Thesis for the Degree of Master of Science

Black Box Optimization in Engine Development

Carl Bohman **Jorild Engkvist**

Department of Engineering Mathematics
Chalmers University of Technology
Göteborg, Sweden

Volvo Car Corporation

Carried out on assignment of Volvo Car Corporation
Göteborg, Sweden

Supervisor: Johan Lennblad, Tech. Lic., VCC
Supervisor: Ann-Brith Strömberg, Tech. Dr., FCC
Supervisor and Examiner: Professor Michael Patriksson, Chalmers

Göteborg, June 2004

Abstract

This thesis discusses the possibility to use modern optimizations methods in engine development and more specifically how to optimize a five cylinder, naturally aspirated engine when serial intake valves (SIV) have been added to its configuration. Both the SIV settings and the hardware of the engine can be optimized.

The optimization problem to be solved is a "black-box" optimization problem, since the objective function's form as well as its derivatives are unknown. Also, each function value takes considerable time to compute. We evaluated five algorithms on a simpler engine problem and found that Powell's NEWUOA and Gutmann's rbfSolve were worth utilizing since they gave the best improvement over the starting solution and also used the least number of function evaluations.

The result is promising. Our calculations show that the SIV can decrease the fuel consumption up to 14% compared to that of an engine without SIV and about 5% compared to that of a non-optimized engine with SIV. Our conclusion is that "black-box" optimization can be a valuable tool for improvements in the area of engine development.

The outcome of this thesis is of interest for engine developers, both for its emphasis on SIV and for its analysis of optimization in engine development. The thesis should also interest someone who wants to introduce the use of optimization algorithms in new areas of specializations, with similar black-box optimization problems.

Contents

Chapter 1

Introduction

1.1 Background

In todays industry there is a need for computer simulations. The simulations can tell a company about the benefits or the drawbacks of a certain design before it is produced. Since simulations are cheap and fast they can save a lot of money and time. And since the simulations are fast, it is possible to try a lot of different solutions to a certain problem or design. This is where simulation optimization can be useful. Simulation optimization is a method that tries to find the best solution possible to a simulated problem using different optimization algorithms.

The work behind this thesis has been a joint venture between the Mathematics Department of Chalmers University of Technology, Fraunhofer-Chalmers centre for Industrial Mathematics (FCC) and Volvo Car Corporation (VCC), Gothenburg, Sweden. VCC is a company in the Ford Motor Company family, which also includes the automotive brands Jaguar, Mazda, Land Rover and Aston Martin. Financially, VCC is totally separated from AB VOLVO but they still share the same trademark.

1.2 Engine basics

The engine considered in this thesis is a 5 cylinder Volvo RNC25 (Revised new compact 2.5 liter) naturally aspirated engine (the engine has no turbocharger). It has twin overhead camshafts and variable valve control (VVT) on the intake valves. For Swedish translations of some engineering vocabulary, see Appendix A.

The subject of interest in this work is the air flow in the engine. When the air has entered the intake and passed an air filter it arrives at a plenum. From the plenum the main pipes lead the air, via inlet valves, to the cylinders where combustion occurs. The exhaust gas then leaves the cylinder through the exhaust pipes via the outlet valves, except for some residuals (EGR) that stay in the cylinders. Several parameters govern the behavior of the air flow in the pipes and thus the performance of the engine. The parameters of interest for this work were initially:

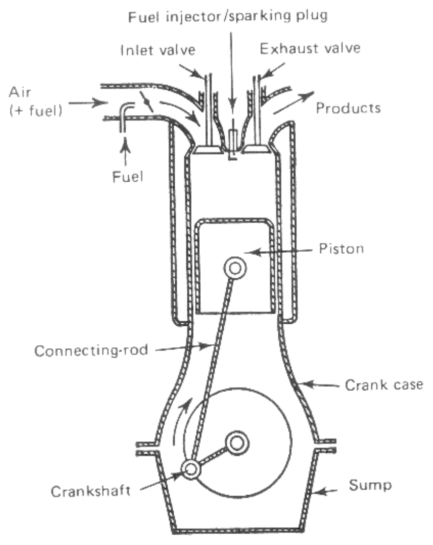


Figure 1.1: A four stroke engine

- The length of the main inlet pipes (hardware),
- the diameter of the main inlet pipes (hardware), and
- the valve timing (hardware and software).

We distinguish between *hardware variables*, that is, parameters that are independent of engine speed and *software variables* that can be adjusted depending on the speed of the engine. The valve timing is worth looking at more closely. In a four stroke engine the piston makes four strokes, that is, two strokes per crankshaft revolution. Referring to Figure 1.1, the strokes are:

1. *The induction (or, intake) stroke.* The inlet valve is open and the piston travels down the cylinder, drawing in an air/fuel mixture.

2. *The compression stroke.* Both valves are closed, and the piston travels up the cylinder. As the piston approaches its highest position, called *top dead center* (tdc), the mixture is ignited.

3. *The expansion (power or working) stroke.* Combustion propagates throughout the mixture, raising the pressure and the temperature, and forcing the piston down. At the end of the power stroke the exhaust valve opens.

4. *The exhaust stroke.* The exhaust valve remains open and as the piston travels up the cylinder the remaining gases are expelled. At the end of the exhaust stroke the exhaust valve closes and some residuals will be left to dilute the next mixture drawn in.

Important in valve timing is the *valve overlap*, that is, the short duration at top dead center after the exhaust stroke when both valves are open. This overlap can increase or decrease the performance of the engine, depending on engine speed. This is one of the reasons for using *Variable Valve Timing* (VVT). The VVT is a mechanism on the camshaft by which the valve timing for different speeds is controlled. It is also important how the crankshaft and the camshaft are related to each other, that is, at which crank angles the valve shall open and close. This relation defines the duration during which the valves are open. The duration can also be controlled by the VVT. On the RNC25, VVT is used on the intake camshaft only because the closing of the intake valve is the most crucial parameter in valve timing. If there is no VVT in the engine, the duration and cam top-angle are considered to be hardware variables. The cam top-angle is where the valves reaches their maximal lift and is measured in crank angles.

An unwelcome effect in gasoline engines is the occurrence of knocking. Knocking means that the gasoline that enters the cylinder ignites before the spark plug fires. This spontaneous ignition causes the engine loss of power. The effect is more severe, when the spontaneous ignition occurs very early in the compression stroke; then the engine can keep firing and firing even when the ignition is turned off. The main reasons why knocking occurs are that pressures and temperatures exceed the levels that the engine is designed for.

1.2.1 Serial intake valves

In this thesis we focus on an engine with *serial intake valves* (SIV). Such valves are placed on the intake pipes. They are supposed to increase the volumetric efficiency of the engine by catching standing waves to aid in the aspiration of the engine. The new valves will replace the throttle, as well as the intake VVT function. They are able to open and close twice during the induction stroke. The timing of the opening and closing of the valves are *software variables*, that is, they depend on the engine speed and load. We consider the dependence on the engine speed at full load only, since at part load we have to use a different approach to the problem. At part load the SIV:s settings are determined using regulation control in Simulink, see Section **??**. Because of the limited time allocated to the thesis and the fact that finding a solution to the full load problem is of greatest importance, we have concentrated on solving that problem.

There are many different SIV constructions; one such valve is shown in Figure 1.2. The switching of the valves are in this design managed by electro-magnets and a spring in a spring-mass system. The valves are able to open and close in around 2 ms. A rule-of-thumb is: The faster the valves can open and close, the better they can aid in the aspiration of the engine.

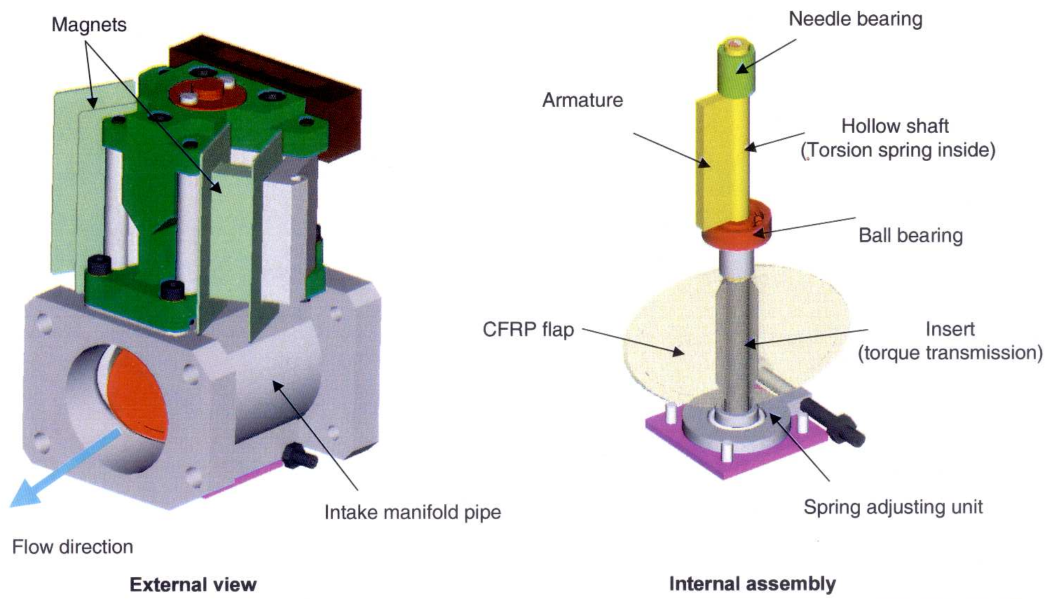


Figure 1.2: An illustration of a SIV

1.3 Introduction to optimization

To optimize refers in this thesis to finding the optimal solution to the problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x), && (1.1a) \\ \text{subject to} \quad & x \in \mathcal{S}, && (1.1b) \end{aligned}$$

where $f : \Re^n \mapsto \Re$ is called the *objective function*, $\mathcal{S} \subseteq \Re^n$ is the set of feasible points and $x \in \Re^n$ is the decision vector. If the problem is a maximization one, simply minimize $-f$. The set $\mathcal{S}$ is usually defined by a set of constraints as in Problem (1.2), below. For problems without constraints the set $\mathcal{S} = \Re^n$.

A constrained optimization problem can be formulated as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x), && (1.2a) \\ \text{subject to} \quad & g_i(x) \leq 0,\ i \in \mathcal{I}, && (1.2b) \\ & g_i(x) = 0,\ i \in \mathcal{E}, && (1.2c) \end{aligned}$$

where the functions $g_i : \Re^n \mapsto \Re, i \in \mathcal{I}(i \in \mathcal{E})$ define the inequality (equality) constraints. If f and $g_i, i \in \mathcal{E} \cup \mathcal{I}$ are linear functions, Problem (1.2) is called a *linear program*.

Now when the problem is defined, we have to specify an *optimal solution*. The basic definition of an optimal solution x^* to Problem (1.1) is that $x^* \in \mathcal{S}$ and

$$f(x^*) \leq f(x), \qquad \forall x \in \mathcal{S}. \tag{1.3}$$

The point x^* is then called a *global minimizer* of f over $\mathcal{S}$. Most of the methods we have tested in this work don't guarantee to find a global minimizer. They find at best a *local minimizer* that is, an x^0 such that $x^o \in \mathcal{S}$ and for a small enough value of $\epsilon > 0$

$$f(x^o) \leq f(x) \qquad \forall x \in \mathcal{S} \text{ such that } \|x - x^o\| \leq \epsilon. \tag{1.4}$$

Many functions have more than one local minimum. Therefore it is hard to know if a minimum found is a global minimum or just a local minimum. There are however cases for which every local minimum is a global minimum. One such case is when the objective is a *convex* function and the feasible region is a *convex* set. A set $\mathcal{S}$ is convex if every x and y both in $\mathcal{S}$ can be connected with a line which also lies in $\mathcal{S}$. Mathematically this is formulated as follows: A set $\mathcal{S}$ is convex if, for any elements x and y of $\mathcal{S}$,

$$\alpha x + (1-\alpha)y \in \mathcal{S}, \qquad \forall\, \alpha \in [0,1] \tag{1.5}$$

A function f is convex on a convex set $\mathcal{S}$ if it satisfies:

$$f(\alpha x + [1-\alpha]y) \leq \alpha f(x) + (1-\alpha) f(y), \qquad \forall\, \alpha \in [0,1], \forall x, y \in \mathcal{S} \tag{1.6}$$

For $n = 1$ the fefinition in (1.6) says that a line connecting $(x, f(x))$ and $(y, f(y))$, where $x, y \in \mathcal{S}$, must lie on or above the graph of the function in order to be convex; see Figure 1.3.

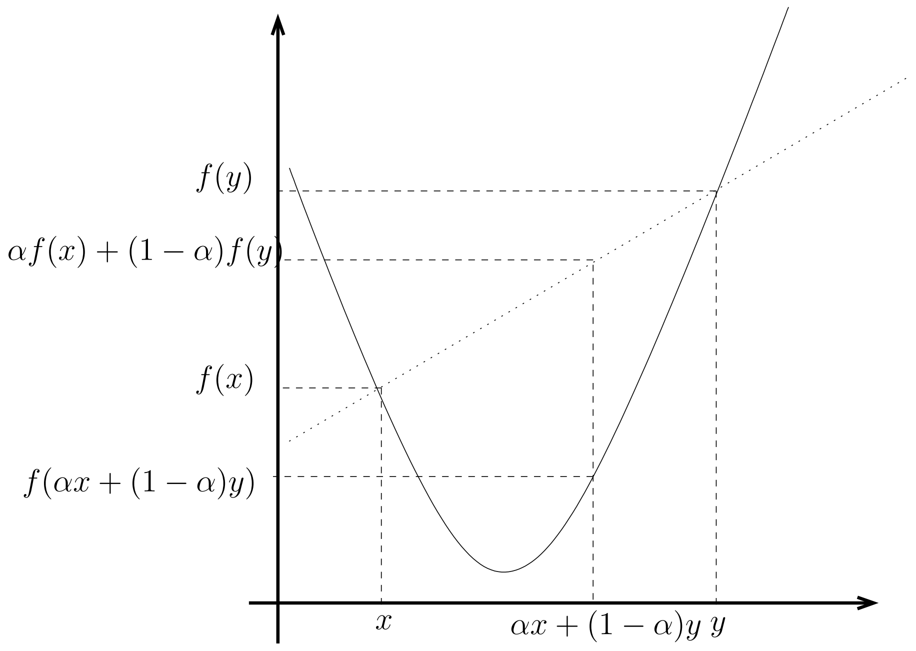


Figure 1.3: A convex function

If the function f is twice continuously differentiable, then an alternative definition of convexity can be given that is sometimes easier to check: If the Hessian matrix $\nabla^2 f(x)$ is positive semi-definite for all $x \in \mathcal{S}$ and $\mathcal{S}$ is a convex set, then the function f is convex on $\mathcal{S}$.

