POLITECNICO DI MILANO

M.Sc. in Aeronautical Engineering

Final dissertation

Inlet-Outlet Optimization Of The Cavity Tutorial
Dakota - OpenFOAM Case Study

Supervisor: Federico Piscaglia

Candidate: Luca Saglimbeni

ACADEMIC YEAR 2018-2019

Contents

Chapter 1

Introduction

1.1 Optimization

Optimization in fluiddynamic problems has always been an important issue. Wether there is the need of obtaining or trying to reach a certain value for a certain variable or combination of variables, the optimization problem allows to change all the other ones until the desired value is found. From now on this "desired value" will be called "objective function". By starting from this general point this work has the goal of finding an optimized state with regards to an objective function for a certain basic model by changing its variables until the best result is found.
The numerical optimization in practical applications is a growing issue in the industry and practical examples can be seen everywhere. There may be the need to find an efficient way to heat a room or to create a safer work environment by avoiding air currents inside an office, a lot of everyday problems can be solved as optimization problems. The complexity of the problem depends on many things, the number of variables, the complexity of the phenomenon under study, the boundary conditions or other constraints. It is impossible to find an optimal solution for a real life problem without the help of an optimization software and even then, the solution is found by using some simplifications or approximations. Optimization is obtained through software packages but it requires extensive computer resources. At least three prerequisites are required to implement successfully optimization in professional practice. These inlcude knowledge of optimization techniques, the mathematical modeling of the design problem and knowledge of programming. Another assumption frequently

made is that each user is able to implement the optimization in a mainframe computing environment. In recent years, though, new exciting possibilities arose. There have been several new software systems that deal with mathematics, graphics, and programming available, accompanied by numerous inexpensive desktop computing resources that aid engineering practice. (Venkataraman, 2009)

There are different software systems that are viable choices as of today. Benchmarking these possible choices is a popular argument and extensive papers have been written on this argument. The evaluation of different solvers involve tables displaying the performance of each solver on each problem for a set of metrics such as CPU time, number of function evaluations, or iteration counts. The problem, though, lies in the interpretation of the results obtained in these tables, which is often cause of disagreement. (Dolan & Moré, 2002) This is why benchmarking the methods used is not tackled in this work, the extensive resources required preclude an adequate number of iterations. A non-conclusive opinion will be given based on a small number of iteration for each method.

The majority of open-source optimization systems found online require extensive programming skill, that is why the Dakota software has been chosen as the optimizer software. The extensive resources online backed up by little programming skills allow to develop many interesting ideas without the need of expensive computing power. The meaning of this work, hence, is to devise a possible application by showing the optimization of an approximated fluiddynamic problem. A room has been approximated as a twodimensional box with current inside due to two openings in its walls. The position of these openings will be optimized with respect to an objective function defined by the user. Even though the problem at hand is simple, the tools used to solve it are powerful and highly customizable to every need.

1.2 Objectives

The first objective of this thesis is to find a way to make the optimizer software interact with the fluiddynamic solver. OpenFOAM will be used as the powerful tool to solve fluiddynamic problems and DAKOTA will be the optimization software. Having established the interaction between the tools, a basic problem was designed next. Since the goal is to prove the feasibility of this approach, the starting problem will be an alteration of the basic cavity tutorial with some tweaks to allow the optimization. The cavity model allows to understand the flow inside a 2D box with 3 fixed walls and a wall moving at constant velocity.

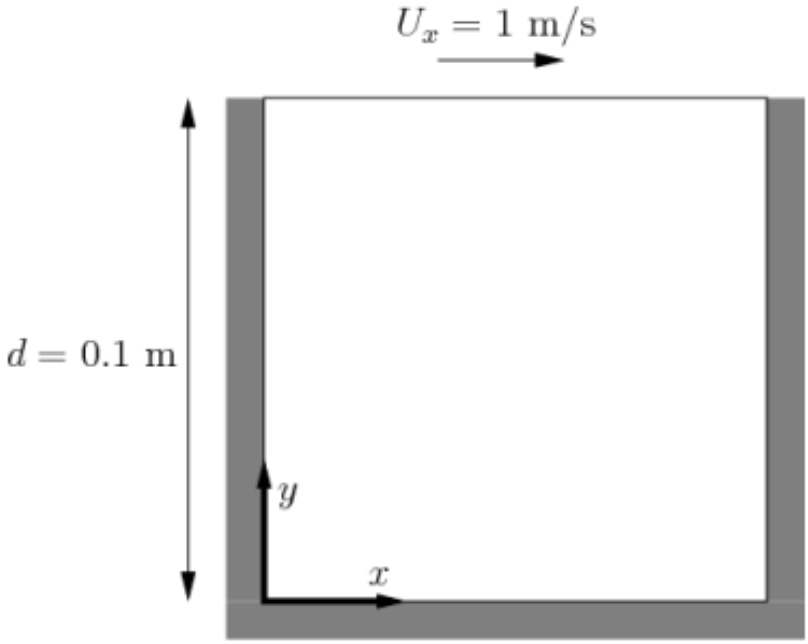

Figure 1.1: Domain of the cavity tutorial

The moving wall will be removed and an inlet and an outlet will be added to allow the circulation of the flow inside the box. An objective function will be defined from the variables of the flow inside the cavity

model and the interaction between the two systems will allow us to find the desired value for our objective function, which can mean more efficient, biggest value, lowest value or any other specific indication. Finally the OpenFOAM model will be tested within Dakota with different algorithms to see wether the results are coherent and to draw simple conclusion as to the performances of the different algorithms launched.

Chapter 2

Software Introduction

2.1 OpenFOAM Introduction

OpenFOAM (for "**Open**-source **F**ield **O**peration **A**nd **M**anipulation") is an open source CFD software. It is written in C++ and extremely flexible, various solvers are already compiled in it but customized numerical solvers are also possible, it has pre-/post-processing utilities for the solution of any physical problem including, most importantly for this work, computational fluid dynamics (CFD). It is developed primarily by OpenCFD Ltd since 2004. OpenFOAM has a considerable amount of features to solve anything from the most basic fluid flows without any chemical reaction, turbulence or heat transfer to the most complex ones including all of the above mentioned processes and more. (Ferziger, Perić, & Street, 2002)

A short explanation as to how the software works follows. The solving process of each case can be seen as 3 different operations:

- Pre-Processing
- Calculation
- Post-Processing

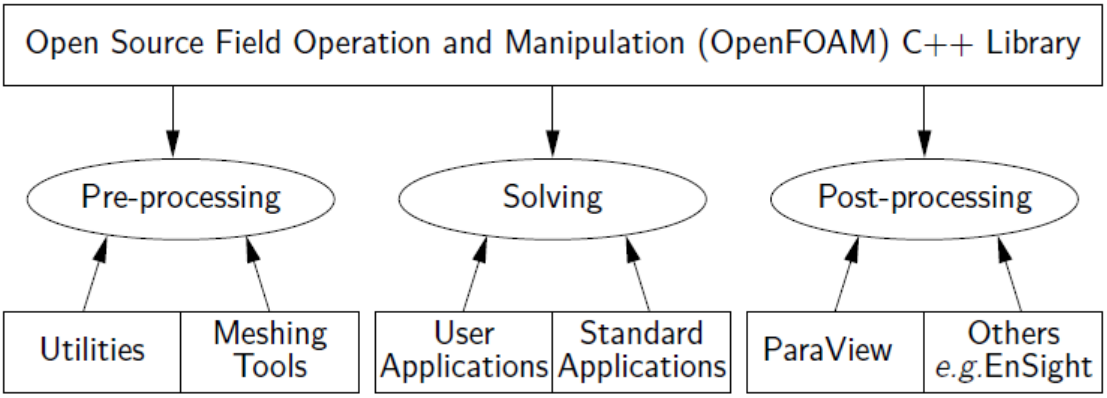

Figure 2.1: Processes followed by the software.

The **pre-processor** consists of the input of a flow problem into a form suitable for use by the solver. It usually involves the definition of the domain and its subdivision into a number of cells, the selection of the physical and chemical phenomena that are to be modelled, the definition of the properties of the flow and the details of suitable boundary conditions. The **calculation** step is structured around numerical algorithms that can simulate the problem at hand. The numerical solver performs the approximation of unknown flow variables and subsequent discretization by substitution of the approximation into the governing flow equations. Finally it solves the equation to find a solution that will be better the fewer approximation are made. The spatial discretization is achieved with the **Finite Volume** method. (F. Piscaglia, 2019)
Approximations of constitutive relations and governing equations are solved for each cell to get the solution of the model. The **post-processor** is used to visualize and extract relevant data from the previously solved simulation. These are the main steps followed by the software. OpenFOAM is a very powerful tool that allows to solve many different flows, hence there are many different solvers already compiled in it, including the one used in this thesis: `icoFoam`. This solver is used in striclty incompressible cases to solve transient problems. It is, although, required to use very small time steps to get converging results.

2.1.1 Introduction to OpenFOAM case structure

Each CFD problem is always composed by 3 folders:

- 0
- constant
- system

The folder **0** contains all the variables needed to simulate the problem at their initial condition. The **constant** directory contains the mesh and dictionaries for thermophysical and turbulence models. The **system** folder contains settings for the run, discretization schemes and solution procedures. It contains at least the following 3 files: *controlDict* where run control parameters are set including start/end time, time step and parameters for data output; *fvSchemes* where the discretisation scheme employed in the solution is chosen; and *fvSolution* where the equation solvers, tolerances and other algorithm controls are set for the run. Depending on the complexity of the case under study, a solver will be selected from the many already written inside OpenFOAM source code or compiled as needed. The case is solved by the solver that follows the settings written inside these files.

2.2 Dakota Introduction

The software Dakota delivers robust and usable perfomances in optimization and uncertainty quantification. Broadly, the Dakota software's advanced parametric analyses enable design exploration, model calibration, risk analysis and quantification of margins and uncertainty with computational models. It provides a flexible, extensive interface between such simulation codes and its iterative systems analysis methods, wich include:

- optimization with gradient and nongradient-based methods
- uncertainty quantification

- parameter estimation
- sensitivity/variance analysis

These capabilities may be used on their own or as components within advanced strategies such as hybrid optimization, surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. (Dakota, 2009)
Here it will be used together with OpenFOAM as an optimizer to find the best possible value for the variables that generate the geometry of our domain. The goal of this work, in general terms, is to find the configuration that allows for the most uniform flow in a specific region of the domain or in the whole domain. More details will be added in the following chapters. A brief explanation as to how optimization within Dakota works will be presented now.

2.2.1 Optimization Capabilities

The Optimization algorithms work to minimize (or maximize) the **objective function**, typically calculated by the simulation code supplied by the user, subject to constraints on design variables and responses. There are different approaches in Dakota including well-tested and proven gradient-based, derivative-free local, and global methods to be used in design applications. Dakota also offers more advanced algorithms to manage multi-objective optimization or perform surrogate-based minimization but these won't be mentioned here. This chapter summarizes optimization problem foundation, some algorithms available in Dakota, which are important to the problem at hand, and a guideline to follow.

2.2.2 Optimization Formulation

Here a basic introduction to the mathematical formulation of optimization problems is provided. A general optimization problem is formulated as follows:

$$\text{minimize: } f(\text{x}) \; x \in R^n$$

$$\begin{gathered}\text{subject to: } g_L \leq g(x) \leq g_U \\ \mathrm{h}(x) = \mathrm{h}_t \\ a_L \leq A_i x \leq a_U \\ A_e x = a_t \\ x_{textitL} \leq x \leq x_U\end{gathered}$$

where vector and matrix terms are written in bold. In this formulation, x is an n dimensional vector of real-valued *design variables* or *design parameters*. $x_L and x_U$ n-dimensional vectors are the lower and upper boundaries for the *design variables* of the problem. The *design variable* can assume any value between the lower and upper bounds of the *design space*. Chosing a set of *design variables* means setting a *design point* within the parameter space. The *objective function*, f(x) has to be minimized or maximized in a optimization problem while satisfying the constraints. There are different types of constraints:

- linear or nonlinear
- inequality or equality

The *nonlinear inequality constraints*, g(x), as *linear inequality constraints*, are "2-sided" because they have both lower and upper bounds, g_L and g_U, respectively. The *nonlinear equality constraints*, $\mathbf{h}$(x), have target values specified by $\mathbf{h}_t$. The *linear inequality constraints* create a linear system $\mathbf{A}_i$x, where $\mathbf{A}_i$ is the coefficient matrix for the linear system. The linear equality constraints create a linear system $\mathbf{A}_e x$, where $\mathbf{A}_e$ is the coefficient matrix for the linear system and $\mathbf{a}_t$ are the target values. The space of parameters is hence divided into *feasible* and *infeasible* values. All this is to say that the optimization problem formulated before can be solved in different ways, all of which are based on the iteration on the value x in some manner. This means that after an initial value for each parameter x is generated, the *response quantities* are computed, f(x), g(x), $\mathbf{h}$(x), usually by running a simulation. Then a new parameter x is found using a chosen algorithm, with it we can either compute a better function object, reduce the infeasibility, or both.
The different optimization methods can be categorized by the

optimization problem type which means on the type of constraints (linear, nonlinear, equality, inequality, constrained, unconstrained); by the **search goal**, *local optimization*, *global optimization*; by the **search method**, which means the approach taken in the algorithm to find a new design point, *gradient-based*, *nongradient-based*. Local optimization means finding an optimal design point relative to the "nearby" region of the parameter space while global optimization aims to find the best possible objective function over the entire parameter space. In gradient-based algorithms, gradients of the response functions are computed to find the direction of improvement. This is usually very efficient in local optimization methods but it may not be the right solution if gradients are computationally too expensive, inaccurate or nonexistent. In this case, nongradient-based algorithms are the best choice. There are a lot of different approaches with nongradient-based optimization. This brief explanation tells us that there is no single "best choice" of a single optimization approach for all types of optimization problems. (Dakota, 2009)

2.2.3 Derivative-Free Methods

The optimization problem at hand is a constrained global optimization problem where a gradient-based algorithm isn't necessarily the best choice since a gradient may be unreliable. This means that the choice falls on nongradient-based global methods. It is important to add though, that derivative-free methods exhibit much slower convergence rates for finding an optimum, and as a result, tend to be much more computationally demanding than gradient-based methods. Now let's dig deeper into the possible choices offered by Dakota.

- Evolutionary Algorithms (EA)
- DIvision of RECTangles (DIRECT)

Evolutionary Algorithms (**EA**) are based on the survival of the fittest theory by Darwin. The EA algorithm selects random points in the design space that form an initial "string" much like DNA. The algorithm then generates new parameters from the best design points of the previous

generation that are considered to be the most promising and are allowed to endure in the following generation. The EA follows the mathematical analogous of reproduction, natural selection and mutation.

DIvision of RECTangles (**DIRECT**) searches both in promising regions of the feasible design space in the neighborhood of a global minimum and in unexplored regions.

2.3 Guideline

The most important file in any Dakota simulation is the **input file**. The software is launched by calling out the input file. After that, Dakota outputs a large amount of information to help track progress in new files:

- The screen output can be saved in a *.`stdout` file if instructed
- An output file *.`out` that contains all the informations about the functions evaluation
- A *.`dat` file (due to an option in the input file) that summarizes the variables and responses for each iteration of the evaluation
- Finally a `dakota.rst` which is a restart file in case the simulation has to be paused and restarted again.

