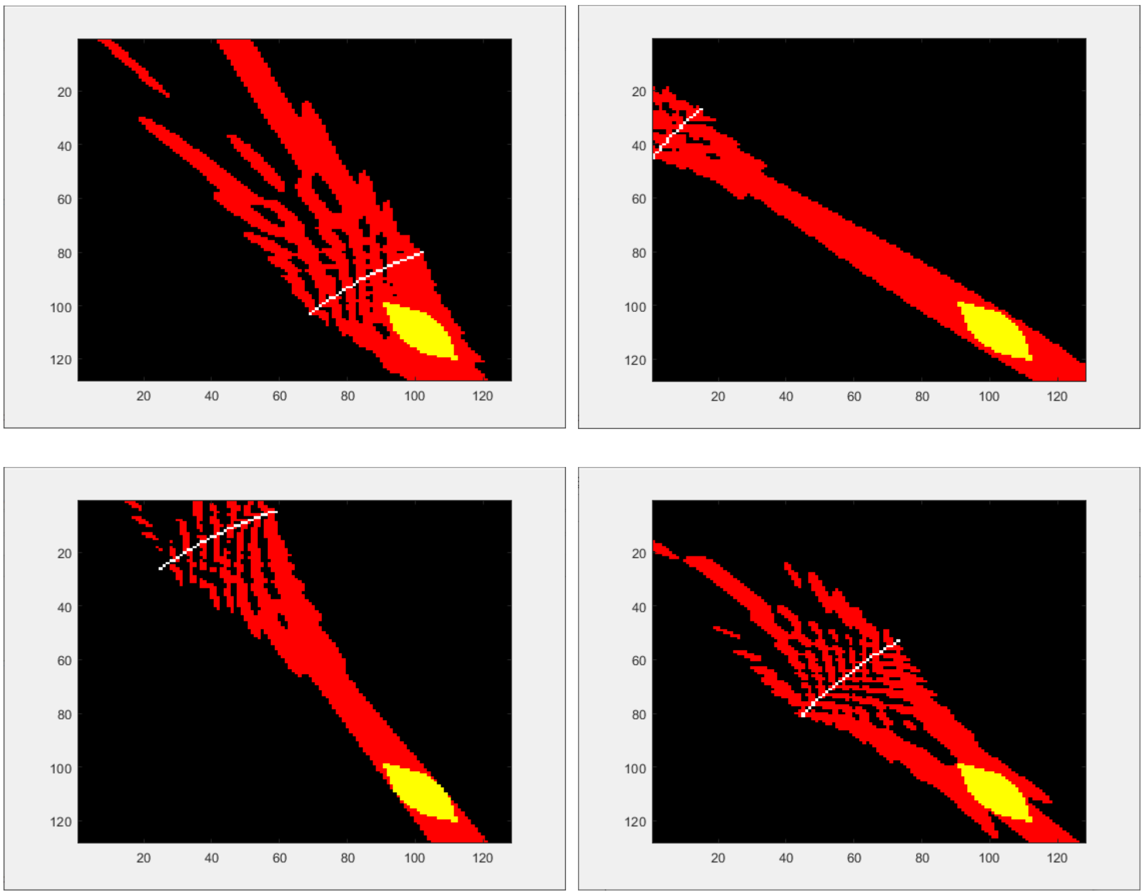


Figure C.4: Solutions for a rotated rice grain found by GA

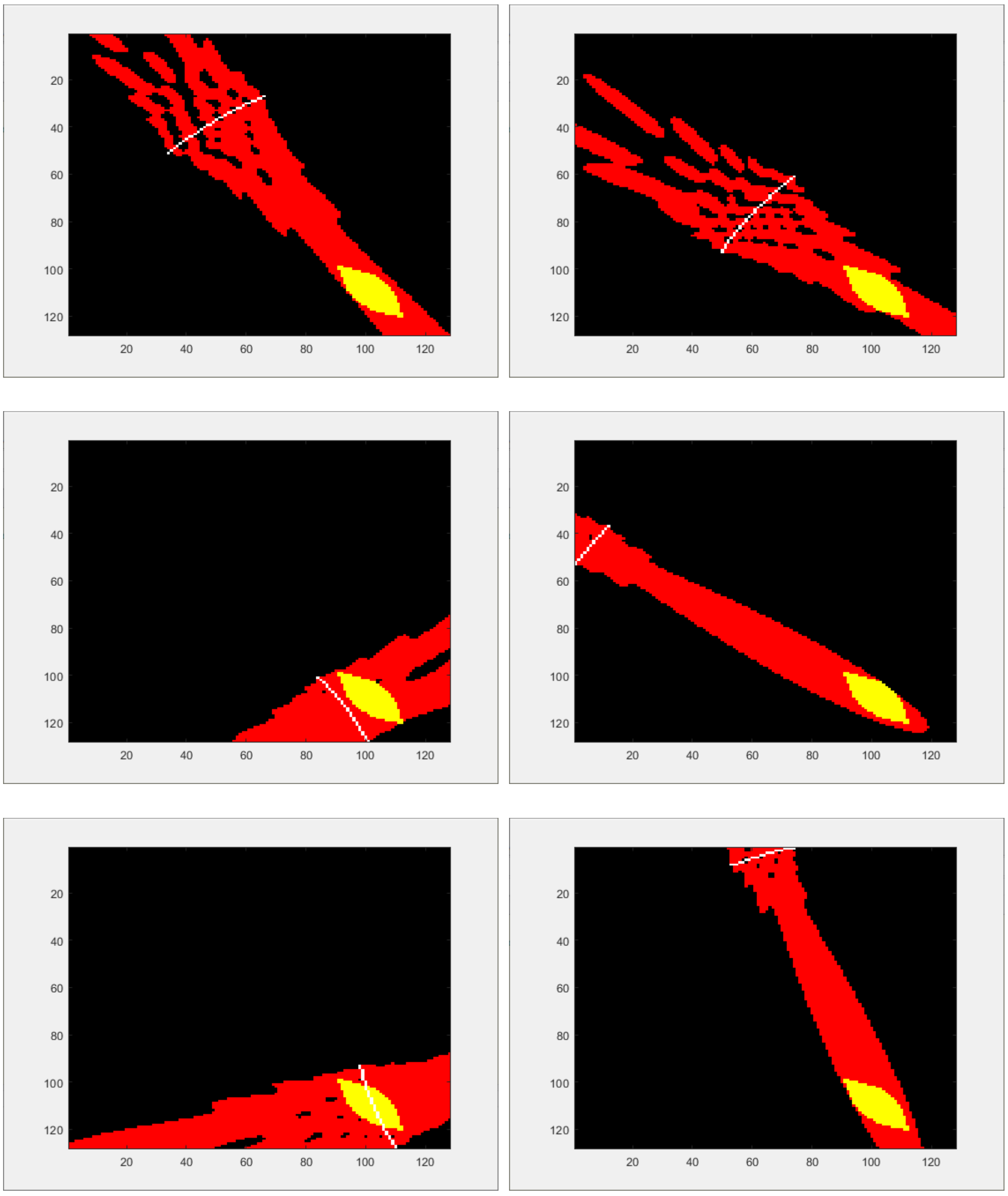


Figure C.5: Solutions for a rotated rice grain found by CMA-ES

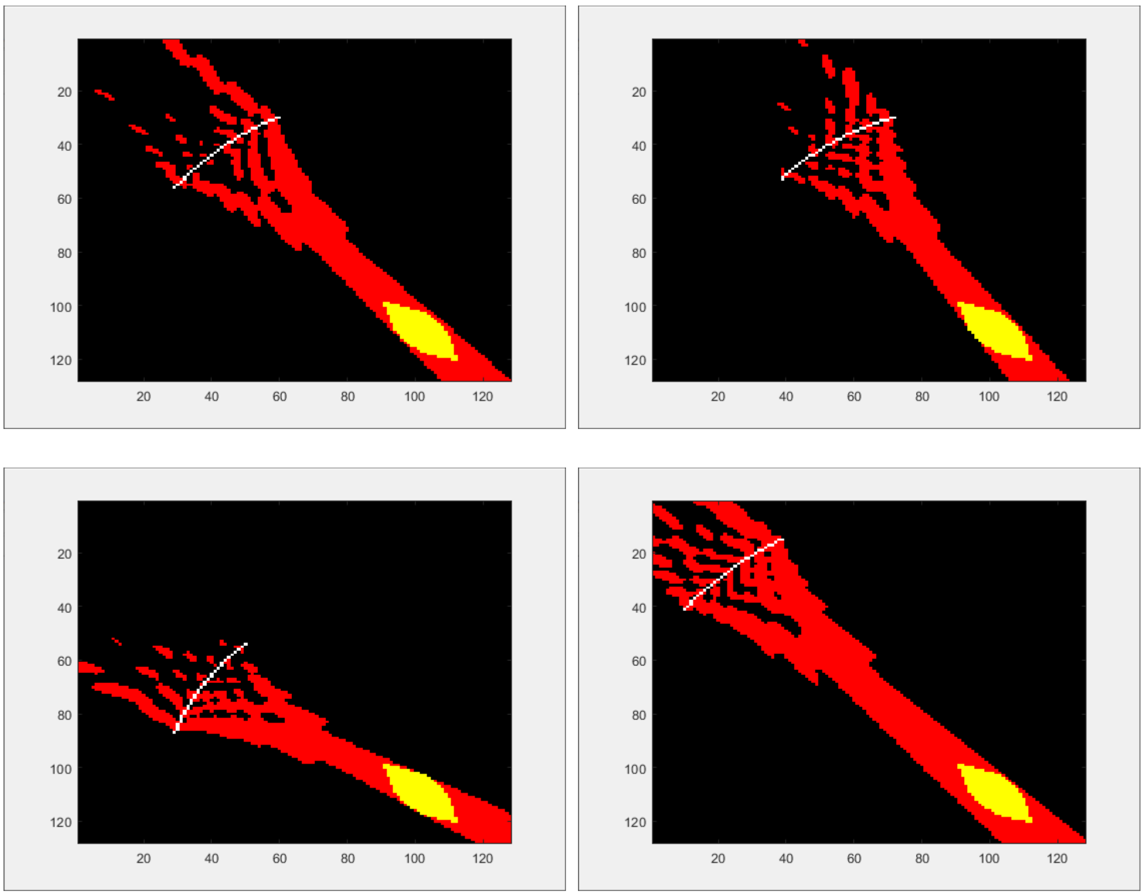