As mentioned above, if f is a convex function on a convex set $\mathcal{S}$, then the following holds: If x^* is a local minimizer of f on $\mathcal{S}$, then x^* is also a global minimizer. A proof for this result can be found in [11].

Sometimes we don't know whether a problem is convex, since the objective function is not known explicitly. Then we have to check if a given point is at least a local minimizer. To use *optimality conditions* are the mathematical way to check that a given point is a local minimizer to an optimization problem. We distinguish between *necessary* and *sufficient* optimality conditions. Necessary conditions has to be fulfilled but they don't guarantee that the given point is a minimizer. The sufficient optimality conditions are "sufficient" to guarantee that x^* is a local minimizer. The first-order necessary condition for an unconstrained optimization problem, Problem (1.1) with $\mathcal{S} = \Re^n$, is:

- *If x^* is a local minimizer of an unconstrained optimization problem and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.*

Optimization methods therefore seek points where the gradient of the objective function vanishes. There is also a second-order necessary condition.

- *If x^* is a local minimizer of Problem (1.1) with $\mathcal{S} = \Re^n$ and $\nabla^2 f$ is continuous in an open neighbourhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.*

The second-order sufficient condition is as follows:

- *The point x^* is a local minimizer to an unconstrained optimization problem, if $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite.*

There are optimality conditions for constrained optimization problems, Problem (1.2), as well. In this introduction to optimization we will just look very briefly into the subject of optimality conditions for constrained optimization. A more complete description on the subject can be found in [11], chapter 14. For unconstrained problems we have the optimality condition $\nabla f(x^*) = 0$, but as seen in Figure 1.4, x^* can be an infeasible point. A intuitive way to formulate optimality conditions for this constrained problem is to look at $-\nabla f(x^o)$ and a vector of opposite direction ∇g which is a vector sum of the normals to the linear constraints. When there is an equilibrium (compare with equilibrium of forces) between ∇g and $-\nabla f(x^o)$ we have an optimal point for the constrained problem. In Figure 1.4 the objective function is displayed with isobars, the constraints are straight lines and the feasible area is shaded.

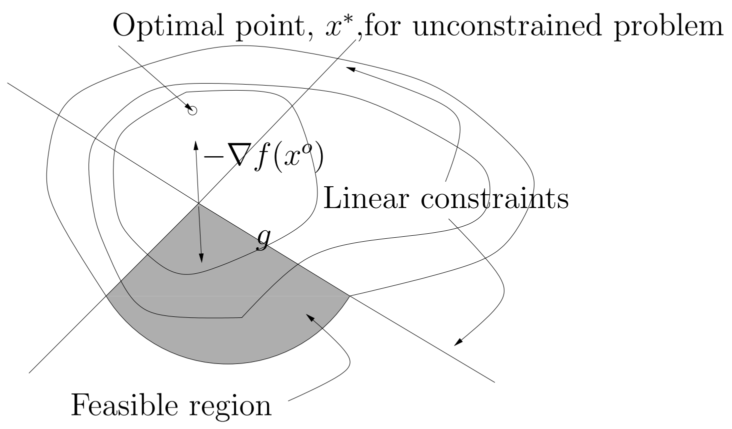


Figure 1.4: A Constrained Problem

1.3.1 Optimization in engineering applications

A common situation in engineering optimization is that the computation of an accurate objective value at a given point requires a call to a simulation program which is only available in a compiled form. Then the simulation program has to be considered as a *black box*. In such cases no derivatives are explicitly availabe. This doesn't necessarily mean that derivatives don't exist, but they must be approximated by numerical differences. An alternative is to build an algebraic model of the objective function which can then be explicitly differentiated. If the function evaluations are inexpensive and without noise, then it is possible to use numerical differences; otherwise it's preferable to build an algebraic model or to use a method based solely on objective values (such as pattern search methods see Section 2.3).

We distinguish between global and local optimization methods.

- Global optimization methods are designed to find the lowest objective value of f over $\mathcal{S}$. These methods may work well on small problems, but as the problems grow larger and more complex the solution time increases extremely rapidly.

- Local optimization methods are iterative processes that start in a point and then search for optima in a region around the currently best point. These methods usually converge faster to a local optimum than the global methods converge to a global optimum.

1.4 Problem description

The idea of SIV is quite old, but not until recently the technology to produce them have been available. VCC is interested in the possible benefits of SIV. However, more information about their performance is needed to motivate the company to start producing engines with SIV. Since the cost of building a first engine with SIV is very high, simulations are used for its design. These can give an indication of the benefits from SIV and motivate the testing with a real engine. To utilize these simulations as much as possible, optimization is used. We have a simulation tool, WAVE, that can simulate the performance of an engine with or without SIV, and which can be used to calculate our objective function, see Section 1.4.1.

The simulation tool WAVE is a black box; the values of the hardware and software variables are input and the output is an objective value. We don't have access to the derivatives which makes it hard to find fast and effective algorithms for optimization.

The fact that the variables governing the behavior of SIV can vary with engine speed makes this problem particularly hard to solve. As mentioned above, we call variables that vary with the speed *software variables*, since these variables will be controlled by the car's main computer. We need one set of software variables for each speed we want to consider. If there are four variables governing the behavior of the SIV and we want to optimize for 12 different speeds, there will be $12 * 4 = 48$ software variables. Together with 6 hardware variables, this makes 54 variables in total. Given the complexity of the objective calculations, this is a large problem, and the long simulation time forces us to limit the number of objective evaluations.

Since the original problem is very large we choose to partition the work into two projects. The first project is devoted to the testing of several different algorithms for solving a small version of the problem, that is, with only hardware variables. This should give a hint of which algorithms to continue with for the full-size problem. Except for giving good results, the algorithms should be fast and stable. The first problem was to maximize the torque at 6000 rpm without the SIV. This way we don't have to deal with the software variables, since we consider one speed only, resulting in totally 6 variables. Also, the simulation time for this problem is much shorter, since we only simulate for one speed.

We don't know much about the properties of the objective function, only what can be seen from the output of the black box. It is not likely that the objective function is convex which makes it difficult to achieve second order necessary or sufficient optimality conditions. We assume that the goal function is twice differentiable and continuous, because this is what our local algorithms, see Section 2, demand to converge to a local optimum.

After finding appropriate algorithms we proceed with solving the full-size problem, project two. This is defined as to modify the engine's hardware parameters and adjust the settings of the SIV in order to gain as much torque and get as low *exhaust gas residuals* (EGR) as possible. The EGR are hot gases which raise the temperature in the cylinder and may cause knocking, see Section 1.2. We assume that the objective function has the same properties as that of the small-size problem.

1.4.1 Mathematical model

The objective considered is a combination of two objectives which are linearly combined with given weights. The first objective is to maximize the torque at each discrete engine speed. With discrete we mean that we have separated the problem so that we optimize at one engine speed at a time. The speeds at which we optimize have to be chosen carefully:

A certain value of a variable can have one effect at low speeds, but a completely different effect at high speeds.

The second objective is to minimize the *exhaust gas residuals* (EGR). The weights, α and β associated with each objective, change the difference in importance between the two objectives. We can even introduce weights, w^s and γ^s, $s \in \mathcal{S}$, where $\mathcal{S}$ are the set of all engine speeds considered, in order to focus more on specific speeds where it is critical to get good results. Upper and lower bounds on the variables are set using current engineering knowledge. There are also constraints that are engine specific, for example, the cam top-angle on the intake side has to be larger than that on the exhaust side. Another constraint forbids the SIV to open a second time before it has closed the first time.

The variables are defined in Table 1.1. $x_1 - x_6$ are hardware variables and $y_1^s - y_4^s$ are software variables.

Table 1.1: Description of the variables to optimize

Variable	Description
x_1	Cam top-angle, intake
x_2	Cam top-angle, exhaust
x_3	Cam duration, intake
x_4	Cam duration, exhaust
x_5	Diameter, intake pipe leaving plenum volume (mm)
x_6	Length, intake pipes
y_1^s	SIV up, first time (crank angle)
y_2^s	SIV duration, first time (crank angles)
y_3^s	SIV up, second time (crank angle)
y_4^s	SIV duration, second time (crank angles)

Our parameters are defined in Table 1.2.

Table 1.2: Description of the parameters

Parameter	Description
$t(\mathbf{x}, \mathbf{y}^s)$	Torque [Nm] from simulation.
$g(\mathbf{x}, \mathbf{y}^s)$	EGR [%] from simulation.
w^s	A torque-weight depending on rpm [1].
γ^s	An EGR-weight depending on rpm [1]
α, β	Objective weights [$1/Nm$], [1/%].
$\mathbf{l}$, $\mathbf{u}$	Bounds on the variables

With this notation, the main problem we are trying to solve can be formulated as follows:

$$
\begin{aligned}
\text{minimize} \quad & f(\mathbf{x},\mathbf{y}) = -\alpha \cdot \sum_{s\in S} w^s \cdot t(\mathbf{x},\mathbf{y}^s) + \beta \cdot \sum_{s\in S} \gamma^s \cdot g(\mathbf{x},\mathbf{y}^s), && (1.7a)\\
\text{subject to} \quad & \mathbf{l}^x \le \mathbf{x} \le \mathbf{u}^x, && (1.7b)\\
& \mathbf{l}^y \le \mathbf{y}^s \le \mathbf{u}^y, \qquad \forall s \in \mathcal{S} && (1.7c)\\
& x_1 \ge x_2, && (1.7d)\\
& y_3^s \ge y_1^s + y_2^s, \qquad \forall s \in \mathcal{S} && (1.7e)
\end{aligned}
$$

1.5 Environment and Software

One of the challenges in this work has been to get different software to communicate. In this section the software used is described.

1.5.1 WAVE

WAVE, see [16], is a tool for simulating the flow in a given system. It solves the simplified Navier-Stokes equations, that is, the time dependent wave equation in one dimension. The input and output files are in ASCII-format but there is also a Wavebuild program in which the engine can be built graphically. A WAVE model of the RNC25 engine is presented in Figure 1.5. The pipes corresponding to the hardware variables, x^5 and x^6, are shown. Since all five intake pipes have the same length and diameter, we only get two intake pipe variables instead of $2 * 5 = 10$. The basic procedure is to run WAVE for a number of engine speed values, each speed value is called a case, run until the output for each case has converged and then plot the results in a graphic interface. The output of interest in this work are torque (Nm) vs. engine speed (rpm) and EGR (%) vs. speed (rpm). We use this output to calculate the objective function, $f(\mathbf{x},\mathbf{y})$. WAVE uses a FEM approach and the space discretization is the limiting factor when speeding up the computations.

1.5.2 Matlab and Simulink

Matlab and Simulink are both products from Mathworks Inc. Matlab is a tool for mathematical and technical computing. There exist various application-specific tool-boxes in Matlab. We use the *Optimization Toolbox* to implement *our method*, see Section 2.2. It is also possible to interface Fortran subroutines with Matlab via mex-functions; this means that an algorithm written in Fortran can be called as if it was a Matlab function. The opposite is also possible; from a Fortran program calculations can be run in Matlab, returning the result to the Fortran program. We have used these features and many more in our work. Matlab is also the program which controls the other programs. For example, Wave is started from Matlab using the "unix"-function which executes a UNIX command.

Simulink is a program for designing and simulating continuous and discrete-time systems. It is possible to execute WAVE simultaneous with Simulink. When using Simulink

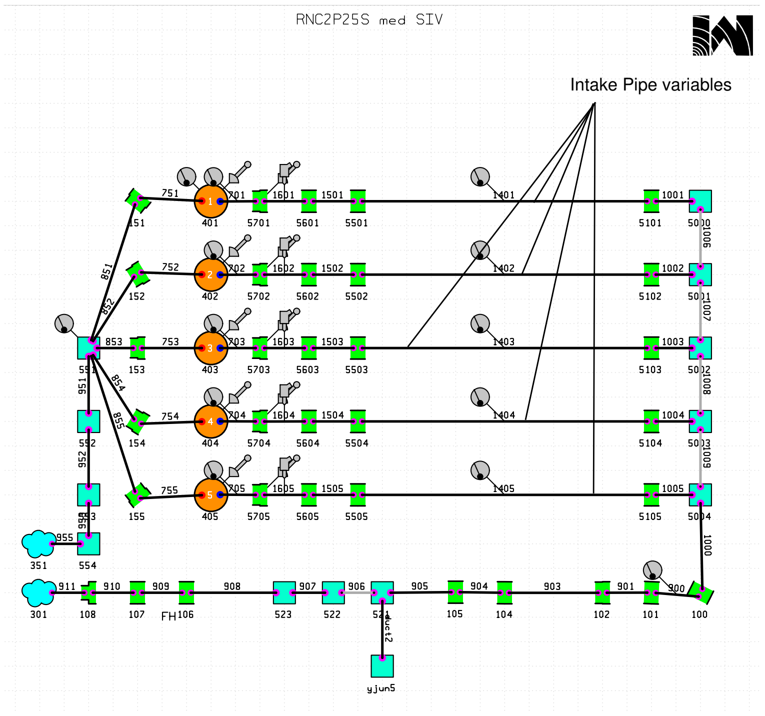


Figure 1.5: An illustration of the WAVE model of the engine RNC2P25C with SIV

the auto-convergence criteria in WAVE is deactivated, which means that the simulation of the engine can be run transiently.

1.5.3 Tomlab

Tomlab is an optimization package for applied optimization problems and works as a Matlab tool-box. It has been developed by Professor Kenneth Holmström at Mälardalens University. The base module includes several optimization algorithms and can also be extended by other tool-boxes like the CGO toolbox for constrained global optimization. We have mostly used the solver rbfSolve in the CGO module. The theory behind this solver is described in Section 2.4.