The *.`stdout` file allows to reduce the screen output to a minimum. The **output file** is much more extensive, it contains information on every iteration carried out by Dakota. It starts with a copy of the **input file** at the top and timing information at the bottom of the file. Generally the file is divided into 3 parts:

- Information on the problem
- Information on each function evaluation
- Summary statistics

2.3.1 Dakota Input File Format

The input file is the controller of the software iterations. It is divided into six blocks identified by keywords:

- variables
- interface
- responses
- model
- method
- environment

The order in which they are written into the input file isn't important but there is an important relation between these blocks which is summarized in the following figure from the User Manual of Dakota. (Dakota, 2009)

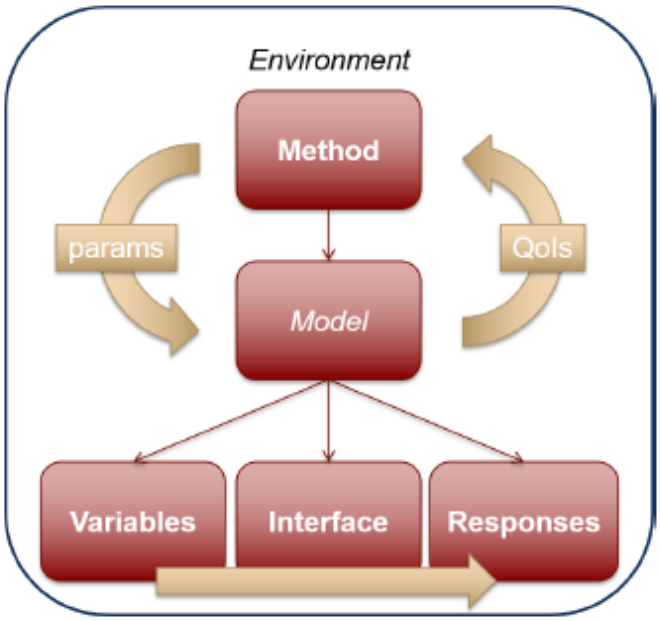

Figure 2.2: Dakota blocks relation

This relationship can be summarized as follows: for every iteration of the chosen algorithm, a *model* is chosen and the *method* block requests a particular *variables-* to *-responses* mapping. The model satisfies the requests through an *interface*. As an example of an input file, let's look at the input file of the case under study.

The `environment` block is optional, it is used to specify the general Dakota settings such as Dakota's graphical output and the tabular data output. The `method` block identifies the iterative method used by Dakota and the options associated. This block is required by Dakota and there also may be more than one. The model specifies how a set of variables is mapped through an interface into a set of responses. The model block is optional just in the default case which allows to specify a single set of variables, interface and response. The `variables` block specifies the number, type and characteristics of the parameters that will be continously generated by Dakota in each iteration. There are three types of variables, design variables, uncertain variables or state variables. These can also be continous or discrete. The sub-specifications for continuous design variables provide the descriptors as well as lower and upper bounds for these variables. All these information are written in column form for readability. The `interface` block specifies the code that will be used to run the simulation and the details as to how the data are passed between Dakota and the simulation code. This is fundamental in this work since Dakota is going to be linked with OpenFOAM. The keyword `direct` is used to indicate the use of a function directly into Dakota, while `fork` invokes an external simulation code, like an OpenFOAM simulation. In this last case, data is passed between Dakota and the simulation via text files. At least one `interface` block is reaquired. The `responses` block contains the type of data that the interface will return to Dakota after each iteration. In **optimization** cases like this one, objective functions will be used, as mentioned previously. There are other types of response data but their description is beyond the scope of this chapter. The keywords `no_gradients` and `no_hessians` means that the simulation won't provide any derivative to the method.

```
# Usage:
#   dakota -i xxx.in -o run.out > stdout.out
#############################################################################
environment
     graphics
     tabular_data
        tabular_data_file = 'table_out.dat'
#############################################################################
method
coliny_direct
max_function_evaluations = 50

#############################################################################
variables
continuous_design = 2
descriptor       'teta1''teta2'
lower_bounds  0.01 5.55
upper_bounds  5.48 6.27

#############################################################################
interface
fork
asynchronous evaluation_concurrency = 4
analysis_driver = 'simulator_script'
parameters_file = 'params.in'
results_file    = 'results.out'
work_directory directory_tag

        copy_files = 'templatedir/*'
dprepro
#############################################################################
responses
objective_functions = 1

no_gradients
no_hessians

sense 'max'
#############################################################################
```

Figure 2.3: Dakota input file

2.4 Dakota Setup

It is useful to underline how the software Dakota is coupled to a user supplied simulation (in our case OpenFOAM). The files needed to run the iterations with the Dakota software are few, the `dakota_ of.in` input file, the **casebase** folder that contains the OpenFOAM simulation to be iterated, the script `dprepro` which is fundamental in setting up each new case together with the folder **templatedir**. Dakota will be launched by

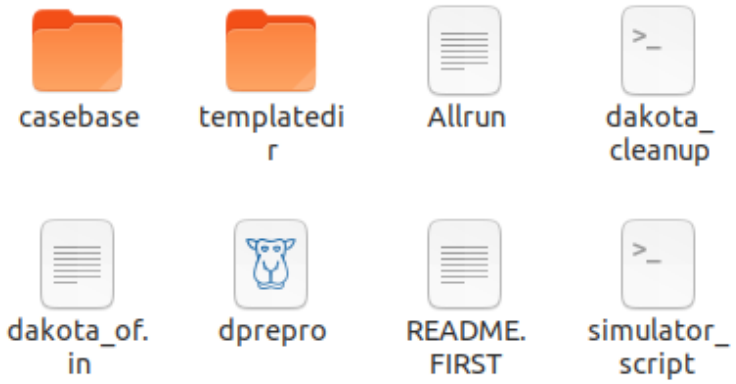

Figure 2.4: Dakota main files

calling out the input file as usual, the only difference from a case run within Dakota will be in the **interface** block. There, the keyword **fork** will be written instead of **direct**. The fork keyword calls out another script wich will act as the **driver script** of the problem, the **simulator_script**. The simulator_script is fairly simple, it contains the commands needed to copy the new parameters into a new OpenFOAM simulation and then to launch the simulation. The preprocessing block calls out an executable file called **dprepro** which is tasked to read each passage in the file **topoSetDict.template**, which is inside the **template** directory, and find each keyword **teta1** or **teta2**(our two variables in this specific case). After that it will substitute the new parameters (which are represented by the keyword $1), generated at each iteration, into the topoSetDict.template file, at each location where the keywords teta1 and teta2 have been read. Finally the file topoSetDict.template will be renamed **topoSetDict.in**, once the variables have been copied inside of it. The analysis block is used to copy the casebase, which contains the

```
# --------------
# PRE-PROCESSING
# --------------
# Incorporate the parameters from DAKOTA into the template

dprepro $1 topoSetDict.template topoSetDict.in

# --------
# ANALYSIS
# --------

pwd
cp -r ../casebase/* .
cp topoSetDict.in system/topoSetDict

./Allclean
./Allrun
```

Figure 2.5: The simulator_script

standard OpenFOAM simulation without the new parameters created by Dakota, into the template directory. The new topoSetDict.in file will substitute the standard topoSetDict file to feed the new variables into the simulation. Finally, the simulation will be launched with the new variables. There may be a **POSTPROCESSING** block in the simulator script to get the response function that results from the OpenFOAM simulation that was run before, back to the Dakota software. Here the postprocessing part needed to extract the numerical value of the **flow uniformity** was written inside the **Allrun** script, inside the casebase folder, hence it was not necessary to write it again into the simulatorscript. Once one iteration reaches its end, the results and the simulation are written in a new directory called **workdir.*** where the number of the iteration is written inside the "*". The response function (the objective function) is read by Dakota which then, following the method guidelines, creates new variables for the next iteration.

Chapter 3

Working Case

3.1 Physics Of The Model

The starting point of this work is the *cavity* model. The OpenFOAM tutorial consists in a square two-dimensional box with 3 fixed walls and a moving wall as boundary conditions. This generates an isothermal, incompressible flow in a two-dimensional square domain.

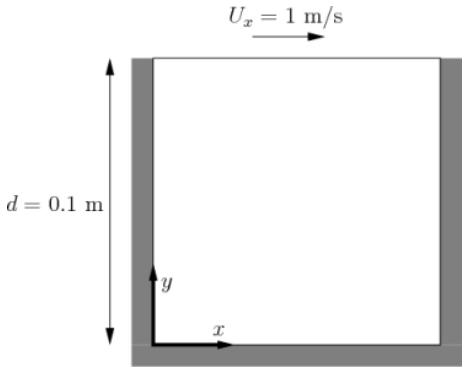

Figure 3.1: Cavity tutorial domain

This tutorial has been modified by adding an inlet and an outlet to the box and by removing the moving wall at the top of the domain. The final result is a box with 4 fixed walls and 2 openings. The physics of the problem does not change, this allows us to use a similar case setup during

the simulation. The particular case at hand enables us to use the solver *icoFoam* for laminar, isothermal, incompressible flow which means to have a continous flow inside the box that our software OpenFOAM is able to solve easily.

3.2 OpenFOAM Case Setup

As previously said the **casebase** folder is composed by five objects, three folders and two scripts:

- 0 folder
- constant folder
- system folder
- Allrun script
- Allclean script

Figure 3.2: OpenFOAM casebase folder elements

The **0** folder contains just the two variables required by the solver icoFoam: `p, U` with their starting value at time 0. The pressure has a set outlet value and a zero gradient boundary condition applied on the walls and on the inlet, while an empty condition is set on the front and back because it is still a two-dimensional case. The velocity has a `noSlip` condition on the walls, a fixed value at the inlet and a zero gradient at the outlet.This concludes the variables section which is straightforward. The `constant` folder contains the mesh and the `transportProperties` file which defines a transport model and the constant `nu`. The **system** folder is more complicated than the other two, it contains the files needed to control the case, `controlDict, fvSolution, fvSchemes`, the files needed to do the postprocessing, `sampling`, the script to generate the

mesh `blockMeshDict`,the script that allows parallel processing `decomposeParDict` and finally the files needed to change the position of the **inlet** and **outlet** during each iteration, `createPatchDict`, `topoSetDict`. The files to control the case are pretty standard, no changes have been made from the tutorial case with the exception of some minor tweaks. The `controlDict` has been changed so that more time steps have been written and the postprocessing has been added at the bottom of the file. The post processing, called out from the `controlDict` file, invokes the sampling file.

3.2.1 Sampling

The **sampling** file allows for multiple sampling operations on surfaces inside the domain. The sampled variables are chosen at the top of the file, in our case just the variable `U` is relevant to our objective. The next step consists in defining the place where the sampling operation is done, this file allows a lot of freedom in defining the sampling location.

1. The **variable** to be sampled is defined
2. The sampling plane is chosen by defining a position and a **normal** direction to it
3. The **extension** of the plane is defined (can also extend outside of the domain)

It can be seen that this means a lot of flexibility in the position of the sampling probe. In practical terms this allows to define a precise position anywhere inside the domain, the potential problems that can be tackled in this way are countless.

Going back to the sampling file, the calculations to be done with the sampled variables follows. The operations included inside the standard file are already extensive:

- massflow
- areaAverage

- areaIntegrate
- flow uniformity
- weighted uniformity

In any case it is straightforward to add any desired operation to it. In this case it is not necessary since the **uniformity** operation is already defined inside OpenFOAM source code. Finally the plane (or planes) previously defined is selected and the operation to be done with the variable (or variables) sampled in that plane is chosen with a keyword. The exact operation done by calling out the desired keyword is written inside the `surfaceFieldValue.C` which is part of OpenFOAM source code.The surfaceFieldValue function object provides options to manipulate surface field data into derived forms without the need to write and compile any calculation from scratch. The **uniformity** equation is written as follows inside `surfaceFieldValue.C`:

$$ui = 1 - numer/(2 * mag(mean * areaTotal) + ROOTVSMALL); \quad (3.1)$$

Uniformity index

Where **numer** and **mean** are:

$$numer = gSum(mag(areaVal - (mean * mag(Sf)))); \quad (3.2)$$

Absolute deviation from unweighted mean value

$$mean = gSum(areaVal())/areaTotal; \quad (3.3)$$

Unweighted mean value (area-averaged)

$$areaTotal = gSum(mag(Sf)) \quad (3.4)$$

$$tmp < scalarField > areaVal(values * mag(Sf)) \tag{3.5}$$

Where `mag()` is the magnitude of a vector, `Sf` is the face area vector and gSum is the global summation of a vector.

3.3 Dakota Setup

Starting off with the `dakota_of.in` input file, some different algorithms will be tested, all of them derivative-free global methods though, since it is a requirement to find a **global maximum** for the **uniformity index**. The tested algorithms are 3 division of rectangles algorithms and one evolutionary algorithm, all already built in Dakota source code:

- ncsu direct
- genie direct
- coliny direct
- coliny ea

3.3.1 Variables Block

The `variables` block contains the values that have to be fed to the **templatedir** folder. This folder contains the necessary script to change the geometry at each iteration, the `topoSetDict.template`. It works together with `createPatchDict` to change the domain in order to shift constantly the location of the **inlet** and the **outlet** before each iteration. The variables are put inside the file and undergo some mathematical operations before becoming the group of coordinates needed to define the **inlet** position and **outlet** position. The variables in the Dakota input file are two, one for the inlet and one for the outlet. The basic idea behind this is to use **polar** coordinates. The polar system is a two dimensional coordinate system that locates a point with a **radius**(distance) and an **angle**. The variables inside `dakota_ of.in` are angles in **radians**. The first one `teta1` ranges from **0** to **5.48** because we have chosen to change

the position of the inlet over all the walls of the domain with the exception of half of one wall where we change, instead, the position of the outlet. That is why the second variable `teta2` has a lower bound of **5.55** and upper bound of **6.27**. The variables do not cover the entire 360 degrees arc to avoid any unwanted overlapping.

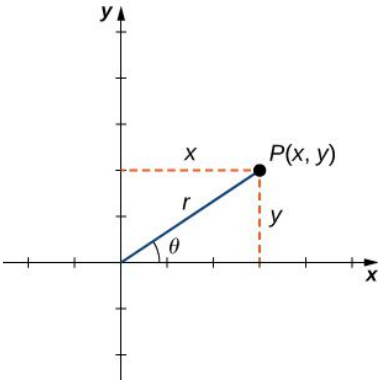

Figure 3.3: Arbitrary point in the Cartesian plane

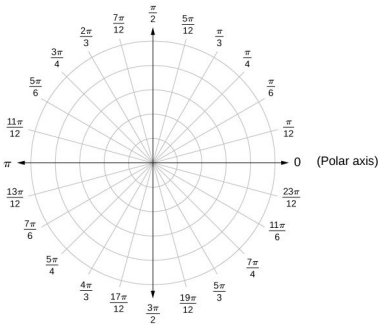

Figure 3.4: Polar coordinates system

The mathematical operations done inside the `topoSetDict.template` file modify the polar coordinates received as variables from the input script to become the **Cartesian** coordinates of the inlet and outlet positions before running the next simulation inside OpenFOAM. Depending on the efficiency of the selected method, the variables will be iterated in a different way and the maximum objective function will be found in a **faster** or **slower** way.

