# Application of Genetic Algorithms to Problems in Computational Fluid Dynamics

Submitted by Björn Fabritius to the University of Exeter
as a thesis for the degree of
Doctor of Philosophy in Engineering
January 2014

Signature: .

# Acknowledgements

This work would not have been possible without the help and support of all my friends and family. Thanks to everyone for seeing this through to the end and trusting in my abilities more than I did myself. A special note of appreciation to Gavin Tabor whose ideas, enthusiasm and encouragement made this research worthwhile and even enjoyable. My brother Ingo for teaching me to believe in myself. But most of all my parents Hubert and Monika without whose continuous support and encouragement I would not be where I am today. They taught me that curiosity and an open mind are the key to the universe.

The research was funded by the University of Exeter, of which I am very grateful.

München, 31. January 2014

# Abstract

In this thesis a methodology is presented to optimise non–linear mathematical models in numerical engineering applications. The method is based on biological evolution and uses known concepts of genetic algorithms and evolutionary computation. The working principle is explained in detail, the implementation is outlined and alternative approaches are mentioned. The optimisation is then tested on a series of benchmark cases to prove its validity. It is then applied to two different types of problems in computational engineering.

The first application is the mathematical modeling of turbulence. An overview of existing turbulence models is followed by a series of tests of different models applied to various types of flows. In this thesis the optimisation method is used to find improved coefficient values for the k–$\varepsilon$, the k–$\omega$-SST and the Spalart–Allmaras models. In a second application optimisation is used to improve the quality of a computational mesh automatically generated by a third party software tool. This generation can be controlled by a set of parameters, which are subject to the optimisation.

The results obtained in this work show an improvement when compared to non–optimised results. While computationally expensive, the genetic optimisation method can still be used in engineering applications to tune predefined settings with the aim to produce results of higher quality. The implementation is modular and allows for further extensions and modifactions for future applications.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

*If we knew what it was we were doing, it would not be called research, would it?*

Albert Einstein

## 1.1. Background

The developments in modelling and understanding the mathematics of physical processes is accompanied by a constant improvement and expansion of numerical methods and computer tools to simulate these processes. Although the basic principals of fluid mechanics are mathematically well understood, an accurate numerical treatment is difficult even with modern high performance computer systems. While new models are emerging to describe more and more complex processes like reactive flow, moving boundaries or multi-physics systems, many equations and models that have been around for decades are still in use. On the one hand these particular models have proven their worth in multiple applications and are well tested and documented, but on the other hand assumptions that were made in the derivation of the model equations might not always be fitting to a specific problem and by exploiting computational resources that were not available several years ago, accuracy and applicability of the "old" models might still be improved.

An example for a family of models describing the same physical effect in a variety of applications with different assumptions and differing levels of abstraction are turbulence closure models. The earliest of these are very simple, reflecting the limited possibilities to solve large systems of equations back in the days they were invented. But with the advent of advanced computer power these models became more sophisticated and, in parts, more accurate. Many of the early models are still in use today. The modeling error introduced by these approaches gets more and more dominant over a discretization error due to the possibility of finer grid resolution and it depends mainly on the accuracy of the model constants. These constants are empirically calibrated to fit to a large variety of problems.

### 1.1.1. Research Question

The empirical constants in traditional turbulence models have been aquired by fitting computational results to experimental observation and mathematical requirements. It is a totally reasonable approach to make the model applicable to a wide range of flow types. Otherwise, when only trying to map the model to one particular flow, finding the right set of coefficients is nothing but an exercise in curve fitting. An engineer who wants to use one of these turbulence models in his or her simulation can rely on it yielding acceptable results, as long as the application is of a similar type as those flows the model has been fitted to. The question now is, can modern computational methods and high performance computers adapt a model to a given problem type and promise better results? Could it even be possible to define a new range of parameters for one model that fits best to a certain type of flow? In order to generate the fitting of a set of constants to something as complicated as a numerical simulation, a robust optimization method is required. Further are the results of such a simulation hard to foresee and the influence of the turbulence model parameters on the outcome is difficult to predict. Therefore an optimization technique that works without explicit knowledge of the topology of the problem and solution spaces is needed. Traditional gradient based methods are clearly not suitable for that kind of optimization for the reasons stated above, because they require the differentiability of the problem formulation. In turbulence modeling it is not granted that the parameters are mutually independent, adding to the problem of finding the correct optimisation method. Furthermore are gradient based methods prone to end their search in local optima if the solution space is not sufficiently smooth. An alternative are non-deterministic methods, such as genetic algorithms (described in detail in Chapter 2.2). In this work genetic algorithms will be presented and used for optimisation.

The original contribution to knowledge is the application of genetic optimisation to improve the modelling aspect of a CFD (computational fluid dynamics) simulation. It differs from existing applications in so far, that it tries to improve the computed results by optimizing the underlying numerical method instead of changing the geometry and topology of the case under investigation as shape or layout optimization would do. To my knowledge this has never been done before in the field of CFD.

## 1.2. Thesis Outline

Chapter 2 gives an overview over the origins and the development of evolutionary computational methods in the last decades. It concentrates on the methods that are used for the solution of difficult optimization tasks. A short review of their use in engineering applications is also given. A comparison between the traditional, biological principle of evolution and the more systematic computational realisation is drawn and examples of applications in various fields are presented.

In Chapter 3 the principles of numerical modeling of fluid flows are briefly outlined, especially the discretization of the governing equations to show the necessity of turbulence model considerations in the finite volume approach. The basic idea behind this approach is also layed out. A review of commonly used turbulence models in CFD is given and the models considered in later chapters are discussed in more detail.

The following Chapter 4 describes in more depth the implementation details of the genetic algorithm used throughout this work. Different interchangeable operators in genetic algorithms are compared and the requirements needed to develop a generic representation are outlined. A detailed description of the various modules implemented for the Thesis is given in conjunction with a brief instruction how they can be used to run a genetic optimisation on an arbitrary problem. In order to prove that the implementation indeed produces an optimized solution to a problem, a series of benchmarking problems is investigated and the known real optimal solutions compared to the ones obtained with the code presented here.

The method implemented in the Thesis is applied to a series of test cases. This is described in Chapter 5. Potential parameters for optimization are identified. A summary of the set up procedure and results aquired by the optimization process are presented and compared to the corresponding experimental data and non-optimized calculations using standard settings for the models.

In Chapter 6 the optimization is used on an OpenFOAM application for automated mesh generation called snappyHexMesh. Improved mesh quality is achieved by changing the values of parameters that control the building of the mesh. A metric to assess the quality of the generated mesh is developed and a comparison between meshes created with different settings is presented here.

Finally, Chapter 7 summarizes the findings presented in the Thesis and provides some concluding remarks and suggestions for future research.

# 2. Evolutionary Computation

*It is not the strongest of the species that survive, nor the most intelligent, but the one most responsive to change.*

Charles Darwin

## 2.1. Principles of Evolution

Biodiversity is the most apparent proof of the effectiveness of evolution. Within millions of years of adaptation and modification based on a shared pool of similar characteristics, all lifeforms we know today and millions more that became extinct over the millenia have developed. Some survived and some were displaced by better, more adopted specimen that had a higher chance to stand their ground in a predator/prey environment. Life as we know it is able to exist in all the different climates and ecosystems found on this planet. From freezing glaciers to soaring deserts, from deep oceans to wind-swept mountain tops. And common to all these manifestations of life is, that it fits into its niche with an optimised architecture and behaviour. This adaptability to the environment is a marvellous mechanism that ensures diversity and durability of species on earth (except for humans, who use clever inventions and technology to evade the natural adaptation process, but that is a different story). The first person to describe this natural selection process was Sir Charles Darwin in 1859 [18].

The adaptation is done in small steps. By means of sexual reproduction genes of both parents are mixed (crossed over) and passed on to their offspring in the next generation. In this crossover process new attributes (phenotypes) might emerge that are more beneficial to the survival of the new individual. With the additional effect of random mutation, phenotypes might be created that were not present in either parent. But not all changes are beneficial to the individual. Many, if not most of the variations might have an adverse effect on the chances of that individual to persist. Two things might happen: It dies before reproduction or its chances to find a mating partner reduce. This effect was coined as '*survival of the fittest*' (erroneously attributed to Charles Darwin). Giving the process of evolution enough time, i.e. a high number of generations, and enough competitive pressure, the high diversity of creatures and plants as we see today on our planet could evolve.

## 2.2. Genetic Algorithms in Computer Science

The principle of simulating evolutionary processes on a computer has been a subdiscipline of artificial intelligence (AI) since the 1970s. Pioneers on this field were the German computer scientists Ingo Rechenberg [84] and Hans-Paul Schwefel [88]. In their respective dissertations they theoretically founded the method of Evolution Strategies (ES). This was originally designed as a set of rules for experimental optimisation, but is nowadays more commonly used as a numerical method for the identification and optimisation of parameters in mathematical models. It was also based on a phenomenological description of the population. It encoded strategic parameters within a set of individuals instead of breaking down the genetics to a chromosome level. That came later with the advent of genetic algorithms (GA), invented by John Holland in the mid-seventies of the last century [46].

Just like the progression in the understanding of evolution principles took place in biology, the adaptation of these principles to numerical optimisation took similar steps. Charles Darwin, as mentioned above, had no concept of the underlying genetics when he first wrote about the origin of species. His assumptions were based on observations purely of the phenotype. Only much later, after the discovery of the DNA structure and its role in the conveyance of genetic information in the late 1920s could evolution be explained on a finer level. In the same way did evolutionary computation mature from a phenotypical rule set used in ES, where characteristics of an individual were modified towards an optimal representative instead of changing the genome, to an optimisation based on the actual chromosome representation, ie. genotypical, in genetic algorithms. At the same time it moved away from the real biological meaning of evolution and became more of a pure numerical and mathematical exercise. A strong indication of this is that in GAs the evolution is always monotonic. There is no going back, whereas in nature things are not always that simple [90] and may allow negative progression.

When compared to traditional methods of optimisation such as gradient-based methods, GAs differ in some fundamental ways:

- GAs do not work with the parameter set, but with a coding of the set
- Instead of searching from a single point, GAs search from a population of points
- No auxiliary knowledge or derivatives are used, but objective information (or payoff)
- Transition rules are not deterministic but probabilistic

Genetic Algorithms are not the only evolutionary technique that can be used for optimisation. Other methods have been developed over the years that copy natural behaviour. Ant Colony Optimisation, for example, is a member of the family of swarm intelligence methods that quickly became popular in the 1990s [24]. The applications for this kind of method are more in the area of graph theory, for example routing or assignment problems, with discrete solution space. It also needs a very large population in order to produce good results, hence it would not be a good choice for the problem presented here, where estimating the quality of a solution is very time expensive. For algorithmic or process optimisation a combination of genetic algorithms and programming is used, collectively named Genetic Programming [56]. The method uses fitness information on a chain of operations, trying to find an optimal way of finishing a given task. This can be used to optimise algorithms or workflows, but is not suited for parameter optimisation.

Other possible, non-deterministic methods are, for example, Differential Evolution [97], thet iteratively tries to improve a single candidatet function, or Simulated Annealing [55], where the search space decreases with time by adapting the probabilities that control the optimisation process. All these approaches share the common feature that no topological information of the problem space is needed and that they try to find an optimal solution with probabilistic methods.

Of all these evolutionary methods, Genetic Algorithms seemed the best choice for the optimisation of turbulence model coefficients. The main reason why it was chosen was that the formulation of the problem in GAs is very simple. No topological knowledge of the solution space is required. That makes it easy to reuse the same algorithm for a variety of optimisation problems in engineering, beyond those discussed in this thesis.

## 2.3. Genetic Applications in Engineering

The question of how to apply genetic algorithms to problems that occur in industrial engineering has been an interesting subject in the fields of management science, operations research or systems engineering [35]. One of the reasons for this interest is the characteristic of genetic algorithms that makes them a versatile and powerful tool to consider problems that would be very difficult to solve using conventional optimization techniques.

The construction of technical equipment has to meet a certain quality standard and often high efficiency is usually desirable. The design of a machine or parts of it requires an experienced engineer. Often, subtle changes to the geometry of a part can have a significant impact on its performance and this impact is not always easy to predict. Genetic algorithms can help to generate a variety of different designs within a simulation environment.

This technique has been used in various fields of engineering. For example, Aliev et al. [1] used GA's in electromagnetics to improve the shape of an accelerator electrode to optimise the shape and stability of the manipulated electromagnetic field. In the field of turbomachinery, Hilbert et al. [44] used multi–objective optimisation to improve the shape of fan blades in a heat exchanger. Even in classic aerodynamics, GA's are used for shape optimisation. Marco et al. [68] described the shape of an airfoil with bezier splines and used a genetic algorithm to find the set of bezier coefficients that produced the best shape w.r.t. lift and drag behaviour. They also mentioned a common problem of GA's in engineering: The computation of a single shape variety takes a long time, so that only few generations with a small number of individuals can be considered. In their case the typical number of generations was 10–50.

Even though evolutionary methods have been proven to produce useful results, their application in mechanical engineering is scarce. This could either be due to their high computational costs, but also because they are not commonly known to most engineers.

# 3. Computational Modelling

*When I meet God, I am going to ask him two questions: Why relativity? And why turbulence? I really believe he will have an answer for the first.*
Werner Heisenberg

## 3.1. Introduction

Numerical simulation of complex flow phenomena is a challenging field in fluid dynamics. Even with computers getting faster and massively parallel in recent years, the accuracy of the computations is still dependent on the models that describe the underlying flows. The execution of a direct numerical simulation (DNS) is still far away from being affordable in terms of computation time. While in the 80s and early 90s of the last century memory was the limiting factor, it now is time. Even with modern high performance computers and massive parallel calculation, simulating a case of relevance to the engineer in industry resolving all length and timescales would theoretically take decades, if not centuries to compute. For example, Pope [81] estimates the time $T_G$ in days required for DNS of isotropic turbulence at a Reynolds number Re based on the integral length scale $L$ is

$$T_G \sim \left(\frac{\mathrm{Re}_L}{800}\right)^3 . \qquad (3.1)$$

That is why most solvers seek to solve the Reynolds-Averaged Navier-Stokes (RANS) equations (see below). The main flow velocity is seperated into a mean velocity component and turbulent fluctuations, expanding the NS-equations by additional terms that need to be modelled. Several different approaches have been developed and applied, ranging from zero-equation approaches like Prandtl's mixing length model [82], one-equation models like the Spalart-Allmaras model [93], over two-equation models like the k-$\varepsilon$ model by Jones and Launder [52] or the k-$\omega$ model by Wilcox [110]. There also exist various combinations and derivations of these. The empirical nature of the model formulations can lead to undesired behaviour, especially in very heterogenous flows or flow regions with highly unsteady turbulent fluctuations. To improve the applicability of all the models to as wide a range of problems as possible several improvements and changes have been proposed. A simple internet search reveals hundreds of different models, some only slight modifications to the most common ones [38, 40], others adjusted to specific flow types like, for example, flow around buildings [25], oceanic flow [79, 39], flow through porous media [100, 99, 15] and many others.

A different approach to handle turbulence in computational fluid dynamics (CFD) is to model only the small scales of turbulence that cannot be resolved on the computational grid while the larger structures maintain to be described by the original formulation. Large Eddy Simulation (LES) is such an approach and is widely used in industrial simulations already. The requirements to the grid in terms of resolution, i.e. computational cost, can be very restrictive for this model, though. Therefore hybrid models are implemented [3, 19], which use RANS on the coarse parts of the grid and switch to LES where the resolution is fine enough to capture the turbulent length scale. In this review, however, mainly RANS models of turbulence are in the focus of the investigations, their application to industrial flows and shortcomings and limitations in their formulation.

## 3.2. Reynolds-Averaged Navier-Stokes Equations

The governing equations that describe the flow of Newtonian fluids are the Navier–Stokes Equations (NS). They combine a set of conservation equations, which typically are
conservation of mass (often referred to as the continuity equation)

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho u_j\right] = 0, \tag{3.2}$$

conservation of momentum

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}\left[\rho u_i u_j + p\delta_{ij} - \tau_{ji}\right] = 0, \quad i = 1,2,3, \tag{3.3}$$

and conservation of energy

$$\frac{\partial}{\partial t}(\rho e_0) + \frac{\partial}{\partial x_j}\left[\rho u_j e_0 + u_j p + q_j - u_i \tau_{ij}\right] = 0. \tag{3.4}$$

Here, $\rho$ is the density of the fluid, $p$ is the pressure, $u_i$ is the velocity in direction $x_i$, $\tau_{ij}$ is the stress tensor and $q$ is heat flow. Any flow property $\phi$ as a variable of time and space in statistically steady flow can be described as the sum of an average value and fluctuations about that value [27]:

$$\phi(x_i, t) = \overline{\phi}(x_i) + \phi'(x_i, t) \tag{3.5}$$

The method used for averaging could be a time or ensemble average. For unsteady flow ensemble average is the natural choice, but for steady flow a time average would suffice. If the time chosen for the averaging is large enough, i.e. large compared to the time scale of the fluctuations, $(\overline{\phi})$ will be independent of the start time of the averaging. Thus, the time averaging would be defined as

$$\overline{\phi}(x_i) = \lim_{T\to\infty} \frac{1}{T}\int_0^T \phi(x_i, t)dt \tag{3.6}$$

or in case of an ensemble average with ensemble size $N$

$$\overline{\phi}(x_i, t) = \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \phi(x_i, t) \tag{3.7}$$

Reynolds first used an approach of expressing turbulent flow as the sum of mean flow $U$ and turbulent contribution $u'$ [86]. He then formulated the time average of the Navier–Stokes equations, introducing momentum fluxes that are a–priori unknown. These need to be modelled which leads to additional unknown parameters in the equation. New equations for these quantities have to be derived to close the equation system. Statistical steadiness implies that $\overline{\phi'} = 0$. Using that information with Eqn. (3.5) shows that averaging a linear term in the conservation equation just produces the identical term for the averaged quantity. Conventional ensemble rules of averaging apply

$$\overline{\overline{a}} = \overline{a} \tag{3.8}$$

$$\overline{a+b} = \overline{a} + \overline{b} \tag{3.9}$$

$$\overline{a\overline{b}} = \overline{a}\overline{b} \tag{3.10}$$

For a quadratic non–linear term when using (3.8)-(3.10) two terms emerge, the product of the averages and a covariant:

$$\begin{aligned}
\overline{u_i\phi} &= \overline{(\overline{u}_i + u'_i)(\overline{\phi} + \phi')} \\
&= \overline{\overline{u}_i\overline{\phi} + u'_i\overline{\phi} + \phi'\overline{u}_i + u'_i\phi'} \\
&= \overline{\overline{u}_i\overline{\phi}} + \overline{u'_i\overline{\phi}} + \overline{\phi'\overline{u}_i} + \overline{u'_i\phi'} \\
&= \overline{\overline{u}_i}\overline{\phi} + \overline{u'}_i\overline{\phi} + \overline{\phi'}\overline{u}_i + \overline{u'_i\phi'} \\
&= \overline{u}_i\overline{\phi} + \overline{u'_i\phi'}
\end{aligned} \tag{3.11}$$

The term $\overline{u'_i\phi'}$ is zero only if the two quantities are uncorrelated which is rarely the case in turbulent flows. Applying these thoughts to all linear and quadratic terms of the NS equations gives the Reynolds–Averaged Navier–Stokes (RANS) equations

$$\frac{\partial(\rho\overline{u}_i)}{\partial x_i} = 0 \tag{3.12}$$

$$\frac{\partial(\rho\overline{u}_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho\overline{u}_i\overline{u}_j + \rho\overline{u'_i u'_j}\right) = -\frac{\partial\overline{p}}{\partial x_i} + \frac{\partial\overline{\tau}_{ji}}{\partial x_j} \tag{3.13}$$

with mean viscous stress tensor components

$$\overline{\tau}_{ij} = \mu\left(\frac{\partial\overline{u}_i}{\partial x_j} + \frac{\partial\overline{u}_j}{\partial x_i}\right) \tag{3.14}$$

where $\rho$ is the density and $\mu$ the dynamic viscosity of the fluid.

The inclusion of the *Reynolds stresses* $\rho\overline{u_i'u_j'}$ and the *turbulent scalar flux* $\rho\overline{\phi' u_i'}$ into the conservation equations introduces new unknowns to the equation system. These cannot be expressed in terms of the known variables. To solve this dilemma, the Reynolds stresses could be computed directly, leading to *Reynolds–Stress–Models* (e.g. [59]). Another way would be to relate the turbulence stresses to the mean flow:

$$-\overline{u_i'u_j'} = 2\nu_t S_{ij} - \frac{2}{3}K\delta_{ij} \tag{3.15}$$

where $S_{ij}$ is the mean rate of strain tensor and $\delta$ is the Kronecker delta. A new proportionality factor $\nu_t > 0$, the turbulence eddy viscosity, is introduced which can be modeled. Following is a short summary of the most commonly applied eddy viscosity models.

## 3.3. Zero-Equation models

The driving force that transports mass, momentum and energy orthogonal to the streamlines is the viscosity. It is therefore natural to assume that the transport of turbulent quantities is governed by a turbulent or eddy-viscosity [27]. The eddy–viscosity can be expressed as the product of the turbulent velocity and a length scale $\nu_T = u^* l^*$. This length scale has to be prescribed on the whole domain for each flow considered, but its actual quantity is very difficult to estimate. Prandtl used a measure he called *mixing length $\ell$*, a characteristic of the flow, but at the same time described his expression as 'only a rough approximation' [82]. This approximation is accurate enough to describe the general behaviour of the turbulence for very simple flows only, but it provides some insight into the nature of turbulence.

The most often used derivations of the mixing–length theory are the Cebeci–Smith [14] and the Baldwin–Lomax [8] models. Estimating the turbulent viscosity separately in the two layers of a boundary-layer flow, Cebeci–Smith uses mean velocity gradients while Baldwin-Lomax calculates the magnitude of vorticity in the outer layer. The viscosity for the inner layer in both models is a function of the distance to the closest wall. Common to both eddy viscosity models is that they rely on quantities calculated on grid lines normal to the walls, leading to problems when using unstructured or multilayered grids. Furthermore they are known to have problems predicting separated flows well [34]. Yet mixing-length models still persist in industrial flow computations, mainly because they are very easy to implement.

## 3.4. One-Equation models

In the original formulation the velocity scale was locally determined by velocity gradients:

$$u^* = l \left| \frac{\partial \langle U \rangle}{\partial y} \right|, \tag{3.16}$$

but it is obvious that the turbulent velocity scale can be far from zero while the velocity gradient is zero, for example in the center of a round jet [81]. Instead of basing the velocity scale on the velocity gradients, Prandtl [83] and Kolmogorov (cited in [30]) have independently proposed to use the kinetic energy $u^* = ck^{1/2}$ as a basis for the velocity scale instead. In this equation $c$ is an empirical constant. To estimate this quantity, they proposed a transport equation for $k$, leading to a one-equation model. A derivation of the equation can be found in [110]. The introduction of a modeling equation for the turbulent kinetic energy has the advantage that it describes the velocity scale locally, whereas the zero–equation models are strongly dependent on the structure of the grid. The most commonly used models probably are the Spalart-Allmaras (SA) [93] and the Baldwin–Barth [7] model.

### 3.4.1. Spalart Allmaras Model

Spalart and Allmaras based their model mainly on dimensional analysis and empirical observations. The transport equation contains up to twelve arbitrary parameters that have to be tuned by experiment to fit a specific type of flow. But the set of values for the parameters proposed by the authors is widely used in industrial applications and gives reasonably good results in a wide range of applications. It is known, though, that this model causes very high diffusion in regions of three–dimensional vortical flow and can produce misleading results for these flow types. Also the prediction of flow quantities gets very inaccurate close to walls.

Based on dimensional analysis and introducing some modifications for the sake of numerical stability the original model by Spalart and Allmaras is most often used in this form:

$$\begin{aligned}\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} &= C_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 \\ &+ \frac{1}{s}\left[\frac{\partial}{\partial x_j}\left((\nu + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_j}\right) + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_i}\frac{\partial \tilde{\nu}}{\partial x_i}\right] \end{aligned} \tag{3.17}$$

with

$$\begin{aligned} \nu_t &= \tilde{\nu} f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + C_{v1}^3}, \quad \chi := \frac{\tilde{\nu}}{\nu} \\ \tilde{S} &\equiv S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \end{aligned}$$

and

$$\begin{aligned} S &\equiv \sqrt{2\Omega_{ij}\Omega_{ij}}, \quad \Omega_{ij} \equiv \frac{1}{2}(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}) \\ f_w &= g\left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6}\right]^{1/6}, \quad g = r + C_{w2}(r^6 - r) \\ r &\equiv \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \\ f_{t2} &= C_{t3}\exp(-C_{t4}\chi^2) \end{aligned}$$

$\tilde{\nu}$ is the modeled viscosity, $d$ is the distance to the closest wall and $\delta$ is the Kronecker delta. The trip $f_{t2}$ was a numerical fix by Spalart and Allmaras that makes $\tilde{\nu} = 0$ a stable solution. This behaviour was desired in conjunction with a trip function $f_{t1}\Delta U$ given in the original reference to semi-automatically determine the transition point from turbulent to laminar flow. But according to Rumsey [87] most users do not employ this trip function, but run the model in fully turbulent mode. The proposed standard values for the coefficients are given in Table 3.1. The parameter $C_{w1}$ is implicitly calculated from

$$C_{w1} = \frac{C_{b1}}{\kappa^2} + \frac{1 + C_{b2}}{s}.$$

The Spalart–Allmaras model is different from other one-equation models like the ones from Baldwin–Barth [7] or Bradshaw, Ferris and Atwell [13]. In contrast to those, the SA model is not a simplified derivation of the k-$\varepsilon$ but was instead built "from scratch". This ensured that it would not inherit some of the flaws of the k and epsilon transport formulations. The equations for SA were developed in four steps, successively adding more complex physical effects in each step. The central quantity in all terms is the eddy viscosity $\nu_t$. Then diffusion, production and destruction terms were formulated based on effects in free shear flows, boundary layer flows and transitional turbulence. All equations as well as model parameters were derived using empiricism, arguments of dimensional analysis, Galilean invariance and selective dependency on the molecular viscosity [93]. Even though the authors gave a suggestion on the parameter values based on their experience, at no point did they claim these values to be of general applicability. [1]

**Table 3.1.:** Standard values for the Spalart–Allmaras model in OpenFOAM

| $C_{b1}$ | $C_{b2}$ | $s$ | $\kappa$ | |
|---|---|---|---|---|
| 0.1355 | 0.622 | 0.666 | 0.41 | |
| $C_{w2}$ | $C_{w3}$ | $C_{v1}$ | $C_{t3}$ | $C_{t4}$ |
| 0.3 | 2 | 7.1 | 1.2 | 0.5 |

## 3.5. Two-Equation models

In the previously described models the length scale $\ell$ had to be prescribed on the whole domain, which required a certain knowledge of the character of the flow in question. To avoid this, the turbulent length scale needs to be modelled. Consequently, two–equation models are considered as *complete*, meaning they can be used to predict the properties of a turbulent flow without prior knowledge of the turbulence structure. There are two main ideas to do so: The first is to solve a transport equation for the rate of dissipation $\varepsilon$ of turbulent kinetic energy leading

[1] In a personal conversation Stephen Allmaras told me he would not be surprised but would rather expect other values to work better for specific physical problems.

to the k-$\varepsilon$ model first proposed by Jones and Launder [52] where $(\sqrt{k^3}/\varepsilon) \sim \ell$. The other is to model the specific dissipation rate $\omega$, as Wilcox [110] suggested in his k-$\omega$ model, in which $(\sqrt{k}/\omega) \sim \ell$. Both these models are very popular in industrial applications since they are easy to implement and generally robust. Since the k-$\omega$ model can deal more efficiently with the region at the walls and the k-$\varepsilon$ model handles the free shear flow regions better, Menter [70] proposed his 'shear stress transport' (SST) hybrid model that, amongst a few other features, switches between these two models depending on the position in the flow.

### 3.5.1. k-$\varepsilon$ Model

Probably the most commonly used turbulence closure model is the k-$\varepsilon$ model originally formulated by Jones and Launder in 1972 [52]. It makes some very strong assumptions about the nature of the vorticity in the flow field. Foremost it is a static model that does not take into account the history of the strain of turbulence. Further it predicts isotropy of turbulence, which is not the case for any flow with stratification or rotation. Despite these limitations it seems to work reasonably well for many applications. In this model, the turbulent kinetic energy is derived directly from the turbulent flow field as $k = \frac{1}{2}(\overline{\mathbf{u}'^2})$. That means, the higher the energy in the turbulent field, the greater the momentum exchange and thus the greater the eddy viscosity $\nu_t$. Following from that, the length scale $\ell$ would be of the order of the integral scale, since the largest eddies contribute most to the momentum exchange, leading to

$$\nu_t \sim k^{1/2}\ell. \tag{3.18}$$

Now using the observation, that in most forms of turbulence $\varepsilon \sim u^3/l$ [20], Jones and Launder calculated the eddy viscosity from

$$\mu_t = C_\mu \rho k^2/\varepsilon \tag{3.19}$$

where $C_\mu$ is a coefficient with the (empirical) value of $\sim 0.09$. There is no fundamental reason, why $\mu_t$ should only be dependent on turbulence parameters such as k, $\ell$, $\varepsilon$ or $\omega$. Thus, two-equation models are no more universal in describing

the behaviour of a flow than one-equation models are. Furthermore, they can be expected to be inaccurate for many non-equilibrium turbulent flows [110].
Further simplification of the physical nature of turbulence and empirical observations lead Jones and Launder to these advection/diffusion equations for k and $\varepsilon$:

$$\begin{aligned}\rho\frac{\partial k}{\partial t}+\rho U_i\frac{\partial k}{\partial x_i}&=\tau_{ij}\frac{\partial U_i}{\partial x_j}-\rho\varepsilon\\&+\frac{\partial}{\partial x_i}\left[(\mu+\mu_T/\sigma_k)\frac{\partial k}{\partial x_i}\right]\end{aligned}\tag{3.20}$$

$$\begin{aligned}\rho\frac{\partial \varepsilon}{\partial t}+\rho U_i\frac{\partial \varepsilon}{\partial x_i}&=C_1\frac{\varepsilon}{k}\tau_{ij}\frac{\partial U_i}{\partial x_j}-C_2\rho\frac{\varepsilon^2}{k}\\&+\frac{\partial}{\partial x_i}\left[(\mu+\mu_T/\sigma_\varepsilon)\frac{\partial \varepsilon}{\partial x_i}\right]\end{aligned}\tag{3.21}$$

The $\varepsilon$ equation 3.21 is almost pure construction. It is based on a general assumption on the structure of the equation which models the turbulent length scale. A proper derivation of the $\varepsilon$ equation would involve more modeling and require a sound understanding of the nature of turbulence which, unfortunately, is not avilable. Launder [60] presents a short overview of different approaches using a variety of relations between k and $\ell$, introducing a general variable $z = k^m\ell^n$. In this case $z = \varepsilon = k^{3/2}\ell^{-1}$. The transport equation for z contains three nominally arbitrary parameters $C_1$, $C_2$ and $\sigma_z$. Launder describes how each of the coefficients can be determined from well documented flows: To obtain a value for $\sigma_z$ in the diffusion term of Eqn. 3.21 he compared computational results with measurements of asymmetric flow between parallel planes of which one was smooth the other roughened. The difference in texture leads to a larger contribution of the diffusion term than in wall boundary layers. From his observations he concluded that $\sigma_\varepsilon$ and $\sigma_k$ should be in the order of unity, settling on 1.0 and 1.3 respectively in his final publication.

$$\sigma_z \simeq 1\tag{3.22}$$

For the value of $C_2$, experimental data on the decay of turbulence behind a fine wire screen was used. In this particular setup the variables become dependent only of one directional coordinate, transforming the equations for k and z to ordinary differential equations. To match the model with the observations, $C_2$ would have to be expressed as

$$C_2 = C_\mu f(m, n) \tag{3.23}$$

where in the case of the k-$\varepsilon$ model $m = 3/2$, $n = -1$.
Finally, $C_1$ was estimated by looking at near–wall turbulence, where convection and diffusion of kinetic energy are negligible. By setting $C_1$ in relation to other constants, Launder derived an initial value of 1.5, but after fine tuning all C's and $\sigma$'s using computer optimisation, recommended a final value of 1.45.

Over the years, these parameters were all slightly modified to fit more universally to a wider range of flow problems. More canonical test cases were considered for tuning the coefficients using regression analysis. As Davidson states in his book about turbulence [20] *'the k-$\varepsilon$ model is a highly sophisticated exercise in interpolating between data sets.'* But just this property makes it a very promising candidate for the purpose of this thesis, that aims to identify an optimal set of parameters for any given flow. The main problem of the k-$\varepsilon$ model is its treatment in the near-wall region of the flow where the destruction-of-dissipation term is singular. To avoid this in a layer close to the wall the flow has to be treated seperately by a wall function. The resolution of the grid close to the walls has to be sufficiently fine for the wall functions to yield reasonable results, meaning additional care needs to be taken when solving a problem using this model.
The standard values for this model as they are found in the OpenFOAM implementation are given in Table 3.2.

**Table 3.2.:** Standard values for the k-$\varepsilon$ model as implemented in OpenFOAM

| $\sigma_k$ | $\sigma_\varepsilon$ | $C_1$ | $C_2$ | $C_\mu$ |
|---|---|---|---|---|
| 1.0 | 1.3 | 1.44 | 1.92 | 0.09 |

### 3.5.2. k-$\omega$-SST Model

Another approach is to model the specific dissipation rate $\omega$, as Wilcox [110] suggested in his version of the k-$\omega$ model, in which $(\sqrt{k}/\omega) \sim \ell$. Menter [71] introduced a modification to that model combining the near-wall treatment of the k-$\varepsilon$ model and the accuracy in predicting the free flow from the k-$\omega$ model. He used blending functions to switch from one model to the other. The eddy viscosity equation is modified to account for the transport effects of the principle turbulent shear stress (hence the name k-$\omega$-SST). Menter's formulation is widely used in aerodynamics and is a good candidate to test the capability of the genetic optimisation as it contains no less than eleven arbitrary coefficients, of which the default values are given in Table 3.3. The implementation of this model in OpenFOAM uses the following equations:

$$\mu_t = \frac{\rho a_1 k}{\max(a_1\omega, SF_2)} \tag{3.24}$$

$$\begin{aligned}\rho\frac{\partial k}{\partial t} + \rho U_i\frac{\partial k}{\partial x_i} &= \tilde{P}_k - \beta^*\rho k\omega \\ &+ \frac{\partial}{\partial x_i}\left[(\mu + s_k\mu_t)\frac{\partial k}{\partial x_i}\right]\end{aligned} \tag{3.25}$$

$$\begin{aligned}\rho\frac{\partial \omega}{\partial t} + \rho U_i\frac{\partial \omega}{\partial x_i} &= \rho\frac{\gamma\tilde{P}_k}{\nu_t} - \beta\rho\omega^2 \\ &+ \frac{\partial}{\partial x_i}\left[(\mu + s_\omega\mu_t)\frac{\partial \omega}{\partial x_i}\right] \\ &+ 2(1-F_1)\frac{\rho s_{\omega 2}}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}\end{aligned} \tag{3.26}$$

using a production limiter

$$P_k = \mu_t\frac{\partial U_i}{\partial x_j}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \rightarrow \tilde{P}_k = \min\left(P_k, c_1\beta^*\rho\omega k\right).$$

Each of the constants $\phi \in \{\beta, \gamma, s_k, s_\omega\}$ is a blend of an inner $\phi_1$ and outer $\phi_2$ constant, blended via:

$$\phi = F_1\phi_1 + (1-F_1)\phi_2$$

with blending function

$$F_1 = \tanh\left[\min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\rho s_{\omega 2}k}{\mathrm{CD}_{k\omega}y^2}\right]^4\right] \tag{3.27}$$

$$\mathrm{CD}_{k\omega} = \max\left(2\rho s_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-10}\right) \tag{3.28}$$

where $\rho$ is the density, $\nu_t = \mu_t/\rho$ is the turbulent kinematic viscosity, $\mu$ is the molecular dynamic viscosity, $y$ is the distance from the field point to the nearest wall. F1 is equal to zero away from the surface (k-$\varepsilon$ model), and switches to one inside the boundary layer (k-$\omega$ model). Note that the production limiter coefficient $c_1$ is proposed as a constant in the original paper by Menter [71], but is implemented as a variable in OpenFOAM.