1.5.4 Previous work

Our supervisor Johan Lennblad at VCC has been working with this problem for a short while, trying to find a good solution changing the software variables "by hand". This solution serves as our initial guess in the local optimization algorithms, which all need a starting solution.

1.6 Hypothesis

The aim of the initial project is to test different algorithms on a small-size problem. Our hypothesis is that it is possible to draw conclusions from the work in this project that are useful in the main project. We also believe that optimization algorithms will find a solution which increases the power of the engine compared to that of the starting solution. A rule-of-thumb is that it is unlikely to find an improvement much greater than 5% in a system which has already been optimized using a large amount of experience and knowledge about the problem combined with simple heuristic methods.

The hypothesis of our work in the main project is that it is possible to improve the performance of an engine equipped with SIV by using derivative-free optimization algorithms, that is, algorithms that solve optimization problems without access to derivative information. A distinct definition of what is a good performance of an engine is difficult to formulate. Our objective can roughly be said to be a combination of the goals to gain "a higher torque at lower speeds", a "smooth curve of the torque with respect to speed", and the "same or higher torque at higher speeds", compared to the initial solution.

The reason for our choice of objective $f(\mathbf{x}, \mathbf{y})$ is that if the torque increases for low speeds, the fuel consumption will decrease provided that one puts in an extra gear like an overdrive. An unsmooth curve of the torque indicates that the car isn't nice to drive: If the torque increases fast when the engine speed increases, the driver will experience a jerk when he/she presses the gas pedal down. The reason why we want to have the same or higher torque at higher speeds is obvious: The performance of the engine may not worsen at any speed value, but it is unlikely that it will improve at high speeds since the SIV have a rather small impact on the performance of the engine in this interval.

Chapter 2

A summary of algorithms utilized

We have studied various derivative-free methods and also constructed one ourselves. As mentioned in the Introduction to optimization in Section 1.3 we distinguish between local and global optimization methods. Of the five algorithms tested, four is for local optimization and one is for global optimization. They are described in this chapter.

2.1 Trust-region methods

A basic trust-region algorithm can be intuitively understood since it is based on a straightforward way of thinking. We start with an initial point in $\Re^n$. Then we build an algebraic model of our objective function. The algebraic model is supposed to represent reality as well as possible and can be built by interpolation or Taylor expansion for example. We then apply an optimization method to find the models minimizer within a radius where we "trust" our model, the trust-region radius. If the model's objective value at this point agrees well with that of the real objective function, and is better than our initial guess, the trust-region center is moved to this point and a new model is built around it, perhaps updating the radius in some way. Trust-region methods are iterative algorithms, that is they need a sequence of iterates to converge to optimality. They are local solvers, that is, they only guarantee local optimality of the solution found. In the methods we study in this work we build a model by interpolation since we do not know anything about the objective function. When interpolating, it is crucial to avoid degeneracy, which can occur when defining a quadratic model and all, or some, of the interpolation points lie on a straight line. Then it is impossible to define a unique quadratic model and it will probably not approximate the real objective very well. Therefore it is important for trust-region methods to deal with these so called geometric conditions. There are various ways to deal with this as shown in the following sections. A basic trust-region method is described in Algorithm 1; see also [5].

Algorithm 1 A basic Trust-Region method

Step 1: Initialization. An initial point $x_0 \in \Re^n$ and an initial trust-region radius $\Delta_0 > 0$ are given. The constants ν_1, ν_2, γ_1 and γ_2 are also given, and satisfy

$$0 < \nu_1 \leq \nu_2 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1. \tag{2.1}$$

Compute $f(x_0)$ and set $k = 0$.

Step 2: Model definition. Choose the norm $\|\cdot\|_k$ and define the model m_k in the trust-region $B_k = \{x \in \Re^n \mid \|x - x_k\|_k \leq \Delta_k\}$.

Step 3: Step calculation. Compute a step s_k that "sufficiently reduces the model" Q_k and is such that $x_k + s_k \in B_k$.

Step 4: Acceptance of the trial point. Compute $f(x_k + s_k)$ and compute

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{Q(x_k) - Q(x_k + s_k)}. \tag{2.2}$$

If $\rho_k \geq \nu_1$, then let $x_{k+1} = x_k + s_k$. Otherwise, let $x_{k+1} = x_k$.

Step 5: Trust-region radius update. Choose

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if} \quad \rho_k \geq \nu_2, \\ [\gamma_2\Delta_k, \Delta_k] & \text{if} \quad \rho_k \in [\nu_1, \nu_2) \\ [\gamma_1\Delta_k, \gamma_2\Delta_k] & \text{if} \quad \rho_k < \nu_1. \end{cases} \tag{2.3}$$

Increase k by 1 and go to Step 1.

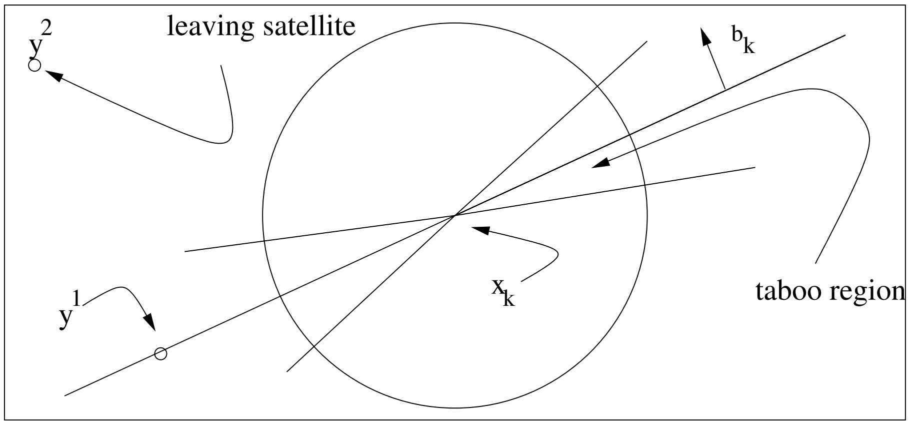


Figure 2.1: Example for $n = 2$. The circle represents the trust-region.

2.1.1 Wedge

Algorithm 1 can be modified in a huge number of ways with varying characteristics. Since we are interested in derivative-free optimization, preferably on quadratic models, we have studied the algorithm *Wedge* by Marazzi and Nocedal [9]. We are interested in quadratic models because they can model dependence between two variables in a way linear models can not. The Wedge algorithm tries to avoid degeneracy when building and updating the model by applying "wedge"-constraints, thus defining a taboo region, W_k, where we may not search for new iterates. These wedge-constraints and the taboo-region is illustrated in Figure 2.1. *Wedge* is an algorithm for unconstrained optimization, that is, it doesn't handle constraints.

In order to uniquely define a quadratic model at iteration k of the form

$$Q_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T G_k s, \qquad s \in \Re^n \tag{2.4}$$

that interpolates between a set of points we need to determine the coefficients $g_k \in \Re^n$, and the symmetric $n \times n$ matrix G_k, a total of $m = \frac{1}{2}(n+1)(n+2) - 1$ scalar unknowns. The number of variables are denoted n. We thus need m so called interpolation points,

$$y^l = x_k + s^l, \qquad\qquad l = 1, ..., m.$$

When the interpolation is done we compute a trial step s_k by solving

$$\begin{aligned} \underset{s \in \Re^n}{\text{minimize}} \quad & Q_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T G_k s, && \text{(2.5a)} \\ \text{subject to} \quad & \|s\| \leq \Delta_k, && \text{(2.5b)} \\ & s \notin W_k && \text{(2.5c)} \end{aligned}$$

A formal description of the algorithm is given in Algorithm 2. Here $y^1, ..., y^m$ is the interpolation points that are used to define the model. The constants α and β are used when updating the trust-region. The norm is euclidean, ℓ_2-norm.

In step 6 we keep a point if it is closer to the current iterate than is $y^{l_{out}}$, the interpolation point farthest away from the currently best point x_k. This helps improve the model in the neighborhood of x_k. Thus, if a sequence of unsuccessful trial steps is generated and Δ_k decreases sufficiently, the trial point will eventually be admitted as an interpolation point. As a result, the interpolation model Q_k will become increasingly accurate, so that eventually a successful step will be computed. The authors of [9] have so far only been able to prove convergence of linear models, they are currently trying to prove convergence of quadratic models, that is, Equation (2.4).

In interpolation-based trust-region methods there are two types of iterations. The first is the *minimization iteration*, in which one tries to reduce the value of the algebraic model Q, and the other a so-called *simplex iteration* designed to define a model that approximates f better. Algorithm 2 only performs one type of iteration, the minimizing one. To avoid problems with bad geometric properties of the model, a geometric constraint is imposed in the minimization problem. It basically forbids the algorithm to search in a taboo region W_k. The taboo region is defined by a maximum angle γ between the vectors $s^l = y^l - x_k$, between x_k and the satellites y^l, and the normal to s^l, b_k; see Figure 2.1 and [9] for details.

The step computation uses a procedure called *Quadstep*, which first finds the solution s_{TR} of the trust-region subproblem (2.5), ignoring the wedge constraint. This method is described in [10]. If the wedge constraint is violated Quadstep computes a sequence of trial steps that move away from the taboo region in a direction in which the violation of the wedge constraint initially decreases. This is done until the constraint is satisfied or until the objective value has increased too much with respect to the starting value at s_{TR}; see [9] for details.

We found an implementation of this algorithm in Matlab code on the internet, WEDGE; see [18]. When we tested it, we got better results if we altered the radius update criteria: We do not shrink the trust-region radius quite as much if the test point, $x_k + s$, is closer to the current iterate than is the farthest satellite even though it generates a worse objective value. This is because we don't want to shrink the trust-region too fast. The new criteria slow down the convergence-rate because we have relaxed the "shrink-criterion". The model will become more and more accurate when we add test points to our satellite set, thus increasing the probability of finding an accurate test point.

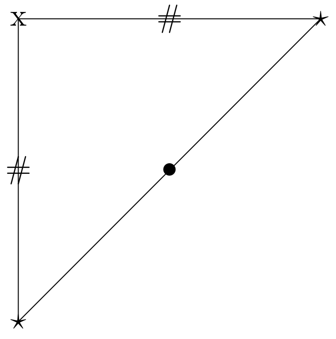
The initial sample points are, in this implementation, the vertices and midpoints on the edges of a simplex in $\Re^n$, see Figure 2.2. The starting point is x. The points marked $\star$, are $x \pm \Delta e_i$, $i = 1, ..., n$. The sign is randomly determined by the algorithm. The point marked $\bullet$ is the midpoint of the edge between two different $x \pm \Delta e_i$. The points marked # are the midpoints of the edges between x and each $x \pm \Delta e_i$. There is a degree of randomness in the initialization of the sample points and also in the output. It is therefore recommended that each problem is solved five times, to increase the chance to get an initial model that fits the real objective well. The randomness is a necessity and disadvantage with this algorithm.

Wedge is developed for unconstrained optimization, that is, it can't handle constraints.

Algorithm 2 Wedge

```
Step 1: Initialization. Choose the trust-region parameters $\alpha, \beta \in (0,1)$ and an initial
        point $x_0$. Select an initial set of satellites $\Sigma_0 = \{y^1, y^2, ..., y^m\}$ such that $x_0 \cup \Sigma_0$
        is non-degenerate. Set $k = 0$.
Step 2: Find the satellite that is farthest from the current iterate:
            $y^{l_{out}} = \operatorname*{argmax}_{y \in \Sigma_k} \|y - x_k\|$
        .
Step 3: Model update. Define a model $Q_k(x_k + s)$ that interpolates $\{x_k\} \cup \Sigma_k$, and
        define the wedge $W_k$.
Step 4: Step calculation. Compute $s_k$ by solving Subproblem (2.5) approximately, and
        evaluate $f(x_k + s_k)$.
Step 5: Set $ared(s_k) \equiv f(x_k) - f(x_k + s_k)$ and $pred(s_k) \equiv Q_k(x_k) - Q_k(x_k + s_k)$.
Step 6: Trust-region update. If $\text{ared}(s_k) > \alpha \cdot \text{pred}(s_k)$, choose $\Delta_{k+1}$ such that $\Delta_{k+1} \geq$
        $\Delta_k$, else set $\Delta_{k+1} = \beta \cdot \Delta_k$.
Step 7: If $f(x_k + s_k) \leq f(x_k)$ (successful iteration)
        Update the current iterate and include $x_k$ in the satellite set, discarding $y^{l_{out}}$:
        a. $x_{k+1} = x_k + s_k$
        b. $\Sigma_{k+1} = \{x_k\} \cup \Sigma_k \setminus \{y^{l_{out}}\}$.
        Else (unsuccessful iteration)
        If the new trial point is closer, in Euclidean distance, to the current iterate than
        $y^{l_{out}}$, admit it to the satellite set, discarding $y^{l_{out}}$; otherwise, discard the new trial
        point:
        c. $x_{k+1} = x_k$
        d.
            $\Sigma_{k+1} = \begin{cases} \{x_k + s_k\} \cup \Sigma_k \cup \setminus \{y^{l_{out}}\} & \text{if} \quad \|y^{l_{out}} - x_k\| \geq \|(x_k + s_k) - x_k\| \\ \Sigma_k & \text{otherwise.} \end{cases}$
Step 8: Set $k = k + 1$. Goto step 2.
```

Figure 2.2: Simplex in $\Re^2$



2.1.2 UOBYQA and NEWUOA

In [13] Powell presents an algorithm that performs both types of iterations (minimization and simplex iterations) mentioned in Section 2.1.1, in contrast to Wedge that only performs the minimization step. Powell uses Lagrangian functions to guarantee the correctness of the interpolation; see below. Also, he does not allow the trust-region radius to increase, the motivation is that more decreases are needed later to achieve convergence if the radius is allowed to increase. Recently, Powell has improved his algorithm *UOBYQA* (Unconstrained Optimiztion BY Quadratic Approximation) into *NEWUOA* (NEW Unconstrained Optimization Algorithm) so that only $m = 2n + 1$ interpolation points are needed instead of $m = \frac{1}{2}(n+1)(n+2)$; see [14]. In both algorithms the user must supply the initial point x_b and the initial and final values of the trust-region radius, ρ_{beg} and ρ_{end}. The user also has to provide a routine that calculates values of the objective function $f(x)$. Both methods are, as the names reveal, developed for unconstrained optimization. Like Wedge they can't handle constraints. The quadratic model is of the same form as that in Wedge; see Equation (2.4).