Figure C.6: Solutions for a rotated rice grain found by SA

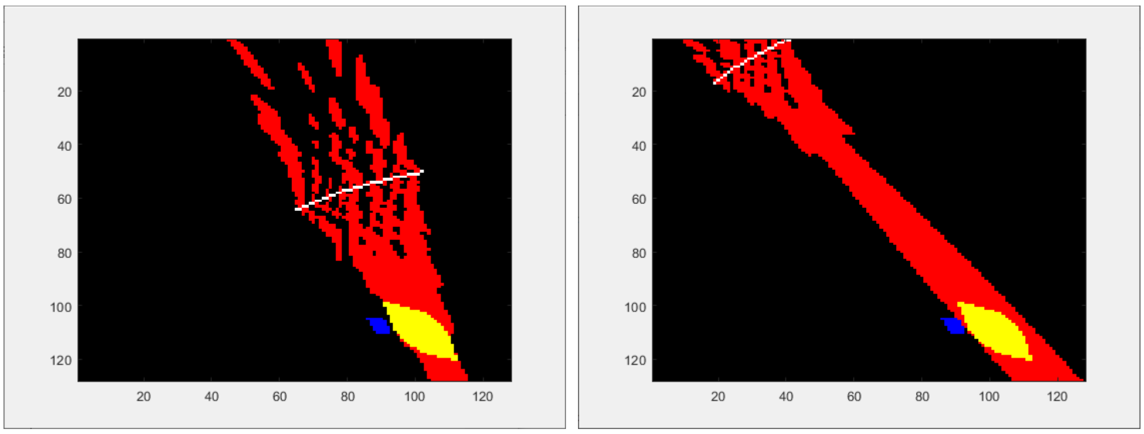


Figure C.7: Solutions for a rotated rice grain with penalty found by GA

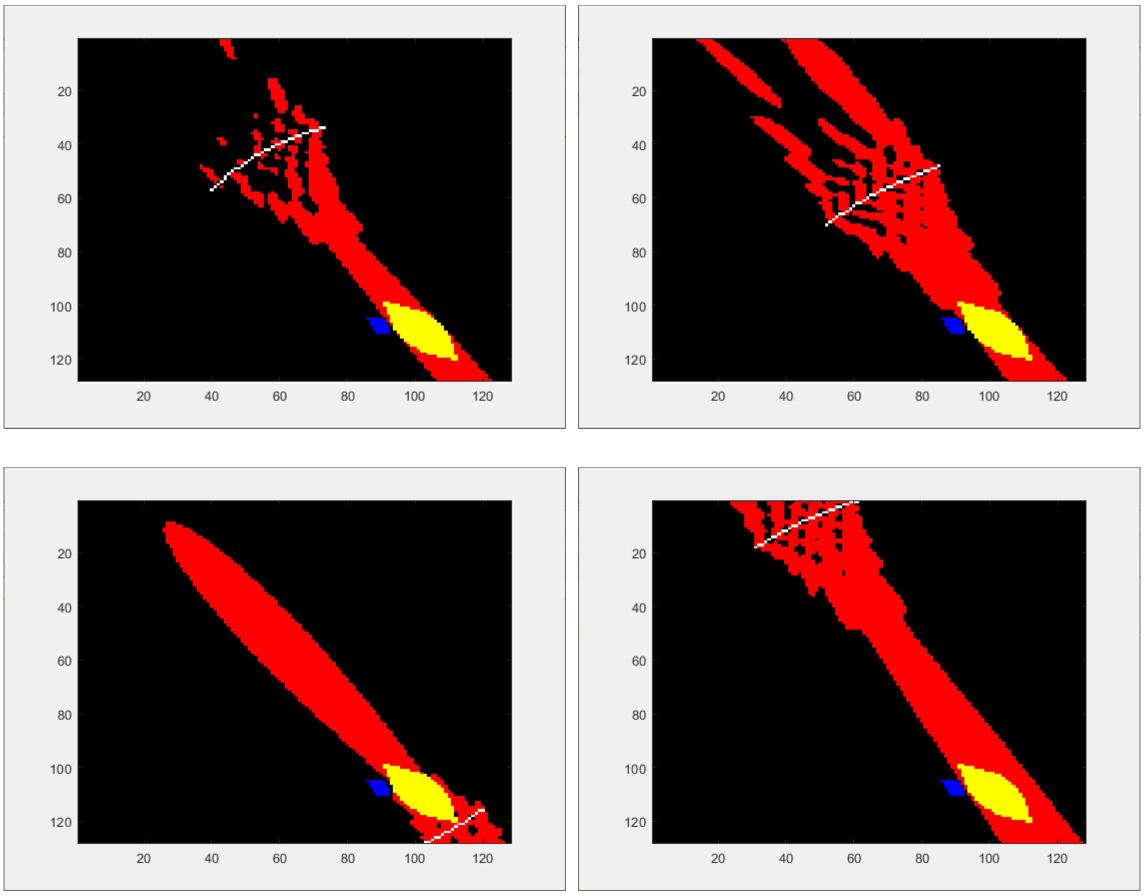