**Table 3.3.:** Standard values for the k-$\omega$-SST model in OpenFOAM

| $s_{k1}$ | $s_{\omega 1}$ | $\gamma_1$ | $\beta_1$ |
|---|---|---|---|
| 0.85034 | 0.5 | 0.5532 | 0.075 |
| $s_{k2}$ | $s_{\omega 2}$ | $\gamma_2$ | $\beta_2$ |
| 1.0 | 0.85616 | 0.4403 | 0.0828 |
| $a_1$ | $c_1$ | $\beta^*$ | |
| 0.31 | 10 | 0.09 | |

## 3.6. Parameter Identification

Most of the above mentioned turbulence models include a number of parameters that need to be calibrated to the type of flow that is to be investigated. Surprisingly, most of these coefficients have little or no physical relevance at all and are merely empirical. The number of parameters vary from model to model with up to twelve in the Spalart–Allmaras equation. Lengthy experiments have to be conducted to estimate a set of values for the coefficients that best describes a specific type of flow covered by the experiment. And even though the authors of the models themselves provided standard values for the flows they investigated, these standards are used by industrial users regardless if they are fit to adequately describe a problem, or not. Some research was done in a-priori parameter identification by Qian et al. [104], Bardow [10], and others, but all these considerations did not lead to a better understanding of the impact of the parameters to the behaviour of the solution.

One of the more well known examples where a specific turbulence is known to give bad results but where changes to the parameters have been proposed to improve the quality of the calculation, is the spreading rate of round jets when using a k-$\varepsilon$ or k-$\omega$ model. Closer investigation of the agreement with experimental data done by Wilcox [110] has shown an underprediction of the spreading rate of up to 60% with the k-$\omega$ model and an overprediction of up to 30% with the k-$\varepsilon$ model. Suggestions of modifications to the dissipation equation have been made to better match the predictions. (e.g. by Pope [80]). Yet, while varying the coefficients might lower the discrepancy between computed and observed data, a statistical analysis by Smith et. al. [91] has shown that it is not possible to find a set of parameters that equally fits a wide range of jet configurations. So optimisation can only be a solution with very limited applicability.

This notion, of course, raises the question what this current research is hoping to achieve. First of all, the idea to generate an optimal parameter set for a turbulence model is of more than just academic value. Consider, for example, a car manufacturer who bases their design constraints on results from a CFD simulation with the overall goal to reduce the drag coefficient. It is known, that small changes to the coefficient affect the fuel consumption and, immediately connected to this, the $CO_2$ emissions. To capture minute changes accurately and reliably, a good working turbulence model is required (if the value is obtained with RANS simulations). The effort to tune the model parameters to the problem at hand is expensive in terms of time and resources, but since the same type of flow will be simulated over and over again within the workflow of the manufacturer, a benefit can be gained from the optimisation procedure in the long run.

## 3.7. Finite Volume Method

In order to solve the partial differential equations that govern the behaviour of fluid flow it is required to transform them into a system of algebraic equations that can be solved numerically. This discretisation has to be performed spatially as well as temporally. The outcome of this procedure is a description of the computational domain in terms of points in space where the solution shall be obtained and a set of boundary conditions prescribing the solution at given points in time and space. By means of this discretization the space is divided into a finite number of distinct regions, called cells or control volumes. That is why it is known as the Finite Volume Method (FVM).

### 3.7.1. Spatial Discretisation

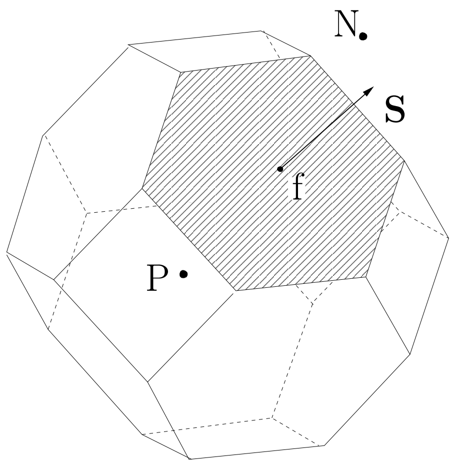


**Figure 3.1.:** A control volume with centre point P, face area vector S of face f and neighbour cell centre N (Source: OpenFOAM Programmer's guide).

The accuracy of a numerical solution of a physical field problem depends not only on the order of approximation, but also on the distribution of grid points in the computational domain. The quality of the grid based on the geometric characteristics, as well as on the solution characteristics influenced by the field properties being simulated is an important aspect in the improvement of the accuracy and convergence rate of the solution. The setup of the computational grid and the structure of the partial differential equation (PDE) to be solved are therefore closely coupled [69].

Figure 3.1 shows an arbitraryly shaped polyhedron that is used as a control volume (CV). For the integration of the flux values in the finite volume formulation of the flow equations, properties of the cell and its faces need to be known. The centre of the CV is denoted here as $P$ and the centre of the cell that is the direct neighbour of face $f$ is $N$. The area vector of the face is $\mathbf{S}$ and it is orthogonal to the face, pointing outward from the cell centre.

The discretisation of the spatial domain into cells of arbitrary shape requires a numerical method that is easily adaptable to this kind of grid. The finite volume approach is such a method [27] and the discretisation of the governing equations shall be discussed in this section. Focus is on the parts of the equations that are susceptible to bad mesh quality. These are mainly the spatial discretisations that are tackled with low-order differencing schemes. The discretisation of the solution domain produces a numerical description of the computational domain, including the positions of points in which the solution is sought and the description of the boundary. The common form of the transport equation for a scalar property $\phi$ is

$$\underbrace{\frac{\partial \rho\phi}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla \bullet (\rho \mathbf{U} \phi)}_{\text{convection term}} - \underbrace{\nabla \bullet (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusion term}} = \underbrace{S_\phi(\phi)}_{\text{source term}} . \tag{3.29}$$

The entities in this equation are the density $\rho$ of the fluid (which is constant for incompressible flow), the velocity vector $\mathbf{U}$ and the diffusivity of the scalar $\Gamma$. The finite volume method requires Eqn. 3.29 to be correct on the control volume $V$ around point $P$:

$$\begin{aligned} \int_t^{t+\Delta t} \left[ \frac{\partial}{\partial t} \int_{V_P} \rho\phi dV + \int_{V_P} \nabla \bullet (\rho \mathbf{U}\phi) dV - \int_{V_P} \nabla \bullet (\rho \Gamma_\phi \nabla\phi) dV \right] dt \\ = \int_t^{t+\Delta t} \left( \int_{V_P} S_\phi(\phi) dV \right) dt. \end{aligned} \tag{3.30}$$

Applying Gauss' theorem to each term of this equation and thereby transforming the volume integrals to surface integrals over the boundary of the control volume, the spatial terms are discretised [50]. The diffusion term becomes:

$$\int_{V_P} \nabla \bullet (\rho \Gamma_\phi \nabla \phi) dV = \sum_f (\rho \Gamma_\phi)_f \mathbf{S} \cdot (\nabla \phi)_f \tag{3.31}$$

The convection term is then handled as:

$$\int_{V_P} \nabla \bullet (\rho \mathbf{U} \phi) dV = \sum_f F \cdot \phi_f \tag{3.32}$$

where $F$ represents the mass flux through the face $f$:

$$F = \mathbf{S} \cdot (\rho \mathbf{U})_f \tag{3.33}$$

Equation 3.32 is of interest in judging the quality of the mesh, especially the mesh skewness. This is addressed in detail in Section 6.3.1.

One should bear in mind that the derivation of higher order CV methods is rather difficult. Ferziger and Perić [27] make it clear, that second order accuracy is the best one can reach with single-point approximations using the mid-point rule and linear interpolation. For interpolation of higher order, more neighbours have to be taken into account, which is difficult if not unmanagable on unstructured, three dimensional grids.

# 4. Genetic Optimisation

*Real stupidity beats artificial intelligence every time.*
TERRY PRATCHETT, HOGFATHER

## 4.1. Genetic Algorithm Basics

Genetic Algorithms are based on the principle of natural selection and natural genetics [37]. GAs are randomly initialised, asserting a diverse set of possible solutions. Compared to conventional optimisation methods they will explore many areas of the solution space simultaneously during the evolution process. That reduces the probability to get trapped in local optima, as common gradient based methods would do, possibly missing the global optimum completely.

Figure 4.1 depicts the sequence of operations in a typical GA. At first an initial population of possible solutions is generated randomly. Using a uniformally distributed random number generator should ensure an equal spread of the population over the solution space. In the evaluation phase each individual is assigned a fitness value. Calculating this fitness is the most time consuming part of the process. The order of application of genetic operators is interchangeable, but will be repeated until termination of the optimisation. Candidates are chosen for reproduction based on their fitness, using one of the implemented selection operators. Then, with a given probability, these candidates are transformed to create new individuals by applying a crossover operator and eventually the chromosome is changed by the mutation operator. After the new population is built that way it will again be evaluated and the termination criterion will be checked. This criterion might be reaching a maximum number of generations or a measure for the diversity of the population. If the criterion is not met, the algorithm repeats by creating a new generation of individuals using the defined genetic operators.

A set of parameters in a GA will generally be coded as a string of finite length, most commonly a binary string. Each of these strings (also *chromosome* or *genotype*) represents one possible solution to the optimisation problem. Two opposed strategies are at work here: Exploitation of a solution versus exploration of the solution space. Classical gradient based methods concentrate on exploiting, while a fully exploratory approach would correspond to a random search. GAs manage to reach a very good balance between those two extremes [73].

The implementation details described in the following sections refer to the realisation chosen for this thesis. Alternative implementations might be possible but were either not considered or did not prove relevant within the scope of this work.

### 4.1.1. Chromosome Encoding

Since the problem variables are integer or real values and their chromosomal representation is a binary string, a mapping has to be defined. For a single coefficient $c \,\epsilon\, [c_{lo}, c_{hi}]$ the length of the bitfield $b = \langle b_0 b_1 \dots b_n \rangle_2$ has to be determined by taking into account the desired resolution $\Delta_c$ of the interval. The number of bits required to represent the interval $[c_{lo}, c_{hi}]$ is

$$n = \left\lceil \log_2 \left( \frac{c_{hi} - c_{lo}}{\Delta_c} + 1 \right) - 1 \right\rceil \tag{4.1}$$

Translation from binary to decimal and vice versa can now easily be done as follows:

$$\langle b_1\, b_2\, \dots\, b_{n-1}\, b_n \rangle_2 = \left( \sum_{i=1}^{n} b_i \cdot 2^i \right)_{10} = c' \tag{4.2}$$

$$c = c_{lo} + c' \cdot \frac{c_{hi} - c_{lo}}{2^{n+1} - 1} \tag{4.3}$$

In a first attempt chromosome encoding was implemented using Gray's algorithm [105], which ensures that successive numbers in a bit coded string only differ by a single bit. Since uniform mutation is used where only a single bit is changed it is obvious that the result on a classical binary coded chromosome results in a greater displacement in the solution space than applying the mutation operator to a Gray coded chromosome. That is why for the remainder of this work all chromosomes are coded in the fashion described in equation 4.2.

To convert a binary number $b_1\, b_2\, \ldots\, b_{n-1}\, b_n$ into its corresponding binary Gray code, the coding algorithm works as follows:

- Take the right most digit $b_n$
- If the next digit $b_{n-1}$ is 1, replace $b_n$ with $1 - b_n$
- proceed with the next digit
- assume that $b_0$ is always 0

For example the binary number $\langle 1\,0\,0\,1\,0\,1 \rangle$ would be transformed to $\langle 1\,1\,0\,1\,1\,1 \rangle$. It is stated by Michalewicz in [73] that Gray coding would move the genetic algorithm closer to the problem space, meaning that the distance between two points in the representation space should be similar to the distance of these points in the problem space.

## 4.1.2. Objective Function

The translation of the chromosome by decoding the binary representation is done by the objective function. This function therefore converts the genotype into a phenotype. It has no other functionality than producing an integer or real number from the binary string. Validity of the value within the requirements imposed by the problem description has already taken place before the value was encoded. The resulting objective value is then mapped onto a fitness value, that is used in the selection process described in the following section. This mapping can be an identity mapping, if the objective value satisfies the conditions described in section 4.2, or it could be any kind of mathematical conversion that allows the value to be used in the selection process.

### 4.1.3. Selection

Individuals are selected for reproduction depending on their fitness value. This selection process is stochastically controlled, assigning fitter individuals a higher probability to get chosen. From those individuals (*parents*) selected in this manner, offspring (*children*) are generated by applying crossover and mutation operators. Two common selection procedures are implemented so far, but others can easily be added.

**Roulette Wheel selection** In this selection operator fitness is directly proportional to the probability of selection. To select an individual from a population all fitness values are summed up and the contribution of one indiviual's fitness to the sum determines the chance to be selected. For example in a population with five individuals, figure 4.2 shows the chances of selection for each individual according to their respective fitness. The benefit of this selection procedure is, that it perfectly reflects the fitness value in the selection procedure. Individuals with very low fitness are not very likely to ever be selected, therefore the population will advance rapidly in very few generations. This is also a downside, because the diversity will also diminish [73]. It will also not work very well if the fitness values are very close together, giving all individuals an almost equal chance of selection. The range of the fitness value is also important. For once it has to be a positive, real number so summing up the fitness makes sense. Further the higher fitness value has to be the optimisation target. If the fitness value shall be minimised it has to be adjusted accordingly.

**Tournament selection** This selection operation is easier but also has a few advantages in many cases. From the population a number of individuals, determined by the *tournament size* parameter, is selected with equal probability. Of this tournament set, the best individual, i.e. the individual with the best fitness, is selected for reproduction. One advantage of this procedure is the maintaining of diversity for more generations, allowing for a more thourough exploration of the solution space. Especially if the topology of the solution space is not known in advance and may contain many local optima, tournament selection is the better choice.

The difference between these two methods is that in tournament selection individuals get chosen based on their ranking rather than their actual objective value. It has been shown that this approach helps to avoid premature convergence and speeds up the search when convergence is approaching [108]. There are other selection algorithms that do not consider all individuals for selection, but only those with a fitness value that is higher than a given arbitrary constant. Other algorithms select from a restricted pool where only a certain percentage of the individuals are allowed, based on fitness value. Other selection methods such as stochastic universal sampling sampling (SUS) or reward-based selection (for multi-objective optimization) can be found in literature [5, 64] and could easily be added to the software framework presented in this research.

### 4.1.4. Crossover

The crossover operator uses two parents and combines elements from one parent with elements from the other, creating a new individual that now contains information from both its ancestors. In single-point crossover, one point is chosen at random at which the two parent individuals are split and reassembled in switched order, as illustrated in Figure 4.3. An example of single point crossover between two chromosomes (binary strings) $a$ and $b$ of length n+1:

$$\begin{aligned} a &= \langle a_n\, a_{n-1}\, \dots\, a_1\, a_0 \rangle \\ b &= \langle b_n\, b_{n-1}\, \dots\, b_1\, b_0 \rangle \end{aligned} \tag{4.4}$$

with a randomly selected crossover point $X \epsilon\, [0, n-2]$, creating children:

$$\begin{aligned} a' &= \langle a_n\, a_{n-1}\, \dots\, a_{X+1}\, b_X\, b_{X-1} \dots\, b_1\, b_0 \rangle \\ b' &= \langle b_n\, b_{n-1}\, \dots\, b_{X+1}\, a_X\, a_{X-1} \dots\, a_1\, a_0 \rangle \end{aligned} \tag{4.5}$$

The selection of crossover points as well as their number can be varied to produce new forms of crossover operators. In multi point crossover the number $M$ of crossover points is fixed, but greater than 1. A set of chromosomes that have sections of bits in common are called a *schema* [37]. Over the course of an optimisation, a small number of schema will dominate the population. In some problems, though, a single point crossover operator is not able to produce certain schema [73]. In that case multi point crossover should be used. Another form is uniform crossover, where a randomly generated mask is laid over the parent chromosomes to produce offspring. The procedure is depicted in Figure 4.4.

The preference of which crossover method to use is still a matter of argument in literature. It was not possible to find a final statement on this topic. It seems rather that the selection of the "right" operator is very much problem dependent [67], but no guideline could be found in related publications.

### 4.1.5. Mutation

Mutation is in most cases implemented as uniform mutation where the value of a single bit in a chromosome is inverted from 1 to 0 or vice versa [74]. The probability of mutation is controlled by an external variable $P_M$. The rate of mutation is usually chosen in relation to the population size [114]. The random nature of this operation is important to maintain diversity within the population. Even with low mutation probability it prevents premature convergence to a possibly false optimum throughout the whole evolution procedure. Another form of mutation is Gaussian mutation, which is only applicable to integer or float variables and instead of mutating the bit representation of the value, a gaussian distributed random number is added or subtracted from the current value. In non-uniform mutation the probability of mutation changes over time. Neither of these alternative mutation operators is implemented, but can easily be added to the modular structure of the existing code.

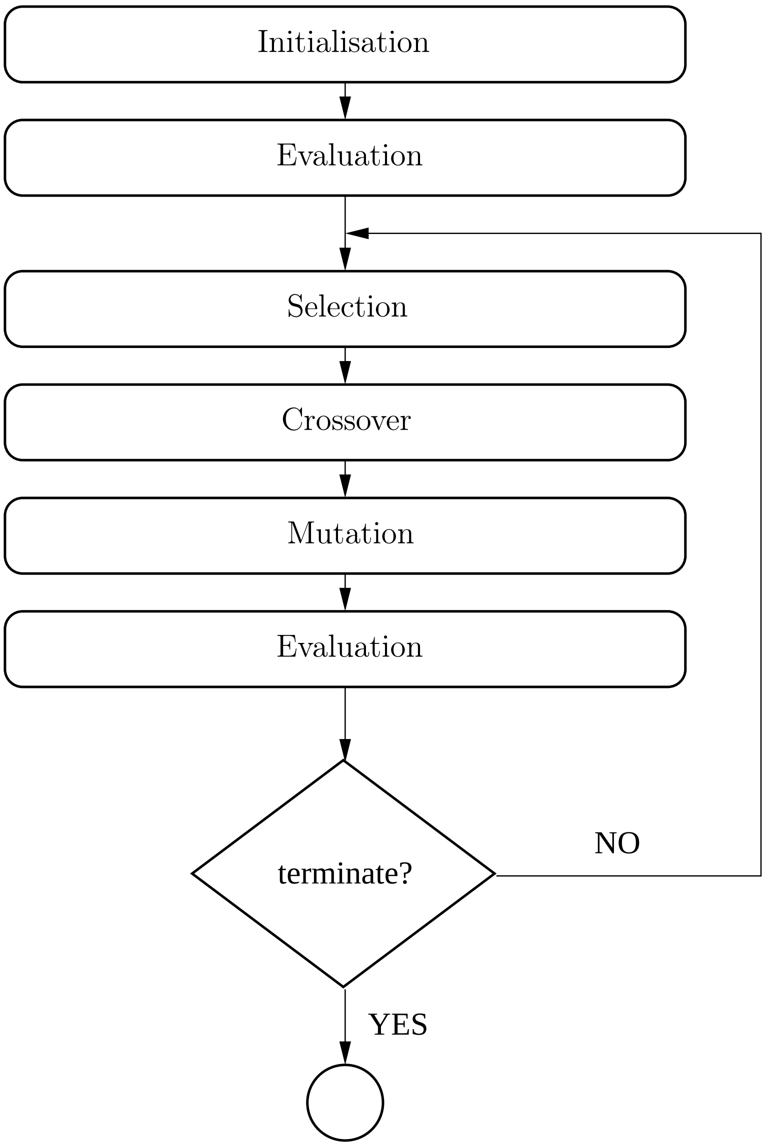


**Figure 4.1.:** Schematic of the workflow of a typical genetic algorithm.

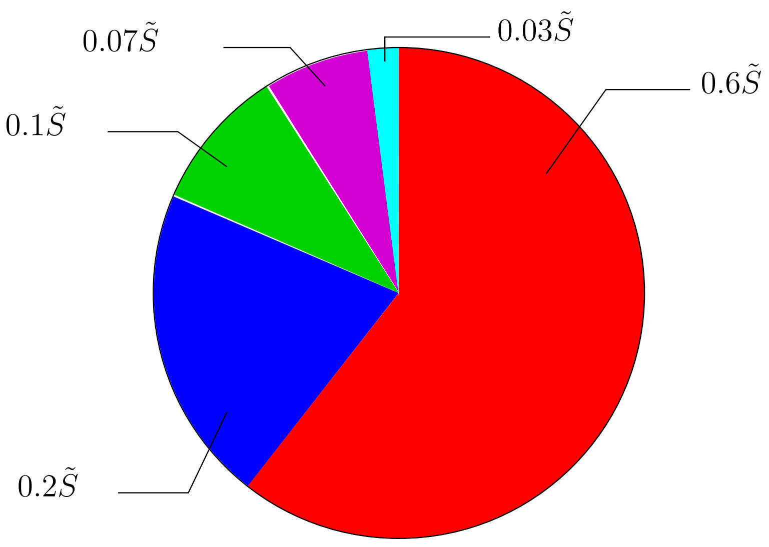


**Figure 4.2.:** Visualisation of the selection probabilty for each of five individuals using roulette wheel selection. $\tilde{S}$ is the sum of all fitness values.

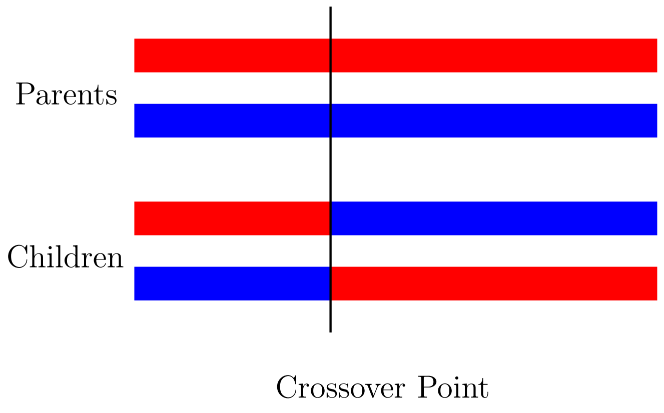


**Figure 4.3.:** Example for a single point crossover operation on two individuals.

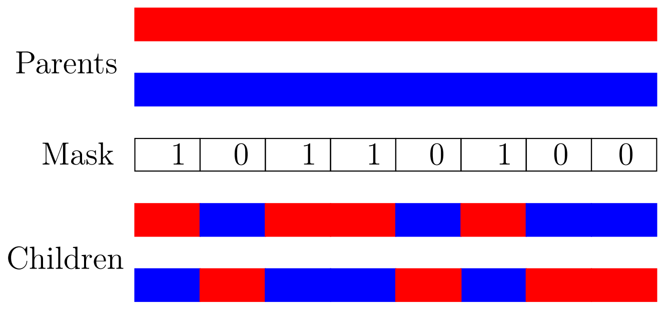


**Figure 4.4.:** Example for a uniform crossover operation.

## 4.2. Fitness

The driving force, so to speak, of evolutionary strategies is the fitness of the individual. It influences the chance of being selected for reproduction, which allows it to pass its characteristics on to new descendants via crossover operations or it might remain unchanged and advance to the next generation. Fitness introduces high inter–individual pressure for survival at different degrees, depending on the type and the implementation of the selection procedure and the probabilities of mating operators.

In traditional genetic algorithms the fitness value definition underlies a set of requirements. Since many selection methods depend on meeting these conditions in order to work, they have to be taken in consideration when conceiving a fitness evaluation function. Foremost fitness should always be represented by a positive, real number. Roulette wheel selection, for example, relies on this requirement because it calculates the sum of all fitness values in one generation and selection is based on each individual's contribution to this sum. Which leads to another condition the fitness value has to meet: The higher the value the higher the chances of survival. These considerations have to be taken into account when defining a fitness evaluation method.

Another requirement is limiting the fitness value to the interval $[0, 1]$. This might be difficult to provide, especially if the range of possible fitnesses is unknown a-priori. If a selection procedure relies on this requirement, for example the crowding distance assignment in the improved Nondomniated Sorting Genetic Algorithm (NSGA-II) (see Section 4.5.1), one could normalise fitness values by dividing by the highest fitness among all individuals in a generation once all have been evaluated. That requires careful design of the operations and the order in which they are applied to avoid unnecessary calls to the evaluation function. It also makes comparison between fitnesses over many generations difficult when they are normalised with different values. This has to be taken into account if performance statistics are of interest. In this thesis fitness requirements are handled rather losely and active consideration of this problem has to be done when implementing the fitness function to meet any requirements the selection function might impose. No automated error handling is in place for this purpose. If it is not possible to ensure this, a certain selection function might not be appropriate for the problem and should be dismissed outright.

## 4.3. Implementation

### 4.3.1. Language Selection

Many implementations of genetic algorithms are available online either as open source software under a form of common license or as commercial software. Also different programming languages are supported and libraries for those languages are available. Yet within the scope of the current research an arbitrary implementation was designed and written to achieve maximum flexibility with minimum overhead. Transparency is gained at the cost of efficiency, but since the computational cost for the genetic algorithm is negligible in comparison to that of the fitness function evaluation, efficiency is less important.

The language chosen for this project is Python (current version 2.7), an object–oriented interpreter language. The reason for this selection is the OpenFOAM library PyFoam [1], which provides functions and applications to control the workflow of a simulation run with OpenFOAM. That includes reading and writing of files in OpenFOAM's own file format. Instantiating OpenFOAM applications such as solvers and post-processing tools is supported and automatically creates log files of the execution. These can then be parsed and analysed by the genetic algorithm. Object–oriented features of Python make design and testing of the code easier, because these processes are well defined in the development cycle of object–oriented software [9].

### 4.3.2. Code Design

In order to write software that is as generic as possible the design process has to be treated with special care. Based on the guidelines by Gagné and Parizeau on how to write generic EC software tools [31], the framework structure should meet the criteria discussed in the following sections. The term 'generic' in this

[1]`http://openfoamwiki.net/index.php/Contrib_PyFoam`

context needs further explanation. According to the computer dictionary[2], generic software is *'Software which can perform many different types of tasks but is not specifically designed for one type of application'.* Taking that into account the development of a generic EC framework should not be tailored to one specific form of optimisation. Operators, such as the crossover or selection operator, should be interchangeable regardless of the objects they are applied to. In addition, the underlying representation of a solution should not affect the way the GA works.

Interchangeability of operators can easily be implemented in modern object-oriented programming languages. The developer can choose from a given set of predefined operators or can add new operators to meet specific needs. This is usually required for the fitness evaluation which is a problem dependent function. Independence from the optimisation problem and reusability are key features of the selection and crossover mechanisms. Commonly used realisations of these are therefore included in the developed framework, but can be altered or new ones can be implemented. This is possible through the realisation of the *strategy* design pattern (see chapter 5 in [32]). In this pattern a family of algorithms is defined, each one encapsulated in an individual module. This allows the different algorithm to be interchangeable, regardless of the underlying calling procedure. Equally flexible is the selection of the coding algorithm that encodes and decodes the chromosome as described in section 4.1.1.

#### 4.3.2.1. Generic representation

The way an individual is represented varies depending on the problem. It should be possible to define a representation that is free in the choice of its underlying data structure while it still provides a generic interface to interact with selection or genetic operators. This criterion is fulfilled by implementing an evaluation function as a member of the Individual class, that translates whichever machine coding the individual uses into a context sensitive value that can be used by the other operators. For example the `IntegerValueIndividual` would evaluate into a single integer value, while the `MultiRealValueIndividual` would translate to an array of real values.

[2]`http://www.computingstudents.com/dictionary`

#### 4.3.2.2. Generic fitness

The fitness of an individual is the most problem dependent part of the GA implementation. There is no way of realising a completely generic function that fits all problems, but instead focus should be on the interface between selection operator and fitness function. This interface should return a single value, or in case of a multi-objective optimisation one value per objective that measures the fitness of this individual. It is now up to the selection procedure to decide if the optimisation problem is to minimise or to maximise the fitness, or to decide which fitness values precede others. The fitness function is also the place in the GA that is least dependent of what other elements of evolutionary methods are used in the solution of the problem at hand. To achieve generality in the implementation of the fitness function, the return value should be abstracted and a comparison function should be provided that returns the fitter of two values to the caller. In my case fitness is always represented by a real number or an array of real numbers, so that this additional level of abstraction was not considered.

#### 4.3.2.3. Generic operations

Manipulation as well as selection operators should be usable for a wide range of possible representations and should not have side effects that influence each other. The usage of any number of operators in an arbitrary order should not alter the behaviour of any other function, in other words operators should be independent of each other. This requirement strongly suggests the use of the aforementioned *strategy* design pattern. In this implementation, a population is assigned a number of operators used for the evolution. These are operators for selection, crossover and mutation. Common specimen like single point crossover or tournament selection operators are provided, but new varieties can easily be added, as long as they fulfil the condition of mutual independence.

#### 4.3.2.4. Generic evolutionary model

Gagné and Parizeau describe the genericity of the evolutionary model as the possibility to interchange the order of operators or the number of times an operator is applied to the population. They make a point that new operators can be added to the model without rewriting it [31]. In the implementation presented here the order of operations is defined in the population's `evolve` function. It is possible to derive new classes from an existing population class that uses a different set of operators or creates a different variety of offspring population.

#### 4.3.2.5. Configurable in-/output

Some of the behaviour of a genetic algorithm is controlled by fixed parameters, such as population size or mutation probability. It is desirable to make these parameters modifiable without changing the code for optimal flexibility. Control parameters are therefore stored in external configuration files. The structure of such a file is explained in detail in Section A.2.1. For every variable that is subject to the evolution process the end user of the software can define lower and upper bounds as well as the desired precision. This allows running different test cases with different initial setups without altering the code. The only element that has to be adapted and implemented for each case is the fitness evaluation function since it is problem dependent. None of the subclasses writes out any information, data is merely stored in utility data structures. That allows full control over the formatting and selection of information to be written from the main function.

### 4.3.3. Fitness Function

Implementation of a fitness function is rather easy. The function signature expects an object of type `individual` as input parameter and should return a single real value as result. A pointer to this function is assigned to the individual's `fitnessFunc` member variable. The `fitness()` method calls this function, passing the objects `self` pointer and returning the fitness value. Internally the value is stored as another member variable within the individual object and the `valid` flag set to true to avoid multiple evaluation of the same individual. Only if the object is flagged as invalid the fitness function is invoked.

### 4.3.4. Parallelisation

The structure of a genetic algorithms makes it suitable for performing parts of the computation in parallel. To evaluate the fitness the individuals do not have to communicate with each other at all. To speed up the process of evaluating a generation, the fitness can be computed on many computer cores simultaneously. In this work the Message Passing Interface (MPI) standard was used to enable interaction between seperate processes. The Python language offers a wrapper class around the API called `mpi4py`. It provides all the functions required to initialise the environment for parallel processing as well as data packaging (*pickling*) and communication methods.

Parallelising the genetic algorithm is very straight forward. One process acts as the master node and distributes the work load to the other processes. On the master node the initialisation of the population takes place and the parameter files are processed. In the main generation loop of the GA each free node receives one individual and performs the fitness evaluation. It will then return the fitness value to the master node, which will either send out another individual if there are any left unevaluated, or it will perform the genetic operations crossover and mutation on the current population and advance to the next generation. Once all individuals have been computed, the master broadcasts a finalisation message, telling all the slave nodes to stop listening for more individuals. Because of the huge difference in computational costs between fitness evaluation and GA operations, the process is totally dominated by the work of the slave nodes. Time spent on communication and workload administration can be completely neglected when estimating the total run–time of the optimisation. Evaluating the fitness should take constant time with minimal variation even with different parameter sets. Therefore it can be said that the total speedup of a parallelised GA compared to a serial one scales linearly with the number of nodes used.

Overall it can be said that running the optimisation in parallel is always advantageous so long as the computational cost of the fitness evaluation massively outweighs the administrative cost of the GA. To gain an even better speed up one could think of a way to include all processes in the evaluation instead of having one process solely do the broadcasting and reception and genetic operations. Also a more dynamic load balancing algorithm based on individual node performance could be devised instead of sending jobs to the next best idling processor.

## 4.4. Software Model

The design of the class model is closely related to the structure of a genetic algorithm. For each entity in the algorithm structure there is one class representing it. In addition to that, auxiliary classes and specialised descendants of the main classes are implemented. This section gives an overview over the existing classes, their main interfaces and a detailed description how to set up an optimisation routine using those classes. The Python language provides all the functionality that allows for generic software code (see Section 4.3.2), i.e. class inheritance and function pointers. Furthermore it has efficient data structures and flexible libraries for standard algorithms.

### 4.4.1. Core Classes and Operators

The core classes are the building blocks for a genetic algorithm. They provide the neccessary interfaces for communication between objects. They are base classes that can be derived for more specialised tasks.

#### 4.4.1.1. BasicIndividual

Objects of this class and its derivatives represent a single individual in a GA. The data stored in `BasicIndividual` is the chromosome, which is an array of undefined type which in itself does not contain any information. In this implementation it is an array of binary values, but the code does not explicitly make that restriction. `BasicIndividual` also stores a fitness value together with the information if this value is valid or has to be reevaluated before usage. It also overwrites the standard comparison functions for classes. Tests for equality and unequality ($=$, $\neq$) internally test for the (un-)equality of the chromosome string. Comparison operators ($<$, $>$, $\leq$, $\geq$) evaluate the relation between fitness values. `BasicIndividual` also holds function pointers for the fitness function and the evaluation function. The latter translates the chromosome (genotype) into its numeric counterpart (phenotype). The signature of an evaluation function takes an object of type `Individual` and returns an arbitrary type dependent on the information coded in the chromosome (for details on chromosome coding see 4.1.1). Available evaluation functions are:

**IntegerValueEvaluate** Converts a string of binary values into an integer in two steps: First it converts the binary into a decimal value, then it maps the resulting decimal to the allowed data range by shifting it such that the lower bounds match. The data range is defined in the `gaDict` (see A.2.1) and might be smaller than the range that is covered by binary strings with the length of the chromosome length. For example, if the allowed data range is $[34, 85]$ the required chromosome length would be calculated as $\lceil \log_2(85 - 34) \rceil = 6$. But using 6 bits the available data range would be $2^6 = 64$. In that case numbers representing values that do not lie inside the interval will be ignored to make sure no values of higher value than would fit into the data range are stored.

**MultiIntegerValueEvaluate** Expects a string of binaries but will return an array of integers. Conversion works as above, but the chromosome is split into sections of lengths defined by the individual data ranges for each integer in the array.

**RealValueEvaluate** A binary string of length $l$ is converted into its decimal equivalent $D$, then this value is mapped to the allowed data range using the equation

$$r_{\text{mapped}} = \frac{(b_U - b_L)}{2^l * D} + b_L \tag{4.6}$$

with $b_L$, $b_U$ being the lower and upper bounds of the data range respectively. In a last step the resulting value is rounded to the decimal accuracy requested in the configuration dictionary.

**MultiRealValueEvaluate** As above, but the input chromosome is first split into blocks, each block representing one real value in an array.

#### 4.4.1.2. BasicPopulation

`BasicPopulation` is an abstract class for the implementation of populations in a GA. It stores an array of individuals, probability parameters for the evolution as well as some book keeping information. More importantly it provides virtual interfaces for genetic operators such as selection, crossover, mutation and generation advancement. All these functions must be assigned or implemented in derived classes to make the population work.

