Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft Institute of Applied Mathematics

Development of a compressor for gas network simulation

(Dutch title: Ontwikkeling van een compressor voor een gas netwerk simulatie)

Report on behalf of the
Delft Institute of Applied Mathematics

to obtain the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

T.M. HODES

Delft, The Netherlands
July 2019

BSc thesis APPLIED MATHEMATICS

"Development of a compressor for gas network simulation"

(Dutch title: "Ontwikkeling van een compressor voor een gas netwerk simulatie")

T.M. HODES

Delft University of Technology

Supervisor

Dr.ir. J.E. Romate

Thesis committee

Prof.dr.ir. C. Vuik	Dr.ir. M. Keijzer

July, 2019 Delft

Preface

Before you lies the thesis '*Development of a compressor for gas network simulation*'. It is written to obtain the degree of Bachelor of Science in Applied Mathematics. The research has been conducted under supervision of Dr.ir Johan Romate from the Delft Institute of Applied Mathematics (DIAM) department of the Electrical Engineering, Mathematics & Computer Science (EEMCS) faculty.

In this thesis I want to give a model of a compressor which can be used in a gas network. A simplified version of a gas network is discussed as well. The network will consist only of a pipeline in front and after the compressor and the compressor itself. Python is the programming language in which the simulation has been done.

I want to thank my supervisor Johan for all the help he gave me during this research. I also want to thank the other two members of my thesis committee Prof.dr.ir C. Vuik and Dr.ir. M. Keijzer for grading my thesis.

I hope you will all enjoy reading my thesis.

Thijmen Hodes
Delft, July 2019

Abstract

In this thesis the development of a compressor for a gas network simulation will be discussed. The modelled compressor is a centrifugal compressor and this compressor is common to use when compressing gas. At first the equations to describe a centrifugal compressor and an example parameter set are given. These equations will be used in an algorithm to calculate either the suction pressure, the discharge pressure or the volume flow. A compressor can only operate when it is in a feasible area. This feasible area is defined by both a surge and choke line as well as the minimum and maximum rotation speed.

The gas network considered consists of a compressor plus a pipeline in front and after the compressor. To model a pipeline the Weymouth pressure-drop equation is used. The pipe-compressor system can be described as a system of seven equations. There are seven unknown variables and two given variables. The given variables are the pressure at the beginning of the first pipeline and the pressure at the end of the second pipeline.

To solve the system of equations a nonlinear solver is used. The Newton method is an example of such a nonlinear solver and will be discussed for scalar equations and a system of equations. Solving the system of equations as part of a simulation, a function in Python called fsolve will be used. This function uses a more robust algorithm to get the solution quickly.

The numerical behaviour of the solver will be considered during multiple simulations. Furthermore a parameter study has been done to understand what kind of pipelines are needed for a certain pressure ratio. Finally two examples are discussed briefly.

Contents

Chapter 1

Introduction

Almost every person in the Netherlands needs gas in their life. It is either to warm their houses or to prepare a meal. The important question is, how gets the gas distributed into the Netherlands? This gas can be distributed by using pipelines and compressors. At about every 100 kilometres there is a compressor station [1]. A compressor station is a station where multiple compressors are working parallel or in serie with each other. These stations make sure that gas will remain at sufficiently high pressure everywhere in the pipeline and that gas is transported to the destination.

At most compressor stations there are two types of compressors [1]. The reciprocating compressor and the centrifugal compressor. A good mix of both compressors will keep the pressure steady. Models of compressors and pipelines are needed to keep the gas at the right pressure. Workers at the compressor stations can then calculate the right settings of each compressor. In this thesis a model for a centrifugal compressor will be discussed. The model is important to calculate the suction pressure (P_s), the discharge pressure (P_d) or the actual volume flow (Q_{ac}). When two of the three variables are known, the third can be calculated. This is important to set the compressors at the right settings.

In the article of Chapman and Abbaspour [1] equations of a centrifugal compressor are elaborated. These equations will form the basis of the model. Every compressor needs to work in a feasible area. When the compressor is outside this area, the performance will drop drastically. What determines a feasible area? Every compressor has a point where there is surge and a point where there is choke. Surge is the operation point where the maximum head (H) and the related volume flow (Q_{ac}) is reached. The head of a compressor is in joule per kilogram. Choke is the point where the volume flow increases and the discharge pressure decreases, such that the compressor does not work anymore. Surge and choke points are both dependent on the rotation speed. Every rotation speed has a specific surge and choke point and together they will form a line. So the feasible area is confined by the surge line and the choke line.

The equations modelling a centrifugal compressor used here are nonlinear equations. Therefore it is not easy to solve them. There are multiple ways to solve nonlinear equations. It can be solved via dedicated nonlinear solvers or via optimization. The algorithm for the known centrifugal compressor will be solved via a nonlinear solver. Examples that can be used to solve the equations are Newton's method, the Secant method or Quasi-Newton methods. The last numerical method is a collective of several numerical methods and they are all derived from Newton's method. The method that will be used is the Newton method. To solve the scalar equation $f(x) = 0$ that method is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

This model is a root-finding method and calculates the x_{n+1} by using the x_n, the function and the derivative of the function. To find the root, the method needs to calculate multiple iterations. A method like Newton's method works the best when the compressor is in the feasible area. This method can sometimes not converge to a root. When the compressor wants to solve the equation outside the feasible area there is a chance that the Newton method will not converge.

As already mentioned, the model is important to calculate either the suction pressure, the discharge pressure or the volume flow. These variables can be calculated when two of the three are already known. But what can be done with the outcome of this model. This model can be used to calculate the settings to get the right pressure. A compressor station is a good example for this. The compressor station will use the model to calculate the right settings for every compressor. Then the flow in the pipeline and the flow in the compressors will be simulated.

The working of centrifugal compressors will be explained in chapter 2 by using [2]. Not only the working of a centrifugal compressor will be elaborated also the equations modelling it will be explained. These nonlinear equations need to be solved. When everything is explained and known, the model will be elaborated at chapter 3. The feasible area is part of this modeling process. Furthermore two different ways to handle the model such that there is always a solution will be discussed.

In chapter 4 there will be a pipe-compressor system discussed. One part of this pipe-compressor system is the pipe-compressor model. The other part is the composition of the system of equations.

In chapter 5 the method to solve a nonlinear equations will be discussed. Not only the Newton method but also the Secant method and the Quasi-Newton methods are discussed. The three methods will be compared and explained which one is the best.

Finally an example compressor and different pipelines will be used to solve the system of equations of chapter 4. There will be checked if the solution is feasible and what to do when the solution is not feasible.

Chapter 2

Centrifugal Compressor

In this chapter the theory of a centrifugal compressor will be discussed. At first the working of a centrifugal compressor will be discussed. At last a couple of formulas will be used to describe a centrifugal compressor. These formulas are needed to calculate one of the variables: P_s (suction pressure), P_d (discharge pressure) or Q_{ac} (volume flow).

2.1 Working of a centrifugal compressor

There are many kinds of compressors in this world. The most important compressor of this report is the centrifugal compressor. In figure 2.1 the schematic working of a centrifugal compressor is shown.

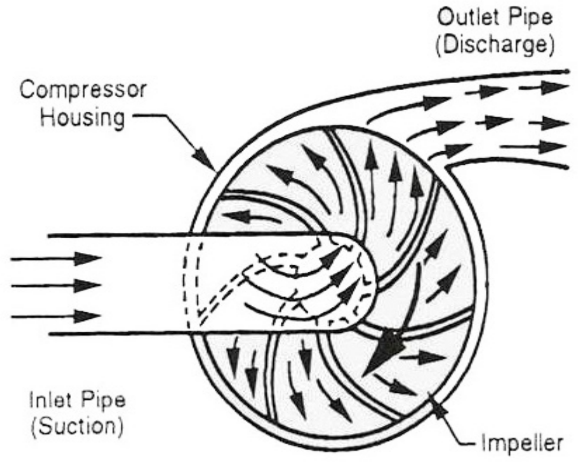


Figure 2.1: Schematic version of a centrifugal compressor (from [3])

How does this compressor work? A higher pressure can be obtained by reducing the volume of the gas. The centrifugal compressor compresses the gas as follows. At the inlet pipe, also called suction, the gas comes into the compressor. A centrifugal compressor contains a rotatably driven impeller inside the housing [2].

The gas can be compressed by the rotation of the impeller. The impeller has radial blades. Due to the rotation of the impeller the speed of the gas will be higher. The diffuser, which can be seen in figure 2.2, will partially transform kinetic energy of the gas into potential energy (pressure) by slowing down the gas. A diffuser has parallel sides without any radial blades. When the gas compressed it will leave the compressor at the discharge side. In real life a part of the pressure build-up will take place in the impeller. Roughly the compressor work like above, the whole explanation can be found in [2] and [3].

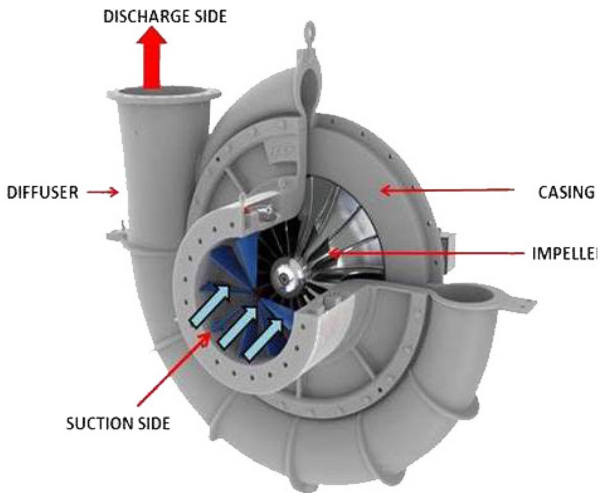


Figure 2.2: Inside a centrifugal compressor (from [3])

2.2 Centrifugal compressor equations

To describe a compressor we need multiple equations. These equations are given in the following article [1]. Some parameters are very important for the performance of the compressor. Those parameters are the isentropic head, the isentropic efficiency and the rotation speed.

2.2.1 Equations

To model the centrifugal compressor we first need the isentropic head (H) of the compressor. The H of a compressor is simply the work expressed in Joule per kilograms and is given as follows:

$$H = \frac{\gamma}{\gamma - 1} Z_s R T_s \left[\left(\frac{P_d}{P_s} \right)^{\frac{\gamma}{\gamma-1}} - 1 \right]. \tag{2.1}$$

The equations are important to calculate either P_s, P_d or Q_{ac}. The P_s and the P_d are part of equation (2.1). The only parameter which now needs to be related is Q_{ac}. The required equation is the ratio of the H and the rotation speed (N_r) and is given as follows:

$$\frac{H}{N_r^2} = b_1 + b_2 \left(\frac{Q_{ac}}{N_r} \right) + b_3 \left(\frac{Q_{ac}}{N_r} \right)^2. \tag{2.2}$$

Also the isentropic efficiency (η) is needed to describe the compressor. The isentropic efficiency is the ratio of work output for an ideal isentropic compression process to the work input to develop the required H. We calculate the η as follows:

$$\eta = b_4 + b_5 \left(\frac{Q_{ac}}{N_r} \right) + b_5 \left(\frac{Q_{ac}}{N_r} \right)^2. \tag{2.3}$$

Here b_1, b_2, b_3, b_4, b_5 and b_6 are coefficients, these coefficients have no dimension. They make equation (2.2) and (2.3) fully characterize the specific centrifugal compressor map.

Last but not least the temperature of the discharged gas (T_d) needs to be calculated. This can be only done when P_s, P_d and Q_{ac} are already known. When these are known the equation of T_d is the following:

$$T_d = T_s + \frac{T_s}{\eta}\left[\left(\frac{P_d}{P_s}\right)^{\frac{\gamma}{\gamma-1}} - 1\right]. \tag{2.4}$$

A centrifugal compressor operates under constraints. They will be discussed in the next chapter.

2.2.2 List of parameters and variables

To define the equation for the centrifugal compressor there are parameters and variables needed. These parameters and variables will be elaborated in this section. All variables, with their units, are viewed in table 2.1.

Variable	**Description**	**Unit**
H	Isentropic head	J/kg
N_r	Rotation speed	RPM
P_d	Discharge pressure of the gas	Pa
P_s	Suction pressure of the gas	Pa
Q_{ac}	Actual volume flow rate	m^3/s
T_d	Temperature discharged gas	K
η	Efficiency	-

Table 2.1: Table of all variables

Not only the variables should be defined, the parameters also need to be defined. All these parameters have a value. Not every parameter has a unit, the parameters are given in table 2.2. Example values are given as well.

Parameter	**Description**	**Value**	**Unit**
$b_1 - b_6$	Coefficients for centrifugal map	$b_1 = 1.059 \cdot 10^{-4}$ $b_2 = 6.418$ $b_3 = -6.401 \cdot 10^4$ $b_4 = 1.727 \cdot 10^{-1}$ $b_5 = 1.942 \cdot 10^4$ $b_6 = -1.505 \cdot 10^8$	-
R	Specific gas constant	$R = 437.60$	J/kmol.K
T_s	Temperature suction gas	$T_s = 300$	K
Z_s	Compressibility factor	$Z_s = 0.96$	-
γ	Isentropic coefficient	$\gamma = 1.27$	-

Table 2.2: Table of parameters and example values

2.3 Generalization

In this section the compressor model is generalized. The important difference for every compressor is that the parameters b_1 up to b_6 can vary. There can be more b_n where $n > 6$. When there are more parameters some equations can be of a degree higher than 2. Usually the degree will not be higher than three because there will be local minima and maxima. An equation of a degree higher than 3 is hard to calculate with. It is also possible that the equations will not be polynomials.