Figure C.8: Solutions for a rotated rice grain with penalty found by CMA-ES

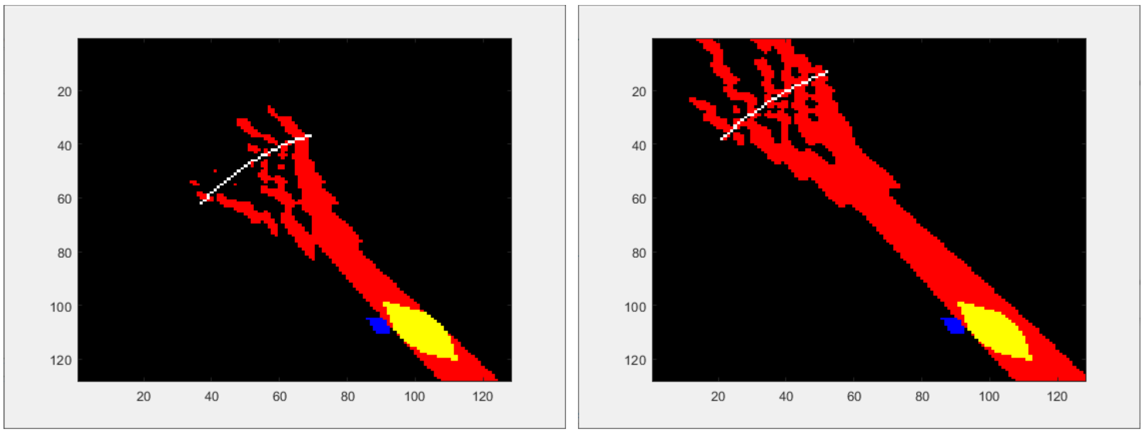


Figure C.9: Solutions for a rotated rice grain with penalty found by SA

Appendix D

Example of found solution for SA in a heterogeneous medium using four sonications

The figures below show four sonications that try to hit the same target. Of all the solutions, this solution had the best fitness function value of 65.77. Figure D.5 shows the resulting solution, where we can see all four sonications simultaneously.

