Oxygen Enhanced Combustion in a Cement Rotary Kiln

Clean Industrial Combustion Case Study

10 July 2020

Jori Holster

Supervisor	Dr. D.J.P. Lahaye
Committee Members	Drs. E.M. van Elderen
	Dr. K. Cools

Bsc. Thesis Applied Mathematics
Delft University of Technology

Abstract

Decreasing NO_x emissions is becoming increasingly important as it has many life-changing implications on humanity and nature. In this work several models were constructed to simulate the combustion of methane in a cement rotary kiln, and it was investigated whether oxygen-enhanced combustion leads to less NO_x production for the obtained models. This was done using the Cantera software package with Python. Starting off with a single zero-dimensional reactor equipped with a reduced one-step global reaction mechanism, which was expanded to include two-step and four-step reaction mechanisms. Combustion inside this homogeneous reactor was simulated for various stoichiometric conditions for an initial temperature of 1000K. Subsequently, a one-dimensional model of chained reactors to simulate flow was considered, equipped with both the two-step and four-step mechanisms. This was realized using the scalar convection-diffusion equation to compute the flow throughout the reactor. All aforementioned models were adjusted to replace air with oxygen-enhanced air, containing higher levels of oxygen for every iteration. The simulated temperature evolution was examined using the exponential relationship of temperature and thermal NO_x.

Contents

Chapter 1

Introduction

In the past few decades, awareness of the dangers of nitrogen oxides (NO_x) emissions has risen. Apart from the respiratory health issues it can cause at high levels, it also has a tremendous impact on nature. As the ground, plants and trees absorb air, they in turn absorb more nitrogen. For some plants this has negative effects such as leaf damage and reduced growth, other plants flourish and overgrow the surrounding vegetation. This affects all other life around it, disrupting the balance of nature [1].

The importance of reducing emissions hopefully has been made clear. One contributor of these emissions are factories that utilise combustion as ways of obtaining their product. Nowadays cement is a widely used substance for building purposes, worldwide billions of metric tons are produced every year. Producing cement is done in a rotary kiln, where combustion of gas takes place to heat a mixture of limestone and clay. During this process NO_x is produced as a side effect.

It is suspected that increasing oxygen levels during this process could reduce the emission of NO_x. However before such things are to be tested within an actual rotary kiln, a computer model must be made to estimate its effects. This report will describe how such a mathematical model has been obtained by trying to obey laws of chemistry and physics, be it a simplification in most cases, as a way to investigate whether oxygen enhanced combustion within a rotary kiln would reduce NO_x emissions.

Chapter 2

Chemical kinetics

Investigation through computer modelling revolves around obeying laws of physics, chemistry and mathematics in order to predict what happens in a real-life situation. The models created within this report have a main focus on the chemical reactions side of the system, however this logically also entails some physics and mathematics. This focus means the basis of the model is about investigating the chemical reaction side of the complex system which occurs during combustion. Chemical kinetics is the branch of chemistry which describes this chemical reaction side of a process such as combustion.

2.1 Reaction equations

The most basic form of a chemical reaction is the collision between two molecules. Within this elementary reaction there exist no intermediate reactions and therefore this reaction describes what happens on a molecular scale. A global reaction can then be described as multiple elementary reactions in rapid succession [2].
Any chemical reaction, be it global or elementary, can be described in the following way:

$$A + B + C + ... \quad \xrightarrow{k} \quad D + E + F + ... \tag{2.1}$$

which indicate that reactants A,B,C... react to products D,E,F... with rate coefficient k. The speed at which these species react with one another depends on various characteristics, and together represent this coefficient k. For example, one could imagine that at higher temperatures as molecules are moving faster with more energy, more collisions take place resulting in a higher reaction rate. This will be discussed further in section 2.3.
In order to obtain a computer model, a mathematical model is required. Looking at the change in concentration over time of species A from equation (2.1), it can be expressed in the way found below.

$$\frac{d[A]}{dt} = -k \cdot [A]^a \cdot [B]^b \cdot [C]^c \cdot ... \tag{2.2}$$

where a,b and c represent the reaction orders and square brackets indicate the concentration of a specified species. The order of a reaction is an indication of the impact of a change in concentration on the rate of a reaction. Differential equation

(2.2) can be solved once k, a, b, c.. are known. These values are determined experimentally and will therefore be provided with a source within this report.

Next, a simple reaction mechanism is considered involving imaginary species A, B and C.

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C \tag{2.3}$$

Which yields a system of ordinary differential equations.

$$\frac{d[A]}{dt} = -k_1[A] \tag{2.4}$$

$$\frac{d[B]}{dt} = k_1[A] - k_2[B] \tag{2.5}$$

$$\frac{d[C]}{dt} = k_2[B] \tag{2.6}$$

The analytical solution to this system takes a considerate amount of work for such a small problem, see equation (2.7) [2]. If one would use a model containing every species of combustion, it would take a huge amount of work to solve it analytically. Reducing the size of this system of equations will be a very useful simplification as it will be computationally efficient. The upcoming section will describe ways of analysing a reaction mechanism to break down a complex reaction system to some global steps.

$$\begin{aligned}
[A] &= [A]_0 \exp(-k_1 t) \\
[B] &= [A]_0 \frac{k_1}{k_1 - k_2}(\exp(-k_2 t) - \exp(-k_1 t)) \\
[C] &= [A]_0 (1 - \frac{k_1}{k_1 - k_2}\exp(-k_2 t) + \frac{k_2}{k_1 - k_2}\exp(-k_1 t))
\end{aligned} \tag{2.7}$$

2.2 Analysis of reaction mechanisms

The combustion of hydrocarbon fuels consists of many more elementary reactions than (2.3). For fuels which contain a lot of carbon or hydrogen atoms, this can reach up to thousands of elementary reactions [2]. Determination of the most and least important steps within a reaction mechanism is therefore very useful. The method found below is an example of how reduced mechanisms are obtained by simplification.

This assumption stems from the observation that reactive species disappear quickly. Lets consider reaction mechanism (2.3), but this time B is a highly reactive species. Any species B that is created from A, is immediately converted to C. This leads to believe that the concentration of B does not fluctuate significantly, or the derivative of the concentration over time is almost equal to zero. The quasi-steady state is the approximation of this derivative to zero [2].

$$\frac{d[B]}{dt} = k_1[A] - k_2[B] \approx 0 \tag{2.8}$$

By combining this approximation with equation (2.6) one obtains that the time derivative of the concentration of C can be expressed in [A];

$$\frac{d[C]}{dt} \approx k_1[A] = k_1[A]_0\exp(-k_1t) \tag{2.9}$$

which can be directly calculated by integrating equation (2.4). When investigating anything involving reaction mechanisms, generally one is interested in the end product. The quasi-steady state assumption terminates the intermediate product in the calculation and thus simplifies the system.

2.3 Arrhenius equation

Any chemical reaction is fundamentally a collision between molecules, according to collision theory. Temperature is an indirect measure of the thermal velocity of a molecule and therefore increasing temperature will increase the reaction rate. This temperature dependence on the rate coefficient of a reaction was first proposed by Svante Arrhenius, in 1889.

$$k = A \cdot \exp(\frac{-E_a}{RT}) \tag{2.10}$$

where A is the pre-exponential factor, E_a the activation energy and R the universal gas constant ($8.314\ J\text{mol}^{-1}K^{-1}$). The pre-exponential factor is a constant. Activation energy can be thought of as an energy barrier to overcome during the reaction. Both are determined experimentally.

In some cases it was noticed the pre-exponential factor did have a temperature dependence [2]. This observation led to the modified Arrhenius equation (2.11), which is used for all the models in this report. Mostly the pre-exponential factor does not depend on temperature resulting in $b = 0$, which corresponds to the original Arrhenius equation.

$$k = AT^b \cdot \exp(\frac{-E_a}{RT}) \tag{2.11}$$

2.4 Equivalence ratio

As combustion takes place fuel and an oxidizer react in a certain ratio: x mole of fuel reacts with y mole of oxidizer. When the fuel and oxidizer consume each other entirely, it is called a stoichiometric mixture. An excess of fuel will not surprisingly be called fuel-rich and an excess of oxidizer is said to be fuel-lean. The equivalence ratio ϕ is the fuel-oxidizer ratio compared to the fuel-oxidizer ratio at stoichiometric combustion conditions:

$$\phi = \frac{m_{fuel}/m_{ox}}{(m_{fuel}/m_{ox})_{st}} \tag{2.12}$$

where m is molecular mass, subscript 'ox' stands for oxidizer and subscript 'st' represents stoichiometric conditions. From this equation it becomes apparent that fuel-lean conditions correspond to $\phi < 1$, stoichiometric conditions means $\phi = 1$ and rich combustion yields $\phi > 1$ [3].

Chapter 3

Implementation

In order to make a computer model mathematics, chemistry and physics have to be implemented. Fortunately, software with such structures is readily available. For this project the interpreter Python was used. It contains software package Cantera, an open-source project which provides a broad spectrum of tools to tackle chemical problems. The official website contains documentation, tutorials and examples to help anyone get started [4]. This object-oriented program has many built-in tools which help greatly with the rotary kiln model. For example automatic computation of temperature increase after a reaction, the reactor network solving integrator: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers (SUNDIALS) and the ability to easily import thermodynamic properties of species. This chapter will present the implementation methods of the model and Cantera, the actual code can be found in Appendix A, but will be discussed later.

3.1 Cantera

Cantera utilizes a wide range of calculations in the background to determine for example temperature increase or species formation. The creation of species has been described in chapter 2. Calculation of the change in temperature will not be discussed within this report. However the documentation and explanation from Cantera itself is thorough so the interested reader is encouraged to read it, see [5].