The equations which were used in section 2.2.1 are almost the same when they are generalized. Two examples of equations which can be different, are equations (2.2) and (2.3). Firstly, see what happens with the head-rotation speed ratio. When this equation is generalized its degree can be higher than 2. So the equation will be like this,

$$\frac{H}{N_r} = b_1 + b_2\left(\frac{Q_{ac}}{N_r}\right) + b_3\left(\frac{Q_{ac}}{N_r}\right)^2 + \cdots + b_n\left(\frac{Q_{ac}}{N_r}\right)^n. \tag{2.5}$$

Secondly the efficiency equation will be generalized. This equation will look the same as the head-rotation speed ratio equation. Only the parameters will start at $n+1$. The equation is as follows,

$$\eta = b_{n+1} + b_{n+2}\left(\frac{Q_{ac}}{N_r}\right) + b_{n+3}\left(\frac{Q_{ac}}{N_r}\right)^2 + \cdots + b_{n+m}\left(\frac{Q_{ac}}{N_r}\right)^m. \tag{2.6}$$

It is possible that n and m are not equal. As already mentioned, the equations are not necessarily polynomials. When they are polynomials the degree normally is not higher than 3.

Chapter 3

Constraints of a centrifugal compressor

In this chapter the compressor model will be discussed. The goal is to calculate the suction pressure (P_s), the discharge pressure (P_d) or the volume flow (Q_{ac}). These variables can be calculated using equations (2.1), (2.2) and (2.3). The algorithm uses the Newton method to calculate one of the three variables (P_s, P_d and Q_{ac}). In this chapter the constraints and their role in the compressor model will also be discussed.

3.1 Constraints

Every compressor operates in a feasible area. This feasible area is determined by constraints. In the figure below equation (2.2) is plotted.

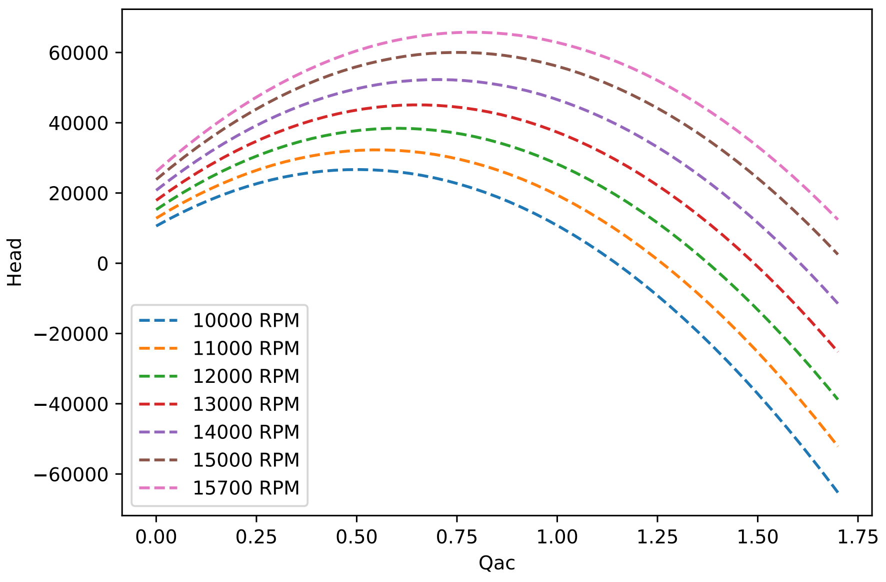


Figure 3.1: Compressor map without constraints

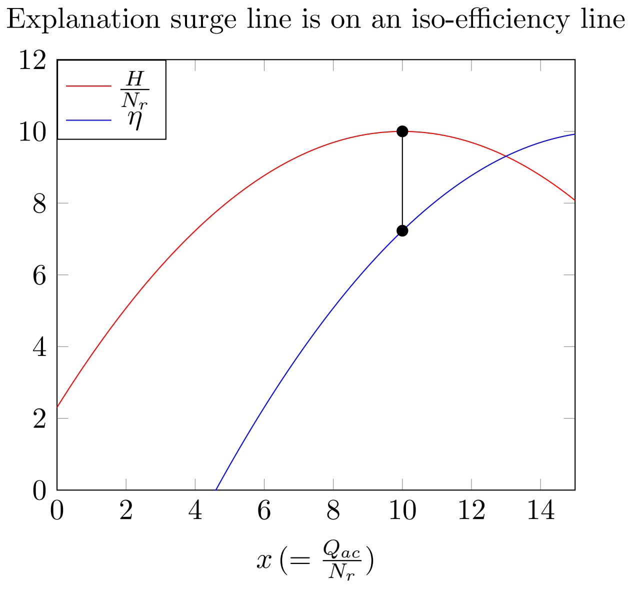
What can be seen here, is when there are no constraints, H can be negative. For a real compressor H can not be negative. Furthermore it shows that there are multiple Q_{ac} for a given H. For example when the rotation speed is 11000 RPM and a $H = 20000\,J/kg$, then $Q_{ac} \approx 0.113353$ m^3/s or $Q_{ac} \approx 0.989568\,m^3/s$. For every H there are two solutions. Therefore the algorithm can give an unrealistic solution. Only at the maximum head there is one solution.

But what constraints are necessary to make sure that the solution is always realistic? When looking at a compressor there are two states when the compressor can not operate. The two states are surging and choking and it will be elaborated in section 3.1.1 and section 3.1.2.

3.1.1 Surge line

Surge is a system-dependent phenomenon which can cause large oscillations of pressure through the compressor system [4]. Operating the system in these unstable conditions induces a dramatic drop of the performance which may cause failure of the compressor. For example when there is surge, your outlet pressure can be lower than your inlet pressure. Surge is the operation point where the maximum head (H) and the related volume flow (Q_{ac}) are reached. Which means that the compressor does not work at a lower Q_{ac}.

Because there is a possibility of failure, there normally is a margin on the surge line. It is important that the compressor is not unstable. Looking at the modeled compressor, the surge line is the maximum head at every rotation speed. Define an iso-efficiency line as a line where the efficiency is equal for different rotation speeds. All of the maximum heads, at different rotation speeds, are on an iso-efficiency line. The rate $\frac{Q_{ac}}{N_r}$ at the maximum head is the same, independent of N_r. Let $x_{\max} = \frac{Q_{ac}}{N_r}$ be the maximum of equation (2.2).



This is

$$x_{\max} = \frac{-b_2}{2 \cdot b_3} \tag{3.1}$$

The corresponding η is found by substituting $x_{\max}$ in (2.3). The iso-efficiency line will be formed by the found η at every rotation speed. The surge line is now defined by an iso-efficiency line and can be put in the compressor map (see figure 3.2).

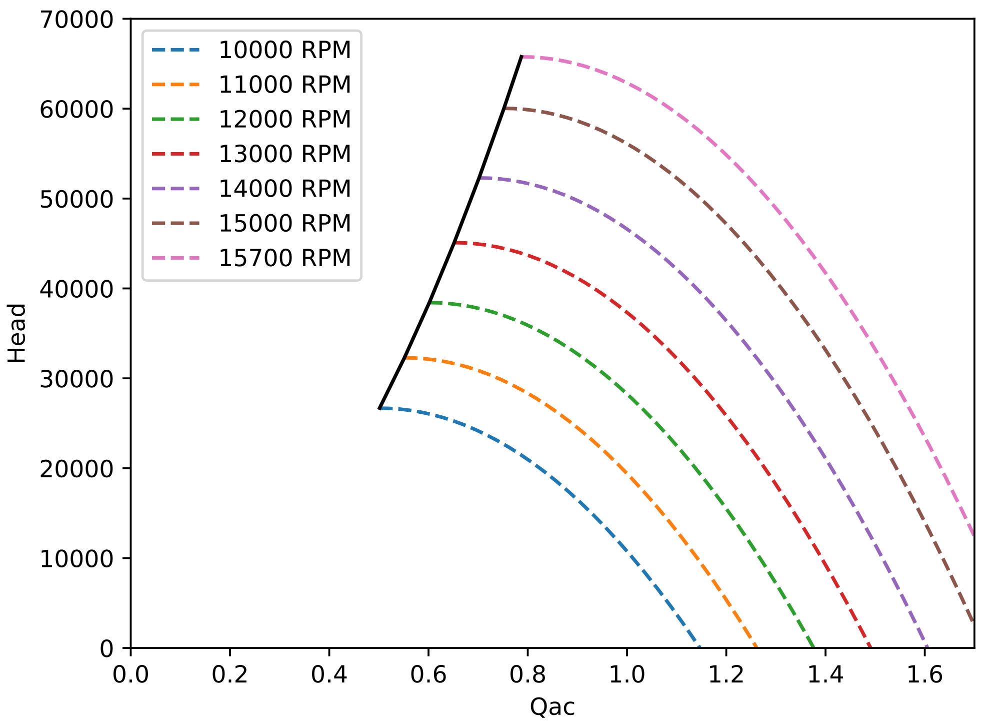


Figure 3.2: Compressor map with surge line

3.1.2 Choke line

In figure 3.2 the compressor map is not yet complete. At some point in the compressor map there is going to be choking. A choking point is also known as a stonewall point. This is the point where the volume flow increases and the discharge pressure decreases, such that the compressor does not work anymore. Moving toward the choke line, the decreasing head has less influence on inlet flow ratio because the curve slope increases. Which can be seen in figure 3.1. As the choke point is approached, changes in head will have negligible effect on inlet flow rate [5].

Now that the definition of a choke point is clear, the modeled compressor needs a choke line to work. The compressor will only work when it is operating between the surge line and the choke line. Which line is allowed to be the choke line? An iso-efficiency line is part of the answer. It is important that the compressor works as efficiently as possible and the maximum efficiency needs to be in the feasible area. The maximum efficiency of this compressor follows from equation (2.3) and is approximately 0.799.

The choke line is normally given by the manufacturer. For this model the choke line is chosen to be an iso-efficiency line at a efficiency of 0.65. The real choking point is probably at a lower iso-efficiency line than is given by the manufacturer of the compressor. The feasible area, with efficiency lines of 0.799, 0.75 and 0.70, is shown in figure 3.3. The red non-dashed line is the maximum efficiency. The code of the compressor map can be found in Appendix A.3. The minimum and maximum N_r, respectively 10000 RPM and 15700 RPM, are boundaries of the compressor map.

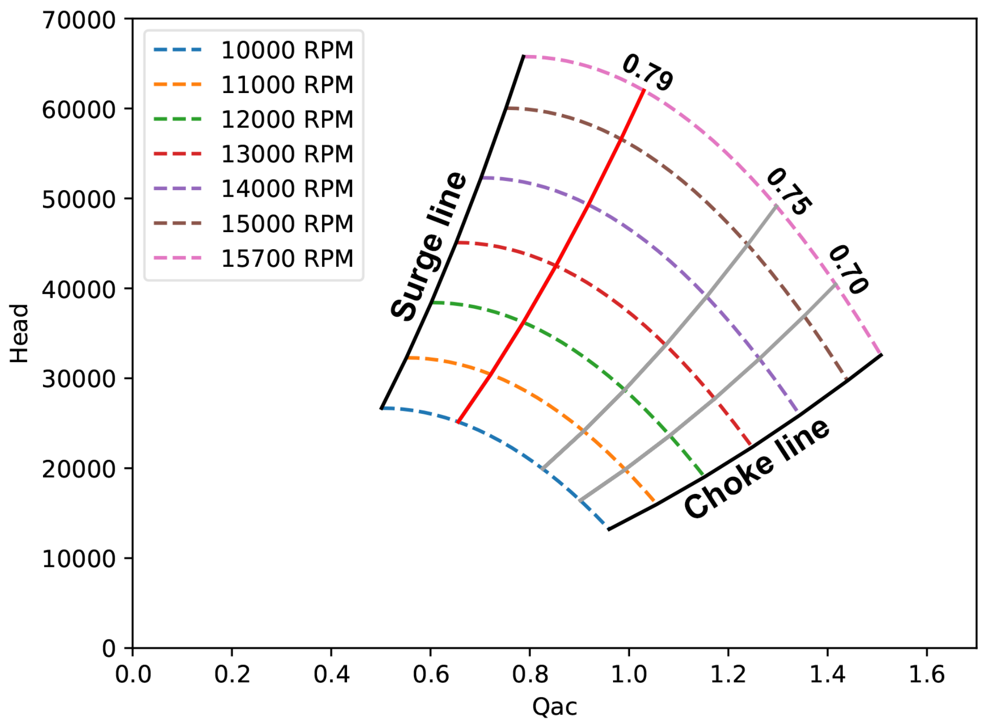


Figure 3.3: Compressor map with surge and choke line

3.2 Solutions in the feasible area

The surge and choke line are defined such that the compressor can operate. The algorithm needs to know that it can only operate between the surge and choke line and the minimum and maximum N_r. This is called the feasible area. Furthermore the algorithm needs to know what to do with a point outside the feasible area. There are two situations, one where P_s or P_d is calculated and one where Q_{ac} is calculated. This method can be used when there is a feasible solution needed such that some input data is changed. In this section we look at the feasibility of the input data. The code of this model can be found in Appendix A.4.

3.2.1 Calculation feasible P_s or P_d

When calculating P_s or P_d the given Q_{ac} needs to be in the feasible area. How does the algorithm get a feasible Q_{ac}? To check Q_{ac} is feasible the Q on the surge line (Q_{surge}) and choke line (Q_{choke}) need to be calculated. When

$$Q_{\text{surge}} \leq Q_{ac} \leq Q_{\text{choke}},$$

then the Q_{ac} is feasible. When Q_{ac} is smaller than Q_{surge}, then $Q_{ac} = Q_{\text{surge}}$. When Q_{ac} is larger than Q_{choke}, then $Q_{ac} = Q_{\text{choke}}$.

First the calculation of the Q_{surge}. The model calculates the maximu head, the surge line, and it gets the same equation as equation (3.1). Recall that $x = \frac{Q_{ac}}{N_r}$ thus,

$$Q_{\text{surge}} = \frac{-b_2 \cdot N_r}{(2 \cdot b_3)}.$$

Q_{surge} is the volume flow on the surge line and only dependent on N_r.