### 4.4.2. Derived Classes

#### 4.4.2.1. Derived from Individual

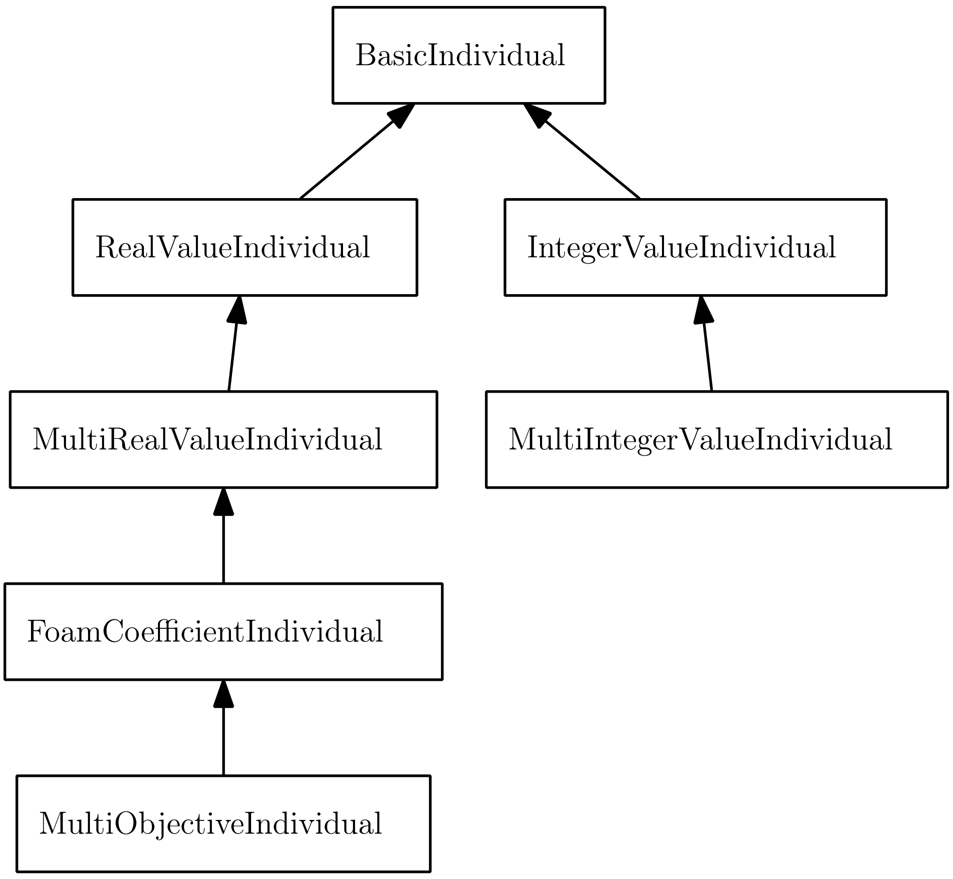


**Figure 4.5.:** Class diagram for the base class `BasicIndividual`.

**IntegerValueIndividual and RealValueIndividual** Derived from `BasicIndividual`. Store lower and upper bounds of the allowed data range as well as automatically assigning the right evaluation function to convert a bit string to integer or real values respectively.

**MultiIntegerValueIndividual and MultiRealValueIndividual** Derived from `Integer-`, `RealValueIndividual`. Store an array of lower and upper bounds of the allowed data ranges for each value and assign the appropriate evaluation function. The `Real` version also stores an array of requested accuracies for each value.

**FoamCoefficientIndividual** Derived from `MultiRealValueIndividual`. Additionally stores names for each value, which makes it easier to use in the context of an OpenFOAM optimisation procedure. The values can be referenced by their position in the array as well as their names. It also introduces a new mandatory index parameter requested upon construction of the object. It enumerates individuals in a population and keeps track of the generation an individual belongs to. This is mainly for statistical purposes.

**MultiObjectiveIndividual** Derived from `FoamCoefficientIndividual`. The fitness function pointer is replaced by an array of function pointers, one for each optimisation objective. Equally the single fitness value is replaced by an array of values. The comparison functions ($<$, $>$, $\leq$, $\geq$) are overwritten and now compare the crowding distance of each individual (see 4.5.1).

#### 4.4.2.2. Derived from Population

**SimplePopulation** Derived directly from `BasicPopulation`. The order of operations is set to be selection, crossover and then mutation. Tournament selection with tournament size 2 and single-point crossover are predefined operators. This is the most common combination of operators used in the optimisation runs presented in this work. Of course all these preset assignments can be changed after instantiation.

**ElitistPopulation** Derived from `SimplePopulation`. In the progression to the next generation, instead of creating $n$ new individuals from $n$ parents, the

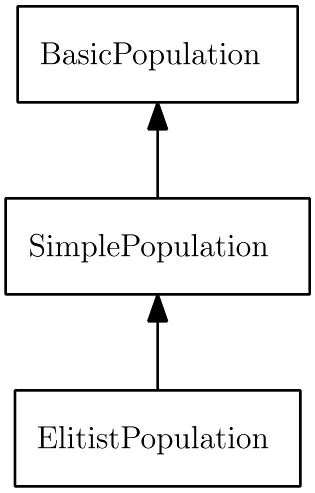


**Figure 4.6.:** Class diagram for the base class `BasicPopulation`.

best two individuals are preserved and automatically advance to the next generation. Therefore only $n-2$ new individuals are created. This has no impact on the selection procedure. This method has proven to be valuable especially towards the converged state of an optimisation, when fitness values tend to lie closer together. Elitism increases the part of the algorithm that is exploiting an existing solution without giving up the diversity and exploration that comes with random reproduction [37].

### 4.4.3. Utility Classes

**Random** This implementation of functions to create random numbers is just a wrapper around Python's regular random number library. It provides convenient methods that are used by the genetic algorithm for generation of normally distributed random numbers. Apart from the basic RNG function `random01` that will return a normally distributed random number in the interval $[0;1)$, it offers a generator for integer values within an interval $[L,H)$ (`randomLoHi(L,H)`) and a biased coin flip that returns 1 with probability $p$, or 0 with probability $1-p$ (`flip(p)`).

**Statistics** The statistics class serves as a repository for performance related data. It provides functions to calculate the average fitness of a population and most importantly stores a history of all chromosomes that have been members of the gene pool at any time during the evolutionary process. That helps to speed up the optimisation considerably, because in later generations some genotypes will appear repeatedly in the population and good individuals will move up to the next generation unchanged. Keeping track of all individuals makes re-evaluation of their fitness unnecessary, saving a lot of computational time.

## 4.5. Multi-Objective Optimisation

It is often of interest to optimise a problem with respect to different objectives. These objectives are not necessarily independent of each other, usually they are even opposing each other. For example a manufacturer wants to optimise his production cycle by maximising the number of items produced per day while at the same time minimising the costs. It is probably easy to find an optimal solution to this two-dimensional problem but with an increase in the number of objectives and with a huge number of influencing parameters, finding such a solution gets more and more difficult and requires the use of heuristic or stochastic methods.

One subset of evolutionary algorithms to deal with that kind of problem that has emerged in the mid-1980s are Multiobjective Evolutionary Algorithms (MOEA). They combine the evolutionary approach discussed in Section 4.1 with algorithms that find a tradeoff between competing objectives. Generally an optimisation problem with $k$ objectives, which are all equally important for the sake of simplicity, is to be solved. Any solution to this problem is represented as a *decision vector* $(x_1, x_2, \ldots, x_n)$ taken from the *decision space* $\mathbf{X}$. A function $\mathbf{f} : \mathbf{X} \rightarrow \mathbf{Y}$, assigns an *objective vector* $(y_1, y_2, \ldots, y_k)$ to the solution in the *objective space* $\mathbf{Y}$ [113]. Different MOEAs have been developed using different ways to assign fitness to a solution that enables the evolutionary operators to work.

In the case of a single objective maximisation problem, i.e. $k = 1$, a solution $\mathbf{x}^1 \in \mathbf{X}$ is better than a different solution $\mathbf{x}^2 \in \mathbf{X}$ if for the assigned objectives

$f(\mathbf{x}^1) > f(\mathbf{x}^2)$ or $\mathbf{y}^1 > \mathbf{y}^2$. Comparison of two solutions in a multiobjective problem is less obvious than in one with only a single objective.

In the case of optimising towards more than one objective, the basic purpose of a fitness function remains unchanged. Instead, several fitness functions are evaluated at the same time and the Pareto criterion is determined by the GA, in this case the NSGA-II algorithm (see 4.5.1). Mathematically speaking all solutions on the Pareto front are optimal solutions. To select the right solution for the actual engineering application can not be automated. For the turbulence model optimisation, for example, looking at the convergence behaviour of each solution might be a good guidance to choose one individual on the Pareto front.

MOEAs make use of a method called Pareto efficiency [37] which defines dominance as a comparison operator: An objective vector $\mathbf{y}^1$ in a maximisation problem strictly dominates a vector $\mathbf{y}^2$, if each objective $y_i^1$ is not strictly less than than objective $y_i^2$ and at least one objective is strictly greater. In other words, if $y_i^1 \geq y_i^2$ for each $i$ and $y_i^1 > y_i^2$ for some $i$ $\mathbf{y}_1$ dominates $\mathbf{y}_2$, written $\mathbf{y}_1 \prec \mathbf{y}_2$. The *Pareto front* is now the set of objective vectors that are not strictly dominated by any other vector. The respective solutions make up the *Pareto set*.

Recent developments calculate the Pareto front of a solution space in situ by sorting the solutions in order of dominance. This class of algorithms is known as Nondominated Sorting Genetic Algorithms (NSGA). Classic NSGAs [21] have been criticised for a number of reasons:

**High computational costs** The cost for the nondominated sorting scales with the population size $N$ and the number of objectives $M$ as $\mathcal{O}(MN^3)$. This leads to expensive calculations of the sorting in large populations. This criticism is of no particular concern for the current work, as the evaluation of the fitness function usually takes considerably longer and therefore the cost for the GA can be neglected.

**Lack of elitism** Classic NSGAs do not preserve the best individual. But research has shown that elitism can speed up the performance of a GA and good solutions will not be discarded [114].

**Introduction of an additional parameter** Traditional methods rely on a sharing parameter $\sigma_{share}$ to ensure diversity in the population. Many suggestions have been made to control this value, but a parameter–free algorithm is always desirable.

### 4.5.1. Fast Non-Dominated Sorting

To overcome the above disadvantages, Deb et al. [22] have developed an improved version of the standard NSGA called NSGA-II. They show that it outperforms existing implementations with respect to efficiency and diversity. The cost improvement of the sorting algorithm is of one order of magnitude, changing it to $\mathcal{O}(MN^2)$. For each generation all individuals of the population are assigned a level representing the Pareto front they belong to. The actual front is level 0, the next level contains all individuals that are Paretooptimal if all level 0 individuals are removed from the population and so forth. This ordering is called non-dominated sorting. Another addition they made to the original algorithm is the introduction of a crowding distance. That is the average side–length of the largest cuboid around a solution that does not contain any other solution. So any solitary solution that has no other individuals in its immediate vicinity would be assigned a large crowding distance, whereas individuals that are clustered together would get a small distance. Using an operator $\geq_n$ that evaluates this value in the selection procedure maintains an equal spread along the Pareto front.

The NSGA-II algorithm works as follows: Initially ($t = 0$) a population $P_0$ is randomly generated. It is then sorted based on the non–domination. Using binary tournament selection, mutation and crossover a new child generation $Q_0$ of size $N$ is formed. From there on the procedure shown in Table 4.1 is repeated until a certain number of generations has been generated or another predefined termination criterion is met.

**Table 4.1.:** Pseudo-code for the NSGA-II algorithm.

```
$R_t = P_t \cup Q_t$                                  combine the parent and child poulations
$\mathcal{F} =$ nondominated-sort$(R_t)$              $\mathcal{F} = \{\mathcal{F}_i\}$ all fronts of $R_t$
**until** $|P_{t+1}| \geq N$                          until new population is full
  assign-crowding-distance$(\mathcal{F}_i)$
  $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$              include i-th nondominated front in P
**Sort**$(P_{t+1}, \geq_n)$                           sort using the the crowding dist operator
$P_{t+1} = P_{t+1}[0 : N]$                            trim population to size N
$Q_{t+1} =$make-new-pop$(P_{t+1})$                    selection, crossover, mutation
$t = t + 1$
```

## 4.6. Benchmarking

In order to prove functionality of my own implementation of a genetic algorithm, the code was tested against a set of benchmarking problems. These are optimisation problems of which the answer is known or can be deduced analytically. For the simple single objective algorithm, deJong [23] suggests a set of benchmark problems, that test different aspects of the GA implementation. These problems are still widely used in order to test new optimisation methods (e.g. [98, 78]). To cover a wide spectrum of solution space topologies, the tests include continous, discontinous, convex, non-convex, unimodal, multimodal, quadratic, non-quadratic, low-dimensional and high-dimensional functions. The function definitions are listed in Table 4.2, along with the parameter ranges and the approximate size of the solution space. In all cases the optimisation target was to minimise the function value. Table 4.3 lists the topological properties of each test function. To test the implementation against these benchmark functions, a common setup of 50 individuals per population evolving over 100 generations was defined. The crossover probability was $P_C = 0.6$ and chance of mutation $p_M = 0.01$ for all tests. Comparison of the real optimum and the one found using the GA implementation presented here is listed in Table 4.4. The statistics were gathered by simulating each problem ten times and averaging the results. Table 4.3 lists the properties of each of deJong's test functions showing the different topological characteristics of the solution space.

**Table 4.2.:** Benchmark minimisation problems to test GA performance as proposed by deJong [23], along with the number of possible solutions given a fixed discretisation of the $x_i$-axis.

| | Function | Limits | No. of Solutions |
|---|---|---|---|
| F1 | $\sum_{i=1}^{3} x_i^2$ | $-5.12 \leq x_i \leq 5.12$ | $\cong 10^9$ |
| F2 | $100(x_1^2 - x2) + (1 - x1)^2$ | $-2.048 \leq x_i \leq 2.048$ | $\cong 1.7 \cdot 10^6$ |
| F3 | $\sum_{i=1}^{5} \lceil x_i \rceil$ | $-5.12 \leq x_i \leq 5.12$ | $\cong 10^{15}$ |
| F4 | $\sum_{i=1}^{30} i x_i^4 + \text{gauss}(0,1)$ | $-1.28 \leq x_i \leq 1.28$ | $\cong 10^{72}$ |
| F5 | $\left[0.002 + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$ | $-65.536 \leq x_i \leq 65.536$ | $\cong 1.6 \cdot 10^{10}$ |

Figures 4.7 to 4.11 show plots of the solution space and projections of contour lines to point out the location of local minima. In case of high-dimensional problems F3 and F4, the space for the corresponding two-dimensional function is shown. The matrix A in function F5 was defined as

$$(a_{ij}) = \begin{pmatrix} -32,-16,\ 0\ ,\ 16\ ,\ 32\ ,-32,-16,\ldots,\ 0\ ,16,32 \\ \\ -32,-32,-32,-32,-32,-16,-16,\ldots,32,32,32 \end{pmatrix}$$

F5 is basically a plane of constant value 500, with 25 local minima centered about $(a_{1j}, a_{2j})$, where it takes the values $1, 2, \ldots, 25$. Finding the global optimum with gradient based methods is rather unreliable in this case. For most of the function space has no gradient at all and any of the local minima 'foxholes' will be identified by these classical methods as the optimum. The non-deterministic character of a GA has a higher rate of success as the results in Table 4.4 show, thanks to a more wide spread exploration of the solution space. The table actually compares simulated results from ten GA runs with analytical solutions of the benchmark problems. The first two columns show the average optimum found by the algorithm and the standard deviation over the ten runs. The third column is the best possible solution, i.e. the optimum found by mathematical analysis of these test functions.

**Table 4.3.:** Topological characteristics of the solution space for test functions F1-F5.

| | F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|---|
| continous | √ | √ | | √ | √ |
| discontinous | | | √ | | |
| convex | √ | | | √ | |
| non-convex | | √ | √ | | √ |
| unimodal | √ | √ | √ | √ | |
| multimodal | | | | | √ |
| quadratic | √ | √ | | √ | |
| non-quadratic | | | √ | | √ |
| low-dimensional | √ | √ | √ | | √ |
| high-dimensional | | | | √ | |

The last column just shows the worst possible solution to give an idea of the ranges in solution space.

Proximity to the optimal value and the 'hit–rate', i.e. the standard deviation, are a measure of the quality of the results. It can clearly be seen that for test functions F1 to F3 the performance of the GA was very good, in the case of F3 it even found the optimal solution in every single run. In F4 the results are still very good, considering that the solution space is several orders of magnitude larger than in the other problems, as shown in Table 4.2. In F5 it seems to be difficult to find the real minimum, even for a non-deterministic search algorithm as a GA. The problem might be that the minima are very localised in the solution space and small movements in parameter space lead to great jumps in the function value. This observation was also made by Goldberg [38] when investigating the benchmark problems. Unfortunately, in case of turbulence model parameters, the topology of the solution space is unknown. A rigorous mathematical analysis could be undertaken which is beyond the scope of this thesis but might be considered for future work. The benchmark results presented here are in good agreement with Goldberg's findings, proving the validity of the GA implementation.

**Table 4.4.:** GA benchmark results after ten simulations compared to real optima.

| | Simulation | | Solution | |
|---|---|---|---|---|
| Function | avg. | stdDev | min | max |
| F1 | 0.00041 | 0.00025 | 0 | 78.64 |
| F2 | 0.15704 | 0.30686 | 0 | 3905.93 |
| F3 | -25 | 0 | -25 | 25 |
| F4 | 0.36352 | 0.18248 | 0 | 1248.2 |
| F5 | 3.4045 | 4.59825 | $\approx 1$ | $\approx 500$ |

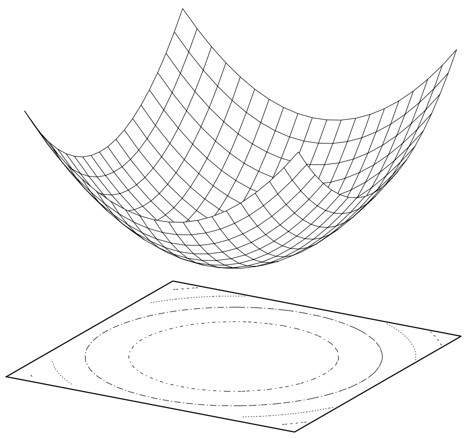


**Figure 4.7.:** deJong benchmark function F1

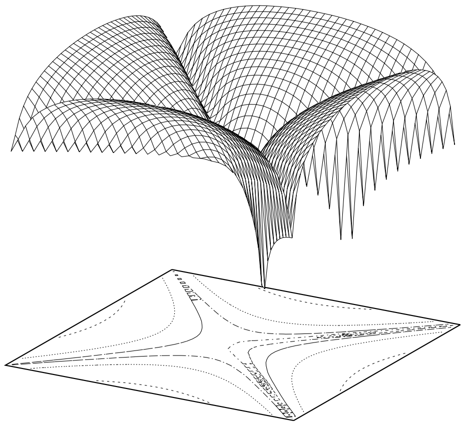


**Figure 4.8.:** deJong benchmark function F2

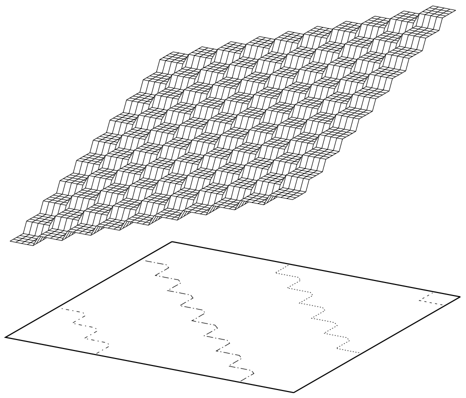


**Figure 4.9.:** deJong benchmark function F3

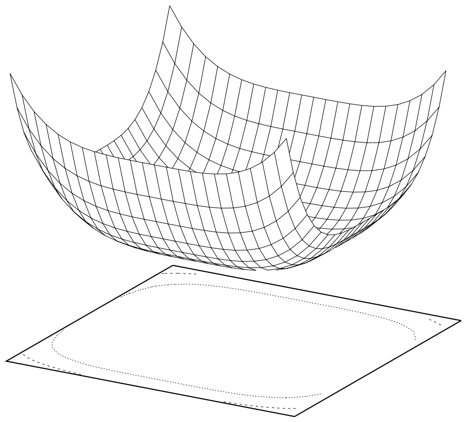


**Figure 4.10.:** deJong benchmark function F4

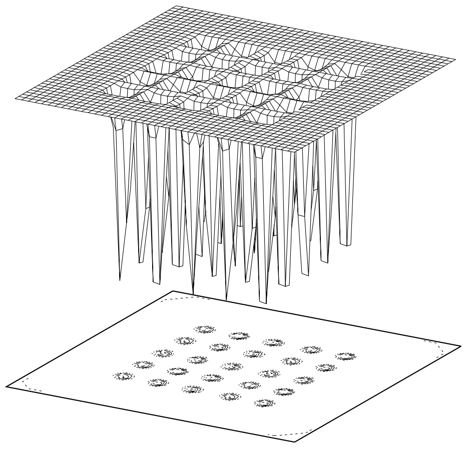


**Figure 4.11.:** deJong benchmark function F5

## 4.7. Example: Parameter Identification

### 4.7.1. Motivation

To prove that the results of a genetic optimisation are valuable to various fields of science, this implementation was used to generate solutions to a problem from neuro science. During the work on this thesis the opportunity arose to participate in a collaboration of medical scientists, mathematicians and computer scientists in research on photo–sensitive epilepsy. The idea to use a genetic algorithms for parameter identification was employed to advance the current knowledge on brain wave activity in epileptic patients. The data base used so far, like EEG (electroencephalogram) imagery and frequency analysis, failed to deliver insight into the differences between healthy and epileptic patients. A new method fopr parameter identification was sought. For the benefit of the research project and of this thesis it was possible to try out the GA implementation on a real world optimisation problem. This provided a good test environment with less computational cost than optimising turbulence model coefficients (see Chapter 5), but with a similar model setup.

The dynamics of the evolution of focal onset epileptic seizures in photo–sensitive patients is not yet understood. It is believed, that it initiates in an 'abnormal' brain region and propagates by employing connections to 'normal' regions. It can therefore be assumed that focal seizure activity can best be described by a model that includes a network of interconnected neuronal regions. To identify connection between clinically observed features and the structure of the measured EEG was part of the aim of a study by Blenkinsop et al [12]. On the clinical side they studied three groups of patients: photo-sensitive epileptic patients, epileptic patients, and a healthy control group. In a series of experiments they confronted the patients with a seizure-inducing stimulus and measured their brain activity with standard EEG methods. By comparing the activity curves they tried to reveal the dynamical evolution of focal-onset epilepsy and further aimed to identify the neurophysical properties of the affected brain regions by reverse engineering the parameters of a neuron-population based mathematical model. Identifiying the parameter values for this model that caused the effects observed in experimental data was the task for the genetic algorithm. In further research, bifurcation analysis of the model has led to a map of the parameter space that clearly identifies regions of distinguishable brain wave shape. With the help of the genetic algorithm it could be shown, that brain activity in epileptic patients coincides with one specific type of bifurcation.

## 4.7.2. Neuronal Model

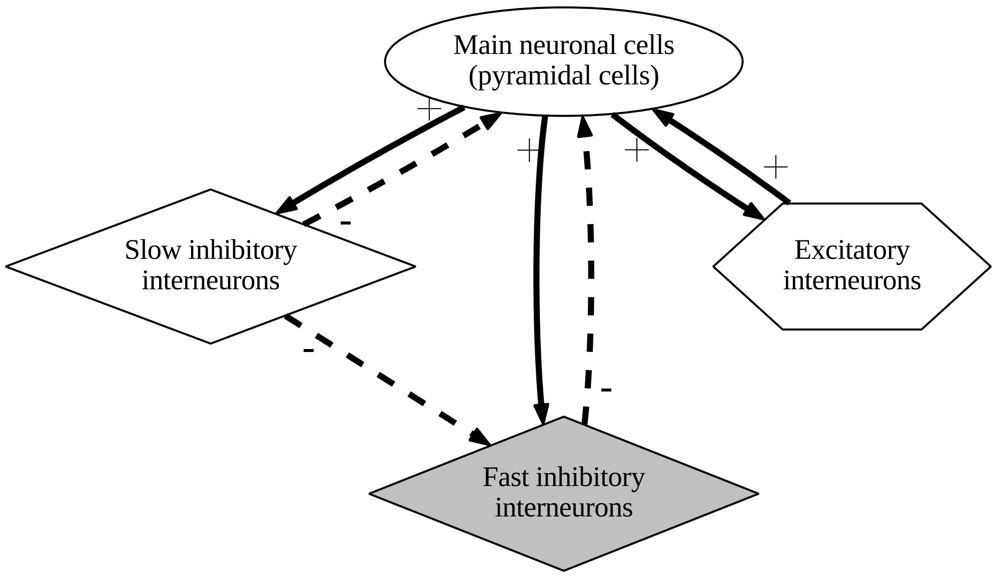

Main neuronal cells (pyramidal cells)
+
-
+
+
+
Slow inhibitory interneurons
Excitatory interneurons
-
-
Fast inhibitory interneurons

**Figure 4.12.:** Schematic representations of the neural model proposed by Wendling [106]. Edges represent interaction between the populations, where solid edges are excitatory and dashed edges are inhibitory feedback.

There are several ways how to best model the oscillatory behaviour of neuronal populations observed in the brain with different levels of granularity. On the cellular (or 'detailed' [109]) level a large number (i.e. several thousands) of single neurons are modelled seperately, together with their structural and functional properties. Neurons are then clustered into networks and EEG activity can be studied in detail with respect to neuron type, network size, connectivity patterns and so forth. A broader approach would be the 'reduced' model [107] with a smaller number of neurons, which studies the dynamic behaviour of the networks. Above this is the population level model as used in this work. According to Wendling et. al [106] this approach has been succesfully used in the past to reconstruct measured EEG data with a rather simple model of neuron populations that interact with each other in a driving (excitatory) or suppressing (inhibitory) fashion. The dynamics of these interactions can be described by second order differential equations with a static nonlinearity in form of an asymmetric sigmoid curve. Some model parameters are deduced from actual physical measurements, others are of statistical nature. In the Wendling model four populations are represented as shown in Figure 4.12. The main subset of neurons contains the main cells in the hippocampus or neocortex. This receives feedback from three minor subsets comprised of local interneurons either excitatory or inhibitory. Neurologically one inhibitory feedback loop links to the pyramidal cells via dendrites while the other exhibits a more direct connection via the somae (nerve cells). This results in two distinct time scales in the model. Any signal fed into a population will cause a membrane potential that is then translated into a pulsed output by the sigmoid function

$$S(v) = 2e_0/[1 + \exp(r(v - v_0))]. \tag{4.7}$$

The parameters identified in this model are connection strengths $C_1 - C_7$ between populations (one per edge) as well as synaptic gains (or signal strengths) $A$, $B$ and $G$, for the excitatory, slow inhibitory and fast inhibitory population respectively. It is also driven by an underlying background Gaussian white noise $p(t)$ fed into the main cell population that ensures initial activity of the feedback loops. The time scales are determined by the answer times of the loops $a$, $b$ and $g$. Further does the nonlinear asymetric sigmoid function contain the parameters $v_0$, $e_0$ and $r$. Model output is the aggregated activity of all feedback loops and reflects an EEG signal.

Transforming the set of second order differential equations into pairs of first order linear differential equations delivers this set of ten governing terms of the model:

$$\dot{y_0}(t) = y_5(t) \tag{4.8}$$

$$\dot{y_5}(t) = AaS[y_1(t) - y_2(t) - y_3(t)] - 2ay_5(t) - a^2y_0(t) \tag{4.9}$$

$$\dot{y_1}(t) = y_6(t) \tag{4.10}$$

$$\dot{y_6}(t) = Aa(p(t) + C_2 S[C_1 y_0(t)]) - 2a y_6(t) - a^2 y_1(t) \tag{4.11}$$
$$\dot{y_2}(t) = y_7(t) \tag{4.12}$$
$$\dot{y_7}(t) = BbC_4 S[C_3 y_0(t)] - 2b y_7(t) - b^2 y_2(t) \tag{4.13}$$
$$\dot{y_3}(t) = y_8(t) \tag{4.14}$$
$$\dot{y_8}(t) = GgC_7 S[C_5 y_0(t) - y_4(t)] - 2g y_8(t) - g^2 y_3(t) \tag{4.15}$$
$$\dot{y_4}(t) = y_9(t) \tag{4.16}$$
$$\dot{y_9}(t) = BbS[C_3 y_0(t)] - 2b y_9(t) - b^2 y_4(t) \tag{4.17}$$

Interpretation of the coefficients and proposed values to produce healthy patient EEG signals are given in Table 4.5. The development of the signal over time is shown in Figure 4.13. It depicts the measured electric potential over time during the stimulation and, in case of the epileptic patients, the seizure onset. With the naked eye it is difficult to spot a difference between healthy patients (left figure) and epileptic patient (right figure). In both cases the stimulation starts at t=0.3 s and a reaction in the epileptic signal can be noted after $\approx$ 0.5 s by an increase in electrical activity. A clearer difference can be observed when looking at the frequencies present in the signal using a fast Fourier transformation (FFT). This is shown in Figure 4.14. In the case of the epileptic patient the power at Frequency 5-7 Hz (*alpha wave*) is much higher than in the healthy control and it is lacking contributions in the higher spectrum range.

The task for the genetic algorithm in this scenario is to identify parameter values for the neuronal model that match the measured EEG data sets. These values should then give the researchers an idea about the difference synaptic signal strengths between healthy and epileptic subjects and how they effect the susceptibility to photonic stimulation.

### 4.7.3. Problem Formulation

The optimisation problem targeted in this investigation was to identify those values for the model parameters A, B and G which best generate the sort of brain wave activity measured on the patients in the medical experiments. To quantify the difference between experimental and numerical data, the frequency spectrum was

**Table 4.5.:** Neurophysical interpretation of parameters in the population model by Wendling *et. al.* [106]. Standard values were established in Jansen and Rit [48].

| Parameter | Interpretation | Standard value |
|---|---|---|
| A | Average excitatory synaptic gain | $3.25\,\mathrm{mV}$ |
| B | Average slow inhibitory synaptic gain | $22\,\mathrm{mV}$ |
| G | Average fast inhibitory synaptic gain | $10\,\mathrm{mV}$ |
| $1/a$ | Dendritic average time constant in the excitatory loop | $a = 100\,\mathrm{s}^{-1}$ |
| $1/b$ | Dendritic average time constant in the slow inhibitory loop | $b = 50\,\mathrm{s}^{-1}$ |
| $1/g$ | Somatic average time constant in the fast inhibitory loop | $g = 500\,\mathrm{s}^{-1}$ |
| $C_1$, $C_2$ | Average number of contacts in the excitatory loop | $C_1 = C$, $C_2 = 0.8C$ (with $C = 135$) |
| $C_3$, $C_4$ | Average number of contacts in the slow inhibitory loop | $C_3 = C_4 = 0.25C$ |
| $C_5$, $C_6$ | Average number of contacts in the fast inhibitory loop | $C_5 = 0.3C$, $C_6 = 0.1C$ |
| $C_7$ | Average number of contacts between inhibitory neurons | $C_7 = 0.8C$ |
| $v_0$, $e_0$, $r$ | Parameters of the nonlinear sigmoid function $S$ | $v_0 = 6\,\mathrm{mV}$, $e_0 = 2.5\,\mathrm{s}^{-1}$ $r = 0.56\,\mathrm{mV}^{-1}$ |

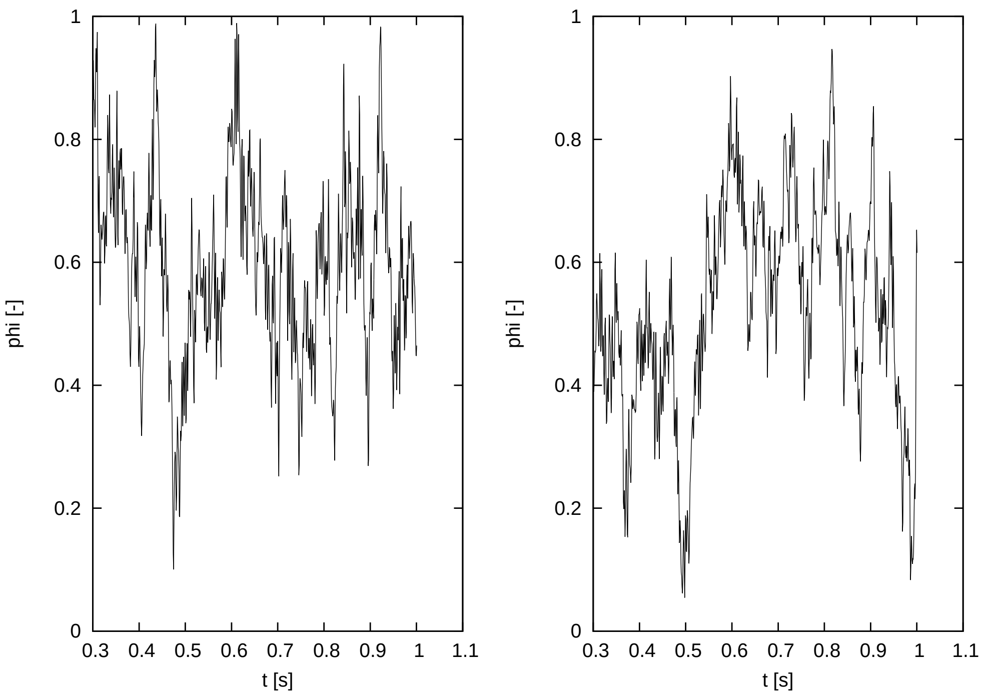


**Figure 4.13.:** Comparison of measured brain wave activity in healthy (left) and photo–sensitive epileptic (right) patients. The measured intensity phi is normalised with the maximum intensity.

calculated for both EEGs and the root mean square error between the data points was the outcome of the objective function.

Range constraints imposed on the decision variables are given in Table 4.6. The resolution of the solution space was 0.1 for all three parameters.

**Table 4.6.:** Value constraints for the objective function in the neuronal model optimisation.

| Parameter | min value | max value |
|---|---|---|
| A | 3.0 | 7.0 |
| B | 5.0 | 25.0 |
| G | 5.0 | 15.0 |

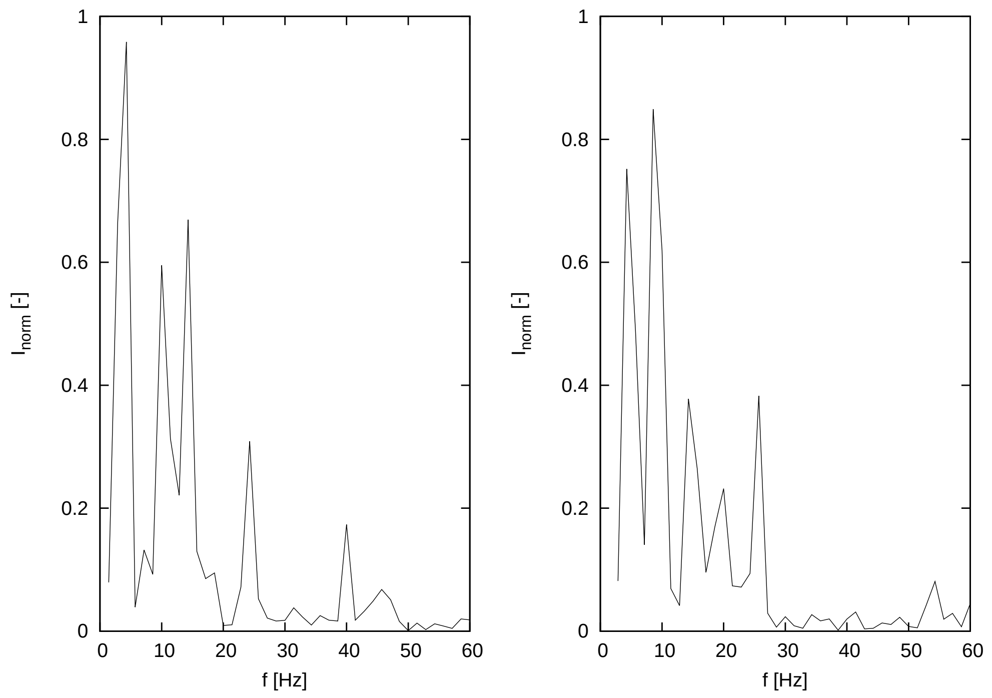


**Figure 4.14.:** Spectral analysis of the brain wave activity in healthy (left) and photo–sensitive epileptic (right) patients. Power is normalised with the maximum power observed.

#### 4.7.3.1. Fitness Function

The workflow of creating data using the model described above was as follows: The model parameters the researchers were interested in were the synaptic gains for the three neural connections A, B and G. A time series was produced solving the set of differential equations that would represent an artificial EEG signal. Using FFT, a spectral analysis was then performed on this signal and compared with the frequency spectrum of the real data. Deviations were measured using a root mean square algorithm. The magnitude of this deviation was used as a fitness value for the GA. Over the course of 40 generations with 100 individuals the parameter configuration that best mimicked the real EEG data was the solution of the optimisation procedure.