To solve the energy equation Cantera needs information about the thermodynamics of species, transport and reactions. Cantera supports two different sources of data input: XML and CTI. The data files that were used in this report were CTI. This parser is of rather high-level which means it is more easily readable and creatable than the lower-level XML file. These data files contain coefficients for calculating all the thermodynamic properties that are required to solve the energy equation. Cantera uses "NASA 7-Coefficient Polynomial Parameterization", with data from NASA to do this [6]. When downloading Cantera it comes with various integrated CTI files which are available to use. However the mechanisms one-step, two-step and four-step which were utilised in this report are not part of this (these mechanisms will be discussed in the next chapter). Fortunately, CTI files for the one-step and two-step mechanisms were found on a forum of cerfacs: "The centre of basic and applied research specialized in modelling and numerical simulation" [7] [8]. After checking these files and website for validity, they were downloaded for use in Cantera. Now that thermodynamic, transport, species and reaction properties have been acquired, a system of differential equations remains to be solved.

3.2 SUNDIALS

Cantera provides an integrated ODE solver for solving reaction mechanisms called SUNDIALS [9]. SUNDIALS is equipped with a set of solvers with various numerical methods aiming to provide robust time integrators [10]. It uses a variable time step and offers two methods of returning the solution to the ordinary differential equation that is inserted. In the background the time step is determined and the system of equations is solved for this point in time. Now it is up to the user to decide whether they want the solution at the internal time step determined by SUNDIALS, or a predescribed time step. When using the latter, SUNDIALS still takes the internal time steps up until the predescribed point in time but returns the value in the requested time, interpolating if necessary. These two options are called ONE_STEP and NORMAL respectively. Both are presented in the following figure 3.1 to help visualize their meanings. Both time step options have been used in the created models.

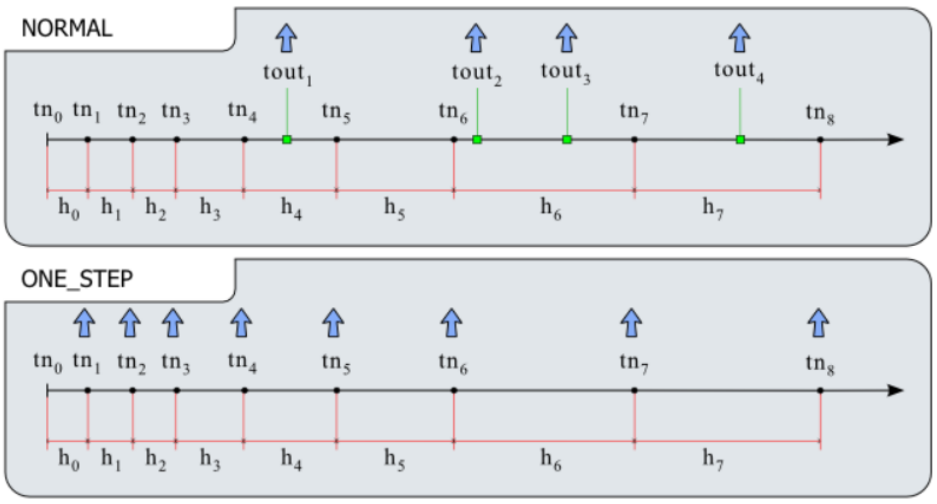


Figure 3.1: SUNDIALS's different time step options, taken from [10]

To visualize the variable time step determined by the SUNDIALS solver, the time was tracked when using the ONE_STEP method within a model (the four-step model which will be discussed in section 4.3, however the exact model is not of importance here). Figure 3.2 shows how much time has passed after taking some amount of steps. The steepness of the curve indicates a tiny step size at the beginning of the simulation. This is necessary in order to determine rapid changes in concentration and temperature right at the start. After some time the reactions within the model are nearing steady state. As the reactions are becoming steadier and thus more predictable, it can be seen that the time step size increases.

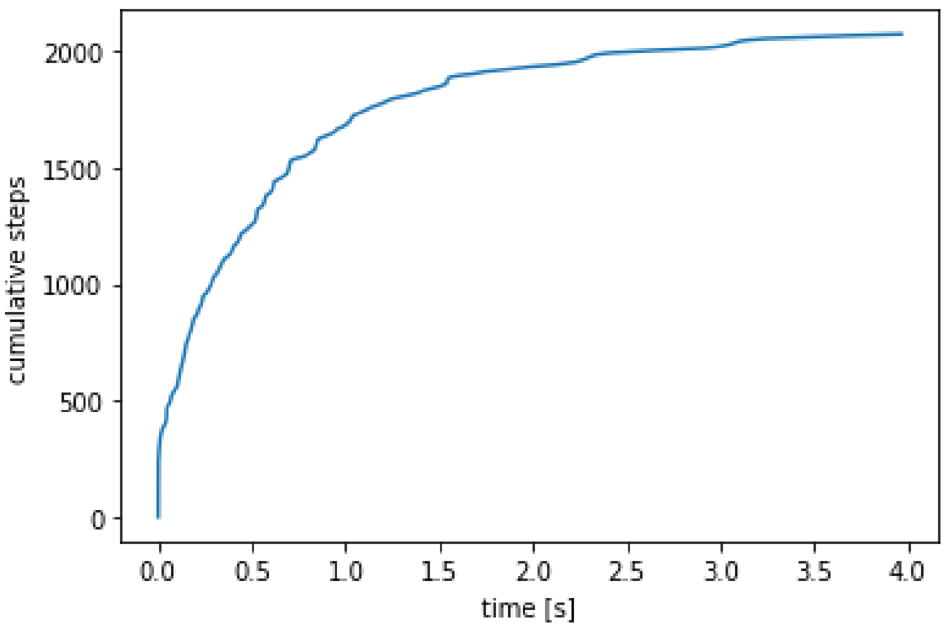


Figure 3.2: Time step size determined by the SUNDIALS solver for a specific model

3.3 Models

Various models have been considered for use within this report. The very first model consisted of a single reactor already containing methane and oxygen, equipped with a very basic global reaction. This model was like an explosion where all the methane and oxygen obviously burned up right away. It was noted that this made the temperature increase all the way up to 4000 to 5000K. The thermodynamic data that was obtained from the CTI file as described in section 3.1, is only experimentally determined for values of 200 to 3500K. By exceeding this limit one would be extrapolating and therefore it was decided to use inlets which propel the fuel into the reactor in order to prevent this sudden explosion. Additionally, using inlets is also closer to reality as new fuel flows into the reactor to heat up new concrete getting poured into the rotary kiln. Now that the chemistry and programs are known, it is time for the actual implementation.

Chapter 4

Global Reaction Models

Combustion of hydrocarbons such as methane consist of many different elementary reactions. With methods such as the quasi-steady state assumption as discussed in chapter 2, these large mechanisms can be reduced to significantly less steps. This simplification is necessary in order to make it computationally more appealing. In this chapter these so-called global reaction mechanisms which are used in the implementation are presented, together with their implementation and results.

4.1 One-step reaction mechanism

The simplest reduced reaction mechanism is governed by only one single step. This global reaction considers the reaction of methane and oxygen directly to carbon dioxide and water.

$$CH_4 + 2O_2 \longrightarrow CO_2 + 2H_2O \tag{4.1}$$

The following values to calculate the rate coefficients were used in this mechanism.

Reaction	A	b	E	Order
(4.1)	$1.1e+10$	0.0	20000	CH_4:1.0, O_2:0.5

Table 4.1: Arrhenius parameters used for the one-step model [11] (units in cm, s, mol and cal/mol).

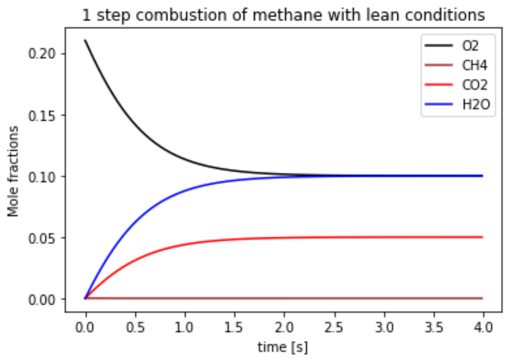
This model has been implemented in Cantera by creating a reactor object filled with air (21% O_2 and 79% N_2). Two inlets have been installed to propel methane and air into this reactor. The amount of gas and air flowing through these inlets depends on the equivalence ratio ϕ, which has been explained in section 2.4. It was decided to use an outlet to an exhaust reservoir since otherwise the mass inside the reactor would keep increasing. Additionally, it was decided to use an initial temperature of 1000K. This homogeneous zero-dimensional reactor is equipped with just one reaction, namely (4.1). Next, the simulation is advanced in time, where the aforementioned reaction with its parameters determine the temperature and concentration of the system. The code for this simulation can be found in Appendix A.1. Additionally, for the reader that is unfamiliar with Cantera the pseudo-code is given below. NORMAL and ONE_STEP functions refer to the SUNDIALS time steps options as discussed in section 3.2.

```
for equivalence ratio 0.5, 1 and 2 do
    initialization of species;
    set correct temperature, pressure and concentration;
    initialization of inlets, outlet and reactor;
    if time step constant then
        set n as number of steps;
        for i=0 to n do
            results[i] $\leftarrow$ temperature, concentrations species;
            advance to time t=$i * (t_{max}/n)$ using NORMAL function;
        end
    else
        while current time $< t_{max}$ do
            append temperature and concentrations species to results;
            advance single time step using ONE_STEP function;
        end
    end
    return results;
end
plot desired results;
```

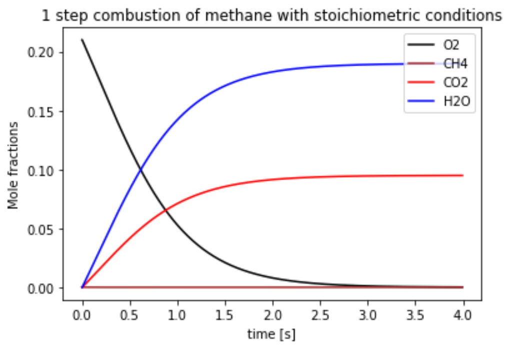
Algorithm 1: Pseudo-code for the single reactor model

The results of this code is presented in figures 4.1 and 4.2. First, figure 4.1a is examined. Fuel-lean combustion conditions means an excess of oxygen is present. The figure reflects this, since methane concentration remains (almost) zero. Any methane that is released into the reactor burns almost instantly because of this overabundance of oxygen. The almost exact halving of oxygen levels is a result of the equivalence ratio being 0.5, since this implies there is twice as much oxygen available than necessary for the combustion reaction. Furthermore, considering no other reactions are present within this reaction mechanism, the products do not react any further. This results in exactly twice as much H_2O production with respect to CO_2 according to reaction (4.1), which is verified in the figure. Next, figure 4.1b will be examined. Equivalence ratio $\phi = 1$ corresponds to stoichiometry and therefore the perfectly balanced air-fuel ratio should imply any methane and air that flows into the reactor will combust. The figure exactly reflects this since concentrations of both air and methane converge to zero. Subsequently, figure 4.1c is examined. As the reactor is filled with air when the simulation starts, it can be seen that at the start any methane that comes into the reactor burns. When the oxygen runs out, methane starts to build up and eventually the concentrations stabilize.

