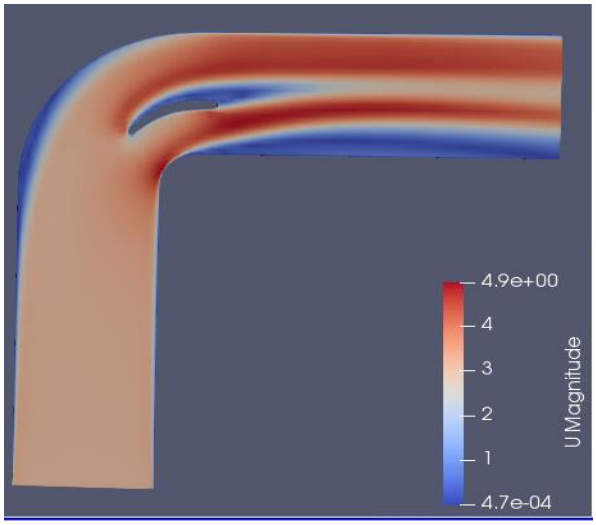
AALBORG UNIVERSITY

Maciej Janiszkiewicz

OPTIMIZATION OF HYDRAULIC SHAPES

WITH APPLICATION OF CFD, GENETIC ALGORITHM AND META MODELLING



Aalborg, 22.05.2020

AALBORG UNIVERSITY

Title:
Optimization of hydraulic shapes with application of CFD, genetic algorithm and meta-modeling

Project:
Master Thesis

Project duration:
01.02.2020 – 28.05.2020

Project group:
TEPE4-1012

Project Member:
Maciej Janiszkiewicz

Supervisors:
Torsten Berning
Szymon Beczkowski

Number of pages (incl. appendix): 58
Number of appendix pages: 0

Abstract:

The aim of the project is to evaluate the performance of the genetic algorithm and a specific type of metamodeling (called kriging) in an engineering optimization. An automated environment for the CFD analysis is built and connected with several types of optimization strategies. The key is to find an optimal genetic algorithm's setup, which means a good trade-off between acquired results and the optimization time. Finally, the performance of several optimization techniques is compared.

AALBORG UNIVERSITY

TABLE OF CONTENTS

1. INTRODUCTION

1.1 MOTIVATION

Nowadays, computer science is one of the most quickly developing branch of the modern industry. Figure 1 shows the Moore's Law, which predicts the increase of computational power of an average computer over years. It can be seen that the power computers may offer nearly doubles every 18 months.

The computational power is widely used in engineering, especially in simulations. This is especially visible in fluid motion science, where the Computational Fluid Dynamics (CFD) is considered to be a viable and trustworthy method of research.[2]

Initially, the CFD was a very time-consuming process. Further development of computational resources and the application of parallel computing allowed to perform it quicker and examine more and more sophisticated cases. Nowadays, CFD is widely used in static simulation, which gives an information about a non-changing geometry.

However, larger computational resources allow to find more applications for this kind of research. One of them is to connect CFD with optimization techniques, where a created script is able to perform simulations on its own and build optimal shapes independently. This approach means large number of individual simulations performed, which makes it very time consuming. Because of that, existing techniques should be investigated to find a setup of genetic algorithm, which allows to perform automated CFD optimization both quickly and accurately. If both of these conditions could be met, artificial intelligence might be considered a powerful tool in solving fluid motion related problems.

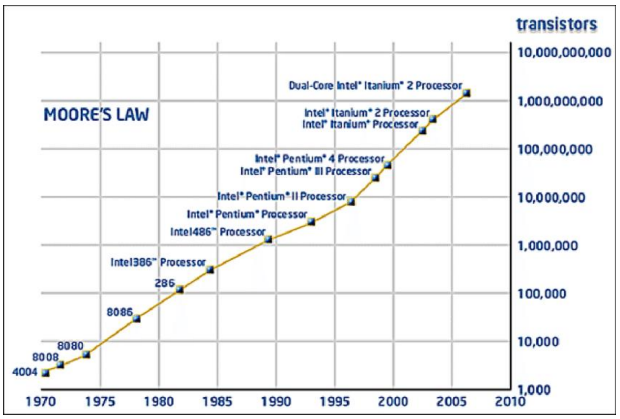


Figure 1. Growth of computation power over years[1]

1.2 COOPERATION WITH GRUNDFOS

Grundfos is one of the biggest pump developer worldwide. The cooperation with it means both the access to specific type of optimization and CFD software, and to the internal network of the company. The investigation will be carried on simple pipe systems to decrease the time required to perform a single simulation. These shapes are a Y-junction (1st case) and a baffled bend (2nd case).

1.3 THE PURPOSE OF THE PROJECT

The aim of this project is to examine the performance of different genetic algorithm features and evaluate the viability of the metamodeling in the process of genetic algorithm performance improvement. The viability between the connection of CFD, genetic optimization and metamodeling will be tested.

1.4 LIMITATION OF RESEARCH

In both examined pipe systems the amount of dynamical geometry is limited. Only specific areas in the pipes might change. While the impact of analysis and these changes will be examined, CFD is assumed a fully-trustworthy method of analysis and the accuracy of it will be not examined, no physical model will be built. Furthermore, because of a large number of simulation calls, mesh independency studies will be limited only to the initial cases.
Only few features of the genetic algorithm will be examined and results will be based mostly on low number of runs. According to the fact that the genetic algorithm is an optimization technique with high amount of randomness, the results cannot be treated as a general trend. Furthermore, the optimization factors like types of sampling or selectors will be only cursorily explained. In many cases, default values of specific factors will be used to avoid too in-depth approach.

Finally, only factors related to fluid motion are examined. The material strength of a baffle and the possibility of its creation in physical world is out of the scope of this research.

1.5 METHODOLOGY

The project consist of several steps, which will be later connected into a fully-working simulation and optimization environment. In general, all of these steps will be applied in two cases.

- In the first case, the Y-junction connects two streams: Hot and cold, tackling each other with different velocities. The goal is to model the shape of baffle inside and the angle between pipes to achieve the best trade-off between pressure loss and the uniformity of a temperature profile at the outlet.

- The second case is a baffle modeling inside of a pipe’s bend – The goal is to optimize baffle shape to achieve a good trade-off between a pressure loss increase caused by the baffle and the uniformity of velocity profile at the outlet.

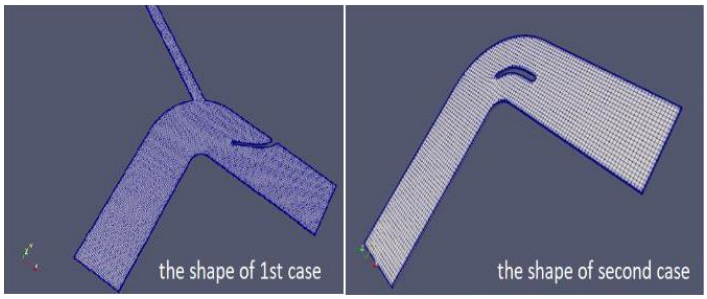


Figure 2. The shapes of examined cases

These steps are listed below:

1. First, an automated meshing tool has to be created. The key to obtain that is an opensource CAD software called “Salome”, which allows to export a modifiable script. This file can be changed easily to create many different geometries, which is especially useful when those changes are performed automatically, for example by a programmed script.

   Instead of just constant values, several variables will be introduced into it, allowing to steer geometry creation by changing values of these variables. The geometry will be passed to an OpenFOAM-based mesher.

   When the mesh is done, the CFD analysis will be conducted. Both the initial conditions and other simulation parameters (like the convergence criteria etc.) will not change over the entire process.

   At the end, postprocessing will be conducted and calculated values (pressure loss, temperature distribution and velocity distribution) will be saved in a text file.
   Entire process described in point 1 will be gathered in a single script, called a “script1”.

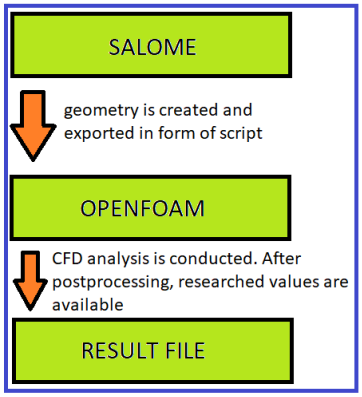


Figure 3. Performance of Script1

2. At the second step, an optimization process will be introduced. Applying a Python-based library called “Platypus”, a specific type of genetic algorithm “NSGAIII” will be used and named as “Script2”. The Script1 will be passed to the Script2.

Script2 will be able to create a geometry “guess” and invoke Script1 to perform the CFD analysis. Postprocessed values will be passed to the Script2, which will use them to calculate values of objective functions (in terms of genetic optimization, often called “fitness functions”). This way, a genetic optimization might be performed using the CFD as a “shape-evaluation” tool.

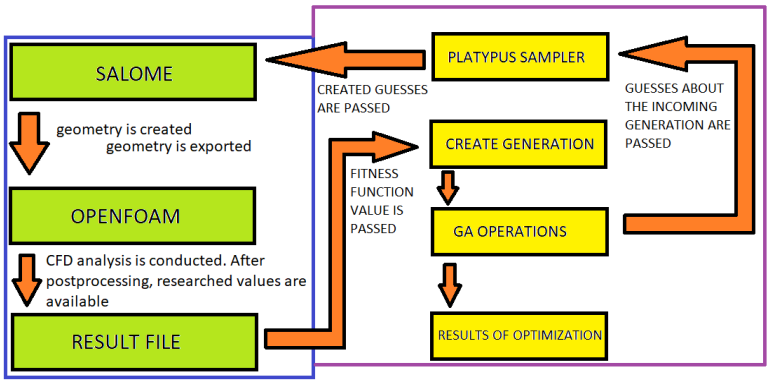


Figure 4. Performance of Script2

3. Next, results of this optimization will be used in a second type of genetic optimization, carried in an Optislang software. An another simulation environment will be built and the genetic optimization will be conducted.
4. Finally, the metamodeling will be introduced. Based on values computed by CFD, the kriging will try to mimic its performance in significantly reduced amount of time. Its accuracy will be evaluated.

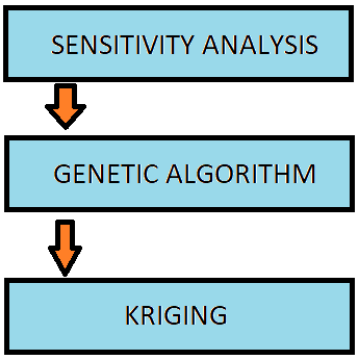


Figure 5. The performance of model built in Optislang

2. COMPUTATIONAL FLUID DYNAMICS

The nomenclature for this chapter is listed below:

p – pressure
u, v, w – velocities in specific cartesian coordination
τ_{ij} – viscous stress
e_0 – energy (usually expressed as temperature)
x, y, z – direction of the flow
ρ – density
q – energy input
ϕ – researched flow property
t – time
τ_{ij} – viscous stress
μ – molecular viscosity
S_ϕ – source term
τ – time constant of turbulence
ν_t – kinematic viscosity
k – turbulent energy
ω – turbulent dissipation
δ_k / δ_ω – constants for description of k and ω
y^+ - dimensionless distance to the wall
u^+ - dimensionless velocity
τ_ω – wall shear stress
u_T – friction velocity
K – Von Karman constant

2.1 INITIAL ASSUMPTIONS

A computational fluid dynamics (CFD) approach was used to explore the flow pattern inside both of cases.

In order to shorten simulation time, the process is assumed steady, which means that there is no variation in flow properties over time. Furthermore, the flow is considered viscid and turbulent. This turbulence was modelled instead of resolving, which comprises the simulation time and accuracy.

In both cases, CFD analysis will be carried out only in 2 dimensions. This is related to the purpose of this analysis – during the optimization phase the process of CFD will be repeated hundreds of times.

2.1.1 Initial assumptions of the first case

In terms of 1st case, the sought features are:

- The pressure loss between the middle of larger pipe's inlet and the outlet
- The variation of temperature at the outlet (how well was the mixing performed)

The baffle was placed in the system of pipes (Y-junction). The working fluid is water in normal atmospheric pressure, pumped with different, uniform velocity through both of the pipes. Both inlets provide the system with a fluid of different temperature. All the properties of fluid which were used in further research are related to these conditions.

2.1.2 Initial assumptions of the second case

In terms of 2nd case, the researched features are:

- The pressure loss between the middle of a pipe and the outlet
- The velocity profile at the outlet

The baffle was placed into a pipe bend of the L-shape. The baffle's role is to steer the fluid and achieve the most uniform velocity distribution at the outlet while also decreasing the pressure loss.

2.2 GENERAL PROCEDURE OF CFD

CFD allows to model fluid flow with a numerical analysis, which allows to research sophisticated fluid behaviors. According to the Eulerian frame of reference, the entire flow area may be divided into a fixed number of small volumes, for which the properties of fluid will be examined. For three – dimensional, time - independent phenomena, this approach may be expressed as [2]:

$$\phi = \phi(x, y, z, t)$$

After division of the flow domain to a computational mesh, a numerical solver solves the conservation equations for flow features. Depending on the chosen software and its properties, a list of conservation equations must be satisfied for the entire flow domain. These values are computed sequentially for every single cell, while the solutions are considered as an input to subsequent equations. This process is repeated, and every repetition is called an "iteration".

Computational Fluid Dynamics (CFD) is the simulation of fluids engineering systems using modeling (mathematical physical problem formulation) and numerical methods. The governing equation of CFD are Navier-Stockes equations. They are basic equation of motion for viscous, heat conducting fluid.
Generally, the term "Navier-Stockes equation" relates to three equations which describe the conservation rules for continuity, momentum and energy.

1) Continuity equation:

$$\frac{\partial p}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho u_j\right] = 0 \qquad 1.$$

2) Momentum conversation:

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left[\rho u_i u_j + p\delta_{ij} - \tau_{ji}\right] = 0 \qquad 2.$$

It can be seen that the change in acceleration equals to the sum of gravity term, pressure term and velocity diffusion term, represented by the viscosity. However, it can be also described in more general form, which comes directly from the force balance on a single fluid element.

Sometimes the energy equation is solved for as well in order to calculate the temperature distribution, e.g. in cases where heat transfer plays a role. The conservation equation for energy is:

3) Energy conservation:

$$\frac{\partial}{\partial t}(\rho e_o) + \frac{\partial}{\partial x_j}\left[\rho u_j e_o + u_j p + q_j - u_i \tau_{ij}\right] = 0 \quad (3.)$$

The equations listed above may be completed into equation 4 – The General form of NS-equation. The sum of pressure gradient and viscous term (right side) is equal to the change of local acceleration and convective term.

4) General form of NS-equation

$$\rho\left[\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u^2) + \frac{\partial}{\partial y}(uv) + \frac{\partial}{\partial z}(uw)\right] = -\frac{\partial p}{\partial x} + \mu\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right] \quad (4.)$$

According to Stokes hypothesis, the viscous stress τ_{ij} may be considered as a product of molecular viscosity μ and local velocity gradients called strain rates.

$$\tau_{ij} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \quad (5.)$$

Especially in CFD, the entire set of equation may be expressed as:

$$\rho\frac{\partial \phi}{\partial t} + \rho\, div(\vec{u}\phi) = div(\Gamma grad\phi) + S_\phi \quad (6.)$$

This approach allows to resolve the main flow. However, the key to get a realistic simulation of the flow is to model a turbulence. In this approach, turbulence was modelled with RANS. First, the mean velocity is introduced:

$$\overline{u} = \frac{1}{\tau}\int_0^\tau u(t)\, dt \quad (7.)$$

Then, the total velocity of fluid element is represented by a sum of time-invariant main flow velocity and random fluctuations, expressed as u', v', w' (for a three-dimensional flow).

$$u(x,y,z,t) = \overline{u}(x,y,z) + u'(x,y,z,t) \quad (8.)$$

Putting it into the Navier – Stokes equations yields:

$$\rho\left[\frac{\partial}{\partial x}\left(\overline{u}^2 + \overline{u'^2}\right) + \frac{\partial}{\partial y}\left(\overline{u}\overline{v} + \overline{u'v'}\right) + \frac{\partial}{\partial z}\left(\overline{u}\overline{w} + \overline{u'w'}\right)\right] = -\frac{\partial \overline{p}}{\partial x} + \mu\left[\frac{\partial^2 \overline{u}}{\partial x^2} + \frac{\partial^2 \overline{u}}{\partial y^2} + \frac{\partial^2 \overline{u}}{\partial z^2}\right] \quad (9.)$$

In this approach, shear stresses relate to fluctuations which allows to model the main flow in turbulence. However, one of the most challenging part of turbulence modelling is near-wall region. To solve this, the $k-\omega\, SST$ model was applied.