### 4.7.4. Results

In total the neuronal activity of 14 patients was measured; 8 photo-sensitive epileptics and 6 healthy control subjects. For each patient a series between 70 and 92 EEGs were recorded with identical temporal exposure to the stimulus. The GA identified a solution for the values A, B and G for each of these data sets, so a total number of 1184 runs were neccessary. An interesting observation was, that the solutions for one patient were usually not localized in the solution space but rather covered a clearly bounded sub-space. Figure 4.15 shows the positions of all solutions found for one patient, plotting the optimal values of parameters B over A. Again the healthy control subject is shown on the left while the epileptic patient is shown on the right hand side.

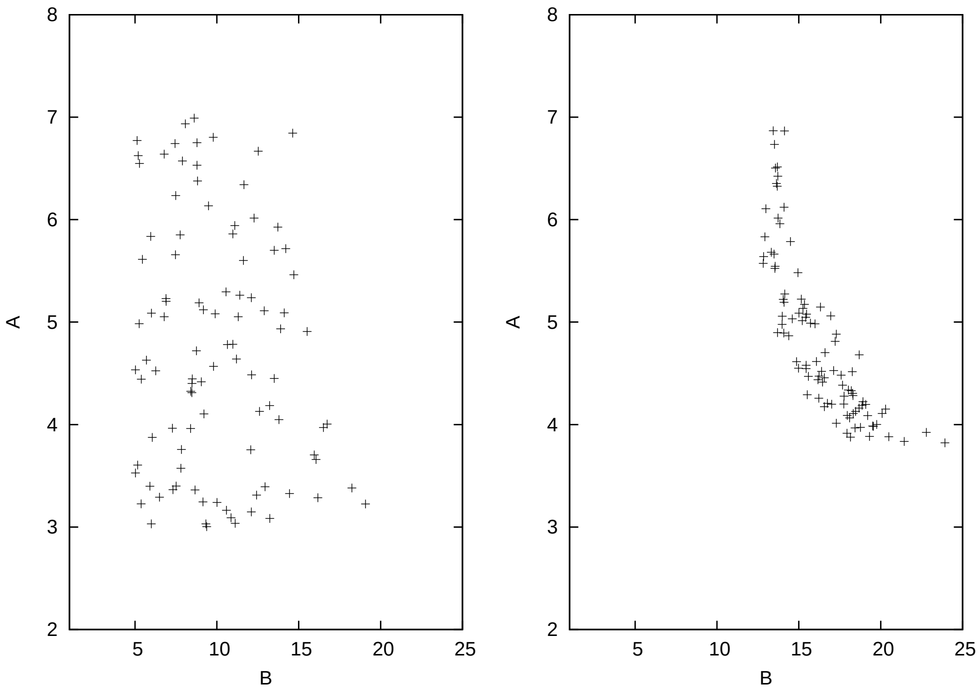


**Figure 4.15.:** Optimal parameter values for synaptic gains A and B for each EEG measurement in a healthy (left) and an epileptic (right) test subject.

It was observed, that for the epileptic subjects the solutions appeared to cluster around the Hopf bifurcation line as identified for this set of differential equations. In comparison, the solutions for healthy patients did not seem to be attracted to any particular area of the solution space. Figure 4.16 shows the identified bifurcation regions and on top the solutions found by the genetic algorithm.

This finding was surprising for the researchers involved in the project and hopefully some conclusions can be drawn that can in future help with the treatment of this particular type of epilepsy. This work shows, that genetic algorithms can be used as a tool to find parameter configurations, that would be very difficult to find with common data fitting methods. The results also proved the reliability of the GA implementation developed in the scope of this thesis, as it was intended to do when this project was approached.

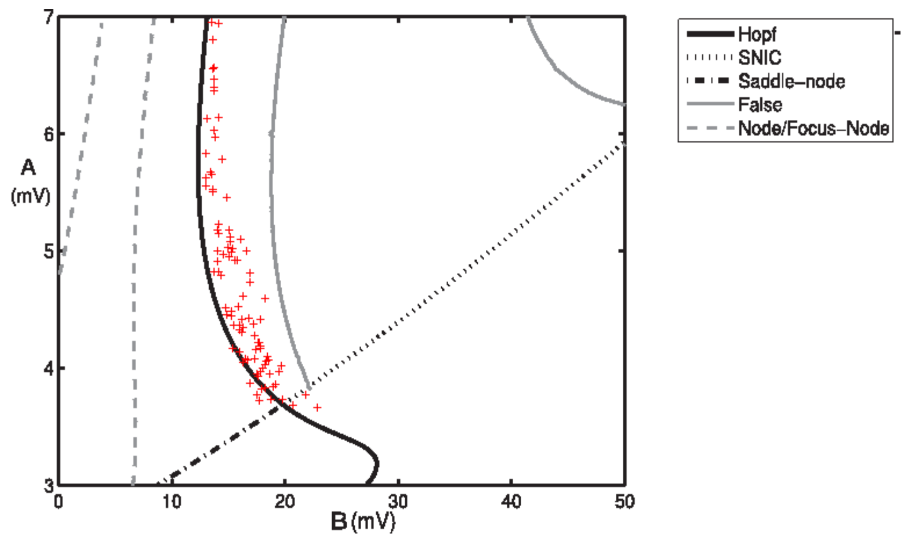


**Figure 4.16.:** Projection of the parameter values obtained by the genetic algorithm (red crosses) onto the bifurcation plane for the neuronal model described by Wendling [106].

# 5. Optimisation of Turbulence Models

*The k-ε model is a highly sophisticated exercise in interpolating between data sets.*

P.A. Davidson

## 5.1. Optimisation Objectives

The criterion that determines the fitness of an individual in a genetic algorithm is the fitness function (see Chapter 4.2). The higher the value calculated by the objective function (or lower, depending on the problem formulation), the higher the chances for this individual to reproduce and progress to the next generation.

In the case of turbulence model coefficient optimisation, the fitness is calculated by comparing experimental data to simulation results. For each experimental data point available, the corresponding value from the simulation is determined either by direct sampling, or in case the measured data points do not agree with the grid points, a linear interpolation is calculated using Eqn. 5.2. The objective value is the square root of the sum $S$ of squared residuals (root square error RSE) between actual data and model:
Let $(x_i, y_i) \in (\mathbf{X}, \mathbf{Y})$, $i = 1 \ldots n$ a number of $n$ experimentally retrieved data points. Further, let $(\hat{x}_j, \hat{y}_j) \in (\hat{\mathbf{X}}, \hat{\mathbf{Y}})$, $j = 1 \ldots m$ be a number $m$ of data points on the computational grid. $S$ is now calculated from all grid points $\hat{x}_j$ using

$$S = \sqrt{\sum_{i=1}^{n} (y_i - y_i^\star)^2}, \quad \text{with} \tag{5.1}$$

$$y_i^\star = \begin{cases} \hat{y}_j \in \hat{\mathbf{Y}} \quad \text{if} \quad \exists\, \hat{x}_j = x_i \\ \frac{\hat{x}_j - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i) + y_i \quad \text{else} \quad \text{with}\, x_i = \max(x \in \mathbf{X} \mid x_i < \hat{x}_j) \end{cases} \tag{5.2}$$

The value calculated with these equations is a direct measure of agreement between experimental (reference) data and computational results. The quantity $Y$ can be any scalar property that can be derived from the simulation. The root square error is a common method in regression analysis to quantify the difference between a discrete data set and a continuous approximation .

Usually, experimental data contains a margin of error due to measuring inaccuracies and precision limitations. CFD simulation cannot reproduce this error, but its deterministic nature will always produce the same results if the calculation was repeated multiple times. In the optimisation procedure discussed in this chapter, an optimal solution will therefore be evaluated against mean values of experimental observations. Numerical treatment of the governing equations in CFD introduces yet another error to the results, since an algebraic solution is not possible. Optimisation of uncertain data sets is not within the scope of this thesis and should be discussed in future work on this problem.

### 5.1.1. Multi-Objective Optimisation

In the case of optimising towards more than one objective, the basic purpose of a fitness function remains unchanged. Instead several fitness functions are evaluated at the same time and the Pareto criterion is determined by the GA, in this case the NSGA-II algorithm (see 4.5.1). Mathematically speaking all solutions on the Pareto front are optimal solutions. To select the right solution for the actual engineering application cannot be automised as was mentioned in the previous chapter.

### 5.1.2. Hardware

For all subsequent test cases the same hard- and software were used to perform the optimisation. CFD simulations were done with OpenFOAM version 2.1.1, while the genetic algorithm was implemented especially for the purpose of this thesis. The workstation contained two six-core AMD Opteron 2427 CPUs (2200 MHz) and 8 GB of RAM. The operating system was OpenSuSE Linux 12.1 “Asparagus”.

## 5.2. Test Cases

### 5.2.1. Backward Facing Step

One of the classical benchmark test cases for turbulence models is the flow over a backward facing step [36, 33]. It is interesting for a variety of reasons. First, the separation caused by the abrupt change of geometry is often found in real engineering applications. Second, it can be seen as an extreme example of the type of flow that occurs at high-lift airfoils at large angles of attack. Even though the cause of the separation in these two cases is different (geometric change in the step case versus adverse pressure gradients in the case of the airfoil) the topologies of both flows are comparable. Further, the instability of the flow is not yet fully understood and the backward facing step offers a non–trivial yet simple setup for more fundamental investigations.

The geometry used here for the turbulence model verification and optimisation is shown in Figure 5.1. The sketch is not to scale, in fact the channel downstream of the step had a length of 11.4 times the step height $H$. The expansion ratio, the ratio between channel height and step height, was $h/H = 2$. This geometry was used because it matched the experimental setup by Makiola [65], which served as the data basis for the optimisation process.

#### 5.2.1.1. Boundary Conditions

In order to mimic the experimental setup as closely as possible, the boundary conditions for the flow were chosen as described by Makiola [65], even though the results were made non-dimensional using the step height $H$ and the maximum inlet

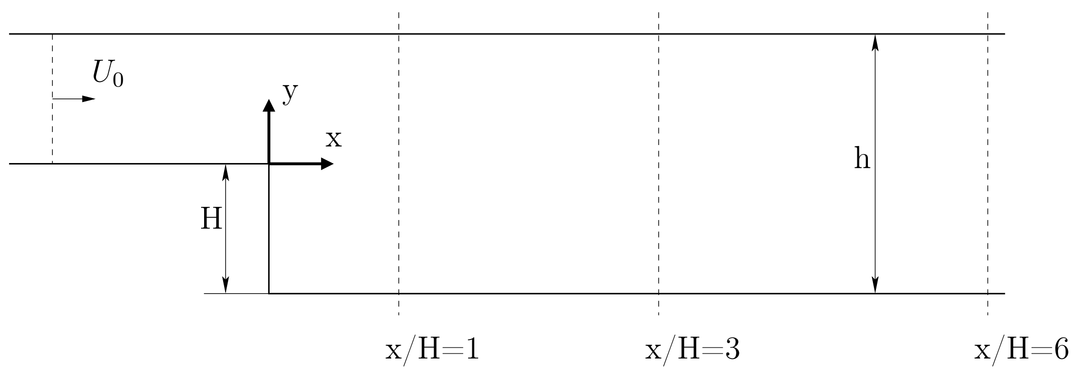


**Figure 5.1.:** Geometry of the backward facing step test case. The dashed vertical line at $x/H = 3$ is an example of one of the lines where flow characteristics were sampled for comparison with experimental data.

velocity $U_0$. Simulations were computed at two different Reynolds numbers $Re = 15,000$ and $Re = 64,000$ and both the standard k-$\varepsilon$ [52] and the k-$\omega$ turbulence model [110] were applied. The Reynolds number was computed as follows

$$\mathrm{Re} = \frac{HU_0}{\nu}, \qquad \nu = \frac{\mu}{\rho}. \tag{5.3}$$

$\nu$ is the kinematic viscosity, the quotient of dynamic viscosity $\mu$ and fluid density $\rho$. The chosen fluid properties were those of air at room temperature.

A grid convergence study was performed using three different grid sizes. The coarsest grid $G_1$ had 3410 cells, the medium grid $G_2$ had 12,885 and the fine grid $G_3$ had 51,540 cells. Although the results from the medium and the fine grid were not significantly different, all simulations were run on the fine grid to avoid grid effects when changing the parameters of the turbulence model.

The top and bottom enclosure and the step of the channel were treated as walls with a non-slip boundary condition for $U$ and a gradient of zero in normal direction, i.e. $\partial p/\partial n = 0$ for pressure. At the outlet pressure was fixed to a reference pressure value, nominally zero. Outlet velocity was set to zero gradient. At the inlet the pressure boundary condition was of type zero gradient and the velocity had a parabolic profile with a value of zero at the walls and a peak value of $U_0$ in the center of the inlet. The turbulent quantities $k$, $\varepsilon$ and $\omega$ were calculated using the following equations [102]

$$k = \frac{2}{3}(U_r I)^2 \tag{5.4}$$

$$\varepsilon = \frac{C_\mu^{3/4} k^{3/2}}{\ell} \tag{5.5}$$

$$\omega = \frac{\sqrt{k}}{C_\mu^{1/4} \ell} \tag{5.6}$$

The quantity $I$ is a measure for the turbulent intensity of the flow and is set somewhere between 0.01 for low Reynolds flow and 0.1 for high intensity turbulence. In this work the upper limit of 10% was selected. The reference velocity $U_r$ in this case was equalt to the maximum inlet velocity $U_0$. The turbulent mixing length $\ell$ is assumed to be $0.07H$, the constant 0.07 being the maximum value of mixing length in fully developed turbulent pipe flow. $C_\mu$ is a parameter of the turbulence model. Table 5.1 lists the boundary conditions as they appear in the OpenFOAM case setup. Dirichlet boundary conditions prescribe a fixed value for the solution on the boundary, while von-Neumann conditions prescribe a value for the spatial gradient (here equal to zero, if not stated otherwise).

**Table 5.1.:** Boundary conditions in the backward facing step test case

| | **inlet** | **outlet** | **walls** |
|---|---|---|---|
| U | $U(y) = -(y - H/2)^2 + U_0$ | von-Neumann | Dirichlet |
| p | von-Neumann | Dirichlet $p_0$ | von-Neumann |
| k | Dirichlet eqn. 5.4 | von-Neumann | wall function |
| $\varepsilon$ | Dirichlet eqn. 5.5 | von-Neumann | wall function |
| $\omega$ | Dirichlet eqn. 5.6 | von-Neumann | wall function |

The equations are solved using a steady–state incompressible solver `simpleFoam`. The total number of simulation steps was 2000 iterations, after which the solution is fully converged. A target residual error of $10^{-2}$ for the pressure equation and $10^{-3}$ for velocity and the turbulent quantities was used to determine convergedness and these values were usually reached after less than 1000 iterations. The residuals are chosen relatively high when compared to simulations performed by other researchers [101, 102], who generally set them an order of magnitude lower. But with respect to computational cost and the simplicit of the problem, the higher values sufficed. To account for changes in convergence behaviour when modifying the turbulence parameters, 2000 generations were thought to be sufficient to ensure convergence in any case. A convergence analysis of each modified case was not made, but it can be assumed that non-converging solutions would produce worse fitness values.

#### 5.2.1.2. Preliminary Studies

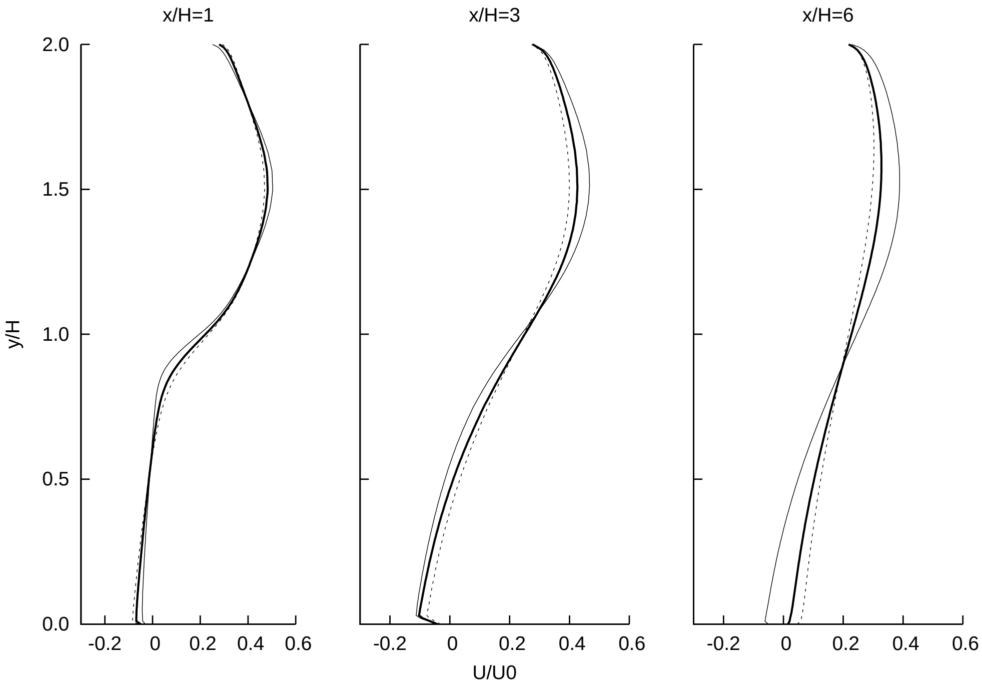
Before running the optimisation algorithm on the full set of parameters in the turbulence models considered for this test case, a study of the influence of each single parameter on the evolution of the flow field was performed. Two simulations were run for each coefficient changing its value by 60% in each direction and the results compared with those obtained by using the standard values (as proposed by the model's authors). The value of 60% was chosen arbitrarily just to ensure that the changes would have a significant impact. This requires at least two simulations per parameter, sometimes more if intermediate values are to be investigated. For example to test the influence of the five parameters in the k-$\varepsilon$ model, ten runs of the backward facing step case were performed and compared to the results achieved by using the standard values (second column in Table 5.2).

**Table 5.2.:** Parameter values for variation study in the k-$\varepsilon$ model

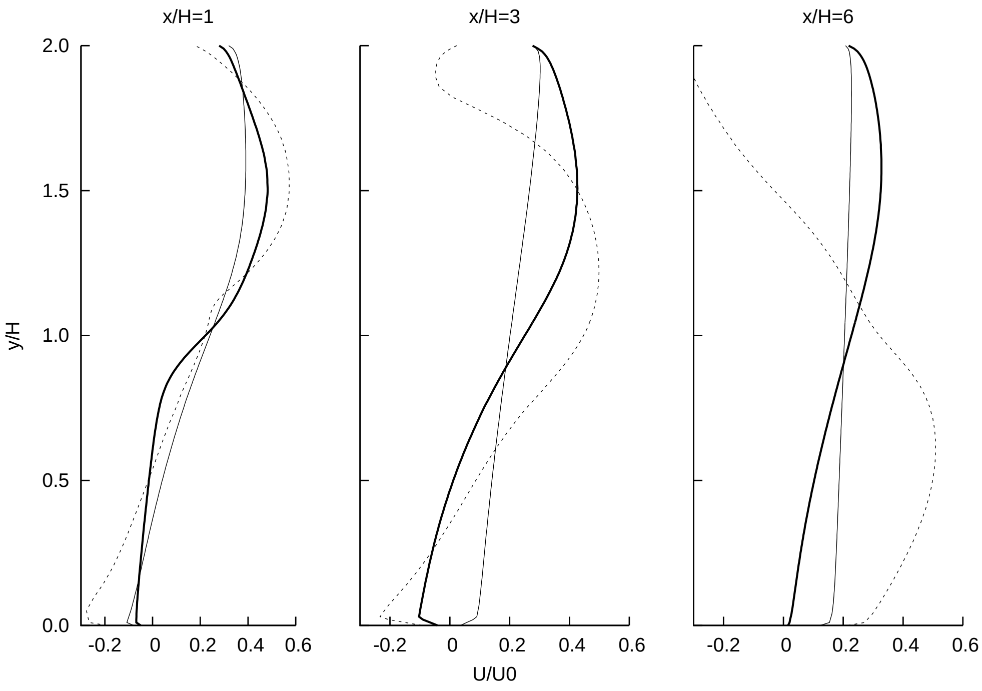
| | -60% | std. | +60% |
|---|---|---|---|
| $\sigma_k$ | 0.4 | 1.0 | 1.6 |
| $\sigma_\varepsilon$ | 0.52 | 1.3 | 2.08 |
| $C_1$ | 0.58 | 1.44 | 2.30 |
| $C_2$ | 0.77 | 1.92 | 3.07 |
| $C_\mu$ | 0.04 | 0.09 | 0.14 |

Figure 5.2 shows velocity profiles at three different positions downstream of the step. The curves each apply to one parameter setting, either incrementing or decrementing the standard value. For parameter $\sigma_\varepsilon$ almost no perceptible change in the flow pattern can be observed, and for turbulent viscosity $C_\mu$ only a moderate influence on the flow is observed. It seems for this particular test case that the influence of these parameters is negligible. On the other hand the linear factors $C_1$ and $C_2$ that serve as weights to the production and dissipation of turbulent kinetic energy have, as would be expected, a significant influence on the development of the flow in the recirculation region downstream of the step. Looking at the graphs also reveals that large modifications of up to 60% can make the resulting flow field become physically unfeasible. For example, when looking at the velocity profiles for parameters $C_1$ and $C_2$, large modifications seem to produce a second recirculation zone on the top wall of the channel. Figure 5.3 shows the results of an identical study for the k-$\omega$-SST model coefficients.

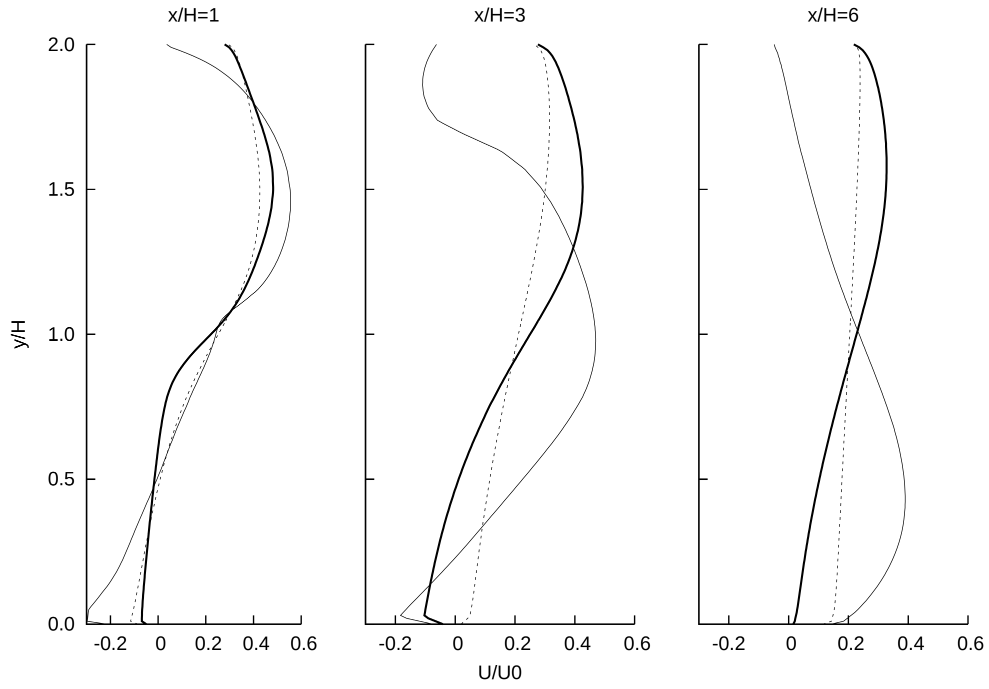
Studying separate parameters in that way reveals information that can be used in the setup of the genetic algorithms. It determines which parameters should be subject to optimisation and gives a general idea of the value range these should fall in. Including non-influential coefficients in the optimisation process tended to prevent the algorithm from converging towards a best solution. That is easily explained, as the fitness value, i.e. the difference between simulated flow field and experiment, does not change even though a parameter might undergo large transformations. It is then impossible for the GA to determine which result is fitter as the selection operator will not give preference to a solution but will treat a set of solutions equally.



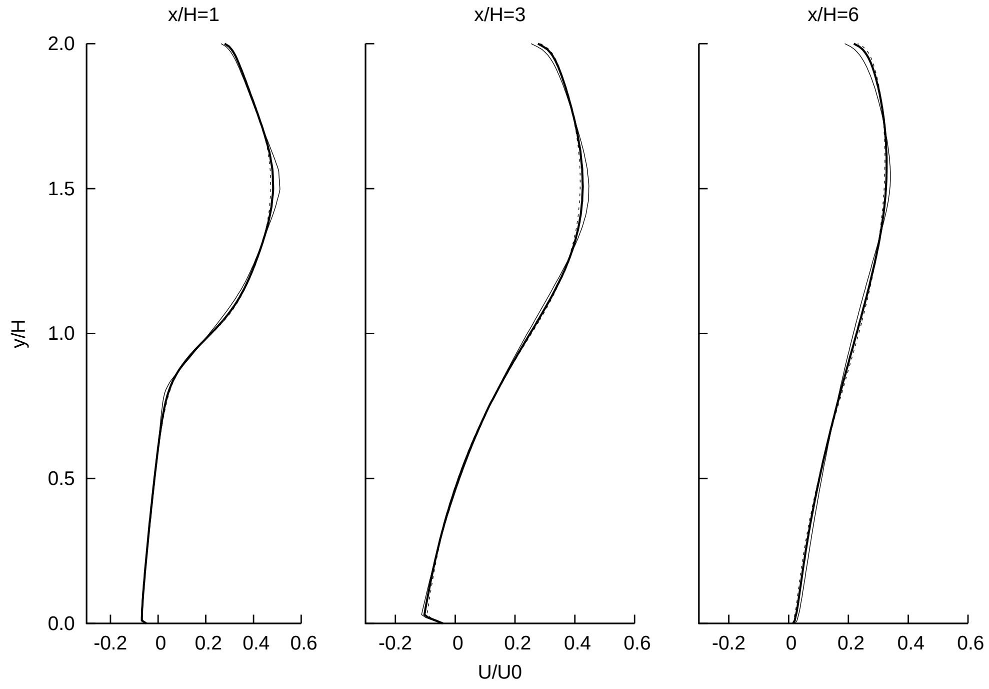
**(a)** Variation study for $C_\mu$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



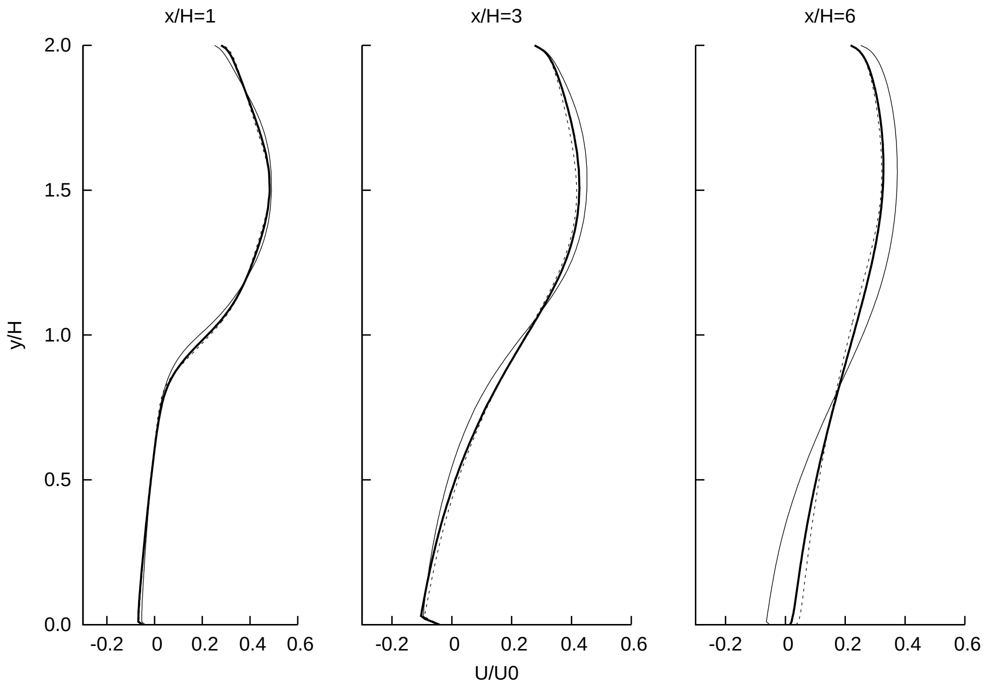
**(b)** Variation study for $C_1$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(c)** Variation study for $C_2$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

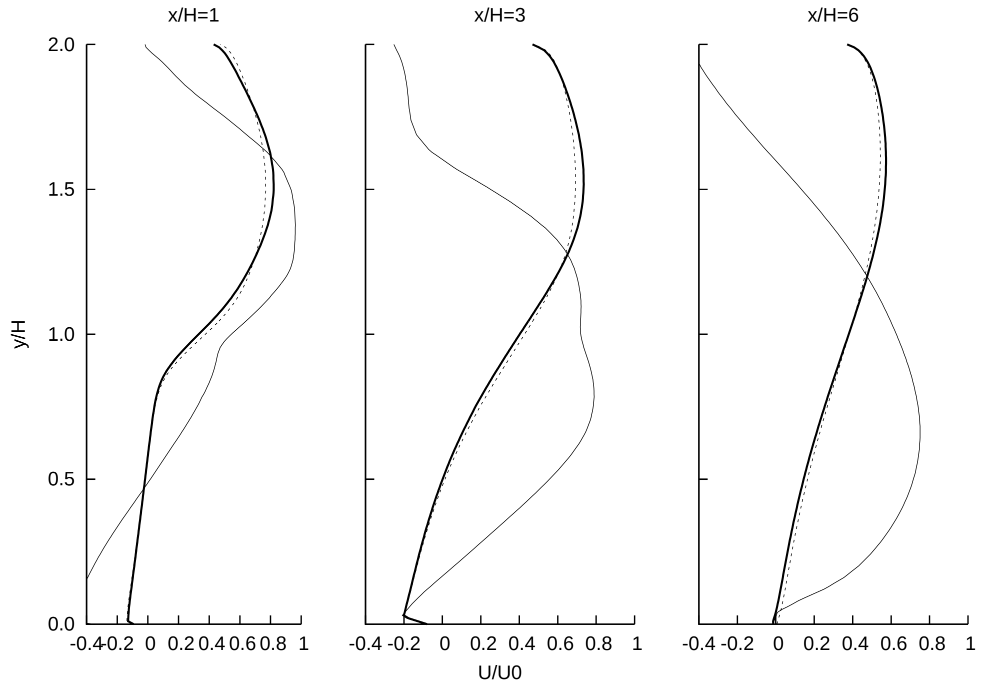


**(d)** Variation study for $\sigma_\varepsilon$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

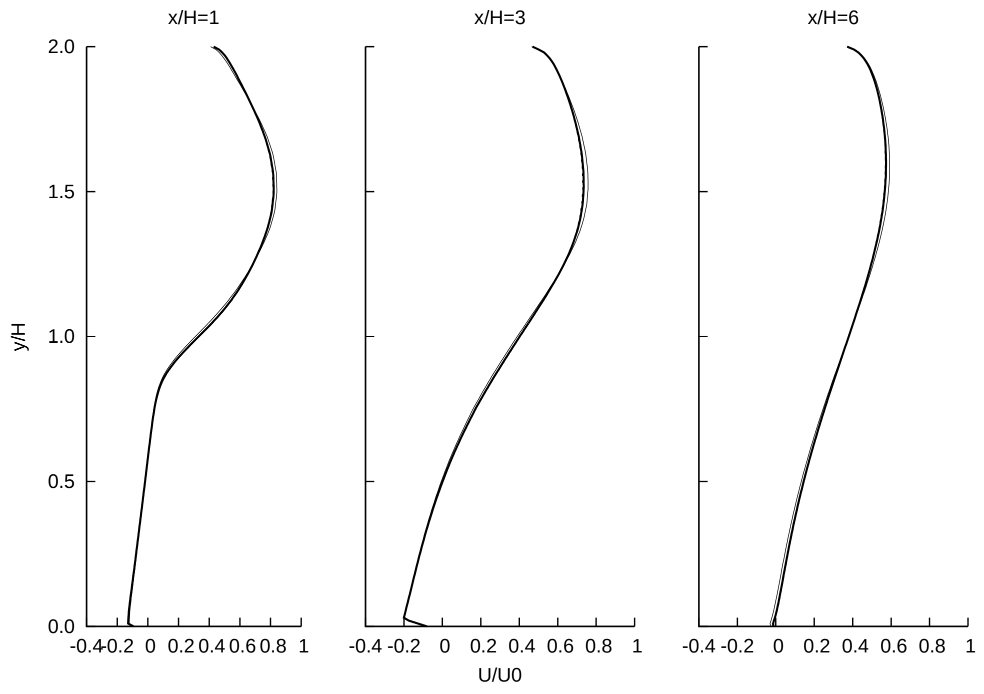


(e) Variation study for $\sigma_k$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

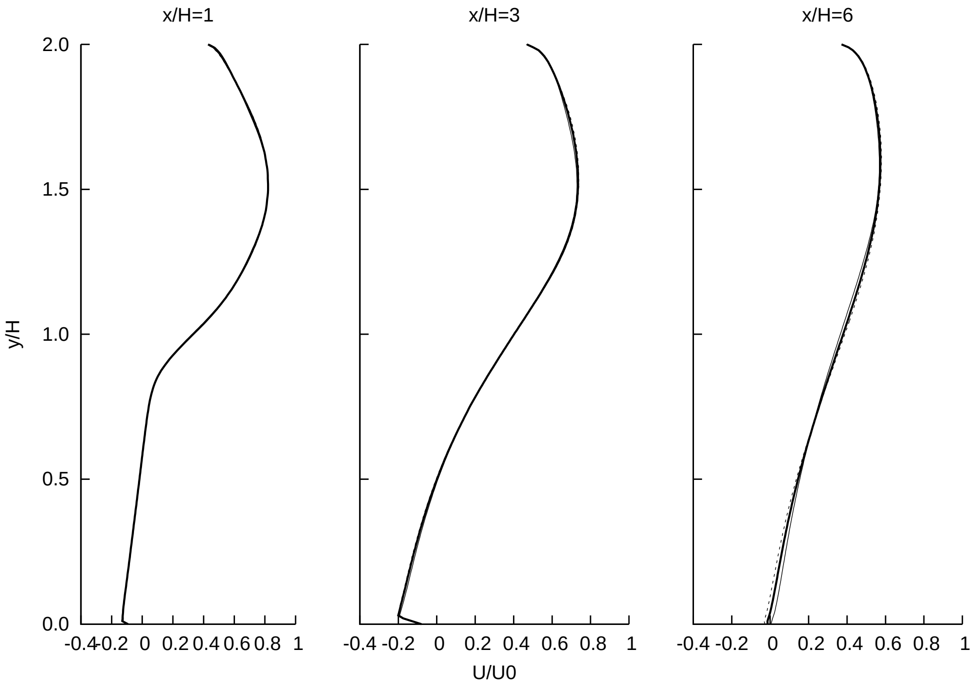
**Figure 5.2.:** Variation study for each of the five parameters in the standard k-$\varepsilon$ turbulence model in the backward facing step case. The graphs show velocity profiles along vertical cuts of the channel at three different positions $x/H$ =1,3 and 6 downstream of the step. Velocity $U$ was normalised with maximum inlet velocity $U_0$ and y-coordinate with step height H. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