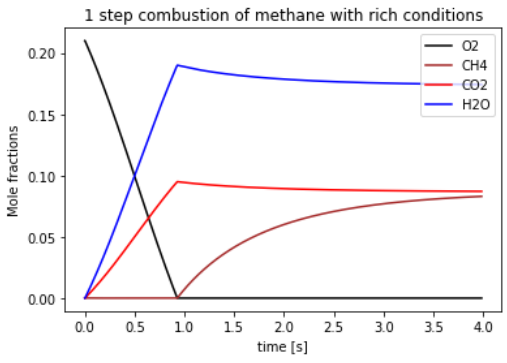
Figure 4.2 represents the temperature evolution inside the reactor for the different equivalence ratios. The high peak temperature under rich conditions can be attributed to the presence of oxygen in the reaction at the start of the simulation. Therefore the combustion conditions are not rich yet, until the initial oxygen is dissipated.



(a) $\phi = 0.5$



(b) $\phi = 1$



(c) $\phi = 2$

Figure 4.1: Concentrations of species in one-step reaction mechanism with several equivalence ratios.

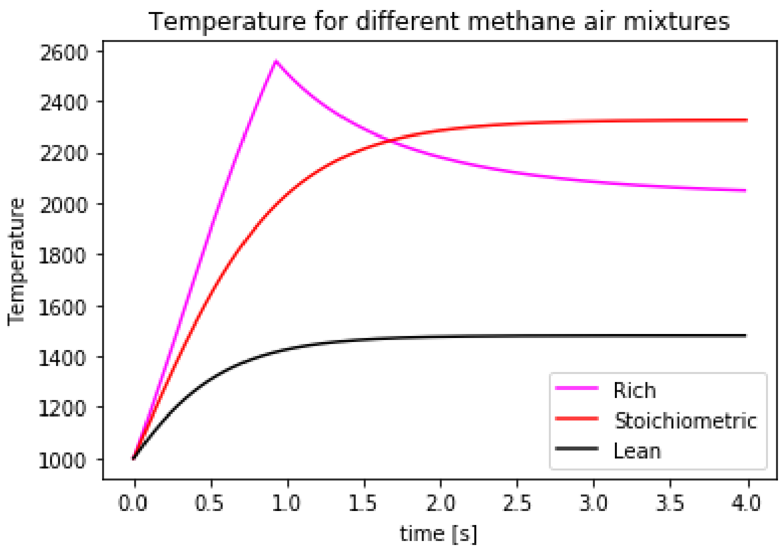


Figure 4.2: Temperature inside the reactor for several equivalence ratios

4.2 Two-step reaction mechanism

The reaction mechanism for this model contains only two reactions. Carbon monoxide is also present within this mechanism. It reads:

$$CH_4 + 1.5O_2 \longrightarrow CO + 2H_2O \tag{4.2}$$
$$CO + 0.5O_2 \rightleftharpoons CO_2 \tag{4.3}$$

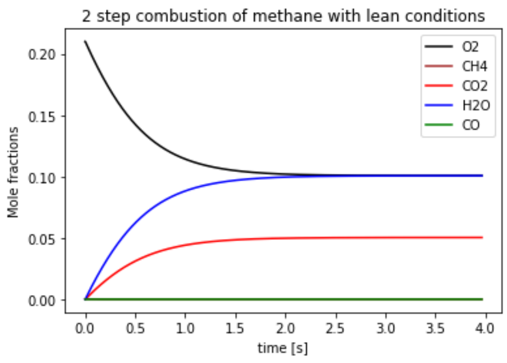
And the values used to calculated the rate laws can be found in table 4.2 below.

Reaction	A	b	E	Order
(4.2)	$4.9e+09$	0.0	35500	CH_4:1.0, O_2:0.5
(4.3)	$2.0e+08$	0.7	12000	-

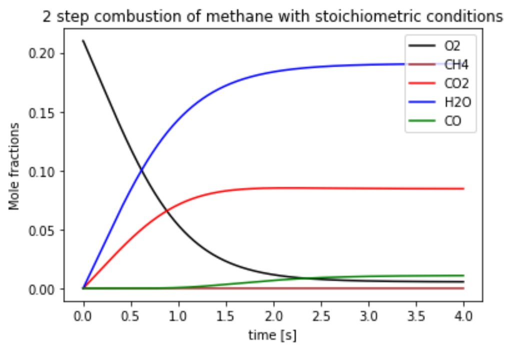
Table 4.2: Arrhenius parameters of 2S_CH4_BFER [12] (units in cm, s, mol and cal/mol).

Implementation of this reaction mechanism has been executed in the same fashion as the implementation of the one-step reaction mechanism from the previous section. The pseudo-code from Algorithm 1 still applies as the setup of the reactor remains the same. Similarly the actual code was realized from altering a few trivial lines from the one-step mechanism, see Appendix A.1. The initialization of the reaction mechanism is different but fortunately, as been discussed in section 3.1, a CTI file has been obtained with data of the required thermodynamic properties.

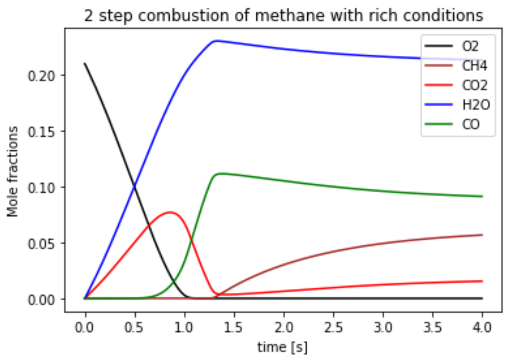
The results of this model can be found in figures 4.3 and 4.4. The first two graphs, 4.3a and 4.3b, have a striking resemblance with the results from the corresponding one-step mechanism. As the carbon monoxide levels within these two models remain close to zero, the impact on the other species is minimal. However the third figure 4.3c with rich combustion conditions differs from its one-step counterpart as carbon monoxide was introduced to the reaction mechanism. The high carbon monoxide levels demonstrate the dangers of low oxygen combustion in every day life, e.g. a fireplace. Figure 4.4 represents the temperature evolution of the two-step reaction mechanism. Similarly to its one-step equivalent, fuel-rich combustion results in a relatively high temperature at first as there is an abundance of oxygen available. Comparing this temperature evolution with the one-step mechanism, the only notable difference would be the smoothness of the fuel-rich conditions curve.



(a) $\phi = 0.5$



(b) $\phi = 1$



(c) $\phi = 2$

Figure 4.3: Concentration of species in two-step reaction mechanism with several equivalence ratios.

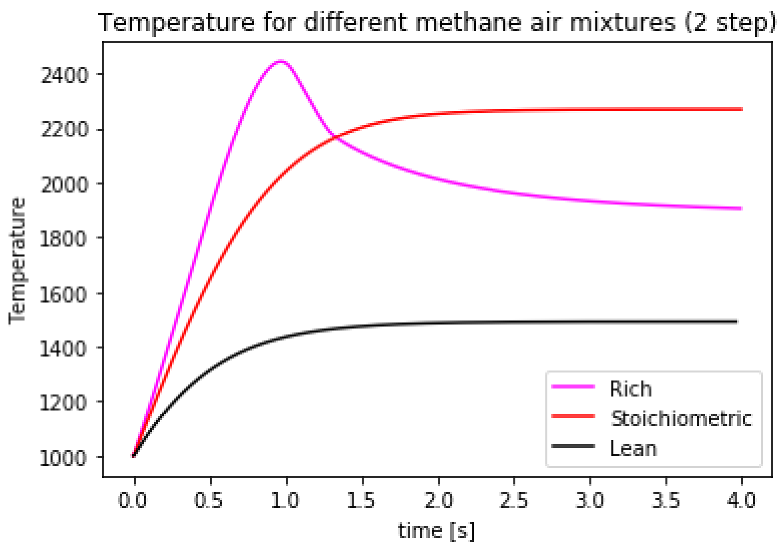


Figure 4.4: Temperature inside the reactor for several equivalence ratios

4.3 Four-step reaction mechanism

This global reaction mechanism was taken from Jones and Lindstedt (1988). There are six species present in this reaction, which are part of the four reactions. Below this system is presented, together with the parameters for the Arrhenius equation.