The interpolation constraints are,

$$Q(x_i) = f(x_i), \qquad i = 1, 2, ..., m, \tag{2.6}$$

where m is the number of interpolation points. Also required are the Lagrange functions of the interpolation Problem (2.6). For $j = 1, 2, ..., m$ The j-th Lagrange function being the quadratic polynomial from $\Re^n$ to $\Re$

$$\ell_j(x_i) = \delta_{ij}, \qquad i = 1, 2, ..., m, \tag{2.7}$$

where δ_{ij} is the Kronecker delta,

$$\delta_{ij} = \begin{cases} 1, \text{ if } i = j, \\ 0, \text{ otherwise} \end{cases}$$

The Lagrange polynomial can also be formulated, compare with Equation (2.4),

$$\ell_j(x) = c_j + g_j^T(x - x_b) + \frac{1}{2}(x - x_b)^T G_j (x - x_b), \qquad x \in \Re^n. \tag{2.8}$$

(2.6) and (2.7) give us the identity

$$Q(x) = \sum_{j=1}^{m} f(x_j)\ell_j(x), \qquad x \in \Re^n. \tag{2.9}$$

Using the quadratic model, (2.9), UOBYQA solves a trust-region sub-problem,

$$\begin{array}{rl} \text{minimize} & Q(x_k + d) \\ \text{subject to} & \|d\| \leq \Delta, \end{array} \tag{2.10}$$

where k is the index corresponding to the minimal value of f over all the previous iterates. The norm is Euclidean and Δ is a trust-region radius that satisfies $\Delta \geq \rho$. We see that Powell has two trust-region radii, ρ and Δ. Δ is used in Problem (2.10) whereas ρ is used in Problem (2.11). ρ is not allowed to increase because this would slow down the convergence rate.

The other trust-region sub-problem is the one where a suitable new interpolation point is moved by a model step,

$$\begin{array}{rl} \text{maximize} & |\ell_j(x_k + d)| \\ \text{subject to} & \|d\| \leq \rho. \end{array} \tag{2.11}$$

The reason for this calculation is that the new value $x_k + d$ is a suitable value for the interpolation point x_j. This choice of interpolation point maintains the non-degeneracy of the interpolation problem, (2.6). Readers interested in the solving of the two subproblems can read about it in [13].

The theoretically hard problem is how to combine these two subproblems so that the algorithm converges to an optimal point. For convergence reasons a constant M is included in the algorithm. M is an upper bound on the third derivatives of the objective function (if they exist). If they exist, the difference between the quadratic model and the objective function satisfies the condition,

$$|Q(x) - f(x)| \leq \frac{1}{6} M \sum_{i=1}^{m} |\ell_i(x)| \|x - x_i\|^3, \qquad x \in \Re^n. \tag{2.12}$$

This inequality speeds up the rate of convergence for general objective functions because it constrains the problem and makes the feasible set smaller. The updating rules for Δ depend on how well the quadratic model approximates the real objective function, much like Wedge. Here we define r as,

$$r = [f(x_k) - f(x_k + d)]/[Q(x_k) - Q(x_k + d)]. \tag{2.13}$$

The value of r decides how to update Δ:

$$\Delta = \begin{cases} \max[\Delta, \frac{5}{4}\|d\|, \rho + \|d\|], & r \geq 0.7, \\ \max[\frac{1}{2}\Delta, \|d\|], & 0.1 < r < 0.7, \\ \frac{1}{2}\|d\|, & r \leq 0.1. \end{cases} \tag{2.14}$$

If the new value $f(x_k + d)$ is lower than $f(x_k)$, $x_k + d$ becomes an interpolation point. Let t be the index of the new interpolation point. A new point $x_k + d$ can also be included as an interpolation point if it satisfies

$$\ell(\tilde{x}_t) \neq 0, \tag{2.15}$$

where $\tilde{x}_t$ is the new position of the interpolation point. Otherwise the nonzero quadratic polynomial ℓ_t would vanish on the new set of interpolation points so that the system of Equations (2.6) would be singular.

If we combine Equations (2.15) and (2.12) we get a measure on which interpolation point is the best one to move. This is done by letting t be the value of i which gives the maximal value of the product

$$|\ell_i(x_k + d)| \|x_k + d - x_i\|^3, i = 1, 2, ..., m. \tag{2.16}$$

The second term promotes the replacement of a point that is far from the current best vector. This value of t is calculated whenever $f(x_k + d)$ is lower than $f(x_k)$ or when one of the terms in 2.16 exceeds one. Otherwise $t = 0$ is preferred.

When the quadratic model seems to be inadequate a model step is carried out. It picks a point from the set

$$\mathcal{J} = \{i : \|x_i - x_k\| > 2\rho\}, \tag{2.17}$$

usually it is the point that has the property

$$\|x_j - x_k\| = \max\{\|x_i - x_k\| : i \in \mathcal{J}\}, \tag{2.18}$$

according to Powell. Then Problem (2.11) is solved to get the move d.

If the situation arises that the quadratic model is satisfactory but we get a $\|d\| \leq \rho$ when solving (2.10),then the algorithm reduces ρ from ρ_{old} to ρ_{new} according to

$$\rho_{new} = \begin{cases} \rho_{end}, & \rho_{end} < \rho_{old} \leq 16\rho_{end} \\ \sqrt{\rho_{old}\rho_{end}}, & 16\rho_{end} < \rho_{old} \leq 250\rho_{end} \\ 0.1\rho_{old}, & \rho_{old} > 250\rho_{end} \end{cases} \tag{2.19}$$

The algorithm is summarized in Algorithm 3. The convergence to a stationary point is taken care of by the properties of ρ. When the model is unable to improve the solution for a certain ρ we shrink the trust-region radius to say, one tenth. When ρ_{end} is reached we have the desired precision of the solution.

Turning to NEWUOA, it is basically the same algorithm as UOBYQA. In UOBYQA the number of interpolation points for the quadratic model is $m = \frac{1}{2}(n+1)(n+2)$ which increases as n^2 with the number of variables. The difference in NEWUOA; see [14], lies in the use of the Frobenius norm in measuring the change in $\nabla^2 Q$. This reduces the number of points to $m = 2n + 1$ which only increases as n. The Frobenius norm of a matrix is the square root of the sum of squares of its elements. In UOBYQA the model Q is uniquely defined by the interpolation conditions. In NEWUOA the remaining freedom in Q is eliminated by minimizing the Frobenius norm $\|\nabla^2 Q_{k+1} - \nabla^2 Q_k\|_F$, while satisfying the linear constraints

$$Q_{k+1}(x_i) = f(x_i), \qquad i = 1, 2, ..., m. \tag{2.20}$$

This procedure is implemented when updating the quadratic model. It follows that the updating technique provides the new model

$$Q_{k+1}(x) = Q_k(x) + \{f(x_t^+) - Q_k(x_t^+)\}\ell_t^+(x), \qquad x \in \Re^n, \tag{2.21}$$

where ℓ_t^+ is chosen so that $\|\nabla^2 \ell_t^+\|_F$ is minimal subject to the Lagrange conditions

$$\ell_t^+(x_i^+) = \delta_{it}, \qquad i = 1, 2, ..., m. \tag{2.22}$$

x_i^+ are the new set of interpolation points after a model step and $x_t^+ = x_k + d_k$. The number $m = 2n+1$ is chosen because it forces the second derivative matrix of every quadratic model to be diagonal, which reduces the work of solving Equation (2.10). At the time this report was written, there was no formal description of the NEWUOA algorithm available since it was released in January 2004. We found an implementation of this algorithm on the internet; see [12]. The algorithm is written in Fortran77.

2.2 Our method

Our method is based on, but not identical to the basic trust-region algorithm, that is, Algorithm 1. The difference is that we don't test whether our trial point has a better objective value than the previous points. We neither increase nor decrease the trust-region. The search for a better objective value starts at a point, x_0, which is believed to be rather good. The next step is to span a ball with the radius Δ in scaled variables and having x_0 as the center. The ball has the dimension n, that is, as many dimensions as variables. Then the idea is to build a quadratic model, Q, of the objective function. The construction of a model is made by calculating the objective function inside as well as on the bounds of the ball. The objective function is calculated four times on each coordinate axis; at $x_0 \pm e_j$ and at $x_0 \pm 0.35 \cdot e_j$ where e_j is the j-th coordinate vector in $\Re^n$. The objective function is also calculated at $x_0 + \frac{1}{\sqrt{2}}(\pm e_j \pm e_i)$ for each pair (i, j) of variables. See Figure 2.3 for $n = 2$.

The objective value is hence calculated in $4n + 4n(n-1)/2$ points, where n is the number of variables. Having these objective values calculated, the data is fitted to a quadratic function Q, which becomes the model of the objective function. The next step

Algorithm 3 UOBYQA

Step 1: **Initialization.** The user must specify the starting point x_b, the original and ending trust-region radii ρ_{beg} and ρ_{end}. The algorithm then sets $\frac{1}{6}M = 0$, because this is the least upper bound on the third derivatives, $\rho = \rho_{beg}, \Delta = \rho_{beg}$, $j = 0$ and k becomes the least integer in {1,...,m} that has the property

$$f(x_k) = \min\ \{f(x_i) : i = 1, 2, ..., m\}. \tag{2.23}$$

Step 2: If $j = 0$, the Problem (2.10) is solved, which provides the trial step d. Further, the number $\texttt{DNORM} = \|d\|$ is noted and if $\texttt{DNORM} < \frac{1}{2}\rho$ goto step 7.

Step 3: If $j > 0$ the value $f(x_k + d)$ is calculated. The numbers $Q(x_k + d)$ and $\ell_i(x_k + d), i = 1, 2, ..., m$ are computed too. The value $\texttt{FOLD} = f(x_k)$ is noted. Then the parameter $\frac{1}{6}M$ is overwritten by,

$$\max\{\frac{1}{6}M, |Q(x_k + d) - f(x_k + d)| / \sum_{i=1}^{m} |\ell_i(x)| \|x - x_i\|^3\} \tag{2.24}$$

to assure that it is large enough.

Step 4: If $j = 0$, then Δ is updated according to Equation (2.14). Further, t is selected as suggested in the paragraph with Equation (2.16). Alternatively, if $j > 0$, then t is set to j.

Step 5: If $t > 0$, then the interpolation point x_t is moved to the position $\tilde{x}_t = x_k + d$. Update the quadratic model and the coefficients of the Lagrange functions so that they satisfy Equation (2.9). Moreover the value of k is changed to t if $f(\tilde{x}_t) < \texttt{FOLD}$. Let DMOVE be the Euclidean distance between the old position of x_t and x_k for the new value of k.

Step 6: If $t > 0$ and if at least one of the four conditions

$$j > 0, \quad f(\tilde{x}_t) < \texttt{FOLD}, \quad \texttt{DNORM} > 2\rho, \quad \texttt{DMOVE} > 2\rho \tag{2.25}$$

holds, then j is set to 0 and we go to step 2 in order to begin a trust-region step.

Step 7: Search for a positive integer j, according to the paragraph including Equation (2.17). If one is found a model step d is calculated, otherwise j is set to 0.

Step 8: If j is positive or both $j = 0$ and $\texttt{DNORM} > \rho$ occur, go to step 2.

Step 9: If $\rho > \rho_{end}$, then decrease ρ according to (2.19) and set x_k to $x_k + d$ before branching to step 2 for the next iteration.

Step 10: Return the current point x_k and terminate.

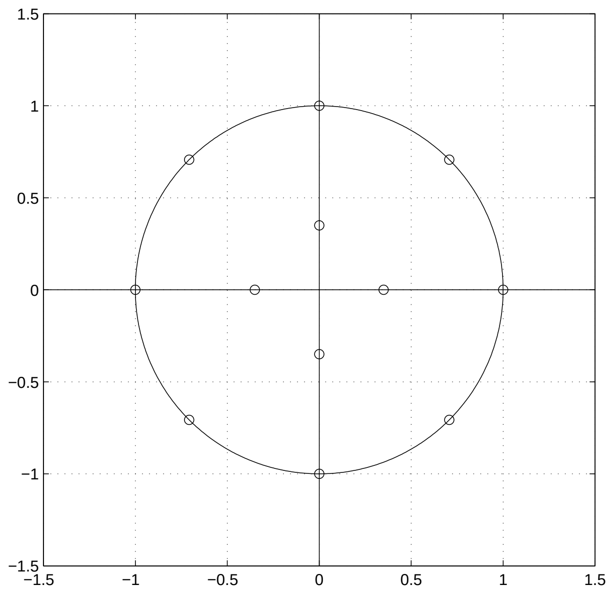


Figure 2.3: Points on and in the unit circle where the objective function is calculated

is to find the minimum of the model inside a cube with side length 2Δ, centered around the origin. In other words we search for x_{k+1} both inside and outside the ball (If we hadn't done so, the bounds wouldn't been linear, since a ball is impossible to describe with linear constraints). The last step is to set x_{k+1} as the origin and then span a new ball with the same radius and start all over. The number of iterations is decided by the user in advance.

Algorithm 4 Our Method

Step 1: **Initialization.** The initial point x_0 and an initial trust-region radius Δ are given. Set $k = 0$.

Step 2: **Model definition.** Define the set $\mathcal{U}$ of finite points, as $\mathcal{U} = \{x_k,\ x_k \pm e_j\, j = 1 \ldots n,\ x_k \pm 0.35 \cdot e_j\, j = 1 \ldots n,\ x_k \pm \frac{1}{\sqrt{2}} \cdot e_j \pm \frac{1}{\sqrt{2}} \cdot e_i\, j = 1 \ldots n\, i = 1 \ldots n\}$. Evaluate the function value $f(x)$, $\forall x \in \mathcal{U}$. Define the quadratic model, Q.

Step 3: **Step calculation.** Compute a step s_k that reduces the model such that $\|x_k + s_k\|_\infty \leq \Delta$.

Step 4: **Move the origin.** Set $x_{k+1} = x_k + s_k$. Increase k by 1 and go to Step 2.