$k-\omega\, SST$ is a model of turbulence which connects the advantages of k – ϵ and k – ω approaches. While k – ϵ focuses on problems with large distance from the wall, k – ω excels in

models where the distance from the wall is smaller. It consists of two separate equation for k, which is turbulence kinetic energy and omega, which is turbulent energy dissipation.

Kinetic energy:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = P - \beta k\omega + \frac{\partial}{\partial x_j}\left[(\nu + \delta_k k \nu_t)\frac{\partial k}{\partial x_j}\right] \quad 10.$$

Turbulence dissipation:

$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = \alpha S^2 - \beta\omega^2 + \frac{\partial}{\partial x_j}\left[(\nu + \delta_\omega k \nu_t)\frac{\partial \omega}{\partial x_j}\right] + 2(1 - F_1)_{\omega_2} + \frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i} \quad 11.$$

Where P, F_1, F_2, β are closure coefficients dependent on k and omega, and ν_t is kinematic eddy viscosity – a parameter calculated also directly from k and omega values and the distance from the wall.

To model the flow near the wall, it is necessary to introduce a standard wall function approach. In fluid dynamics, the law of the wall states that turbulent flow's average velocity is proportional to the logarithm of the distance from that point to the wall. Furthermore, the area of the flow may be divided into an area of viscous sublayer, where the viscous forces play important role in fluid behavior and the buffer layer, which represents an area of transition between viscous region and "free-stream" flow. To separate these areas, values of y+ and u+ are used.[2]

$$u^+ = \frac{1}{K}\ln y^+ + C \quad 12.$$

$$y^+ = y u_T / v \quad 13.$$

$$u_T = \sqrt{\frac{\tau_\omega}{\rho}} \quad 14.$$

$$u^+ = \frac{u}{u_T} \quad 15.$$

To resolve the viscous sublayer properly with wall function approach, its necessary to achieve y+ values between 20 and 500. This value gives a hint to the research of computational mesh size.[2]

According to the schema, the values of velocity achieved through momentum equation are used as starting guesses of pressure. While the values of pressure and velocity depend on each other, only the correct values of pressure may result in fulfilled set of equations. If the convergence cannot be reached after all the computations, the values for a pressure are updated with "results" obtained via the velocity computation and the entire algorithm iterates again.

The CFD software calculates the difference between specific conservation equations in succeeding iterations. This value is called a "residual". Generally, in a properly set simulation the value of residuals should decrease with the number of iterations. When the normalized difference is low enough, the iteration procedure stops, and received values may be considered as a result of the entire process. To enhance the ability to converge, an under-relaxation factor is implemented. Its application allows to use only a part of values achieved in previous iterations, which increases the number of iterations required to obtain converged

solutions but reduces the residuals in order to keep the simulation stable. When the size of iterations cross a fixed value, the simulation breaks, and its results cannot be considered trustworthy.

Depending on the solver and software, conservation equations may vary. However, they always deal with several “basic” flow properties – velocity, pressure, temperature and turbulence.

In this research, a solver named “simpleFoam” and “buoyantSimpleFoam” was used. They work with four variables describing velocity in cartesian coordination system, one variable for pressure, one for temperature and – depending of chosen turbulence model – a set of variables describing turbulence.

First, the solver attempts to solve momentum equation. It assumes that during the flow, the entire mass is conserved, which means that no creation and no destruction of mass occurs. In a constant-density flow, this means that flowrate must be constant. Once is solved, this equation delivers a velocity field $u*$, which is considered as a guess – it does not satisfy continuity equation.

Next, the continuity and momentum equations are used to build an equation for the pressure. Solution of this equation gives access to pressure field p^n. Inserted into momentum equation, a corrected field of velocity u is calculated.

After that, simpleFoam (case 2) or buoyantSimpleFoam (case 1) attempts to solve turbulence and considers the velocity field u as an input to this process. In this case, the turbulence is modelled with Reynolds-averaged Navier–Stokes equations (RANS) with feature called “Wall function”.

2.3 MESH INDEPENDENCY STUDIES

The purpose of mesh independency studies is to find the smallest size of mesh which gives trustworthy results. Initially, a mesh made of low number of cells is refined as long, as the results obtained through further refinement does not change by a significant amount.

Mesh independency studies were performed for both initial geometries. In a classical CFD approach, the mesh independency should allow to reveal the smallest size of a mesh which allows to obtain proper results. However, this approach does not work in this case. This is caused by the fact that during the application of genetic algorithm, the amount of change performed might significantly affect the character of the flow, including the values of velocities, pressure and others. This was often leading to instability of optimization algorithm performance. To address the problems, the initial mesh size was refined until one optimization run was successfully executed.

During the execution, the value of $y+$ was recorded. Depending on the geometry shape, its value could either increase or decrease dramatically, so the mesh was refined as long as the average value for $y+$ could be keep within an interval of 20>y+>500 to keep proper turbulence modelling. While the average $y+$ returns generally all the time the same value, there are some areas where $y+$ can be either very high or very low.

To achieve trustworthy results with the the $k-\omega$ Wall-function model, the value of $y+$ should be kept between 20 and 500.[2]

2.3.1 Mesh independency studies – the first case

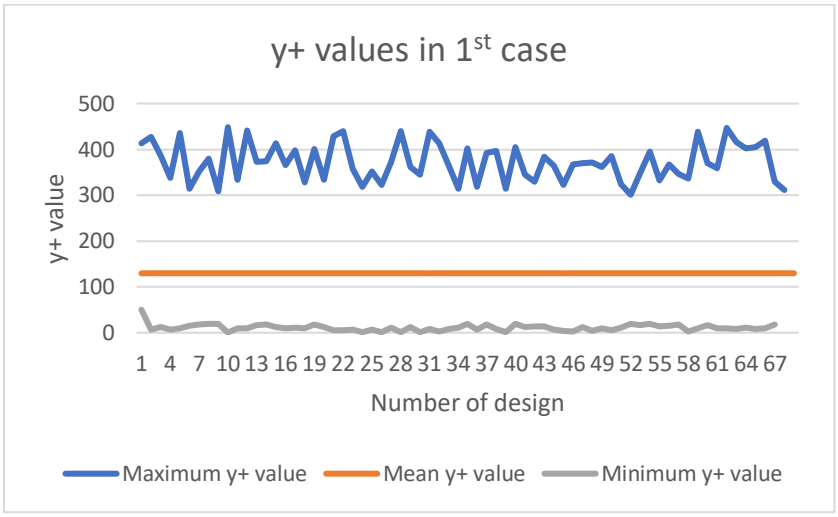


Figure 6. y+ values obtained in the first case

While the minimum $y+$ value might be too low sometimes (in small number of regions), average $y+$ is around 130 and maximum $y+$ never cross the value of 500.

For this specific case consisting of 13 972 cells, the convergence of pressure is shown at Figure 7.

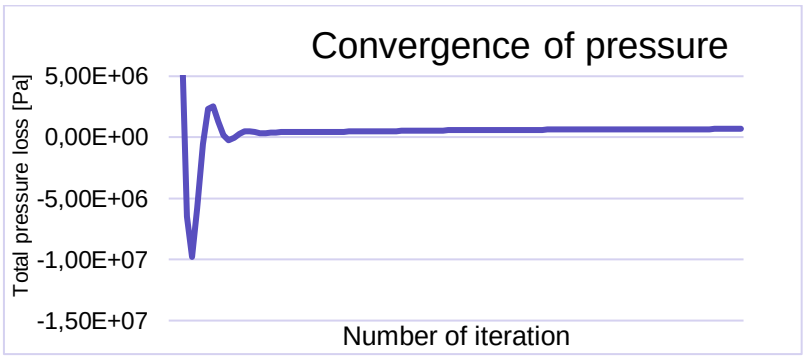


Figure 7. Convergence of first case in terms of pressure

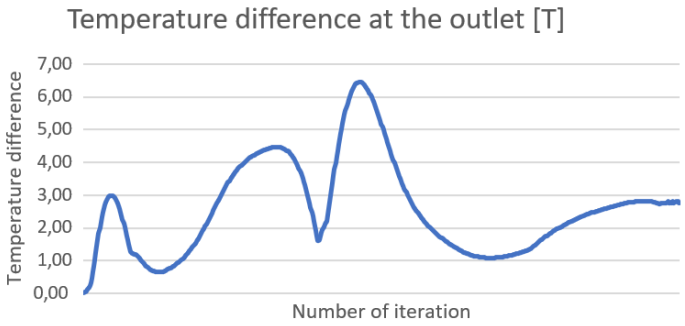


Figure 8. Convergence of first case's initial geometry in terms of temperature distribution at the outlet

Figure 9 shows the y+ value distribution over the ribbon of the shape. It might be seen that the vast majority of cells is kept between 30>y+>200 value, while some areas (f. ex. surroundins of the smaller inlet, the bottom of the larger pipe) are exposed for larger values (up to 390).

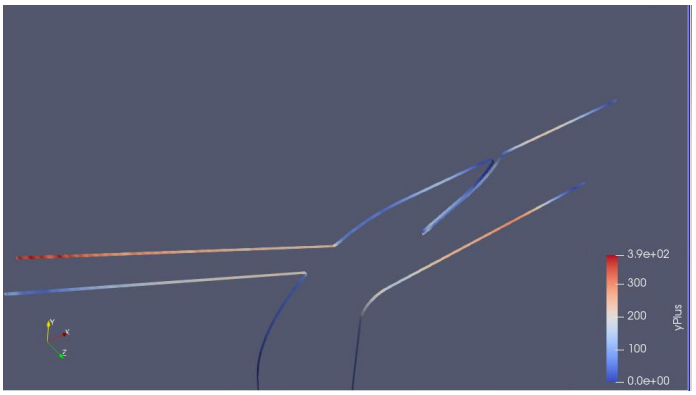


Figure 9. y+ value distribution of the first case

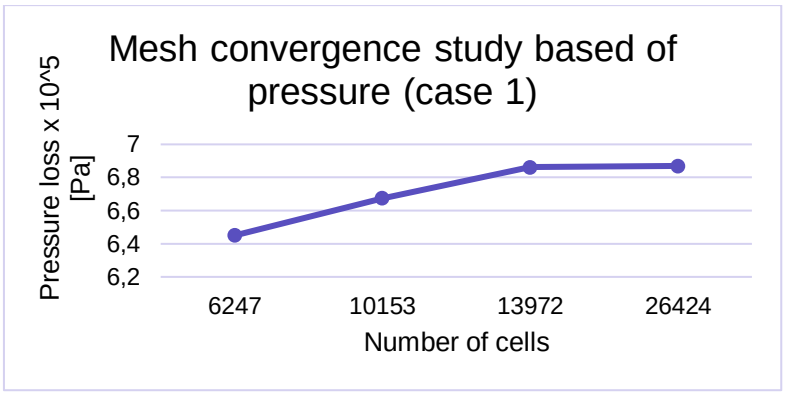


Figure 10. Mesh convergence studies of the first case based on pressure

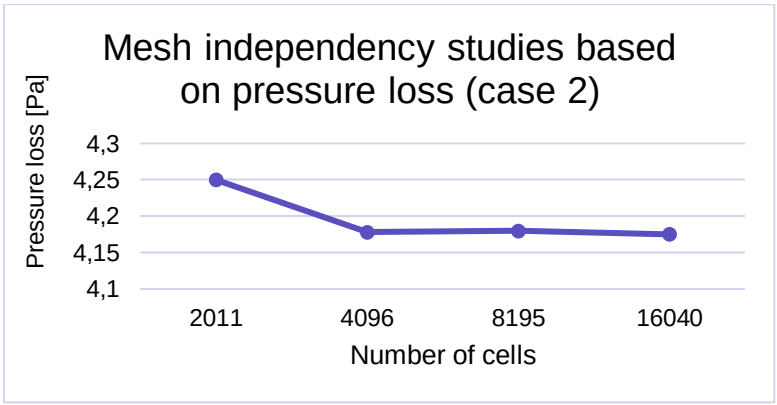


Figure 11. Mesh convergence of the first case based on temperature distribution

2.1.2 Mesh independency studies – the second case

In terms of 2nd Case, $y+$ is held most of the time at the very low values (between 0 and 50), but there are some areas (mostly around the inlet) where red color (which means values around 350) may be seen. Additionally, the low number of cells allow to perform one simulation over several seconds. In comparison to that, more time consuming is the process of mesh creation, so the entire iteration takes around one minute. To keep the high flexibility of an initial mesh, it was left in this state.

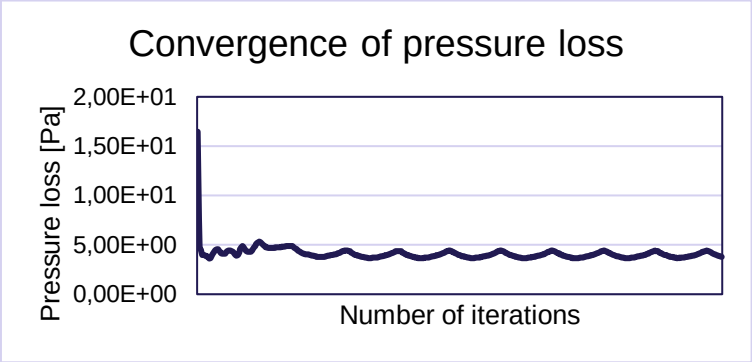


Figure 12. Mesh convergence of 2nd initial case based on pressure loss

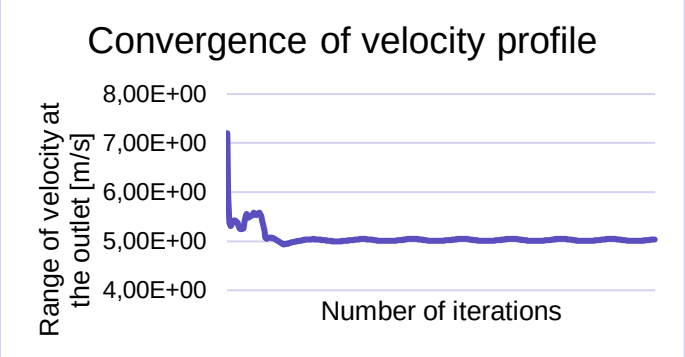


Figure 13. Mesh convergence of 2nd initial case based on the velocity profile at the outlet

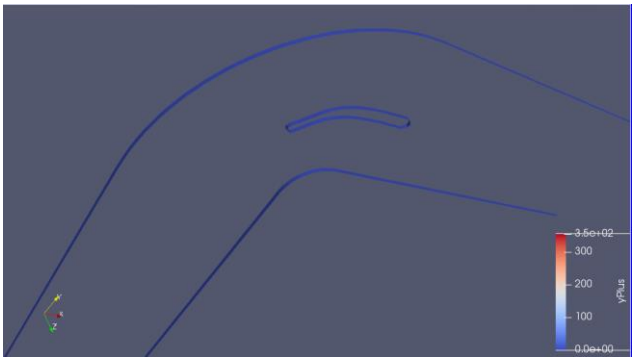


Figure 14. y+ distribution at the initial stage of 2nd case

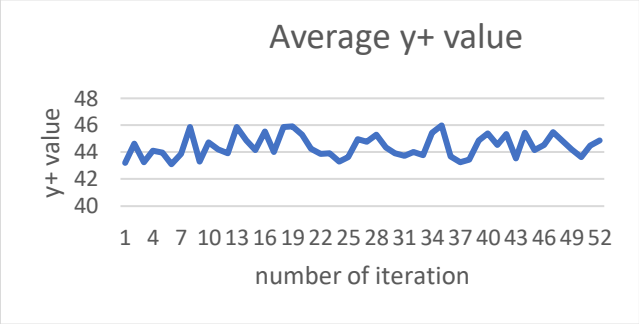


Figure 15. Average y+ value for 2nd case

The results for velocity, pressure and temperature are considered converged when the residuals are not higher than 10^{-1} , while the convergence criteria for turbulence related residuals (k and omega) were set to 10^{-5}. Additionally, the maximum number of iterations was fixed at 1000 (case 1) and 500 (case 2), respectively. This means that if the simulation is not converged till this point, It will be automatically finished.