Now the Q_{choke} on the choke line needs to be computed. Recall from section 3.1.2 that the choke line is the efficiency of 0.65 and recall the formula for the efficiency:

$$\eta = b_4 + b_5\Big(\frac{Q_{ac}}{N_r}\Big) + b_5\Big(\frac{Q_{ac}}{N_r}\Big)^2.$$

The Q_{choke} can be solved from the fact that the efficiency is 0.65:

$$b_4 + b_5\Big(\frac{Q_{ac}}{N_r}\Big) + b_5\Big(\frac{Q_{ac}}{N_r}\Big)^2 - 0.65 = 0.$$

This can be solved analytically. The solution is the Q_{choke}.

3.2.2 Calculation feasible Q_{ac}

When calculating Q_{ac} with given P_s and P_d the technique in section 3.2.1 is not the right technique. In this case the given P_d will be checked if it is feasible. The given P_s does not have to be checked because when P_d is feasible, P_s will follow.

First of all the Q_{surge} and Q_{choke} are needed. With the Q_{surge} a minimum P_d can be computed via a nonlinear solver and the equations in section 2.2.1. On a similar way the maximum P_d can be computed, the Q_{choke} is needed for that.

Finally the given P_d can be checked. If

$$P_{d\min} \leq P_d \leq P_{d\max},$$

then the given P_d is feasible. When the P_d is bigger than $P_{d\max}$ it takes that value. Similarly when P_d is smaller than $P_{d\min}$.

3.3 Change of RPM's

In section 3.2 we saw that the input data may be not feasible. So the input data were changed, if necessary. The Q_{ac} was changed when either P_s or P_d were calculated. The P_d was changed when Q_{ac} was calculated. Now the input data is fixed. Thus Q_{ac} and P_d cannot be changed anymore. Only one variable can change, that is the rotation speed. There are multiple rotation speeds for the same Q_{ac}. The change of N_r can be used when there is a solution needed with the input data. So, if the result is not in the feasible area, the rotation speed may be changed. The Python code of this model can be found in Appendix A.5.

First of all make sure the compressor work at the maximum efficiency. Which means that the compressor has the best performance at a given head and volume flow. The maximum efficiency is calculated in section 3.1.2. Recall,

$$\frac{Q_{ac}}{N_r} = -\frac{b_5}{2 \cdot b_6}.$$

From the calculation of the maximum efficiency, the rotation speed can be calculated. So,

$$N_r = -\frac{Q_{ac} \cdot 2 \cdot b_6}{b_5}.$$

Now the N_r is the rotation speed at the maximum efficiency. It is important to note that the maximum efficiency is always in the feasible area. In figure 3.3 the line of highest efficiency is marked red.

There is one situation where it is hard to generate a solution for P_s. This happens when the pressure ratio is too high. The pressure ratio $\frac{P_d}{P_s}$ can be found in equation (2.2). For example, if the suction pressure is too low and the discharge pressure too high, the ratio would be too high. A ratio like this is not likely to happen.

Chapter 4

Pipeline simulations

In chapter 3 the compressor model was discussed. This compressor works fine in the feasible area. The next step is to connect a pipeline in front of the compressor and behind the compressor. This creates a small pipe-compressor system. The pressure at the begin of the first pipeline is fixed as well as the pressure at the end of the second pipeline. To calculate the gas flow inside a pipeline the Weymouth pressure drop equation will be used.

4.1 Pipe-compressor model

To make a pipe-compressor system, the model needs to be clarified. It starts with a pipeline where there is a start pressure P_1 and a mass flow rate in the pipeline $\dot{m}_s$. At the end of the pipeline the pressure will be lower than P_1. The pressure drop can be calculated via the Weymouth equation. This equation is as follows

$$\dot{m} = \text{sign}(P_1^2 - P_2^2) c_w \sqrt{\frac{S|P_1^2 - P_2^2| D^{5.3333}}{TZL}}. \tag{4.1}$$

The parameters are defined in table 4.1. Example values are given as well.

Parameter	Description	Value	Unit
c_w	Pipe constant	$c_w = 0.47857$	-
D	Pipe diameter	$D = 0.3$	m
L	Pipe length	$L = 10^5$	m
S	Specific gravity of the gas	$S = 19/29$	-
T	Average temperature	$T = 300$	K
Z	Compressibility factor	$Z = 0.96$	-

Table 4.1: Table of pressure-drop equation parameters and example data

Before gas is going into the compressor, the pressure at that moment is equal to the suction pressure (P_s). Then the pressure of the gas after compression is the discharge pressure (P_d). The mass flow $\dot{m}_s$ before the compression is not the same mass flow after compression ($\dot{m}_d$). This is because the compressor need some 'gas fuel' to operate. This fuel can be obtained by using a small amount of gas of the mass flow $\dot{m}_s$ and is denoted as $\dot{m}_f$. At last there is another pipeline where P_2 is fixed and the rest can be calculated via the Weymouth pressure drop equation (4.1). The model is schematically shown in figure 4.1.

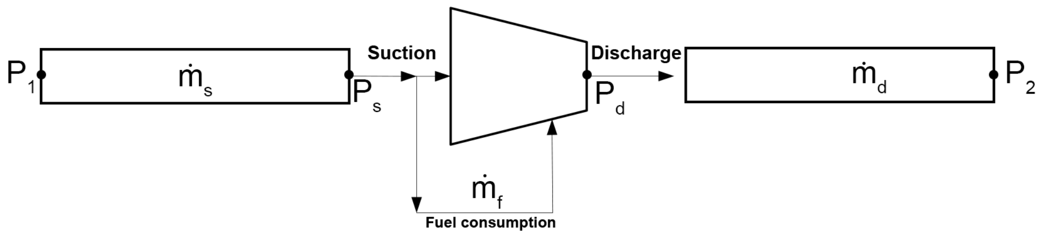


Figure 4.1: Pipeline simulation schematic

4.1.1 System of equations

To solve the pipe-compressor system a system of equations needs to be collected. Not every equation is already given. Equations (2.1), (2.2), (2.3), (2.4) and (4.1) are already given and these are needed to compile a model for the pipe-compressor system. There are three equations which are not given yet. The first one is to calculate the power of the compressor.

$$\text{Power} = \frac{H \cdot \dot{m}_d}{\eta}. \tag{4.2}$$

In figure 4.1 is shown that the fuel consumption is important. An equation for that mass flow $\dot{m}_f$ is not yet known. The equation for the fuel consumption is as follows,

$$\dot{m}_f = \frac{\text{Power}}{\text{LHV} \cdot \eta_{\text{turbine}}}. \tag{4.3}$$

For example, where LHV $= 47 \cdot 10^6$ and $\eta_{\text{turbine}} = 0.8$. It is important that the total mass flow is not changed throughout the system. The $\dot{m}_s$ is not equal to $\dot{m}_d$ because some part of the $\dot{m}_s$ becomes fuel. The last equation is,

$$\dot{m}_s = \dot{m}_f + \dot{m}_d. \tag{4.4}$$

Every equation that is needed to formulate the model is known. What are the unknown variables? The pressure at the begin (P_1) and the pressure at the end (P_2) are fixed. Furthermore the rotation speed (N_r) and volume flow (Q_{ac}) are important to calculate because the compressor will work on a certain iso-efficiency line. So the ratio of volume mass and RPM needs to be equal to that iso-efficiency line. So, the variables are $\dot{m}_d$, $\dot{m}_f$, $\dot{m}_s$, N_r, P_d, P_s and Q_{ac}

All the variables and equations are known and the system can be formed. The system of equation for this model is,

$$\begin{cases} f_1 = \dot{m}_s - \text{sign}(P_1^2 - P_s^2)\, c_w \sqrt{\frac{S|P_1^2 - P_s^2| D^{5.3333}}{T_s Z L}} \\ f_2 = \dot{m}_s - \dot{m}_f - \dot{m}_d \\ f_3 = \frac{H}{N_r^2} - \left(b_1 + b_2 \left(\frac{Q_{ac}}{N_r} \right) + b_3 \left(\frac{Q_{ac}}{N_r} \right)^2 \right) \\ f_4 = \eta_{\text{line}} - \left(b_4 + b_5 \left(\frac{Q_{ac}}{N_r} \right) + b_5 \left(\frac{Q_{ac}}{N_r} \right)^2 \right) \\ f_5 = Q_{ac} - \frac{\dot{m}_d}{\rho} \\ f_6 = \dot{m}_f - \frac{\text{Power}}{\text{LHV}\, \eta_{\text{turbine}}} \\ f_7 = \dot{m}_d - \text{sign}(P_d^2 - P_2^2)\, c_w \sqrt{\frac{S|P_d^2 - P_2^2| D^{5.3333}}{T_d Z L}} \end{cases} \tag{4.5}$$

Having formed this system, the H (equation 2.1), η (equation 2.3), T_d (equation 2.4) and the Power (equation 4.2) can be calculated using the equations which are already known. The η_{line} is the efficiency at a certain iso-efficiency line. This needs to be higher than 0.65 and less or equal to the maximum efficiency.

4.2 Solver for system of equations

The system of equations is determined. But which method is the best to solve this problem? Their are multiple ways to solve a system of equations numerically. Existing Python functions will be used for this. The two functions that are used to solve this specific problem are Newton and fsolve in the scipy.optimize package.

The Newton method in Python is similar to the method explained in section 5.4. The convergence of the system is quadratic, if it is possible to converge. Newton's method is known of its fast convergence but is also known that it will not always converge. In comparison to fsolve, it is harder to converge a system of equations for Newton's method.

There are multiple reasons why Newton may not converge when it is solving a system of equations. First of all, the initial values can be taken wrong. When these values are too far away from the solution Newton's method may not converge. It could get further away from the actual solution and diverge or maybe iterate without converging.

Fsolve is a function in the Python scipy.optimize package. This root-finding algorithm is used to find the solutions of a system of nonlinear equations. This algorithm has a lot of great features to find the solution. Fsolve uses, among other things, Powell's method, also known as Powell's conjugate direction method [6]. This method finds the local minimum of a function and works roughly like this. The derivative of the function does not need to be calculated and no derivatives are taken. This method minimizes the function by using bidirectional search along each vector. More information about fsolve can be found in the source code of fsolve.

Fsolve works most of the time better than Newton's method. Newton's method can diverge very easily when the initial values are taken wrong. Fsolve has features like bidirectional search to make sure the next iteration is closer to the solution. In conclusion, to solve the pipe-compressor system in Python fsolve is a better tool to use.

4.3 Feasible area

The solver is determined but is the solution feasible? Recall that the feasible area of the compressor is as in figure 3.3. Where the minimum rotation speed is 10000 RPM and the maximum rotation speed is 15700 RPM. When solving the system of equations (4.5) the Q_{ac} can have two solutions when the iso-efficiency line is not equal to the maximum efficiency. But when the efficiency is equal to the maximum efficiency, the system can only have one Q_{ac} as a solution. The maximum efficiency is calculated, where $\eta_{\max} \approx 0.799172$. So, the

$$f_4 = \eta_{\text{line}} - \Big(b_4 + b_5\Big(\frac{Q_{ac}}{N_r}\Big) + b_5\Big(\frac{Q_{ac}}{N_r}\Big)^2\Big),$$

can be changed into

$$f_4 = 0.799172 - \Big(b_4 + b_5\Big(\frac{Q_{ac}}{N_r}\Big) + b_5\Big(\frac{Q_{ac}}{N_r}\Big)^2\Big).$$

The system of equations has only one solution at the maximum efficiency. This does not mean that the solution is also in the feasible area. The found rotation speed can be lower than 10000 RPM and can be higher than 15700 RPM. If the rotation speed is between 10000 RPM and 15700 RPM the solution is feasible right away.

4.3.1 Rotation speed outside feasible area

The found rotation speed can be higher than feasible or lower than feasible. What happens with the solution when it is outside the feasible area? When the rotation speed is higher then feasible, the rotation speed will be reduced until it reaches 15700 RPM. When the rotation speed is lower than feasible, the rotation speed will be increased until it reaches 10000 RPM. The user can choose a rotation speed somewhere between 10000 RPM and 15700 RPM.

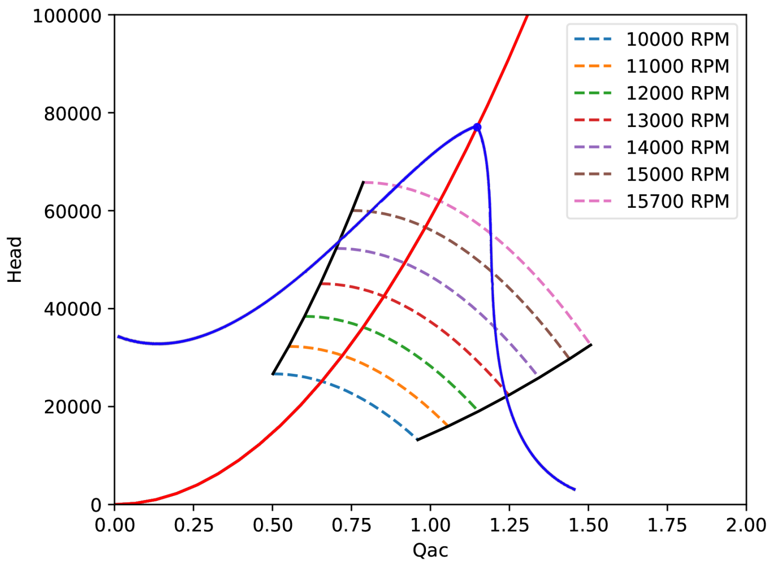


Figure 4.2: Line of solutions from a point with too high RPM

First, lets see what happens when the rotation speed is reduced. When it is reduced, the solution is not the most efficient one anymore. That is because the $\frac{Q_{ac}}{N_r}$ ratio is different, N_r is smaller. The Q_{ac} is also different and so the ratio is different. Q_{ac} can have two different solutions as given in the figure 4.2. There are two lines of solutions. The one left of the iso-efficiency line with the maximum efficiency and one right of the iso-efficiency line with the maximum efficiency. When reducing the rotation speed, the Q_{ac} can be either feasible or not feasible. The solution of the system needs to be in the feasible area. The solver should find the Q_{ac} on the right line.