At first sight one could find our method unorthodox, especially for the reader who is familiar with the basic trust-region algorithm.The fact that we never compare the result of the optimization of the algebraic model $Q(x_s)$ with the real function value $f(x_s)$ might cause problem. We believe that our model is good and gives a proper value of the real function value. We draw that conclusion from the premise that $4n+4n(n-1)/2$ interpolation points are used when the number of interpolation points necessary are only $(n+1)(n+2)/2$. All of our interpolation points x lie inside a ball with radius Δ, that is $\|x\|_2 \leq \Delta, x \in \mathcal{U}$, but we look for new points in a cube with side length 2Δ, $\|x_{k+1}\|_\infty \leq \Delta$. This isn't the common procedure for trust-region methods, but once again we believe that the numerous interpolation points will produce a good model so that we can trust the result even if some point lie outside the ball but inside the cube.

We choose Δ to be 1. The software used were WAVE and Matlab. The decision of where to evaluate the objective function is made in Matlab. In WAVE the objective function values are calculated. The function *lsqcurvefit* in the Matlab Optimization Toolbox is used to fit data to the algebraic model by solving the non-linear curve-fitting problem in the least-squares sense [8]. Another function in the Matlab Optimization Toolbox is *quadprog* [15], which is used to find the optimum of the objective function.

In our method convergence is not guaranteed, since no comparison is made between the function values of different iterates; it is not even guaranteed that the solution found is as good as the starting point. A simple comparison between $f(x_{k+1})$ and $f(x_k)$ could fix the problem that we might move to a point that is worse than the origin.

2.3 Generalized Pattern Search (GPS)

Pattern Search methods have been around for a while. Hooke and Jeeves formulated the original pattern search method in 1961, but it was not until 1997 that V. Torczon constructed the general pattern search method and proved its convergence [17]. Since the description of the algorithm in the article of C. Audet, J.E. Dennis Jr. [2] is equivalent to Torczon's version, but easier to understand, we choose to use their formulation and Torczon's convergence results.

Pattern search algorithms generate a sequence of iterates x_k, $k = 1, \ldots,$. At each iteration, the objective function is evaluated at a finite number of points on a mesh to try to find an objective value that is lower than the lowest found so far. A search strategy may be used to select mesh points in which the objective function will be evaluated. If the search strategy fails to find a point where the objective value is lower than the best solution found so far, the objective will be evaluated at all the neighbors of x_k in the mesh until a better solution is found. This is called the *poll* step. If the objective function is evaluated at all the neighbors of x_k in the mesh and no better solution is found, the mesh will be refined and the search for a better solution will start over in the refined mesh. If a better solution is found the parameter defining the size of the mesh in each direction is kept the same or increased.

Pattern search algorithms are defined through a finite set of matrices $\mathcal{S}$, each of whose columns are a positive set in $\Re^n$. However, for technical convergence reasons, [17], every column s of each matrix must be generated from a single matrix $G^B \in \Re^{(n\times n)}$ and from a finite set of integer generating matrices $G_\ell \in \mathcal{Z}^{(n\times n)}$ for $\ell = 1, 2, \ldots, \ell_{max}$ as follows: $s = G^B G_\ell z$ for some $z \in \mathcal{Z}^n$. The current mesh M_k is defined as $M_k = \{x_k + \Delta_k S_z : z \in$

$Z^{n_s}, S_z \in \mathcal{S}\}$, where $\mathcal{S}$ is a finite set of matrices S such that S span $\Re^n$, $\Delta_k \geq 0$ is the mesh size parameter, and n_s is the number of columns in the matrix S_z. During the poll step the objective is evaluated at the mesh points neighboring the current point. They are defined as $\{x_k + \Delta_k s : s$ is a column in $S_k\}$ for some positive spanning matrix $S_k \in \mathcal{S}$.

Algorithm 5 GPS

```
Step 1: Initialization. An initial point $x_0$ and an initial mesh $M_0$ defined by $\Delta_0 > 0$ are
        given such that $x_0 \in M_0$. Let $X_0 \subset M_0$ contain $x_0$. Set $k = 0$.
Step 2: Search. Use some search strategy to find $x_{k+1} \in X_k$ such that $f(x_{k+1}) \leq f(x_k)$.
        If such a vector $x_{k+1}$ is found, declare the Search successful and let $\Delta_{k+1} \geq \Delta_k$,
        increase $k$ by 1 and go to Step 2. Otherwise go to step 3.
Step 3: Poll. If Search is unsuccessful and $x_k$ minimizes $f(x)$ over $x \in \{x_k + \Delta_k s : s$
        is a column in $S_k\}$ then declare Poll unsuccessful, set $x_{k+1} = x_k$, and refine the
        mesh size: $\Delta_{k+1} \leq \Delta_k$. Else declare Poll successful, set $x_{k+1}$ to a point in $M_k$
        such that $f(x_{k+1}) \leq f(x_k)$ and set $\Delta_{k+1} \geq \Delta_k$. Increase $k$ by 1 and go to Step 2.
```

We used the software package NOMADm, Nonlinear Optimization for Mixed variables And Derivatives-Matlab, which is a Matlab code that runs various GPS algorithms to solve nonlinear and unconstrained optimization problems [1]. The algorithm chosen don't have any **Search** step, which means that the search strategy is to have *no* points as candidates for the next iteration. The poll directions are the Euclidean coordinate directions and their negatives and the objective function for the points in the poll set are calculated in each of these directions; that is $e_1, -e_1, e_2, -e_2, \ldots$. If an iteration **Poll** is successful in finding an improved mesh point, then the mesh is enlarged by double the mesh size parameter Δ_k, else the mesh size parameter is halved. The initial mesh size parameter is a tenth of the initial point, that is $\Delta_0 = 0.1x_0$. The algorithm terminates when the first of the following criteria is met;

- the mesh size parameter falls below a certain level,
- the CPU time is longer than a constant specified by the user,
- the number of iterations, or the function evaluations exceed a fixed number.

The algorithm guarantees that if $f(x)$ is continuosly differentiable where $f(x) < f(x_0)$, then for the sequence of iterates $\{x_k\}$ produced by the generalized pattern search method,

$$\lim_{k \to +\infty} \|\nabla f(x_k)\| = 0 \tag{2.26}$$

In other words, after infinitely many function evaluations the algorithm will converge to a point which fulfills the first-order necessary optimality condition for unconstrained optimization problems.

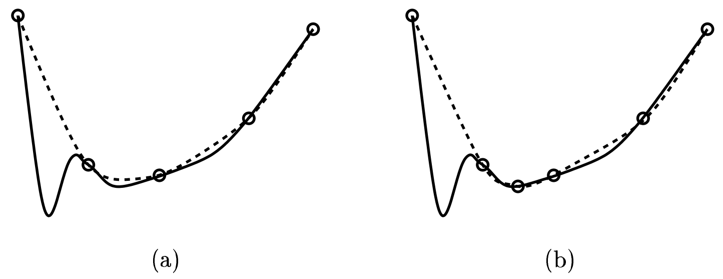


Figure 2.4: The algorithm misses the global minimum if the global minimizer of the response surface is the next local observation point.

2.4 rbfSolve

Gutmann has formulated a method [7, 6] whose target is to find a global optimum for continuous not necessarily convex functions on a compact subset of $\Re^n$. The solver *rbfSolve* [3] in the Tomlab package CGO is an implementation of Gutmann's algorithm. The algorithm uses radial basis functions for the interpolation, when building an algebraic model of the objective function. This algorithm is efficient for problems for which the function evaluations are costly, because the algorithm tries to estimate the global optimum when searching for it.

The solver rbfSolve builds a model of the objective function within the feasible area, $\mathcal{S}$. The model is also called a *response surface*. The model is built by interpolating a number of function values using both radial basis functions and polynomials. The next step is to determine where to evaluate the objective function next. The naive and straightforward idea would be to apply some optimization method to find the global minimizer of the algebraic model and then evaluate the objective function at this point. If this strategy is used, a global optimum may easily be missed. Figure 2.4 shows the objective function as a solid curve and the response surface as a dashed curve. The circles indicate the points where the objective function is calculated. If one choose the global minimizer of the response surface as the next point to evaluate the objective function, one misses the global minimum to the left in the figure. A better strategy is to base the decision where to evaluate the objective function next on the response surface's "bumpiness". The trick is to estimate the global optimum value and for every point $y \notin \{x_1, \ldots, x_k\}$, assume that it is the global optimum. $\{x_1, \ldots, x_k\}$ being the interpolation points. The next step is to fit the surface through y and $x_1, \ldots, x_k$. If the surface is very "bumpy", one might not accept that y is a global minimizer. The bumpiness is measured with the function σ, defined in Equation (2.31) below. The strategy is now to choose the next evaluation point to be the point whose radial basis function minimizes σ. Since this includes no objective function evaluations, it is a rather cheap operation.

We next give a more formal description of this algorithm. The goal function is calculated at the points $x_1, \ldots, x_k$. The crude estimate f_k^* of the global optimum is computed as described below. For each $y \notin \{x_1, \ldots, x_k\}$ there exists a radial basis function s_y such that

$$\begin{aligned} s_y(x_i) &= f(x_i), \qquad i = 1, \ldots, k, && (2.27a)\\ s_y(y) &= f_k^*. && (2.27b) \end{aligned}$$

The next point x_{k+1}, where the objective function is calculated, is equal to the vector whose radial basis function is the least bumpy, that is, minimizes $(\sigma(s_y))$. The radial basis function interpolant has the form

$$s_k(x) = \sum_{i=1}^{k} \lambda_i \phi(\|x - x_i\|_2) + b^T x + a, \tag{2.28}$$

where $\lambda = (\lambda_1, ..., \lambda_k)^T \in \Re^k$, $b \in \Re^n$, $a \in \Re$, and $\phi(r) = r^3$ or $\phi(r) = r^2 \log r$. To find the parameters λ, b and a, one solves the system of linear equations:

$$\begin{pmatrix} \phi & P \\ P^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}, \tag{2.29}$$

where ϕ is a $k \times k$ matrix with element (i, j) equal to $\phi(\|x_i - x_j\|_2)$,

$$P = \begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_k^T & 1 \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_k \end{pmatrix}, \quad c = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ a \end{pmatrix}, \quad \text{and} \quad F = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_k) \end{pmatrix} \tag{2.30}$$

The function σ is defined as

$$\sigma(s_k) = \sum_{i=1}^{k} \lambda_i s_k(x_i). \tag{2.31}$$

It is shown in [7] that $\sigma(s_y)$ equals

$$\sigma(s_y) = \sigma(s_k) + \mu_k(y)[s_k(y) - f_k^*]^2, \qquad y \in \mathcal{S} \backslash \{x_1, ..., x_k\}, \tag{2.32}$$

where $\mu_k(y)$ is the coefficient corresponding to y of the Lagrangian function L that satisfies

$$\begin{aligned} L(x_i) &= 0, \qquad i = 1, \ldots, k, \\ L(y) &= 1. \end{aligned} \tag{2.33}$$

Thus minimizing $\sigma(s_y)$, $y \in \mathcal{S}$ is equivalent to minimizing g_k, where g_k is defined as

$$g_k(y) = \mu_k(y)[s_k(y) - f_k^*]^2, \qquad y \in \mathcal{S} \backslash \{x_1, ..., x_k\}, \tag{2.34}$$

How the target values f_k^* are chosen is important for the performance of the algorithm. If the value of f_k^* is very low the algorithm will perform a global search. If f_k^* is nearer the minimum of $s_k(y)$ the search will be more local. So a good mixture of low and high values of f_k^* for different k is vital. Two different strategies are implemented in *rbfSolve*. In [7] Gutmann rejects one of the strategies as "not useful anymore"; left is just one strategy to perform a cycle of length $N+1$, $N \in \mathcal{N}$ and choose f_k^* as

$$f_k^* = \min_{y \in \mathcal{S}} s_k(y) - W \left(\max_i f(x_i) - \min_{y \in \mathcal{S}} s_k(y) \right), \tag{2.35}$$

where

$$W = \left[\frac{(N - [k - k_{init}]) \text{ modulo } (N+1)}{N} \right]^2 \tag{2.36}$$

where k_{init} is the initial number of points. In the Tomlab implementation of Gutmann‘s algorithm N is fixed to be 5. The maximum of $f(x_i)$ is not taken over all points, except for the first step of the cycle. In each of the subsequent steps in the cycle, the $k - k_{max}$ points with the largest function values are not considered when computing the maximum. The maximum of $f(x_i)$ is decreasing, and therefore f_k^* will increase throughout the cycle. The cycle starts with global searches and tends towards more local searches. Left to define is:

$$k_{max} = \begin{cases} k, & \text{if } [(k - k_{init}) \text{ modulo } (N+1)] = 0, \\ \max\{2, k_{max} - \text{floor}((k - k_{init})/k)\} & \text{otherwise} \end{cases} \tag{2.37}$$

A more concise formulation of the algorithm is given in Algorithm 6.

Convergence is guaranteed for every continuous function f if the sequence of target values f_k^* is chosen carefully. Then there will be infinitely many function evaluations and the sequence (x_k) will be dense in the feasible area $\mathcal{S}$. In other words, if we evaluate every point $y \in \mathcal{S}$ the algorithm will converge to the global optimum. In practice, however, the algorithm will not have to visit every point in $\mathcal{S}$, in order to get an improvment in the objective function.

Algorithm 6 rbfSolve

Step 1: **Initialization.** Choose k initial points $x_1, ..., x_k$. Compute $F_i = f(x_i)$, $i = 1, 2, ..., k$, and set $k_{init} = k$. In the Tomlab implementation $k_{init} = 2n + 1$.

Step 2: **Start cycle.** Start a cycle of length $N + 1$. For every run in the cycle, perform Step 4–9. In the first run perform also step 3, before Step 4–9.

Step 3: **Stop criterion.** If the maximum number of function evaluations is reached, then quit.

Step 4: **Compute** s_k**.** Compute the radial basis function interpolant s_k by solving the system (2.27) of linear equations.

Step 5: **Minimize** $s_k(y)$**.** Solve the minimization problem $\text{minimze}_{y \in \mathcal{S}}\, s_k(y)$. The output, y^*, is used in the step 6.