If the simulation cannot meet the convergence criteria, a final value of pressure loss and either temperature of velocity profile will be calculated based on the mean of last 100 iterations.
In terms of mesh independency, it is important to mention that depending on the geometry, fluctuations may increase, but because of the large number of simulations the effect of those variation cannot be examined in detail.

2.4 BOUNDARY CONDITIONS

2.4.1 Boundary conditions – the first case

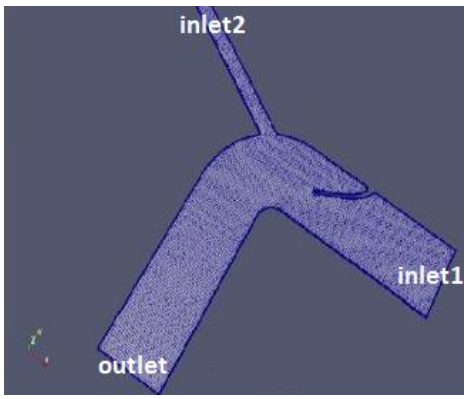


Figure 16. Boundaries of the first case

Inlet1:

- Initial temperature: 290K Dirichlet condition
- Initial Velocity: Fixed volumetric flow rate 0.0001 m^3/s achieved with openfoam's boundary condition flowRateInletVelocity (fixed volumetric flow in the direction of the vector normal to given surface)

- k and omega: Starting guess based on the $k-\omega$ equations 16-19
  $k = 0.0026$
  $\omega = 0.672$
- Initial pressure: 0, Neumann condition $zeroGradient$

Inlet2:

- Initial temperature: 310K
- Initial Velocity: Fixed volumetric flow rate 0.00005 m^3/s achieved with openfoam boundary condition $flowRateInletVelocity$ (fixed volumetric flow in the direction of the vector normal to given surface)
- k and omega: Starting guess based on the $k-\omega$ equations 16-19
  $k = 0.0048$
  $\omega = 0.892$
- Initial pressure: 0, Neumann condition $zeroGradient$

Outlet:

- Fixed pressure: 0, Dirichlet condition
- All other fields calculated based on other inputs
- k and omega: The same values as on inlet1

Walls and baffle:

- Velocity fixed at 0: No-slip condition
- Turbulence: k-omega wall functions, internal field with starting guesses k = 0.003, ω = 0.7
- Pressure 0, Neuman condition $zeroGradient$
- Other fields calculated based on inputs

2.4.2 Boundary conditions – the second case

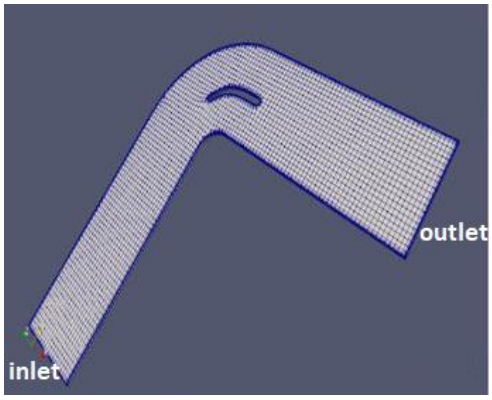


Figure 17. Boundaries of the second case

Inlet:

- Initial velocity: 3 m/s Dirichlet condition
- Initial pressure: 0 Neuman condition $zeroGradient$
- k and omega: Starting guess based on the $k-\omega$ equations 16-19
  $k = 0.3, \omega = 0.7$

Outlet:

- Velocity: 0, Neuman condition $zeroGradient$
- Pressure: 0, Dirichlet condition
- k and omega: Starting internal field k = 0.3, ω = 0.7

Walls and baffle:

- Velocity fixed at 0: No-slip condition
- Pressure: 0, Neuman condition $zeroGradient$
- k and omega: Starting internal field k = 0.3, ω = 0.7

In terms of k and ω values, OpenFoam is not that much sensitive software[2]. Even a roughly proper starting guess leads to the convergence and while the size of bend is the same in both cases, initial values for k and ω were just copied from the previous case.

k and omega initial guesses were calculated based on equations shown below:

Turbulence length scale:

$$l = 0.038\, d_h \tag{16.}$$

Turbulence intensity:

$$I = 0.16\, Re_{d_h}^{-1/8} \tag{17.}$$

Turbulence kinetic energy:

$$k = \frac{3}{2}\,(UI)^2 \tag{18.}$$

Specific dissipation rate:

$$\omega = \sqrt{k}/l \tag{19.}$$

Initial guess for kinematic eddy viscosity was set to 0.0001 just to avoid computational difficulties. All of the cells were also specified with an initial guess of 0.0001 for every boundary condition because of the same reason. This process enhanced the ability to obtain convergence.

2.5. POSTPROCESSING – INITIAL SOLUTION

2.5.1 Postprocessing – the first case

While the valve's presence enhances the fluid mixing, its presence increases the pressure necessary to pump water through it. According to figure 18, while the streams approach each other, mixing is performed. It can be seen that water changes its temperature quickly, and approaches the outlet at nearly uniform temperature.

Temperature distribution of the 1st case is similar to the velocity distribution, where it is visible that both of the streams lose the velocity at the encounter and the outlet distribution is nearly uniform.

In terms of the pressure, It might be noted that the largest pressure loss occurs around the area of initial mixing, where the pressure behind it is more uniform.

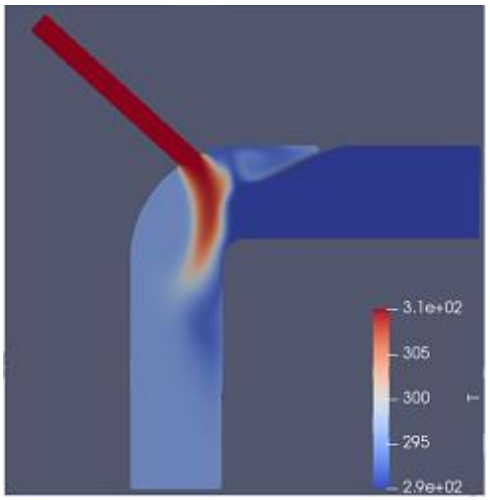


Figure 18. Temperature distribution of the initial 1st case

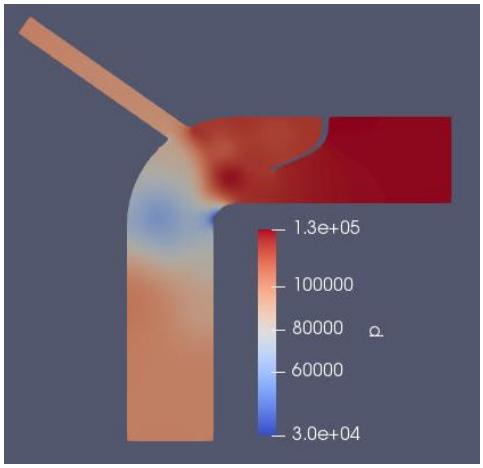


Figure 19. Pressure distribution of the first case

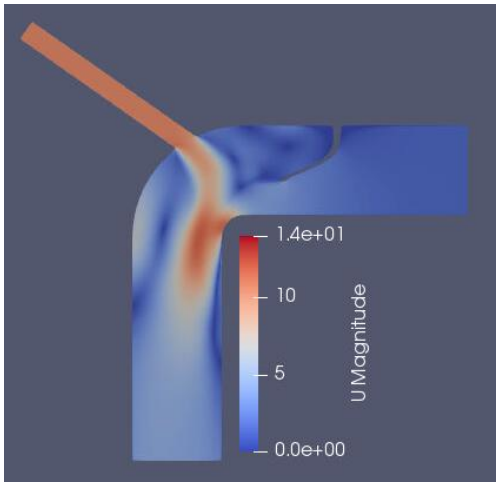


Figure 20. Velocity distribution for the 1st Case

2.5.2 Postprocessing – the second case

In the second case, the baffle influences the flow, allowing to achieve more uniform velocity distribution at the cost of higher pressure loss.

It can be seen that presence of baffle separates two areas of high and low pressure of the bend, while the biggest losses are visible at the bottom of the bend and at the beginning of the baffle.

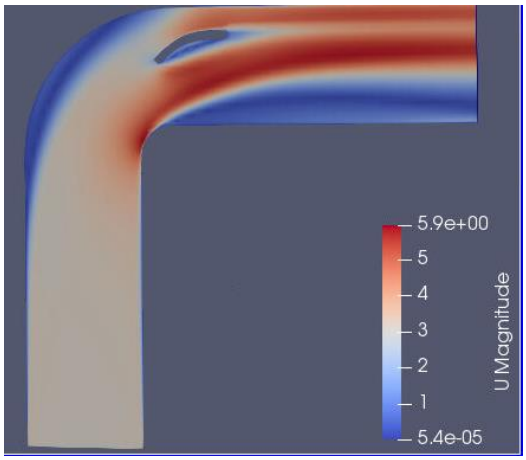


Figure 21. Velocity distribution of the initial 2nd case

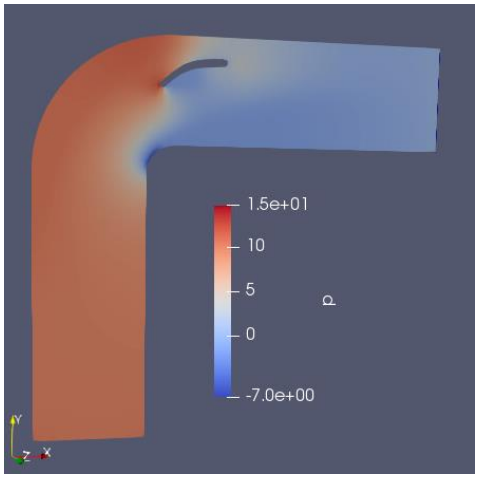


Figure 22. Pressure distribution of the initial 2nd case

3. GENETIC ALGORITHM STUDY

3.1 BASICS OF GENETIC ALGORITHMS

The genetic algorithm is an optimization strategy which tries to mimic the process of an evolution in a micro scale. Unlike a gradient-based optimization, the genetic algorithm approach does not require to compute derivatives or to have any deeper insight into a function designing the performance of the process. Because of that, they may be connected with the CFD software to look over a pool of geometries and find the best shape[3]

In the most simple approach, GA starts from creation of a random pool of designs, described by a given number of variables. Every specific design is called a “chromosome” and contains a fixed number of genes. Every gene represents a specific variable, which describes a physical parameter of the investigated design[3].

In the nature, chromosomes may mix in the process of crossover. Two chromosomes disassemble into four parts, which mix later in a random way. In this process, two new chromosomes are created. The point where chromosomes disassemble is called “point of crossover” and in the most basic approach, there is just one of it.
It is important to realize that new designs are built only from available genes, which means that no new genes can be created though crossover (New “blocks” cannot be inserted into a gene pool) [3].

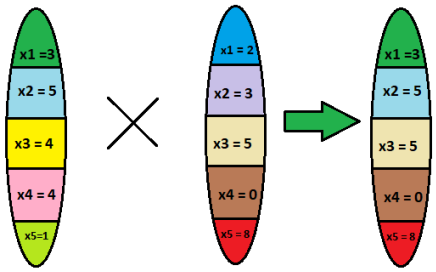


Figure 23. A simple crossover

To change the value of specific gene (expand the gene pool), it is necessary to perform a mutation. In its most basic version, a mutation is a purely random process which changes one gene value by a random amount. An application of mutation (the mutation chance higher than a zero) has both positive and negative consequences: [3]

The positives are:

- It may allow to find a fit design quickly
- Expands the gene pool

The negatives are:

- It might destroy fit designs
- It might slow down the convergence or destroy the convergence completely
- Its performance is hard to predict

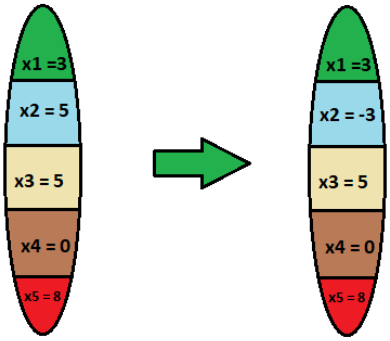


Figure 24. A simple mutation

Before the round of crossover and mutation, the selection is performed. All the genes are tested and their fitness function value is calculated. Higher value of fitness function increases the probability that a specific design will be selected into a “crossover-mutation” step. This kind of approach allows to assume, that the chance of getting a “more fit” offspring pool through genetic operations is higher than 50%. This fact is mathematically proven[3].

After selection, crossover and mutation, the entire process iterates further. A higher number of iterations allows to improve the result of optimization as the algorithm should converge at some point, showing mostly fit designs[3].

3.2 ADVANCED FEATURES OF GENETIC ALGORITHM

To decrease the amount of randomness, basic genetic algorithms were improved and several new features were introduced. Applying them, an algorithm’s ability to converge and find optima might be enhanced.

3.2.1 Sampling type

The first factor in genetic algorithm optimization which has a crucial influence on the algorithm performance is the type of an initial pool sampling[4]. Depending on a sampling type, the algorithm may be fed with either already fit designs or many kinds of point distribution, which allows to either look into very tight or very wide spectra of solutions. The first approach is s good decision if there are any guesses or previous surveys available, the second – when there is no access to previously gathered data of the specific case[3].

In this project, the second approach was used. Two sampling types were introduced. First, the uniform sampling, which divides the interval into given number of equally spaced subintervals. Second is the Latin Hypercube Sampling, which is considered well – performing type of sampling for many kinds of genetic algorithm application[5].

Latin Hypercube Sampling works similar way to the uniform distribution but it is more flexible. First, for every variable the research interval is divided into *n* subintervals. From every of these subintervals, a given number of points is drawn. After the specific point has been drawn, it is removed from the search pool, which means that it cannot be re-drawn. Depending on the dimensionality of the case, this process is repeated for every gene at the chromosome. The thus found values are connected into genes.

Latin Hypercube Sampling provides more available values for the starting pool.

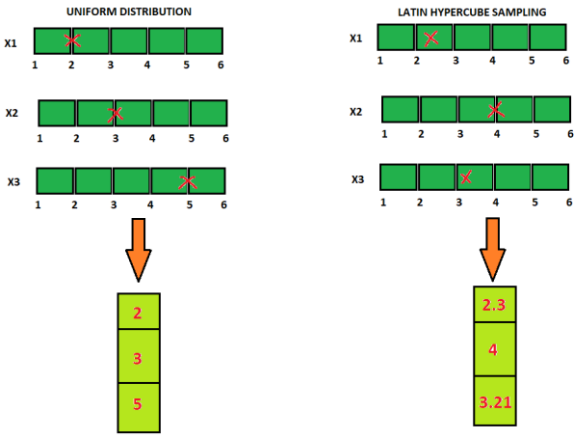


Figure 25. Comparison of sampling types

3.2.2 Crossover

The second feature added is aligned to the crossover type. Instead of performing a one-point crossover (which means building a chromosome out of 2 parts excluded from “parents-chromosomes”), more than two parents may be included into breeding an offspring chromosome. This approach may result in a larger variety of an offspring generation, however, this can reduce the ability to convergence.