3.3.2 Interface Block

The interface block works as explained in the previous chapters, simulation interfaces which employ system calls and forks to create separate simulation processes must communicate with the simulation code through the file system. This is accomplished through the reading and writing of input parameters and results files. The input file invokes the `simulator_script` file as analysis driver. This script is customized to

this specific problem, it calls the standard Dakota executable `dprepro` to write the variables, generated by the input file, into the simulation file `topoSetDict.template` where the variables in radians become geometrical cartesian coordinates. In the next step the analysis driver will create a copy of the folder casebase, where the OpenFOAM simulation is run, and copy the new `topoSetDict` into the system folder. Finally the OpenFOAM simulation is launched inside this new folder, the results will be written in a file called `results.out` in a form readable by the Dakota software and the analysis driver job is done. The results contain just the **uniformity index** of this specific iteration which means a number between 0 and 1. The `results.out` is read by the input file that prepares the next iteration of variables according to the chosen **method**.

3.3.3 Response Block

The bottom of the custom input file is quite standard. It tells us that only one **objective function** will be defined (the uniformity index) and it also tells us if there is the need for derivatives from the OpenFOAM simulation. In this case, as previously said, there is no need for either **gradients** or **hessians**. Finally, the last option to specify is wether we search for a minimum or a maximum value for our objective function. Since we are searching for the "most uniform" flow we need to have a uniformity index as close as possible to **1**, hence we are searching for a **global maximum**.

Chapter 4

Results

4.1 Optimization Results

Previously we talked about the potential of this combination of softwares that allows a lot of flexibility depending on the goal to be reached. In particular the extensive customization that allows the sampling tool gives us the possibility to easily set up different goals equally important. This is what the title refers to, we have the means to find the best case scenario in either a small section of the domain or the whole domain. We can change the sampling script to our needs, for example to find the "most uniform" flow in terms of speed uniformity in the center of the domain, or the lower left corner of the domain or the entirety of the domain. These will be the cases tackled to prove the effectiveness of the optimization. We will also see the different results if higher precision is used, a finer mesh and more iterations or even different solvers. The solvers that will be launched are:

- coliny_direct
- genie_direct
- ncsu_direct
- coliny_ea

All the compared simulations will run the same number of iterations and the same settings for the OpenFOAM simulation. The chosen number of iteration is **50** because of the computing power and time required for each simulation to run.

4.1.1 Regional Domain Optimization

Starting from the basic case, the sampling has been changed according to the goal set. We are still going to study the **global optimization** of the flow but localized in a precise position of the domain instead of the whole domain (which will be studied after). The two cases of global optimization in a precise area of the domain are:

- Global optimization in a small area in the center of the domain
- Global optimization in a small area in the lower left corner of the domain

Center Regional Optimization

In this first case the sampling script is changed so that the sampling area is located in (0.05 0.05) with an extension of 0.01 in each direction. The exact sampling area (in the **y** direction) is shown next. The same center of the plane with a (1 0 0) normal vector and same extension is the sampling area in the **x** direction, it is not shown because it is essentially the same, just rotated.

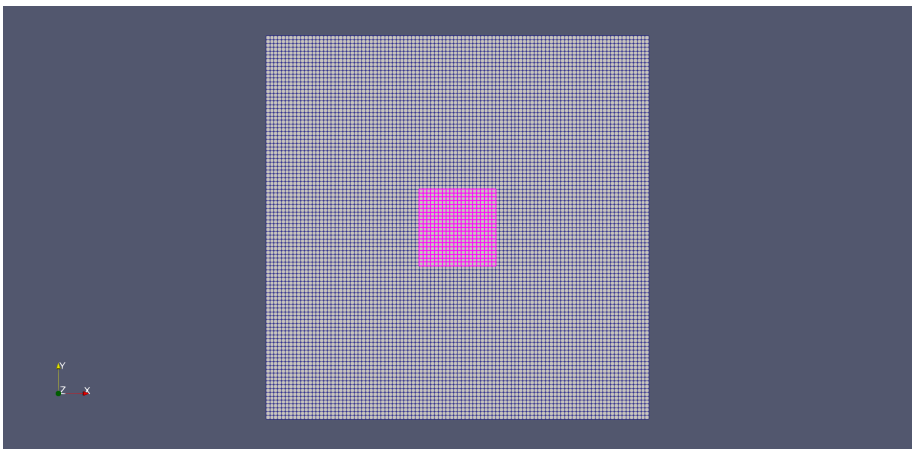

Figure 4.1: Local center sampling area

Different solvers will be used and the results will be compared to each

other. The first solver that will be launched is **coliny_direct**. A brief conclusion of the optimization results is found at the bottom of the run.out file.

```
300 <<<<< Function evaluation summary: 57 total (57 new, 0 duplicate)
301 <<<<< Best parameters          =
302                      2.2153703704e+00 teta1
303                      5.7944444444e+00 teta2
304 <<<<< Best objective function  =
305                      9.8481500000e-01
306 <<<<< Best data captured at function evaluation 41
307
308
309 <<<<< Iterator coliny_direct completed.
310 <<<<< Environment execution completed.
311 DAKOTA execution time in seconds:
312   Total CPU        =    4.30474 [parent =    4.30479, child =   -5.3e-05]
313   Total wall clock =    19267.6
314 Exit graphics window to terminate DAKOTA.
```

Figure 4.2: Values of the objective function obtained from the algorithm coliny_direct

The **41st** iteration was the most successful one for this particular configuration and this result is the same that is shown in the brief conclusion summary previously seen. It may be that this same result is obtained in multiple configurations similar to this one. This can be avoided with an increased precision in the solving process or with a finer mesh.

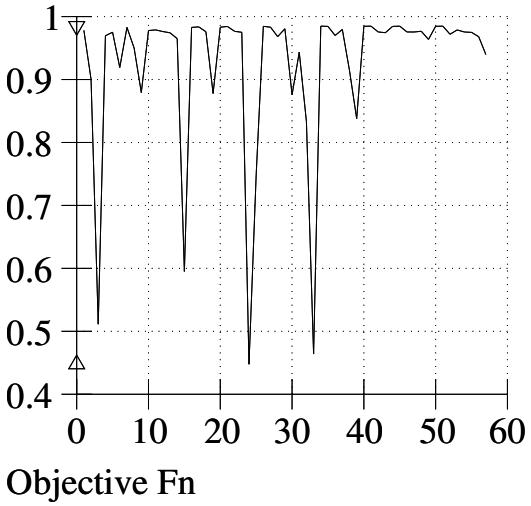

Figure 4.3: Values of the objective function obtained from the algorithm coliny_direct

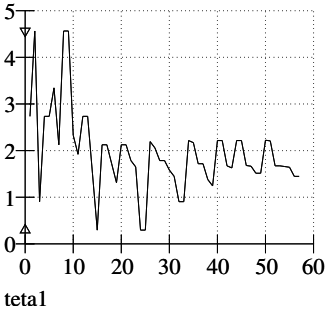

Figure 4.4: Values of the variable of the position of the inlet

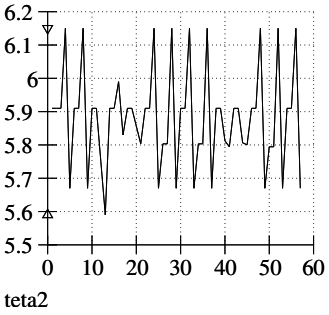

Figure 4.5: Values of the variable of the position of the outlet

The inlet and outlet positions in this cofiguration are shown next, the inlet is on the top of the domain while the outlet is on the lower right of the domain. The configuration of the cases that bear same solution is almost the same as this one.

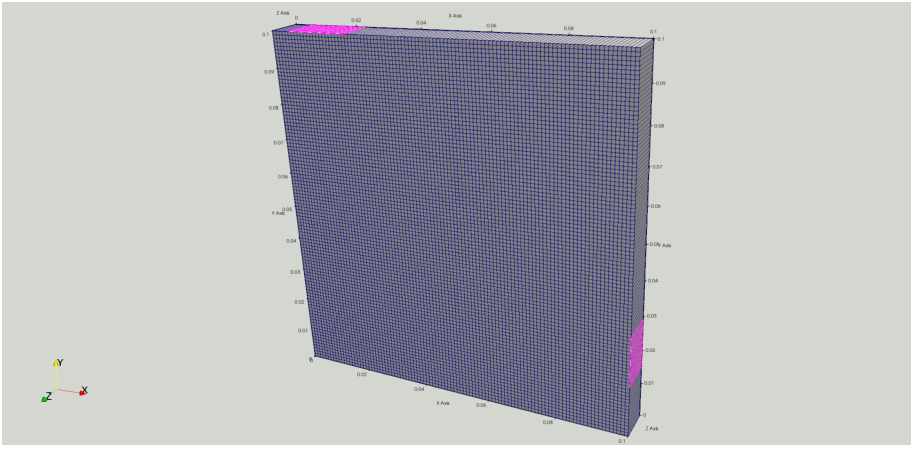

Figure 4.6: Representation of the positions of inlet and outlet obtained in the best configuration found by the algorithm coliny_direct and sampling in the centre of the domain

The **uniformity** value in this configuration is **0.984815**. The resulting flow is represented next.

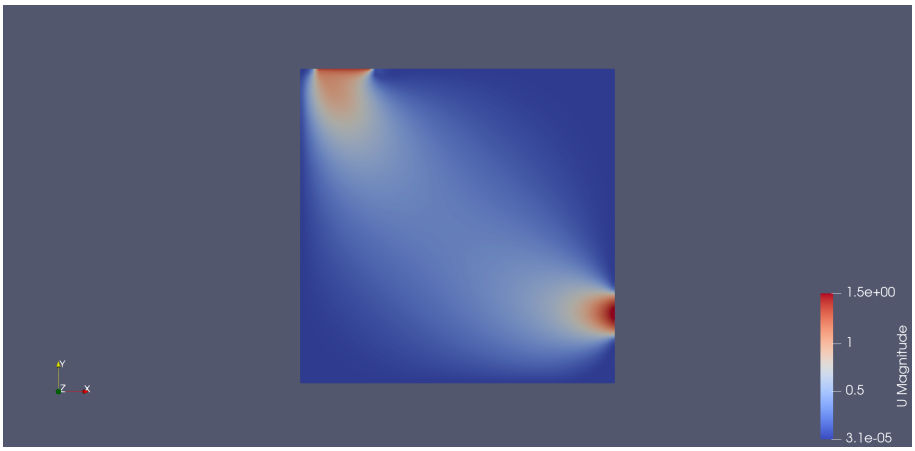

Figure 4.7: Representation of the flow obtained in the best configuration found by the algorithm coliny_direct and sampling in the centre of the domain

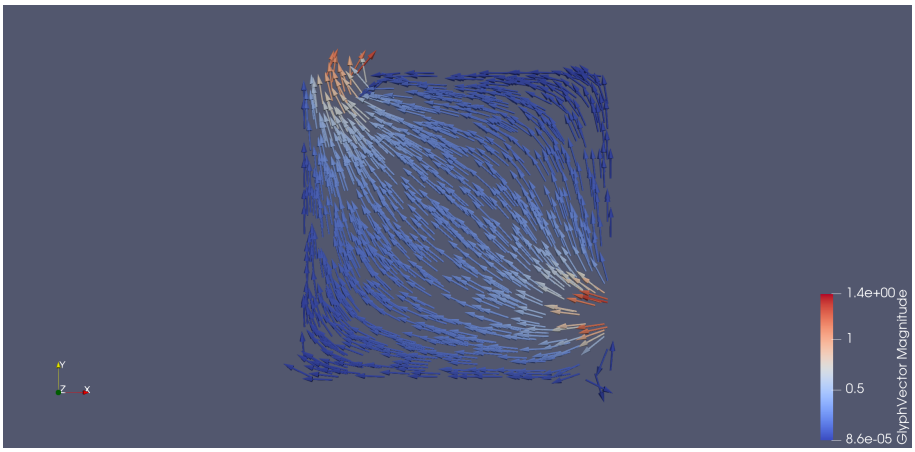

Figure 4.8: Vectorial representation of the flow obtained in the best configuration found by the algorithm coliny_direct and sampling in the centre of the domain

Now the same will be done with another algorithm, the `genie_direct` algorithm. We expect to get similar result, if not the same, even though it is obtained through a different approach.

The `genie_direct` algorithm finds the lowest value for an objective function, hence, to get the highest uniformity value, its inverse is used as objective function.

```
<<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
<<<<< Best parameters          =
                     2.4638888889e+00 teta1
                     5.8300000000e+00 teta2
<<<<< Best objective function  =
                     1.0074600000e+00
<<<<< Best data captured at function evaluation 44


<<<<< Iterator genie_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    2.61772 [parent =     2.6178, child =   -8.4e-05]
  Total wall clock =      15920
```

Figure 4.9: Values of the objective function obtained from the algorithm genie_direct

This tells us that the best result found is **1.00746** which corresponds to a uniformity value of **0,99259524**. The `genie_direct` algorithm finds, with the same 50 iterations, a higher uniformity value with respect to the `coliny_direct` algorithm. The various iterations are shown next.

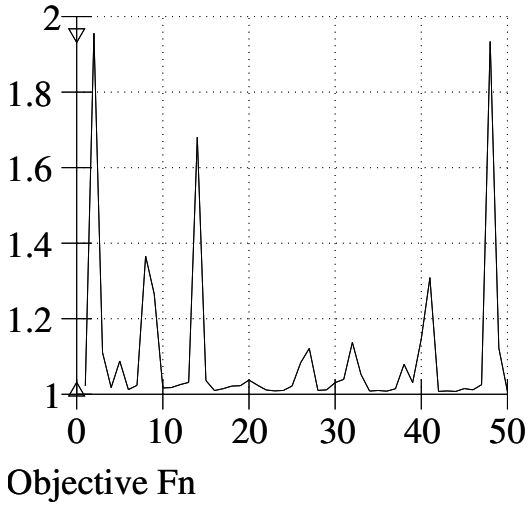

Figure 4.10: Values of the objective function obtained from the algorithm genie_direct

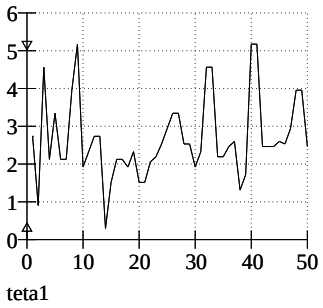

Figure 4.11: Values of the variable of the position of the inlet

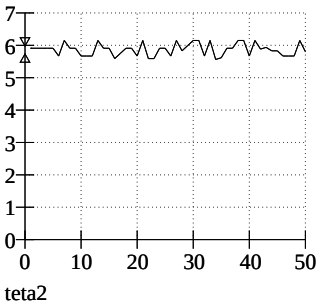

Figure 4.12: Values of the variable of the position of the outlet

The inlet and outlet positions for the 44th iteration, the one that yields the highest uniformity value, are shown next.

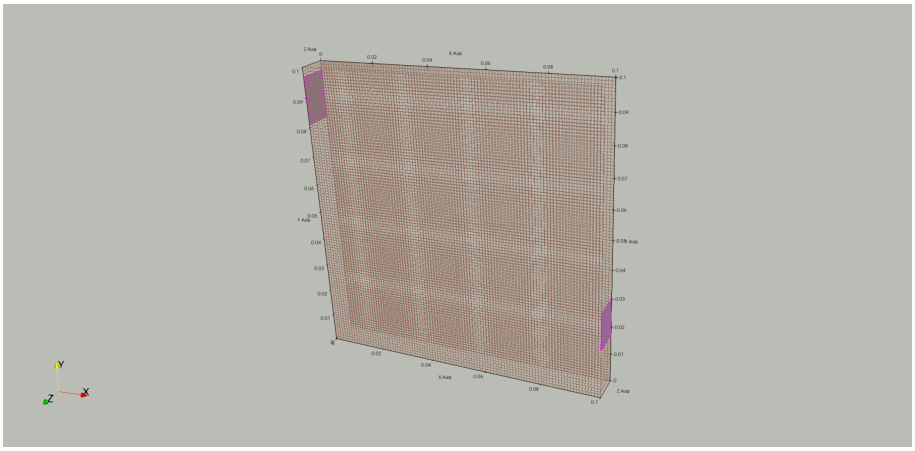

Figure 4.13: Inlet and outlet positions in the 44th iteration by the algorithm genie_direct and sampling in the centre of the domain

The configuration is similar to the one obtained from the previous algorithm, although the inlet position here is shifted with respect to the other case. The resulting flow is represented next.