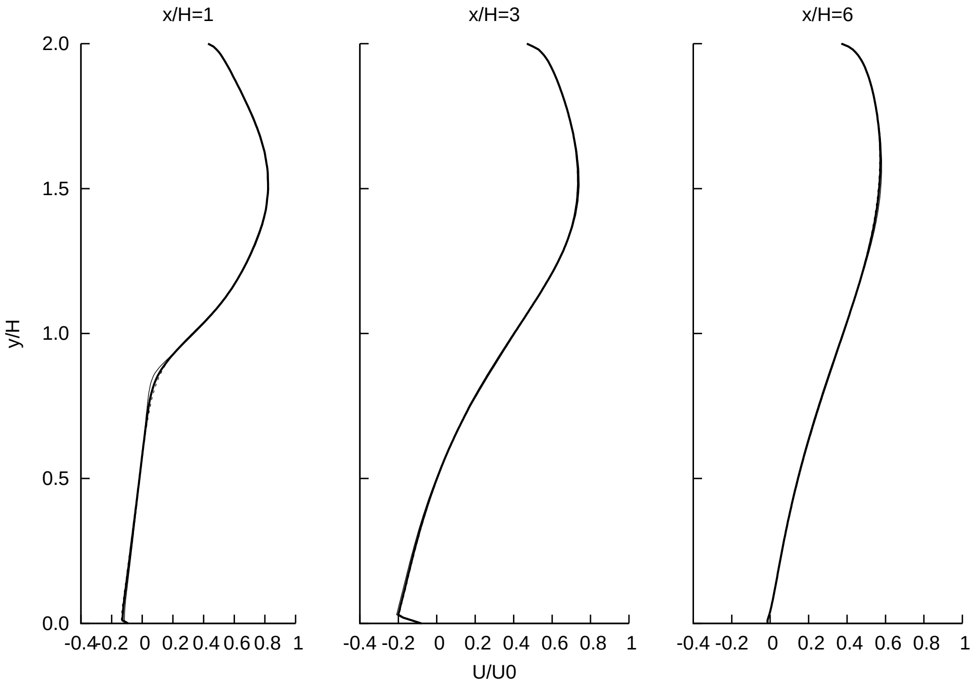
**(a)** Variation study for $a_1$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



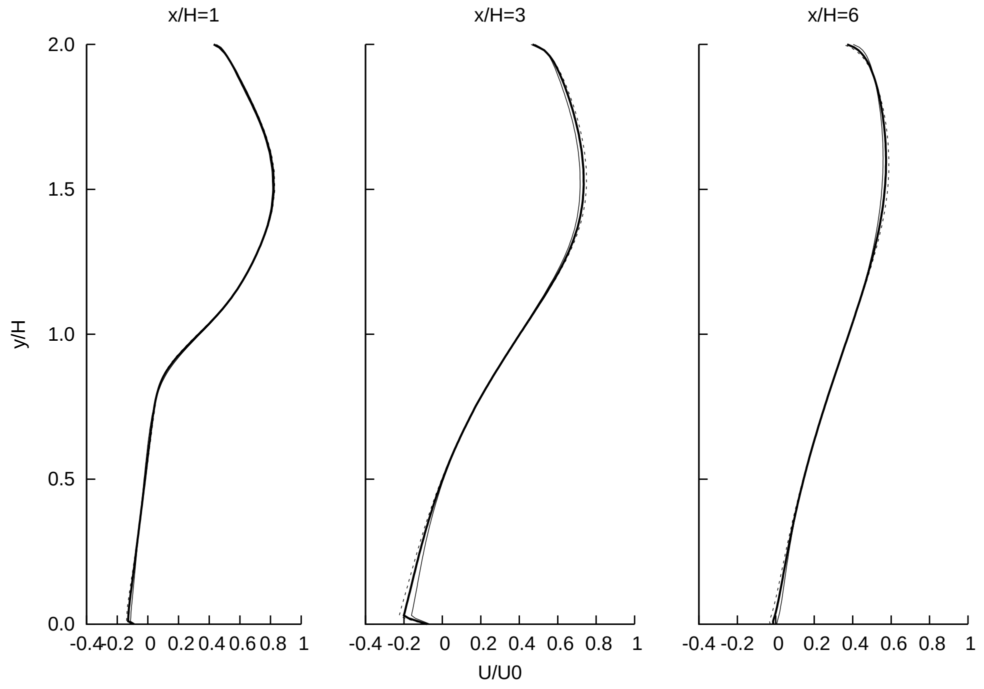
**(b)** Variation study for $c_1$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



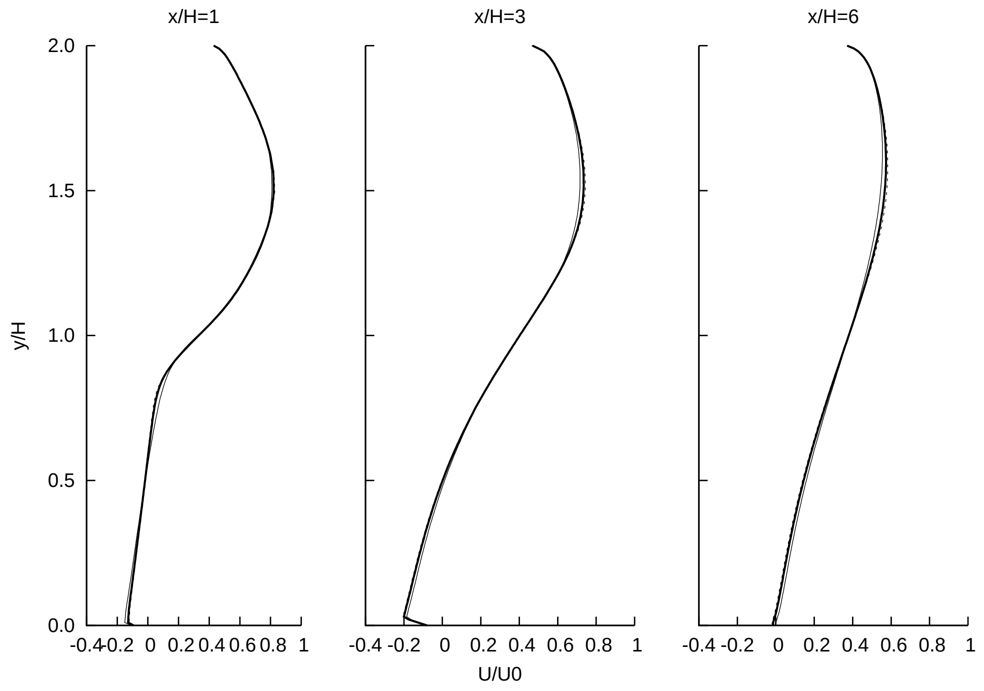
**(c)** Variation study for $s_{K1}$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



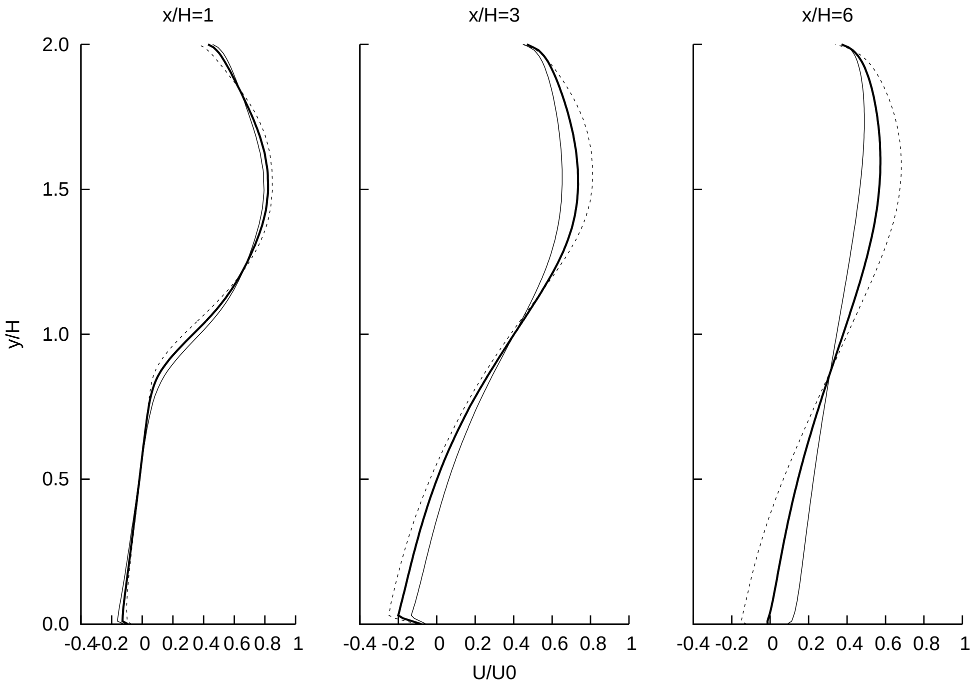
**(d)** Variation study for $s_{K2}$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



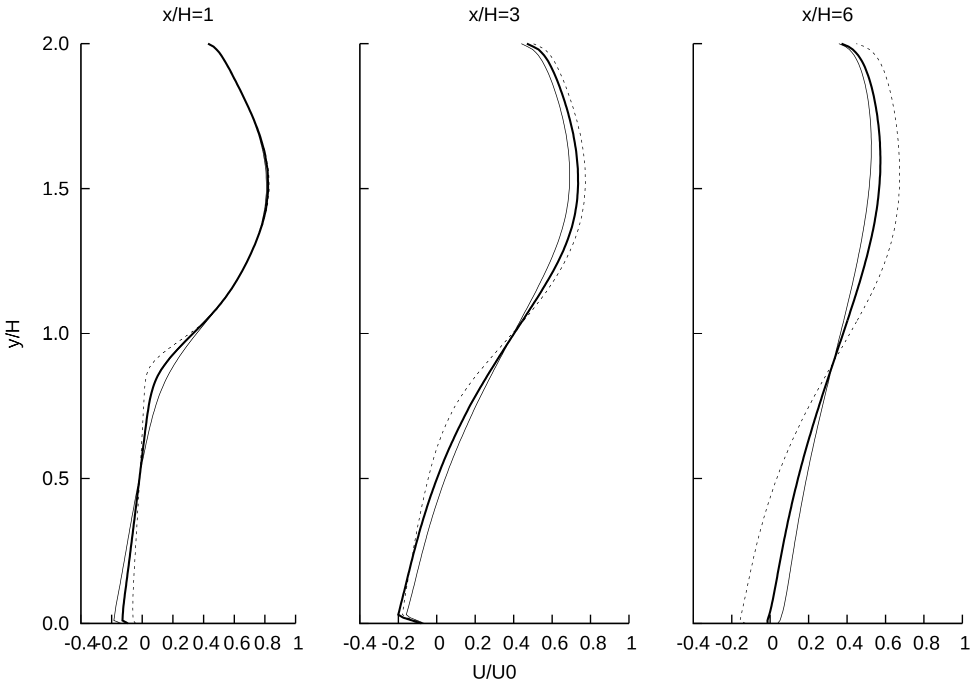
**(e)** Variation study for $s_{\omega 1}$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



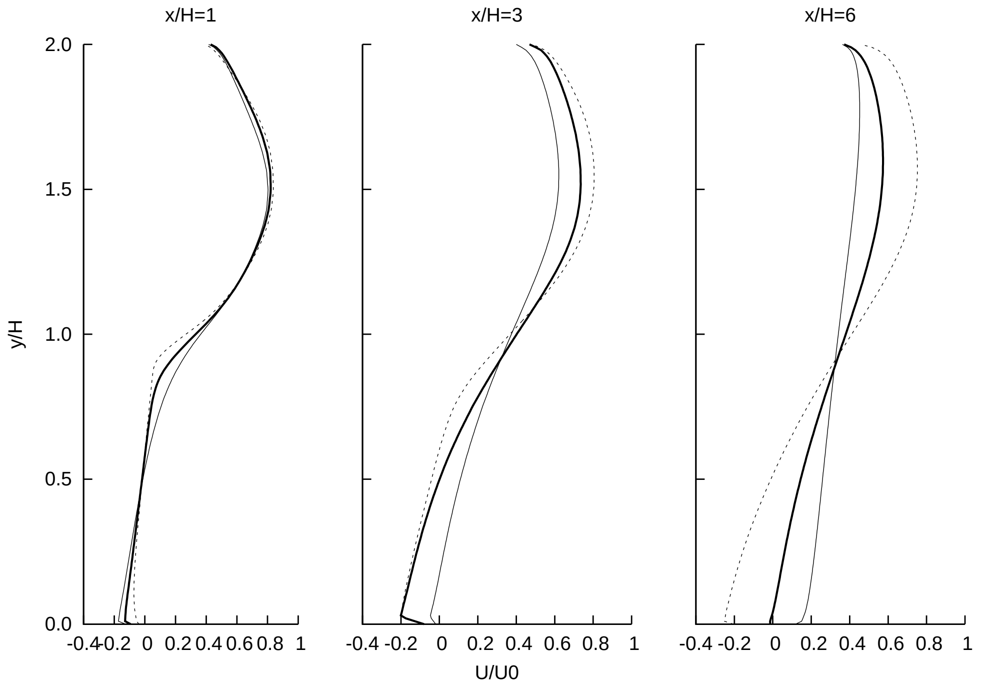
**(f)** Variation study for $s_{\omega 2}$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



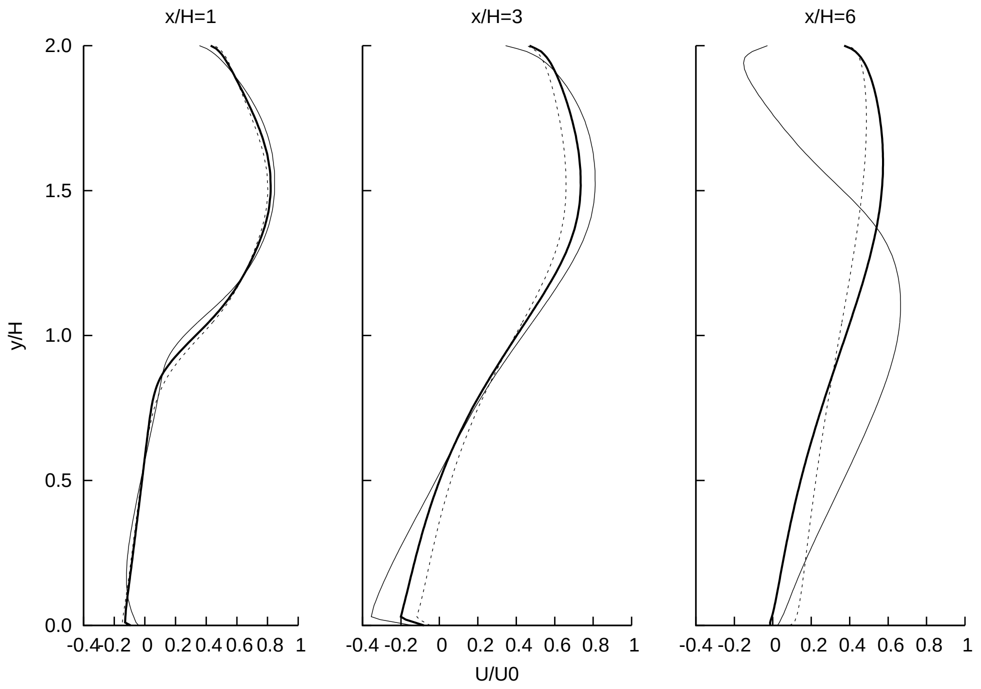
**(g)** Variation study for $\gamma_1$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



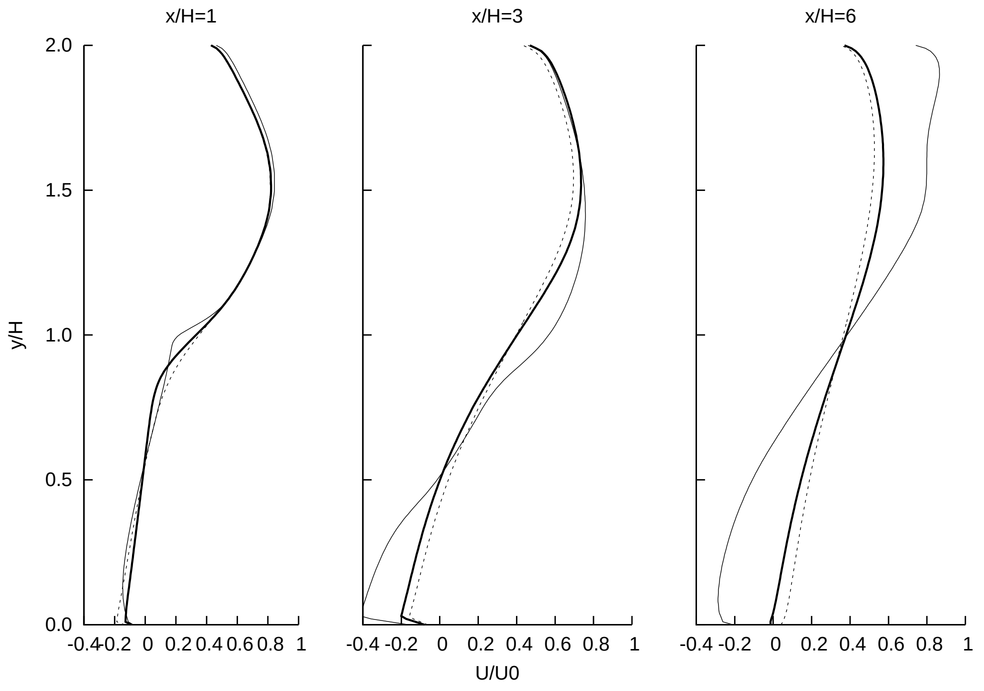
**(h)** Variation study for $\gamma_2$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(i)** Variation study for $\beta^*$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(j)** Variation study for $\beta_1$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(k)** Variation study for $\beta_2$. Plotted is normalised velocity over normalised channel height at three positions $x/H$ =1,3 and 6 downstream of the step. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

**Figure 5.3.:** Variation study for each of the eleven parameters in the standard k-$\omega$ turbulence model in the backward facing step case. The graphs show velocity profiles along vertical cuts of the channel at three different positions $x/H$ =1,3 and 6 downstream of the step. Velocity $U$ was normalised with maximum inlet velocity $U_0$ and y-coordinate with step height H. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

#### 5.2.1.3. Genetic Algorithm Setup

The optimisation using a genetic algorithm was run using the following genetic operators. For all simulations tournament selection was used with a tournament size of two. Tournament selection is less sensitive to fitness values that are very close together. Because no normalisation was performed to map fitness values into the range $[0, 1]$, most solutions had very similar fitnesses. Using roulette wheel or any other selection method that did not put too much competitive pressure on the individuals led to slow convergence of the optimisation. In subsequent runs tournament selection was used as the default setting because of the faster convergence.

Further, single point crossover was used in all optimisation runs. No other crossover operators were tested in this study. The crossover probability was set to 60%. Single bit turnover mutation occured with a probability of 3% and was applied to the child individual after the crossover. These values are based on recommendations by Goldberg [38]. The total number of individuals per population was 30. That is rather low for genetic optimisation [73], but turned out to be a good compromise between computational cost and GA performance. After already 20 generations the changes to the population were miniscule so that after 25 generations the evolution was terminated.

#### 5.2.1.4. Fitness Function

The data used is published on the ERCOFTAC classic online database [1]. Details on the experimental setup can be found there and in the PhD thesis by Makiola [66]. Data is available for different Reynolds numbers and a range of sloping angles of the step. Only an orthogonal step is considered here (step angle $90\,^\circ$) and configurations with Reynolds numbers of 15,000 and 64,000 were simulated, but because the results were almost identical, only the higher Reynolds number of 64,000 was pursued for the statistical analysis.

The objective variables were chosen based on the preliminary studies presented in the previous sections. For the k-$\varepsilon$ model these were the parameters $C_1$, $C_2$ and for the k-$\omega$-SST model the parameters $\gamma_1$, $\gamma_2$, $\beta_1$, $\beta_2$ and $\beta^*$. The parameter

[1] `http://cfd.mace.manchester.ac.uk/ercoftac/`

constraints are listed in table 5.3. To compare experimental and simulated data and derive an objective value, the root square error between velocity measurements taken by Makiola [66] and sample data from the simulation in the same geometric locations was calculated using the method described in Section 5.1.

**Table 5.3.:** Value constraints for the objective variables in the backward facing step case.

| Parameter | min value | max value | accuracy |
|---|---|---|---|
| **k-$\varepsilon$ model** | | | |
| $C_1$ | 0.864 | 2.020 | $10^{-3}$ |
| $C_2$ | 1.150 | 2.680 | $10^{-3}$ |
| **k-$\omega$ SST model** | | | |
| $\gamma_1$ | 0.4426 | 0.6638 | $10^{-4}$ |
| $\gamma_2$ | 0.3522 | 0.5284 | $10^{-4}$ |
| $\beta_1$ | 0.045 | 0.105 | $10^{-3}$ |
| $\beta_2$ | 0.0662 | 0.0993 | $10^{-4}$ |
| $\beta^*$ | 0.072 | 0.128 | $10^{-3}$ |

### 5.2.2. Results

To simulate one generation of fifty individuals takes approximately 21 minutes on ten computing cores in parallel. The algorithm saves time by not recalculating individuals that have been passed on from previous generations, so the total runtime for 30 generations averaged around seven hours.

The geometry of the case had an expansion ratio of $h/H = 2$ and the examined quantity was the normalised velocity $u/U_0$ at three different positions $x/H = 1, 3$ and 6 in the channel. That means the fitness was estimated as being the root mean square error between the simulated results and the velocity data measured in the experiment. The smaller the difference between the results, the better was the fitness of the solution. A parabolic velocity profile was prescribed at the inlet (as shown in Table 5.1).

The estimated optimal values are listed in Table 5.4 for the k-$\varepsilon$ model and in Table 5.5 for the k-$\omega$ model respectively. The values shown have been calculated in five GA runs with the settings mentioned above and presented results are averaged over these runs. The standard deviation over the five samples is also shown in the Tables. The variation is due to the non-deterministic nature of the optimisation algorithm.

Figures 5.4 and 5.5 show the velocity profiles at different positions downstream as calculated using the optimised k-$\varepsilon$ and k-$\omega$ coefficients respectively compared to the results obtained using the standard values included in OpenFOAM. The parabolic shape of the velocity profile is better captured by the optimised setup further downstream of the step. Using the standard coefficients, transition to fully developed channel flow takes place considerably faster, while the optimised profile maintains the dominance of the flow in the upper half of the channel in accordance to experiment. In both cases a dicrepancy between measured and simulated peak velocity can be observed. This could be due to the fact that the inlet velocity in the simulation is parabolic, which is only an approximation of real channel flow profile. Also the reduction to two dimensions might be an issue here. Qualitatively, though, the shape of the profiles is in accordance to experimental data.

Another interesting feature of the flow is the length of the recirculation eddy that forms downstream of the sudden geometry change. The k-$\varepsilon$ model is known to largely underestimate this entity [81]. When looking at the model equation 3.21 for the dissipation of turbulent kinetic energy, one would suggest that increasing the coefficient $C_1$ in the production term and at the same time decreasing coefficient $C_2$ in the dissipation term, turbulent eddies would prevail longer in the flow. Looking at the optimisation results in Table 5.4, that is exactly what the genetic algorithm determined as being the optimal solution. Similarly does an increase of the $\gamma$ values in the k-$\omega$ model lead to a decrease in energy dissipation and one would expect better agreement with the data.

**Table 5.4.:** Standard vs. optimised values for coefficients in the k-$\varepsilon$ model and standard deviations from five optimisation runs.

| | Std | Opt | $\sigma$ |
|---|---|---|---|
| $C_1$ | 1.44 | 1.92 | 0.082 |
| $C_2$ | 1.92 | 1.86 | 0.093 |

**Table 5.5.:** Optimum values and standard deviations for the k-$\omega$-SST model

| | Std | Opt | $\sigma$ |
|---|---|---|---|
| $\gamma_1$ | 0.553 | 0.606 | 0.018 |
| $\gamma_2$ | 0.440 | 0.510 | 0.021 |
| $\beta_1$ | 0.075 | 0.053 | 0.003 |
| $\beta_2$ | 0.083 | 0.076 | 0.019 |
| $\beta^\star$ | 0.09 | 0.095 | 0.0008 |

In summary it should be clear from these results, that the turbulence model optimisation works well in improving the simulation results on this particular flow type. The trends of the predicted optimal values agree well with the expectations built when looking at the model equations.

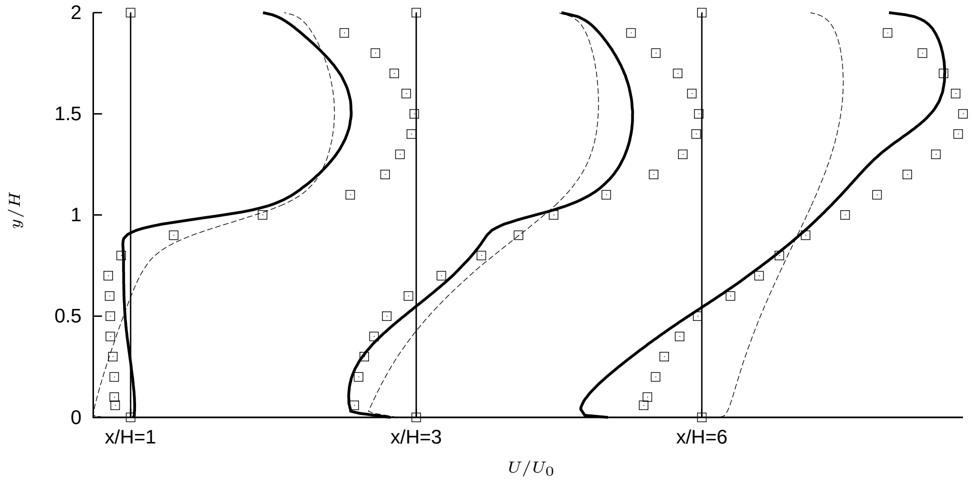


**Figure 5.4.:** Velocity profiles at positions $x/H = 1$, $x/H = 3$, $x/H = 6$ downstream of the step. The dashed line shows profiles calculated using the standard k-$\varepsilon$ model parameters as implemented in OpenFOAM. The bold line shows results obtained by the optimised set of parameters. Rectangles mark experimental values measured by Makiola [65]. Reynolds number of the flow based on step height and inlet velocity was 64,000.

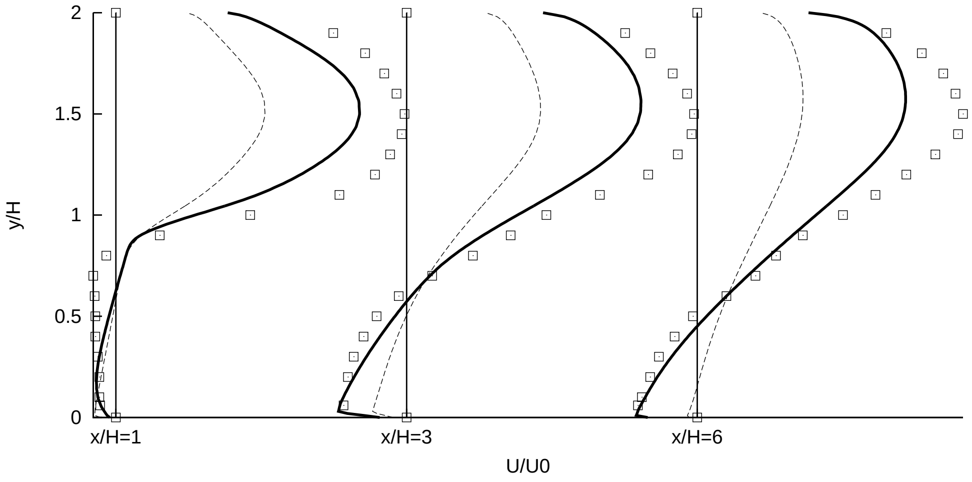


**Figure 5.5.:** Velocity profiles at positions $x/H = 1$, $x/H = 3$, $x/H = 6$ downstream of the step. The dashed line shows profiles calculated using the standard k-$\omega$ model parameters as implemented in OpenFOAM. The bold line shows results obtained by the optimised set of parameters. Rectangles mark experimental values measured by Makiola [65]. Reynolds number of the flow based on step height and inlet velocity was 64,000.

### 5.2.3. Impinging Jet

Impinging jets are used in industrial cooling, heating or drying processes such as annealing of metal, air curtains or cooling of turbine blades and many other applications. Localised mass, momentum and heat transfer make them very useful and a good understanding of their behaviour is essential for correct flow predictions. Due to the importance of impinging jets, measurements of the velocity, vorticity and temperature distributions are available [4, 16] as well as numerical data for round and plane jets both in 2d and 3d using RANS modeling, LES/RANS hybrid models, pure LES calculations or DNS for low Reynolds numbers (e.g. [17, 77, 103]).

The performance of different turbulence models in the impinging jet case with respect to heat transfer and flow field were investigated by Jaramillo et al [49]. They concentrated on non-linear eddy viscosity models (NLEVM) such as $k - \varepsilon$ and $k - \omega$ models, but also aquired data from Large Eddy Simulations and Direct Numerical Simulations. Their results show that most $k - \epsilon$ models underpredict the turbulent decay when the flow changes its main direction to spread parallel to the impingement wall. The character of the flow changes at the stagnation point from a free jet flow to a wall bounded shear flow. It is difficult for any turbulence model to predict both configurations accurately. When the distance between inlet and impingement plate is small and the jet reaches the wall before it has completely developed to a free jet, the influence of the error in modelling the turbulence of the stream is obviously smaller. When the distance increases and the core jet has fully converged to a turbulent flow, the modelling error becomes more dominant and is transported into the wall-shear dominated part of the flow. This feature makes the impinging jet problem a good candidate for turbulence model parameter optimisation.

#### 5.2.3.1. Boundary Conditions

The case setup used in this work is a plane jet impinging perpendicularly onto a wall of higher temperature $T_W$. The dimensioning parameter for this case is the jet width $B$ (see Figure 5.6). The case investigated here had a distance of $H/B = 4$ from jet entry to the impingement wall. The Reynolds number for all cases was

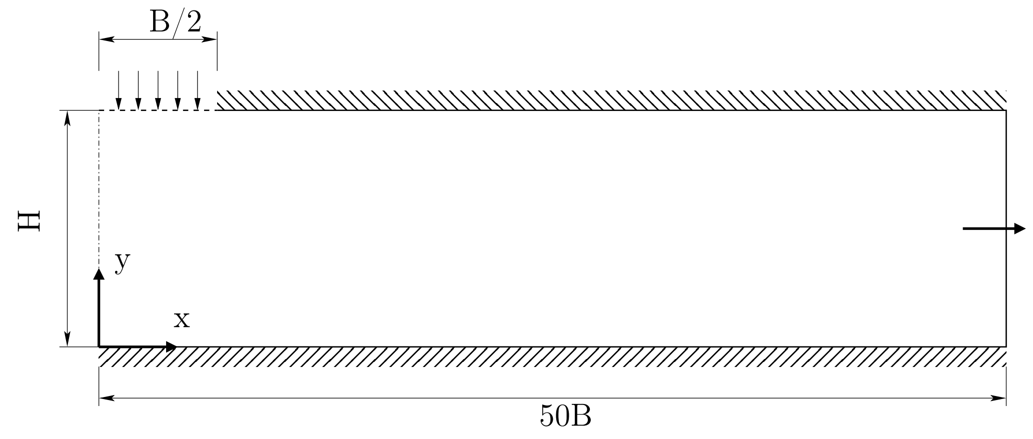


**Figure 5.6.:** Geometry of the impingement jet test case

20,000 based on the jet width and the inflow velocity. The mesh was created based on the suggestions by Jaramillo [49] to ensure grid independent solutions. A detailed grid convergence study has been performed by the authors and the resulting mesh consists of 270 × 180 cells. Only half the domain was calculated since the case is symmetric in the x-direction with the centerline of the jet as the axis of symmetry. Results from their group were also used as reference for the fitness evaluation.

Inlet turbulent kinetic energy ($k_{in}$) and turbulent kinetic energy dissipation rate ($\varepsilon_{in}$) are calculated using equations 5.4 and 5.5 respectively with a turbulent intensity of $U_i = 0.02\,U_{max}$, and the characteristic length scale for the epsilon equation is $\ell = 0.015\,B$. The walls are isothermal, with the impingement plate at a constant 310 K and the confinement plate at 300 K. No-slip conditions were imposed at the solid walls. The outlet was realised as a pressure outflow with zero gradient conditions for the turbulent quantities. Table 5.6 gives an overview over the boundary conditions in the terminology of OpenFOAM.

As it turned out, the convergence behaviour of this case in OpenFOAM was less good than in an identically set up case in the commercial CFD code FLUENT [2]. Changing relaxation factors for the p and U equations did improve matters considerably, but in the end one simulation run of the case took 6000 iterations with the `buoyantBoussinesqSimpleFoam` solver to converge. Unfortunately it was

[2]http://www.ansys.com

**Table 5.6.:** Boundary conditions in the impinging jet test case

| | **inlet** | **outlet** |
|---|---|---|
| U | Dirichlet $U_0$ | von-Neumann |
| p | von-Neumann | Dirichlet $p_0$ |
| k | Dirichlet eqn. 5.4 | von-Neumann |
| $\varepsilon$ | Dirichlet eqn. 5.5 | von-Neumann |
| $\omega$ | Dirichlet eqn. 5.6 | von-Neumann |
| T | Dirichlet 300 | von-Neumann |
| | **top wall** | **impingement wall** |
| U | `fixedValue 0` | `fixedValue 0` |
| p | von-Neumann | von-Neumann |
| k | wall function | wall function |
| $\varepsilon$ | wall function | wall function |
| $\omega$ | wall function | wall function |
| T | Dirichlet 300 | Dirichlet 310 |

not possible to find the source for this discrepancy. To work around this problem, the case was simulated using standard model parameters and the output of this simulation was used as initial state for any run with modified coefficients. That saved a considerable amount of time it would normally take for the flow to develop the characteristic outward spreading parallel to the impingement wall. With this slight modification it was possible to cut down the number of steps for the case to converge to about 2000. For good measure all cases were run for 3000 iterations of the SIMPLE algorithm [27], which took on average 25 minutes on a single core.

#### 5.2.3.2. Genetic Algorithm Setup

The crossover probability was set to 60%. One bit turnover mutation occured with a probability of 3%. The total number of individuals per population was 40.

As in the backward facing step test case, the optimisation converged after a few generations, so the maximum number of iterations for the jet case was set to 30 generations. Taking into account the run time of a single simulation, the optimisation procedure took about 380 hours without parallelisation. By distributing the workload onto ten computational cores and not simulating recurring individuals repeatedly the overall runtime could be cut down to approximately 30 hours, depending on the evolution of the population.

#### 5.2.3.3. Fitness Function

For this case both experimental data as well as previous simulation results by other researchers were used as input. The experiments were conducted by Ashforth-Frost [4]. The measurements presented were heat–transfer distribution expressed as local Nusselt number as well as velocity and turbulence values. To compare experimental and simulated data and derive a fitness value, the root square error was calculated using the method described in Chapter 5.1. Fitness was based on derivation from the experimental data, while the available simulations were used to judge the initial performance of the OpenFOAM code and see if it were able to produce similar results.

### 5.2.4. Results

The runtime of the impinging jet case was about 40 minutes per simulation. That was due to the slow convergence especially of the pressure equation. Modifying the relaxation factors of the solver improved matters slightly, but to be on the safe side when it came to changing the model coefficients the decision was to set the relaxation conservatively low to ensure full convergence in all runs during the optimisation.

The velocity profiles shown in Figure 5.7 clearly show improved performance with modified turbulence model coefficients. Further away from the jet core ($x/B \ll 1$) the improved equations capture the flow behaviour almost perfectly in comparison with experiment. While the standard results predict almost inlet velocity in the region close to the wall (at about $y/B \approx 0.15$), experiment and

**Table 5.7.:** Optimum values and standard deviations for the k-$\omega$-SST model

| | Std | Opt | $\sigma$ |
|---|---|---|---|
| $\gamma_1$ | 0.553 | 0.457 | 0.024 |
| $\gamma_2$ | 0.440 | 0.486 | 0.0 |
| $\beta_1$ | 0.075 | 0.104 | 0.003 |
| $\beta_2$ | 0.083 | 0.0941 | 0.019 |
| $\beta^\star$ | 0.09 | 0.075 | 0.0008 |

optimised model show more energy loss and therefore lower total velocity values. This is due to the increase in the $\gamma$ parameters.

Comparing these parameters to those found for the backwarding face case (see Table 5.5 a significant difference can be observed. This proves the initial postulation that turbulence model coefficients are mainly problem dependent.

For the impinging jet case it can be said that using an optimisation procedure to adapt the turbulence model coefficients can improve the accuracy of the simulation significantly.