To steer the process of crossover, different kind of variators were invented. Their role is to decide which gene out of the given “crossover pool” will be finally placed into newly created gene. Depending on an optimization type, the most common approach is to either create an offspring very similar to parents (according to the fact that the most fit designs are chosen) or rather different than parents (this kind of approach may expand the search). There is also a midpoint solution, where a “center of the mass” for pool of chromosomes is being calculated and breed offspring is in the “middle”. Nowadays, many types of variators are present in modern genetic algorithms.

In this project, three types of variators are examined:

- UNDX is the abbreviation for an Unimodal Normal Distribution Crossover. This type of multiparent variator chooses at least three parents to create a given number of offspring (one by default). According to the normal distribution, the created children will be described most likely by values from the center of distribution (around the so called “center of mass”), which is calculated separately for every single variable. In theory, the biggest advantage of this variator is a good performance on low initial population size. Its ability to converge is rather weak, which means that UNDX should construct more generations than other methods to get converged.[6]

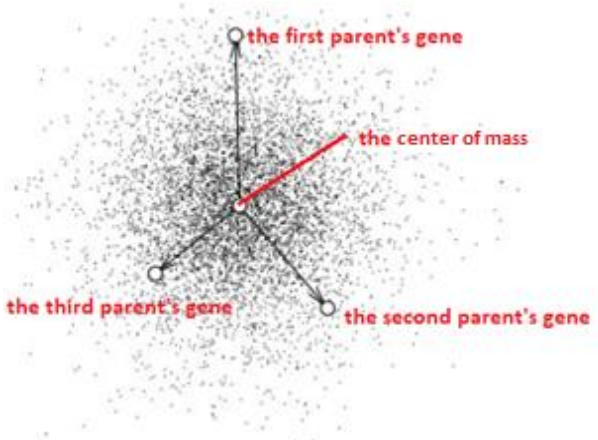


Figure 26. The UNDX Variator[6]

- The Simplex Crossover (SPX) makes use of a uniform distribution, where all the designs have similar chance of being created. The only limitation is a previously predefined area from where values for every specific variable may be drawn. This area is defined based on the parents pool.

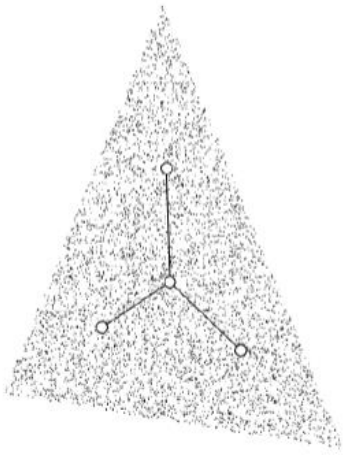


Figure 27. The SPX Variator[6]

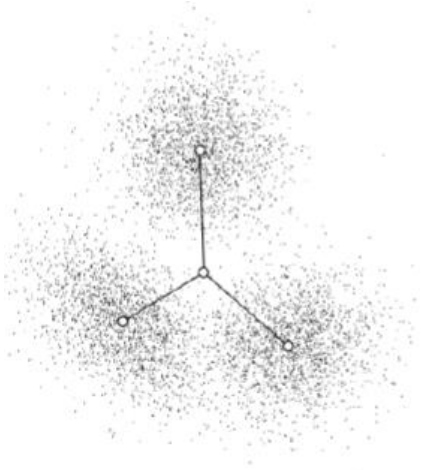


Figure 28. The PCX Variator[6]

- PCX (Parent Centric Crossover) assigns a higher probability for an offspring to remain closer to the parents than away from parents. It might be assumed that parents which were chosen in a “fitness competition” are actually high-quality ones and their features should be strengthen in the population. In terms of that it is a reasonable approach to construct offspring similar to the parents. This approach works well when looking for local optima[6].

3.2.3 Selector

The selector role is to choose which of the parents will be chosen for the crossover-mutation pool. The most important factors of a specific selector are the ability to improve convergence behavior and increase “preservation” of valuable genes in a gene pool. Sometimes a very fit gene is locked in an unfit chromosome. The genetic algorithm checks the value of a specific design based only on the fitness function, which means that it might remove a valuable gene during the removal of an unfit design. “Preservation” means that some unfit designs are keep in the population just to prevent this kind of scenario.
If the computational cost of a single design is low, it is reasonable to keep some value of unfit designs to achieve a better global optimum. Otherwise, an selector tuned for a faster convergence might be a better idea. In this project, two types of selectors are examined:

- Linear selector

Linear selector calculates the mean of fitness value for every single created chromosome. From this pool, the chance of choosing specific designs is calculated as ratio of its fitness to average fitness. This means that all the designs from a pool have a chance to reproduce, however, some of them will be extremely unlikely to do that. [3]

- Tournament selector

As the name suggest, tournament selector organizes a tournament between chromosomes, choosing only the best design for the next generation. This means that only one chromosome from a gene pool will survive, which means focusing on the fitness at the cost of ability to preserve genes. The gene pool is usually reduced over iterations, its convergence ability is improved.
Increasing the tournament size, the ability to converge should grow[3].

3.2.4 Mutation

In terms of mutation, there are many different strategies, and the most recognizable factor is its probability and strength. To limit mutation's random character, it might be set either to 100, 0 or a very little percent. The first "very-likely-mutation" approach assumes a common mutation whose value is limited by a normalized factor. The sampling of the mutation value might be done according to different kind of distribution, where normal distribution is in common use[3]. The default value of change is described around 10% for every variable, where the value of 5% has the largest chance of being drawn.

Zero chance for mutation means that the algorithm works steadily until it reaches an optimum on the given gene pool while it cannot expand it. This kind of approach is used mostly in discrete types of optimization, where a certain variable can represent only finite amount of numbers[3].

The last approach – low mutation rate – is a trade-off between the ability to improve global optimum and high convergence ability.

3.3 PLATYPUS BASED OPTIMIZATION

Platypus is a library of programming language python, which allows to use a wide variety of evolutionary algorithms. From this package, a NSGAIII optimization algorithm was used. This kind of a solution works best for two and more objective functions where at least four variables are used. NSGAIII shows good performance in many kinds of engineering optimizations, working with or without constraint and its enhanced ability to converge is mathematically proven.[13] [14]

For this optimization, the used setup was:

- Initial population size of 10, 15, 20 (First case) and 15, 22 and 30 (Second case). In terms of population size, there is an rule of thumb which assumes that initial population size depends on number of variables. It should be around 3 to 10 times bigger than number of variables used, which was 4 and 5 for specifically case 1 and case 2 – this is a rule of thumb. [8]
  Especially for CFD, It might be good idea to choose a value from a lower part of this interval to decrease the number of iterations.
- Sampling type – Uniform sampling
- Crossover type – Multipoint, based on chosen variator
- Variator type - UNDX, SBX, PCX
- Mutation chance – 1% (Default value), Mutation value default
- Selector type – Tournament selector with Tournament size of 10. The maximum value for tournament is the size of the initial population and the minimal value is two. This means that a value around the middle results in a fair trade-off between ability to preserve genes and achieve convergence.
- Convergence criteria – at least 10 succeeding designs whose fitness value difference is not bigger than one. Maximum number of iterations set for every population size. In the second case, the convergence criteria was changed due to higher fluctuations of fitness value to 5 iterations.

3.3.1 The algorithm built in platypus

The environment built to perform this genetic algorithm study is presented at Figure 24. First, the genetic algorithm creates a set of variables which is exported to a file. This file is read by a mesh generator and a specific mesh is created.

In the next step, CFD analysis is performed. As soon as it ends, the results acquired by postprocessing are gathered and directed back into the genetic algorithm. The fitness function is calculated and next iteration of genetic algorithm starts.

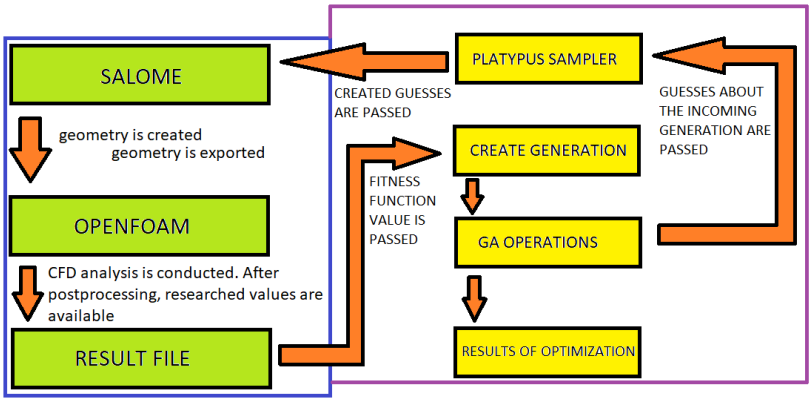


Figure 29. Platypus optimization scheme

Optimized variables are:

a) In the first case:

- X – the length of a baffle in the X plane
- Y – the length of a baffle in the Y plane
- P – the location of a baffle – while the minimum is directly at the outlet, and maximum – at the joint of pipes
- C – the angle between the pipes connection – from -20 to 85 degrees

b) In the second case:

- X1 – the location of a baffle inside the pipe (y coordination)
- X2 – the first point creating the baffle's curvature (x coordination)
- X3 – the second point creating the baffle's curvature (x coordination)
- X4 – the third point creating the baffle's curvature (x coordination)
- X5 – the fourth point creating the baffle's curvature (x coordination)

3.3.2 Results of the optimization

3.2.2.1 Results of optimization – First case

The first round of optimization allows to compare the performance of all chosen kind of variators while the rest of the genetic algorithm factors does not change (including population size, sampling type, crossover, mutation etc.).

The fitness function for the first case is expressed as:

$$Fitness = (difference\ between\ maximum\ and\ minimum\ temperature\ at\ the\ outlet\ [K]) * 10 + (Pressure\ loss\ between\ an\ inlet\ and\ an\ outlet) * 0.00001$$

This fitness function has to be minimized.

The weights were set to promote temperature distribution at the outlet over the pressure loss. While the amount of mixing might improve the temperature difference by roughly 2-3 Kelvins, the pressure loss can change significantly – from around $1.7*10^5$ Pa to $1.3*10^5$ Pa. Because of that, a weight of 0.00001 was added before the pressure loss term and the weight of 10 was added to the temperature term.

While small increases in pressure loss will not affect fitness function value by a large amount, this approach will protect the algorithm from converging at designs where pressure loss is extremely high.

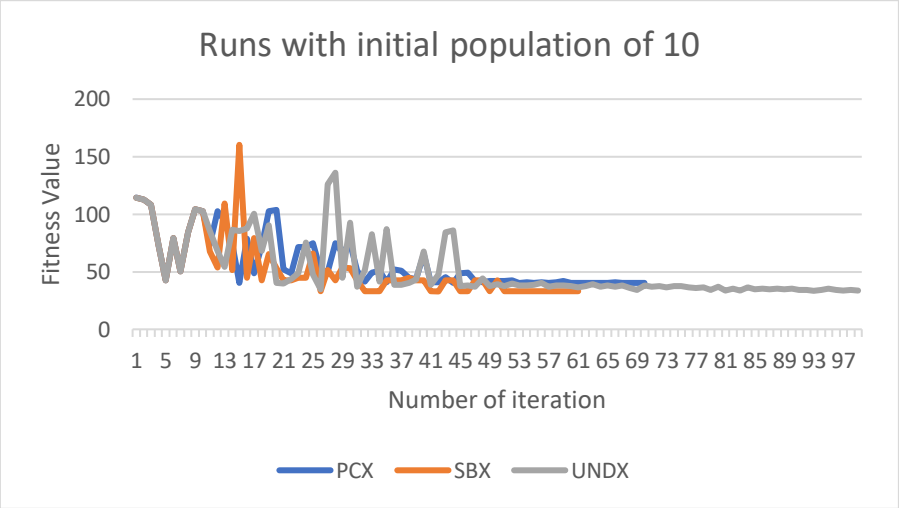


Figure 30. Comparison of results obtained with initial population of 10

All of these optimizations start from exactly the same initial population. All of the variators got converged. It might be seen that different kind of variators have large impact on GA's performance. First, there is a small difference in terms of found optima.

- The UNDX found the optimum at 33.6 and required 97 iterations to converge. According to the figure, it is significantly more than the rest of variators. Additionally, it might be seen that the fluctuation of fitness value are the largest while using this variator.
- The PCX performed a lot worse than the rest of variators, finding an optimum of 40.2 at 72 iterations. The optimum value is around 20% higher (which means worse) compared to the rest of variators. It might be seen that this variator is the most "conservative", which means the smallest fluctuations over the pool of chosen variators.
- The SBX needed only 61 iterations to find an optimum, which is 32.8. This is the best result in terms of both optimization time and quality of optima.

While the results found by SBX and UNDX are similar, PCX performed unexpectedly bad – probably because of the small initial population pool. To examine the effect of initial population on the optimization performance, the optimization was repeated with larger initial population pool – 15 and 20, respectively.

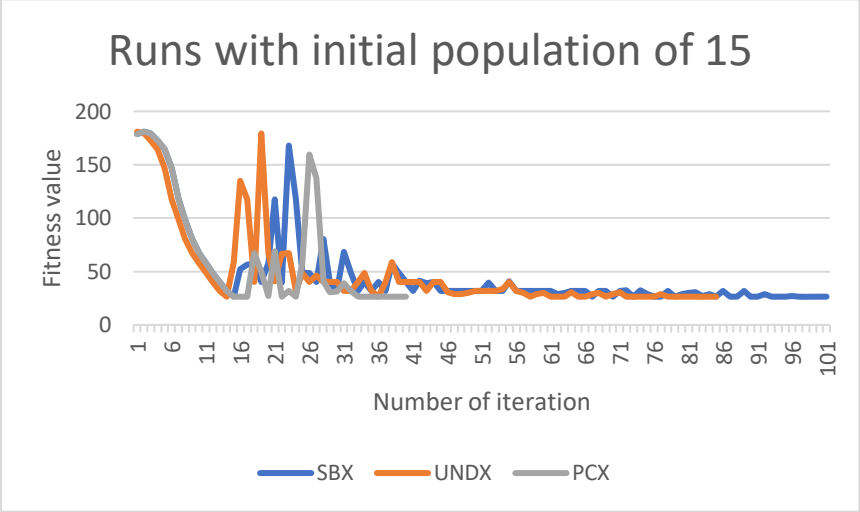


Figure 31. Comparison of results obtained with initial population of 15

The results of optimization with the starting pool of 15 are shown at Figure 31.

- Over 101 iterations, the SBX found an optimum at 26.3.
- The UNDX found an solution at 26.3 which required 86 iterations.
- The PCX found an optimum at 26.3 over just 40 iterations.

It might be seen that expanding the starting pool increased the effectiveness of the simulation. All the variators found the same optimum of 26.3, while the number of iterations required to do that is not the same in every case. At the graph, the most recognizable is PCX performance, which required only 40 simulations to find an optimum. It might be seen that this value on the graph is preceded by a sudden peak in the fitness value. The unexpectedly quick minimum might be just the result of a lucky, very fit mutation or crossover. In this population size, the worst performance is provided by UNDX, which requires as much as 101 iterations to find an optimum value.

The results of optimization with the starting pool of 20 are shown at Figure 32.

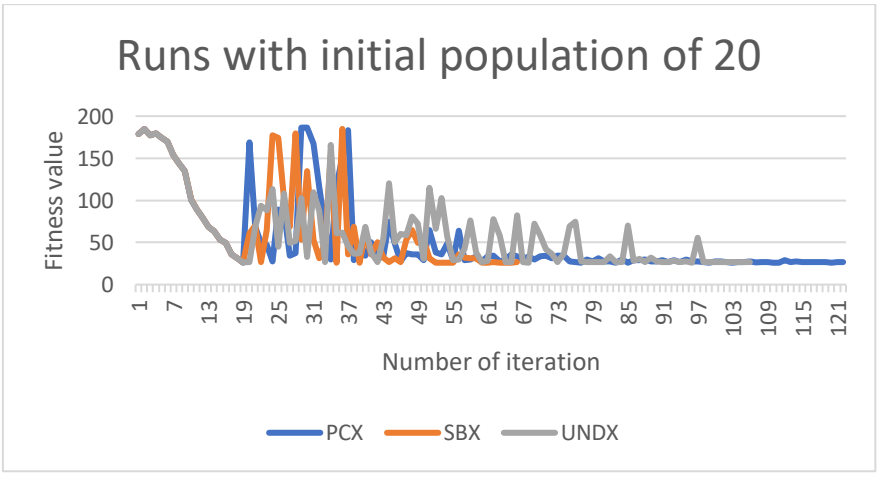


Figure 32. Comparison of results obtained with initial population of 20

- The UNDX requires 106 iterations to find an optimum at 26.6.
- The PCX finds optimum at 26.4 in 121 iterations
- The SBX finds optimum at 26.3 during 66 iterations.