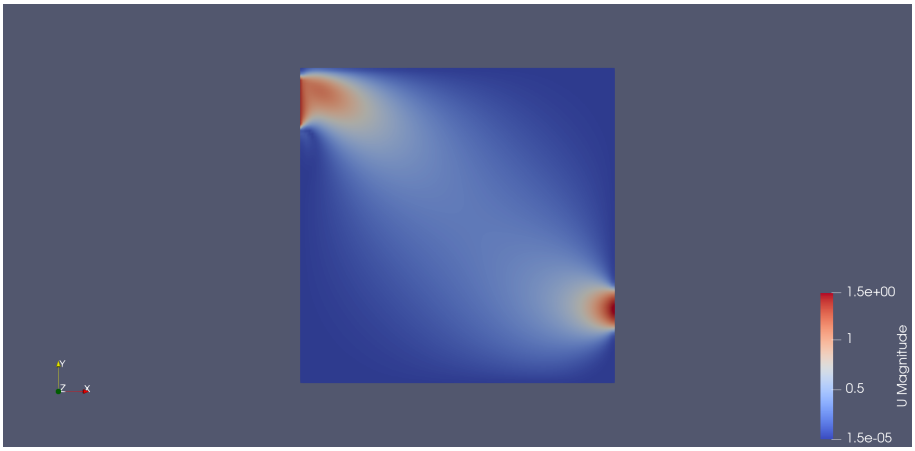

Figure 4.14: Flow in the best configuration found by the algorithm genie_direct with sampling in the centre of the domain

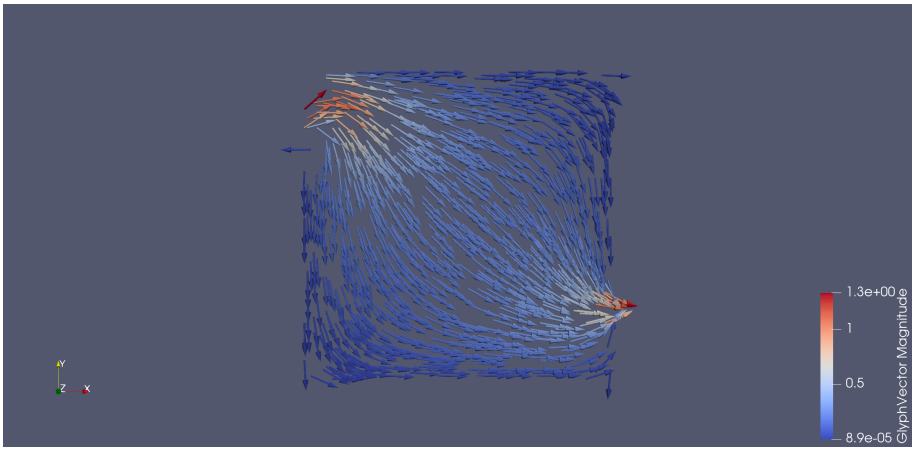

Figure 4.15: Vectorial representation of the flow obtained in the best configuration found by the algorithm genie_direct and sampling in the centre of the domain

The `ncsu_direct` algorithm follows.

```
NCSU DIRECT succeeded with code 1
(maximum function evaluations exceeded)
<<<<< Function evaluation summary: 55 total (55 new, 0 duplicate)
<<<<< Best parameters          =
                     2.1927777778e+00 teta1
                     5.5544444444e+00 teta2
<<<<< Best objective function  =
                     9.9170000000e-01
<<<<< Best data captured at function evaluation 55


<<<<< Iterator ncsu_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    1.39317 [parent =    1.39317, child =          0]
  Total wall clock =    6232.15
Exit graphics window to terminate DAKOTA.
```

Figure 4.16: Summary of the `ncsu_direct algorithm`

The best uniformity value found is **0,9917**. It is found in function evaluation number **55**. The values of the two variables in this configuration are **teta1** equal to **2,193** and **teta2** equal to **5,554**.

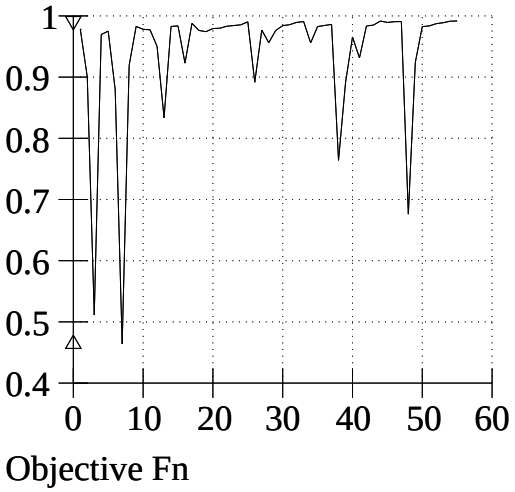

Figure 4.17: Values of the objective function obtained from the algorithm ncsu_direct

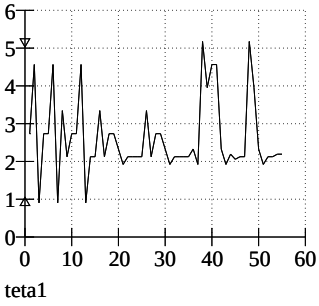

Figure 4.18: Values of the variable of the position of the inlet

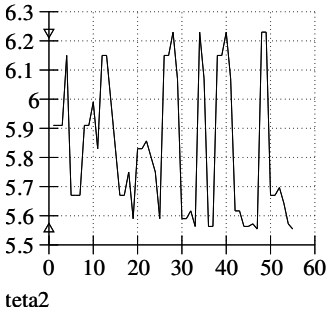

Figure 4.19: Values of the variable of the position of the outlet

The inlet and outlet position for the 55th iteration and the relative flow:

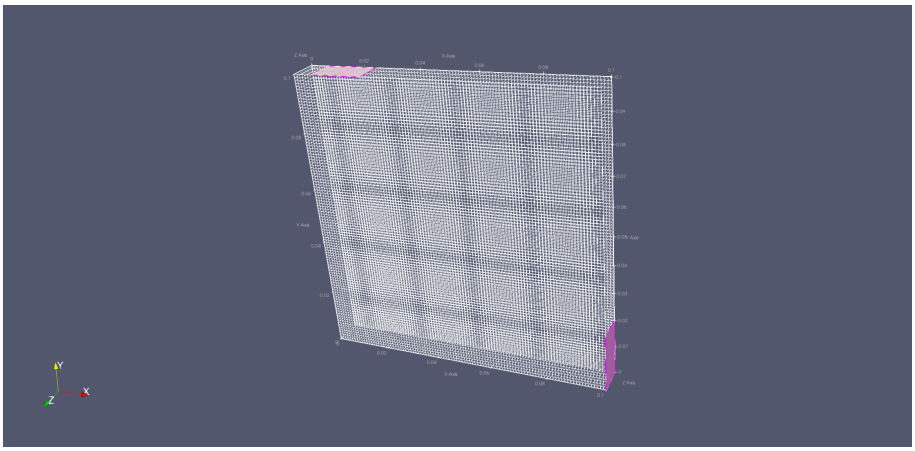

Figure 4.20: Inlet and outlet position for the best configuration found by `ncsu_direct algorithm`

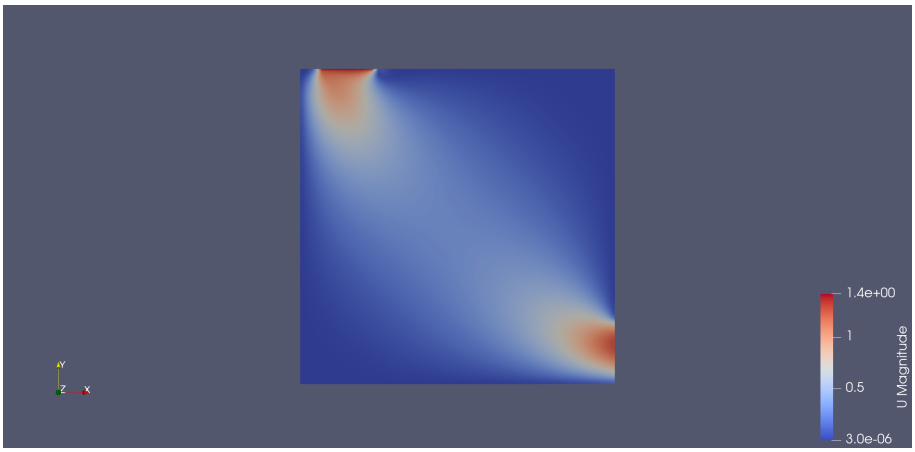

Figure 4.21: Resulting flow for the best configuration found by `ncsu_direct` algorithm with sampling in the centre of the domain

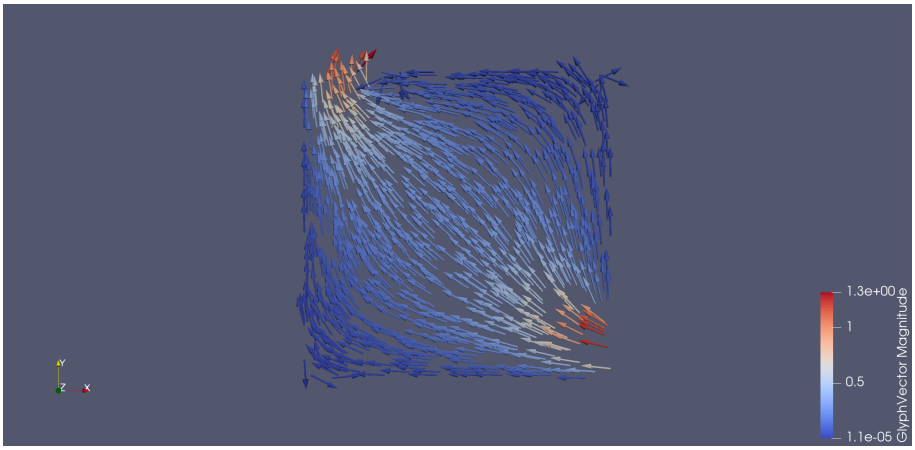

Figure 4.22: Vectorial representation of the flow obtained in the best configuration found by the algorithm ncsu_direct and sampling in the centre of the domain

The evolutionary algorithm `coliny_ea`.

```
<<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
<<<<< Best parameters          =
                     2.2834525605e+00 teta1
                     5.6506142615e+00 teta2
<<<<< Best objective function  =
                     9.8930500000e-01
<<<<< Best data captured at function evaluation 8


<<<<< Iterator coliny_ea completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    1.62836 [parent =    1.62836, child =      1e-06]
  Total wall clock =    6819.88
Exit graphics window to terminate DAKOTA.
```

Figure 4.23: Summary of the `coliny_ea algorithm for optimization in the centre of the domain`

The best uniformity value found is **0,9893**. It is found in function evaluation number **8**. The values of the two variables in this configuration are **teta1** equal to **2,283** and **teta2** equal to **5,651**.

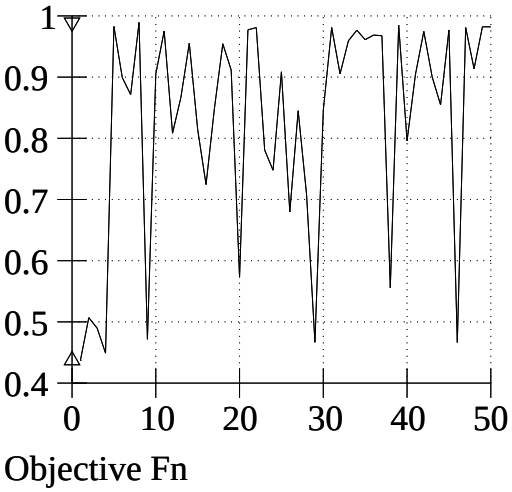

Figure 4.24: Values of the objective function obtained from the algorithm coliny_ea

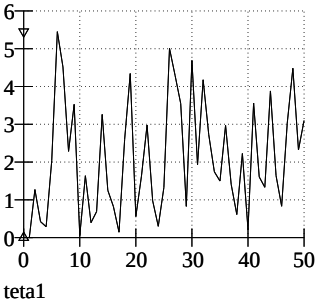

Figure 4.25: Values of the variable of the position of the inlet

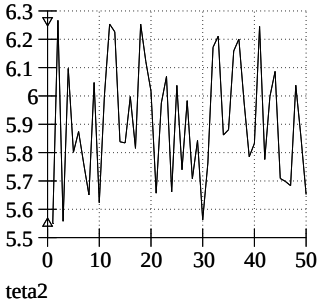

Figure 4.26: Values of the variable of the position of the outlet

The inlet and outlet position for the iteration number 8 and the relative flow:

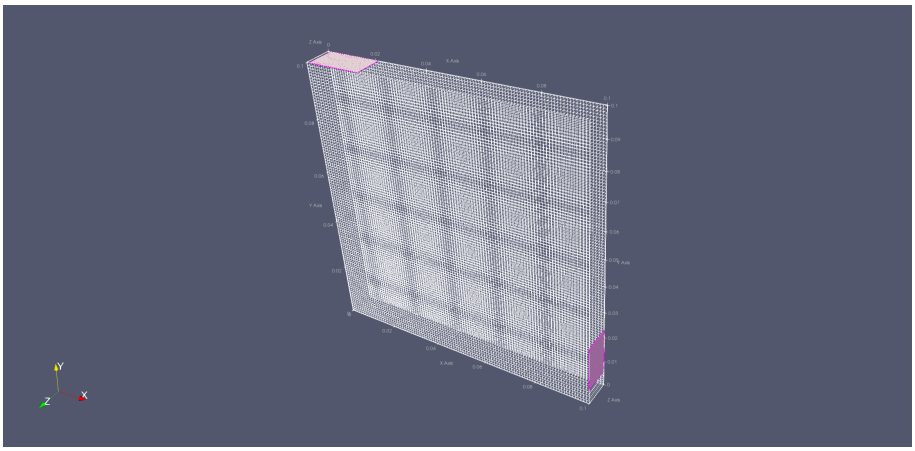

Figure 4.27: Inlet and outlet position for the best configuration found by `coliny_ea algorithm`