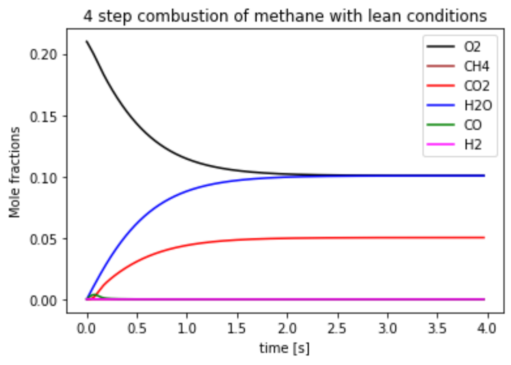
$$\mathrm{CH_4 + 0.5O_2 \longrightarrow CO + 2H_2} \tag{4.4}$$
$$\mathrm{CH_4 + H_2O \longrightarrow CO + 3H_2} \tag{4.5}$$
$$\mathrm{CO + H_2O \rightleftharpoons CO_2 + H_2} \tag{4.6}$$
$$\mathrm{H_2 + 0.5O_2 \rightleftharpoons H_2O} \tag{4.7}$$

Reaction	A	b	E	Order
(4.4)	7.82e13	0.0	29900	CH_4:0.5, O_2:1.25
(4.5)	3.00e11	0.0	29900	CO:1.0, O_2:1.0
(4.6)	2.75e12	0.0	20000	-
(4.7)	1.79e13	0.0	34900	-

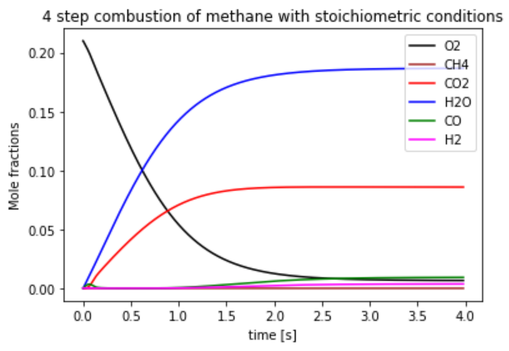
Table 4.3: Arrhenius parameters used for the four-step mechanism [13]

Yet again this global reaction mechanism is implemented in Python with the Cantera software package. The CTI file for this mechanism was acquired by modifying the two-step counterpart manually. Reactions (4.4) through (4.7) and the parameters from table 4.3 were inserted into the file. Next, the implementation was accomplished once again by utilizing the pseudo-code from Algorithm 1, since the remainder of the setup remains the same as the previous mechanisms. The entire code for this model is omitted from the Appendix as well; it is obtained by small trivial changes to the one-step model, see Appendix A.1.

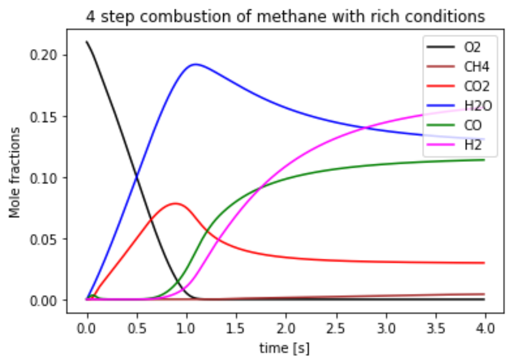
Results from this model are displayed in figures 4.5 and 4.6. As one would expect the majority of these graphs coincide with the two-step model. A notable difference would be the small bump of carbon monoxide right at the start of the simulation. This bump excellently portrays the characteristics of a highly reactive species such as CO. Conversely, in figure 4.5c when oxygen runs out the carbon monoxide levels do rise rather quickly. Finally the temperature evolution of figure 4.6 is examined. For this mechanism the temperature of fuel-rich conditions is notably lower than its two-step equivalent. This difference is of importance later on, and thus both the two-step and the four-step mechanism should be considered when moving on.



(a) $\phi = 0.5$



(b) $\phi = 1$



(c) $\phi = 2$

Figure 4.5: Concentration of species in four-step reaction mechanism with several equivalence ratios.

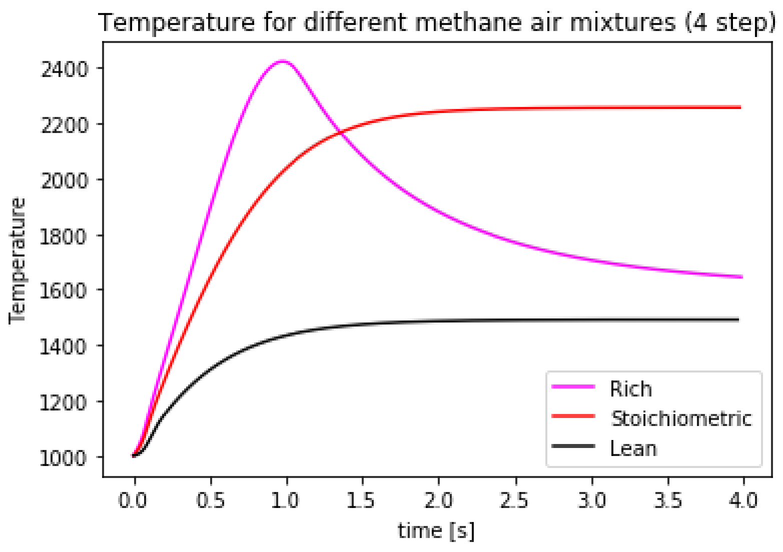


Figure 4.6: Temperature inside the reactor for several equivalence ratios

Chapter 5

One-dimensional Flow Model

Up until now a single reactor with homogeneous contents was considered. Cement rotary kilns are massive cylinders and therefore assuming that this reactor is well-stirred would be a rather large simplification. This chapter will propose a model where several reactors are linked together in order to simulate flow through a one-dimensional reactor. Thus each individual reactor represents a fraction of the kiln, where it tracks the concentrations and temperature inside. Within this model a rather simple system is used to solve the flow for one dimension, namely the scalar convection-diffusion equation.

5.1 Convection-diffusion equation

As its name might suggest, a convection-diffusion problem consists of two parts: convection and diffusion. Convection is the movement of fluids caused by the hotter parts moving up and the colder parts moving down. Diffusion is the process where species flow from highly concentrated areas to the lesser concentrated areas. Together they describe the transport of a pollutant in a certain medium. In this case the transport of a fuel-mix in a reactor. Before the reaction mechanisms of the previous chapter can be implemented, a way of transport of the fuel-mix inside the reactor has to be investigated.

The scalar convection-diffusion equation looks rather simple. A one-dimensional rod with $0 \leq x \leq 1$ is considered. It represents the rotary kiln where the inlet is located at $x = 0$, and the reactor ends at $x = 1$. $u(x)$ represents the speed of the fuel-mix at place x of the reactor. The assumption that the fuel flow at the inlet is constant gives the first boundary condition (5.2). The flow out of the reactor is taken to be 0, which is represented by (5.3). This yields a convection-diffusion equation with Dirichlet boundary conditions.

$$-\epsilon \frac{d^2u}{dx^2} + \frac{du}{dx} = 0 \tag{5.1}$$

$$u(0) = 1 \tag{5.2}$$

$$u(1) = 0 \tag{5.3}$$

The rod is divided into a grid with N subintervals of length $h = 1/N$ as can be seen in figure 5.1, which also brings a new notation $x_i = ih$. u_i represents the solution u at grid point i, or $u(x_i)$. Central differences is used to make a numerical approximation of this system [14]. The second divided difference appromixation to $\frac{d^2u}{dx^2}$ together with the central divided difference of $\frac{du}{dx}$ yields:

$$-\epsilon\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+\frac{u_{i+1}-u_{i-1}}{2h}=0 \tag{5.4}$$

with local truncation error $\mathcal{O}(h^2)$ for both approximations.

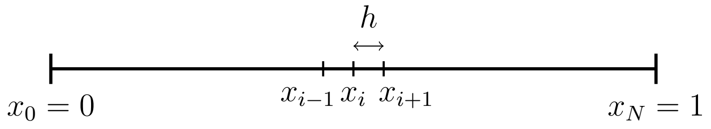


Figure 5.1: One-dimensional rod with N grid points of size h

Subsequently, equation (5.4) is applied to every internal node of the interval. This system of equations is split into three parts, since $u_0 = 1$ and $u_N = 0$ reside in the first and last equation. Substituting these gives:

$$\begin{cases} -\epsilon\frac{u_2-2u_1+1}{h^2}+\frac{u_2-1}{2h}=0, & \text{for} \quad i=1 \\ -\epsilon\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+\frac{u_{i+1}-u_{i-1}}{2h}=0, & \text{for} \quad 2\le i\le N-2 \\ -\epsilon\frac{0-2u_{N-1}+u_{N-2}}{h^2}+\frac{0-u_{N-2}}{2h}=0, & \text{for} \quad i=N-1 \end{cases} \tag{5.5}$$

Rewriting this system to matrix-vector notation yields $A\mathbf{u} = \mathbf{f}$, where:

$$A=\frac{1}{h^2}\begin{pmatrix} 2\epsilon & -\epsilon+\frac{h}{2} & 0 & \dots & 0 \\ -\epsilon-\frac{h}{2} & 2\epsilon & -\epsilon+\frac{h}{2} & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & -\epsilon-\frac{h}{2} & 2\epsilon & -\epsilon+\frac{h}{2} \\ 0 & \dots & 0 & -\epsilon-\frac{h}{2} & 2\epsilon \end{pmatrix},$$

$$\mathbf{u}=\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}, \qquad \text{and} \quad \mathbf{f}=\frac{1}{h^2}\begin{bmatrix} \epsilon+\frac{h}{2} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

As the fuel travels through the inlet into the rotary kiln its flow can vary greatly. If the flow of a fluid moves smoothly in a near straight line without any mixing, it is called laminar. After some point in time, a series of events take place which radically change the flow character. As it hits resistance the flow becomes chaotic and unsteady. This is called a turbulent flow [3]. A great example in everyday life is the smoke of a candle. Figure 5.2 shows a photograph of the transition of a candle plume from laminar to turbulent.