It might be seen that a further expansion of the initial population size worsens the quality of results. Not only are the resulting optima worse than in the previous case, but also the number of iterations increased. Only the SBX performance is improved – it found the same optimum in lesser number of iterations compared to the previous run.

The comparison of results is listed in Table 1.

	Initial population of 10		Initial population of 15		Initial population of 20	
	Optimum	Number of iterations	Optimum	Number of iterations	Optimum	Number of iterations
PCX	40.2	72	26.3	40	26.4	121
SBX	32.8	61	26.3	101	26.3	66
UNDX	33.6	97	26.3	86	26.6	106

Table 1. Comparison of results of 1st case optimization achieved by Platypus

From the table, it is visible that SBX is the best choice for a variator in this specific case. Regardless on the initial population size, it always find the best value for the solution. On the other hand it might be seen that regardless of variator type, found optima are usually similar – only the number of iteration required to achieve it varies.

On the other hand, UNDX performs worse than the rest of variators, mostly because of its fluctuating nature which does not allow to achieve quick convergence.

3.2.2.2 Results of optimization – Second case

In the second case, the Fitness function is expressed as:

$$Fitness = (Pressure\ loss\ between\ the\ middle\ of\ an\ inlet\ and\ an\ outlet\ [Pa]) + (The\ difference\ between\ the\ largest\ and\ the\ smallest\ velocity\ module\ at\ the\ outlet\ [\frac{m}{s}]) * 100$$

This fitness function has to be minimized.

Again, the role of fitness function is to promote mixing over pressure loss. It might be seen that in the most of the iterations, pressure loss fluctuates usually between 50 and 100 Pa. While the velocity difference is 5.5 m/s for an unoptimized shape, the baffle optimization might reduce this value by roughly 1-2 m/s. This means that the range of pressure loss values is around 100 times wider than the range of velocity difference, thus a weight of a 100 was added to the pressure term.

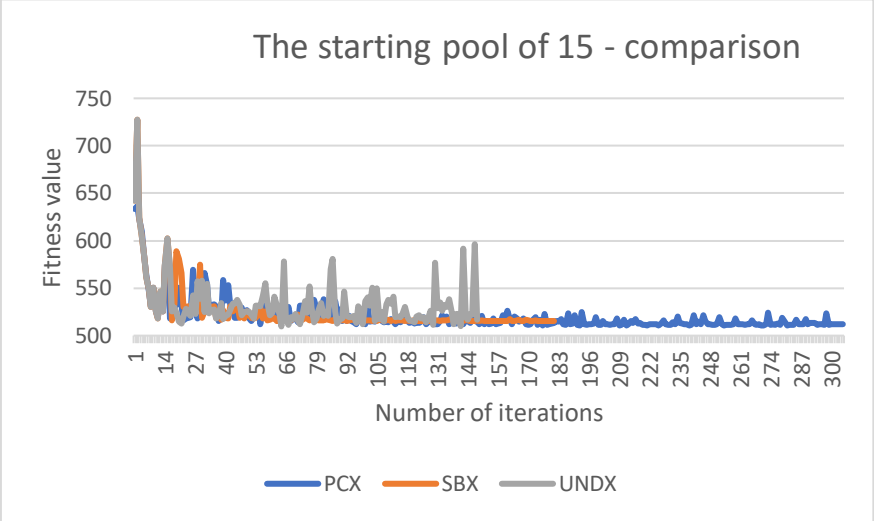


Figure 33. Comparison of results obtained with initial population of 15

The results of optimization with the starting pool of 15 is shown at Figure 33.

- PCX converges with the value of 516.1 over 309 iterations
- SBX converges with the value of 516 over 183 iterations
- UNDX converges at the value of 511.9 after 148 iterations

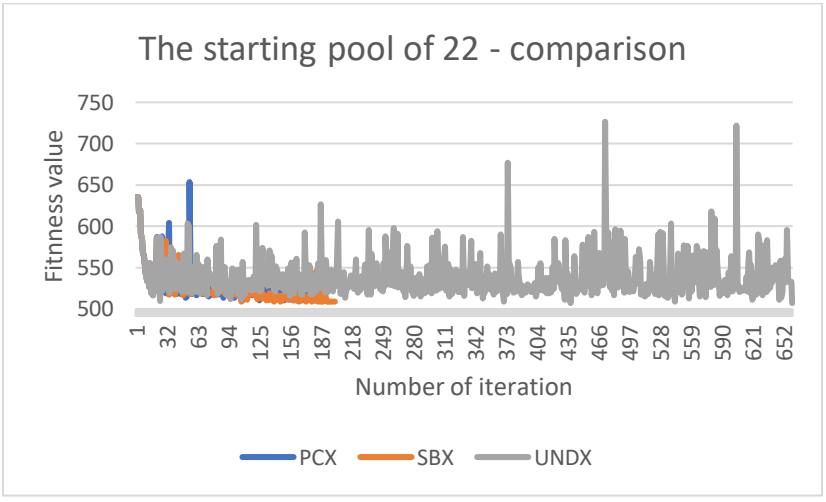


Figure 34. Comparison of results obtained with initial population of 22

The results of optimization with the starting pool of 22 is shown at Figure 34.

- PCX converges with the value of 511 over 190 iterations
- SBX converges with the value of 516.1 over 198 iterations
- UNDX finds the value of 510.9, while it cannot obtain convergence over more than 650 iterations

In terms of convergence, it might be seen that the results vary significantly over the variator type. However, this might be caused by very strict convergence criteria which assumes convergence as a change of parameter less than one. If these requirement was relaxed to less than five, the result might be considered converged at 75th iteration in terms of SBX and 195th iteration in the PCX. UNDX cannot achieve the convergence, however there might be found some fit results over the fluctuations. The difference between the found optima is less than 1%.

The results of optimization with the starting pool of 30 is shown at Figure 35.

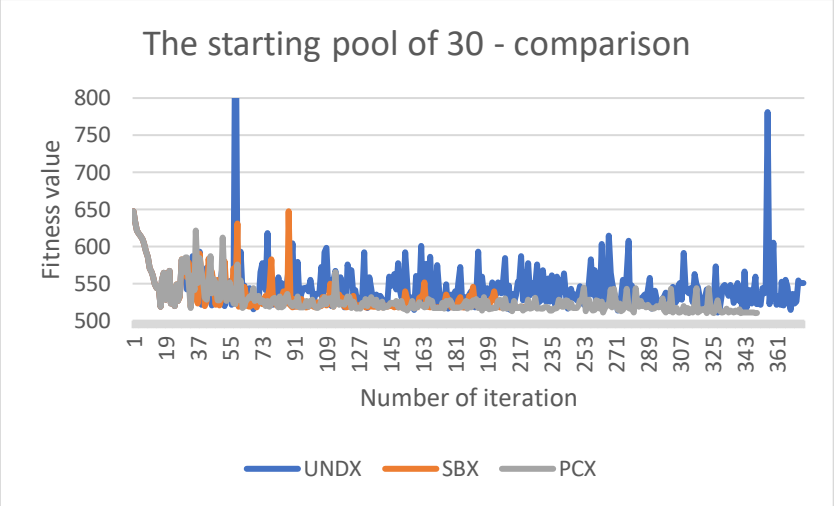


Figure 35. Comparison of results obtained with initial population of 30

- UNDX does not converge over 500 iterations, while the most fit value found is 510.9
- PCX converge after 339 iterations with fitness value 510.9
- SBX converges in 214 iterations with fitness value of 517.1. This value does not decrease over more than 30 iterations.

It might be seen that the initial pool of 30 does not result in better optima. However, it drastically increases the number of required iterations. The UNDX does not converge and fluctuates over its entire spectrum, however, it still finds the optimum value of 510.9, which is the best value found in this optimization.

	Initial population of 15		Initial population of 22		Initial population of 30	
	Optimum	Number of iterations	Optimum	Number of iterations	Optimum	Number of iterations
PCX	516.1	309	511	190	510.9	339
SBX	516	183	513.1	198	517.1	214
UNDX	511.9	148	510.9	unconverged	510.9	unconverged

Table 2. Comparison of results of 2nd case optimization achieved by Platypus

It seems that an optimum value for initial size population in second case is 22. Not only this size achieves the best result in terms of optimization but also does it in the smallest amount of iterations. The difference is found optima is small – around 1%.

3.4 OPTISLANG OPTIMIZATION

3.4.1 The setup built in Optislang

OptiSLang is a software platform for CAE-based sensitivity analysis, multi-disciplinary optimization (MDO) and robustness evaluation. It is developed by Dynardo GmbH and provides a framework for numerical Robust Design Optimization (RDO) and stochastic analysis by identifying variables which contribute most to a predefined optimization goal. This includes also the evaluation of robustness, i.e. the sensitivity towards scatter of design variables or random fluctuations of parameters.[15]

To perform an optimization in Optislang, a sequence of systems was built. Each of these systems consists of the same blocks. Blocks contain commands written in either bash or python programming language, which built entire optimization environment. One of these systems “Kriging” was presented at Figure 36.

In every single block, there are 5 scripted sub-blocks:

- “Initialize” – loads all the modules required to the simulation, including specific python version and opensource CFD software, OpenFOAM
- “Create input file” – reads the values sampled by Optislang and converts them into a file, which is further exported to the location where CFD will be performed
- “Create Geometry” - runs the python-based CAD software “Salome” in the batch mode using a previously generated script. Application of specific python library “pickle” allows to connect both the exported input file and geometry script to create an unique CAD model, described by specific inputs.
- “Perform CFD” – the geometry is loaded to an automatic meshing tool (CfMesh). After the mesh is created, OpenFOAM performs a CFD analysis, calculating the pressure loss between specific point (an inlet and an outlet of pipe) and either temperature or velocity distribution at the outlet.
- “Gather results” – gathers the values of both distributions and pressure loss and calculates the mean of them. They are used to calculate objective function value and passed to the Optislang.

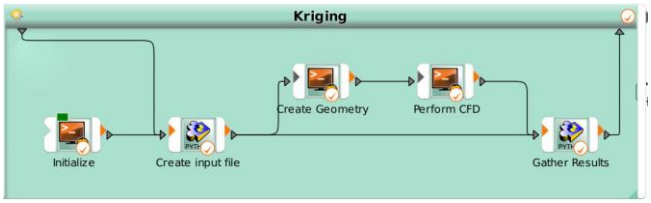


Figure 36. Example of a single system built in Optislang

From these kinds of blocks, three types of systems were built:

- Sensitivity systems – create input variables with specific type of sampling and checks the relation between parameters, especially the linear correlation between sampled input points and between input and output values
- Evolutionary Algorithm systems – perform a genetic algorithm study. Must be either fed with a points created previously or do independent sampling.
- AMOP systems – try to perform several types of metamodeling and check the accuracy between them and models created by CFD. Creates a visual 3D prediction of the function surface based on both metamodels and points used to create metamodel pool. [7]

Having these three kind of systems, three models were built:

1) Sensitivity systems -> Evolutionary Algorithm systems
2) Sensitivity systems -> AMOP systems -> Evolutionary Algorithm systems
3) Sensitivity systems -> Evolutionary Algorithm systems -> AMOP systems -> Evolutionary Algorithm systems -> AMOP systems -> Evolutionary Algorithm systems

Models 2) and 3) will be described in Chapter 4 Metamodeling.

3.4.2 Sensitivity analysis

One of the Optislang feature is the sensitivity analysis, which allows to check the influence of given variables on the fitness function performance. Based on sampled initial points, the influence of specific variables was checked.

3.4.2.1 Sensitivity analysis – the first case

In terms of 1st case, it might be seen that the fitness function changes by a large amount especially based on two specific variables. They are:

- the angle of the pipes connection
- the length of a baffle

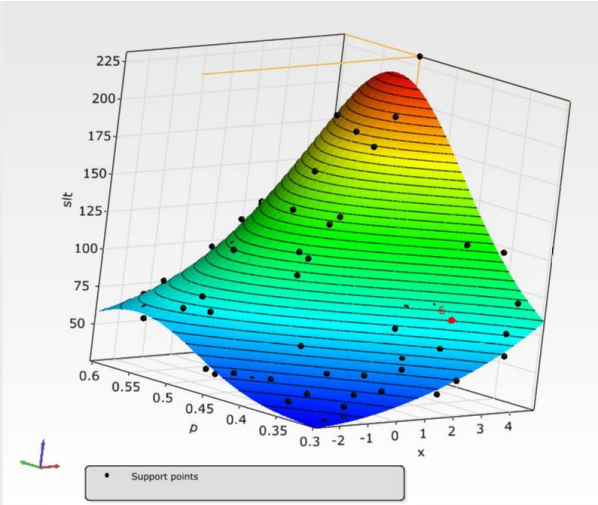


Figure 37. The shape of problem surface in the 1st case

According to Figure 38, to achieve the optimal shape of flow system, algorithm will try to maximize the angle (up to 90 degrees) and lengthen the baffle. Additionally, the value of “c” (which is the distance of baffle from the connection of pipes) will be rather low, which means that the baffle should be placed close to the pipes connection. However, both variables describing the baffle (c and y) are not as much influential as baffle length and angle of pipes connection.

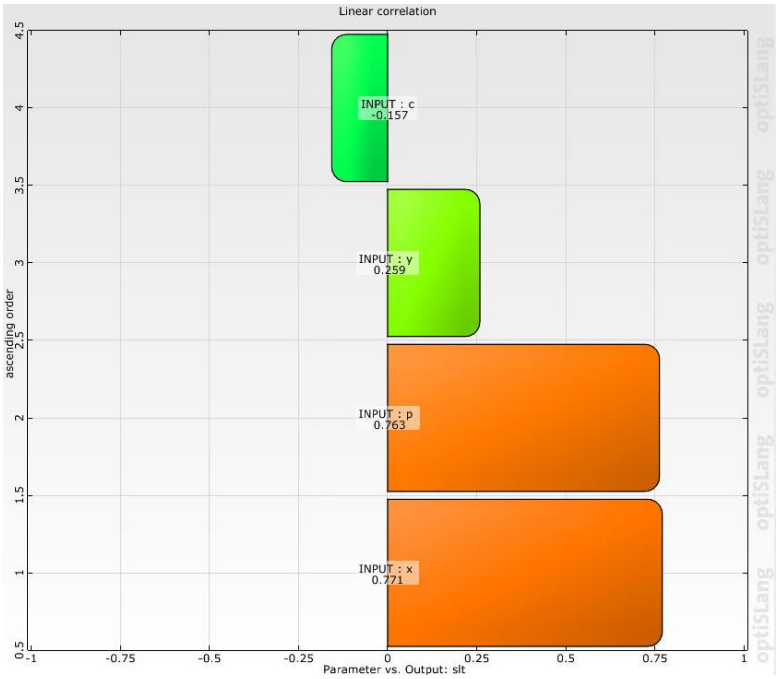


Figure 38. Linear correlation between input variables and the fitness function

3.4.2.1 Sensitivity analysis – the second case

It might be seen that the surface is a lot more flat compared to case one. This is right – objective function in terms of second case is a lot more stable (varies from around 520 to 650, compared to 20 and 200 in the first case). According to Figure 40 it might be seen that variables x2, x3, x4 and x5 have quite similar influence on the objective function. Indeed, they describe the x-coordination of point from which baffle is created. However, one variable shows a totally different trend than the rest – x1. This variable describes the vertical baffle coordination.