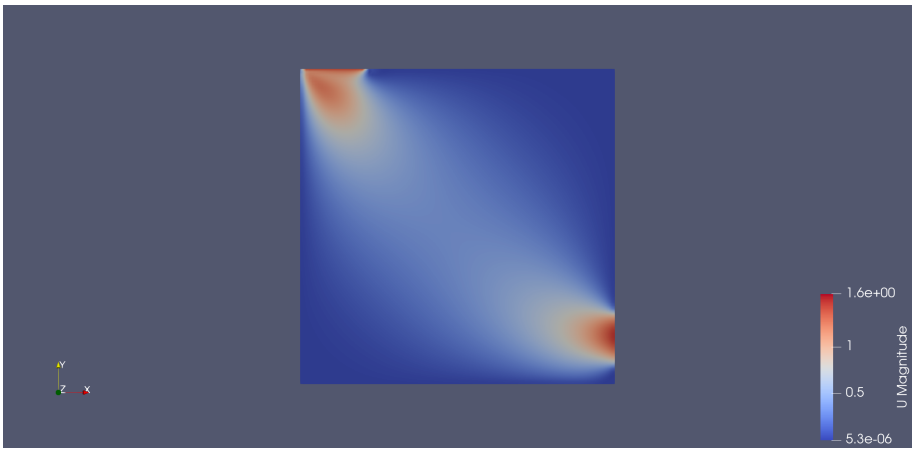

Figure 4.28: Resulting flow for the best configuration found by `coliny_ea algorithm`

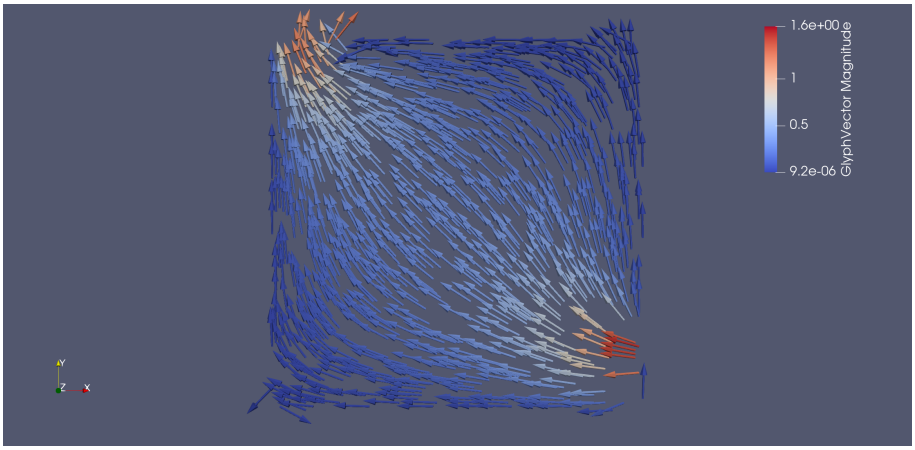

Figure 4.29: Vectorial representation of the flow obtained in the best configuration found by the algorithm coliny_ea and sampling in the centre of the domain

Lower Left Regional Optimization

In this second case the sampling script is changed so that the sampling area is located in (0 0) with an extension of 0.01 in each direction. The exact sampling area is shown next (in the **y** direction). The same center of the plane with a (1 0 0) normal vector and same extension is the sampling area in the **x** direction, it is not shown because it is essentially the same, just rotated.

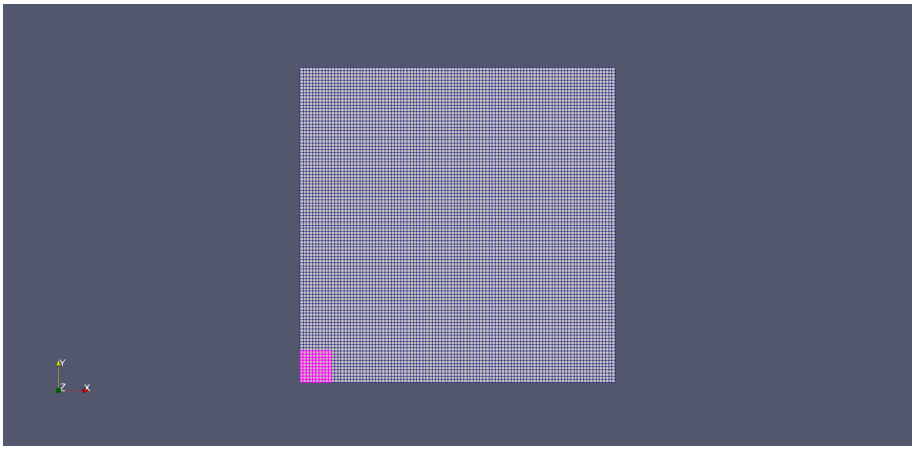

Figure 4.30: Lower left sampling area

The first solver, as before, will be `coliny_ direct`, here is the brief

conclusion of the results obtained.

```
<<<<< Function evaluation summary: 59 total (59 new, 0 duplicate)
<<<<< Best parameters          =
                     2.5542592593e+00 teta1
                     5.5900000000e+00 teta2
<<<<< Best objective function  =
                     8.1755500000e-01
<<<<< Best data captured at function evaluation 50


<<<<< Iterator coliny_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    4.66305 [parent =    4.66305, child =      1e-06]
  Total wall clock =    19673.5
Exit graphics window to terminate DAKOTA.
Signal Caught!
```

The optimal result in this case is found in the iteration number 50, it corresponds to a uniformity value equal to **0.817555**. The following graphs summarize the whole algorithm process.

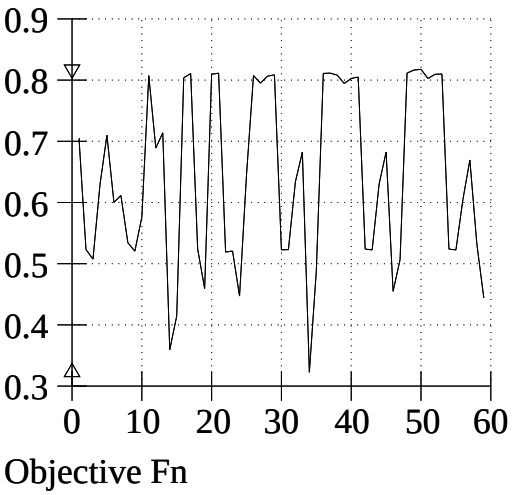

Figure 4.31: Values of the objective function obtained from the algorithm coliny_direct and sampling in the lower left of the domain

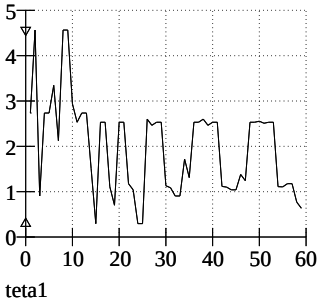

Figure 4.32: Values of the variable of the position of the inlet

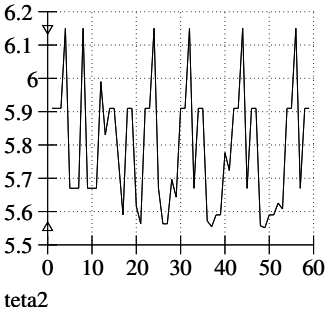

Figure 4.33: Values of the variable of the position of the outlet

The inlet and outlet optimal positions as evaluated by the `coliny_direct` algorithm are shown next.

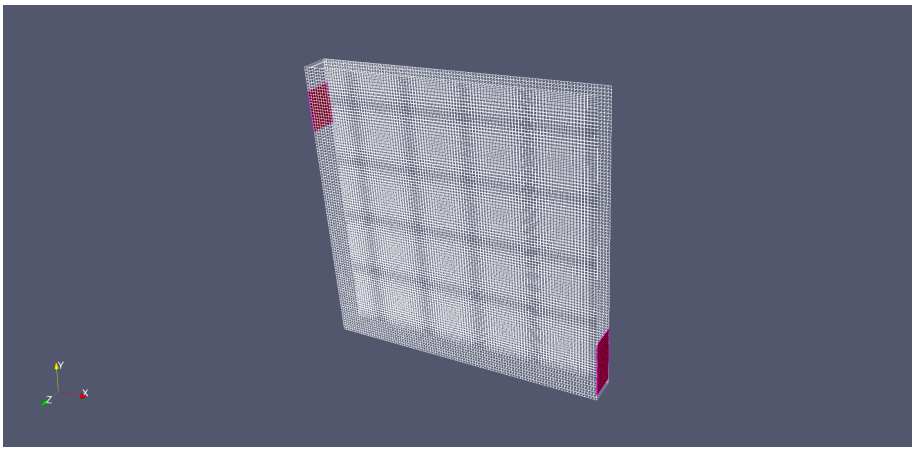

Figure 4.34: Inlet and outlet optimal positions found with algorithm coliny_direct and sampling in lower left of the domain

The resulting flow is:

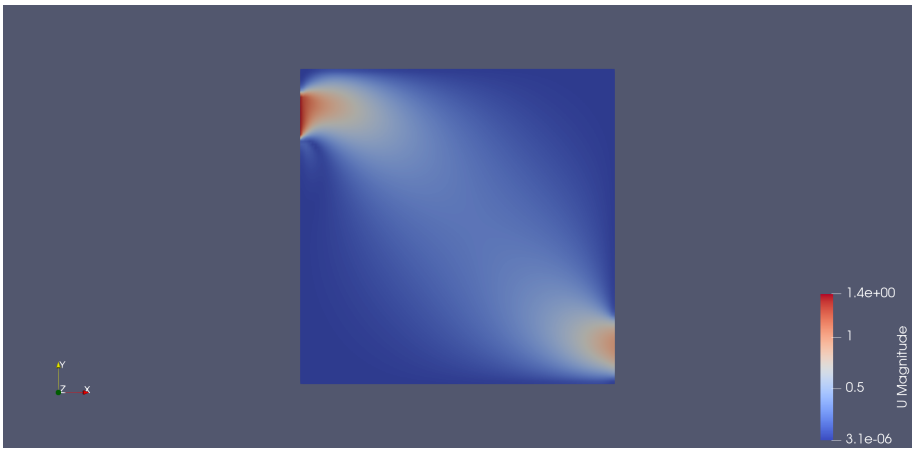

Figure 4.35: Resulting flow in optimal configuration found with algorithm coliny_direct and sampling in lower left of the domain

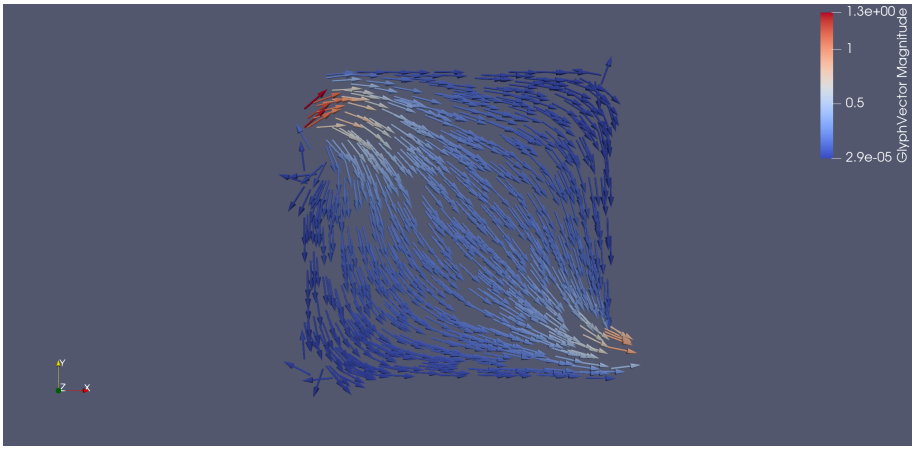

Figure 4.36: Vectorial representation of the flow obtained in the best configuration found by the algorithm coliny_direct and sampling in the lower left part of the domain

The second algorithm is tested next with the same sampling area in the lower left corner. The `genie_direct` algorithm, again, works by finding the minimum of an objective function, hence the inverted function will be the response function fed to the Dakota software. The results of the optimization are:

```
<<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
<<<<< Best parameters          =
                     3.8872222222e+00 teta1
                     5.6700000000e+00 teta2
<<<<< Best objective function  =
                     1.0002700000e+00
<<<<< Best data captured at function evaluation 44


<<<<< Iterator genie_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    2.13883 [parent =    2.13883, child =          0]
  Total wall clock =    15755.1
Exit graphics window to terminate DAKOTA.
```

Figure 4.37: Summary of the results obtained from the algorithm genie_direct

This algorithm manages to find a better result in the defined number of

iterations. The highest objective function value found is **1.00027** that corresponds to a uniformity value of **0,999730073**, which means that the `genie_direct` algorithm does a much better job with the same sampling area. Since the results greatly differ from one another, we expect also to see a great difference in the inletoutlet configuration.

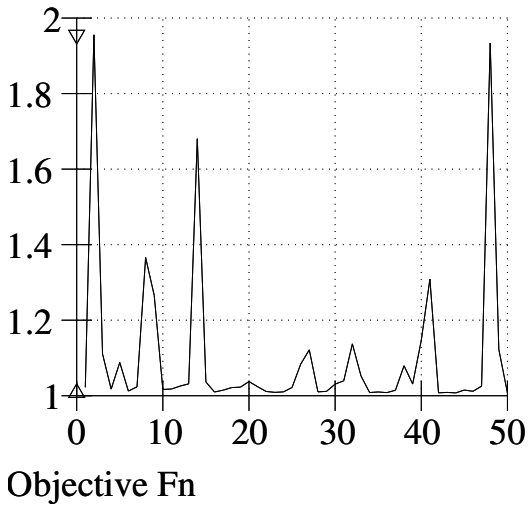

Figure 4.38: Values of the objective function obtained from the algorithm genie_direct

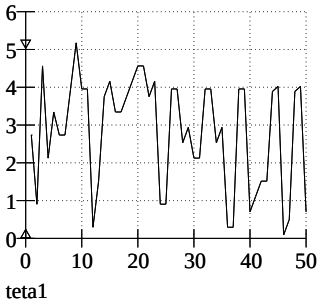

Figure 4.39: Values of the variable of the position of the inlet

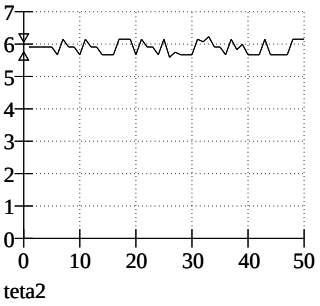

Figure 4.40: Values of the variable of the position of the outlet

The inlet and outlet optimal positions as evaluated by the `genie_direct` algorithm are shown next.