The moment and distance where this transition takes place is dependant of the Reynolds number. This value gives a measure to where these forces take over. It is

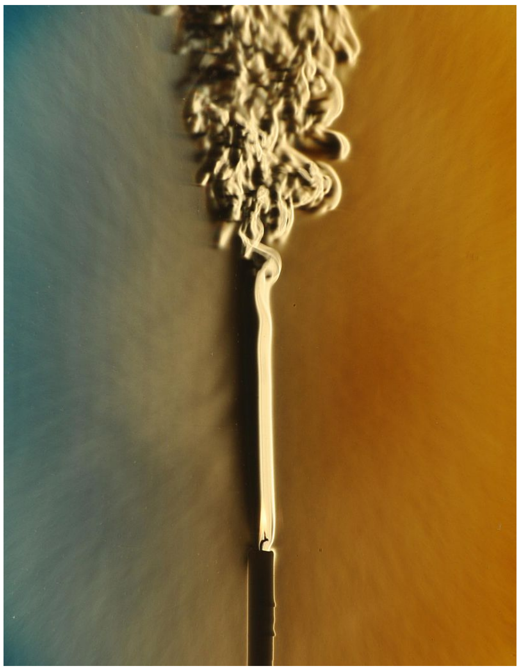


Figure 5.2: The transition of laminar to turbulent flow of a candle flume [15]

dependant of flow speed u, diameter of the tube D and the kinematic viscosity of the fluid ν.

$$\text{Re} = \frac{uD}{\nu} \tag{5.6}$$

The kinematic viscosity of a gas mixture is not easily calculated (See [3]). Since simplifications have been made the exact Reynolds number would not provide satisfactory results. The Reynolds number is related to ϵ, where the latter describes the relation of convection and diffusion. Therefore it was decided that the parameter ϵ will be varied as to investigate the effects on the system.

The implementation to solve the convection-diffusion equation is rather straightforward once A, **u** and **f** are known. The code written in Python can be found in Appendix A.2.

5.2 Reactor Network

Now that the flow speed based on convection-diffusion for one dimension has been established, it has yet to be implemented. To create this one-dimensional reactor including flow, a set of zero-dimensional reactors are chained together as a reactor network. The flow from one reactor to the other is based on its place inside the network. Inserting this place value into the convection-diffusion equation from previous section returns the flow at the corresponding place. To give a simple example, a network with 10 reactors is considered. If the convection-diffusion equation has been solved for N=1000, then this would mean the flow from the first reactor to the second reactor occurs at $x = 100/1000 = 0.1$, see figure 5.1. This implies the flow based on convection-diffusion is u_{100} which can easily be extracted from the solution.

The implementation of this reactor network proved to be more complicated than initially thought. At first, the idea was to implement it as an actual chain of reactors, connecting them together with Cantera's MassFlowController objects which simulate a flow. The implementation of such a network took some time and as time passed more and more problems surfaced. For some problems solutions were found, but would also increase the complexity of the problem. Such a network where several reactors depend on the contents of the other, which have to be integrated simultaneously across the entire network, are more prone to errors like unphysical values or discontinuities. After many failed attempts this implementation of the reactor network model was discarded. Fortunately, a new approach was discovered based on an example from the Cantera website [16]. Instead of all the reactors linked together inserted into the reactor network object, the reactor network object contains only one reactor which loops from the first reactor to the last. Every iteration the reactor is advanced to its steady state and its contents and state are saved. In the next iteration this saved state is loaded into the reservoir which feeds the reactor, and it is also advanced to steady state, and so on. This way every inlet composition is fixed at the composition of the reactor immediately upstream. One of the downsides of this implementation is the fact that only steady state conditions can be monitored. It seems possible to extend this script to also track intermediate values with some effort, however was not considered yet as steady state values were sufficient for now. Furthermore, implementation of two inlets with this approach seemed to make matters more complex than necessary. It was decided to use one inlet which propels a gas mixture of methane and air into the first reactor. The method above was implemented in Python with the Cantera module, the pseudo-code is found below in Algorithm 2.

```
for equivalence ratio 0.5, 1 and 2 do
    initialization of species;
    set correct temperature, pressure and concentration;
    call convection-diffusion module;
    initialization of inlet, outlet and reactor;
    define information vectors;
    set n as number of reactors;
    for i=0 to n do
        state ← current state;
        update inlet reservoir to state;
        compute and set mass flow rate according to place;
        advance to steady state;
        results[i] ← current state;
    end
    return results;
end
plot desired results;
```

Algorithm 2: Pseudo-code for the chain of reactors model

For this model the initial flow speed of the inlet to the reactor has to be known in order to calculate the convection-diffusion equation. Additionally, the mass flow rate has to be specified for the MassFlowController objects to determine the mass flow of contents from one reactor to the next. Its SI unit is kg/s and the formula reads:

$$\dot{m} = \rho \cdot v \cdot A \tag{5.7}$$

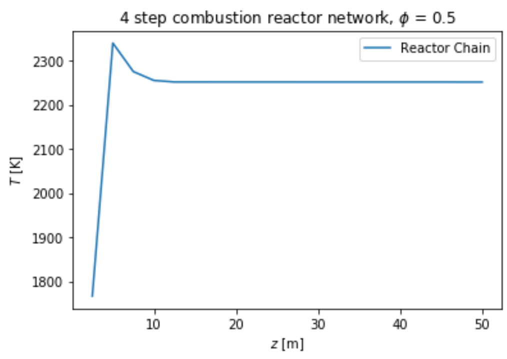
where ρ represents the fluid density, v the flow speed and A the cross-sectional area. Luckily Cantera is able to compute the density of the current fluid. Since the values for area and initial flow speed can't be ignored, it was decided to use approximate values for these. Subsequently, other values like the length of the rotary kiln suddenly become important now as well. The exact values are not as important, but approximating the ratio of them is. If this ratio would deviate orders of magnitude it could affect the results of the system. The following values were used within this model, see table 5.1. Lastly, it was decided to use 20 reactors for this network (so n=20 in algorithm 2). More reactors would just increase the line segment indicating steady state whereas less reactors would not reach this steady state. The code for the model which is described in the previous two sections can be found in Appendix A.3.

Name	Value	Unit
Length reactor	50	m
Diameter reactor	2	m
Initial flow speed	5	m/s

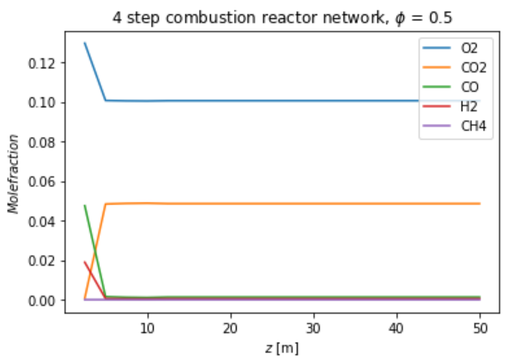
Table 5.1: Values used for this reactor network model

5.3 Results

The results are presented in figures 5.3, 5.4 and 5.5 below. Please note that the x-axis is labelled distance instead of time now. These results might seem a bit dull, since steady state values for different reactors inside the network wont vary much. Starting off with examining figure 5.3, the peak at the start of the reactor indicates the peak temperature at the heart of the flame, as one would expect. Before this peak the temperature is relatively lower, where some fuel combusts but most part flows along the kiln. Moving along to the concentrations subfigure 5.3b, lean combustion keeps some characteristics from its single reactor model. Right at the start of the reactor some CO is present which dissipates as an abundance of oxygen is still available in the next section, where CO_2 is created. Next, figure 5.4 for stoichiometric combustion conditions is taken into consideration. Here the argumentation for the rise and drop of temperature at the start of the reactor is similar to the last figure. Subfigure 5.4b shows that the only species which is not present anymore is methane. Note that H_2O was omitted in this graph for clarity reasons, its mole fraction was about 0.18 throughout the reactor. At first it might seem peculiar that both CO and H_2O are present within the system, since they react with each other according to this four-step mechanism. However this quite nicely portrays what chemical equilibrium is. Remember that both reactions (4.6) and (4.7) are reversible. Methane being the only species which is only present in the irreversible reactions, dissipates at the steady state. Finally figure 5.5 is examined. The temperature throughout the reactor is relatively lower than with other equivalence ratios. This can partly be attributed to the fact that there is a lack of oxygen. Another consequence of this is the rather high level of carbon monoxide.

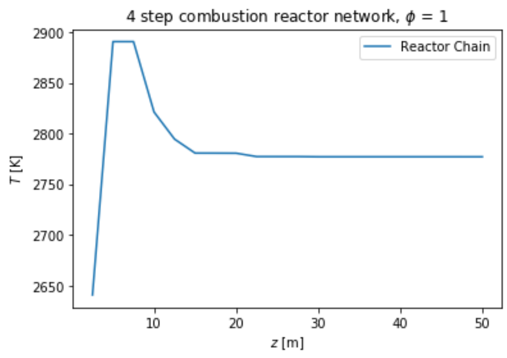


(a) Temperature

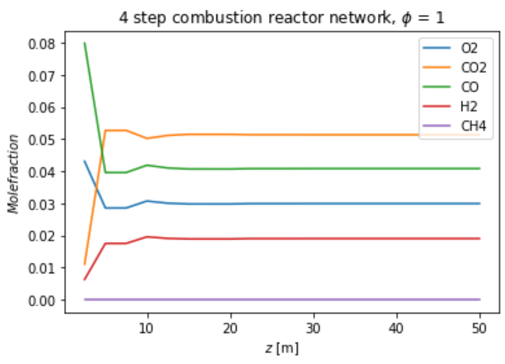


(b) Mole fractions

Figure 5.3: Obtained results for the four-step reactor network model, for $\phi = 0.5$

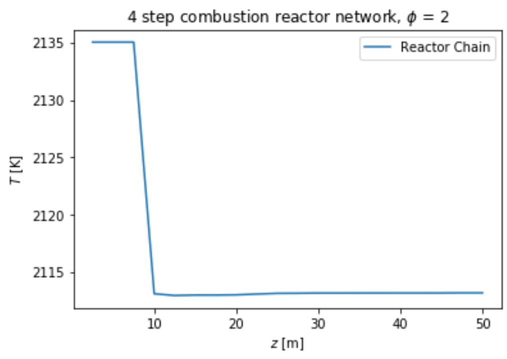


(a) Temperature

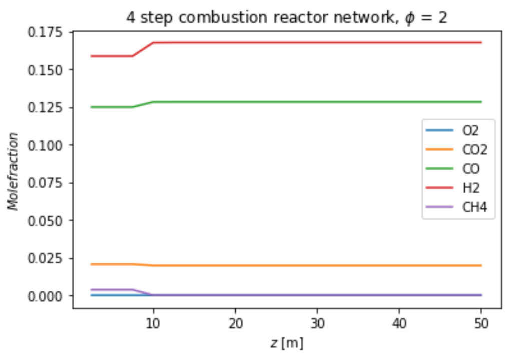


(b) Mole fractions

Figure 5.4: Obtained results for the four-step reactor network model, for $\phi = 1$