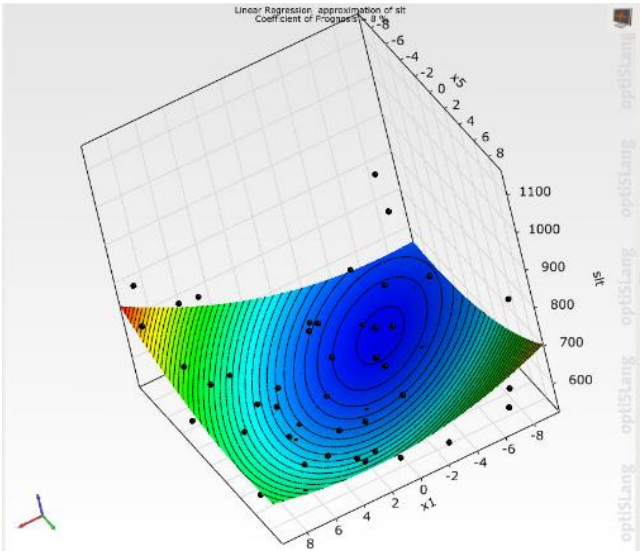


Figure 39. The shape of 2nd case function's surface

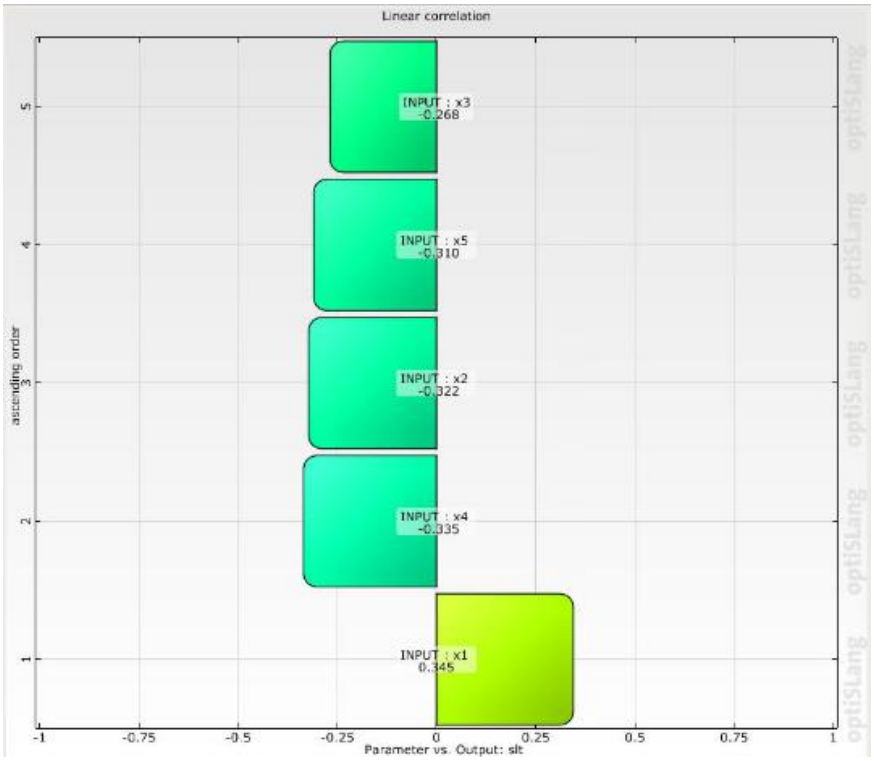


Figure 40. Linear correlation between input variables and the fitness function

3.4.3 Optislang optimization and results

Optimization parameters used in Optislang was changed since the Platypus optimization. First, the Optislang does not allow to choose type of variator while it leaves more space in terms of mutation and crossover manipulation. The optimal initial population size was preserved from Platypus and reused.

The optimization criteria was:

- Initial population size: 15 (for case 1.) and 22 (for case 2.)
- Sampling type – Latin Hypercube Sampling
- Crossover type – Multipoint with three crossover points (default value)
- Variator type - No variator
- Mutation chance – 98% chance for a mutation. The mutation mechanism adds or subjects a random value from interval $(0, 0.1x)$, where x is actual variable value. The draw follows the rules of normal distribution, which means that the value of $0.05x$ has the largest chance of being chosen.
- Selector type – Tournament selector with Tournament size of 10. The maximum value for tournament is the size of initial population and the minimal value is two. This means that a value around the middle results in a fair trade-off between ability to preserve genes and achieve convergence.
- Convergence criteria – Maximum number of iteration or an improvement of the fitness value in relation to the results obtained by the Python Platypus

The results of Optislang optimization was presented in Figure 41 and Figure 42.

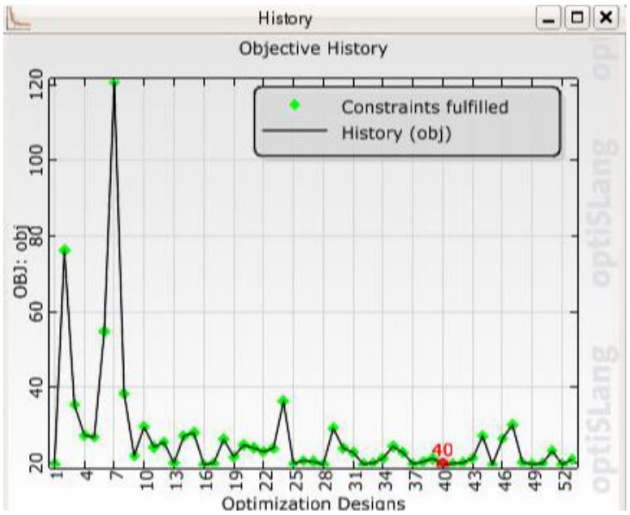


Figure 41. Genetic optimization of the 1st case done in Optislang

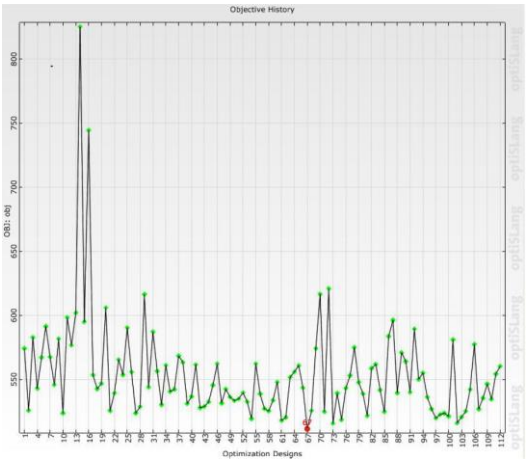


Figure 42. Genetic optimization of the 2nd case done in Optislang

It might be seen that the Optislang allows to perform a lot smoother optimization which finds the optimum value a lot quicker than the Platypus.

In terms of first case, fitness value was also improved (19.6 compared to 26.3 in python library), while in the second case the found optimum is worse by around 0,2% (511 compared to 510).

4. METAMODELLING

4.1 THE PURPOSE OF METAMODELING

Metamodels, called often surrogate models or low-fidelity models, are designs created from high-fidelity models (in this case, results of CFD analysis) which predict the value of fitness function for entire spectrum of variables. The topic of metamodeling is often related to the Machine Learning, where the computer tries to build a continuous function out of given discrete values. There are many kinds of metamodeling and for a specific problem, there is usually a type which outperforms the rest of them. [8]

The results of CFD analysis are generally hard to predict, however the topic of metamodeling optimization is getting a lot of attention in the industry due to the reason, that constructed metamodels can expand both initial population pool and generation size, which allow to spare time previously used to compute high-fidelity designs.

According to scientific research, depending on the case, properly set metamodeling environment can shorten the optimization time by a large amount (even several dozen of percent).[9]

According to the previous scientific work[8], the most promising type of metamodeling for CFD optimization is Kriging, Artifical Neural Network and Response Surface Approximation. In this project, Kriging was chosen as the metamodeling type because of its decent performance on low starting pool cases. [8]

4.2 THE KRIGING

4.2.1 The basics of Kriging

The kriging, called also Gaussian Regression, is a type of interpolation primary introduced to geostatistical sciences, where its purpose was to predict the location of gold vein from the fixed amount of non-uniformly scattered, low number of boreholes. [10]

The idea of kriging is to predict the value of an unknown function at a given point by the computation of weighted average of the know values of this function in enough close neighborhood. In terms of that, Kriging is similar to regression analysis.

As an interpolation technique, Kriging is a covariance-based type of prediction, where all the found values lies directly at the found function's neighborhood, where the distance depends on random number achieved from sampling of either normal or different kind of distribution.

The most basic type of kriging may be expressed with given formula:

$$Z(s_x) = \Sigma\lambda Z(s_i)$$ [9]

Where:

$Z(s_x)$ – value of fitness function in a surveyed location

$Z(s_i)$ – value of fitness function in a previously-known location

λ – weight parameter

Which means that the prediction of function value is just the sum of weighted values found in the neighborhood. However the method of weight coefficient computation may be often complicated. Removing the weighted sum value, kriging may be expressed with Formula:

$$y(x) = p(x) + Z(x)$$ [9]

Where:

$y(x)$ – value of fitness function in surveyed location

$p(x)$ – value of fitness function in a known location

$Z(x)$ – a random bias sampled with some kind of distribution or other sampling strategy

The $Z(x)$ is the realization of a Gaussian random process, with the mean of zero, variance of σ^2 and non-zero covariance. While the $p(x)$ reflect so-called "global" approximation, $Z(x)$ stands for a "local" deviation which expresses the effect of N point used to approximation. [9]

In a one-dimensional case, Kriging may be expressed with the approach similar to the one shown at Figure 43. At the figure, z stands for function value and x is an input variable.

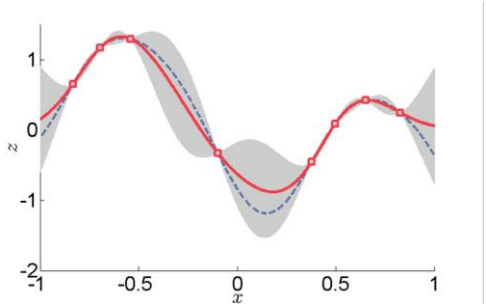


Figure 43. A simple 1D Kriging example [11]

The red point shows the values computed or measured (high-fidelity model). They are connected with some type of curve (depending on kriging type) and for every two point (which are neighbors) there is an area of uncertainty, which is expressed by normal distribution (only in the most basic approach; values closer to the curve have higher chance of being drawn). After the Z values are computed, the new "prediction" curve is computed and unknown values predicted. [12] In terms of that, Kriging may be an iterative process with the ability to improve itself.

In terms of this project, a four-dimensional and five-dimensional kriging is used.

4.2.2 Results obtained through Kriging and the accuracy of prediction

To improve genetic algorithm performance, Kriging was used as a generator of additional designs. This allows to expand gene pool and give the genetic algorithm a guess where it should look for an optimum.

Optislang generated kriging points based on one initial population of samples gathered with Advanced Hypercube Sampling. This means 15 points in terms of first case and 22 point in terms of the second case. Both of these were shown at the graphs.

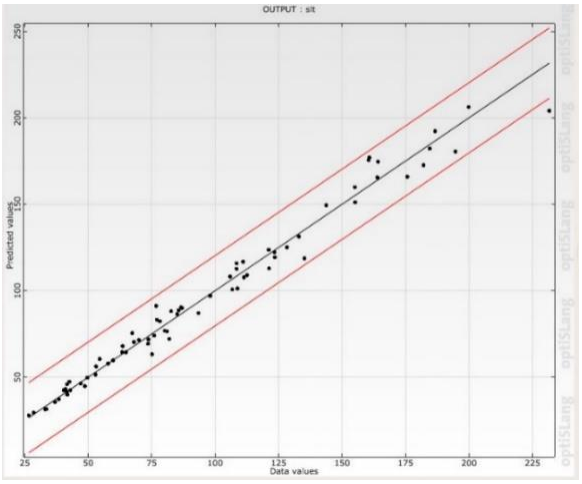


Figure 44. Kriging done in Optislang on 22 initial points (First case) and its accuracy

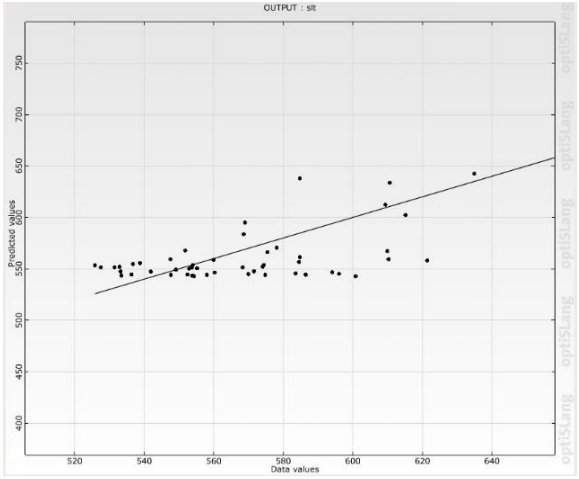


Figure 45. Kriging done in Optislang on 22 initial points (Second case) and its accuracy

It might be seen that the kriging can predict values with good accuracy basing on initial population of genetic algorithm. Figures 46 and 47 shows comparison of results between objective function values obtained by CFD and the metamodeling. In the first case, the relative error of kriging might be either very little (less than 1%) or quite large (around 16%). However, most of the predictions achieved shows rather good fit and the mean value of the relative error achieved is equal to 3,41%.

According to second case, Kriging performs even better than in the first one. While the approximation may produce some highly biased designs (maximum relative error higher than 23%), most of the values are very similar to the results obtained via CFD. The mean error of approximation is equal to 2,51%.

Based on that it might be good approach to use kriging as an intermediate step for genetic algorithm performance but certainly not in an independent analysis. Relative error fluctuates and there is no visible trend, so the quality of an individual metamodel is hard to predict without comparison to CFD analysis.

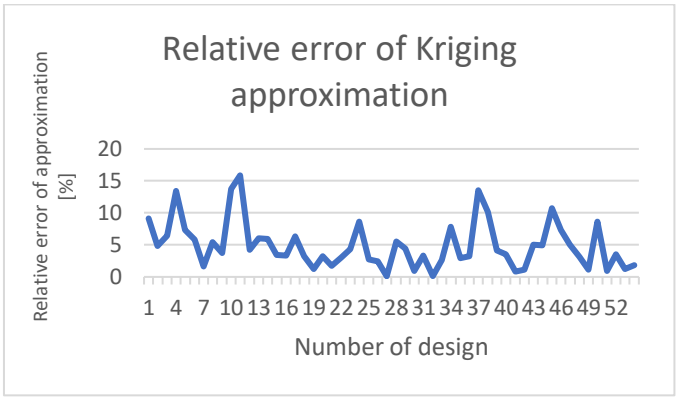


Figure 46. Relative error of Kriging approximation in created designs (First case)

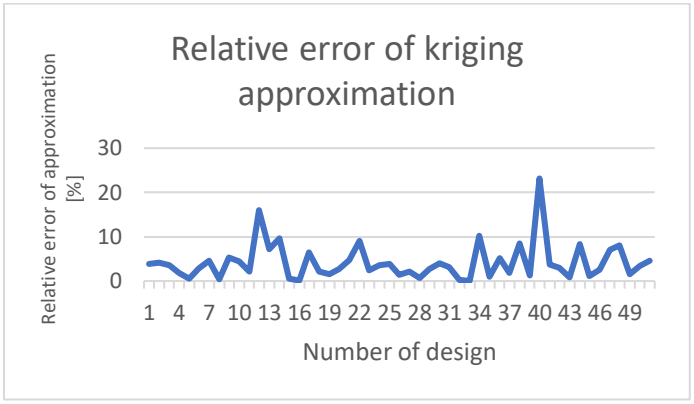


Figure 47. Relative error of Kriging approximation in created designs (Second case)

On the other hand, Kriging can also result in achieving very unfit solutions, which may disrupt the simulation performance. On the Figure 48 a result of Kriging done on the initial population size of 100 might be seen, where large amount of the predictions is just a nonsense. This figure was created with a different software called Dakota. While Kriging is a sophisticated metamodeling which is described by many inputs, it can be seen that the good accuracy could not be achieved with this specific software.