Step 6: **Compute** f_k^***.** Compute f_k^* according to (2.35).

Step 7: **Find** x_{k+1}**.** The point x_{k+1} is the value of y that minimizes $g_k(y)$ according to (2.34).

Step 8: **Compute** F_{k+1}**.** Compute $F_{k+1} = f(x_{k+1})$ and set $k = k + 1$.

Step 9: **Goto.** If we have reached the end of cycle, go to step 2. Otherwise go to step 4.

Chapter 3

Project 1 - Testing the algorithms

The aim of this project is to test and compare the different algorithms on a smaller problem. The goal is to discharge some methods as unsuitable for this type of problems and not having to test them on the larger problem in the main project. The problem in this project is to maximize the power of the engine. In other words, we wish to maximize the torque at 6000 rpm, since power equals speed times torque, and 6000 rpm is the highest speed of the engine. A popular measure of engine performance is the number of horsepowers, which is also frequently used in the marketing of cars. Therefore, great effort is made to maximize the power.

3.1 Experimental design

In this section the experimental design will be presentated. The work with the experiment is divided into three parts, improving the WAVE-model in order to gain faster calculations, see Section 3.1.1, configuration of the software so that all the different software communicate with each other, see Section 3.1.2 and finally the optimization, see Section 3.1.3.

3.1.1 Improving Wave

Because of the nature of the optimization methods explored in this work we had to speed up WAVE. Several iterations are needed and every iteration includes many evaluations of the objective function. The time step in explicit methods depends on the size of the smallest component cell and is therefore almost constant for different values of speed (rpm). This causes the computational time to be longer at low speeds than at high speeds. With ten different speeds, one sweep between 1500 and 6000 rpm took more than 20 minutes in the beginning. When we increase the discretization lengths for the elements which were the bottlenecks, the computational time shrinks to about 6 minutes with fairly little difference in output results, see Figure 3.1.

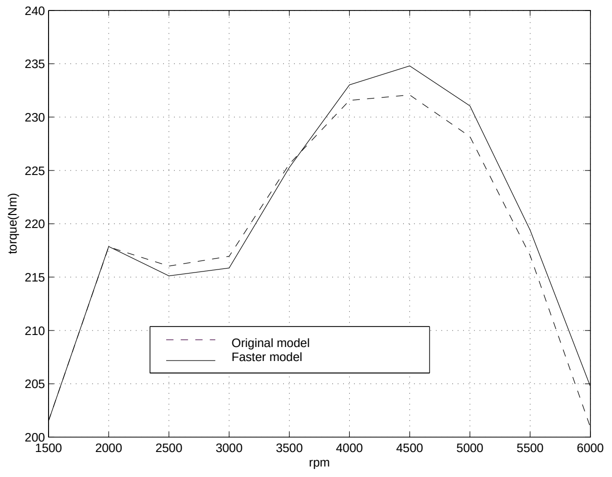


Figure 3.1: Comparison between the original model and the "fast" model with increased discretization lengths

3.1.2 Software configuration

Next we have to get Wave and Matlab to communicate. The input to Wave is a .wvm -ASCII file which defines the geometry of the engine, valvetiming etc. We create a Matlab program *autowvm.m* that produces a .wvm-file given a certain set of variables x. This file is used in all of our algorithms. The .wvm-file is run in Wave and the output is a .stb-file, which is a .ASCII file containing the value of the torque. For the algorithms written in Matlab code, fully or partly, the process looks like in Figure 3.2:

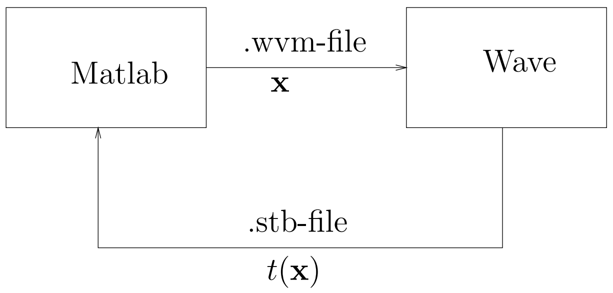


Figure 3.2: Flow chart for algorithms written in Matlab code

For NEWUOA, which is written in Fortran 77, the process scheme can been seen in Figure 3.3, below. Matlab is still used to start and control Wave. The vector x is placed in Matlab's workspace, using the possibility to call Matlab from a Fortran application.

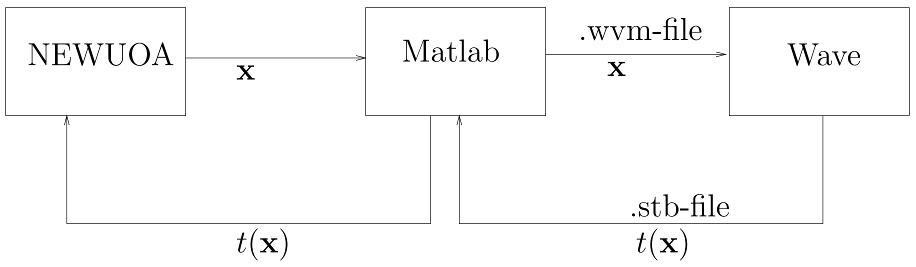


Figure 3.3: Flow chart for NEWUOA

In Table 3.1 we present a short list of m-files developed for the different methods we choose to implement in project 1.

3.1.3 Optimization

When the software development is complete for the different algorithms, we test them on our problem. The objective function is to maximize torque at 6000 rpm, (see section 1.4.1 for notation), which we formulate as:

$$\underset{x}{\text{minimize}} \quad -t(\mathbf{x}) \tag{3.1}$$

In other words we have an unconstrained problem and all the methods except rbfSolve will solve it as an unconstrained problem. RbfSolve can only solve box-bounded problem so we have to convert our unconstrained problem into a box-bounded one. It is possible the optimal solution becomes infeasible when the unconstrained problem is converted to a box-bounded one. The bounds are therefore set so that it is unlikely that the optimal solution becomes infeasible.

The local solvers need a starting point which is chosen to be the current values for the existing engine as seen in Table 3.2. The global solver rbfSolve does not need a starting point, as mentioned above it needs only the upper and lower bounds on the variables, which are also specified in Table 3.2. The trust-region radius and upper and lower bounds are set after consultation with our supervisor at VCC.

Table 3.2: Starting values and bounds for the different variables

Variable	x_0^i	Lower bound	Upper bound	Trust-region radius
Cam-top angle intake (°)	487	430	510	8
Cam-top angle exhaust (°)	250	220	300	8
Cam duration intake	1	0.95	1.3	0.025
Cam duration exhaust	1	0.95	1.3	0.025
Intake-pipe diameter (mm)	43.5	20	60	3
Intake-pipe length (mm)	325	100	500	10

The stopping criteria varies between the different algorithms, according to the following:

Table 3.1: Matlab files Project 1

Our Method	
TheMother.m	The executable file where the starting point, x_0, is set and the variables are defined
plan.m	Decides which points to interpolate and scales them into a unit cube
Polynom.m	Creates a polynomial by interpolation by calling *lsqcurvefit*
Opt.m	Applies *quadprog* to our polynomial
rbfSolve	
setMoreSpec.m	Definition of the problem. The variables to use, how many iterations to do, etc.
drive.m	Executes the algorithm, sets the algorithm's specifics (which are the same for all problems).
FuncEval.m	Evaluates the objective function by running Wave
NEWUOA	
main.f	A Fortran file where the number of variables and iterations are set. Its subroutines manages the optimization
calfun.f	A Fortran subroutine that calls Matlab to execute start.m
start.m	Call on calfun.m and receives the objective value and exits Matlab
calfun.m	same as FuncEval.m
Wedge	
driver.m	The driver of wedge.m
wedge.m	The framework for the algorithm
setSpecs.m	Sets the number of iterations, defines the update of the trust-region, and the convergence tolerance
user_ufn.m	Objective function evaluation and scaling of the variables
user_usetup.m	The origin
GPS	
func.m	Objective function evaluation
func_x0.m	The starting point

Table 3.3: Result from project 1

Algorithm	Torque (Nm)	# function evaluations
Our method	209.15	168
GPS	210.80	200
rbfSolve	213.06	200
NEWUOA	212.417	118
WEDGE	212.00	127

Our Method: Stops after a specified number of iterations.

rbfSolve: Stops after a specified number of function evaluations.

NEWUOA: Stops when $\rho = \rho_{end}$.

Wedge: Stops when the trust-region radius Δ was smaller than 0.1.

GPS: Stops after a specified number of function evaluations.

3.2 Results and discussion

Before the optimization of the variables, the torque at 6000 rpm was 204.75 Nm. All of the algorithms find solutions which increase the torque. The algorithms and their stopping criteria are very different from each other. It is hard to compare different algorithms without knowing the optimal solution in advance, and since the function evaluations are costly we don't want to perform too many. Especially in the main project, when the function evaluations will become even more expensive, it is important to reach a result quickly. The most fair test to perform is to let the different algorithms use the same number of function evaluations. We set the maximum number of function evaluations allowed to 200. The result of the optimization and the number of function evaluations used by different algorithms is presented in Table 3.3. One of them - our method - can't use all the function evaluations because each iteration takes a certain amount of evaluations that have to be carried out before a new point can be found. NEWUOA and WEDGE converges and does not improve after 118 and 127 evaluation respectively.

In Table 3.3 one can see that rbfSolve gives the best result after 200 function evaluations and NEWUOA gives the second best value using fewer function evaluations than rbfSolve. The different algorithms find different solutions to the problem as seen in Table 3.4.

Table 3.4: x^* for project 1

Algorithm	x^1	x^2	x^3	x^4	x^5	x^6
Our method	483.17	247.5	1.050	1.043	49.2	330.8
GPS	487.0	250.0	1.075	1.025	66.7	427.4
rbfSolve	482.1	247.2	1.078	1.033	53.0	277.5
NEWUOA	480.4	246.1	1.071	1.011	54.7	284.0
WEDGE	479.1	247.4	1.060	1.035	63.0	303.8

Since the variables don't have the same unit it is hard to see what a large change in the value of a variable is and what a smaller one is, but the size of the trust-region radius in Table 3.2 gives us a hint. In Table 3.5 the change in the different variables in number of original trust-regions are displayed. The unit is a bit awkward for the global solver rbfSolve, which neither have a trust-region nor does it has a start value x_0. GPS don't have a trust-region radius, but it has an initial mesh size which in some ways plays the same role as a trust-region. The initial mesh size doesn't have the same value as the trust-region in Table 3.2. Despite these differences Table 3.5 still gives an idea of which variables have changed the most in the algorithm's respective solutions.

Table 3.5: x^* for project 1 in the unit number of original trust-regions

Algorithm	x^1	x^2	x^3	x^4	x^5	x^6
Our method	-0.4792	-0.3066	2.0000	1.7213	1.9069	0.5769
GPS	0	0	3.000	1.000	7.7333	10.2400
rbfSolve	-0.6159	-0.3441	3.1080	1.3360	3.1495	-4.7452
NEWUOA	-0.8438	-0.4931	2.8500	0.4281	3.7307	-4.1039
WEDGE	-0.9871	-0.3252	2.7893	1.4109	6.4926	-2.1191

As one can see, the changes are small with some notable exceptions. It's interesting to see that rbfSolve finds a solution fairly close to both NEWUOA's and WEDGE's solution. This indicates that the starting point lies near the point we believe to be a global optimum, since we find a solution very close to the original with both global and local search methods. It also indicates that the objective function behaves well near the starting point; an objective function that, for example, have a barrier near the global optimum will trick the trust-region to wander away in a wrong direction. The algorithms which are provided with a starting point change the variable values in a way that is typical for each algorithm. Our method uses many points to build a model and is therefore very slow. GPS changes one variable at a time and the length of the step is due to the mesh size; the large change in x^6 shows that at some point during the optimization the mesh size has been very large and the algorithm found a better value for the objective function when increasing the value of x^6. GPS can't find a solution that makes smaller changes in all variables at the same time, since it only changes the value of one variable each iteration. Both NEWUOA and WEDGE are trust-region methods; it is therefore not surprising that they show the same behavior when given the same starting point and initial trust-region radius.

3.3 Conclusions

We have tested five algorithms on the same problem. All the algorithms tested improved the performance of the engine at 6000 rpm. Some algorithms improved the performance less than the others. The two algorithms whose improvment of the objective functions wasn't as good as the other algorithms was our method and GPS. Some algorithms used more function evaluations than the others to achieve almost the same result. Since function evaluations are expensive, that is, time consuming and will be even more expensive in the next project, the number of function evaluations is critical when choosing which algorithms to use in the next project. For the algorithms that build models of the goal function, the number of function evaluations used to do this is also important. So there are three criteria to consider when choosing the algorithms to use in the main project:

1. Expected improvment of the objective function;
2. Few function evaluations;
3. Fast model building;

NEWUOA, rbfSolve and WEDGE gave the best results; NEWUOA and WEDGE used the fewest function evaluations. The natural conclusion would be to use NEWUOA and WEDGE in the main project. However, WEDGE uses $\mathcal{O}(n^2)$ functions evaluations to build a model of the goal function which means the number of function evaluations grows as n^2 when the number of variables increases compared to NEWUOA and rbfSolve which just use $\mathcal{O}(n)$. Even worse is the fact that we have to solve the problem five times in WEDGE with different sets of initial points, to be sure that the result is stable and can't improve with a different set of initial points. With this in mind it is not wise to use WEDGE when the number of variables increases and the function evaluations become more costly. The conclusion is that NEWUOA and rbfSolve is worth utilizing in the main project.

Chapter 4

Project 2 - Optimizing the serial intake valves

In this chapter we will solve and discuss the problem to optimize the engine with the SIV described in Section 1.2.1. In Section 4.1.1 we solve the problem with a dip in the torque graph. The way the optimization is performed, how the objective function is chosen and similar matters are discussed in Section 4.1.2. The result of the optimization is presented in Section 4.2 and discussed in Section 4.3, together with conclusions drawn from our work.