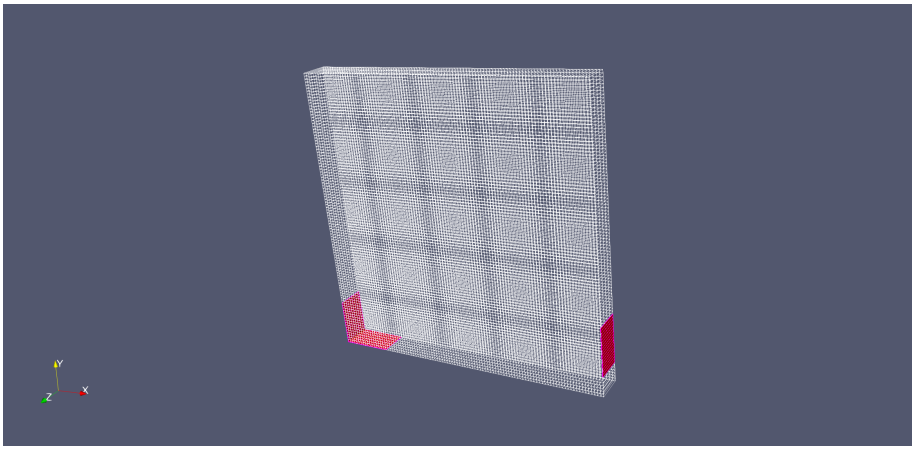

Figure 4.41: Inlet and outlet optimal positions found with algorithm genie_direct and sampling in lower left of the domain

The resulting flow is:

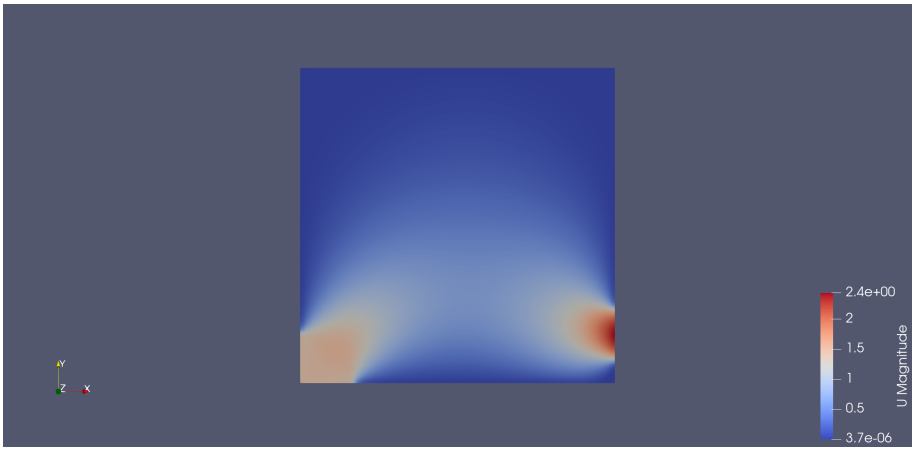

Figure 4.42: Resulting flow in optimal configuration found with algorithm genie_direct and sampling in lower left of the domain

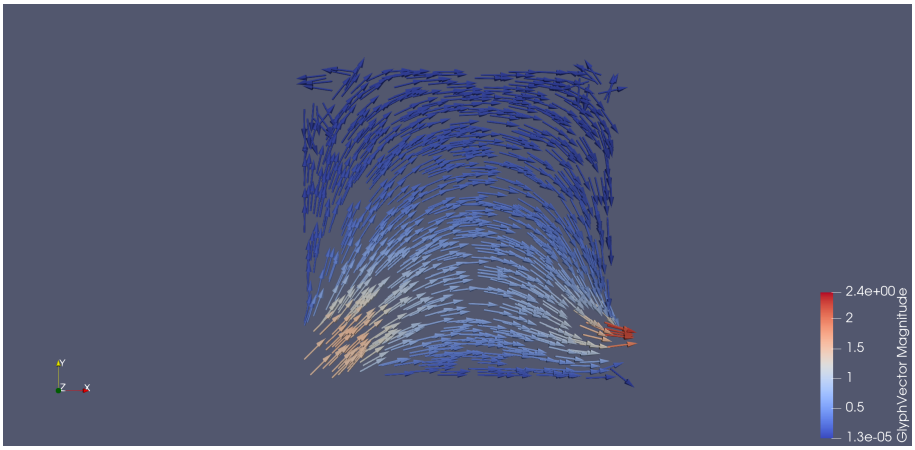

Figure 4.43: Vectorial representation of the flow obtained in the best configuration found by the algorithm genie_direct and sampling in the lower left part of the domain

It can be seen that this last configuration is radically different from the previous ones. If the iterations are increased to 200 for the `coliny_direct` algorithm, the results do not change much, the best uniformity value found in this case is **0,8178**. The `ncsu_direct` algorithm follows.

```
NCSU DIRECT succeeded with code 1
(maximum function evaluations exceeded)
<<<<< Function evaluation summary: 55 total (55 new, 0 duplicate)
<<<<< Best parameters          =
                     2.5768518519e+00 teta1
                     5.7351851852e+00 teta2
<<<<< Best objective function  =
                     8.1734000000e-01
<<<<< Best data captured at function evaluation 54


<<<<< Iterator ncsu_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    1.53269 [parent =     1.5327, child =     -2e-06]
  Total wall clock =    6434.78
Exit graphics window to terminate DAKOTA.
```

Figure 4.44: Summary of the `ncsu_direct algorithm`

The best uniformity value found is **0,8173**. It is found in function evaluation number **54**. The values of the two variables in this configuration are **teta1** equal to **2,576** and **teta2** equal to **5,735**.

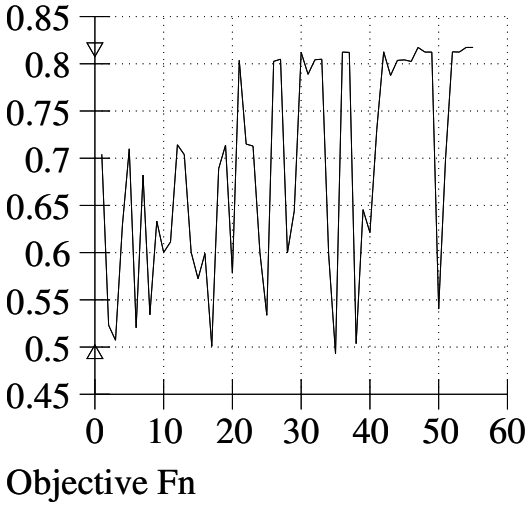

Figure 4.45: Values of the objective function obtained from the algorithm ncsu_direct

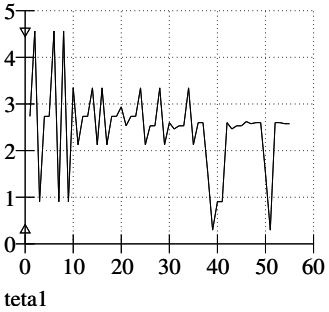

Figure 4.46: Values of the variable of the position of the inlet

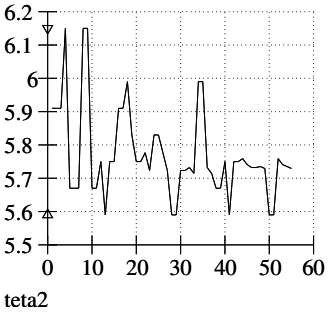

Figure 4.47: Values of the variable of the position of the outlet

The inlet and outlet position for the 54th iteration and the relative flow:

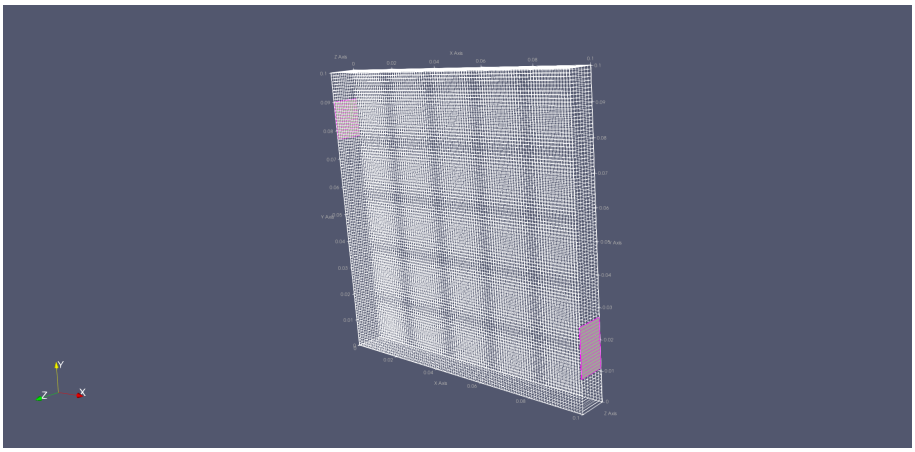

Figure 4.48: Inlet and outlet position for the best configuration found by `ncsu_direct algorithm`

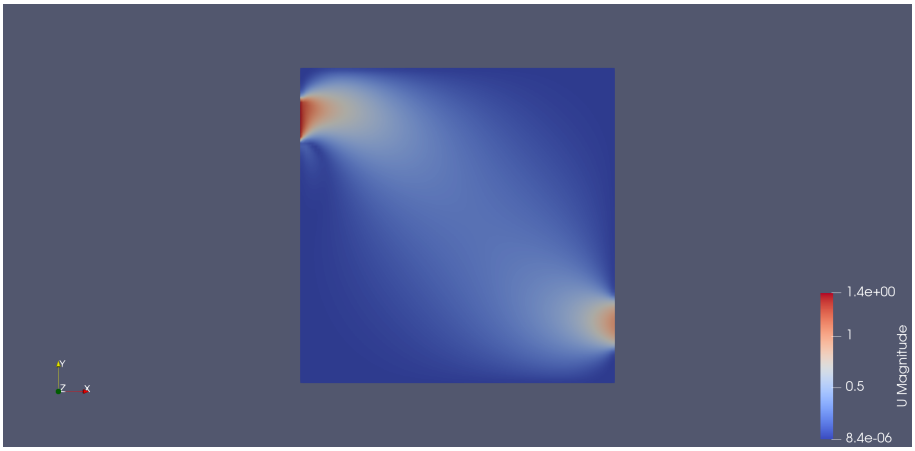

Figure 4.49: Resulting flow for the best configuration found by `ncsu_direct algorithm`

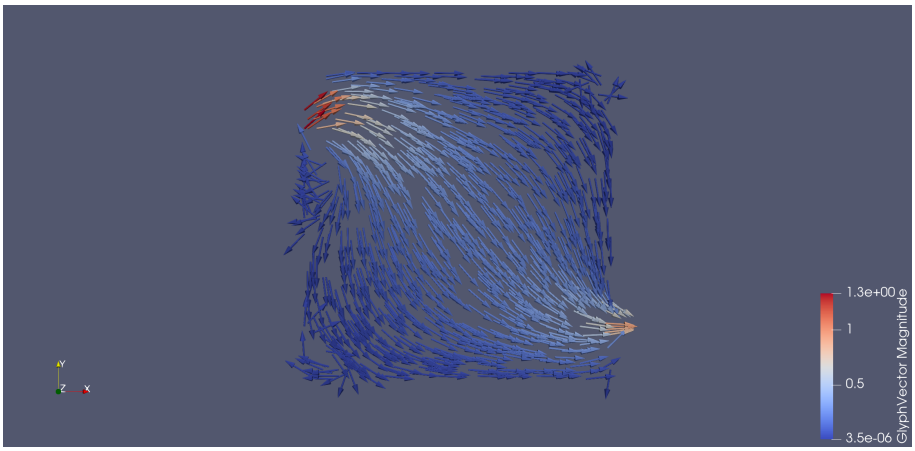

Figure 4.50: Vectorial representation of the flow obtained in the best configuration found by the algorithm ncsu_direct and sampling in the lower left part of the domain

The evolutionary algorithm `coliny_ea`.

```
<<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
<<<<< Best parameters          =
                     2.4411581643e+00 teta1
                     6.2247892739e+00 teta2
<<<<< Best objective function  =
                     8.0823000000e-01
<<<<< Best data captured at function evaluation 39


<<<<< Iterator coliny_ea completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    1.62536 [parent =    1.62536, child =     -2e-06]
  Total wall clock =    6354.12
Exit graphics window to terminate DAKOTA.
```

Figure 4.51: Summary of the `coliny_ea algorithm`

The best uniformity value found is **0,8173**. It is found in function evaluation number **54**. The values of the two variables in this configuration are **teta1** equal to **2,576** and **teta2** equal to **5,735**.

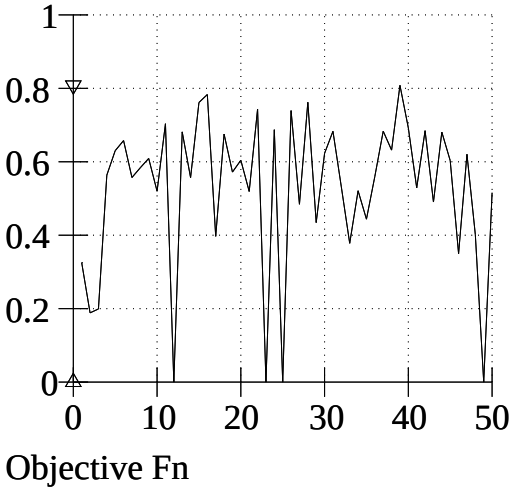

Figure 4.52: Values of the objective function obtained from the algorithm coliny_ea

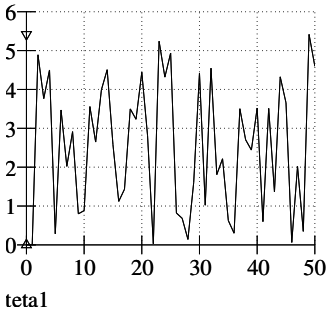

Figure 4.53: Values of the variable of the position of the inlet

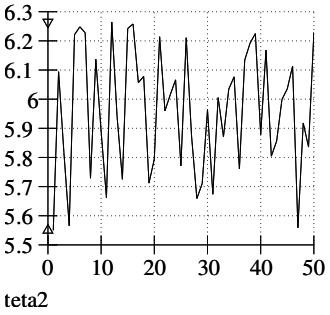

Figure 4.54: Values of the variable of the position of the outlet

The inlet and outlet position for the 54th iteration and the relative flow:

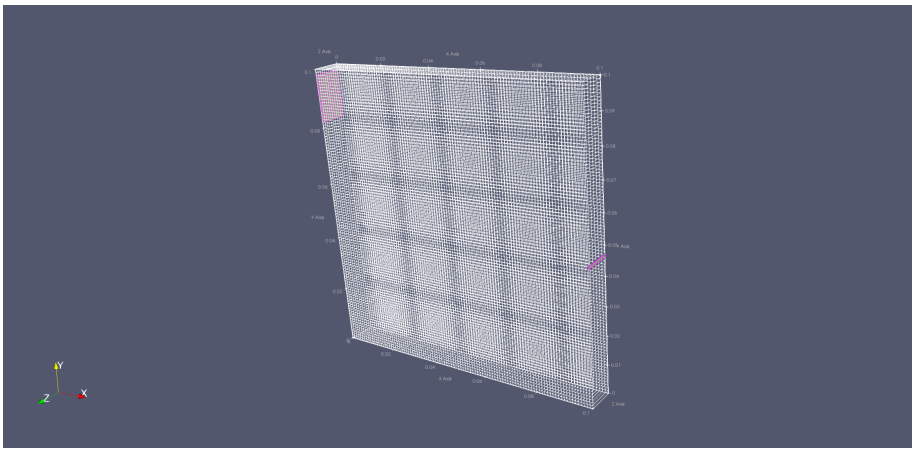

Figure 4.55: Inlet and outlet position for the best configuration found by `coliny_ea algorithm`

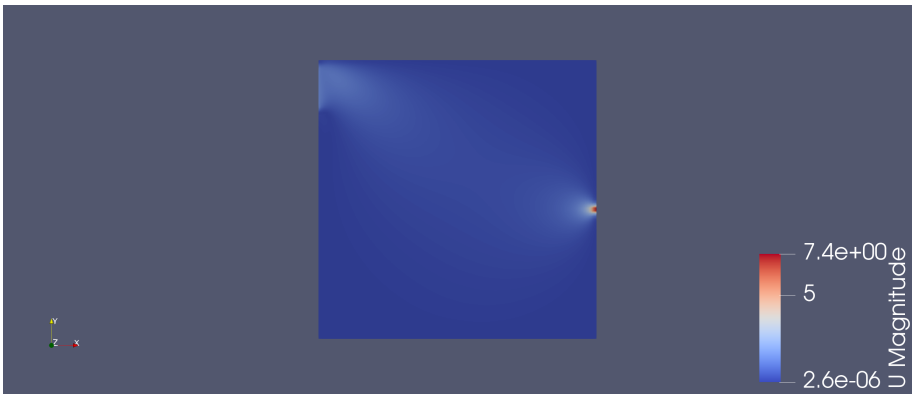

Figure 4.56: Resulting flow for the best configuration found by `coliny_ea algorithm`

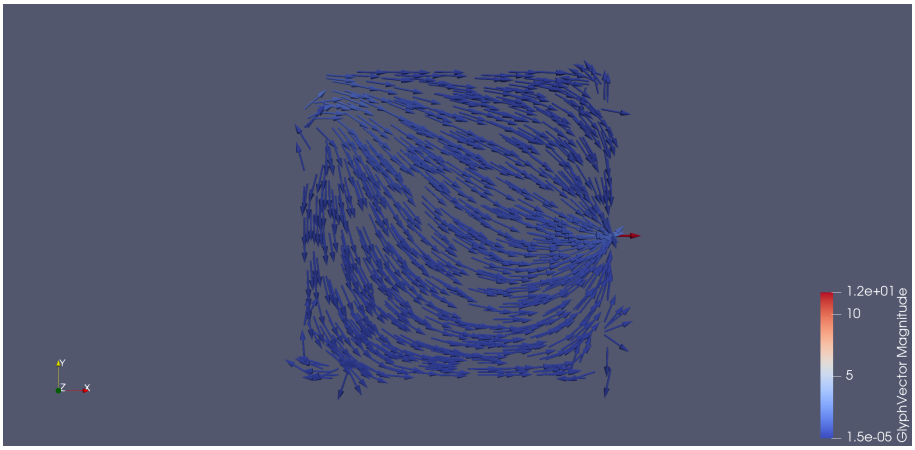

Figure 4.57: Vectorial representation of the flow obtained in the best configuration found by the algorithm coliny_ea and sampling in the lower left part of the domain

4.1.2 Global Domain Optimization

The final test case that was launched aimed to have a uniform flow considering the whole domain. To do this multiple sampling planes were set up in the sampling file. In this case **10 planes** for the x direction and **10 planes** for the y direction were added as sampling locations. Each plane is extended through the whole domain. The final **uniformity** value is found as the **mean value** between the uniformity values of all the planes.