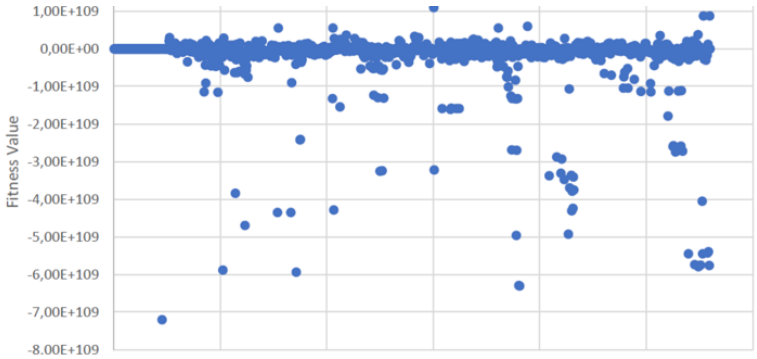


Figure 48. Kriging performed on large initial pool

This issue was partially resolved by lowering the initial design pool to 20. However, the relative error was still huge compared to results obtained by the optislang.

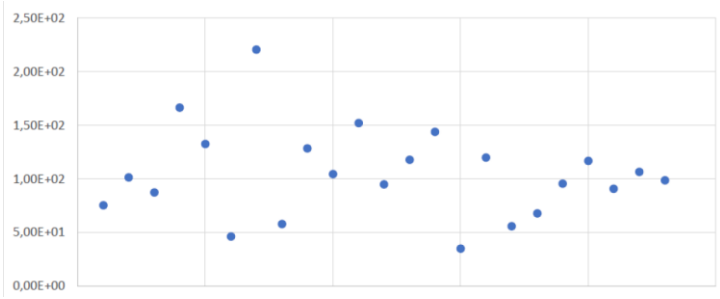


Figure 49. Kriging performed on small initial pool of 25 designs with Dakota

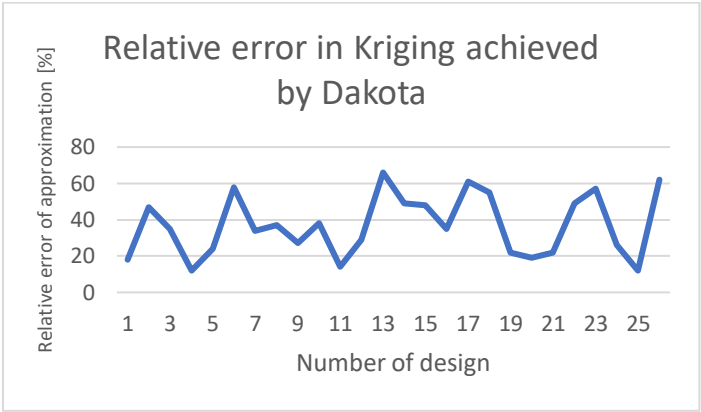


Figure 50. Relative error obtained by Kriging approximation in Dakota

It is clearly seen that low-fidelity models achieved by Optislang are a lot more accurate than those produced by Dakota. This is caused mostly by the black-box character of Optislang, where it can optimize its predictions based on previous models and change its own guesses the iterative way. Additionally, the software allows to choose the maximum amount of iterations where optislang is allowed to perform improvement of created models. Because of this, Dakota was discarded in further work.

In the next step, a second round of Kriging approximation was performed. The results of first Kriging was connected with initial pool of solutions created by CFD analysis. The next Kriging was performed based on values randomly chosen out of this pool – 15 in the first case and 22 in the second case. The results obtained are presented at Figures 51 and 52. It can be seen that the relative error of approximation is similar to the first round of Kriging.

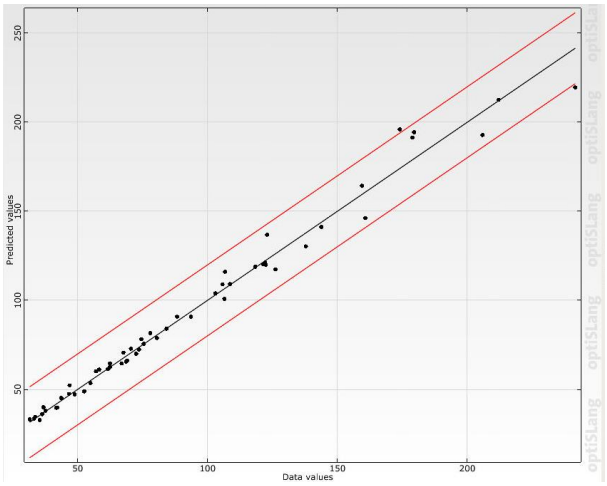


Figure 51. 2nd round of Kriging approximation in the first case

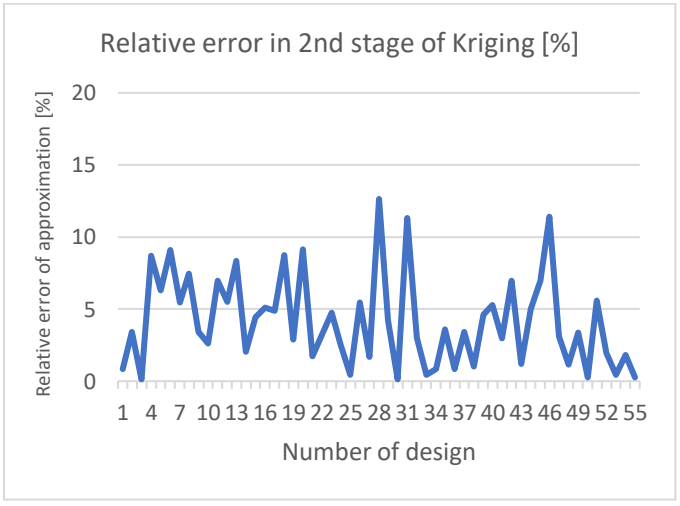


Figure 52. 2nd round of the Kriging approximation relative error (first case)

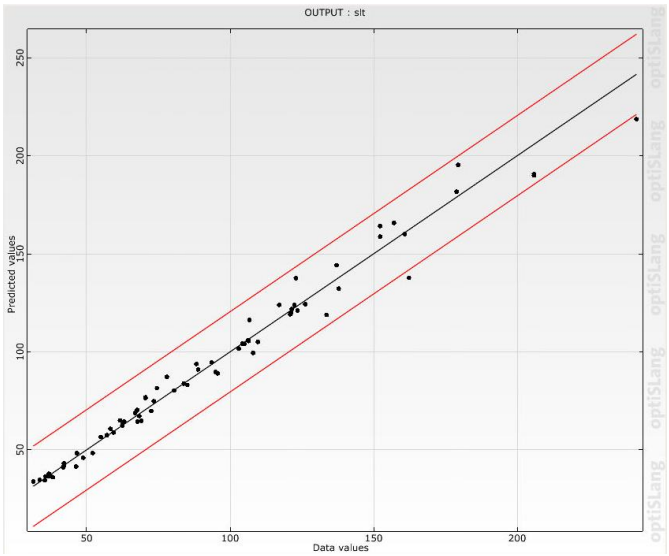


Figure 53. 2nd round of Kriging approximation in the second case

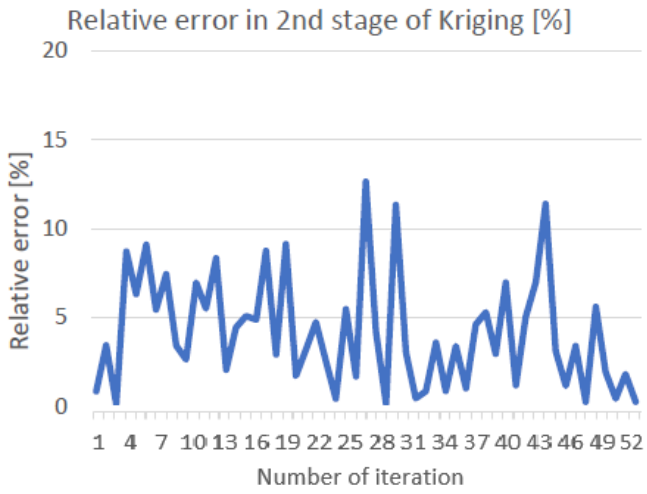


Figure 54. Relative error of the 2nd round of Kriging approximation (second case)

4.2.3 Implementation of Kriging in genetic algorithm

4.2.3.1 Implementation of Kriging in genetic algorithm – the first case

Created metamodels was used as an initial population for genetic algorithm in both case 1 and 2. Obtained results was shown at Figure 55 and Figure 56.

In the first step, results of Kriging was passed to the genetic algorithm, so the initial population size was expanded by generated metamodels. According to figure 55, It might be seen that obtained metamodels does not increase the effectiveness of genetic algorithm performance. The optimum was found after 41 iterations (compared to 40 iterations of “raw” genetic algorithm) while the found optimum is nearly the same as before (less than 0,1% difference). While the kriging passed some good guesses into the algorithm (point 1 is nearly as fit as the found optimum with the value

of 21.1), their lack of accuracy slowed down the convergence. The discontinuities in the graph line are caused by mesh breaks.

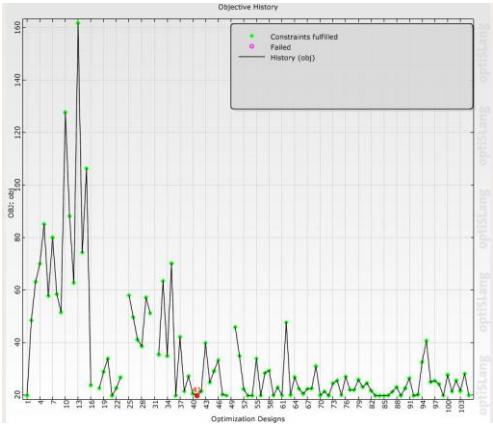


Figure 55. Optimization of 1st case with genetic algorithm and one round of Kriging

To strengthen the influence of Kriging in entire optimization, it was introduced twice. Kriging generated by AMOP systems were first created based on 30 solutions found by 2 generations made by genetic algorithm. Later, both the results of Kriging and a high-fidelity models were passed to another Evolutionary Algorithm system, where another 2 generations were performed.

The Figure 50 shows the effect of research carried with the last genetic algorithm round – again, locked at two generations. It might be seen that the graph presents more flat but consistent convergence, which finds out an optimum at 20.2 (compared to 19,6 its around 3% less fit minimum). However, this kind of approach is very time inefficient since it required to compute all the previous blocks.

Finally, the cost of entire process was evaluated for 105 iterations (three Evolutionary Algorithm systems with 30 iterations each and Sensitivity system with 15 iterations), which means more than two times longer optimization for a slightly weaker result.

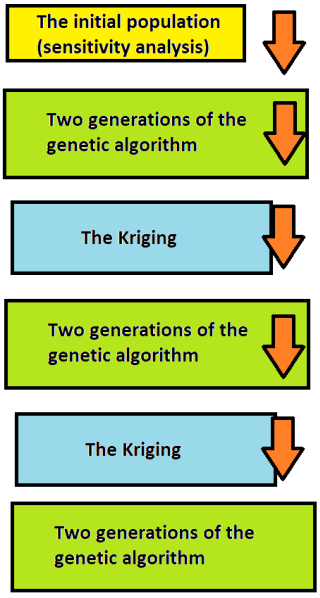


Figure 55a. The “two-kriging” method

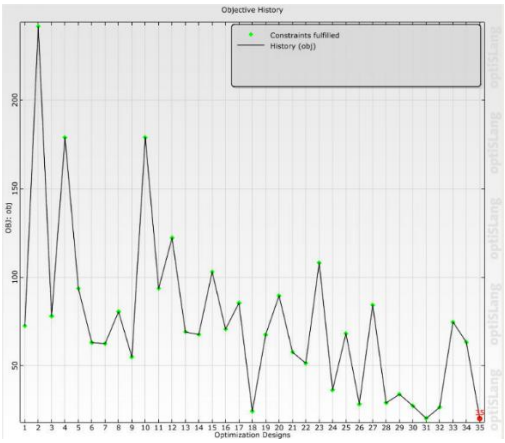


Figure 56. Last stage of 1st case optimization with application of genetic algorithm and two rounds of Kriging

4.2.3.2 Implementation of Kriging in genetic algorithm – the second case

The previous procedure was repeated for second case. The only difference made is an increase in maximum number of iterations permitted in the approach of “two rounds of kriging application”. The purpose of it is to check if higher number of iterations can obtain fitter solution through kriging.

The result of first optimization was shown at Figure 57.

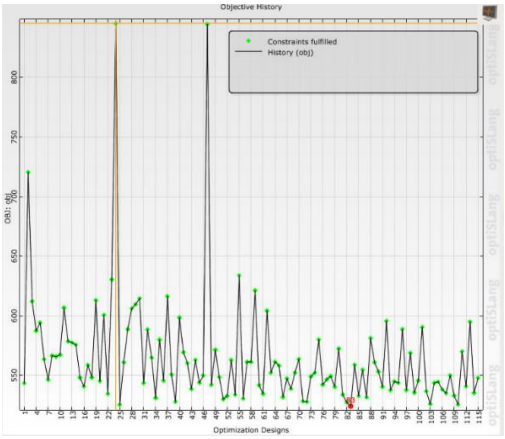


Figure 57. Optimization of 2nd case with genetic algorithm and one round of Kriging

According to Figure 57, The most fit value was found at 25th iteration – just in the second generation. The expansion of the initial population size made by Kriging was enough to find an optimum value in a very short time. Compared to optimization without kriging, the result was obtained 47% quicker (47 compared to 89).

After that, Kriging was introduced twice into the genetic algorithm. The result was shown at Figure 58.

It might be seen that the curve fluctuates and shows no visible trend. It might be explained with the selection, where genetic algorithm started its optimization from an inaccurate results. Additional input provided with kriging only

disrupted the convergence. Finally, algorithm finds an optimum after 307 iterations, which is the worst result compared to the rest of optimizations with the same starting population size.

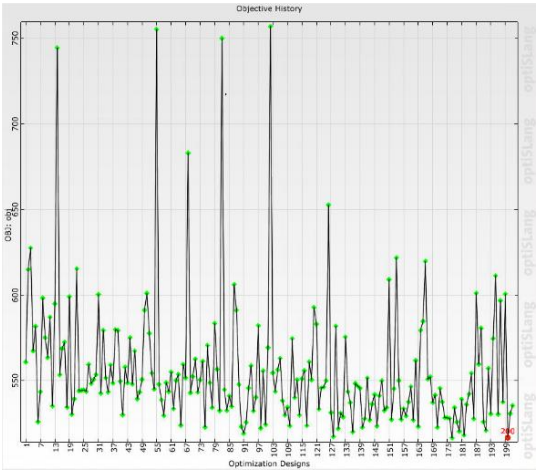


Figure 58. Last stage of 2nd case optimization with application of genetic algorithm and two rounds of Kriging

The comparison or Kriging results is shown in Table 3.

	Raw genetic algorithm		One round of kriging applied		Two rounds of kriging applied	
	Found Optimum	Number of iterations	Found Optimum	Number of iterations	Found Optimum	Number of iterations
Case 1	19.6	55	19.6	56	20.2	105
Case 2	511.3	89	511.4	47	517	307

Table 3. Comparison of optimization results achieved with and without Kriging

It might be seen that application of two kriging rounds resulted in worse performance in both cases. While in the first case application of kriging caused no positive or negative effect, in the second case kriging application resulted in large reduction of simulation time (47% compared to no-kriging case in Optislang, 76% compared to Platypus-based optimization).

5. ANALYSIS OF OPTIMIZATION RESULTS

5.1 Analysis of optimization results – the first case

It might be seen that the low velocity at the bigger inlet causes the baffle to have a really small effect in pressure increase compared to the increase coming from the second inlet flow. Additionally, The objective function is tuned to support mixing more than pressure loss, so the algorithm decides to maximize the baffle size and increase the value of velocity and which streams encounter each other.