4.1 Experimental design

4.1.1 The mystery at 2000 rpm

When we began our work with this thesis, our supervisor at VCC told about a dip in the torque graph at 2000 rpm when serial intake valves are used. This phenomenon struck him as a bit odd, because when the SIV are not in use there is instead a peak at this speed. After some investigations we found that this phenomenon originates on the intake side, that is, between the air-intake and the plenum. When we studied the flow at 2000 rpm in the frequency spectrum we saw a standing wave in the intake system over the air filter. This standing wave seems to cause the dip. To deal with this problem, there is a number of opportunities:

1. One can lengthen the intake pipes in order to move the dip towards lower engine speeds where it doesn't cause much damage,

2. one can reduce the diameter of the intake pipes for the same reason as above, or

3. one can place a resonator on the air filter, which works at a certain frequency with a suitable bandwidth.

We tried all of the methods above and came to the conclusion that the last alternative was the best, since that was the alternative that altered the original engine the least. The resonator that we choose was a Helmholtz resonator, placed on the air filter.

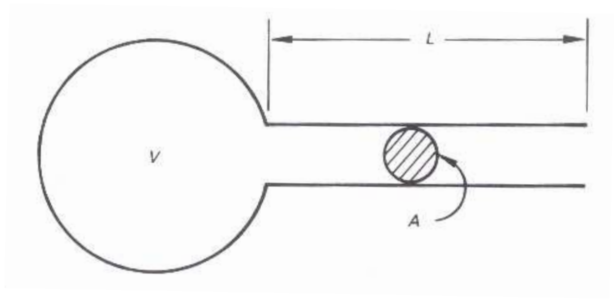


Figure 4.1: Helmholtz resonator

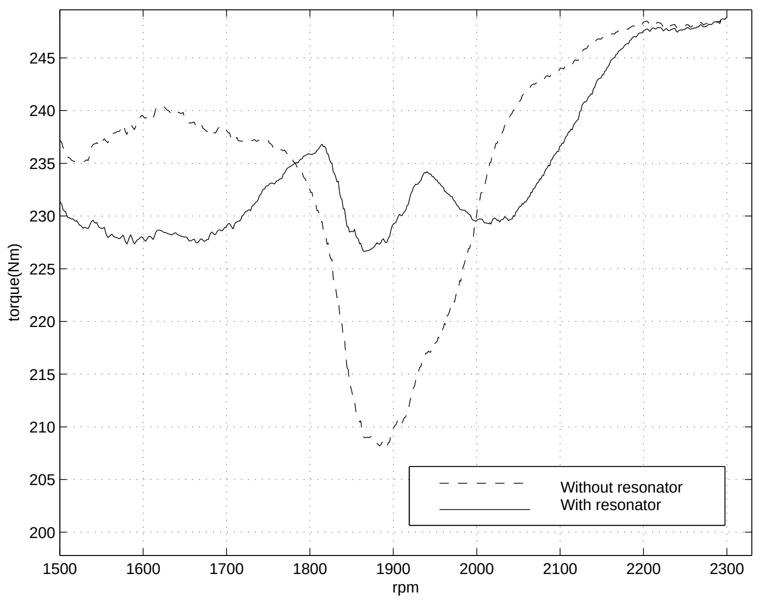


Figure 4.2: The effect of a Helmholtz resonator

A Helmholtz resonator is an acoustic device, often found in audio speakers. It is built like a bottle, with a neck of length L, a cross sectional area A and a volume V, see Figure 4.1. To calculate L when A, the frequency f and V are given, we use Equation (4.1),

$$f = \frac{c}{2\pi}\sqrt{\frac{A}{VL}} \tag{4.1}$$

where c is the speed of sound. After studying the engine in the frequency domain we could pin down the frequency we wanted to suppress to $f = 84$ Hz. With $V = 1\,\mathrm{dm}^3$ and $A = 1250\,\mathrm{mm}^2$ the length of the pipe becomes $L = 534$ mm. When we attached this resonator to our engine we saw that it did exactly what it was supposed to do, suppress the dip, as illustrated in Figure 4.2. This design was not optimized due to the limited time allocated for this thesis.

4.1.2 Optimization

The problem and its mathematical model are found in Section 1.4.1. As it is formulated there, the number of variables are too many. The large number of variables would be a problem even if the function evaluations were not so costly since we have no information about the derivatives. Our solution is to separate the problem into two: One problem deals with the hardware variables and the other deals with the software variables. First, the entire problem is solved for a small number of engine speeds. This is done in order to fix the values of the hardware variables. Then the hardware variables are fixed and the software variables only are optimized for a larger number of speeds.

It is not useful to optimize the hardware for just one speed. We have to optimize for several speeds simultaneously, otherwise the hardware variables can be set to something that is unsuitable for many other speed values. As an example, long cam durations are preferable at high speeds whereas the opposite is true for low speeds.

The plan is to optimize hardware and software variables at four different speeds: 1200, 3000, 4500 and 6000 rpm. These speeds are chosen so that as much as possible of the speed range is covered. It is necessary to optimize both software and hardware variables during this phase, in spite of the fact that the main purpose is to set the hardware variables. If both hardware and software variables are optimized simultaneously, the benefits of the SIV (software) are taken into consideration.

Since we are only concerned about four different speed values, we don't run the simulation of the engine transient from 1000 to 6000 rpm. Instead we simulate the engine at each speed considered until WAVE converges. The mathematical formulation of the hardware optimization is (see Section 1.4.1 for notation):

$$\begin{aligned}
\text{minimize} \quad & f(\mathbf{x},\mathbf{y}) = -\sum_{s\in\mathcal{S}} t(\mathbf{x},\mathbf{y}^s), && \text{(4.2a)}\\
\text{subject to} \quad & \mathbf{l}^x \le \mathbf{x} \le \mathbf{u}^x, && \text{(4.2b)}\\
& \mathbf{l}^y \le \mathbf{y}^s \le \mathbf{u}^y, \qquad \forall s \in \mathcal{S} && \text{(4.2c)}\\
& x_1 \ge x_2, && \text{(4.2d)}\\
& y_3^s \ge y_1^s + y_2^s, \qquad \forall s \in \mathcal{S}, && \text{(4.2e)}
\end{aligned}$$

where $\mathcal{S} = \{1200\ 3000\ 4500\ 6000\}$. The difference between this formulation and the one in Section 1.4.1 is that here the EGR is not minimized, since just adding SIV to the engine will lower the EGR and we believe that this reduction will be enough to avoid knocking, see Section 1.2. To receive an even lower EGR, one should choose $\gamma_s > 0$ in (1.7).

Next, we implement the two optimizing algorithms which we expect to give the best results, based on the experience from the smaller problem, that is, rbfSolve and NEWUOA. We allow the algorithms to use at most 300 function evaluations. However, NEWUOA is for unconstrained optimization and Problem (4.2) is a constrained problem. This problem is solved by the fact that when the algorithm gives WAVE an infeasible point as input, the output from WAVE is very bad, that is, the objective value $f(\mathbf{x},\mathbf{y})$ is very high. This means that an infeasible point will give a result that the algorithm considers as bad and it will therefore reject that point. One could say that the constraints are embedded in the objective function. rbfSolve, on the other hand, handles linear constraints.

In part two of the larger problem, our goal is to optimize the software variables. It is not possible to optimize all the software variables at all speeds since then there will be too many variables. We choose to optimize for one speed at a time. This is possible since the software variables at two different speed values are not much related. The fact we have to consider is that if the software variables varies too much with speed, there will be transient effects when the SIV are too slow to keep up with the rapid changes.

Since it is not possible to optimize the software variables for every possible speed value, one must discretize the speed interval and let the input to the engine be a linear interpolation of the software variables between the discrete speed values. Our solution is to solve the problem for each speed given in Table 4.1. The constraints (4.3b) keep the changes of the values of the variables between speed values at a reasonable level. These constraints yield more gentle transitions in the values of software variables and reduce the problem that the SIV can't keep up with rapid changes. More speed values are chosen among the lower end values and where dips in the torque-curve typically occur than among the high values.

Table 4.1: The engine speed values $s \in \mathcal{S}$ at which the software variables are optimized

s	rpm	s	rpm	s	rpm
1	1300	7	2000	13	3300
2	1450	8	2100	14	3450
3	1600	9	2200	15	3600
4	1700	10	2600	16	4000
5	1800	11	2800	17	4400
6	1900	12	3000	18	4800

Hence, the following sequence of problems is solved in the order, $s = 1, ..., 18$:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{y}^s) = -t(\mathbf{y}^s), && (4.3a)\\ \text{subject to} \quad & \mathbf{y}^{s-1} - \Delta \leq \mathbf{y}^s \leq \mathbf{y}^{s-1} + \Delta, && (4.3b)\\ & y_3^s \geq y_1^s + y_2^s, && (4.3c) \end{aligned}$$

where $\Delta \geq 0$[crank angles] are the allowed differences in software variables between two subsequent speed values. In other words we have 18 problems with four variables in each problem. The problems are solved separately, each time increasing the speed, starting with 1300 rpm. The starting point of the problems is taken from the previously solved problem $(s-1)$, except when $s = 1$ where it is taken from the hardware optimization solution at 1200 rpm in the first project. The limit of the function evaluations is 50 per speed value.

For this problem we choose not to use the algorithm NEWUOA, since it does not handle constraints. Furthermore its strength is to solve problems with many variables and these problems only have four variables each, so we will use rbfSolve here.

4.2 Results and discussion

The improvement made in the first part, optimizing the hardware variables according to Problem (4.2), are shown in Figure 4.3. Both NEWUOA and rbfSolve was used to solve this problem. NEWUOA and rbfSolve used 200 and 300 function evaluations respectively.

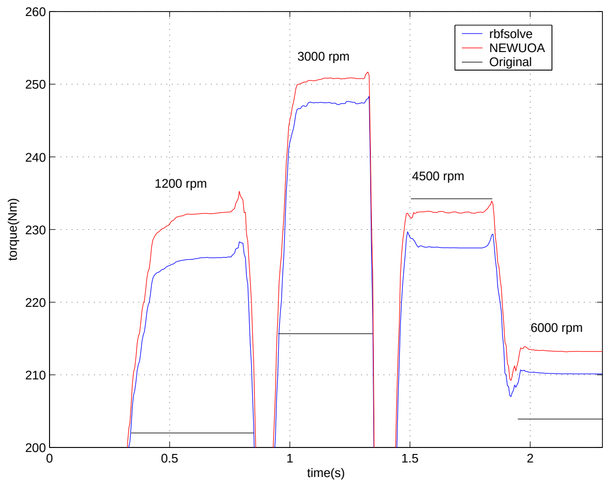


Figure 4.3: Results of hardware optimization

Clearly, NEWUOA performs better with a fewer number of function evaluations. There is an improvement in the torque at almost all speeds considered. The resulting torque at each speed value is taken right before the little peak at the end of each speed considered. This peak arises when we change the speed in WAVE rapidly and it can be ignored. Notable is the remarkably high torque at 6000 rpm, that is, the highest power obtained. It is almost as high as in project one where we considered 6000 rpm only, without SIV. This indicates that the use of SIV results in an engine where the hardware design is focused on the high-end speed range whereas the SIV themselves increase the torque at lower speeds. The values of the optimized hardware variables are shown in Table 4.2

Table 4.2: Values of the optimized hardware variables

	rbfSolve	NEWUOA
x_1	466	470
x_2	258	251
x_3	1.20	1.18
x_4	0.95	1.08
x_5	49.1	49.7
x_6	333	326

With the values of the hardware variables from NEWUOA:s solution we started the software optimization with rbfSolve. With 50 function evaluations for each $s \in \mathcal{S}$ and

$\Delta = [20\ 15\ 20\ 15]$, the values of the software variables converged to those shown in Figure 4.4.

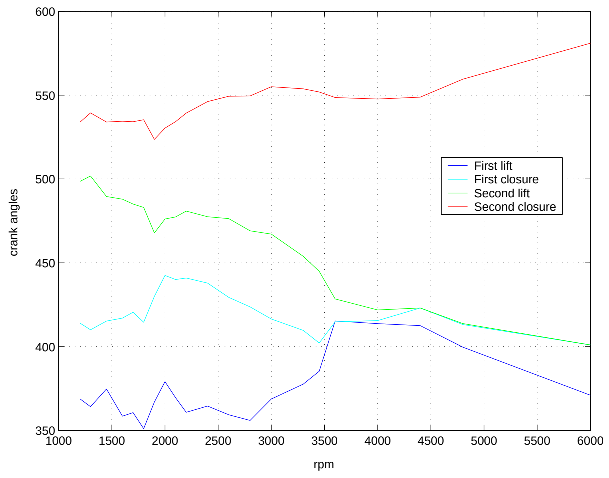
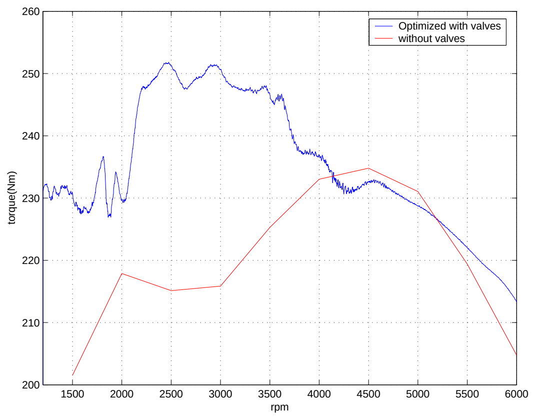


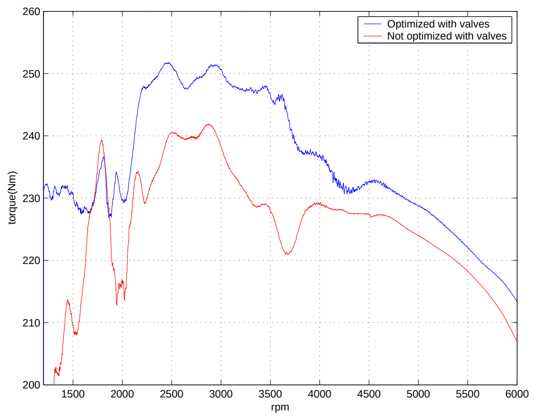
Figure 4.4: The changing of software variables

In Figure 4.4 it can be seen that there are also some ripples around 2000 rpm. This indicates that the objective function is still a bit unsmooth, despite our efforts with the Helmholtz resonator. When the two lower curves coincide around 3500 rpm we get only one lift and one closure since the first closure and second lift occurs at the same time. This is optimal since it is favourable to keep the SIV open as long as possible at higher speeds and since the SIV aren't fast enough to open and close twice at high speeds.