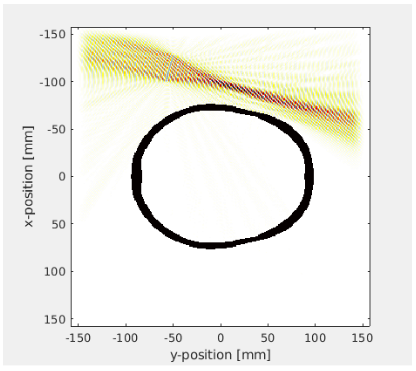


Figure D.1: Example of simulation of the first sonication.

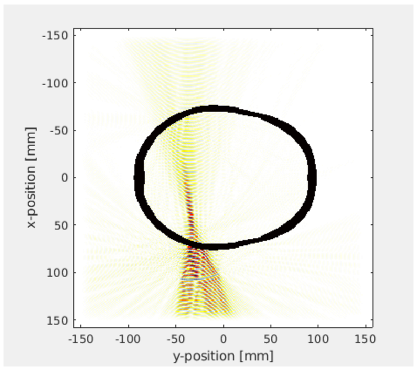


Figure D.2: Example of simulation of the second sonication.

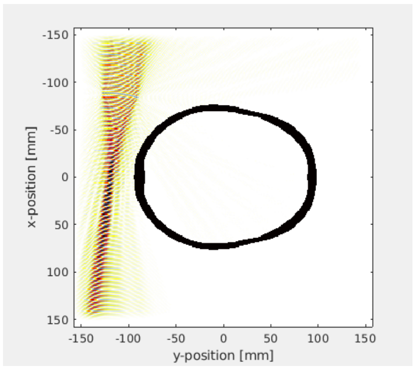


Figure D.3: Example of simulation of the third sonication.

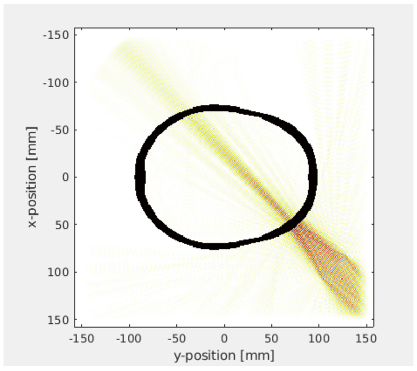


Figure D.4: Example of simulation of the fourth sonication.

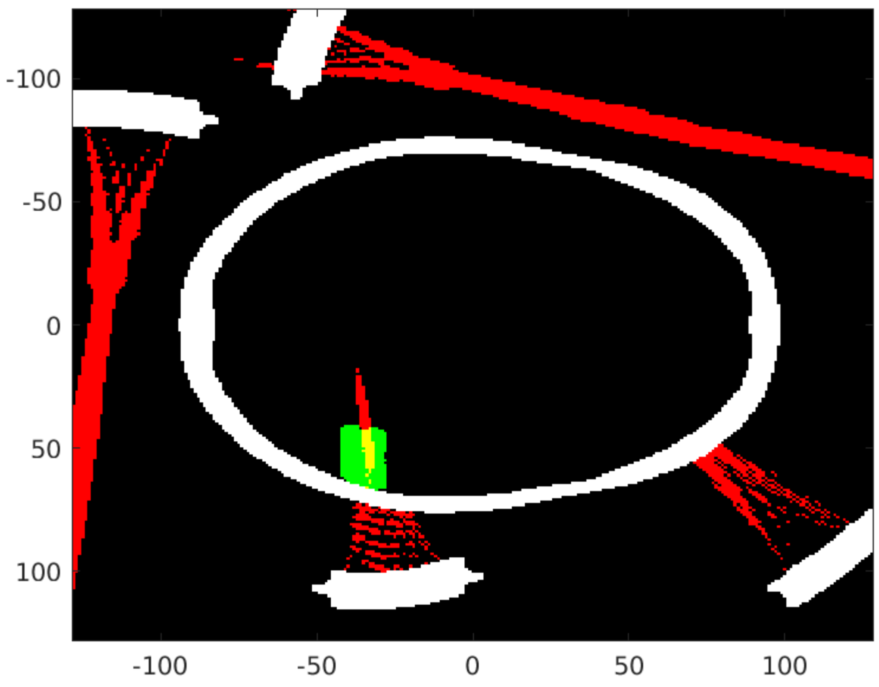


Figure D.5: The result of all four sonications. Only one sonication hit the target.