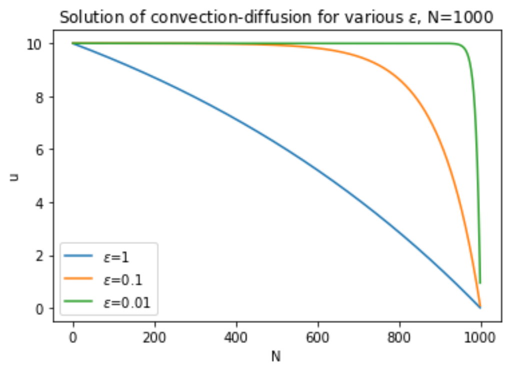
(a) Temperature



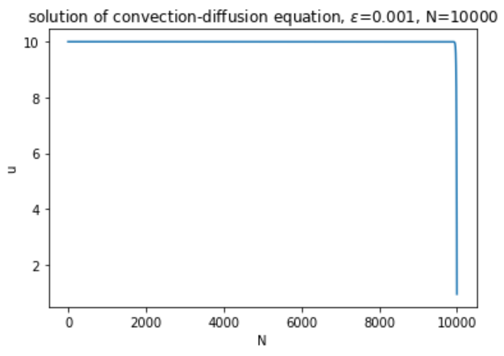
(b) Mole fractions

Figure 5.5: Obtained results for the four-step reactor network model, for $\phi = 2$

Additionally, different values for ϵ were evaluated. ϵ describes the relation between convection and diffusion, as been described in section 5.1. Its influence on the flow speed throughout the reactor is presented in figure 5.6 below. It represents the flow speed u as a solution of the convection-diffusion equation, plotted against the number of grid point steps N. It was decided to use more grid steps for such a small ϵ as in figure 5.6b to get more accurate results nearing the end of the grid. Thus it was plotted separately.



(a)



(b)

Figure 5.6: Solution **u** of the convection-diffusion equation plotted against total steps N

Next, the reactor network temperature has been plotted to investigate the impact of different ϵ values. It was expected to show a clear distinction in temperature as a lower value for epsilon indicates more convection and thus a relatively faster downstream. This should in turn lead to a higher temperature nearing the end of the reactor as it flows further into the reactor before cooling down. Unfortunately, this is not what was found. Figure 5.7 shows the results of plotting these temperature against various epsilon values. The expected relation was not found. It is suspected the reason for this is hides within implementation as the reactor network is solved iteratively for every reactor, a higher flow from the inlet results in more mass flowing into current reactor. An increase in mass results in a larger spread of chemical energy.

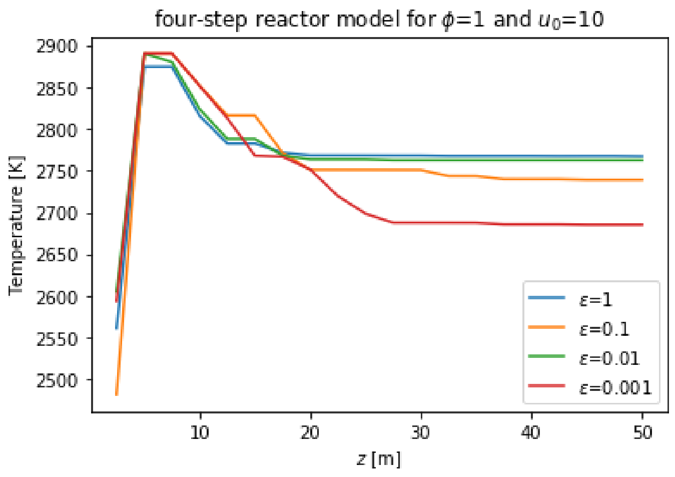


Figure 5.7: Network model for various ϵ values

Chapter 6

Oxygen-Enhanced Combustion

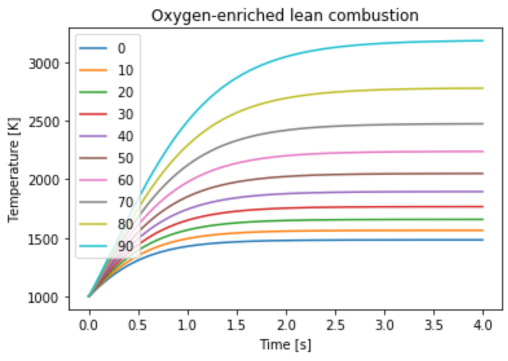
After creating various models of increasing complexity, it is time to investigate the impact of enhancing oxygen levels on NO_x emissions. Oxygen enhanced combustion is the addition of pure oxygen to the air that is propelled into the reactor. This way the flow of air contains more than the 21 percent of oxygen it normally carries. Naturally this can be done for increasing amounts of oxygen, all the way up to inserting pure oxygen into the reactor. In theory, if one would use pure oxygen as oxidizer no NO_x would be produced. However oxygen is rather expensive, and besides that the kiln would have to be completely airtight. Therefore the previously discussed models will be altered such that the percentage of oxygen that flows into the reactor will be increased.

6.1 Implementation

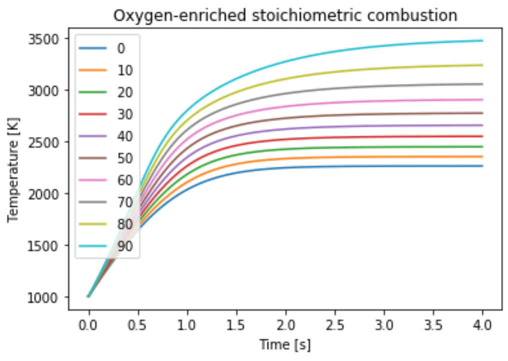
Increasing the percentage of oxygen present in the air that flows into the reactor implies less air is needed for combustion, since the amount of oxygen that flows into the reactor should remain constant in order to maintain its stoichiometric property. This in turn implies that the effect of enrichment would actually be reducing the percentage of nitrogen in the air flow. So to realize a model with oxygen injection, all there is left to do is rerun the models with incrementally decreased nitrogen and plot the results. This was accomplished with the following definition of the mole fraction composition for the contents of the air inlet in the model.

```
composition = {'O2':1, 'N2':((1-injection)*3.76)}
```

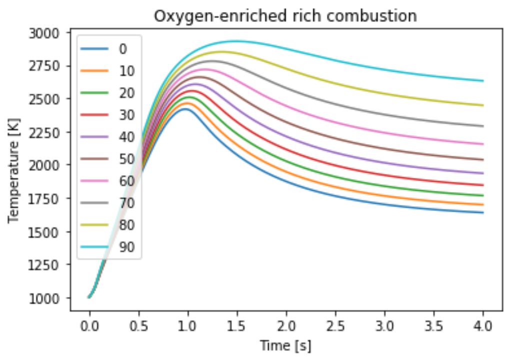
Here the variable *injection* is varied from 0 to 1, where 0 indicates normal air without oxygen injection, and 1 would imply pure oxygen. It was decided to use increments of 0.1. The Python code for this implementation can be found in Appendix A.4 and A.5, although it is rather similar to the code of the corresponding models. Results of various methods with this implementation can be found in figure 6.1. Note that subfigure 6.1a, 6.1b and 6.1c have different global reaction mechanisms with various combustion conditions. Other combinations are omitted as the gist of those remain the same.



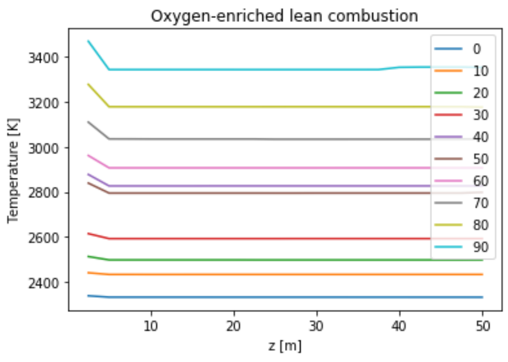
(a) Lean one-step mechanism



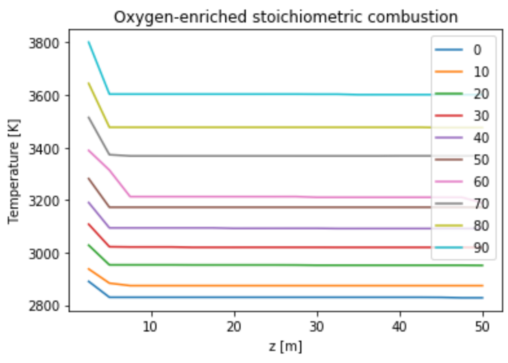
(b) Stoichiometric two-step mechanism



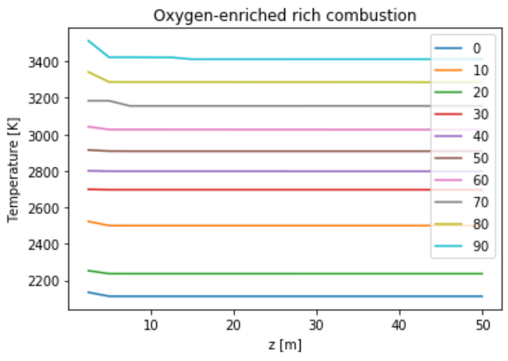
(c) Rich four-step mechanism



(d) Reactor network, lean conditions



(e) Reactor network, stoichiometric conditions



(f) Reactor network, rich conditions

Figure 6.1

6.2 Results

Formation of NO_x can mostly be attributed towards three mechanisms: prompt, fuel and thermal. For temperatures above 1400K thermal NO_x production is the dominant factor as this process exponentially increases with temperature. The mechanism which describes this is called the Zeldovich mechanism [17]. The aforementioned exponential relation of thermal NO_x and temperature indicate that higher peak temperatures increase NO_x emissions. Results from 6.1 show that any percentage oxygen-enriched combustion leads to a higher temperature. The reason being that a constant methane and oxygen flow with a decreased nitrogen flow results in less mass entering the reactor. Additionally, the mass flow of methane and oxygen remain the same yielding the same chemical energy from the combustion reaction. The consequence of the above two statements is that the same chemical energy will be spread across a smaller total thermal mass of gas which leads to an increase in temperature. This final reasoning was vital to understanding the results and was aided by a correspondence, see Appendix B.