The first solver that will be launched is **coliny_direct**. A brief conclusion of the optimization results is found at the bottom of the run.out file.

```
<<<<< Function evaluation summary: 53 total (53 new, 0 duplicate)
<<<<< Best parameters          =
                     2.2229012346e+00 teta1
                     5.6700000000e+00 teta2
<<<<< Best objective function  =
                     5.7433000000e-01
<<<<< Best data captured at function evaluation 22


<<<<< Iterator coliny_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =   0.675945 [parent =   0.675946, child =     -1e-06]
  Total wall clock =    2081.83
Exit graphics window to terminate DAKOTA.
Signal Caught!
```

Figure 4.58: Summary of the results obtained with `coliny_direct` algorithm and sampling in the whole domain

We can see that the **22nd** iteration was the most successful one for the global domain optimization case. It may be that this same result is obtained through various configurations similar to this one. This can be avoided with an increased precision in the solving process or with a finer mesh.

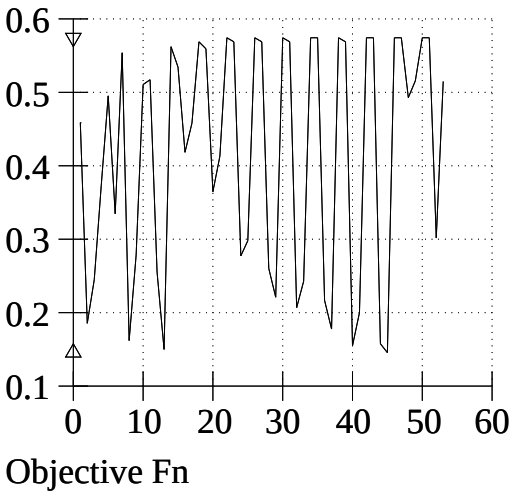

Figure 4.59: Values of the objective function obtained from the algorithm coliny_direct

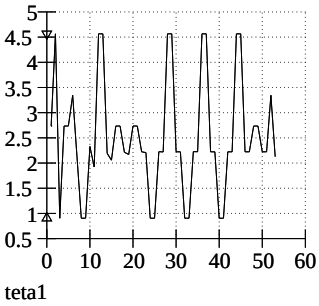

Figure 4.60: Values of the variable of the position of the inlet

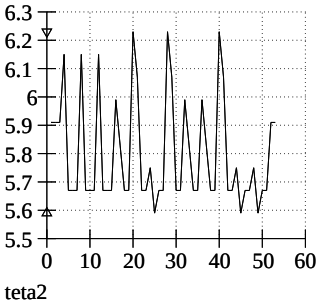

Figure 4.61: Values of the variable of the position of the outlet

The inlet and outlet positions in this cofiguration are shown next, the

inlet is on the top of the domain while the outlet is on the lower right of the domain. The configuration of the cases that bear same solution is almost the same as this one.

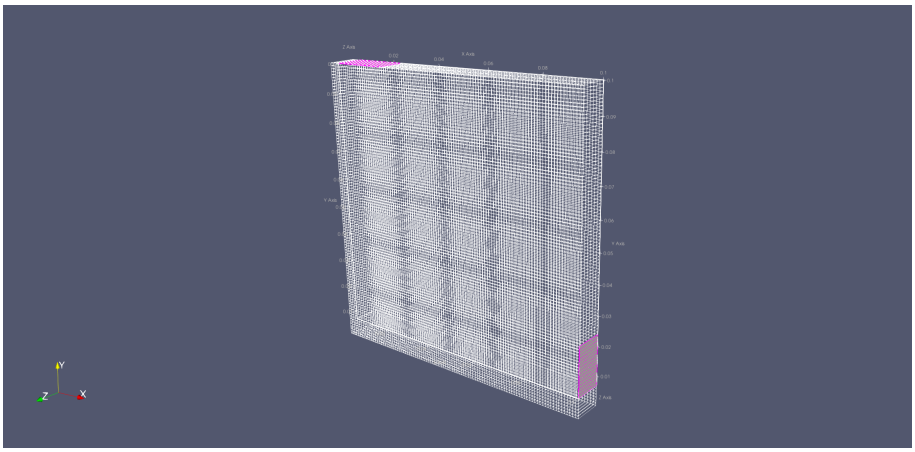

Figure 4.62: Inlet and outlet optimal positions found with algorithm `coliny_direct` and sampling in the whole domain

The **uniformity** value in this configuration is **0,57433**. The resulting flow is represented next.

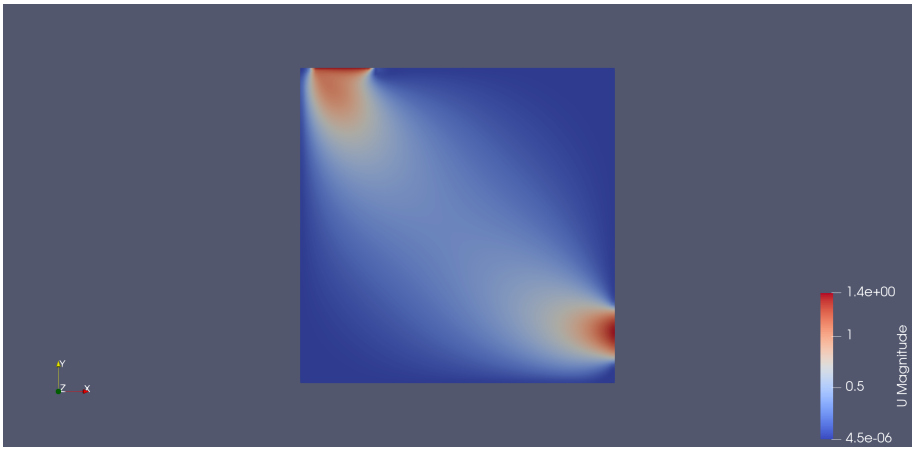

Figure 4.63: Resulting flow from the best configuration found with algorithm coliny_direct and sampling in the whole domain

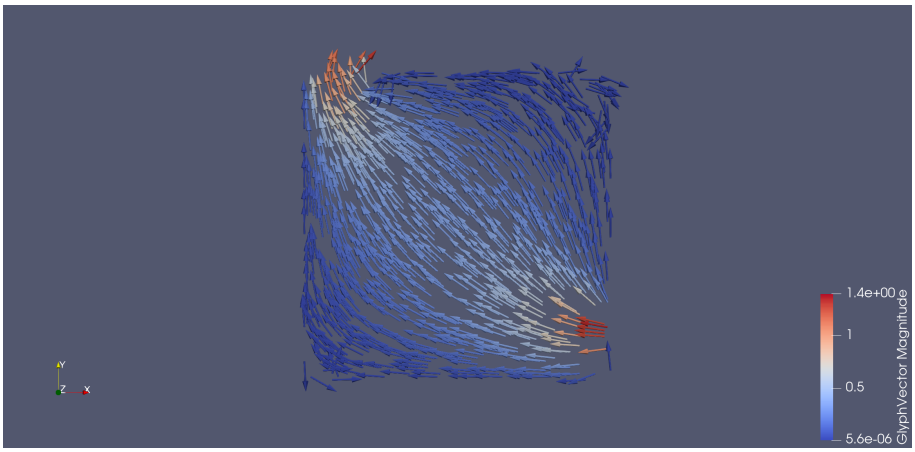

Figure 4.64: Vectorial representation of the flow obtained in the best configuration found by the algorithm coliny_direct and sampling in the whole domain

The `genie_direct` algorithm is tested next with the same sampling area on the whole domain. The inverse function is the objective function, this algorithm works, as before, by minimizing instead of finding a maximum. The results of the optimization are:

```
83 <<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
84 <<<<< Best parameters          =
85                      2.2153703704e+00 teta1
86                      5.6166666667e+00 teta2
87 <<<<< Best objective function  =
88                      1.6889000000e+00
89 <<<<< Best data captured at function evaluation 37
90
91
92 <<<<< Iterator genie_direct completed.
93 <<<<< Environment execution completed.
94 DAKOTA execution time in seconds:
95   Total CPU        =    2.33511 [parent =    2.33511, child =      1e-06]
96   Total wall clock =    15601.8
97 Exit graphics window to terminate DAKOTA.
```

Figure 4.65: Summary of the results obtained with `coliny_direct` algorithm and sampling in the whole domain

Again this algorithm manages to find a better result in the defined number of iterations. The highest objective function value found is **1.6889** that corresponds to a uniformity value of **0,5921**. The results are similar though, so we don't expect to see a big difference in the "best" configuration found by both algorithms.

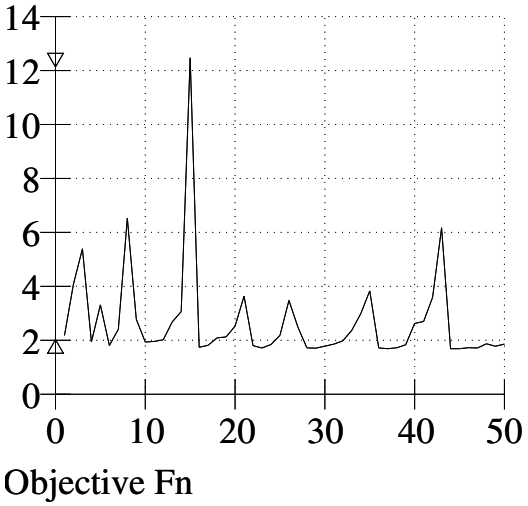

Figure 4.66: Values of the objective function obtained from the algorithm genie_direct

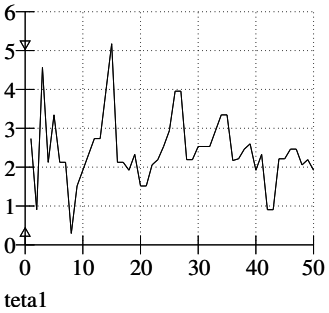

Figure 4.67: Values of the variable of the position of the inlet

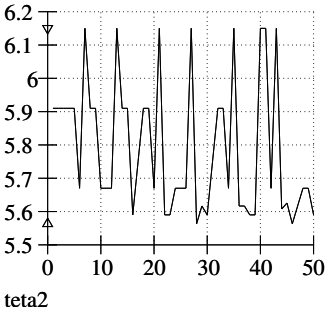

Figure 4.68: Values of the variable of the position of the outlet

The inlet and outlet optimal positions as evaluated by the `genie_direct` algorithm are shown next.

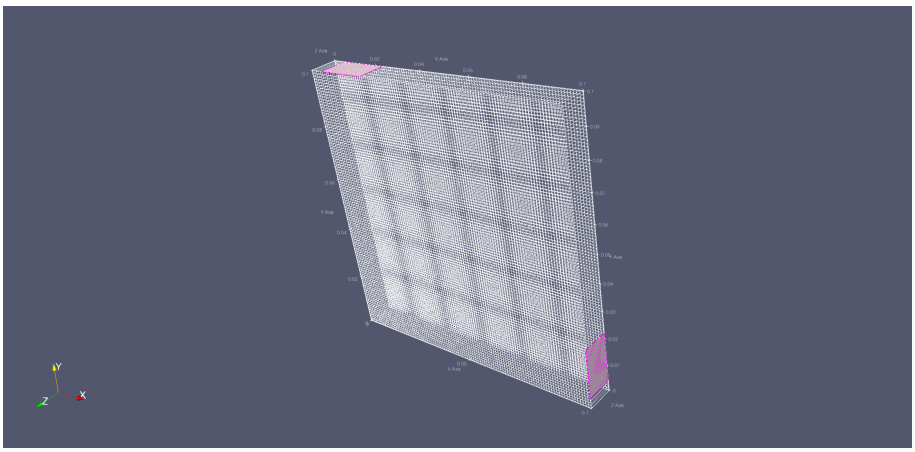

Figure 4.69: Inlet and outlet optimal positions found with algorithm `genie_direct` and sampling in the whole domain

The resulting flow is:

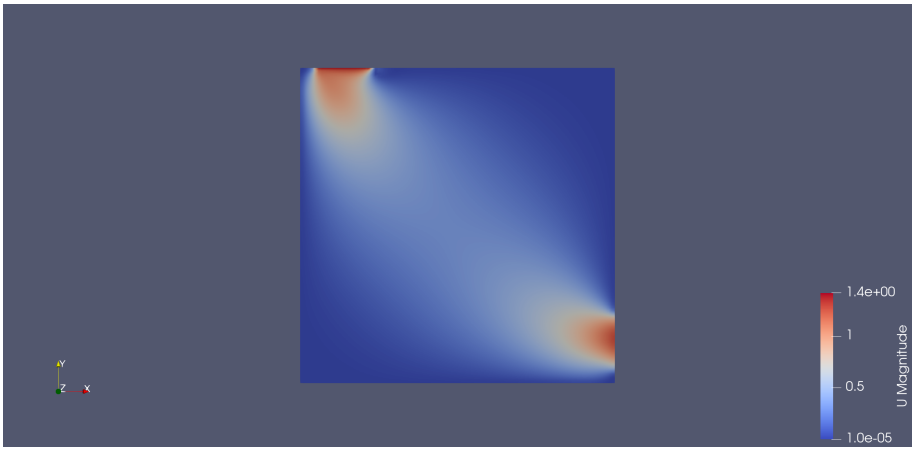

Figure 4.70: Resulting flow from the best configuration found with algorithm genie_direct and sampling in the whole domain