When looking at figure 4.3 the Q_{ac} can have two different solutions. With one line that goes into the feasible area and one goes into something which is physically not possible. In this case the solver also wants to find the solution on the right line. When the solver chooses that line the solution is a feasible solution.

For both kind of problems there will be a new system of equations defined. The rotation speed

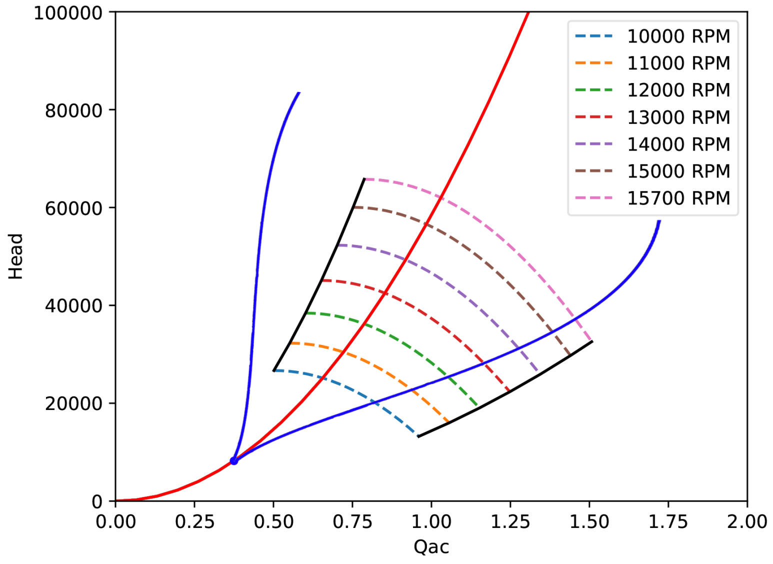


Figure 4.3: Line of solutions from a point with too low RPM

(N_r) is now fixed. There are six variables and six equations to solve. The new system of equations is as follows,

$$\begin{cases} f_1 = \dot{m}_s - \text{sign}(P_1^2 - P_s^2)\, c_w \sqrt{\frac{S|P_1^2 - P_s^2| D^{5.3333}}{T_s Z L}} \\ f_2 = \dot{m}_s - \dot{m}_f - \dot{m}_d \\ f_3 = \frac{H}{N_r^2} - \left(b_1 + b_2 \left(\frac{Q_{ac}}{N_r} \right) + b_3 \left(\frac{Q_{ac}}{N_r} \right)^2 \right) \\ f_4 = Q_{ac} - \frac{\dot{m}_d}{\rho} \\ f_5 = \dot{m}_f - \frac{\text{Power}}{\text{LHV}\, \eta_{\text{turbine}}} \\ f_6 = \dot{m}_d - \text{sign}(P_d^2 - P_2^2)\, c_w \sqrt{\frac{S|P_d^2 - P_2^2| D^{5.3333}}{T_d Z L}} \end{cases} \tag{4.6}$$

Where the Power is still defined as in equation (4.2). This system is solving $\dot{m}_s$, P_s, P_d, $\dot{m}_f$, $\dot{m}_d$ and Q_{ac}. It is already mentioned, but Q_{ac} can have two possible solutions at the fixed rotation speed. The solution needed is the one that lies on the right line in figure 4.2 or figure 4.3. To check if this is true, the program checks if the Q_{ac} is in the feasible area. When the given Q_{ac} is on the left line, the program determines automatically the Q_{ac} on the right line. When the Q_{ac} is the right one, the system will stop. The right solution is given.

With another Q_{ac} all the other variables are different. So the system of equations needs to be calculated again. The only difference with the system of equations (4.6) is that the Q_{ac} and the N_r are both fixed. So, the new system has five unknown variables with five equations. This system looks as follows,

$$\begin{cases} f_1 = \dot{m}_s - \text{sign}(P_1^2 - P_s^2)\, c_w \sqrt{\frac{S|P_1^2 - P_s^2| D^{5.3333}}{T_s Z L}} \\ f_2 = \dot{m}_s - \dot{m}_f - \dot{m}_d \\ f_3 = Q_{ac} - \frac{\dot{m}_d}{\rho} \\ f_4 = \dot{m}_f - \frac{\text{Power}}{\text{LHV}\, \eta_{\text{turbine}}} \\ f_5 = \dot{m}_d - \text{sign}(P_d^2 - P_2^2)\, c_w \sqrt{\frac{S|P_d^2 - P_2^2| D^{5.3333}}{T_d Z L}} \end{cases} \tag{4.7}$$

Finally, the right solution is given and is feasible. The Python code can be found in Appendix A.6.

4.3.2 Other cases

It might be possible the system has not the correct solution right away. When it does have a solution it does not mean it is feasible. There are multiple cases that come after and before calculating the solution described in 4.3.1. First of all what to do when there is no solution given.

It is not possible that the system of equations (4.5) has no solutions. There are always solutions possible. That comes from the fact that the maximum efficiency is calculated for the ratio Q_{ac} over N_r. This iso-efficiency line is always in the feasible area, so there must be a solution of the compressor. When there is a solution of the compressor, the pipeline equations also has a solution. But what to do when the algorithm does not give a solution or does not converge?

The problem then is that the initial values are not defined properly. This system of equations is a very unstable system. That means that for a small change in the initial values, the whole system may not converge anymore. The two values that are the most fragile are P_s and P_d. When these values are too high or too low, the whole system does not work anymore. This is because the ΔP needs to be high enough. When this is too low, the compressor can not compress it well. Therefore the solution is not feasible and thus it does not work. If the system is not giving an answer, change the initial value of P_s or P_d. When it is still not converging change the other initial values.

Another case is that the lines of solutions are not in the feasible area at all. An example is seen in figure 4.4. In this figure also the right line needs to be calculated when the first solution is underneath or above the feasible rotation speed. The left blue line is still not physically correct. What is going to happen with the right blue line?

The right blue line in figure 4.4 is outside the feasible area. Recall that choke is the point where the volume flow increases and the discharge pressure decreases, such that the compressor does not work anymore. The change that choke occurs outside the feasible area is very high. The choke line is set on the iso-efficiency line of 0.65, which is overestimated (see section 3.1.2).

The same is obtained when the rotation speed is above 15700 RPM. In that case the line of solutions can also be outside the feasible area. It seems that this kind of problems have no solutions in the feasible area. The only solution is that the chosen pipeline is not suitable with the chosen compressor or vice versa. When there is another pipeline or compressor, which is suitable for both, the system of equations will give a feasible solution.

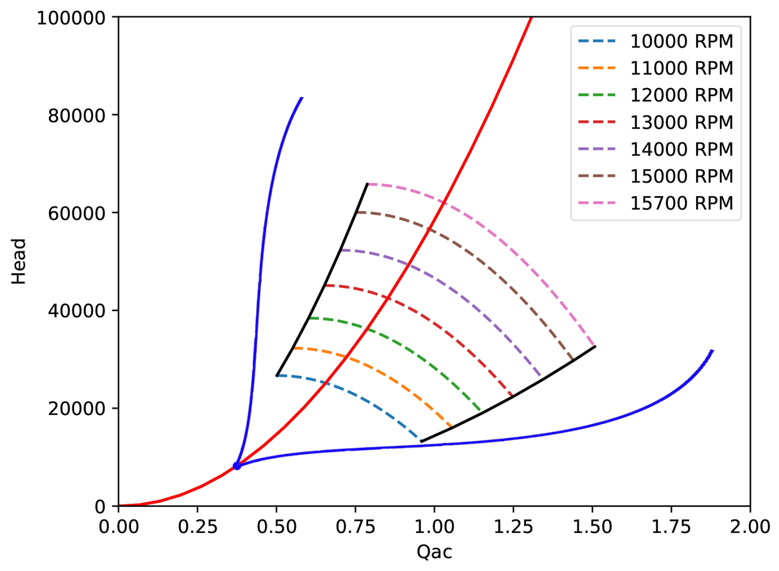


Figure 4.4: Line of solutions not in feasible area

Chapter 5

Solving the nonlinear equations

In this chapter one specific numerical method will be discussed. The most important numerical method to solve the equations in chapter 2 is the Newton method. The Newton method can be used for scalar algebraic equations or for system of algebraic equations. First we look at a scalar algebraic equation.

5.1 Newton's method

To solve a nonlinear equation $f(x) = 0$, the Newton method is one of the best-known numerical methods. The followed derivation is from [7]. This numerical method can be explained by using a Taylor polynomial. First suppose $f \in C^2[a,b]$ and let $\widetilde{x} \in [a,b]$ be an approximation of the root x such that $f'(\widetilde{x}) \neq 0$. Suppose that $|x - \widetilde{x}|$ is small, then the Taylor polynomial about $\widetilde{x}$ is as follows,

$$f(x) = f(\widetilde{x}) + (x - \widetilde{x})f'(\widetilde{x}) + \frac{(x-\widetilde{x})^2}{2} f''(\xi(x)),$$

where $\xi(x)$ is between x and $\widetilde{x}$. Because $f(x) = 0$ the equation above can be written as,

$$0 = f(\widetilde{x}) + (x - \widetilde{x})f'(\widetilde{x}) + \frac{(x-\widetilde{x})^2}{2} f''(\xi(x)).$$

Then knowing that $|x - \widetilde{x}|$ is very small, $|(x - \widetilde{x})|^2$ can be neglected. Then following equation appears,

$$0 \approx f(\widetilde{x}) + (x - \widetilde{x})f'(\widetilde{x}).$$

Rewrite this formula into,

$$x \approx \widetilde{x} - \frac{f(\widetilde{x})}{f'(\widetilde{x})}.$$

Now let x_0 be a approximation and this generates the sequence of $\{x_n\}$. The Newton method is,

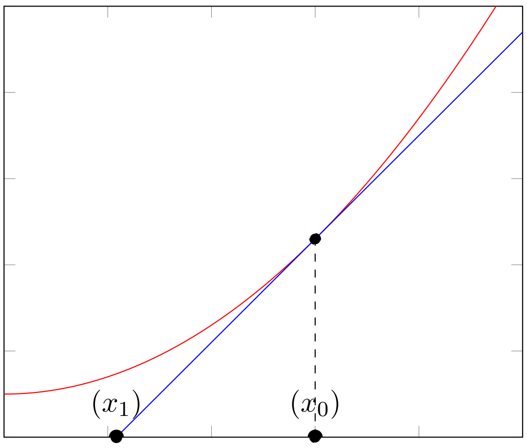
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Remark that the right part of the equation is the tangent at the point x_n. A Python code for the Newton method can be found in Appendix A.1.

5.2 Geometric explanation Newton's method

The Newton method is now explained, but how does it work? Assume $f(x) = 0$ has one root. The Newton method starts with an approximation x_0 and finds the tangent at x_0. Then the point (x_1) where the tangent hits the x-axis becomes the new point. This continues until $f(x_n) \approx 0$ and then the method stops. Then x_n is the root of the function $f(x)$. To get a better understanding of the above, the figure below shows how the x_n is calculated.

Explanation Newton's method



So, when the tangent line at point x_0 is found, the x_1 can be found by the intersection of the tangent line and the x-axis. By repeating this the root will eventually be found. It converges if x_0 is in the radius of convergence.

5.3 Convergence of Newton's method

To get a solution from a nonlinear solver the method needs to converge the solution. So each nonlinear solver generates a sequence $\{x_n\} = x_0, x_1, x_2, \dots$ which should converge to x. Thus $\lim_{n\to\infty} x_n = x$. We assume the method does converge. If there exist a positive constants λ and α satisfying

$$\lim_{n\to\infty} \frac{|x - x_{n+1}|}{|x - x_n|^\alpha} = \lambda, \tag{5.1}$$

then $\{x_n\}$ converges to x with order α and asymptotic constant λ.

The Newton methods converges with an order of 2. This can be shown as follows. First make the observation that the Newton method can be computed by Taylor expansion:

$$0 = f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n) \text{ for } \xi_n \text{ between } x_n \text{ and } x. \tag{5.2}$$

Newton method is defined such that

$$0 = f(x_n) + (x_{n+1} - x_n)f'(x_n). \tag{5.3}$$

When subtracting equation (5.3) from (5.2) yields

$$(x - x_{n+1})f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n) = 0,$$

such that,

$$\frac{|x - x_{n+1}|}{|x - x_n|^2} = \left|\frac{f''(\xi_n)}{2f'(x_n)}\right|.$$

Then from equation (5.1) it can be seen that the Newton method converges with order 2, which means it converges quadratically. Thus $\alpha = 2$ and

$$\lambda = \lim_{n\to\infty} \left|\frac{f''(\xi_n)}{2f'(x_n)}\right| = \left|\frac{f''(x)}{2f'(x)}\right|.$$

The followed derivation is from [7].