For the production process of cement the temperature within the rotary kiln should remain the same as before. An increase in temperature which comes along with oxygen-enhanced combustion therefore implies less fuel should be used to maintain the same temperature within the kiln. Furthermore, less N_2 available in the kiln results in smaller or equal NO_x production, when keeping in mind the temperature remains constant. Therefore the presented results imply oxygen-enriched combustion does not only reduce NO_x emissions, it also increases fuel efficiency. Other literature concludes the same reduction in NO_x emission for oxygen-enhanced combustion [18].

Chapter 7

Conclusion

Researching cleaner combustion is getting more traction by researchers as the consequences of NO_x emissions are becoming more transparent over the past few decades. Oxygen-enhanced combustion could be part of the solution. In this report the effects of oxygen-enhanced combustion were simulated through various models with increasing difficulty. It was found that this led to an increase in temperature for every single model, while keeping the methane and oxygen inflow constant. This implies the rotary kiln could be heated up to the same temperature but with less fuel, increasing fuel efficiency. A higher concentration of supplied oxygen means a lower concentration of nitrogen entering the reactor, which in turn implies the formation of NO_x is decreased. So to summarize, the results found in this report imply oxygen-enriched combustion decreases the formation of NO_x while increasing fuel efficiency.

It should however be noted that this is an implication, and further research with direct calculations of NO values could describe a more precise calculation. The GRI3.0 mechanism is a complex mechanism containing 325 elementary reactions. Expanding the model to this complicated system seems like the next logical step. As it also contains various reactions containing nitrogen, it seems like a rather promising expansion to the models presented within this report.

Bibliography

[1] *Rijksinstituut voor Volksgezondheid en Milieu.* URL: `https://www.rivm.nl/stikstof`.

[2] Jurgen Warnatz, Ulrich Maas, and Robert W. Dibble. *Combustion: Physical and Chemical Fundamentals, Modelling and Simulation, Experiments, Pollutant Formation.* 1996.

[3] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics.* 1995.

[4] *Cantera Official Website.* URL: `cantera.org`.

[5] David G. Goodwin et al. *Cantera: An Object-oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes.* Version 2.4.0. URL: `https://cantera.org/science/reactors.html`.

[6] Bonnie J. McBride, Sanford Gordon, and Martin A. Reno. *Coefficients for Calculating Thermodynamic and Transport Properties of Individual Species.* 1993.

[7] *Centre Européen de recherche et de formation avancée en calcul scientifique.* URL: `https://cerfacs.fr/en/`.

[8] *Cerfacs forum.* URL: `https://www.cerfacs.fr/cantera/mechanisms/meth.php`.

[9] Alan C Hindmarsh et al. “SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers”. In: *ACM Transactions on Mathematical Software (TOMS)* 31.3 (2005), pp. 363–396.

[10] Lawrence Livermore National Laboratory. *SUNDIALS website.* URL: `https://computing.llnl.gov/projects/sundials`.

[11] C.K. Westbrook and F.L. Dryer. “Simplified Reaction Mechanisms for the Oxidation of Hydrocarbon Fuel in Flames”. In: *Combustion Science and Technology* 27.1-2 (1981), pp. 31–43.

[12] B. Franzelli et al. “Large-Eddy Simulation of combustion instabilities in a lean partially premixed swirled flame”. In: *Combustion and Flame* 159.2 (2012), pp. 621–637.

[13] W.P.Jones and R.P.Lindstedt. “Global reaction schemes for hydrocarbon combustion”. In: *Combustion and Flame* 73.3 (1988), pp. 233–249.

[14] J. van Kan et al. *Numerical Methods for Partial Differential Equations.* 2019.

[15] Wikipedia contributors. *Laminar-Turbulent transition.* accessed 12 June 2020. 10 March 2009. URL: `https://en.wikipedia.org/`.

[16] *Cantera Official Website.* URL: `https://cantera.org/examples/python/reactors/pfr.py.html`.

[17] Y.B. Zeldovich. “The Oxidation of Nitrogen in Combustion and Explosions: Acta”. In: *Acta Physiochemica* (1946).

[18] Charles E. Baukal Jr. *Oxygen-Enhanced Combustion.* 2nd ed. 2013.

Appendix A

Python Code

A.1 One-step mechanism

```
import numpy as np
import matplotlib.pyplot as plt
import cantera as ct

"""
Function for 1 step methane combustion
input:
    string equiv_rat - either lean, stoichiometric or rich mixture
output:
    array result - 6 column array with time, concentrations and temperature
    for each time step
        column 1 time,
        column 2:5 concentrations O2, CH4, CO2, H2O
        column 6 temperature
"""
def onestep_fun(equiv_rat):
    integration = "internal" #constant or internal time step
    # Initialize our methane solution and set the temperature, pressure
    # and concentration
    solu = ct.Solution('1step_CH4.cti')
    solu.TPX = 300, 101325, 'CH4:1'
    molweight_m = solu.mean_molecular_weight/1000   #kg/mol

    # Initialize our air solution and set the temperature, pressure
    # and concentration
    lucht = ct.Solution('1step_CH4.cti')
    lucht.TPX = 300, 101325, 'O2:0.21, N2:0.79'
    molweight_l = lucht.mean_molecular_weight/1000  #kg/mol

    # Create the reservoir objects with corresponding solutions
    inlet_methane = ct.Reservoir(solu)
    inlet_lucht = ct.Reservoir(lucht)
    exhaust = ct.Reservoir(solu)

    # Create our combustion chamber, an IdealGasReactor object
    solu.TPX = 1000, 101325, 'O2:0.21, N2:0.79'
    combustor = ct.IdealGasReactor(solu)

    # Create MassFLowControllers with mass flow rate such that it corresponds
    # to lean, stoichiometric or rich fuel conditions. [kg/s]
    equiv_rat_dict = {"rich":1, "stoichiometric":2, "lean":4}
    equiv_rat_help = equiv_rat_dict[equiv_rat]
    mfc_m = ct.MassFlowController(inlet_methane, combustor,
        mdot = molweight_m)
mfc_l = ct.MassFlowController(inlet_lucht, combustor,
        mdot = molweight_l*equiv_rat_help/0.21)

# Create an exhaust outlet and a reactornetwork
outlet = ct.Valve(combustor, exhaust, K=10)

sim = ct.ReactorNet([combustor])

time = 0
tmax = 4

if integration == "constant":
    steps = 100
    timestep = tmax/steps
    result = np.zeros((steps,6))
elif(integration == "internal"):
    sim.set_initial_time(0)
    sim.set_max_time_step(tmax)
    result = np.zeros((0,6))

print('{0:>14s} {1:>14} {2:>14} {3:>14s} {4:>14s}'.format(
    'Temperature', '[O2]', '[CH4]', '[CO2]', '[H2O]'))


    # next time step, constant or the internal time step given
    # by Cantera
if integration == "constant":
    for i in range(steps):
        print('{0:14.5f} {1:14.5f} {2:14.5f} {3:14.5f} {4:14.5f}'
            .format(
            combustor.T, combustor.thermo['O2'].X[0],
            combustor.thermo['CH4'].X[0],
            combustor.thermo['CO2'].X[0],
            combustor.thermo['H2O'].X[0]))

        result[i,0] = time
        result[i,1:5] = combustor.thermo['O2', 'CH4', 'CO2', 'H2O'].X
        result[i,5] = combustor.T
        time += timestep
        sim.advance(time)
else:
    while time < tmax:
        print('{0:14.5f} {1:14.5f} {2:14.5f} {3:14.5f} {4:14.5f}'
            .format(
            combustor.T, combustor.thermo['O2'].X[0],
            combustor.thermo['CH4'].X[0],
            combustor.thermo['CO2'].X[0],
            combustor.thermo['H2O'].X[0]))

        newres = [[time, combustor.thermo['O2'].X[0],
                combustor.thermo['CH4'].X[0],
               combustor.thermo['CO2'].X[0],
               combustor.thermo['H2O'].X[0],
               combustor.T]]
        result = np.append(result, newres, axis=0)
        time = sim.step()

plt.plot(result[:,0], result[:,1], 'black', label="O2")
plt.plot(result[:,0], result[:,2], 'brown', label="CH4")
plt.plot(result[:,0], result[:,3], 'red', label="CO2")
plt.plot(result[:,0], result[:,4], 'blue', label="H2O")
plt.legend()
```

```
    plt.xlabel('time [s]')
    plt.ylabel('Mole fractions')
    plt.title('1 step combustion of methane with ' +
            equiv_rat + ' conditions')
    plt.show()

    return result


res_lean = onestep_fun("lean")
res_stoi = onestep_fun("stoichiometric")
res_rich = onestep_fun("rich")

plt.plot(res_rich[:,0], res_rich[:,5], 'magenta', label="Rich")
plt.plot(res_stoi[:,0], res_stoi[:,5], 'red', label="Stoichiometric")
plt.plot(res_lean[:,0], res_lean[:,5], 'black', label="Lean")

plt.legend()
plt.xlabel('time [s]')
plt.ylabel('Temperature')
plt.title("Temperature for different methane air mixtures")
```