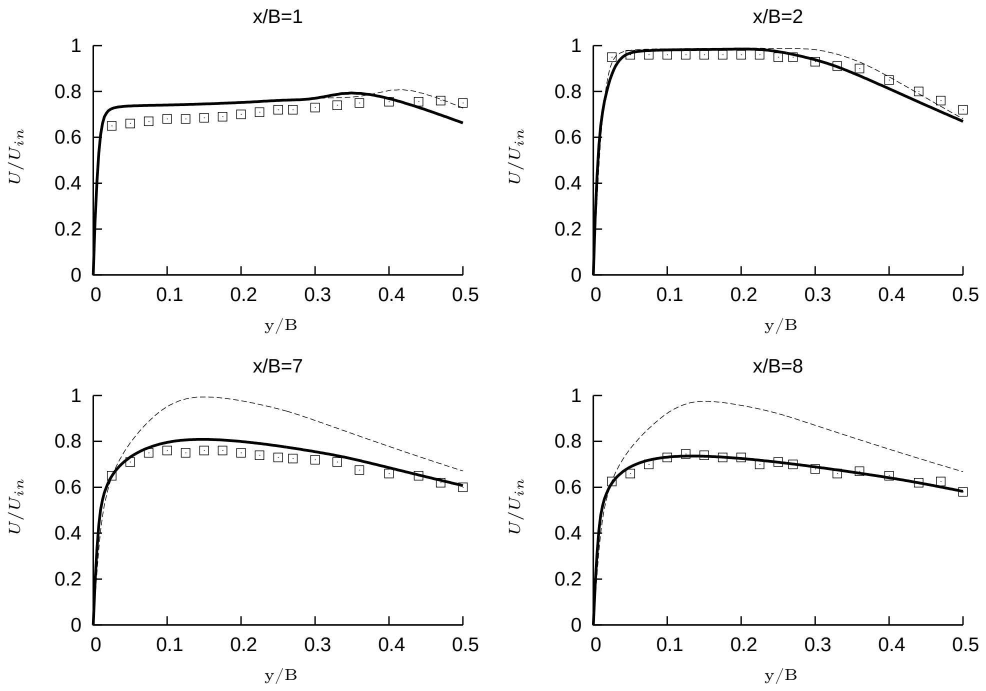


**Figure 5.7.:** Normalised velocity profiles at positions $x/B = 1$, $x/B = 2$, $x/B = 7$, $x/B = 8$ away from the jet. The dashed line shows profiles calculated using the k-$\omega$-SST model parameters as implemented in OpenFOAM. The bold line shows results obtained by the optimised set of parameters. Rectangles mark experimental values measured by Ashforth-Frost et al. [4]

### 5.2.5. Conical Concentrator and Sudden Expansion

Stewart et. al. [96] from the United States Food & Drug Administration (FDA) designed a benchmark case to develop guidelines for CFD users in industry consisiting of a cylindrical nozzle with a conical collector and a sudden expansion. Their interest stems from the fact that the nozzle characteristics are typical for medical devices transporting blood, such as catheters, syringes, blood tubing etc. Reliability of results from flow simulations using these devices is naturally of high importance and defining benchmarks to test new models and methods against experimental data is a logical step.

The intention behind this is that although CFD is widely used in the development of medical devices, validation of the simulations lacks standardised methods and regulations. For their study the researchers invited twenty-eight groups that offer CFD services on an international level to submit results of calculations of five different flow rates, covering laminar, transitional and turbulent flow. In addition to this they employed three independent laboratories to provide the required experimental validation data using particle image velocimetry (PIV). As it turned out, the computational results showed large discrepancies between each other and to the experimental data, which shows that CFD studies should not be taken for granted and require a close examination and careful experimental validation.

The main sources of errors identified by Stewart's group involved false estimations of centerline velocities in the laminar regions and inaccurate prediction of the recirculation in the sudden expansion. Figure 5.8 shows the dimensions of the geometry used in the laboratories to obtain experimental results. The model includes a radial step, sharp edges, and a cross-sectional contraction that combine to induce shear stresses related to reported problems in these devices. The device was designed to include accelerating flow, decelerating flow, variations in shear stress and velocities, and recirculating flow, all of which may be related to blood damage in medical devices [96].

The nozzle is rotationally axisymmetric and contains a neck of 0.04 m in length connecting a diffuser with a 20° incline on one end and a sudden diameter change on the other end. The device can be operated in both flow directions, but in this treatment of the benchmark study only the flow from left to right is considered. That would model a conical collector and a sudden expansion. The length of the inlet and outlet channels were not specified and could be chosen by the experimentators to ensure fully developed turbulent flow before entering the conical concentrator and the outflow condition should not influence the reattachment point in the model. In the simulation the length of the inlet and outlet channels were chosen as $15d$ and $300d$ respectively, with $d$ being the diameter of the throat. For a throat Reynolds number of 5000, the inlet velocity was specified as $0.46\,\frac{\mathrm{m}}{\mathrm{s}}$. The best simulation results according to the study [96] were achieved using the Spalart-Allmaras one-equation turbulence model [93] (see also Chapter 3.4.1). Hence this turbulence model was used in the GA optimisation for this test case.

The ultimate findings of the study were, as expected, a strong coupling between simulation results and the choice of turbulence model. Especially the transitional Reynolds regime produced a wide spread of numerical results. Surprisingly not using any form of turbulence modeling and instead doing a laminar computation resulted in good agreement to experimental data. Yet the aim of this work is to try and reproduce the experimental results with two possible turbulence models, the k-$\omega$ and Spalart-Allmaras model. A genetic optimisation will then be used to tune these models to better match the nozzle configuration. It would be interesting to see, if these modified models hold if the flow direction is reversed. Unfortunately by the time the current research was concluded, the experimental data for the reverse case has not yet been published.

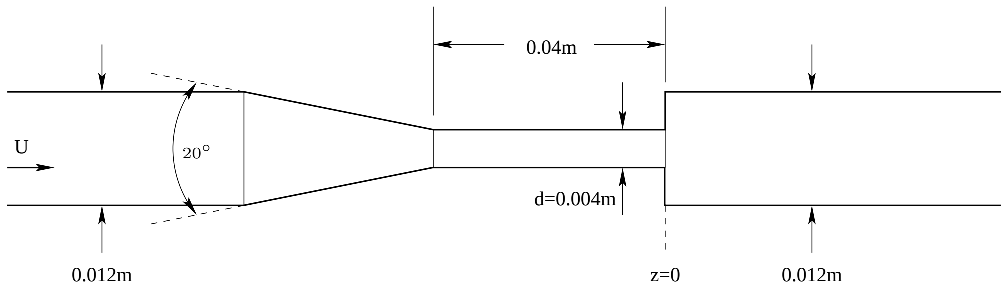


**Figure 5.8.:** Nozzle specifications for the FDA test case

#### 5.2.5.1. Boundary Conditions

The computational mesh used in the optimisation study was provided by one of the authors of the FDA paper (Eric Patterson). The required grid refinement study has already been conducted and the results achieved using this mesh have been fairly close to the experimental data. Because of the rotational symmetry of the geometry only a wedge shaped mesh with a symmetry axis boundary condition was used. That simplifies the case to quasi 2–D, with cyclic boundary conditions on the cut faces. The total number of cells was 25,000 with most of the cells making up the leading and trailing pipe surrounding the throat and collector to ensure fully developed turbulence when the flow reaches the diffuser and minimising the influence from the outlet to the recirculation downstream of the sudden expansion.

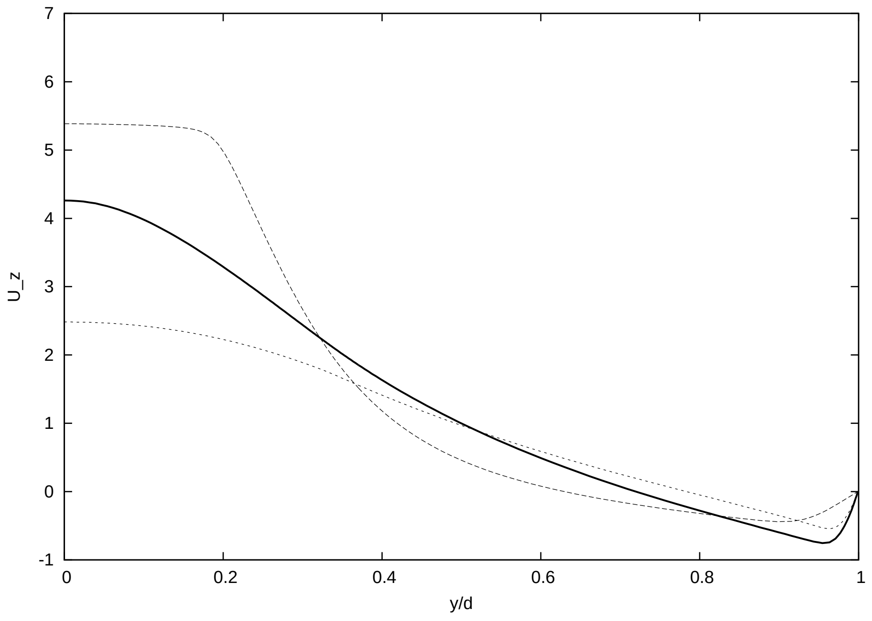
Two throat Reynolds numbers were considered, $\mathrm{Re}_t = 3,500$ and $\mathrm{Re}_t = 6,500$ based on the diameter of the throat, a dynamic viscosity of $\mu = 0.0035\,\frac{\mathrm{Ns}}{\mathrm{m}^2}$ and a fluid density of $\rho = 1056\,\frac{\mathrm{kg}}{\mathrm{m}^3}$. These are the physical values for water, which was the fluid used in the laboratory experiments. The Reynolds numbers at the inlet, where the diameter of the nozzle is three times larger than along the throat, were $\mathrm{Re}_i = 1,167$ and $\mathrm{Re}_i = 2,167$ respectively. It is important to note, that these Reynolds numbers mark the transitional phase between laminar and turbulent flow which is especially challenging for a turbulence model. Table 5.1 lists the boundary conditions as they appear in the OpenFOAM case setup.

**Table 5.8.:** Boundary conditions in the concentrator and sudden expansion test case

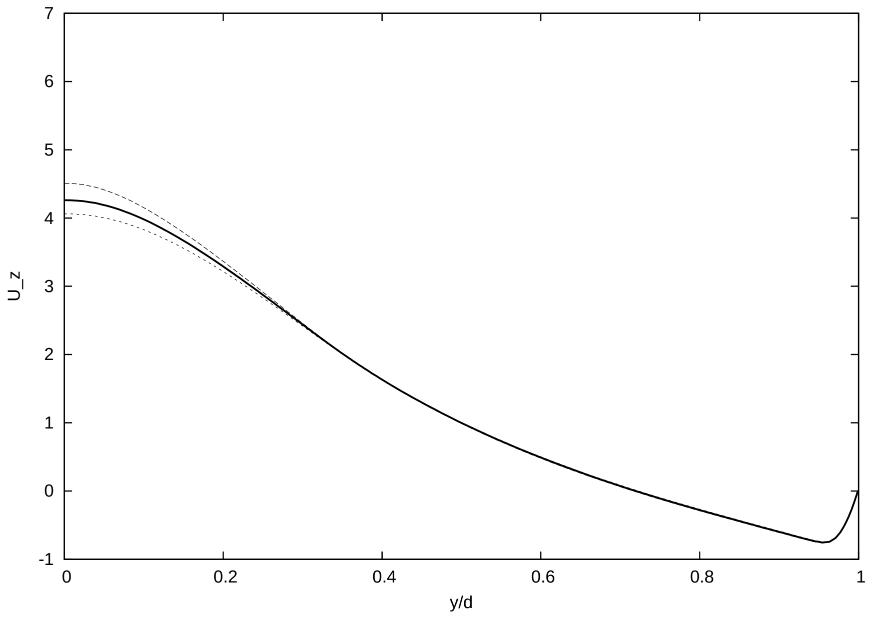
| Field | inlet | outlet | walls |
|---|---|---|---|
| U | $U(y) = -(y - H/2)^2 + U_0$ | von-Neumann | Dirichlet |
| p | von-Neumann | Dirichlet $p_0$ | von-Neumann |
| k | Dirichlet eqn. 5.4 | von-Neumann | wall function |
| $\varepsilon$ | Dirichlet eqn. 5.5 | von-Neumann | wall function |
| $\omega$ | Dirichlet eqn. 5.6 | von-Neumann | wall function |

#### 5.2.5.2. Preliminary Studies

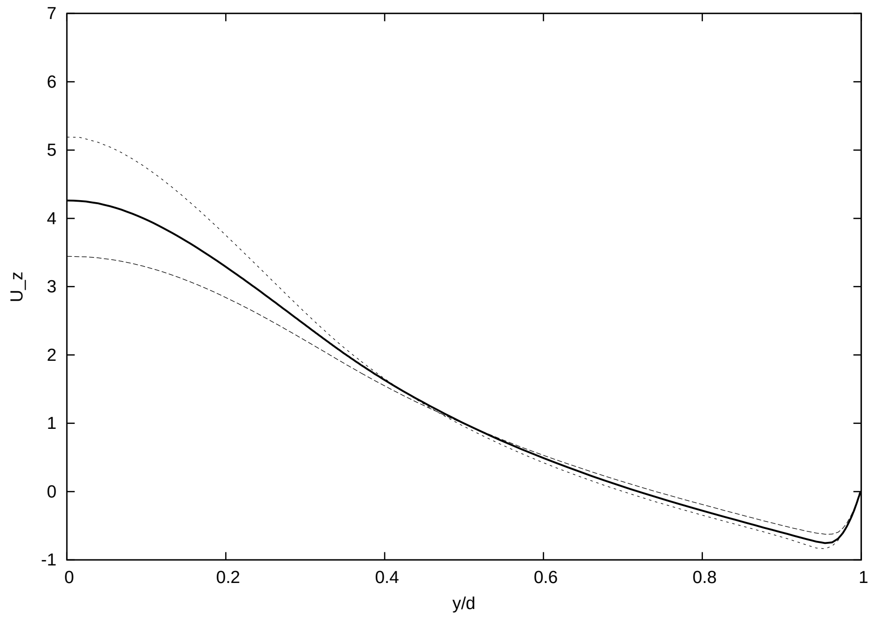
Similar to the variation studies for the turbulence models in the backward facing step test case (see Section (5.2.1.2)), an investigation of the single parameters as they appear in the OpenFOAM implementation and their influence on the development of the flow field was conducted. The results of this study are shown in Figure 5.9.



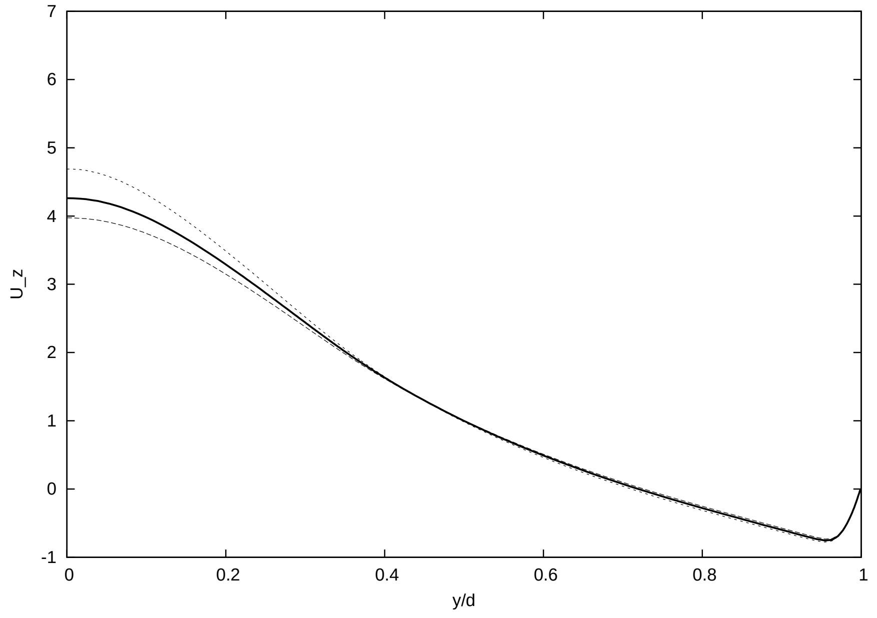
**(a)** Variation study for $C_{b1}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.$C_{b1}$



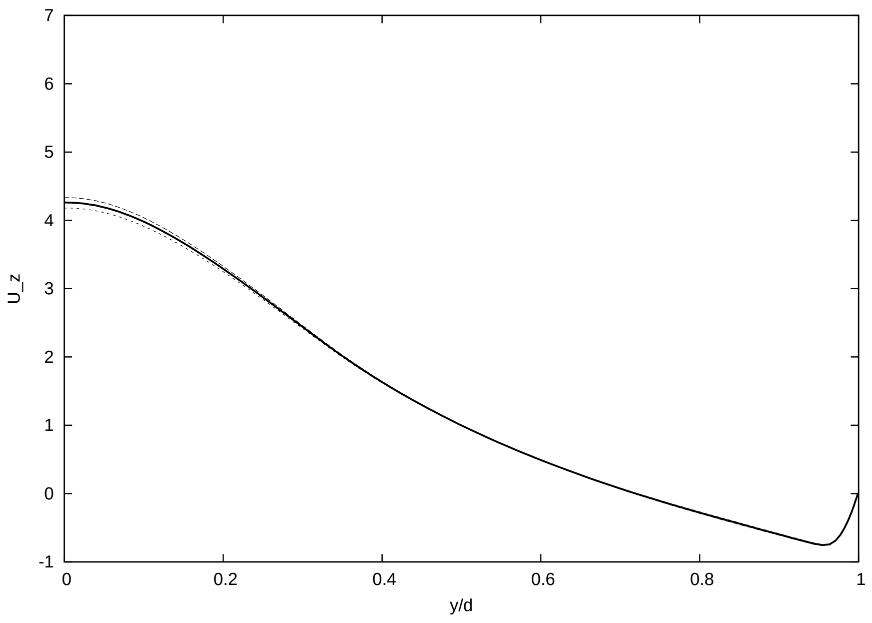
**(b)** Variation study for $C_{b2}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



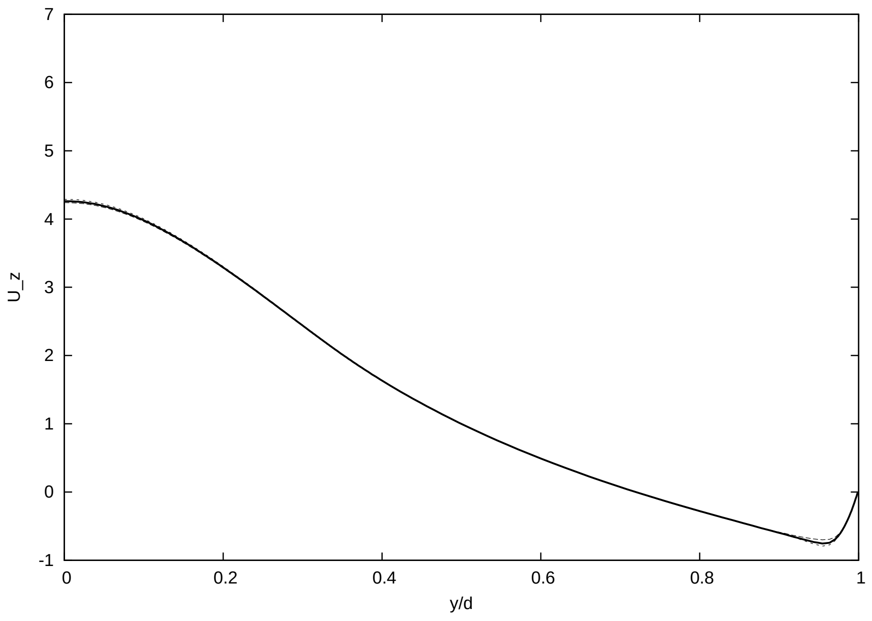
**(c)** Variation study for $C_{v1}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



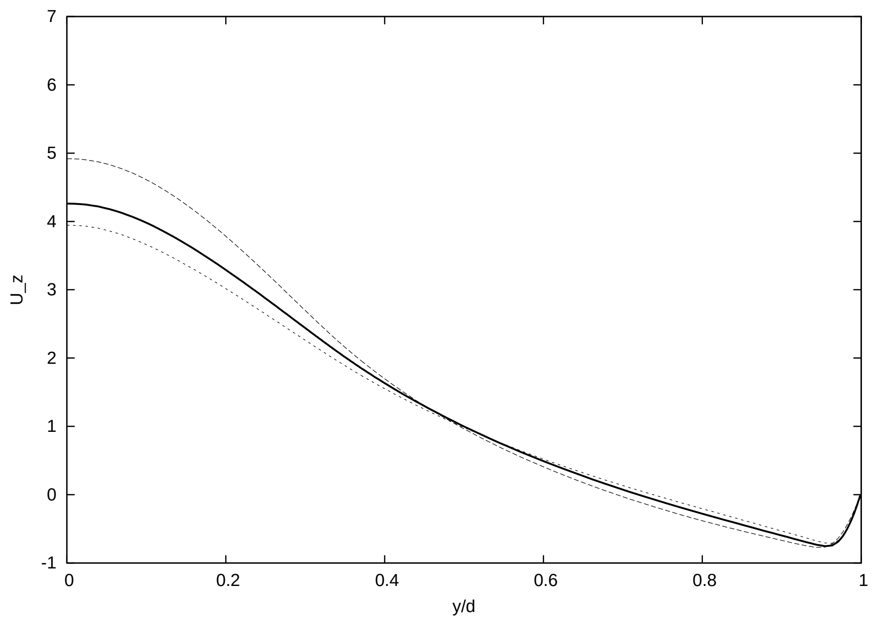
**(d)** Variation study for $C_{v2}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



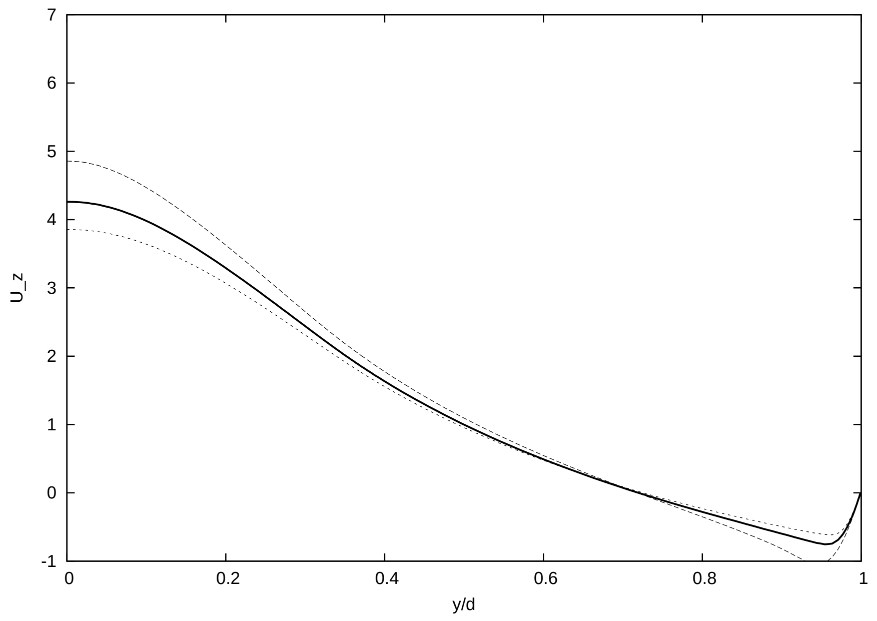
**(e)** Variation study for $C_{w2}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(f)** Variation study for $C_{w3}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(g)** Variation study for $\sigma_{\nu t}$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.



**(h)** Variation study for $\kappa$. Plotted is velocity over radial distance at z–position $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

**Figure 5.9.:** Variation study for each of the eight parameters in the standard Spalart Allmaras turbulence model for the FDA case. The graphs show velocity profiles along a vertical cut of the nozzle at positions $z/d = 6$. The thick line shows the profile calculated with the standard values, the dotted line represents results with 40% and the dashed line with 160% of the standard values.

#### 5.2.5.3. Genetic Algorithm Setup

Since this case setup is slightly more complicated and requires a larger grid than the previous ones, computation time was considerably longer. One set of 30 individuals over 30 generations takes on average about 72 hours to complete on 10 cores.

The objective variables were chosen based on the preliminary studies presented in the previous section. Following these results a decision was made to only include five parameters in the optimisation procedure. These were $C_{b1}$, $\sigma_{\nu t}$, $\kappa$, $C_{v1}$ and $C_{v2}$ for the Spalart–Allmaras model. The parameter constraints are listed in table 5.9.

**Table 5.9.:** Value constraints for the objective variables from the Spalart–Almaras model in the conical concentrator and sudden expansion case.

| Parameter | min value | max value | accuracy |
|---|---|---|---|
| $C_{b1}$ | 0.0810 | 0.1900 | $10^{-4}$ |
| $\sigma_{\nu t}$ | 0.3000 | 0.6666 | $10^{-4}$ |
| $\kappa$ | 0.245 | 0.410 | $10^{-3}$ |
| $C_{v1}$ | 7.10 | 14.20 | $10^{-2}$ |
| $C_{v2}$ | 5.00 | 10.00 | $10^{-2}$ |

The fitness of a solution was calculated by taking experimental results gathered by Stewart et. al. [96]. All data is publicly available on the U.S. Food and Drug Administration website [3]. Three seperate datasets were available and an ensemble average was calculated to serve as reference data. The measured quantity used for the objective function was the flow velocity at multiple positions along the nozzle geometry. Figure 5.10 shows the radial samples on cross-sections along the geometry. Further data was collected along the centerline.

The same positions were sampled in the CFD results and an overall r.m.s. deviation of the streamwise velocity component z from experiment was calculated. The smaller this deviation, the fitter the individual solution. When running the simulation with standard model coefficients, agreement with the experiment was

[3]`https://fdacfd.nci.nih.gov/` , last accessed 1st Dec 2013

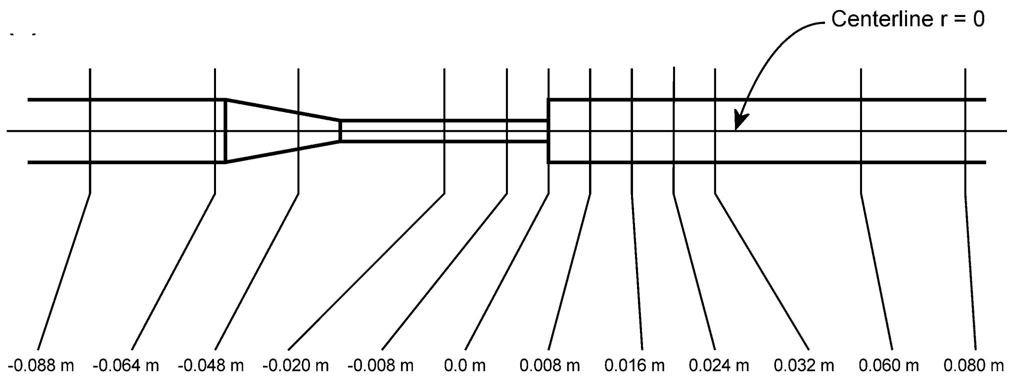


**Figure 5.10.:** Sampling positions along z-axis of the nozzle geometry.

not particularly good before the flow entered the throat, but got better after the sudden expansion ($z = 0$).

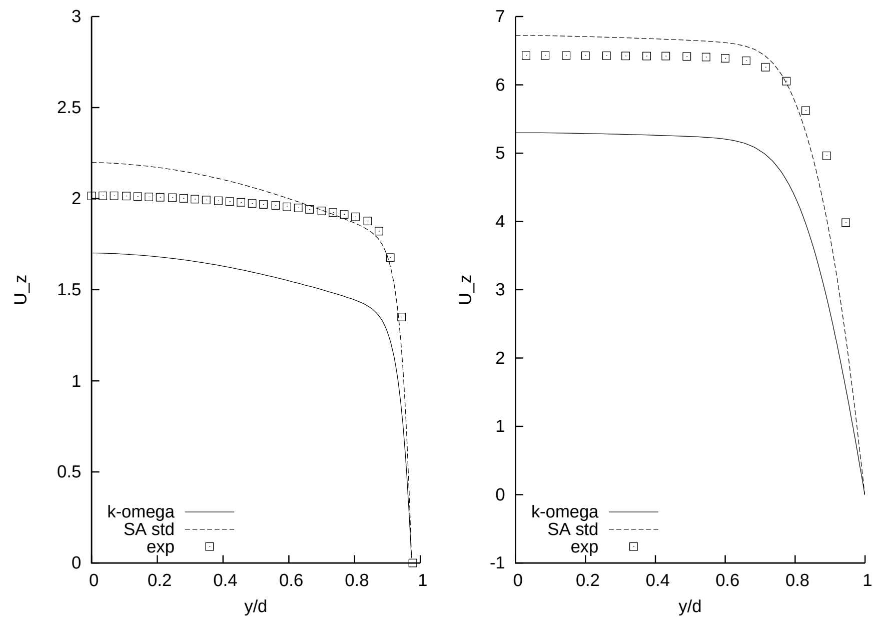
## 5.2.6. Results

As mentioned earlier, the time to complete an optimisation run on the geometry described in Section 5.2.5 was over 72 hours. For that reason only five optimisations could be performed. Figure 5.11 shows velocity profiles at four distinct points along the nozzle for the k-$\omega$ and the Spalart–Allmaras models using standard model parameters. These results were the basis for further optimisation. The graphs indicate, that in the simulation with the k-$\omega$ model mass conservation seems not satisfied. These results were reproducable and the source is unclear.

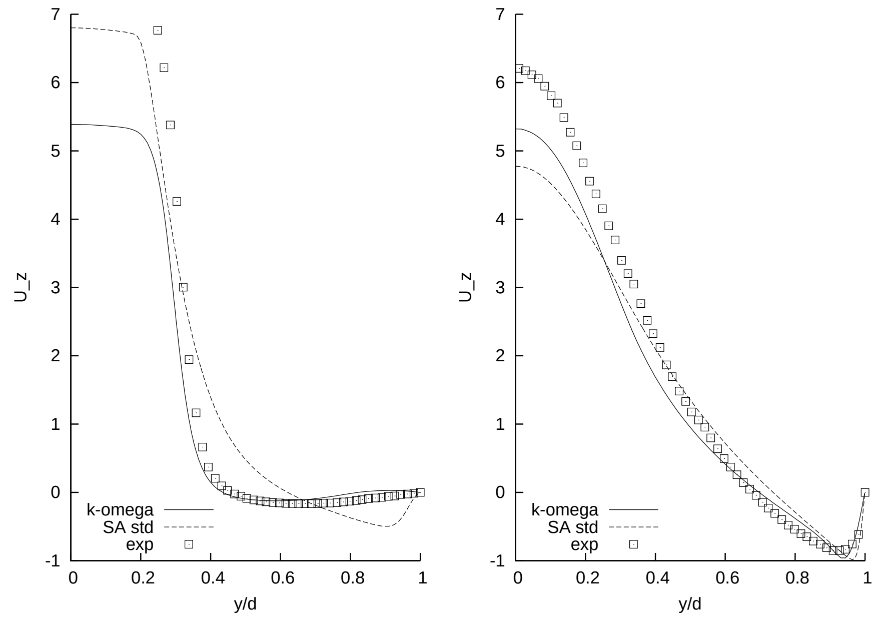
Figure 5.12 shows z-directional velocity at four different slices along the nozzle. It clearly shows an improvement when compared to the experimental data. The parameter values that were the outcome of the optimisation procedure are listed in Table 5.10. It is interesting to find one set of parameters that improves both the flow profile inside the throat as well as the profiles after the expansion. While inside of the throat simple pipe flow is present, a free spreading jet occurs after the step. Yet the proposed values seem to be of general validity. This is counter intuitive in so far, no turbulence model is known to model both these flow types with the same accuracy. On the other hand it is important to see that the standard deviation calculated from five optimisation runs for the $\kappa$ and $C_{b1}$ parameter are relatively large in relation to the absolute values ($> 25\%$). That means these two coefficients do not have as large an impact on the development of the flow as the preliminary studies have suggested.

**Table 5.10.:** Standard values for the Spalart–Allmaras model in OpenFOAM

| | Std | Opt | $\sigma$ |
|---|---|---|---|
| $C_{b1}$ | 0.1355 | 0.146 | 0.041 |
| $\sigma_{\nu t}$ | 0.666 | 0.492 | 0.10 |
| $\kappa$ | 0.41 | 0.363 | 0.12 |
| $C_{v2}$ | 10 | 9.734 | 0.622 |
| $C_{v1}$ | 7.1 | 5.791 | 1.08 |

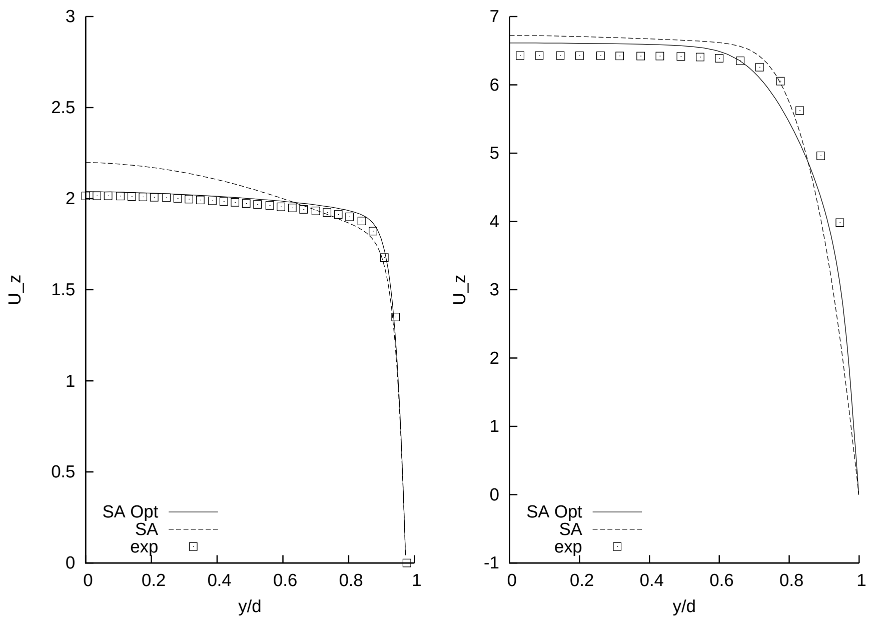


**(a)** $z = -12d$ (left), $z = -2d$ (right)

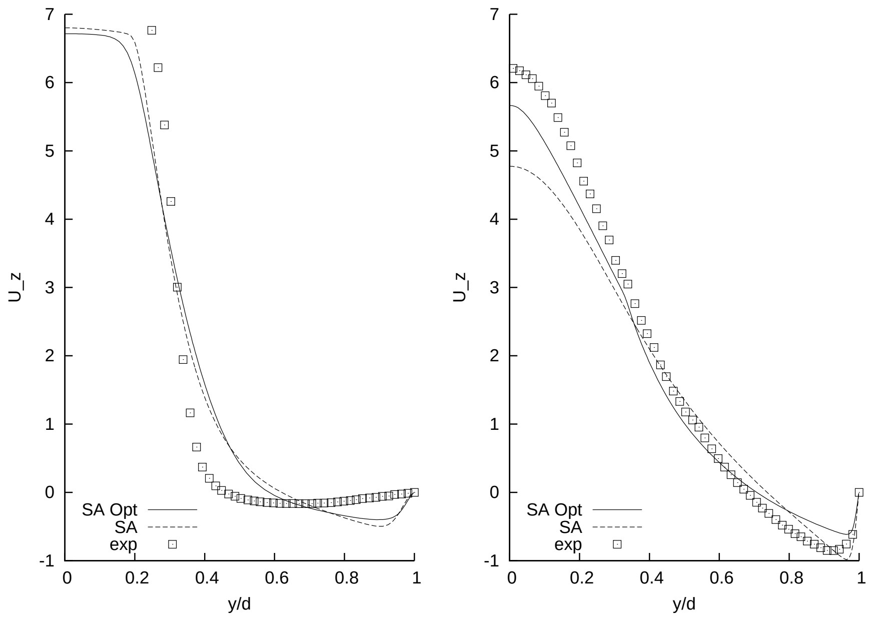


**(b)** $z = 2d$ (left), $z = 6d$ (right)

**Figure 5.11.:** Velocity profiles for streamwise velocity component at radial positions $z = -12d$, $z = -2d$, $z = 2d$ and $z = 6d$ along the nozzle, where d is the diameter of the throat. Data was calculated using the Spalart–Allmaras (SA) and k-$\omega$ model with standard parameter values. X-axis is normalised with nozzle diameter at the current z-position. Squares represent experimental data.