5.4 Newton's method for systems

The Newton method can be generalized to solve a system of nonlinear equations. Let there be a system of n nonlinear equations in n variables given by

$$\begin{aligned} f_1(x_1, x_2, x_3, \ldots, x_m) &= 0 \\ f_2(x_1, x_2, x_3, \ldots, x_m) &= 0 \\ f_3(x_1, x_2, x_3, \ldots, x_m) &= 0 \\ &\vdots \\ f_m(x_1, x_2, x_3, \ldots, x_m) &= 0. \end{aligned}$$

Equivalent to the the above is:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2, x_3, \ldots, x_m) \\ f_2(x_1, x_2, x_3, \ldots, x_m) \\ \vdots \\ f_m(x_1, x_2, x_3, \ldots, x_m) \end{pmatrix} = \mathbf{0}.$$

Use the Taylor multivariate expansion to linearize the function f about $\mathbf{x}^{(n-1)}$:

$$\begin{aligned} f_1(\mathbf{x}) &\approx f_1(\mathbf{x}^{(n-1)}) + \frac{\partial f_1}{\partial x_1}(\mathbf{x}^{(n-1)})(x_1 - x_1^{(n-1)}) + \cdots + \frac{\partial f_1}{\partial x_m}(\mathbf{x}^{(n-1)})(x_m - x_m^{(n-1)}), \\ &\vdots \\ f_m(\mathbf{x}) &\approx f_m(\mathbf{x}^{(n-1)}) + \frac{\partial f_m}{\partial x_1}(\mathbf{x}^{(n-1)})(x_1 - x_1^{(n-1)}) + \cdots + \frac{\partial f_m}{\partial x_m}(\mathbf{x}^{(n-1)})(x_m - x_m^{(n-1)}). \end{aligned}$$

Now the Jacobian matrix has to be defined. The Jacobian matrix of $\mathbf{f(x)}$ is as follows:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_m}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_m}(\mathbf{x}) \end{pmatrix}.$$

The Jacobian is determined and the linearization can be written more compactly.

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}^{(n-1)}) + \mathbf{J}(\mathbf{x}^{(n-1)})(\mathbf{x}^n - \mathbf{x}^{(n-1)}).$$

The next iteration $\mathbf{x}^{(n)}$ is obtained when putting the $\mathbf{f(x)}$ to zero. This also happens when the system has just one equation (see section 5.1).

$$\mathbf{f}(\mathbf{x}^{(n-1)}) + \mathbf{J}(\mathbf{x}^{(n-1)})(\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}) = 0,$$

this can be rewritten as

$$\mathbf{J}(\mathbf{x}^{(n-1)})(\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}) = -\mathbf{f}(\mathbf{x}^{(n-1)}).$$

So the new iteration is,

$$\mathbf{x}^{(n)} = \mathbf{x}^{(n-1)} - \mathbf{J}^{-1}(\mathbf{x}^{(n-1)})\mathbf{f}(\mathbf{x}^{(n-1)})$$

The Newton method for a system of nonlinear equations is defined. What we obtain is that this method is the same as for the scalar equation. When there is a scalar equation the Jacobian is just the derivative of $f(x)$. Then the calculation of the new iteration is the same as in section 5.1. A Python code for the Newton method for systems can be found in Appendix A.2.

5.5 Other nonlinear solvers

Besides the method of Newton, there are many more methods that can be used. Many well-known methods besides Newton's Method are derived from the Newton method, such as the Secant method and Quasi-Newton methods. These methods will be explained briefly and compared to Newton's method.

5.5.1 Secant method

The Secant method is a root-finding numerical method for scalar equations. This method uses the roots of the secant lines. A secant line is a line that intersects at least two (distinct) points of the curve. It can be seen as a finite-difference approximation of the Newton method. Where the Newton method uses the derivative of a function, the Secant method uses an approximation of the derivative. The approximation for $f'(x_n)$ in the Secant method is given by

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

What is the difference between this method and the Newton method besides the approximation of $f'(x_n)$? To answer this question the order of convergence needs to be calculated. From [8] the following is obtained.

$$\frac{|x - x_{n+1}|}{|x - x_n||x - x_{n-1}|} = \left|\frac{f''(\xi_n)}{2f'(x_n)}\right|,$$

this differs from the Newton method. Let $C = \left|\frac{f''(\xi_n)}{2f'(x_n)}\right|$ and $M = \frac{f''(\xi_n)}{2f'(x_n)}$. Assume from the Newton method that $|x - x_{n+1}| \approx C|x - x_n|^p$. Then the next result follows,

$$\begin{aligned} C|x - x_n|^p &= |M||x - x_n||x - x_{n-1}| \\ |x - x_n|^{p-1} &= \frac{|M|}{C}|x - x_{n-1}| \\ |x - x_n| &= \left(\frac{|M|}{C}\right)^{\frac{1}{p-1}} |x - x_{n-1}|^{\frac{1}{p-1}}. \end{aligned}$$

Now the equation is in the form of $|x - x_{n+1}| \approx C|x - x_n|^p$ and p can be calculated with $p = \frac{1}{p-1}$. After a calculation $p \approx 1.618$. The order of convergence of the Secant method is lower than 2 and thus not quadratic. Therefore the Secant method converges slower to a point than the Newton method. Taken together the reason why the Newton method is more preferable is because the order of convergence is higher.

5.5.2 Quasi-Newton methods

Quasi-Newton methods are a collection of root-finding or local minima and maxima finding numerical methods. These methods also are for systems. They can be used if the Jacobian or Hessian of a function is unavailable or if it is too expensive to calculate them. There are many Quasi-Newton methods. Some well-known Quasi-Newton methods are: BFGS and Broyden. Broyden can be seen as a generalization of the secant method to systems. In general, Quasi-Newton methods are an approximation of the Jacobian (or the Hessian in case of optimization). The methods differ in how they do this. The Quasi-Newton method is defined as,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

where $f'(x_n)$ is approximated by,

$$f'(x_n) = \frac{f(x_n + h) - f(x_n)}{h}.$$

From [9] is it obtained that Quasi-Newton methods also converge superlinearly. But why is it better to use the Newton method with the model of a compressor? The Newton method is chosen because the Jacobian of the model is available as will be clear later.

Chapter 6

Examples & Results

In this chapter the results and applications of the model will be discussed. First there will be elaborated what can and can not be done with this model. Also the difference between what the compressor does when it is operating in a network and when it is operating alone. Next the working of a compressor with and without a network is discussed, the potential problems with the solver are discussed. After that the numerical behaviour of the algorithm for two different examples will be discussed and there will be a parameter study for two different examples. At the end the two examples and their results will be shown.

6.1 The model

Model (4.5) has seven variables. The seven variables are $\dot{m}_d$, $\dot{m}_f$, $\dot{m}_s$, N_r, P_d, P_s and Q_{ac}. When calculating these variables it does not automatically mean that the solution is feasible. Recall that a feasible solution lies in the compressor map (see figure 3.3).

With the help of the model the algorithm can be formed, see Appendix A.6. This algorithm can calculate the seven unknown variables. What can this algorithm do and what can this algorithm not do? First of all, the algorithm can calculate a solution. This solution is mathematically correct, but that does not mean it is physically correct. The model is set up in a way that the algorithm can calculate a feasible solution, even if it was not feasible at first. It does not mean that the new solution is feasible.

Looking at the model, there are multiple solutions possible if the efficiency is not equal to the maximal efficiency. When the efficiency is equal to the maximal efficiency then there is only one solution possible. The algorithm first checks if the solution, with seven unknown variables, is feasible. When this is not feasible the algorithm tries to make it feasible as described in chapter 4. Figure 6.1 is a flowchart of what the algorithm does with an unfeasible solution.

As already mentioned the solution after all the modifications can still not be feasible. This is not because the model is not correct, but this is because some of the parameters are not well defined. It is possible that the chosen compressor and the chosen pipelines are not compatible with each other. The compressor can be too small or too big for the chosen pipeline. When this happens the solution will not lie in the compressor map of that (incompatible) compressor.

Next the question is what is not possible for the model and the algorithm to do. It is not possible for the model to determine right away if the pipelines are compatible with the compressor.

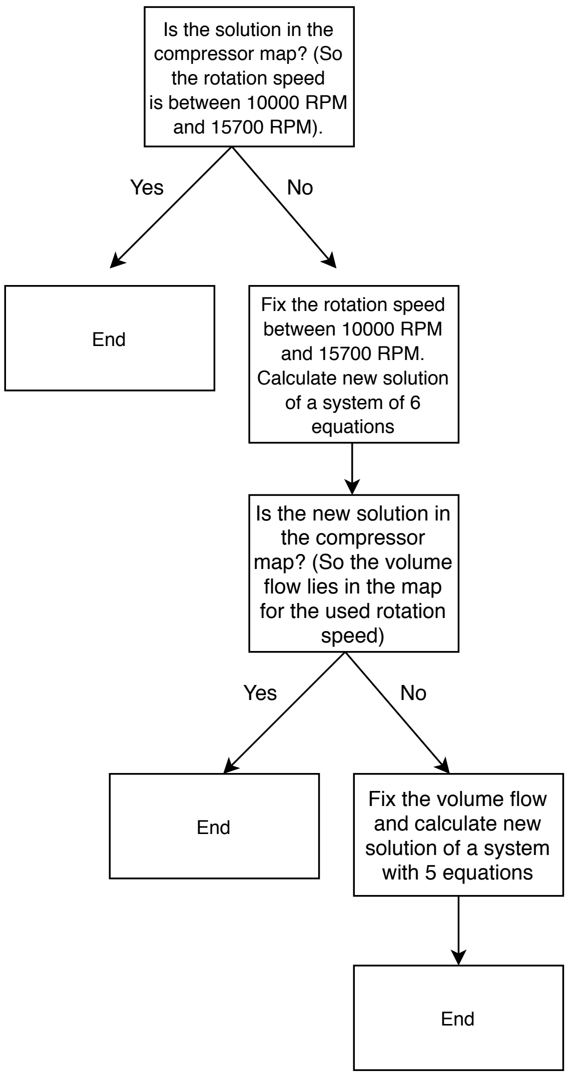

Is the solution in the compressor map? (So the rotation speed is between 10000 RPM and 15700 RPM).
Yes
No
End
Fix the rotation speed between 10000 RPM and 15700 RPM. Calculate new solution of a system of 6 equations
Is the new solution in the compressor map? (So the volume flow lies in the map for the used rotation speed)
Yes
No
End
Fix the volume flow and calculate new solution of a system with 5 equations
End

Figure 6.1: Flowchart feasible solution

It can be observed by testing the algorithm. The algorithm has more problems then only the compatibility of the compressor and pipelines. For example, the algorithm can not get a solution for some values of the inlet pressure P_1 and the outlet pressure P_2. The problem is that the initial values for the solver are hard to determine. A standard set of initial values such that the algorithm converge every time is hard or even impossible to find.

Thus when the model and the algorithm are set up to calculate the seven unknown variables. First the algorithm determines if the solution is physically feasible and uses the flowchart in figure 6.1 to make the solution as feasible as possible. When the algorithm can not converge to a solution, the initial values are taken incorrect. Another option is that the compressor and pipelines do not belong to each other.

6.2 Compressor with and without a network

In this section the difference between a compressor alone and a compressor in a network will be discussed. What happens with the compressor when it is operating alone and is this comparable to a compressor operating in a network.

There are two kind of problems, a feasible solution and a solution that is not feasible. First we will look at the problem when the solution is feasible. If the solution is feasible the efficiency will be the maximum efficiency or lower, but is still within the compressor map. At the maximum efficiency the compressor alone works the same as when it is in a network. When the solution is feasible but not at the maximum efficiency, the compressor alone works the same as when it is in a network. This efficiency is also inside the feasible area.

Then if the solution is not feasible, does the compressor still work the same alone as in a network? Unfortunately this is not the same. Because the N_r and Q_{ac} are fixed there are two equations less than the original system of equations. The two deleted equations are the efficiency equation and the head equation. The head equation is an important equation when calculating the pressure ratio. Deleting this equation gives the discharge and suction pressure a little freedom. The N_r and Q_{ac} and the P_d and P_s are not linked anymore. Thus, the compressor will work differently in a system than operating alone when the solution is not feasible, the reason why will be discussed further in this section. It is possible to remove two other equations but then the variables associated with them are having too much freedom. The best option is to remove the equations for the head and the efficiency.

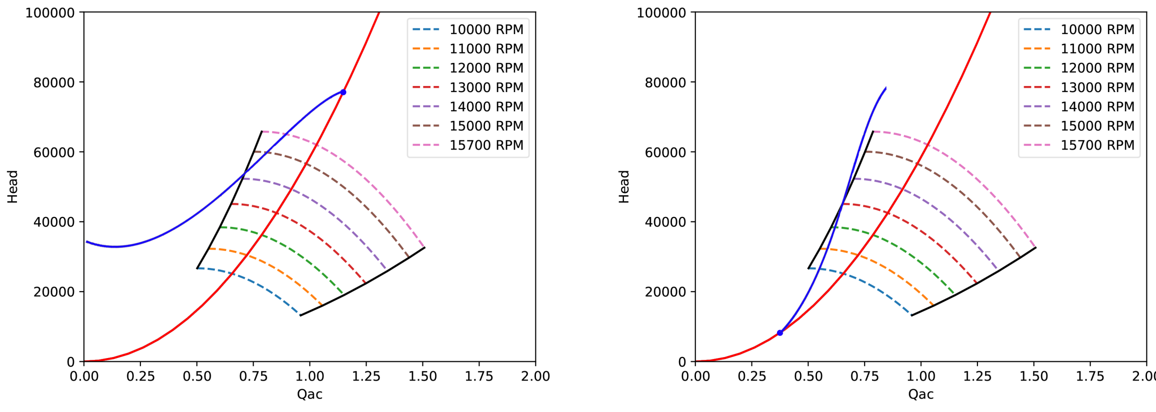
In section 4.3.1 there is assumed that there are two solutions possible when the efficiency is not equal to the maximum efficiency. Is this assumption still true? Let us find out. When there are two solutions possible then the residual of the system of equations (4.6) for both solutions need to be approximately zero. The first solution is when only N_r is fixed and the second solution is when the same N_r and the related Q_{ac} are fixed. Fill the first solution in to the system of equations (4.6) and the residual will be approximately zero. The second solution can also be filled in to the system of equations (4.6). The result is that the residual of five of the six equations are approximately zero, but one residual is not approximately zero. This means that the second solution is not a solution of the system of equations.

The assumption is, if only N_r is fixed, there is just one solution for the system of equations (4.6). For every unfeasible solution of system of equations (4.5), a N_r can be fixed such that it is feasible. Then the new solution of the system of equations (4.6) only has one solution, because not every residual of the 'second' solution is not approximately zero. So the assumption is true.