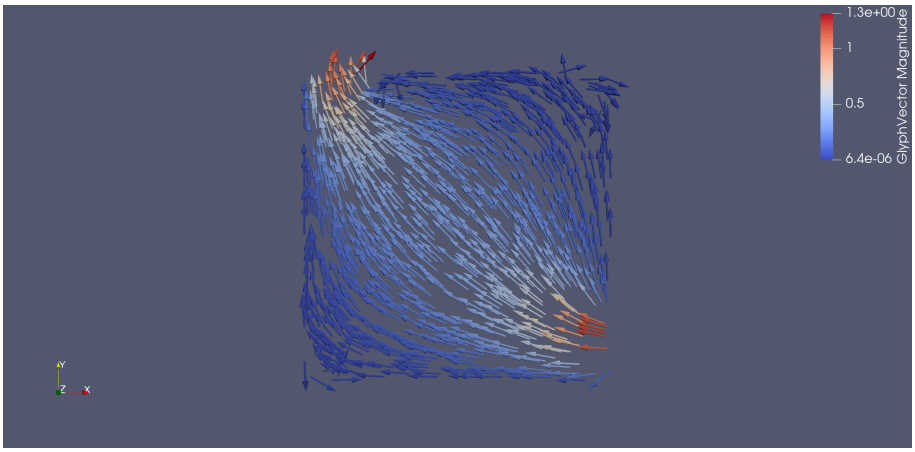

Figure 4.71: Vectorial representation of the flow obtained in the best configuration found by the algorithm genie_direct and sampling in the whole domain

This last configuration is almost the same as the previous one with the `coliny_direct` algorithm which means either that a higher value is nowhere to be found or that both algorithms need more iterations to find a better value. The `ncsu_direct` algorithm follows.

```
NCSU DIRECT succeeded with code 1
(maximum function evaluations exceeded)
<<<<< Function evaluation summary: 53 total (53 new, 0 duplicate)
<<<<< Best parameters          =
                     2.2153703704e+00 teta1
                     5.6255555556e+00 teta2
<<<<< Best objective function  =
                     5.9209500000e-01
<<<<< Best data captured at function evaluation 50


<<<<< Iterator ncsu_direct completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    5.39437 [parent =    5.39436, child =      1e-06]
  Total wall clock =    34171.5
Exit graphics window to terminate DAKOTA.
Signal Caught!
```

Figure 4.72: Summary of the `ncsu_direct algorithm`

The best uniformity value found is **0,8173**. It is found in function evaluation number **54**. The values of the two variables in this configuration are **teta1** equal to **2,576** and **teta2** equal to **5,735**.

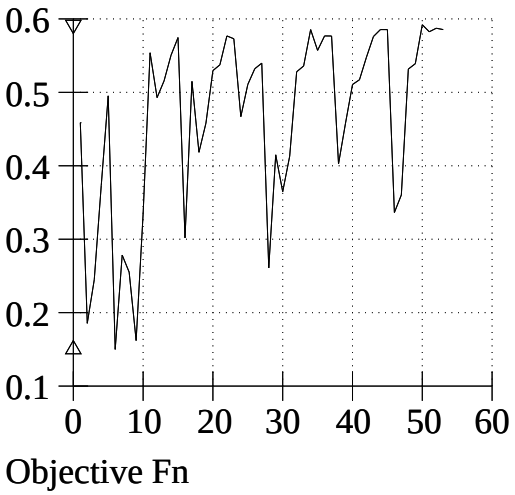

Figure 4.73: Values of the objective function obtained from the algorithm ncsu_direct

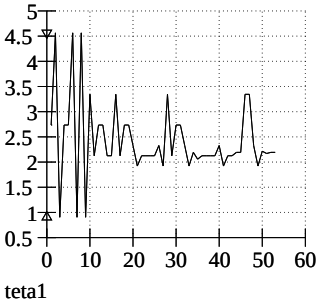

Figure 4.74: Values of the variable of the position of the inlet

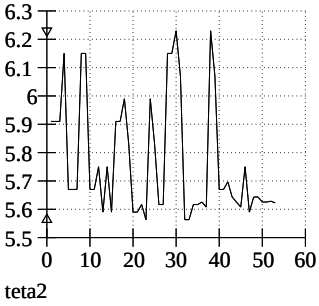

Figure 4.75: Values of the variable of the position of the outlet

The inlet and outlet position for the 54th iteration and the relative flow:

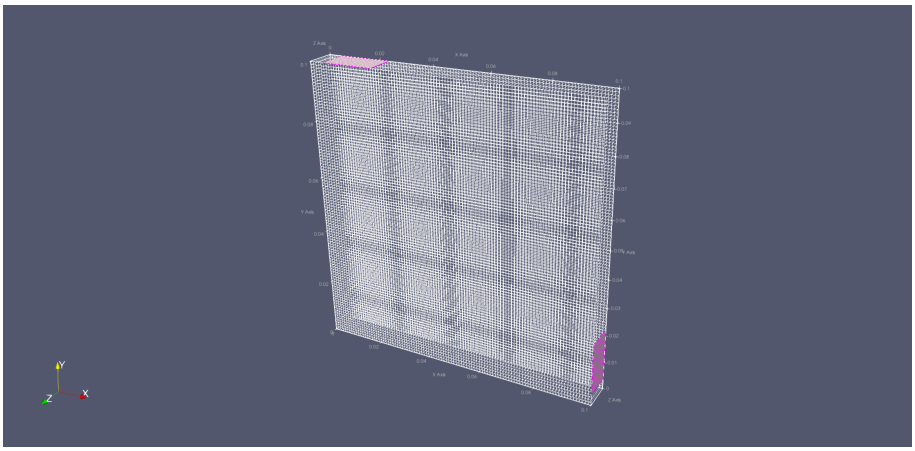

Figure 4.76: Inlet and outlet position for the best configuration found by `ncsu_direct algorithm`

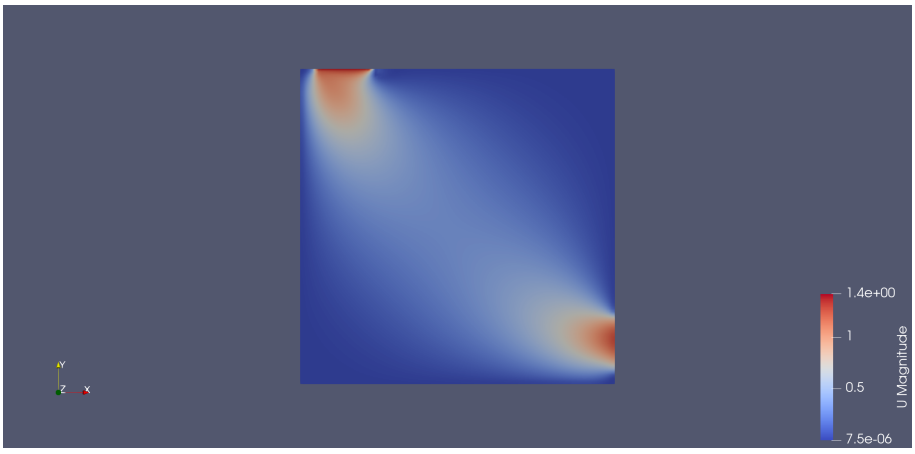

Figure 4.77: Resulting flow for the best configuration found by `ncsu_direct algorithm`

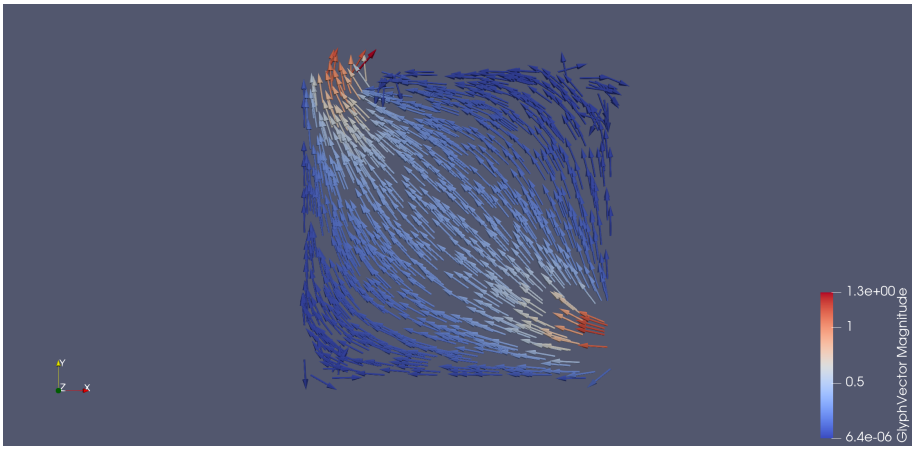

Figure 4.78: Vectorial representation of the flow obtained in the best configuration found by the algorithm ncsu_direct and sampling in the whole domain

The evolutionary algorithm `coliny_ea`.

```
<<<<< Function evaluation summary: 50 total (50 new, 0 duplicate)
<<<<< Best parameters          =
                     1.3475553910e+00 teta1
                     6.2691419033e+00 teta2
<<<<< Best objective function  =
                     5.6369000000e-01
<<<<< Best data captured at function evaluation 15


<<<<< Iterator coliny_ea completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =    2.24889 [parent =    2.24889, child =     -1e-06]
  Total wall clock =    8942.32
Exit graphics window to terminate DAKOTA.
Signal Caught!
```

Figure 4.79: Summary of the `coliny_ea` algorithm

The best uniformity value found is **0,8173**. It is found in function evaluation number **54**. The values of the two variables in this configuration are **teta1** equal to **2,576** and **teta2** equal to **5,735**.

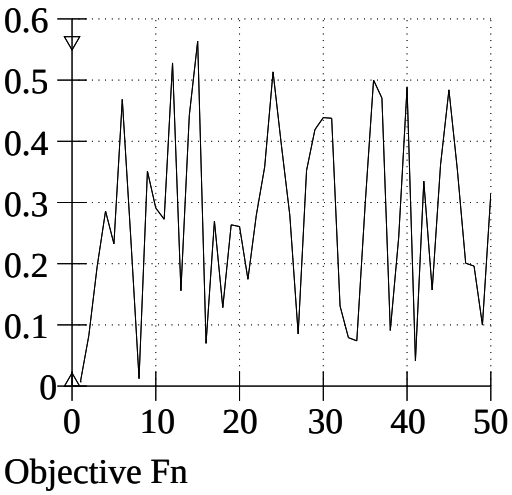

Figure 4.80: Values of the objective function obtained from the algorithm `coliny_ea`

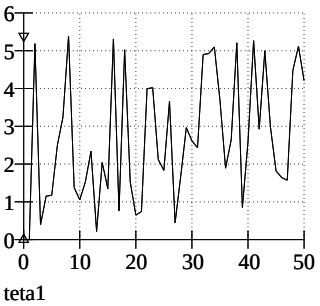

Figure 4.81: Values of the variable of the position of the inlet

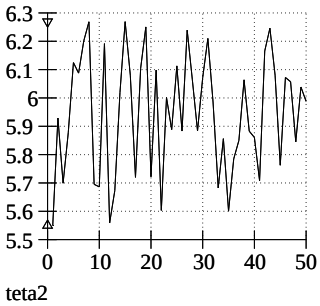

Figure 4.82: Values of the variable of the position of the outlet

Chapter 5

Summary and outlook

The starting point of this work was to think of a possible optimization problem to be studied by the software OpenFOAM from the fluiddynamic point of view and the software Dakota from the optimization point of view. After the theoretical problem was designed a fitting baseline was found in the cavity tutorial case. This starting point provided the fluiddynamic component of the problem, indeed very few changes were made to the cavity tutorial case domain and solving process since it is mostly the same as the desired baseline. The addition of an inlet and an outlet to the mesh domain are the most notable changes made to the tutorial case. The time-step of the solving process has been reduced to achieve convergence as well as a more accurate solving process has been used to increase precision, although not too much to slow down excessively the solving process. The finer mesh also allowed to understand better the final result. Most importantly a post-processing operation was added to allow the following steps of optimization, indeed OpenFOAM was tasked to find the unweighted uniformity index value of the present configuration launched.

After the fluiddynamic step was developed, the optimization component had to be examined. The Dakota software was studied extensively, its optimization algorithms and its interaction with outsourced solvers. The theoretical approach as well as the practical approach of optimization and optimization within Dakota was studied. At this point the fluiddynamic model was linked to the iterative one which means that the iterative variables have been defined as the position of the inlet and of the outlet,

and the fork keyword calls out OpenFOAM to solve each iteration. Most importantly the objective of the optimization has been defined as finding the maximum unweighted uniformity index. At this point the working case was ready for testing, the iterative software fed each time the position of the inlet and outlet to the fluiddynamic case. OpenFOAM solved each case while evaluating the uniformity index and Dakota, depending on the algorithm chosen would find a new set of variables to try to increase the previous value of the index.

Different iterative algortihms and also different sampling locations inside the domain have been tested. The results told us that different sampling locations yield similar results depending on the algorithm chosen. A difference in the position of the inlet was found between optimization in the center of the domain and the optimization in the lower left part of the domain although some algorithm would find such difference while others would not, at least not in the small number of iterations chosen for this cases. It is important to say that there is no best optimization algorithm but, for these specific cases, the `genie_direct` algorithm has been found to yield the best results while the `coliny_direct` algorithm to yield the fastest results. It is nevertheless important to say that such a small number of iteration tells us nothing conclusive apart from the confirmation that such an undertaking can indeed be solved in this way as well as any optimization problem regarding fluiddynamics. These are but few examples that prove the effectiveness and relative simplicity of this optimization procedure. Here, a numerical goal based on the velocity uniformity value has been chosen but any variable, or combination of variables can be fed to Dakota as an objective function in an optimization problem.

Conclusively the following step in this particular case could be asking how a substance in the air diffuses in a twodimensional (or even threedimensional) box (or any domain). How to reach a higher uniformity value of the composition of the air inside that domain after the injection of a gaseous substance. The results could be studied further by adding another inlet, another outlet, or both. The inlet velocity of the gaseous substance could be changed and the inlet could be developed to yield

higher diffusion with a spray-like inlet. The most important conclusion to draw here is the reach and endless possibilities of this approach, outside the value of any single theoretical case, in real world applications.

Bibliography

Adams, B. M., Hough, P. D., & Swiler, L. P. (2015). *Dakota software training: Model calibration.* (Tech. Rep.). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ....

Dakota, A. (2009). *Multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification and sensitivity analysis: Version 6.1 user's manual.* Livermore: Sandia National Laboratories.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical programming*, *91*(2), 201–213.

Ferziger, J. H., Perić, M., & Street, R. L. (2002). *Computational methods for fluid dynamics* (Vol. 3). Springer.

F. Piscaglia, A. M. (2019). *Computational techniques for thermochemical propulsion.*

Greenshields, C. J. (2017). Openfoam user guide. version 6. *OpenFOAM Foundation Ltd July*.

kan Nilsson, H. (2013). A look inside icofoam (and pisofoam). *MSc/PhD course in CFD with OpenSource software.*

Rao, S. S. (2019). *Engineering optimization: theory and practice.* John Wiley & Sons.

Venkataraman, P. (2009). *Applied optimization with matlab programming.* John Wiley & Sons.

Weller, H. G., Tabor, G., Jasak, H., & Fureby, C. (1998). A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, *12*(6), 620–631.