**(a)** $z = -12d$ (left), $z = -2d$ (right)



**(b)** $z = 2d$ (left), $z = 6d$ (right)

**Figure 5.12.:** Velocity profiles for streamwise velocity component at radial positions $z = -12d$, $z = -2d$, $z = 2d$ and $z = 6d$ along the nozzle, where d is the diameter of the throat. Comparing Spalart–Allmaras (SA) standard (std) and optimised (opt) parameter sets. X-axis is normalised with nozzle diameter at the current z-position. Squares represent experimental data.

## 5.3. Discussion

The results presented in this Chapter clearly show the potential of a non-deterministic optimisation method in finding improved model coefficents for turbulence closure formulations. The robustness of the method guarantees to produce solutions for each of the problems approached here. On the down side the computation of the optimum is expensive in relation to the time used to simulate the actual flow problem. That makes it only useful to realistically sized engineering problems, if a long term benefit can be expected from the model optimisation. Its ability to function without knowledge of the solution space topology, on the other hand, allows for a very versatile use of the method. Approaching a similar task with traditional, deterministic algorithms is not likely to be fruitful if the complexity of the problem is high. This is especially the case with multi–objective optimisation tasks.

The examples shown above do reveal the lack of universality that is inherent to all turbulence closure formulations. Knowing that this can also be expected from any models developed in the future, genetic algorithms can be used to help with the initial adjusting of these new models to canonical flow types using a multi-objective approach.

# 6. Mesh Generation Quality Optimisation

*It doesn't make a difference how beautiful your guess is. It doesn't make a difference how smart you are, who made the guess, or what his name is. If it disagrees with experiment, it's wrong.*

Richard P. Feynman

## 6.1. Motivation

Creating a computational mesh for a CFD simulation is a tedious process, especially for complicated geometries with small angles, many spherical surfaces or small wall-to-wall distances. It takes a lot of experience and patience to manually build a mesh from scratch. There are many software tools to support the engineer in the process and as many different proprietary data formats how meshes are stored on the computer. Until recently it was neccessary to create meshes in external applications and convert them to OpenFOAM format. OpenFOAM offers a wide range of conversion programs from the most common software vendors like, for example, StarCD, Gambit or CFX. From early on OpenFOAM had its own mesh creation tool called `blockMesh`. But, as the name suggests, this is used for block structured meshes and since it is completely file driven with no GUI or any other visual design support, creating meshes with `blockMesh` is not very comfortable and for more complex geometries just unfeasible. Since version 1.6 of OpenFOAM a new tool is available called `snappyHexMesh`. The idea is that in order to create a mesh around a solid body one would need the geometry of the body's surface (usually in STL[1] file format) and a simple hexagonal mesh describing the computational domain. The algorithm would then try to align the edges of the mesh to the surface of the body by 'snapping' grid points onto the STL surface. The quality of the resulting mesh is improved with local mesh refinement and re-iteration of the snapping algorithm until a user defined termination criterion is reached.

The `snappyHexMesh` tool is very interesting from an optimisation point of view, because it uses a long list of control parameters to steer the algorithm towards a result of reasonably good quality. Since a lot of these parameters' influence is not easily predictable, changing them might have devastating effects on the outcome. The mesh generation is very time and memory consuming and sometimes, while the overall mesh quality might increase by running an additional iteration, local quality requirements might be violated. Tuning all available parameters manually is pretty much a trial-and-error procedure. Hence the idea is to let a genetic algorithm do the work of finding the optimal settings for good mesh quality while keeping the size of the mesh manageable.

### 6.1.1. snappyHexMesh Algorithm

Meshing a geometry using `snappyHexMesh` is divided into three distinct parts, which are building up on each other. The first part is the castellation of the background mesh. In this step the algorithm identifies the cells of the original mesh that are intersected by edges of the surface geometry. These cells are then refined by repeated cell splitting. Hexagonal cells are split into eight refined cells, by splitting each edge of the cell and connecting the newly created points with the cell centre, thus creating eight internal faces which are then connected to make up eight new cells. The minimum and maximum level of refinement can be defined in the dictionary `snappyHexMeshDict`. Further surface refinement can be achieved by providing feature edges of the target geometry to maintain the curvature of the cells. These feature edges can be constructed automatically, using OpenFOAM's

[1]STL stands for STereoLithographie. It is a hierarchical description language defining vertices and using these vertices to define faces and then combining faces to make up solids.

`surfaceFeatureExtract` application. To finish the castellation step, all cells that lie outside of the refined surface, controlled by the definition of a point in the target mesh in the dictionary, are removed from the mesh.

The next step is the snapping of the outer gridpoints to the target surface. Here it is important to capture the features of the geometry. An iterative process of mesh movement, cell refinement and face merging dictates the quality of the result. Parameters like the number of iterations and the mesh quality constraints are again defined in the dictionary.

In a final and optional step, cell layers can be added to the surface to move the mesh away from the boundary to specifically refine a boundary layer. The surface that shall be treated and the number of layers to be added is user-defined. To control the size of the resulting grid the layer addition and the refinement parameters are the major adjustable settings. For enhanced grid quality, i.e. capturing the target surface and maintaining a grid that is suitable for a finite volume computation, the snapping parameters are most important. The following section lists a selection of parameters from the snappyHexMesh dictionary, explains their meaning in the meshing process and gives a value range used in the optimisation.

### 6.1.2. snappyHexMesh Parameters

The following section describes ther influence of each parameter on the mesh generation process. All these parameters were used as decision variables in the optimisation and Table 6.1 lists these variables and their value constraints used in the optimisation.

#### 6.1.2.1. Mesh Quality Controls

**maxNonOrtho** Non-orthogonality measures the angle between two faces of the same cell. In a grid with only rectangluar cells the value would be zero. Any deviation from this counts as non-orthogonal. High values mean there are very low angles that usually occur in a prism layer.

**maxSkewness** Skewness is a ratio between the largest and the smallest face angles in a cell. A velue of 0 is the perfect cell and 1 is the worst. For tetrahedral cells the value should not be greater than 0.95 to ensure accuracy of the calculation. Within the dictionary different quality constraints can be assigned to boundary cells and internal cells. Because in a simple geometry the cells on the boundaries are more likely to be affected by skewness problems, only this value was part of the optimisation. 6.3.1 explains this property in more detail.

**minVolRatio** The ratio in cell volume between adjacent cells should not be too large. A large aspect ratio leads to interpolation errors of unacceptable magnitude.

#### 6.1.2.2. Snap Controls

**nSmoothPatch** Number of patch smoothing operations before a corresponding point is searched on the target surface. Smooth patches are more likely to be parallel to the target surface, making it more probable to find a matching point.

**nRelaxIter** Number of iterations to relax the mesh after moving points. When points are snapped to the target, the displacement propagates through the underlying layers of points that are not on the surface. By relaxing this propagation, a smoother displacement can be achieved.

**nFeatureSnapIter** The total number of iterations trying to snap points to the target. If no sufficient quality is reached after nFeatureSnapIter iterations, the snapping is cancelled and the last state is recovered.

#### 6.1.2.3. Castellated Mesh Controls

**resolveFeatureAngle** Maximum level of refinement is applied to cells that intersect with edges at angles exceeding this value.

**Table 6.1.:** Value constraints for the objective variables in mesh generation optimisation. Accuracy value of 1 signifies an integer variable.

| Parameter | min value | max value | accuracy |
|---|---|---|---|
| maxNonOrto | 30 | 80 | 1 |
| maxSkewness | 0.5 | 1 | 0.01 |
| minVolRatio | 0.01 | 0.1 | 0.01 |
| nSmoothPatch | 5 | 50 | 1 |
| nRelaxIter | 3 | 15 | 1 |
| nFeatureSnapIter | 10 | 30 | 1 |
| resolveFeatureAngle | 30 | 80 | 1 |

## 6.2. Optimisation Objectives

When generating a mesh with `snappyHexMesh` one tries to find the trade-off between mesh quality and a feasible number of cells. These two criteria are usually not brought together easily. As one can imagine, with an unlimited number of cells even the most complex surfaces could be captured accurately. On the other hand introducing too many cells in the refinement process will lead to unfavourable cell shapes and will also increase the cost of the simulation or even break it completely. That calls for a multi–objective optimisation approach that will try to balance these competing objectives to find meshes of desirable quality and size. The actual criteria to look at in this problem will be the total number of cells in the final grid, the results of the snapping algorithm in terms of deviation from the original STL surface and overall mesh quality.

So the three objectives for the multi–objective optimisation are

i) total number of mesh cells

ii) accuracy of snapping to STL surface

iii) mesh quality

## 6.3. Fitness Function

Three fitness functions were implemented to solve this problem. One each for the three optimisation objectives mentioned above. Because of the strucure of the algorithm, the fitness evaluation operator had to be defined such that it would try to minimise the value of each objective function. Multi-objective optimisation with mixed objective value interpretation, where for example one objective value has to minimised while another one has to be maximised, is not possible in the current implementation of the NSGA-II algorithm.

To get a measure of the achieved mesh quality the output of `snappyHexMesh` is logged and analysed once the tool has terminated. Another source of quality information is the output of `checkMesh`. This OpenFOAM application summarises some mesh statistics and logs them into a file that is read and evaluated by the fitness function. This is admittedly not the best way to obtain the data because it depends on the output having a specific format and order, but it is efficient and saves time and memory. The reasoning is, that the log file would be written anyway by the `checkMesh` utility and obtaining the desired information another way would require to load the mesh into memory again and execute the necessary OpenFOAM functions.

The first objective is the easiest to evaluate: total number of grid cells. Because `snappyHexMesh` reads a parameter from the dictionary file that controls the maximum mesh size and uses it as a termination criterium, the number of cells is both an optimisation objective and a variable in the decision space. The second objective is cell quality and to measure that, the output of `checkMesh` is analysed further. Listing 6.1 shows an example of such a log file. Over the complete mesh the checking application gathers information on cell shape and connectivity and calculates the total range of occuring values. The upper and lower limits of these cell quality measurements can now be considered as optimisation variables, e.g. maximum cell skewness or minimum cell volume.

**Listing 6.1:** Excerpt from output generated by `checkMesh`

```
Checking geometry...
    Overall domain bounding box (0 0 0) (2.362 0.16 0.001)
    Mesh (non-empty, non-wedge) directions (1 1 0)
    Mesh (non-empty) directions (1 1 0)
```

```
All edges aligned with or perpendicular to non-empty
   ↩ directions.
Boundary openness (1.68653e-19 3.41982e-18 2.81651e-15)
   ↩ OK.
Max cell openness = 1.66354e-16 OK.
Max aspect ratio = 148.524 OK.
Minumum face area = 4.46498e-08. Maximum face area =
   ↩ 5.98541e-05.  Face area magnitudes OK.
Min volume = 4.46498e-11. Max volume = 5.98541e-08.
   ↩ Total volume = 0.00037792.  Cell volumes OK.
Mesh non-orthogonality Max: 0 average: 0
Non-orthogonality check OK.
Face pyramids OK.
Max skewness = 5.59845e-05 OK.
Coupled point location match (average 0) OK.
```

The evaluation of a solution's fitness now depends on how a quality value needs to be interpreted. In case of cell volume, for example, good fitness would mean that the minimum volume is not lower than a given value, while the average cell volume lies within a certain range of values. All these individual fitness measures then have to be accumulated into one number that represents the mesh quality, i.e. the third objective in the multi–objective optimisation. Agreement with the quality constraints of each parameter calculated by the `checkMesh` utility was not realised as a different objective function for each value. Instead the grades of agreement (or disagreement) were combined into a single fitness value. To account for different orders of magnitude in the actual calculated numbers, the fractional biased error was used to limit the fitness value for each entry to a certain range. Equation 6.1 shows how such a value is computed per quality constraint. The symbol $\xi_O$ represents the observed value obtained by running `checkMesh` and $\xi_P$ is the prescribed value set in an optimisation objective.

$$FB(I) = 2 \times \frac{\xi_O - \xi_P}{\xi_O + \xi_P} \tag{6.1}$$

The advantage of the fractional bias is that it limits the values to the interval $[-2, 2]$. The sign just represents the direction of disagreement and a value of zero means a total agreement of prescription and observation. If the direction is not of interest, the bias can be squared to assure positive numbers only. The fractional bias is a useful method to compare real data with predicted data, because it equally weighs positive and negative bias estimates.

The last objective considered is the accuracy of the snapping algorithm, that means how close does the resulting mesh coincide with the desired surface. To quantify this criterion the distance between external mesh faces and any of the STL surfaces is measured and the sum of all these distances represents the fitness value. This is of course limited to the cells that are near an STL surface in their normal direction. To this end an application was developed within the OpenFOAM framework that loops over all exterior faces and calculates the distance to the nearest STL patch. Exterior faces in this sense are those, that lie on the surface of the domain.

## 6.3.1. Mesh Quality

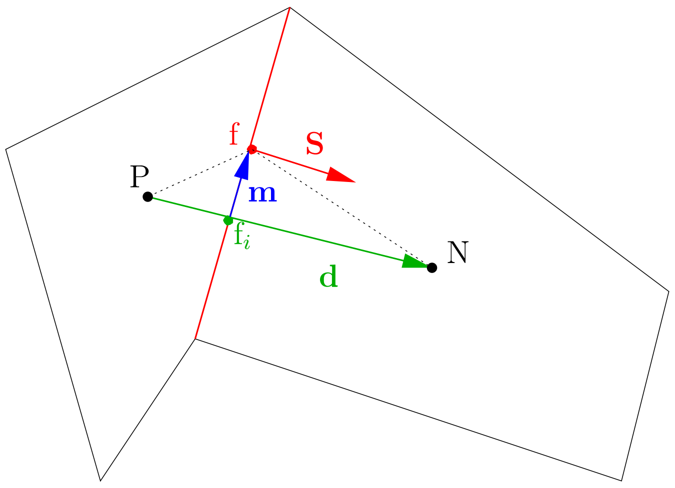


**Figure 6.1.:** Determining skewness on a face

The skewness condition needs a closer investigation. The skewness error is another numerical diffusion-type error emerging from the finite volume discretisation [50]. Figure 6.1 shows a typical situation causing the skewness error in two adjacent cells P and N connected by a face with centre f, and face area vector **S**. The value of the face integral requires the variable value at point f.

$$\int_f d\mathbf{S}\phi = \mathbf{S}\phi_f \tag{6.2}$$

In the finite volume implementation the value $\phi_f$ is often calculated from a linear interpolation between points $P$ and $N$. This yields the value of $\phi$ at the point $f_i$, which is not necessarily equal to $f$. The error $E_S$ of the convection term in Eqn. 3.32 is estimated as:

$$E_S = \sum_f \mathbf{S} \cdot [(\rho \mathbf{U})_f \mathbf{m} \cdot (\nabla \phi)_f] \,. \tag{6.3}$$

On meshes of reasonable quality, $|\mathbf{m}|$ should be much smaller than $|\mathbf{d}|$, but when this condition is no longer met, as in very skewed meshes, the influence of $\mathbf{m}$ in Eqn. (6.3) becomes more significant. The accuracy will suffer when the mesh is highly skewed. This results mainly from the way in which the face–centered pressure gradients are computed using cell–centered pressure values. Usually a second order central-difference approximation is used and the accuracy might drop to first order for very high skewness [112]. In other words, skewness is a measure of how far off the face center between two adjacent cells does the connecting vector $d$ of the two cell centers intersect the face.

A similar measure is non–orthogonality, which describes the angle between the vector $d$ and the face normal $S$. In a good quality mesh, these two vectors should be parallel, i.e. $d$ is orthogonal to the face. Since the diffusive terms in the finite volume discretisation of the Navier-Stokes-Equationsin OpenFOAM use the face normal vector to calculate fluxes between cells, it is desirable to minimise non–orthogonality.

## 6.4. Genetic Algorithm Setup

As mentioned above, the mesh creation optimises towards multiple objectives. The NSGA-II algorithm used is described in detail in Section 4.5.1. Running `snappyHexMesh` on a case with a target size of about 250,000 cells is computationally very expensive in terms of time and memory. To save disk space the workflow was slightly modified so that only the Pareto optimal individuals of each generation are physically stored, while the others are deleted after their evaluation and before the evolution proceeds to the next generation. Since the coefficients of each individual in every generation are logged anyway, this could yet be improved by not storing any meshes, but reconstruct a solution on demand using the values stored in the log file.

The parameters that were subject to the optimisation can be split into two groups: cell quality and snapping accuracy. For the first group of cell quality the `snappHexMesh` sub-dictionary `meshQualityControls` contains the values that were of interest here. From experience using snappyHexMesh and because the bearing test case was a rather simple geometry without any sharp angles, the constraints listed in Table 6.2 were considered.

**Table 6.2.:** Mesh quality settings in snappyHexMesh

| Parameter | Min | Max |
|---|---|---|
| maxBoundarySkewness | 1.1 | 2.4 |
| maxNonOrtho | 40 | 80 |
| minVolRatio | 0.01 | 0.1 |

## 6.5. Test Cases

### 6.5.1. Bearing

This simple test case is comprised of two pipes of different diameter that are connected by a planar disk. The inside of this assembly is to be meshed using snappyHexMesh. Figure 6.2 shows the three parts and how they are arranged in the structure. A detailed view of the connector disk (Figure 6.3) reveals a chamfer at the inlet to the smaller pipe. From a meshing standpoint this geometry is relatively easy to describe, but contains a few difficulties that can have severe impacts on the mesh quality. For example where the base of the bigger pipe meets the connector disk, a combination of straight and curved edges in one cell is required. The curvature should be captured by all cells along the joint and should be reasonably smooth to represent good cell quality. On the other hand around the chamfer different angles between faces have to be created to fully capture the geometry change in this area. While being a rather simple geometry, it offers enough difficulties for an automatic mesh generator to be of academic interest here.

The initial rectangular mesh outlined on the left of Figure 6.2 was created using OpenFOAM's blockMesh utility. It consists of 1372 cells, or 28 by 28 by 14 in three dimensions. The axial direction of the tubes is the z-axis. The target mesh size was limited to 200,000 cells in the snappyHexDict with refinement along the tube walls and around the diameter change at the position of the connector.

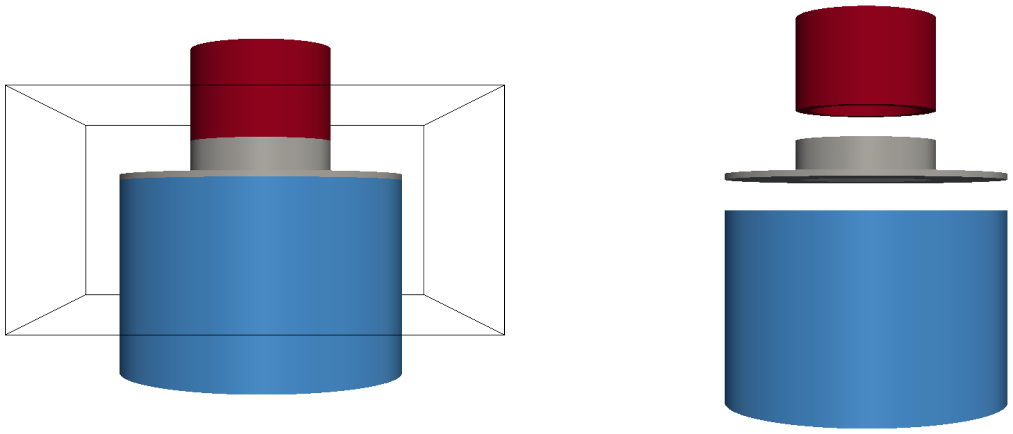


**Figure 6.2.:** Geometry of the snappyHexMesh bearing test case. The black box on the left is the outline of the original mesh that will be snapped to the inside of the geometry. The right image shows the three parts that make up the bearing.

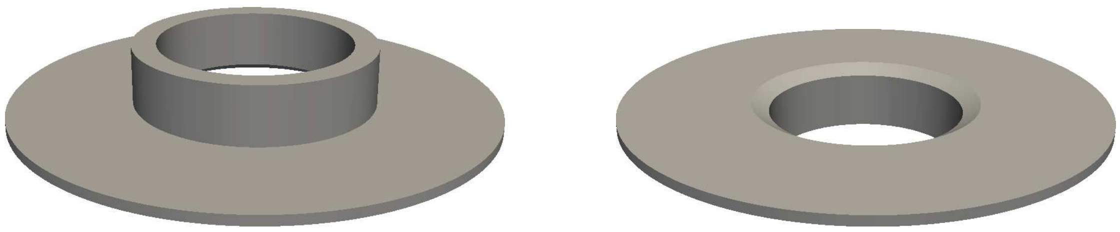


**Figure 6.3.:** Detailed view of the connector disk's top and bottom side showing the chamfered edges.

#### 6.5.1.1. Results

The decision if a mesh created with snappyHexMesh is of good quality can not automatically be decided by the computer. Even if all mesh quality constraints are met and the snapping accuracy is good, the resulting meshes might still be very different. From a CFD engineering point of view it does not make sense to tighten the requirements too much. Even with a weak constraint on, for example, the cell orthogonality, the results of the simulation and the numerical behaviour will most likely still be within the desired range. Choosing one set of parameters from the Pareto set should be done manually, for example using a visualisation tool.

Table 6.3 shows the final parameter settings in the `snappyHexMeshDict`. The bad quality example was randomly selected from the dominated population of the last generation and the good example was taken from the Pareto front. The results of the mesh optimisation are visualised in Figure 6.4. These images should highlight those parts of the mesh that are clearly of different quality. The total number of cells was almost identical in both meshes, with 60,452 in the bad example versus 62,195 in the optimal case. Comparing the parameter settings in all individuals of the Pareto front showed that for the `minVolRatio` the value was always 0.01 or very close to it. It can be assumed that this is actually the optimal setting for this parameter. Table 6.3 lists the meshing parameters for these two example meshes as well as the value ranges found in the Pareto front of the final generation.

**Table 6.3.:** Parameter settings for `snappyHexMesh` for the bearing test case reffering to the two examples depicted above and value ranges in the Pareto front.

| Parameter | bad example | good example | Pareto range |
|---|---|---|---|
| maxNonOrtho | 70 | 72 | 60–79 |
| maxSkewness | 6.0 | 10.7 | 8.0–12.3 |
| minVolRatio | 0.07 | 0.01 | 0.01–0.03 |

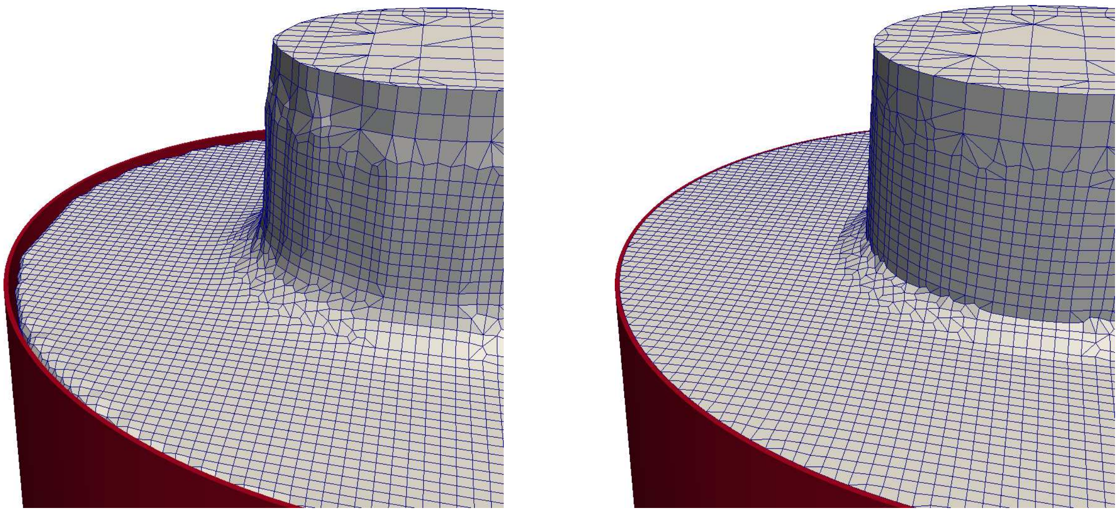


**Figure 6.4.:** Examples for bad (left) and good (right) snapping quality at the intersection of the large tube (red) and the connector disk in the bearing test case.

### 6.5.2. Ahmed Body

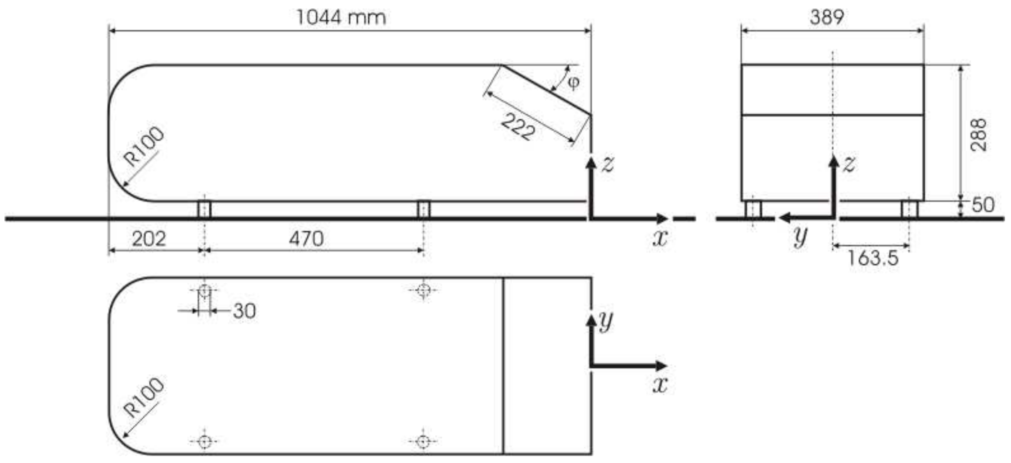


**Figure 6.5.:** Geometry of the Ahmed body as a simplified car model for aerodynamic investigations.

The characteristics of the *Ahmed* body were first described by Ahmed [2] in an experimental paper. It has become a well documented benchmark test case for car aerodynamics and is widely used to test turbulence models or other modelling techniques. Also many experimental data sets are available (e.g. [63, 62]). To accurately predict lift and drag coefficients, as these are important quantities in automobile aerodynamics, good grid quality has to be assured especially in the area of eddy detachment at the back of the car and also on the underside of the body. This is even more the case for Large-Eddy Simulations as performed on this test geometry by various researchers [45, 57, 75]. The geometry pictured in Figure 6.5 was used here, again to test the meshing quality of `snappyHexMesh`.

As was the case for the bearing discussed in the previous chapter, the Pareto set after the end of the optimisation procedure was rather large. In this case it still contained up to 50 % of the total population which were identified as being mutually non–dominant. This could mean, that the parameters modified in the `snappyHexMeshDict` had little or no influence on the outcome of the meshing process. Or it could be that creating a really 'bad' mesh for this geometry was actually difficult. One explanation for the latter could be that the fitness measurements as defined in the previous chapter were insufficient to identify discrepancies between target and result. In comparison to the bearing case, bad mesh quality would be very localised, mainly around the 'wheels' at the bottom of the body. If the quality restrictions were met on the majority of the surface, maybe small local errors do not influence the fitness very much. Unfortunately, there was not enough time and resources within the scope of this study to find the reason behind these optimisation difficulties. This could be a subject for future contributions.

#### 6.5.2.1. Results

The initial rectangular mesh created with OpenFOAM's blockMesh utility consisted of 12,000 cells, or 40 by 30 by 10 in three dimensions. Figure 6.6 shows the results of the `snappyHexMesh` optimisation around the body's wheels while Figure 6.7 highlights the curved edge of the rear end of the body. It can be seen that the parameter values listed in Table 6.4 did not only better capture the feature edges, but also led to more cells in the resulting mesh. Actually the bad quality example had a total of 163,723 cells, while the example taken from the Pareto front consisted of 632,073 cells. If such a large difference in grid size is not desired, the total number of cells could be used as another fitness requirement. Also in this test case a larger number of parameters was subject to the optimisation. A total of six values was modified, this time not only taken from the mesh quality sub-dictionary, but also from some controlling the castellation and the snapping procedure. The respective sub-dictionaries and the prescribed values are listed in Table 6.4.

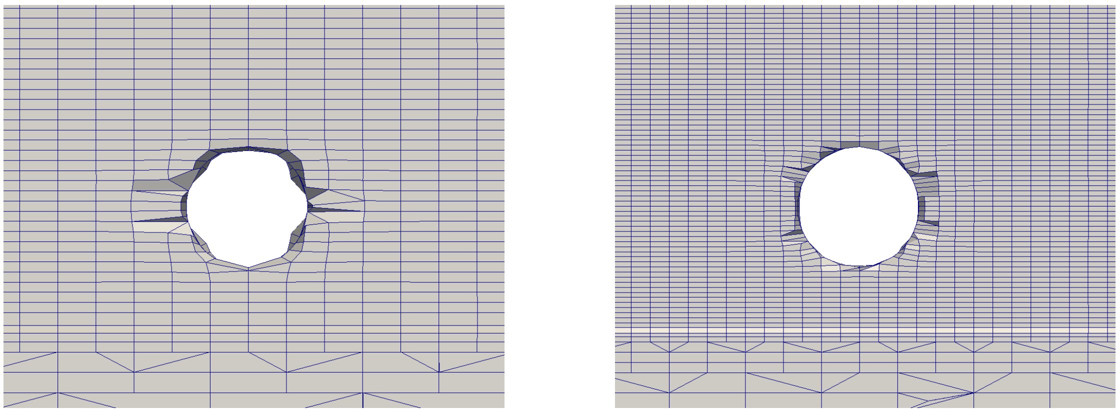


**Figure 6.6.:** Examples for bad (left) and good (right) snapping quality in the wheel region of the Ahmed body.

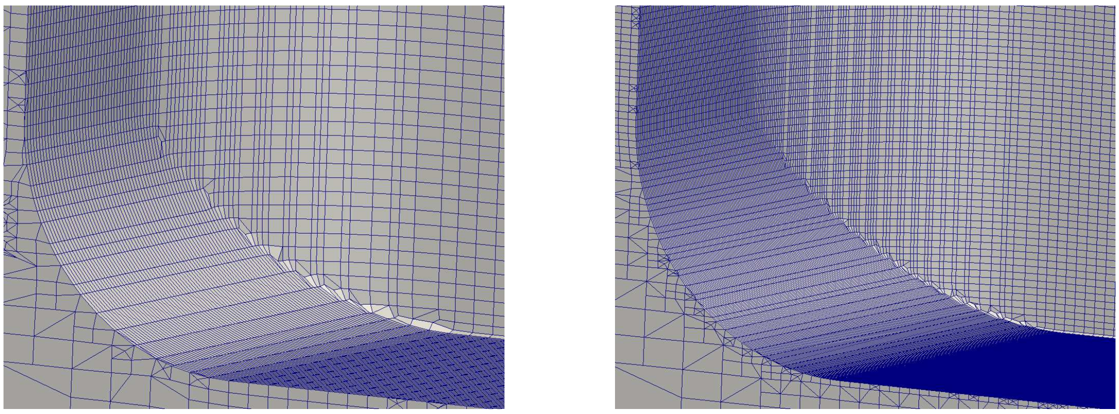


**Figure 6.7.:** Examples for bad (left) and good (right) snapping quality in the rear region of the Ahmed body.

**Table 6.4.:** Parameter settings for `snappyHexMesh` for the two examples of the Ahmed body test case depicted above.

| Parameter | bad example | good example |
|---|---|---|
| **castellated mesh controls** | | |
| resolveFeatureAngle | 45 | 32 |
| **mesh quality controls** | | |
| maxNonOrtho | 65 | 80 |
| maxSkewness | 20 | 22 |
| **snap controls** | | |
| nSmoothPatch | 3 | 7 |
| nRelaxIter | 3 | 6 |
| nFeatureSnapIter | 10 | 10 |

### 6.5.3. Packed Bed

In simulations of granular media on a macroscopic scale, material granules are often modelled in an idealised manner as spheres. These spheres are then stacked to tesselate the computational domain. That leaves small spaces between individual particles which need to be meshed in order to simulate flow through such a region, known as a “packed bed” [6]. Because of the spheres only touching in one point, the cells around this connection need to be wedge shaped, resulting in high skewness and non–orthogonality. Finding a good compromise between cell shape and mesh quality is vital for a reliable numerical treatment of the flow through a packed bed. Thus, automatically generating a mesh that meets the quality requirements is a difficult task. Using a genetic algorithm to improve the mesh generation can therefore be a useful tool.

The case setup for this problem consisted of eight spheres enclosed by a rectangular box. Each of the spheres touches its three neighbouring spheres in a very small area. Figure 6.8 shows an axial and an isometric view of the geometry as well as the background mesh created with blockMesh, used in `snappyHexMesh` to confine the computational domain.

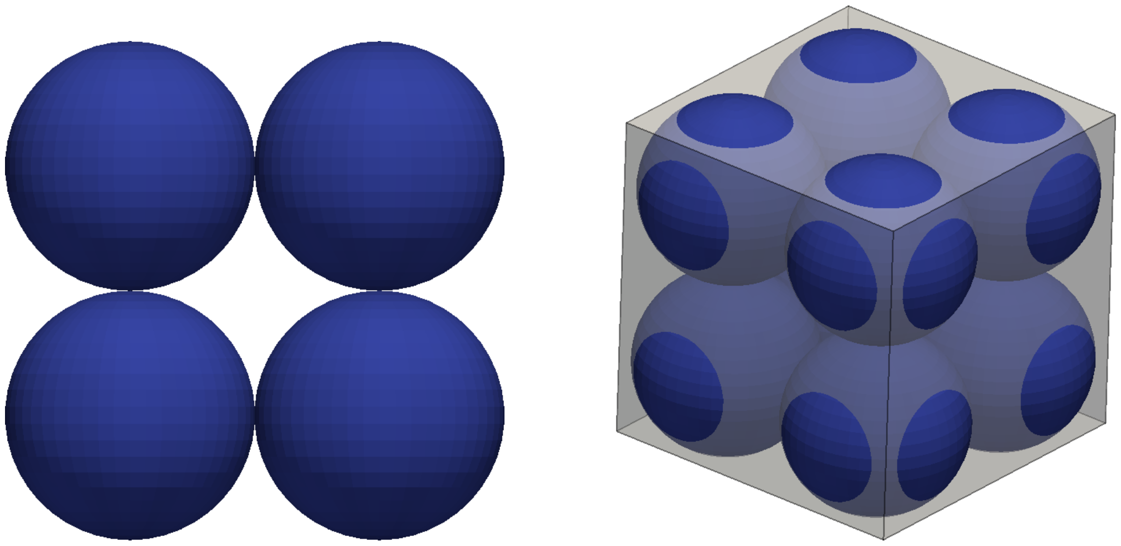


**Figure 6.8.:** Geometrical setup for the packed bed. Axial view (left) and isometric view with background mesh (right).

#### 6.5.3.1. Results

The initial mesh created with OpenFOAM's blockMesh utility consisted of 80,000 cells, or 20 by 20 by 20 in three dimensions, forming a cube with edge length $L = 1.8R$ not quite enclosing eight spheres of radius $R$. The snappyHexMesh parameters that were subject to optimisation and their allowed value ranges are listed in Table 6.5. The size of the solution space can be calculated from this table as $\approx 1.5 \times 10^{10}$. Three optimisation targets were prescribed in this case: Overall cell quality, accuracy of capturing the geometric features and total number of cells. Interestingly, just changing the quality restrictions in the snappyHexMeshDict had no influence on the resulting mesh size. Hence all individuals produced equally sized meshes, rendering the third optimisation objective obsolete.

After 25 generations the optimisation was terminated. The fitness values for the two remaining objectives of each individual on the Pareto front are shown in Figure 6.9. The actual fitness values were normalised by their maximum and minimum values, to map them onto the interval $[0, 1]$. The graph depicts the advancement

**Table 6.5.:** Optimisation parameter value ranges for the packed bed test case as defined in the `gaDict`.

| Parameter | min value | max value | accuracy |
|---|---|---|---|
| **castellated mesh controls** | | | |
| resolveFeatureAngle | 30 | 60 | 1. |
| **mesh quality controls** | | | |
| maxNonOrtho 40 | 80 | 1. | |
| maxSkewness | 2.0 | 10.0 | 0.1 |
| **snap controls** | | | |
| nSmoothPatch | 5 | 50 | 1. |
| tolerance 1. | 2.5 | 0.1 | |
| nRelaxIter | 3 | 15 | 1. |
| nFeatureSnapIter | 10 | 30 | 1. |

of the Pareto front exemplary for three generations. The second generation was chosen because it is very close to the initial, i.e. random, population. From halfway through the optimisation, the 10th generation was selected and for obvious reasons the final generation. The second generation's Pareto front only contained two elements, but the size of the front settled toward 70-80% of all individuals in the population towards the end of the optimisation. This is achieved through the crowding distance assignment described in Section 4.5.1 which assures a more balanced spread of solutions along the Pareto front.