These values of the software and hardware variables give the optimized torque-curve seen in Figure 4.5.



(a) Effect of both valves and optimization

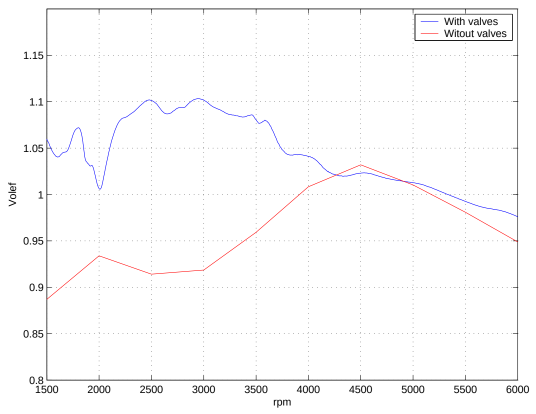


(b) Effect of optimization

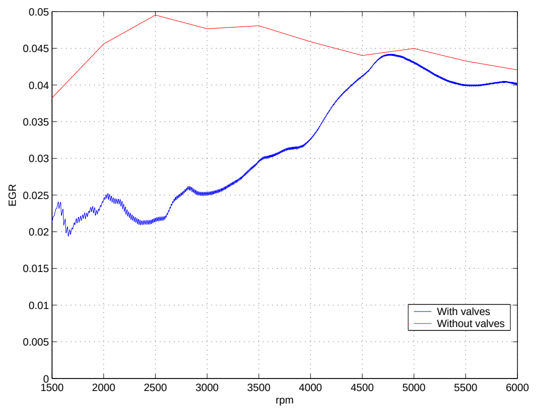
Figure 4.5: Effects on torque due to optimization and SIV

As can be seen in Figure 4.5 there is a significant improvement of the torque in the range between 2000 and 3500 rpm which are the most frequently used engine speeds.

Another interesting result which, to some extent, explains the raise in the torque is the improvement achieved in volumetric efficiency as seen in Figure 4.6(a). It is easy to see that there is a correlation between volumetric efficiency and torque. A well known fact in the automotive industry is that high volumetric efficiency is a necessary, but not sufficient, condition for high torque.



(a) Volumetric Efficiency



(b) Exhaust Gas Residuals

Figure 4.6: Optimization results that may effect knocking

The big concern is now the effect of knocking; see Section 1.2. We do not know whether or not it will occur. The risk for knocking increases with higher volumetric efficiency. As mentioned, one way to lower the probability of knocking is to lower the exhaust gas residuals (EGR). What happened to the EGR after the optimization is shown in Figure 4.6(b); we can see that the residuals are almost halved compared to the original values for low speeds. This helps keeping the temperature down and hopefully prevents knocking.

4.2.1 Fuel consumption

With the torque-curves shown above, there is a potential for reducing the fuel consumption. The high, low-end speed torque, allows a change in the gear ratio, that is, to put in an extra gear like an overdrive. We did not have much time to focus on fuel consumption but Jan Karlsson at VCC helped us to estimate the fuel consumption using a computer program developed for such calculations. We applied the standard driving cycle, the ECUC (European Combined Urban Cycle), which is a combination of the EUDC (Extended Urban Driving Cycle) and UDC (Urban Driving Cycle), and which is used to certify the engine. The fuel consumption, together with vehicle speed and gear for EUDC and UDC are shown in 4.7.

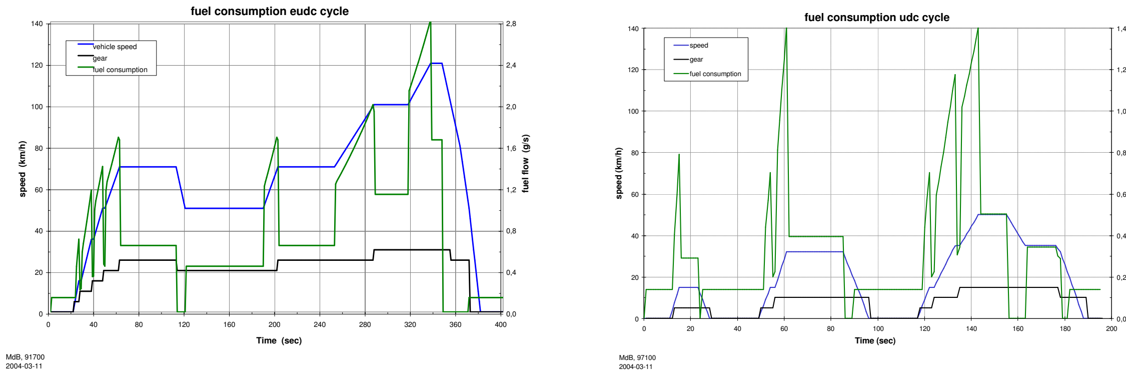


Figure 4.7: Driving cycles and fuel consumption

Because of the high, low-end torque we managed to decrease the fuel consumption with about 14% compared to an engine without SIV and with about 5% to one with SIV but not optimized, as seen in Table 4.3.

Table 4.3: Fuel consumption (g/kWh)

	Without SIV	SIV	**Optimized with SIV**
UDC	616	538	507
EUDC	342	314	306
ECUC	441	395	**379**

4.3 Conclusions

If resonance phenomena occur we have to find out why they happen and then find a way to compensate for them. It is very unlikely that the optimization process will fix them, since most of the algorithms require a smooth goal function; resonance phenomena result in a very unsmooth objective function.

We have tested two algorithms, NEWUOA and rbfSolve, on part 1 of this problem and one algorithm, rbfSolve on part 2. The result is a higher torque at the low-end speeds and more power, that is, higher torque at 6000 rpm. Between the low and the high-end speeds the torque is almost the same (or higher) as for the engine without serial intake valves. We believe that the reason why the engine without SIV has a higher torque than the engine with SIV between 4250 rpm and 5250 rpm is that the engine without SIV has a variable valve timing (VVT) which the engine with SIV doesn't have. The volumetric efficiency is above one for all speeds below 5000 rpm, and the EGR is almost halved, compared to the original engine. There is a tradeoff between EGR and volumetric efficiency; the higher the volumetric efficiency is the lower the EGR has to be to avoid knocking. Otherwise, the improvements that we predicted in our hypothesis has been obtained.

In Figure 4.3 one can see that NEWUOA, using fewer function evaluations, finds a solution that is better than that found by rbfSolve. The comparison might be a bit unfair: NEWUOA starts from a good point and rbfSolve starts with an area around the good starting point. But on the other hand the algorithms have different requirements. The conclusion is that if a good starting point exists a local optimizer like NEWUOA should be used.

So far everything looks good, but high volumetric efficiency gives high pressure and as mentioned in Section 1.2, high pressure might cause knocking. The problem is that WAVE does not predict knocking. It should be necessary to perform some real engine testing to see if knocking occurs. On the other hand, we believe that the SIV scavenging of the cylinders will allow a higher volumetric efficiency than traditional engines, since lower EGR-values indicate lower temperatures. To avoid knocking one could redo the optimization with a goal function that gives priority to volumetric efficiency close to one at low speeds, and to a high torque at higher speeds.

The minimum time required for opening and closing the SIV is probably a limiting factor in how large improvement that can be acchieved. We have touched this issue very briefly, and the preliminary results show that it would be good if the SIV were able to open and close in 1 ms, instead of today's 1.8 ms. If it isn't possible to speed up the SIV one has to consider to open just once during the intake stroke which results in two software variables only for each engine speed.

We have tested just one objective function for each part of the project. As mentioned in Section 1.6, it is hard to mathematically formulate a goal function for the problems in this thesis. More work could thus be done on improving the expression of the objective function.

Chapter 5

Conclusions

This thesis shows the value of utilizing mathematical optimization in engine development. We have studied and implemented various derivative-free optimization methods. The expectancy of a 5% improvement in fuel consumption and torque compared to the initial solution has been reached and sometimes exceeded. Depending on problem type, different choices of algorithms are advantageous. Below follows a summary of the methods that we recommend to be used in the future and how the implementation of the respective algorithms works.

rbfSolve (Tomlab) is a global solver, that is, it needs no starting point and guarantees to find the global optimum to a box-bounded problem given enough time (maybe eternity). However it produces good results in a reasonable time for our type of problem. rbfSolve is the only method presented here that can handle constraints explicitly. It possesses a user-friendly syntax (Tomlab's) in Matlab and an extensive manual. We found some bugs in the source-code, these were corrected by Tomlab personnel. The results are displayed in a good way and stored while running so that warm start is possible (Warm start means that it is possible to continue the optimization process from the last iteration, even if the process has terminated.). Since WAVE sometimes terminates unexpectedly, warm start is a useful feature.

NEWUOA is a local solver and needs a starting point which is often available in industrial simulations since the problem often is to improve something that is already developed. It gives excellent results for large problems, with more than 15 variables, in reasonable time. For smaller problems it seems like rbfSolve performs better, but this depends of course on the starting point. NEWUOA has better convergence properties than rbfSolve. The downside is the implementation which is made in Fortran 77; it can be rather difficult to get a deep understanding of what happens inside the algorithm. We have made it possible to use Matlab for function evaluations, which makes NEWUOA a lot easier to use.

WEDGE is also a local solver. It gives good results for smaller problems with less than 10 variables. Since it requires $m = \frac{1}{2}(n+1)(n+2) - 1$ function evaluations to build a model of the goal function it is not useful for larger problems. It also has a convergence criterion, which is as good as NEWUOA's. Some of the code is written in Fortran but has a Matlab interface; see [18].

5.1 Further work

The methods presented here are very general. They apply to almost all conceivable optimization problems, even though there are better methods when explicit derivatives are available. Since there exists a vast number of simulation tools in industry today, where derivatives are not explicitly available, we see a great potential in applying these methods elsewhere.

Of course, much work is left on developing user-friendliness and solving stability issues. We think, but are not sure, that the results found here are stable; it is possible to repeat our calculations and get the same result. Also, if a completely new problem with another simulation tool is to be solved, more experience with the algorithms and their usage are necessary.

There are non-derivative algorithms which we have not considered due to the limited amount of time avaible for this thesis. New algorithms are developed continuously, for example; DFO, see [4], which is a trust-region method that can handle constraints.

Another path is also interesting. It may be possible to extract the derivatives directly from WAVE. We have not been able to investigate this option since it seems to be very difficult. If derivatives were available, it would be possible to make use of a large number of Newton-methods, for example.

As a closing remark, we would like to emphasize that simulation optimization can be a valuable tool to improve or even discover new solutions in the area of engine development. The components that are required are a simulation tool, an efficient optimization algorithm, and an objective function.

Bibliography

[1] "Mark Abramson's NOMADm Page,"
`http://en.afit.edu/ENC/Faculty/MAbramson/NOMADm.html`, 2003.

[2] C. Audet, J.E Dennis Jr., *Analysis of Generalized Pattern Searches*, SIAM Journal of Optimization, 13 (2003), pp. 889-903.

[3] M. Björkman, K. Holmström *Global Optimization of Costly Nonconvex Functions Using Radial Basis Functions*, Optimization and Engineering, 1 (2000), pp. 373-397.

[4] "COmputational INfrastructure for Operations Research",
`http://www-124.ibm.com/developerworks/opensource/coin/index.html`, 2004.

[5] A.R. Conn, N.I.M Gould, Ph. Toint, *Trust-Region Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[6] H.-M. Gutmann, *A Radial Basis Function Method for Global Optimization*, Journal of Global Optimization 19 (2001), pp. 201-227.

[7] H.-M Gutmann, *Radial Basis Function Methods for Global Optimization*, PhD Thesis, Department of Applied Mathematics Theoretical Physics, University of Cambridge, Cambridge, 2001.

[8] "lsqcurvefit (Matlab function reference),"
`http://www.mathworks.com/access/helpdesk/help/toolbox/optim/lsqcurvefit.shtml`, 2003.

[9] M. Marazzi, J. Nocedal, *Wedge Trust Region Methods for Derivative Free Optimization*, Mathematical Programming, 91 (2002), pp. 289-305.

[10] J. Moré, D.C. Sorensen, *Computing a trust region step*, SIAM Journal on Scientific and Statistical Computing, 4 (1983), pp. 553-572.

[11] S. Nash, A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill, Singapore, 1996.

[12] "NEWUOA, fortran optimization code",
`http://plato.la.asu.edu/topics/problems/nlounres.html`, 2004.

[13] M.J.D. Powell, *UOBYQA: unconstrained optimization by quadratic approximation*, Mathematical Programming, 92 (2002), pp. 555-582.

[14] M.J.D. Powell, *On the use of quadratic models in unconstrained minimization without derivatives*, Report No. DAMTP 2003/NA03, University of Cambridge, Cambridge, U.K., 2003.

[15] "quadprog (Matlab function reference)",
`http://www.mathworks.com/access/helpdesk/help/toolbox/optim/quadprog.shtml`, 2003.

[16] "Ricardo WAVE 5.1 Overview",
`http://www.ricardo.com/wave`, 2004.

[17] V. Torczon, *On the convergence of pattern search algorithms*, SIAM Journal of Optimization, 7 (1997), pp. 1-25.

[18] "Wedge Nonlinear Optimization Code",
`http://www.ece.northwestern.edu/~nocedal/wedge.html`, 2003.

Appendix A

Technical dictionary

Table A.1: English-Swedish vocabulary

English	Swedish
camshaft	kamaxel
constraint	bivillkor
crankshaft	vevaxel
exhaust	avgas
exhaust gas residuals (EGR)	restgaser
feasible/infeasible	tillåten/otillåten
Finite Element Method (FEM)	finita element metoden
inlet	insug
ignite	tända
knock	knack
manifold	grenrör
naturally aspirated engine	sugmotor
objective function, goal function	målfunktion
plenum	plenumvolym
revolutions per minute (rpm)	varv per minut
(scavenging	spolning
serial intake valves (SIV)	lufttaktventiler
spark plug	tändstift
throttle	gasregleringsspjäll
torque	moment
valve	ventil
variable valve control (VVT)	variabel ventilstyrning
volumetric efficiency (volef)	volumetrisk verkningsgrad