The only question left is, is the solution of the system of equations (4.6) in the compressor map or outside the compressor map (which can be seen in figure 4.3 or 4.2)? By trying multiple examples an assumption can be made about where the solution of the system of equations (4.6) lies in the compressor map. The assumption is that the solution of the system always lies left of the line of the maximum efficiency. The solution is feasible when it is between the surge line and the line of the maximum efficiency, but is unfeasible when the solution lies left of the surge line. In figure 6.2 the possible line of solution is given for a point with a very high rotation speed and for a point with a very low rotation speed.

It is hard to verify if every solution of the system of equations (4.6) lies left of the maximum efficiency. By trying a lot of different possibilities it looks like the assumption is true. In figure 6.3 the convergence of Q_{ac} is shown, when the N_r is fixed. Form this figure it is clear that the Q_{ac} is staying on 0.96 for a while. After that it drops really hard. When this happens the feasible Q_{ac} is missed and the solver converge to the other Q_{ac}.

What definitely is true is that there is only one solution of the system of equations (4.6). Thus, when assuming the solution is left of the maximum efficiency and if the solution is feasible there is no problem. But when the solution is unfeasible, it is impossible to make it feasible again, because there is just one solution. As mentioned before, a reason that the solution is unfeasible is that the compressor and the pipeline do not belong to each other. For example, the compressor can be too big for the used pipelines.



(a) Too high RPM

(b) Too low RPM

Figure 6.2: Line of solution from a point too high or too low RPM

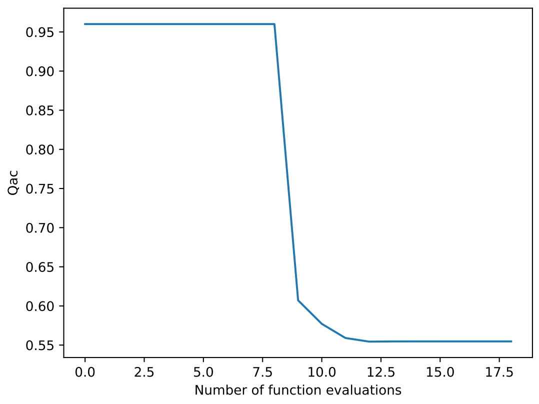


Figure 6.3: Convergence of a Q_{ac}

6.3 Problems with the solver

In chapter 4 we discussed that the solver to solve the system of equations is fsolve. This is a function in the scipy.optimize package and this will find the roots of the equations. Fsolve is a better option than Newton's method because Newton's method may not converge for many different values. The solver fsolve uses a more robust algorithm to get as quickly as possible to the solution. However solving with this solver may cause some problems.

As discussed in section 6.2 one of the problems is that the solver may not converge. The algorithm and model are well defined, so there must be a solution. The only drawback is that the initial values are taken wrongly. Which means that the initial values are not in the radius of convergence. The solver can not get the converged solution when the initial value is outside the radius of convergence. So, when the solver can not converge, the initial values are taken wrong.

Another problem with this solver is that the convergence is not quadratic. Newton's method is a solver that converges quadratically, so there are not so much iterations needed, but fsolve will not converge quadratically. Therefore this solver needs many more iterations to solve a problem. Newton's methods stops iterating when the difference of 0 and $f(x_n)$ is less than $1.48 \cdot 10^{-8}$ and fsolve stops iterating when difference of 0 and $f(x_n)$ is less than $1.49012 \cdot 10^{-8}$. Both the solvers have almost the same stopping criterion, but the total iterations can vary enormously. When the stopping criterion is low the iterations will be less. You can only increase the stopping criterion if the solution does not need to be precisely correct.

Unfortunately, there is in this case not an explanation for the amount of iterations. The only solution might be to use Newton's method, but this solver will not converge way to much for this system of equation. So, when the solver is not converging, the initial values need to be changed. Initial values that are generalized are almost impossible to determine for this system. When the solver needs many iterations, to increase the stopping criterion the total number of iterations can be less but only when there is some tolerance left.

Another problem with known functions, in Python, like newton and fsolve is that it is hard to get the total amount of iterations needed. The total function evaluations is easy to get. Because the 2 used functions are written by someone else, it is unknown if the total iterations is equal to the total function evaluations. The total iterations is less or equal to the total function evaluations. The total function evaluations are used in the rest of this chapter.

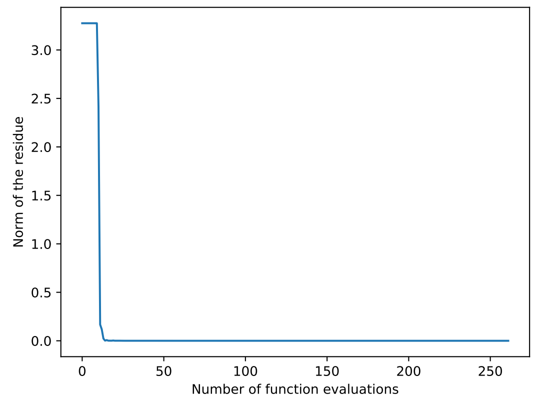
6.4 Numerical behaviour

In this section the numerical behaviour of the solver be will discussed based on two different examples. The first example is when the solution is feasible by solving the system of equations (4.5). The other example is when the solution is not feasible at all. The numerical behaviour can be analyzed using multiple plots.

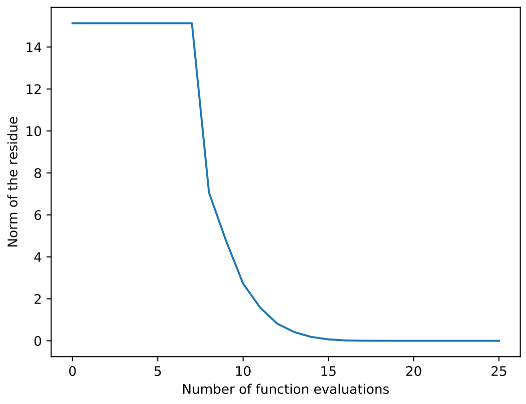
To check what the solver does with the convergence the norm of the residual vector is needed. Let $x_1, \dots, x_n$ be in the residual vector then the chosen norm is

$$||\text{Residual}|| = \sqrt{x_1^2 + \cdots + x_n^2}$$

The converging of the whole system can be shown in one figure when the norm is calculated. In figure 6.4 the norm of both the residual for a feasible solution and for an unfeasible solution is shown.



(a) Feasible solution

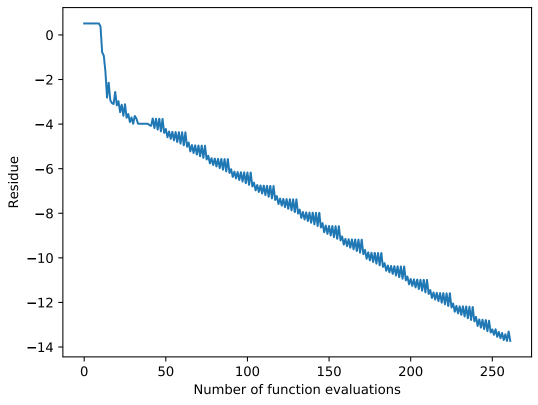


(b) Unfeasible solution

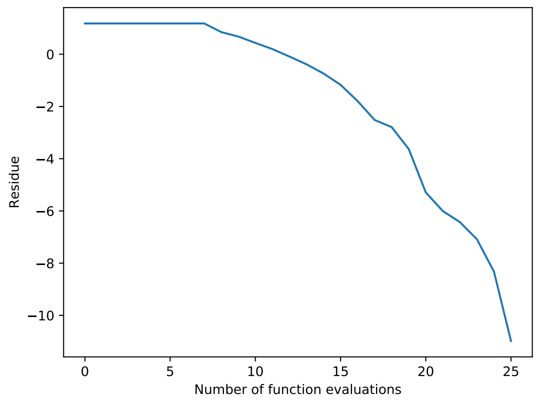
Figure 6.4: The norm of the residual for feasible and unfeasible solution

What can be seen from the norm is that the feasible solution needs more function evaluations to converge than the unfeasible solution. This is because the unfeasible solution has two more fixed point in his system of equations. Therefore it is easier to solve the system. The feasible solution also goes quick to zero, within 15 function evaluations, but takes way longer to converge eventually. When increasing the stopping time, the function evaluations will be less.

As can be seen in figure 6.4 both the algorithms converge well. This means that there is no problem with the solver, with the system of equations or the initial values. Everything is well implemented. What also can be seen is how quick the convergence is. Unfortunately this convergence is not quadratic.



(a) Feasible solution



(b) Unfeasible solution

Figure 6.5: The rate of convergence for feasible and unfeasible solution

The $\log_{10}$ is taken of the norm of the residual vector in figure 6.5. When taking this logarithm the size of the residual is shown. For example, at the 250th function evaluation the residual is the size 10^{-14}. The convergence of the feasible solution is thus not quadratic, but is linear which can bee seen in figure 6.5a. When looking at figure 6.5b the convergence of the unfeasible solution looks like it is quadratic. However this is not true, this is also a linear convergence.

The solver that is used, fsolve, does not converge quadratically but linear. The reason that the solver converges linearly is because this solver uses a more robust algorithm to get the solution sooner and therefore loses the quadratic convergence. The solver converges thus well for the used system of equations. Why 250 function evaluations are needed to get a feasible solution is unknown. A reason can be that this system of equations has two more variables and thus two more equations, which makes it harder to solve.

6.5 Parameter studies

In this section a parameter study will be done. This study gives more inside in what kind of pipelines the used compressor can handle and what kind of pipelines it can not handle.

First of all make sure the compressor works on the maximum efficiency. There are three cases to discuss, the first one is with a rotation speed of 13000 RPM, the last two are with the minimum and maximum rotation speed. Based on these three cases the pipelines that are suitable for the used compressor can be found.

The first thing to do is to check what the $\frac{P_d}{P_s}$ ratio is. The purpose of this ratio is that the suction pressure and discharge pressure differ always a factor, which is the ratio. This ratio is not only applicable for the suction and discharge pressure, but also for the inlet pressure and outlet pressure at the pipelines.

Let's start with the pressure ratio at a rotation speed of 13000 RPM. This ratio can be calculated by using equations (2.1) and (2.2). Then the ratio $\frac{P_d}{P_s} = 1.3886$. So, when $P_s = 20$ bar the compressor compresses the gas up to $P_d \approx 28$ bar. The ratio is also applicable for the pressure drop in the pipeline. When the pressure is 20 bar at the end of a pipeline, it is approximately 28 bar at the beginning of the pipeline. Now that this is clear, some calculations with the length and diameter of the pipeline can be done.

Looking at the Weymouth pressure drop equation (4.1) the diameter and length are both parameters. Changing the diameter will have a much bigger impact on the mass flow rate then changing the length. The interesting thing is: for a fixed length the diameter is the same independent of the inlet/outlet pressure ratio. This is because the pressures have to deal with the same pressure ratio, for a certain rotation speed, for every suction and discharge pressure. So for every length of the pipeline there is a diameter and with those two parameters the system of equations (4.5) should have a solution. This is true for every rotation speed in the feasible area.

When the rotation speed is between 10000 RPM and 15700 RPM the pressure ratio must be between 1.2178 and 1.6024. When the length of the pipeline is fixed, the diameter of the pipe is the same for a pressure ratio somewhere between 1.2178 and 1.6024. This does not mean that every pressure ratio has the same diameter, the diameter is dependent on the pressure ratio.

When the length is longer, the diameter will also increase.

Assume the pipelines for gas transportation are 100 km long. When the pipeline is fixed at 100 km, the diameter can be something between 0.4574 m and 0.4816 m. The system of equations (4.5) only has a solution when when the right pressure ratio is used. So when the rotation speed is determined, the pressure ratio and the diameter can be calculated by a fixed length. If the solver does not converge, the length can be adjusted. Some small adjustments of the length give almost the same diameter, but the solver will converge.

6.6 Two examples

In this section two examples will be shown. The first example is feasible and the second example will be unfeasible. The unfeasible solution will be calculated by the flowchart in figure 6.1. In this example a no will be answered when a question is asked. So the N_r and the Q_{ac} will be fixed.

In the first example the solution of the system is feasible after solving the system of equations (4.5). The input data, for this example, are given in table 6.1. When filling in these input data into the algorithm, the algorithm will solve the system of equations (4.5) and the results are given in table 6.2. The red dot, in figure 6.6a, is the solution as given in table 6.2. This solution lies in the compressor map, so it is feasible.

Now for the second example, the input data will be the same only the pipeline will be much longer. Now the length of the pipeline is 100 km in stead of 20 km. When looking at figure 6.6b there is a red dot and a blue dot. The red dot is the solution when only the rotation speed is fixed. It is right of the surge line, which means the solution is physically not possible. The blue dot in the same figure is the solution, which is given in table 6.2. This blue dot is the solution when Q_{ac} is also fixed and is just outside the feasible area. So this solution is not feasible for the chosen compressor.

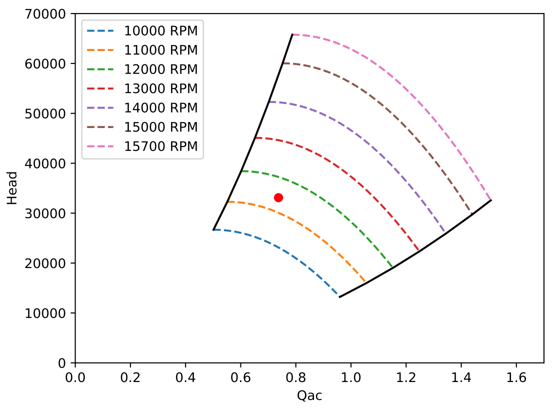
In conclusion, the input data for example 1 is feasible. Because the length of the pipeline is way shorter than the length in example 2. The solution in example 2 is not feasible. The reason why this is not feasible is because the compressor is too small for the pipeline. The length is much longer that the compression will be too high for this compressor. Taken all together, with this pipeline the compressor needs to be bigger to be feasible.