While it increase pressure loss value by around 18%, more uniform distribution of temperature might be achieved. Sudden contraction result in fluid acceleration when both fluids approached themselves at maximum velocity nearly countercurrent. This causes a splash after which the velocity decreases followed by pipe expansion which strengthen this effect.

The temperature distribution at the outlet is compared at the Figure 63.

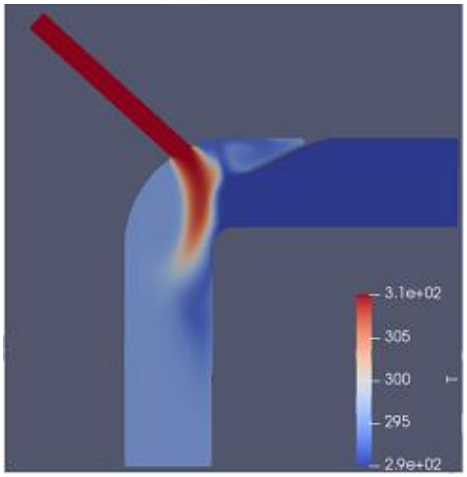


Figure 59. Temperature distribution in unoptimized Y-junction

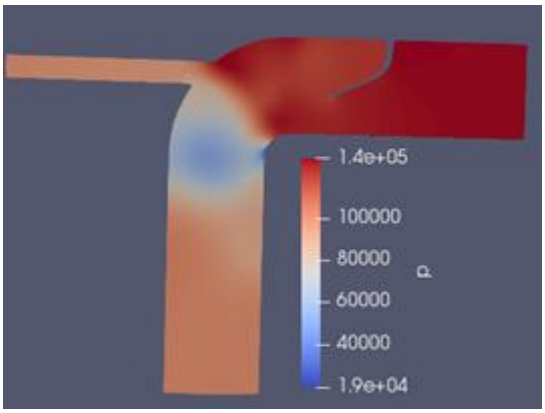


Figure 59a. Pressure distribution in optimized Y-junction

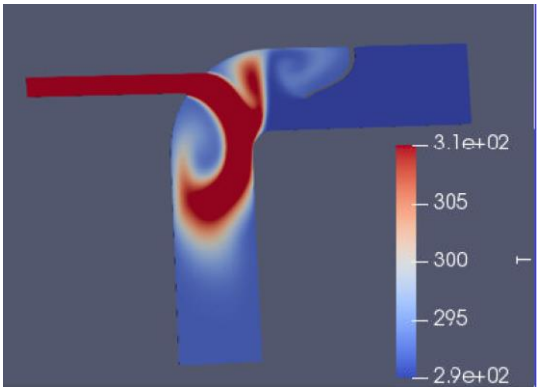


Figure 60. Temperature distribution in optimized Y-junction

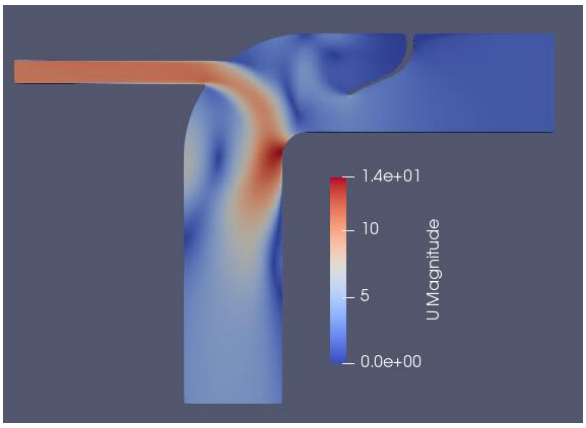


Figure 61. Velocity profile of optimized Y-junction

,

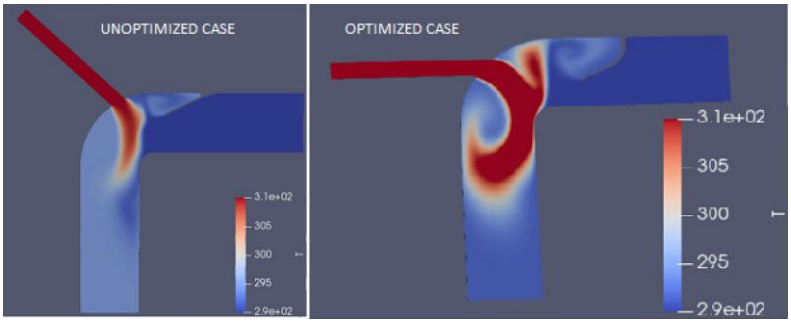


Figure 62. Comparison of temperature distribution between optimized and unoptimized shape

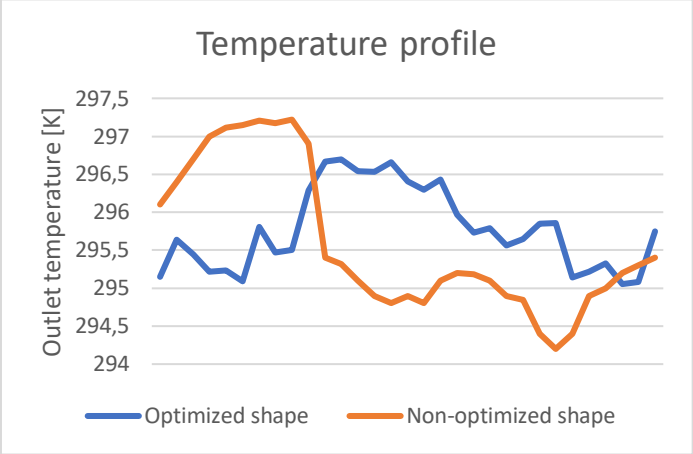


Figure 63. Temperature distribution at the outlet of optimized and unoptimized Y-junction

5.2 Analysis of optimization results – the second case

It can be seen clearly that the existence of the baffle improved the uniformity of velocity distribution in the bend. Moreover, the baffle also reduced the pressure loss over its length by 21%.

It might be seen that algorithm tries to match the shape of baffle to the fluid streamlines. Created object has an aerodynamical shape, the amount of separation is little.

At the graph, four designs were compared:

- The first one is the most fit result achieved by optislang with fitness value 511.3
- The second one is an unfit solution, which shows that the placement of baffle may be also disturbing if done incorrectly
- There is also an case of a pipe bend without baffle included which allows to compare the effect.

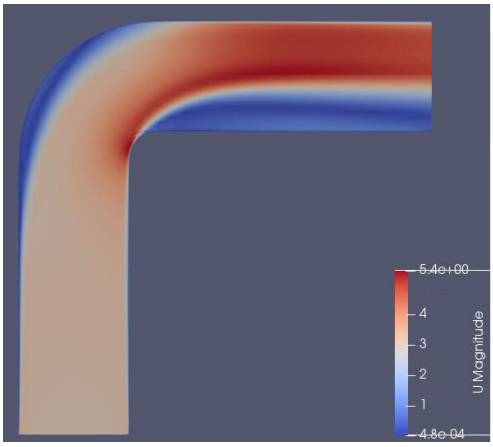


Figure 64. Velocity profile of case 2 pipe bend without a baffle (2nd case)

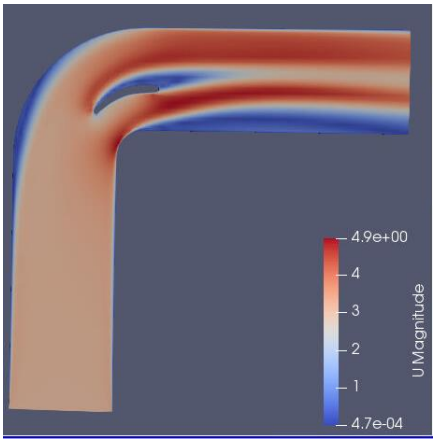


Figure 65. Velocity profile of optimized 2nd case (baffle with a bend, 2nd case)

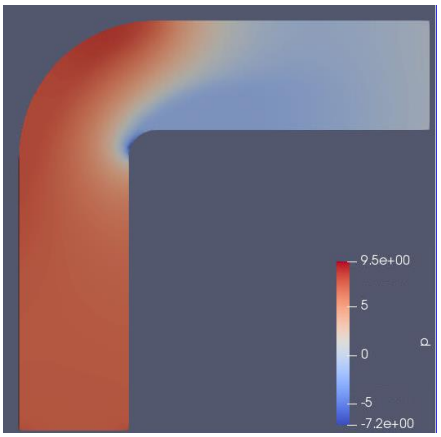


Figure 66. Pressure profile of bend without a baffle (2nd case)

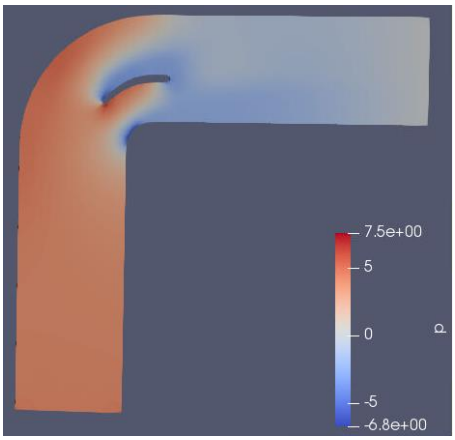


Figure 67. Pressure profile of optimized baffle in a pipe bend (2nd case)

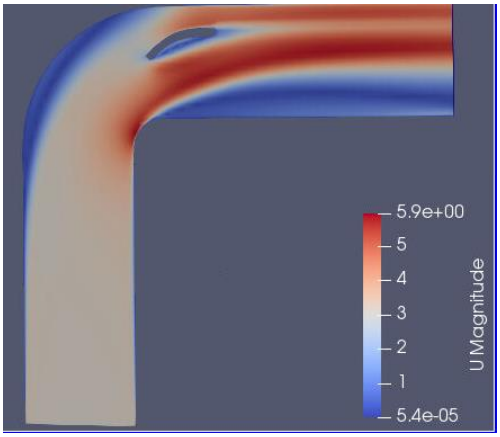


Figure 68. Velocity profile of a low-fitness baffle (2nd case)

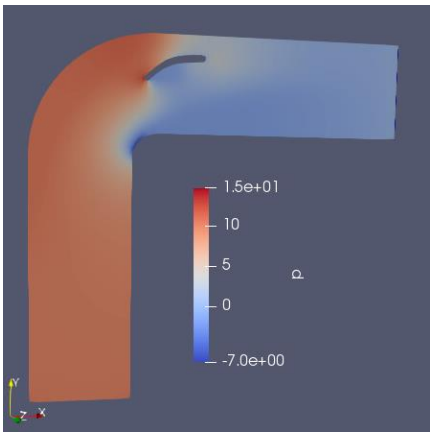


Figure 69. Pressure distribution of unfit pipe bend with a baffle

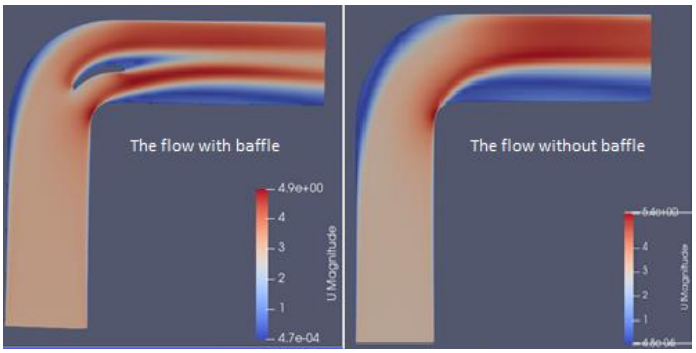


Figure 70. Comparison of velocity profiles of optimized bend and a bend without a baffle

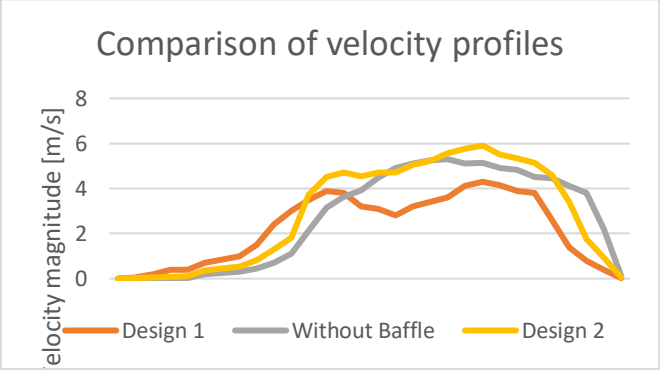


Figure 71. Comparison of velocity profiles

6. DISCUSSION

It is clearly seen that genetic algorithm can be used as an optimization tool and its accuracy is good. In most of the cases it could get a convergence, while under the most proper conditions it can find a solution with really low number of iterations (in examined cases, around 60-80). This is especially valuable when a sophisticated design is analyzed since no information about the gradient is required.

The role of metamodeling is mostly to expand initial population and the more it is applied, the more bias it introduces. On the other hand, even a low number of iterations (around 20 in this project) might lead to good predictions if the kriging is applied. While usually helpful, metamodeling might be sometimes very misleading and creates totally nonsense predictions. Furthermore, its effect is very random while the range of relative error very wide (from less than 1% till around 20%). There is no trend in relative error size and if not compared to the results gained by some trustworthy method (like CFD), the information is mostly useless – especially if only one specific design is examined.

The role of metamodeling is usually to “discover” a function hidden behind a discontinues data. The more “function-similar” the trend is, the better metamodeling prediction accuracy should it offer.

To expand this research, It might be good idea to work with bigger number of variables and more objective functions. According to the research conducted in this project it might be seen that metamodeling did not really work in case 1st optimization, while improved 2nd case by a large amount (in terms of simulation time). However, it should be discussed if and when this kind of metamodeling is worth the effort. How to predict if metamodeling will strengthen or destroy the convergence of specific case optimization? The research done in this project does not answer this question.

However, it might be a rather safe assumption, that a simple expansion of initial population pool with application of kriging is usually worth the effort (of course, only if the relative error of kriging approximation is low).

In terms of genetic algorithm and in general genetic optimization, a lot of research is conducted. There are many types of variators, mutation strategies, selectors etc. The same rule applies to Kriging and metamodeling types – there are a lot of methods of data prediction methods which potential in the CFD optimization should be examined.

7. BIBLIOGRAPHY

[1] https://www.researchgate.net/figure/3-Trend-of-Moores-Law-RD-5_fig6_259897062
[2] https://www.cfd-online.com/
[3] “Genetic Algorithms in Search, Optimization and Machine Learning”, 1st Edition, David A. Goldberg
[4] An Introduction to Genetic Algorithms, Jena Cars, 2014
[5] “Application of Latin Hypercube Sampling in the Immune Genetic Algorithm for Solving the Maximum Clique Problem”, Zhou Benda and Chen Minghua
[6] "Multi-parental Extension of the Unimodal Normal Distribution Crossover for Real-coded Genetic Algorithms," Proceedings of the 1999 Congress on Evolutionary Computation, pp. 1581-1588, 1999, Kita, H., Ono, I., and Kobayashi, S.,
[7] Optislang manual
[8] “Design Optimization of Fluid Machinery: Applying Computational Fluid Dynamics and Numerical Optimization”, Kwang-Yong Kim, Abdus Samad, Ernesto Benini
[9] “A Kriging Metamodel Assisted Multi-Objective Genetic Algorithm for Design Optimization”, M .Li, S. Li, S. Azarm
[10]” "Fifty Years of Kriging". Handbook of Mathematical Geosciences.”, Chilès, Jean-Paul, Desassis, Nicolas
[11] https://upload.wikimedia.org/wikipedia/commons/f/f5/Example_of_kriging_ interpolation_in_1D.png
[12] “Basic Linear Geostatistics”, Chapter 7 The Theory of Kriging, Margaret Armstrong
[13] https://platypus.readthedocs.io/en/latest/
[14]”An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach”, Part I: Solving Problems With Box Constraints, Kalyanmoy Deb, Himanshu Jain

[15] https://www.dynardo.de/en/software/optislang.html