When visualising the resulting meshes, it is possible to discern good from bad quality meshes in terms of capturing the geometric features. When looking at the thin volume in between two neighbouring spheres, the optimal shape would be a perfectly round circle with a small radius. Comparing a Pareto optimal mesh and a non–optimal mesh, as shown in Figure 6.10, one can see the higher roundness in the good mesh. Unfortunately this characteristic is not easily measurable au-

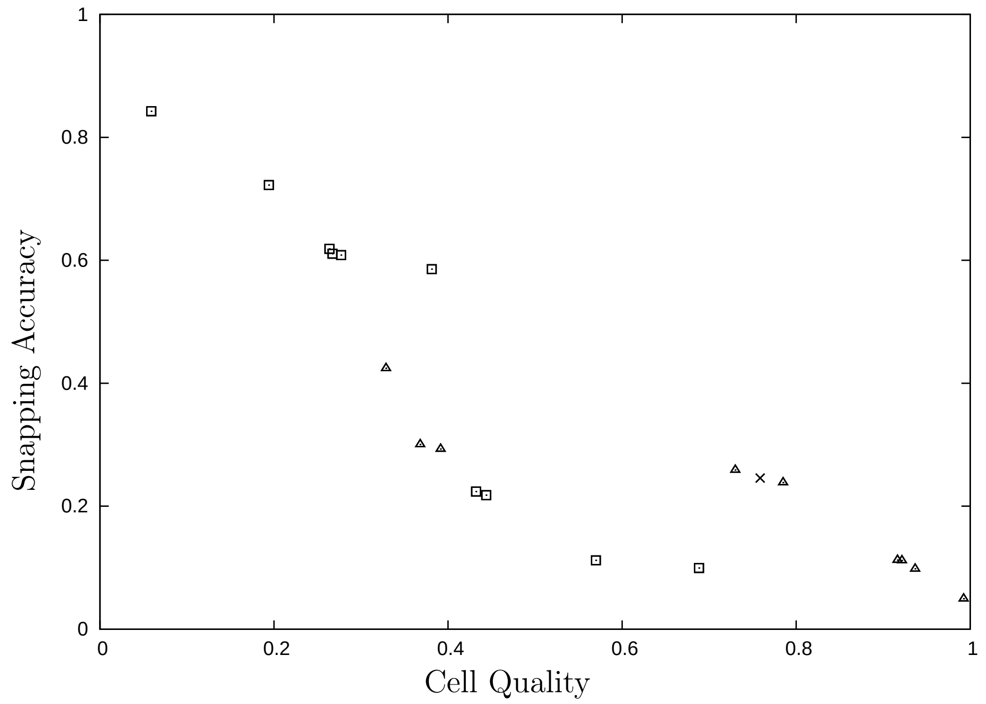


**Figure 6.9.:** Solutions of the sphere meshing optimisation on the Pareto front comparing three generations. Crosses ($\times$) show the front after the 2nd generation, triangles ($\triangle$) after the 10th, and squares ($\square$) after the final 25th generation. Objectives were normalised by their minimum and maximum values.

tomatically, otherwise it could be used as an additional optimisation objective. The results presented here with this simple test case show that with the inclusion of further parameters or objectives, a tricky geometry like a packed bed can be discretised with a good quality mesh.

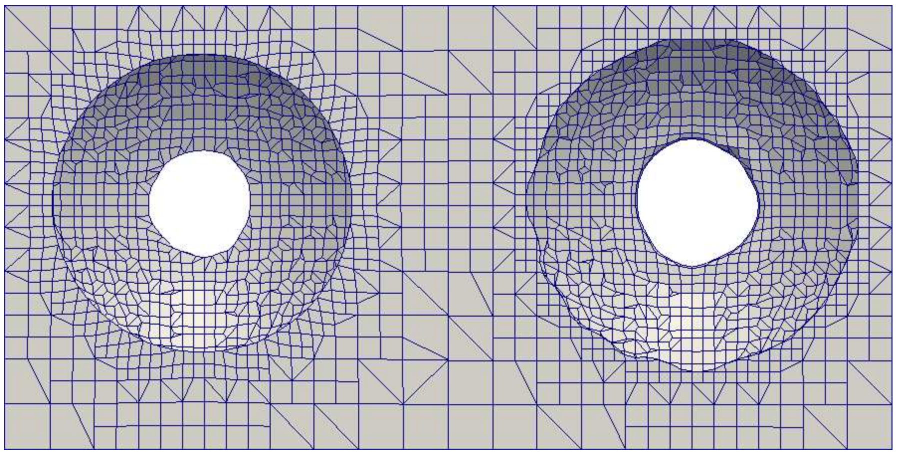


**Figure 6.10.:** Comparison of a Pareto front individual (left) versus a non-optimal solution (right). Notable is the difference in roundness and radius of the connecting area.

## 6.6. Discussion

Automated mesh generation is a highly useful tool in the pre–processing of computational fluid dynamics. Building meshes is not always part of the skill set of a CFD engineer, so that person has to rely on the generator to do a good job with respect to preset quality parameters. The studies presented in this chapter have shown that a genetic algorithm is capable of finding a satisfying solution in conjunction with a versatile and highly adjustable mesh generator like OpenFOAM's `snappyHexMesh`. It shifts the problem from finding the right setting for an algorithm that creates the mesh to defining quality requirements that the generated result has to meet, thus allowing the engineer to concentrate on the desired outcome rather than worrying about the details of the underlying algorithm.

In the work presented here, only a fraction of all possible settings were adjusted in the optimisation process. Future work on this topic will show to what extent other coefficients can be modified to further improve the quality of the generated spatial discretisation. Given the importance of the mesh quality to the accuracy and stability of a finite volume flow simulation, the amount of time needed to find the optimal mesh generator settings is well worth being invested.

# 7. Summary

*An expert is a person who has made all the mistakes that can be made in a very narrow field.*

NIELS BOHR

The method presented in this thesis is already established in other fields of research. Yet here it was for the first time applied to problems of modelling in computational fluid dynamics. The implementation of the genetic algorithm was shown to produce reliable results by testing it against academic benchmarking problems as presented in Chapter 4.6. After this proof of method, it has been applied to realistic engineering problems. The challenges discussed in this work were the optimisation of turbulence model coefficients and the optimisation of automated mesh generation. The presented studies as a whole demonstrate the capability of a genetic algorithm to find improved parameters even if the topology of the problem or solution space are unknown and if the parameters are not mutually independent.

## 7.1. Turbulence Modelling

Although it is without doubt that the closure models for the RANS equations lack a theoretical foundation and a formal connection to the solution of the Navier–Stokes equations can not be made, these models are still widely used in the description of turbulent flow phenomena. Johnson [51] is optimistic that with more computational power and further development of the models approximation comes closer to reality. I cannot share this optimism. Recent developments have shown that the models need to be modified in order to agree with corresponding experiments, ultimately creating a whole zoo of turbulence models but without progressing the basic understanding of the nature of turbulence. Unless the latter is achieved, there will be no universal model that describes the complete range of flows with relevance to the engineer. Further, tuning the coefficients to match a given problem inevitably narrows the applicability to a small subset of problems. To improve the accuracy of computational fluid dynamics w.r.t. turbulent flows further research needs to be invested in the underlying physics which could then lead to a completely different approach to a numerical formulation. But until that is done, one has to rely on the output of existing formulations and has to live with the error they incorporate.

On the other hand, using a genetic algorithm approach can simplify the task of adjusting a turbulence model to an existing flow configuration. Where tuning parameters manually or via mathematical deduction could be very difficult or even impossible, a GA finds a solution or a set of solutions that yields a satisfactory improvement to the accuracy of the simulation. Projecting these results onto similar flow problems can possibly help to get a better insight into the physical nature of the flow. When comparing Tables 5.5 and 5.7 which show the estimated optimae for the k-$\omega$ SST model in two completely different flow configurations, it can be seen that the derivation from the so called standard values is very much problem dependent. This finding justifies the use of GAs to adapt models to the flow problem at hand.

Using the findings of this thesis, further work on turbulence model optimisation can be done. It would be interesting to use multi–objective optimisation to fit one model to a series of different flow problems. The objectives would be to find the best parameter set for each type with respect to agreement with experimental or DNS data. That way a more universal model would emerge that can be applied to a wider range of problems.

## 7.2. Mesh Generation

Since spatial discretisation of the computational domain is a major part of simulations using the finite volume method, automated mesh generation with full control over grid quality requirements is a very desirable functionality. OpenFOAM's snappyHexMesh utility is a powerful tool, but because it tries to be of universal use for any kind of geometric setup, the sheer amount of possible settings makes it difficult to use in the best possible way. By employing genetic algorithm based search strategies, a prescribed level of accuracy and cell quality can be achieved, that does not require manual adjustments. Some of the benefits of using this technique are that mesh generation is still fully automated, no manual grid correction needs to be done. Also the focus can be laid on the outcome of the meshing procedure, while the process is taken care of by the GA.

Including additional aspects of the mesh generation workflow of snappyHexMesh into the parameter space of the genetic algorithm can further improve the overall quality of the resulting computational grid. For instance controlling surface refinement or refinement of local areas within the mesh could be subject to optimisation. Also running actual calculations on the meshes generated by the GA and comparing the results to those obtained with manually created meshes would allow us to quantify the reliability of a fully automated mesh generation process.

## 7.3. Conclusion

With evolution based optimisation a wider range of problems can be solved at the cost of higher computational effort. The major advantage is the fact, that no a–priori knowledge of the solution or problem space topology is required to obtain results. On the contrary, using this non–deterministic approach might even reveal insight into the structure of the problem, that could not be deduced with the knowledge available before. This research has proven that genetic algorithms are a useful addition to the tool set available to the CFD engineer. Not only are they powerful enough to generate optimised solutions that would be difficult if not impossible to find with deterministic approaches, but they are also numerically very robust and can be employed to solve problems about which not much is known in advance.

In summary, the methods and test cases presented in this thesis have shown that

- genetic algorithms are a powerful tool to identify optimised parameters in turbulence modelling
- a genetic algorithm can solve an optimisation problem if the solution space is unknown or mathematically inaccessible, like mesh generation
- with sufficient computational resources, GAs are a handy tool for engineers without much knowledge of computer science
- GAs can find solutions thast could not easily be identified with common sense and experience alone, but could be rather unexpected, like the findings from Section 4.7.

Applying evolutionary principles to problems of modeling in engineering is a new approach that is yet in its infancy, but will itself evolve to a multi–purpose tool that, given sufficient resources, can improve our knowledge about the physics whose models are operating on, or make difficult optimisation tasks more approachable.

## 7.4. Future Work

Based on the findings in this thesis further research into the matter can be done. On the methodological side it would be interesting to compare results found by other optimisation methods, both in terms of accuracy as well as computational effort. For example a swarm intelligence based method could be employed or variations of the genetic algorithm presented here.

Another interesting point that was neglected in this work is the influence of measurement errors and numerical error on the parameter optimisation in turbulence modelling. Both sources of error introduce a degree of uncertainty to the results that should be quantified mathematically, leading to a parameter range rather than specific values.

The model parameters identified in Chapters 5 to 6 should also be tested on a series of different physical problems. This thesis does not proclaim universality to the parameter values, but for similar problems to those the values were optimised for, an improvement should be observable.

During the course of further work in the field, the source code should also be expanded with more GA operators like new crossover and selection methods. The modularity of the code should make this an easy task. The performance of these new methods could then be tested against the problems discussed in this work.

# A. Using the Code

*Always be wary of any helpful item that weighs less than its operating manual.*
Terry Pratchett, Jingo

## A.1. System Requirements

The Genetic Algorithm implementation presented in this thesis requires a working installation of python 2.6 or higher. To use the parallelisation facilities it needs the `mpi4py` package and any MPI library that is supported by this package. It was tested with Open MPI version 1.4.3 that came with the OpenFOAM bundle. The `PyFoam` library is used to read dictionary files, but is not required if parameters are passed to the program in any other way.
Obviously to run OpenFOAM solvers from within the GA framework, a full installation of OpenFOAM is essential. It should work with any version that is supported by PyFoam and I used it with versions 1.7.x, 2.1.0 and 2.1.x. There is no specific hardware required to use this library.

## A.2. Setting up the GA

There are mainly three things that need to be adapted by the user to tailor the genetic algorithm to any optimisation problem. The parameters that control the behaviour of the algorithm and define the optimisation objectives are placed in the dictionary file. The selection of operators and assignment of a fitness function is placed in the main program and the actual fitness function should be defined in a separate file but can also be placed next to the main function. The following sections describe the format of these items in more detail.

## A.2.1. Dictionary File

To control the settings for the execution of a genetic algorithm an external configuration file is used. That allows changing parameters and modifying environment variables without making changes to the source code. The structure of the parameter file is in accordance with the OpenFOAM dictionary file format. The default name is `gaDict` (short for genetic algorithm dictionary). A dictionary contains keyword/value pairs, where the type of the value might be a real or integer number, a string, a list or a subdictionary. The first three are self-explanatory. A list value consists of a list name followed by a list of values of any of the above types in round brackets (). A subdictionary is defined by a unique name and a list of keyword/value pairs enclosed in curly brackets {}. Listing A.1 shows an example of a gaDict using all the types defined above. In the example `mutationProbability` is a real value, `variables` is a list and `C1, C2` etc. define a subdictionary. The string value `workingDirectory` is optional but can be used to define an output directory. The file format is flexible in a way that the user can add new entries ad libitum to be used in the main code.
The syntax to define an optimisation variable takes three items of information. The lower and upper bound define a range constraint within which the variable will lie. The accuracy sets the number of digits that are considered. Those values together define the length of a variable in the chromosome representation. For example the variable C1 in the example below is limited to the range $[0.864; 1.888]$ and the accuracy is 0.001, allowing a total number of 1,024 different values. In a bit encoded chromosome it would therefore take up 10 genes ($2^{10} = 1,024$). If the length of the interval is not a power of two, a mapping takes place to assert fulfillment of the constraint.

**Listing A.1:** Example `gaDict` file to control parameters for the genetic algorithm externally.

```
populationSize          50;
generations             30;
```

```
mutationProbability     0.03;

crossoverProbability    0.6;
tournamentSize          2;

workingDirectory        bfsRe64k-kEps;

variables
(
    C1
    {
        lowerBound   0.864;
        upperBound   1.888;
        accuracy     1.e-3;
    }
    C2
    {
        lowerBound   1.152;
        upperBound   2.149;
        accuracy     1.e-3;
    }
);
```

### A.2.2. Operator Selection

Selection of genetic operators is implemented by using function pointers within the source code. After insantiating an object of class `BasicPopulation` or one of its derivatives, it can be assigned pointers to selection and crossover functions. Standard operators for selection are roulette wheel and tournament selection, found in module `SimpleEA.SimplePopulation`. The same goes for crossover functions. To use these the modules have to be imported into the main file.

**Listing A.2:** Example setting up a simple population with selection function.

```
from SimpleEA import SimplePopulation

pop = SimplePopulation()
pop.selectionFunc = pop.tournamentSelect
pop.tournamentSize=2
{...}
mate=pop.select()
pop.crossover()
```

The function signature for selection expects no parameters and returns the index of the selected individual in the list of individuals that is a member of the population. In simple and elitist populations the crossover function neither receives nor returns any values. It rather performs the crossover operation in situ on the population's list of individuals, creating a new list that can then replace the old one at the end of one generation. Other implementations are possible by just overwriting the existing one or assigning a new crossover function to the population object. The calling methods are `select()` and `crossover()` respectively.

### A.2.3. Fitness Function

The fitness function is problem dependent, therefore it needs to be redefined for each case individually. To connect the fitness function to an individual a function pointer is assigned to objects of class `BasicIndividual`. In Python function pointers can have any signature, allowing a very flexible implementation of the fitness. That means it can have any return type and the user has to make sure to interpret the returned value appropriately. The method `fitness()` in the Individual class will call the assigned function.

**Listing A.3:** Example for defining and assigning a fitness function. Here the number of ones in the chromosome is counted as a measure of fitness

```
def maxOne(indiv):
    sums=0
    for a in indiv.chromosome:
        if a==1: sums=sums+1.
    return sums

{...}
indiv=BasicIndividual()
indiv.fitnessFunc = maxOne
{...}
fit=indiv.fitness()
```

## A.3. Execution

Running the genetic algorithm requires runnable Python code in the main module. There the population should be set up, individuals assigned and randomly generated and the main evolution loop should be placed there. The code can then simply be executed by calling `python main.py` from the commandline where `main.py` is the name of the main module. If parallel processing is required the MPI code has to be in the main module and it is simply executed by calling

```
mpirun -np 10 python main.py
```

The `-np` directive passes the number of processes to be used on to the MPI main routine.

# Bibliography

[1] *Using Genetic Algorithms for Electrode Shape Optimization in Accelerators with RF Focusing* (2012).

[2] Abe, K., and Ohtsuka, T. Some salient features of the time-averaged ground vehicle wake. Tech. Rep. SAE-Paper 840300, SAE, 1984.

[3] Abe, K., and Ohtsuka, T. An investigation of LES and hybrid LES/RANS models for predicting 3-d diffuser flow. *International Journal of Heat and Fluid Flow 31*, 5 (2010), 833–844.

[4] Ashforth-Frost, S., Jambunathan, K., and Whitney, C. Velocity and turbulence characteristics of a semiconfined orthogonally impinging slot jet. *Experimental Thermal and Fluid Science 14* (1997), 60–67.

[5] Baker, J. E. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the 2nd Int. Conference on Genetic Algorithms* (1987), L. E. Assoc., Ed., pp. 14–21.

[6] Baker, M., and Tabor, G. Computational analysis of transitional air flow through packed columns of spheres using the finite volume technique. *Computers & Chemical Engineering 34*, 6 (2010), 878–885.

[7] Baldwin, B. S., and Barth, T. J. A one-equation turbulence transport model for high reynolds number wall-bounded flows. Tech. Rep. AIAA PAPER 91-0610, AIAA, 1991.

[8] Baldwin, B. S., and Lomax, H. Thin layer approximation and algebraic model for separated turbulent flows. In *AIAA 16th Aerospace Sciences Meeting* (1978), vol. 78-257.

[9] Balzert, H. *Lehrbuch der Software-Technik : Software-Entwicklung.* Spektrum, Akadem. Verlag, 1996.

[10] Bardow, A., Bischof, C. H., Bucker, H. M., Dietze, G., Kneer, R., Leefken, A., Marquardt, W., Renz, U., and Slusanschi, E. Sensitivity-based analysis of the k-$\varepsilon$ model for the turbulent flow between two plates. *Chemical Engineering Science 63*, 19 (2008), 4763–4775.

[11] Behzadian, K., Kapelan, Z., Savic, D., and Ardeshir, A. Stochastic sampling design using a multi-objective genetic algorithm and adaptive neural networks. *Environmental Modelling & Software 24*, 4 (2009), 530–541.

[12] Blenkinsop, A., Valentin, A., Richardson, M., and Terry, J. The dynamic evolution of focal-onset epilepsies: Combining theoretical and clinical observations. *European Journal of Neuroscience 36*, 2 (2012), 2188–2200.

[13] Bradshaw, P., Ferris, D. H., and Atwell, N. P. Calculation of boundary-layer development using the turbulent kinetic energy equation. *Journal of Fluid Mechanics 28*, 3 (1967), 593–616.

[14] Cebeci, T., and Smith, A. M. O. *Analysis of Turbulent Boundary Layers.* Academic Press, New York, 1974.

[15] Chandesris, M., Serre, G., and Sagaut, P. A macroscopic turbulence model for flow in porous media suited for channel, pipe and rod bundle flows. *International Journal of Heat and Mass Transfer 49*, 15-16 (2006), 2739–2750.

[16] Cooper, D., Jackson, D., Launder, B., and Liao, G. Impinging jet studies for turbulence model assessment - I. flow-field experiments. *International Journal of Heat and Mass Transfer 36*, 10 (1993), 2675–2684.

[17] Cziesla, T., Tandogan, E., and Mitra, N. Large-eddy simulation of heat transfer from impinging slot jets. *Numerical Heat Transfer, Part A Applications 32*, 1 (1997), 1–17.

[18] Darwin, C. *On the Origin of Species by means of natural selection.* John Murray, London, 1859.

[19] Davidson, L., and Peng, S. Hybrid LES-RANS modelling: A one-equation SGS model combined with a k-$\omega$ model for predicting recirculating flows. *International Journal for Numerical Methods in Fluids 43*, 9 (2003), 1003–1018.

[20] Davidson, P. *Turbulence.* OUP Oxford, 2004.

[21] Deb, K., Anand, A., and Joshi, D. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation 10*, 4 (2002), 371–395.

[22] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.

[23] deJong, K. A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.

[24] Dorigo, M. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, Italie, 1992.

[25] Ehrhard, J., and Moussiopoulos, N. On a new nonlinear turbulence model for simulating flows around building-shaped structures. *Journal of Wind Engineering and Industrial Aerodynamics 88*, 1 (2000), 91–99.

[26] Erturk, E., Corke, T., and Gökçöl, C. Numerical solutions of 2-D steady incompressible driven cavity flow at high reynolds numbers. *Int. J. Numer. Meth. Fluids 48* (2005), 747–774.

[27] Ferziger, J. H., and Perić, M. *Computational Methods for Fluid Dynamics*, 3rd rev. ed. Springer-Verlag, Berlin, 2002.

[28] Fogel, L. J., Walsh, A. J., and Owens, A. J. *Artificial Intelligence through simulated evolution.* John Wiley, New York, 1966.

[29] Fonseca, C. M., and Fleming, P. J. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation 3* (1995), 1–16.

[30] Frisch, U. *Turbulence: The Legacy of A. N. Kolmogorov.* Cambridge University Press, 2005.

[31] Gagné, C., and Parizeau, M. Genericity in evolutionary computation software tools: Principles and case-study. *International Journal on Artificial Intelligence Tools 15*, 2 (2006), 173–194.

[32] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[33] Gartling, D. A test problem for outflow boundary conditions – flow over a backward facing step. *Int. J. Numer. Meth. Fluids 11* (1990), 953–967.

[34] Gatski, T. B., and Rumsey, C. L. Linear and nonlinear eddy viscosity models. In *Closure Strategies for Turbulent and Transitional Flows* (2002), B. Launder and N. D. Sandham, Eds., Cambridge University Press, pp. 9–46.

[35] Gen, M., and Cheng, R. *Genetic Algorithms and Engineering Optimization.* John Wiley & Sons, 2000.

[36] Ghia, K., Osswald, G., and Ghia, U. Analysis of incompressible massively separated viscous flows using unsteady navier-stokes equations. *Int. J. Numer. Meth. Fluids 9* (1989), 1025–1050.

[37] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[38] Goldberg, U. Exploring a three-equation r-k-$\varepsilon$ turbulence model. *Journal of Fluids Engineering, Transactions of the ASME 118*, 4 (1996), 795 – 799.

[39] Hakimzadeh, H. Numerical simulation of oscillatory shallow-water flow around a conical headland using the k-$\varepsilon$ and algebraic stress turbulence models. *ASME Conference Proceedings 2008*, 48227 (2008), 987–992.

[40] Han, X.-S., Ye, T.-H., Zhu, M.-M., and Chen, Y.-L. A new k-$\varepsilon$ turbulence model with compressibility modifications. *Kongqi Donglixue Xuebao/Acta Aerodynamica Sinica 27*, 6 (2009), 677–682.

[41] Hanjalić, K., and Launder, B. A reynolds stress model of turbulence and its application to thin shear flow. *Journal of Fluid Mechanics 52* (1972), 609–638.

[42] Hariharan, P., Giarra, M., Reddy, V., and Day, S. W. Multilaboratory particle image velocimetry analysis of the FDA benchmark nozzle model to support validation of computational fluid dynamics simulations. *Journal of Biomechanical Engineering 133* (2011), 410021–14.

[43] Hattori, H., and Nagano, Y. Direct numerical simulation of turbulent heat transfer in plane impinging jet. *International journal of heat and fluid flow 25*, 5 (2004), 749–758.

[44] Hilbert, R., Janiga, G., Baron, R., and Thèvenin, D. Multiobjective shape optimization of a heat exchanger using parallel genetic algorithms. *International Journal of Heat and Mass Transfer 49*, 15-16 (2006), 2567–2577.

[45] Hinterberger, C., GarcÃŋa-Villalba, M., and Rodi, W. Large eddy simulation of flow around the ahmed body. In *Lecture Notes in Applied and Computational Mechanics / The Aerodynamics of Heavy Vehicles: Trucks, Buses, and Trains* (2004), J. R. R. McCallen, F. Browand, Ed., Springer Verlag.

[46] Holland, H. J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[47] Hollstien, R. B. *Artificial genetic adaptation in cmoputer control systems.* PhD thesis, University of Michigan, 1971.

[48] Jansen, B., and Rit, V. Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns. *Biological Cybernetics 73* (1995), 357–366.

[49] Jaramillo, J., Pèrez-Segarra, C., Rodriguez, I., and Oliva, A. Numerical study of plane and round impinging jets using RANS models. *Numerical Heat Transfer 54* (2008), 213–237.

[50] Jasak, H. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows.* PhD thesis, Imperial College of Science, Technology and Medicine, London, 1996.

[51] Johnson, R. W. *The handbook of fluid dynamics.* Springer, 1998.

[52] Jones, W. P., and Launder, B. E. The prediction of laminarization with a two-equation model of turbulence. *International Journal of Heat and Mass Transfer 15* (1972), 301–314.

[53] Jourdan, L., Corne, D., Savic, D., and Walters, G. Preliminary investigation of the 'learnable evolution model' for faster/better multiobjective water systems design. In *EMO 2005*, C. A. C. et al., Ed., vol. LNCS 3410. Springer-Verlag Berlin Heidelberg, 2005, pp. 841–855.

[54] Jourdan, L., Corne, D., Savic, D. A., and Walters, G. A. LEMMO: Hybridising rule induction and NSGA II for multi-objective water systems design. In *CCWI2005, Computing and Control in the Water Industry* (2005), pp. 45–50.

[55] Kirkpatrick, S., jr., C. D. G., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[56] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[57] Krajnovic, S., and Davidson, L. Large eddy simulation of the flow around a simplified car model. In *SAE 2004 World Congress, Detroit, USA* (2004).

[58] Lai, Y., So, R., and Hwang, B. Calculation of planar and conical diffuser flows. *AIAA Journal 27*, 5 (1989), 542–548.

[59] Launder, B., Reece, G., and Rodi, W. Progress in the development of a reynolds–stress turbulent closure. *Journal of Fluid Mechanics 68*, 3 (1975), 537–566.

[60] Launder, B., and Spalding, D. *Lectures in Mathematical Models of Turbulence.* Academic Press, 1972.

[61] León-Rovira, N., Uresti, E., and Arcos, W. Fan shape optimisation using CFD and genetic algorithms for increasing the efficiency of electric motors. *Int. J. Computer Applications in Technology 30*, 1/2 (2007), 47–58.

[62] Lienhart, H., and Becker, S. Flow and turbulence structure in the wake of a simplified car model. In *SAE 2003 World Congress, Detroit, USA* (2003).

[63] Lienhart, H., Stoots, C., and Becker, S. Flow and turbulence structures in the wake of a simplified car model (ahmed model). In *DGLR Fach Symp. der AG STAB, Stuttgart University* (2000).

[64] Loshchilov, I., Schoenauer, M., and Sebag, M. Not all parents are equal for MO-CMA-ES. In *Evolutionary Multi-Criterion Optimization 2011* (2011), Springer Verlag, pp. 31–45.

[65] Makiola, B. *Experimentelle Untersuchungen zur Strömung über die schräge Stufe.* PhD thesis, Institut für Hydromechanik, Universität Karlsruhe, 1992.

[66] Makiola, B., and Ruck, B. Flow separation over the step with inclined walls. In *Near-Wall Turbulent Flows* (1993), R. So, C. Speziale, and B. Launder, Eds., Elsevier Press, p. 999.

[67] Man, K. F., Tang, K. S., and Kwong, S. Genetic algorithms: Concepts and applications. *IEEE Transactions on Industrial Electronics 43*, 5 (1996), 519–534.

[68] Marco, N., Désidéri, J.-A., and Lanteri, S. Multi-objective optimization in CFD by genetic algorithms. Tech. Rep. 3686, Institut National de Recherche en Informatique en Automatique, 1999.

[69] McRae, D., and Laflin, K. Dynamic grid adaption and grid quality. In *Handbook of Grid Generation*, J. Thompson, B. Soni, and N. Weatherhill, Eds. CRC Press, 1999, ch. 34.

[70] Menter, F. Two-equation eddy-viscosity turbulent models for engineering applications. *AIAA Journal 32* (1994), 1598–1605.

[71] Menter, F. R., Kuntz, M., and Langtry, R. *Ten Years of Industrial Experience with the SST Turbulence Model.* Turbulence, Heat and Mass Transfer 4. Begell House, Inc, 2003.

[72] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2.* High-Performance Computing Center Stuttgart, 2009.

[73] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, Heidelberg, 1996.

[74] Michalewicz, Z., and Schoenauer, M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation 4*, 1 (1996), 1 – 32.

[75] Minguez, M., Pasquetti, R., and Serre, E. High-order large-eddy simulation of flow over the 'ahmed body' car model. *Physics of Fluids 20*, 9 (2008), 095101.

[76] Moin, P., and Mahesh, K. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics 30* (1998), 539–78.

[77] Olsson, M., and Fuchs, L. Large eddy simulations of a forced semiconfined circular impinging jet. *Physics of fluids 10* (1998), 476.

[78] Ortiz-Boyer, D., Hervás-Martínez, C., and García-Pedrajas, N. Cixl2: A crossover operator for evolutionary algorithms based on population features. Tech. rep., University of Cordoba, Spain, 2005.

[79] Petersen, M. R., Hecht, M. W., and Wingate, B. A. Efficient form of the LANS-$\alpha$ turbulence model in a primitive-equation ocean model. *Journal of Computational Physics 227*, 11 (2008), 5717–5735.

[80] Pope, S. An explanation of the turbulent round-jet/plane-jet anomaly. *AIAA Journal 16* (1978), 279–281.

[81] Pope, S. B. *Turbulent Flows.* Cambridge University Press, 2000.

[82] Prandtl, L. no title. In *Proceedings of the second international congress in applied mechanics* (Zurich, 1926), p. 341.

[83] Prandtl, L. Über ein neues Formelsystem für die ausgebildete Turbulenz. *Nachrichten der Akademie der Wissenschaften zu Göttingen* (1945), 6–19. (german).

[84] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Technical University of Berlin, Germany, 1971.

[85] Rechenberg, I. *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann Holzboog, 1973.

[86] Reynolds, O. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society London A 186* (1895), 123–164.

[87] Rumsey, C. L. Apparent transition behavior of widely-used turbulence models. *International Journal of Heat and Fluid Flow 28* (2007), 1460–1471.

[88] Schwefel, H.-P. *Evolutionsstrategie und numerische Optimierung.* PhD thesis, Technical University of Berlin, Germany, 1975.

[89] Schwefel, H.-P. *Evolution and Optimum Seeking.* Wiley & Sons, New York, 1994.

[90] Schwefel, H.-P. Artificial evolution: How and why? In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, Eds. John Wiley & Sons Ltd., Chichester, 1998.

[91] Smith, E., Mi, J., Nathan, G., and Dally, B. The ”round jet inflow-condition anomaly” for the k-e turbulence model. Tech. Rep. unpublished, University of Adelaide, 2012.

[92] So, R., Lai, Y., Zhang, H., and Hwang, B. Second-order near-wall turbulence closures. a review. *AIAA journal 29*, 11 (1991), 1819–1835.

[93] Spalart, P. R., and Allmaras, S. R. A one-equation turbulence model for aerodynamic flows. *AIAA Paper 92* (1992), 439.

[94] Speziale, C. G., and Thangham, S. Analysis of an RNG based turbulence model for separated flows. Tech. Rep. ICASE Report No 92-3, NASA Langley Reseach Center, 1992.

[95] Srinivas, N., and Deb, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation 2*, 3 (1995), 221–248.

[96] Stewart, S., Paterson, E., Burgreen, G., Hariharan, P., Giarra, M., Reddy, V., Day, S., Manning, K., Deutsch, S., Berman, M., Myers, M., and Malinauskas, R. Assessment of CFD performance in simulations of an idealized medical device: Results of FDA's first computational interlaboratory study. *Cardiovascular Engineering and Technology* (2012), 1–22.

[97] Storn, R., and Price, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11* (1997), 341–359.

[98] Tagen, A. *Performance comparison DE vs PSO for benchmark function: De Jong's in [-50:50].* PhD thesis, University of Trento, Italy, 2013.

[99] Takatsu, Y., Masuoka, T., and Tsuruta, T. Turbulence model for flow through porous media. *Nippon Kikai Gakkai Ronbunshu, B Hen/Transactions of the Japan Society of Mechanical Engineers, Part B 60*, 571 (1994), 965–970.

[100] Teruel, F. E., and Rizwan-uddin. A new turbulence model for porous media flows. Part I: Constitutive equations and model closure. *International Journal of Heat and Mass Transfer 52*, 19-20 (2009), 4264–4272.

[101] Thangam, S., and Speziale, C. G. Turbulent flow past a backward-facing step - a critical evaluation of two-equation models. *AIAA Journal 30* (May 1992), 1314–1320.

[102] Versteeg, H. K., and Malalasekera, W. *An introduction to Computational Fluid Dynamics*, 2nd ed. Pearson Prentice Hall, 2007.

[103] Voke, P. R., and Gao, S. Numerical study of heat transfer from an impinging jet. *International journal of heat and mass transfer 41*, 4 (1998), 671–680.

[104] W., Q., and J., C. Parameter estimation of engineering turbulence model. *Acta Mechanica Sinica 17*, 4 (2001), 302–309.

[105] Weisstein, E. W. Gray code, Last Accessed: 04.03.2012. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/GrayCode.html.

[106] Wendling, F., Bartolomei, F., Bellanger, J., and Chauvel, P. Epileptic fast activity can be explained by a model of impaired GABAergic dendritic inhibition. *European Journal of Neuroscience 15* (2002), 1499–1508.

[107] White, J., Chow, C., Ritt, J., Soto-Trevino, C., and Kopell, N. Synchronization and oscillatory dynamics in heterogeneous, mutually inhibited neurons. *Journal of Computational Neuroscience 5* (1998), 5–16.

[108] Whitley, D. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd Int. Conference on Genetic Algorithms* (1989), J. D. Schaffer, Ed., pp. 116–121.

[109] Whittington, M., Traub, R., Kopell, N., Ermentrout, B., and Buhl, E. Inhibition-based rhythms: experimental and mathematical observations on network dynamics. *International Journal of Psychophysiology 38* (2000), 315–336.

[110] Wilcox, D. C. *Turbulence Modeling for CFD*, 3rd ed. DCW Industries, Inc., La Cañada, 2006.

[111] Xuesong, Y., Wei, W., Qingzhong, L., Chengyu, H., and Yuan, Y. Designing electronic circuits by means of gene expression programming II. In *Proceedings of the 7th international conference on Evolvable systems: from biology to hardware* (Berlin, Heidelberg, 2007), ICES'07, Springer-Verlag, pp. 319–330.

[112] Zang, Y., Street, R. L., and Koseff, J. R. A non-staggered grid, fractional step method for time-dependent incompressible navier-stokes equations in curvilinear coordinates. *Journal of Computational Physics 114*, 1 (1994), 18–33.

[113] Zitzler, E., Deb, K., and Thiele, L. *A Tutorial on Evolutionary Multiobjective Optimization.* Swiss Federal Institute of Technology (ETH), Computer Engineering and Networks Laboratory (TIK), Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2002.

[114] Zitzler, E., Laumanns, M., and Bleuler, S. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation 8* (2000), 173–195.