Parameter	Values Example 1	Values Example 2	Unit
$b_1 - b_6$	$b_1 = 1.059 \cdot 10^{-4}$ $b_2 = 6.418$ $b_3 = -6.401 \cdot 10^4$ $b_4 = 1.727 \cdot 10^{-1}$ $b_5 = 1.942 \cdot 10^4$ $b_6 = -1.505 \cdot 10^8$	$b_1 = 1.059 \cdot 10^{-4}$ $b_2 = 6.418$ $b_3 = -6.401 \cdot 10^4$ $b_4 = 1.727 \cdot 10^{-1}$ $b_5 = 1.942 \cdot 10^4$ $b_6 = -1.505 \cdot 10^8$	-
c_w	0.47857	0.47857	-
D	$D = 0.3$	$D = 0.3$	m
L	$L = 2 \cdot 10^4$	$L = 1 \cdot 10^5$	m
P_1	$P_1 = 45 \cdot 10^5$	$P_1 = 45 \cdot 10^5$	Pa
P_2	$P_2 = 30 \cdot 10^5$	$P_2 = 30 \cdot 10^5$	Pa
γ	$\gamma = 1.27$	$\gamma = 1.27$	-
R	$R = 437.60$	$R = 437.60$	J/kmol.K
S	$S = 19/29$	$S = 19/29$	-
T	$T = 300$	$T = 300$	K
Z	$Z = 0.96$	$Z = 0.96$	-

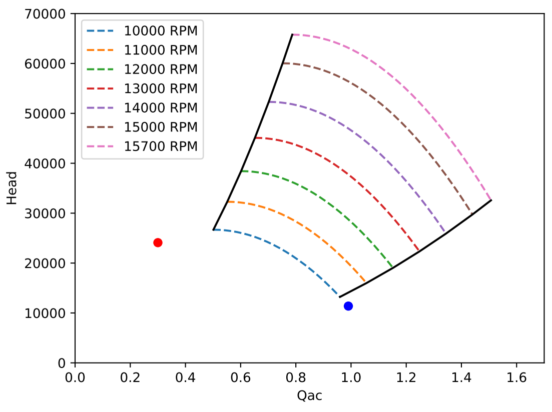
Table 6.1: Input data example 1 and 2

Variable	Values example 1	Values example 2	Unit
$\dot{m}_d$	19.57080319595879	12.19602268455741	kg/s
$\dot{m}_f$	0.021559910505937	0.103256404180448	kg/s
$\dot{m}_s$	19.59236310646472	12.29927908873785	kg/s
N_r	11425.52147920879	10000	RPM
P_d	43.20394176705705	59.96916905759383	bar
P_s	33.45975646318788	15.52116377947649	bar
Q_{ac}	0.737154989058859	0.990300221637407	m^3/s

Table 6.2: Output data example 1 and 2



(a) Example 1 is feasible



(b) Example 2 is not feasible

Figure 6.6: Solution in compressor map

Chapter 7

Conclusion

The model of the compressor is defined in a way that there is a feasible area. This feasible area is confined by the surge line, choke line, the minimum rotation speed and the maximum rotation speed. This minimum N_r is 10000 RPM and the maximum N_r is 15700 RPM.

After the compressor model is defined, the pipe-compressor model will be defined as a compressor with a pipeline in the front and in the back. The length and diameter of the pipeline can be differed. To solve this problem there is a system of equations (4.5).

First let the length of the pipeline be 20 km and the diameter 30 cm. The system of equations can be solved by the input data and this solution was feasible. This way the input data for the pipe-compressor system is chosen well. The number of iterations to converge the program was 250.

At the second example the length of the pipeline was longer (100 km). The diameter was still the same (30 cm). When solving the system of equations with the new input data, the solution was not feasible. After fixing the rotation speed and the volume flow, the solution should be feasible. Unfortunately, the solution was just outside the feasible area. Which means that the pipeline and compressor do not belong to each other.

If the solution is unfeasible, for a fixed N_r, after solving the system of equations then the solution can not be feasible anymore. For a certain efficiency (everything excepted the maximum efficiency) and a certain rotation speed, there would be two solutions for the Q_{ac}. But when solving the system of equations with a fixed N_r there is just one solution possible. So, if the solution is unfeasible after solving the system of equations then the solution can not be feasible anymore.

In conclusion, when solving a system of equations for the pipe-compressor model, the length and diameter are important. For a certain pressure ratio the diameter of the pipe is the same when the length is fixed. The compressor used in this report is too big for the a 100 km pipeline with a diameter of 30 cm, but is perfect for a 20 km pipeline with a diameter of 30 cm. If the solution is not feasible, the compressor is too big or too small for the used pipeline. The compressor needs to be changed to make the solution feasible.

Chapter 8

Recommendations

If the model in this thesis is used for the question if a certain centrifugal compressor is suitable with a certain pipeline, there are some recommendations I would like to make. First check what the range of the rotation speed of the used compressor is. Then the compressor map can be made, via Appendix A.3. The compressor map is useful to see where the compressor is operating.

With the algorithm in Appendix A.6 the initial values are really important. If the initial values are not suitable for the parameter data the solver will not converge. Then the initial values need to be chosen differently. This can be done by looking at the radius of convergence of each equation. The initial values can be chosen such that the solver will converge with the given parameter data.

Finally we assumed that the solution is always left of the efficiency line when the rotation speed was fixed. This is still not proved, but it seems that it is true for every value. To prove if the solution converges always to the left of the maximum efficiency the convergence behaviour of the equations should be investigated. So, for further research this issue would be great to tackle.

Bibliography

[1] K.S. Chapman, M. Abbaspour, et al. Non-isothermal compressor station transient modeling. In *PSIG Annual meeting*. Pipeline Simulation Interest Group, 2003.

[2] Michael C Swarden, Shankar S Magge, Willem Jansen, and Anthony F Carter. Centrifugal compressor, July 15 1980. US Patent 4,212,585.

[3] Mech4Study. Centrifugal compressor: Principle, construction, working, types, advantages, disadvantages with its application, nov 2017.

[4] Yannick Bousquet, Xavier Carbonneau, Guillaume Dufour, Nicolas Binder, and Isabelle Trebinjac. Analysis of the unsteady flow field in a centrifugal compressor from peak efficiency to near stall with full-annulus simulations. *International Journal of rotating machinery*, 2014, 2014.

[5] Scott Golden, Scott A Fulton, and Daryl W Hanson. Understanding centrifugal compressor performance in a connected process system. *Petroleum Technology Quarterly*, 2002.

[6] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

[7] C. Vuik, F.J. Vermolen, M.B. van Gijzen, and M.J. Vuik. *Numerical Methods for Ordinary differential equations.* Delft Academic Press (DAP), 2016.

[8] J.G.P. Barnes. An algorithm for solving non-linear equations based on the secant method. *The Computer Journal*, 8(1):66–72, 1965.

[9] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

Appendix A

Python Code

A.1 Code Newton's method

```python
def derivative(f, a, h=1e-4):
    return (f(a + h) - f(a))/h

def newton_method(f, x0, e = 1e-10):
    delta = abs(0-f(x0))
    while delta > e:
        x0 = x0 - f(x0)/derivative(f, x0)
        delta = abs(0-f(x0))
    return x0
```

A.2 Code Newton's method for systems

```python
import numpy as np
from numpy.linalg import pinv

def J(f,x,dx=1e-8):
    n=len(x)
    func= f(x).T[0]
    jac=np.zeros((n,n))
    for j in range(n): #through columns
        Dxj=(abs(x[j])*dx if x[j]!=0 else dx)
        x_plus=[(xi if k!=j else xi+Dxj) for k,xi in enumerate(x)]
        jac[:,j]=(np.subtract(f(x_plus).T[0],func))/Dxj
    return jac

def newton_method2(f, x0, e = 1e-8):
    delta = abs(np.subtract(np.zeros((len(f(x0)),1)), f(x0)))
    for i in range(len(f(x0))):
        while delta[i][0] > e:
            transpose = J(f,x0).T
            product = pinv(J(f,x0))
            sub = np.array([])
            for j in range(len(f(x0))):
                sub = np.append(sub, [product.dot(f(x0))[j]])
            x0 = x0 - sub
            delta = abs(np.subtract(np.zeros((len(f(x0)),1)), f(x0)))
    return x0
```

A.3 Code Compressor Map

```python
from math import *
import numpy as np
from scipy.optimize import *
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt


Ts = 288
gamma = 1.27
alpha = (gamma - 1)/gamma
LHV = 47e6
M = 19
UniR = 8314.46
R = UniR/M
n_turbine = 0.8
N_r = 46498
Z = 1

b1 = 1.0595e-4
b2 = 6.418
b3 = -6.401e4
b4 = 1.727e-1
b5 = 1.942e4
b6 = -1.505e8

eff = 0.65
x1 = np.empty(0)
y1 = np.empty(0)

Qac = np.linspace(0,  1.6, 24000)
N_r = [10000, 11000, 12000, 13000, 14000, 15000, 15700]
x = np.empty(0)
y = np.empty(0)
for i in N_r:
    Head = i**2*(b1 + b2*(Qac/i) + b3*(Qac**2/i**2))
    y = np.append(y,np.max(Head))
    xbegin = -b2*i/(2*b3)
    x = np.append(x, xbegin)

    def efficiency(z):
        E = np.empty((1))
        E[0] = eff - b4 - b5*(z/i) - b6*(z**2/i**2)
        return E
    z = np.array([90])
    xeind = newton(efficiency, z)
    x1 = np.append(x1,xeind)
    y1 = np.append(y1,i**2*(b1 + b2*(xeind/i) + b3*(xeind**2/i**2)))

    Qac1 = np.linspace(xbegin,  xeind, 1200)
    head = i**2*(b1 + b2*(Qac1/i) + b3*(Qac1**2/i**2))
    plt.plot(Qac1, head, '--')


for i in range(3):
    eff = [0.799, 0.75, 0.70]
    x2 = np.empty(0)
    y2 = np.empty(0)
    for j in N_r:
        def efficiency(z):
            E = np.empty((1))
            E[0] = eff[i] - b4 - b5*(z/j) - b6*(z**2/j**2)
            return E
        z = np.array([5])
        newx = newton(efficiency, z)
        x2 = np.append(x2,newx)
        y2 = np.append(y2,j**2*(b1 + b2*(newx/j) + b3*(newx**2/j**2)))
        plt.plot(x2,y2, 'grey')

plt.plot(x, y, 'black')
plt.plot(x1, y1, 'black')

plt.legend(('10000 RPM', '11000 RPM', '12000 RPM', '13000 RPM', '14000 RPM', \
            '15000 RPM', '15700 RPM'))
plt.axis([0, 1.7, 0, 70000])
plt.xlabel('Qac')
plt.ylabel('Head')
plt.show()
```

A.4 Code Compressor part I

```python
from math import *
import numpy as np
from scipy.optimize import *

class Compressor(object):

    def __init__(self, Ps, Pd, Qac):
        self.Ps = Ps
        self.Pd = Pd
        self.Qac = Qac

    def Values(self):
        if type(self.Ps) == str:
            xbegin, xeind = self.Checken()
            zguess = np.array([self.Pd])
            if self.Qac >= xeind[0]:
                self.Qac = xeind[0]
                self.Ps = newton(self.Func1,zguess)
                return self.Ps, self.Qac
            if self.Qac <= xbegin:
                self.Qac = xbegin
                self.Ps = newton(self.Func1,zguess)
                return self.Ps, self.Qac
            else:
                self.Ps = newton(self.Func1,zguess)
                return self.Ps, self.Qac

        elif type(self.Pd) == str:
            zguess = np.array([self.Ps])
            xbegin, xeind = self.Checken()
            if self.Qac >= xeind[0]:
                self.Qac = xeind[0]
                self.Pd = newton(self.Func2,zguess)
                return self.Pd, self.Qac
            if self.Qac <= xbegin:
                self.Qac = xbegin
                self.Pd = newton(self.Func2,zguess)
                return self.Pd, self.Qac
            else:
                self.Pd = newton(self.Func2,zguess)
                return self.Pd, self.Qac

        elif type(self.Qac) == str:
            xbegin, xeind = self.Checken()
            zguess = np.array([self.Ps])
            self.Qac = xeind
            Pdmin = newton(self.Func2,zguess)
            self.Qac = xbegin
            Pdmax = newton(self.Func2,zguess)
            if self.Pd >= Pdmax[0]:
                self.Pd = Pdmax[0]
                zguess = np.array(xeind)
                self.Qac = newton(self.Func3, zguess)
                return self.Qac, self.Pd
            if self.Pd >= Pdmin[0]:
                self.Pd = Pdmin[0]
                zguess = np.array(xeind)
                self.Qac = newton(self.Func3, zguess)
                return self.Qac, self.Pd
            else:
                zguess = np.array(xeind)
                self.Qac = newton(self.Func3, zguess)
                return self.Qac, self.Pd

    def Checken(self):
        xbegin = -b2*N_r/(2*b3)
        z = np.array([xbegin*2])
        xeind = newton(self.Efflines, z)
        return xbegin, xeind

    def Efflines(self, z):
        E = np.empty((1))
        E[0] = eff - b4 - b5*(z/N_r) - b6*(z**2/N_r**2)
        return E

    def Head(self, Ps, Pd, Qac):
        H = (1/alpha)*Z*R*Ts*((Pd/Ps)**alpha - 1)
        return H/(N_r**2) - b1 - b2*(Qac/N_r) - b3*(Qac**2/N_r**2)

    def Func1(self,z):
        head = np.empty((1))
        head[0] = self.Head(z[0], self.Pd, self.Qac)
        return head

    def Func2(self,z):
        head = np.empty((1))
        head[0] = self.Head(self.Ps, z[0], self.Qac)
        return head

    def Func3(self,z):
        head = np.empty((1))
        head[0] = self.Head(self.Ps, self.Pd, z[0])
        return head

    def TempUit(self):
        return Ts + (Ts/self.Efficiency()) * ((self.Pd/self.Ps)**alpha - 1)

    def Efficiency(self):
        return b4 + b5*(self.Qac/N_r) + b6*self.Qac**2/N_r**2

############################# VARIABLES #############################

Ts = 288
gamma = 1.27
alpha = (gamma - 1)/gamma
LHV = 47e6
M = 19
UniR = 8314.46
R = UniR/M
n_turbine = 0.8
N_r = 6000
Z = 1
eff = 0.65


b1 = 1.0595e-4
b2 = 6.418
b3 = -6.401e4
b4 = 1.727e-1
b5 = 1.942e4
b6 = -1.505e8

############################## EXAMPLES ##############################

Pd = 100e5
Qac = 20
test = Compressor('Ps', Pd, Qac)
Ps, Qac = test.Values()
print(Ps[0]/1e5, Qac)


Ps = 13e5
Qac = 0.3
test1 = Compressor(Ps, 'Pd', Qac)
Pd, Qac = test1.Values()
print(Pd[0]/1e5, Qac)

Ps = 22e5
Pd = 47e5
test2 = Compressor(Ps, Pd, 'Qac')
Qac, Pd = test2.Values()
print(Qac[0], Pd/1e5)
```

A.5 Code Compressor part II

```
from math import *
import numpy as np
from scipy.optimize import *
from Newton_Method import *
from Newton_Method2 import *

class Compressor(object):

    def __init__(self, Ps, Pd, Qac):
        self.Ps = Ps
        self.Pd = Pd
        self.Qac = Qac

```

```python
    def Values(self):
        if type(self.Ps) == str:
            zguess = np.array([1e-10])
            self.N_r = self.Qac*2*b6/(-b5)
            self.Ps = newton(self.Func1, zguess)
            print('1ste:', newton_method(self.Func1, zguess)[0]/1e5)
            print('2de:', newton(self.Func1, zguess)[0]/1e5)
            return self.Ps, self.N_r

        elif type(self.Pd) == str:
            zguess = np.array([self.Ps])
            self.N_r = self.Qac*2*b6/(-b5)
            self.Pd = newton(self.Func2, zguess)
            print('1ste:', newton_method(self.Func2, zguess)[0]/1e5)
            print('2de:', newton_method2(self.Func2, zguess)[0]/1e5)
            return self.Pd, self.N_r

        elif type(self.Qac) == str:
            zguess = np.array([self.Ps])
            C = (b1 + b2*(-b5/(2*b6)) + b3*(-b5/(2*b6))**2)
            self.N_r = sqrt(self.Head1()/C)
            self.Qac = newton(self.Func3, zguess)
            print('1ste:', newton_method(self.Func3, zguess)[0])
            print('2de:', newton_method2(self.Func3, zguess)[0])
            return self.Qac, self.N_r

    def Head(self, Ps, Pd, Qac):
        H = (1/alpha)*Z*R*Ts*((Pd/Ps)**alpha - 1)
        return H/(self.N_r**2) - b1 - b2*(Qac/self.N_r) - b3*(Qac**2/self.N_r**2)

    def Func1(self,z):
        head = np.empty((1))
        head[0] = self.Head(z[0], self.Pd, self.Qac)
        return head

    def Func2(self,z):
        head = np.empty((1))
        head[0] = self.Head(self.Ps, z[0], self.Qac)
        return head

    def Func3(self,z):
        head = np.empty((1))
        head[0] = self.Head(self.Ps, self.Pd, z[0])
        return head

    def Head1(self):
        return (1/alpha)*Z*R*Ts*((self.Pd/self.Ps)**alpha - 1)

    def TempUit(self):
        return Ts + (Ts/self.Efficiency()) * ((self.Pd/self.Ps)**alpha - 1)

    def Efficiency(self):
        return b4 + b5*(self.Qac/N_r) + b6*self.Qac**2/self.N_r**2

    def Rho(self):
        return self.Ps/(Z*R*Ts)

    def Power(self):
        return self.Head1()*(self.Rho()*self.Qac)/self.Efficiency()

    def MassFlowFuel(self):
        return self.Power()/(LHV*n_turbine)

    def MassFlowIn(self):
        return self.Rho()*self.Qac + self.MassFlowFuel()

############################ VARIABLES ###########################

Ts = 300
gamma = 1.27
alpha = (gamma - 1)/gamma
LHV = 47e6
M = 19
UniR = 8314.46
R = UniR/M
n_turbine = 0.8
Z = 0.96

b1 = 1.0595e-4
b2 = 6.418
b3 = -6.401e4
b4 = 1.727e-1
b5 = 1.942e4
b6 = -1.505e8

############################## EXAMPLES ##############################

Pd = 12.84141422159631e5
Qac = 2.829418011428186
test = Compressor('Ps', Pd, Qac)
Ps, N_r = test.Values()
print(Ps[0]/1e5,N_r)
print()

Ps = 5.719903163969866e5
Qac = 2.829418011428186
test1 = Compressor(Ps, 'Pd', Qac)
Pd, N_r = test1.Values()
print(Pd[0]/1e5,N_r)
print()

Ps = 22e5
Pd = 47e5
test2 = Compressor(Ps, Pd, 'Qac')
Qac, N_r = test2.Values()
print(Qac[0]*test2.Rho(),N_r)
print(test2.MassFlowFuel())
```

A.6 Code Pipeline simulation

```python
from math import *
import numpy as np
from scipy.optimize import *
from Newton_Method import *
from Newton_Method2 import *
import matplotlib.pyplot as plt

class Compressor(object):

    def __init__(self, P1, P4):
        self.P1 = P1
        self.P4 = P4

    def Values(self):
        zguess = np.array([19.03, self.P4, 19, 0.03, self.P1, 1, 13000])
##        z[0] = m_s   z[1] = P_s   z[2] = m_d   z[3] = m_f
##        z[4] = P_d   z[5] = Qac   z[6] = N_r
        self.z = np.empty((0))
        self.opl = np.empty((0))
        Arr = fsolve(self.Func1,zguess)
        if Arr[6] <= 15700 and Arr[6] >= 10000:
            return Arr, self.z, self.opl
        elif Arr[6] >= 15700:
            self.z1 = np.empty((0))
            self.opl1 = np.empty((0))
            self.N_r = 15700
            xbegin, xeind = self.Checken()
            zguess = np.array([8.5, self.P4, 8, 0.5, self.P1, xeind[0]])
            Arr = fsolve(self.Func2,zguess)
            Arr = np.append(Arr, np.array(self.N_r))
            if Arr[5] < xbegin:
                self.z2 = np.empty((0))
                self.opl2 = np.empty((0))
                self.eff = self.Efficiency(Arr[5],self.N_r)
                self.Qac = fsolve(self.Efflines2, xeind[0])
                zguess = np.array([8.5, 1e5, 8, 0.5, self.P1])
                Arr = fsolve(self.Func3,zguess)
                Arr = np.append(Arr, np.array([self.Qac[0],self.N_r]))
                return Arr, self.z2, self.opl2
            else:
                return Arr, self.z1, self.opl1
        elif Arr[6] <= 10000:
            self.z1 = np.empty((0))
            self.opl1 = np.empty((0))
            self.N_r = 10000
            xbegin, xeind = self.Checken()
            zguess = np.array([8.5, self.P4, 8, 0.5, self.P1, xeind[0]])
            Arr = fsolve(self.Func2,zguess)
            Arr = np.append(Arr, np.array(self.N_r))
            if Arr[5] < xbegin:
                self.z2 = np.empty((0))
                self.opl2 = np.empty((0))
                self.eff = self.Efficiency(Arr[5],self.N_r)
                self.Qac = fsolve(self.Efflines2, xeind[0])
                zguess = np.array([8.5, 1e5, 8, 0.5, self.P1])
                Arr = fsolve(self.Func3,zguess)
                Arr = np.append(Arr, np.array([self.Qac[0],self.N_r]))
                return Arr, self.z2, self.opl2
            else:
                return Arr, self.z1, self.opl1

    def Head(self, Ps, Pd, Qac, N_r):
       H = (1/alpha)*Z*R*Ts*((Pd/Ps)**alpha - 1)
        return H/(N_r**2) - b1 - b2*((Qac)/N_r) - b3*((Qac)**2/N_r**2)

    def Func1(self,z):
        opl = np.empty((7))
        TempUit = self.TempUit(z[1], z[4],z[5],z[6])
        opl[0] = self.Efficiency2(-b5/(2*b6)) - self.Efficiency(z[5], z[6])
        opl[1] = self.Head(z[1], z[4], z[5], z[6])
        opl[2] = z[3] - self.MassFlowFuel(z[1], z[4], z[2], z[5], z[6])
        opl[3] = z[5] - z[2]/self.Rho(z[1])
        opl[4] = z[2] - self.Weymouth(z[4], self.P4, TempUit)
        opl[5] = z[0] - z[2] - z[3]
        opl[6] = z[0] - self.Weymouth(self.P1, z[1], Ts)
        self.z = np.append(self.z, z)
        self.opl = np.append(self.opl,opl)
        return opl

    def Func2(self,z):
        opl = np.empty((6))
        TempUit = self.TempUit(z[1], z[4],z[5], self.N_r)
        opl[0] = self.Head(z[1], z[4], z[5], self.N_r)
        opl[1] = z[3] - self.MassFlowFuel(z[1], z[4], z[2], z[5], self.N_r)
        opl[2] = z[5] - z[2]/self.Rho(z[1])
        opl[3] = z[2] - self.Weymouth(z[4], self.P4, TempUit)
        opl[4] = z[0] - z[2] - z[3]
        opl[5] = z[0] - self.Weymouth(self.P1, z[1], Ts)
        self.z1 = np.append(self.z1, z)
        self.opl1 = np.append(self.opl1,opl)
        return opl

    def Func3(self, z):
        opl = np.empty((5))
        TempUit = self.TempUit(z[1], z[4], self.Qac, self.N_r)
        opl[0] = z[3] - self.MassFlowFuel(z[1], z[4], z[2], self.Qac, self.N_r)
        opl[1] = self.Qac - z[2]/self.Rho(z[1])
        opl[2] = z[2] - self.Weymouth(z[4], self.P4, TempUit)
        opl[3] = z[0] - z[2] - z[3]
        opl[4] = z[0] - self.Weymouth(self.P1, z[1], Ts)
        self.z2 = np.append(self.z2, z)
        self.opl2 = np.append(self.opl2,opl)
        return opl

    def Checken(self):
        xbegin = -b2*self.N_r/(2*b3)
        z = np.array([xbegin*2])
        xeind = newton(self.Efflines, z)
        return xbegin, xeind

    def Efflines(self, z):
        E = np.empty((1))
        E[0] = 0.65 - b4 - b5*(z/self.N_r) - b6*(z**2/self.N_r**2)
        return E

    def Efflines2(self, z):
        E = np.empty((1))
        E[0] = self.eff - b4 - b5*(z/self.N_r) - b6*(z**2/self.N_r**2)
        return E

    def Efficiency2(self, x):
        return b4 + b5*(x) + b6*(x**2)

    def Weymouth(self, P1, P2, T):
        return np.sign(P1**2-P2**2) * c_w * \
               np.sqrt((S*np.abs(P1**2-P2**2)*D**5.333)/(T*Z*L))

    def Head1(self, Ps, Pd):
        return (1/alpha)*Z*R*Ts*((Pd/Ps)**alpha - 1)
    def TempUit(self, Ps, Pd, Qac, N_r):
        return Ts + (Ts/self.Efficiency(Qac, N_r)) * ((Pd/Ps)**alpha - 1)

    def Efficiency(self, Qac, N_r):
        return b4 + b5*(Qac/N_r) + b6*(Qac**2)/(N_r**2)

    def Rho(self, P):
        return P/(Z*R*Ts)

    def Power(self, Ps, Pd, m, Qac, N_r):
        return (self.Head1(Ps, Pd)*m)/self.Efficiency(Qac, N_r)

    def MassFlowFuel(self, Ps, Pd, m, Qac, N_r):
        return self.Power(Ps, Pd, m, Qac, N_r)/(LHV*n_turbine)


############################## VARIABLES ##############################

Ts = 300
gamma = 1.27
alpha = (gamma - 1)/gamma
LHV = 47e6
M = 19
UniR = 8314.46
R = UniR/M
n_turbine = 0.8
Z = 0.96

D = 0.3
L = 2e4
S = 19/29
c_w = 0.47857

b1 = 1.0595e-4
b2 = 6.418
b3 = -6.401e4
b4 = 1.727e-1
b5 = 1.942e4
b6 = -1.505e8

############################## EXAMPLES ##############################

P1 = 45e5
P4 = 30e5
test = Compressor(P1, P4)
Ps,z,opl = test.Values()
print(Ps[0], Ps[1]/1e5, Ps[2], Ps[3], Ps[4]/1e5, Ps[5], Ps[6], \
      test.Efficiency(Ps[5],Ps[6]))
print(test.Func1(Ps))

y = np.empty((0))
x = range(0,int(len(opl)/7))
for i in range(0,int(len(opl)/7)):
    norm = sqrt(opl[0+i*7]**2+opl[1+i*7]**2+opl[2+i*7]**2+opl[3+i*7]**2\
             +opl[4+i*7]**2+opl[5+i*7]**2+opl[6+i*7]**2)
    y = np.append(y,norm)

plt.plot(x,y)
plt.xlabel('Number of function evaluations')
plt.ylabel('Norm of the residue')
```

```python
plt.show()

plt.plot(x,np.log10(y))
plt.xlabel('Number of function evaluations')
plt.ylabel('Residue')
plt.show()
```