A.2 Convection-diffusion function

```
import numpy as np
import matplotlib.pyplot as plt
import math

"""
Function to compute the speed of the rotary kiln's contents.
Calculated using Finite Difference Method on scalar
convection-diffusion equation u'(x) = epsilon*u''(x) with
Dirichlet or Robin boundary conditions, and 0 < x < 1.
"""

def convdifffun(u0, vaten):
    # vaten = amount of reactors in the chain
    # u0 = initial flow speed at inlet
    c=u0
    N = 1000
    h = 1/N
    epsilon = 0.1

    A = np.zeros((N-1,N-1))
    f = np.zeros((N-1,1))

    for i in range(N-1):
        A[i,i] = 2*epsilon
        if(i != N-2):
            A[i,i+1] = -1*epsilon + h/2

        #Dirichlet
        if(i != 0):
            A[i,i-1] = -1*epsilon - h/2

    f[0] = epsilon*c + (h*c)/2

    f = (1/(h**2))*f
    A = (1/(h**2))*A
    u = np.linalg.solve(A, f)

    # Return values depending on the amount of barrels
    xvat = N/vaten
    res = np.zeros(vaten)
    for i in range(vaten):
        temp = math.floor((i+1)*xvat)-2
        res[i] = u[temp]

    return res
```

A.3 Reactor Network

```
import numpy as np
import matplotlib.pyplot as plt
import cantera as ct
import math
from convdiff import convdifffun    #manually made function

T_0 = 1000.0                # inlet temperature [K]
pressure = ct.one_atm       # constant pressure [Pa]
composition_0 = 'CH4:1, O2:2, N2:7.52'
length = 50                 # approximate reactor length [m]
u_0 = 10                    # inflow velocity [m/s]
area = math.pi*(2/2)**2     # cross-sectional area [m**2]
n_steps = 20                # amount of reactors in the chain

# import the gas model and set the initial conditions
# 4 step:
solu = ct.Solution('4step_CH4.cti')
# 2 step:
# solu = ct.Solution('2step_CH4.cti')

solu.TPX = T_0, pressure, composition_0
equiv_rat = 1
solu.set_equivalence_ratio(equiv_rat, 'CH4', 'O2:2, N2:7.52')
condif = convdifffun(u_0, n_steps)
mass_flow_rate = u_0 * solu.density * area

dz = length / n_steps
r_vol = area * dz

# create a new reactor
r = ct.IdealGasReactor(solu)
r.volume = r_vol

# Create reservoirs for up and downstream
upstream = ct.Reservoir(solu, name='upstream')
downstream = ct.Reservoir(solu, name='downstream')

# The mass flow rate into the reactor will be fixed over time
# by using a MassFlowController object.
m = ct.MassFlowController(upstream, r, mdot=mass_flow_rate)

v = ct.PressureController(r, downstream, master=m, K=1e-5)

# Prepare the solver
sim = ct.ReactorNet([r])
```

```
sim.rtol = 1e-5
sim.atol = 1e-12
sim.set_max_time_step(1)
sim.max_err_test_fails = 100

# define time, space, and other information vectors
z = (np.arange(n_steps) + 1) * dz
t_r = np.zeros_like(z)  # residence time in each reactor
u = np.zeros_like(z)
t = np.zeros_like(z)
states = ct.SolutionArray(r.thermo)

# iterate through the network
for i in range(n_steps):
    # Set the state of the reservoir to match that of the previous reactor
    solu.TDY = r.thermo.TDY
    m.set_mass_flow_rate(condif[i]*solu.density*area)
    upstream.syncState()
    # integrate the reactor forward in time until steady state is reached
    sim.reinitialize()
    sim.advance_to_steady_state()
    # compute velocity and transform into time
    u[i] = condif[i] / r.thermo.density / area
    t_r[i] = r.mass / mass_flow_rate  # residence time in this reactor
    t[i] = np.sum(t_r)
    # write output data
    states.append(r.thermo.state)

# Plot results

plt.figure()
plt.plot(z, states.T, label='Reactor Chain')
plt.title('4 step combustion reactor network, $¥phi$ = ' + str(equiv_rat))
plt.xlabel('$z$ [m]')
plt.ylabel('$T$ [K]')
plt.legend(loc=0)
plt.show()

plt.figure()
plt.plot(z, states.X[:, solu.species_index('O2')], label='O2')
plt.plot(z, states.X[:, solu.species_index('CO2')], label='CO2')
plt.plot(z, states.X[:, solu.species_index('CO')], label='CO')
plt.plot(z, states.X[:, solu.species_index('H2')], label='H2')
plt.plot(z, states.X[:, solu.species_index('CH4')], label='CH4')
plt.title('4 step combustion reactor network, $¥phi$ = ' + str(equiv_rat))
plt.xlabel('$z$ [m]')
plt.ylabel('$Mole fraction$')
plt.legend(loc=0)
plt.show()
```

A.4 Oxygen-enriched Single Reactor

```
import numpy as np
import matplotlib.pyplot as plt
import cantera as ct
import math
import sys

def enriched_single(equiv_rat, injection):
    # equiv_rat = string representing the equivalence ratio
    # injection = oxygen enrichment. 0 normal air, 1 pure oxygen.

    # Initialize our methane solution depending on the mechanism,
    # and set the temperature, pressure and concentration
    solu = ct.Solution('4step_CH4.cti')
    #solu = ct.Solution('2step_CH4.cti')
    #solu = ct.Solution('1step_CH4.cti')

    solu.TPX = 300, 101325, 'CH4:1'
    molweight_m = solu.mean_molecular_weight/1000   #kg/mol

    # Initialize air and set the temperature, pressure
    # and concentration
    lucht = ct.Solution('air.cti')
    composition = {'O2':1, 'N2':((1-injection)*3.76)}
    lucht.TPX = 300, 101325, composition
    molweight_l = lucht.mean_molecular_weight/1000  #kg/mol


    # Create the reservoir objects
    inlet_methane = ct.Reservoir(solu)
    inlet_lucht = ct.Reservoir(lucht)
    exhaust = ct.Reservoir(solu)

    # Create our combustion chamber, an IdealGasReactor object,
    # filled with air initially
    solu.TPX = 1000, 101325, 'O2:0.21, N2:0.79'
    combustor = ct.IdealGasReactor(solu)

    # Create MassFLowControllers with mass flow rate such that
    # it corresponds to lean, stoichiometric or rich
    # fuel conditions. [kg/s]
    # Calculation of mdot for air is based on molar mass of O_2
    # which is 0.032, and N_2 which is 0.028.
    equiv_rat_dict = {"rich":1, "stoichiometric":2, "lean":4}
    equiv_rat_help = equiv_rat_dict[equiv_rat]
    mfc_m = ct.MassFlowController(inlet_methane, combustor,
        mdot = molweight_m)
    mfc_l = ct.MassFlowController(inlet_lucht, combustor,
        mdot = equiv_rat_help*(0.032 + (1-injection)*3.76*0.028))

    outlet = ct.Valve(combustor, exhaust, K=10)

    sim = ct.ReactorNet([combustor])

    time = 0
    tmax = 4
    steps = 300
    timestep = tmax/steps
    result = np.zeros((1,steps))

    for i in range(steps):
        result[0,i] = combustor.T
        time += timestep
        sim.advance(time)

    return result

maxO2inj = 100
res_lean=np.zeros((maxO2inj,300))
res_stoi=np.zeros((maxO2inj,300))
res_rich=np.zeros((maxO2inj,300))
```

```
for i in range(maxO2inj):
    res_lean[i,:] = enriched_single("lean", i/100)
    res_stoi[i,:] = enriched_single("stoichiometric", i/100)
    res_rich[i,:] = enriched_single("rich", i/100)

t = np.linspace(0, 4, 300)
for i in range(maxO2inj):
    if (i%10 == 0):
        plt.plot(t, res_lean[i,:], label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='best')
plt.title('Oxygen-enriched lean combustion')
plt.xlabel('Time [s]')
plt.ylabel('Temperature [K]')
plt.show()

for i in range(maxO2inj):
    if (i%10 == 0):
        plt.plot(t, res_stoi[i,:], label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='best')
plt.title('Oxygen-enriched stoichiometric combustion')
plt.xlabel('Time [s]')
plt.ylabel('Temperature [K]')
plt.show()

for i in range(maxO2inj):
    if (i%10 == 0):
        plt.plot(t, res_rich[i,:], label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='best')
plt.title('Oxygen-enriched rich combustion')
plt.xlabel('Time [s]')
plt.ylabel('Temperature [K]')
plt.show()
```

A.5 Oxygen-enriched Reactor Network

```
import numpy as np
import matplotlib.pyplot as plt
import cantera as ct
import math

#manually made function for convection diffusion
from convdiff import convdifffun

T_0 = 1000.0                # inlet temperature [K]
pressure = ct.one_atm       # constant pressure [Pa]
length = 50                 # approximate reactor length [m]
u_0 = 5                     # inflow velocity [m/s]
area = math.pi*(2/2)**2     # cross-sectional area [m**2]
n_steps = 20                # number of reactors within the network
condif = convdifffun(u_0, n_steps)

def enrich_network(equiv_rat, injection):
    # equiv_rat = equivalence ratio value
    # injection = oxygen enrichment. 0 normal air, 1 pure oxygen.

    # import the gas model and set the initial conditions
    composition_0 = 'CH4:1'
    composition_1 = 'O2:2, N2:' + str((1-injection)*7.52)
    composition = composition_0 + ', ' + composition_1
```

```
        solu = ct.Solution('4step_CH4.cti')
        #solu = ct.Solution('2step_CH4.cti')
        solu.TPX = T_0, pressure, composition
        solu.set_equivalence_ratio(equiv_rat, composition_0, composition_1)
        mass_flow_rate = u_0 * solu.density * area

        dz = length / n_steps
        r_vol = area * dz

        # create a new reactor
        r = ct.IdealGasReactor(solu)
        r.volume = r_vol

        # Create reservoirs for up and downstream
        upstream = ct.Reservoir(solu, name='upstream')
        downstream = ct.Reservoir(solu, name='downstream')

        # The mass flow rate into the reactor will be fixed over time
        # by using a MassFlowController object.
        m = ct.MassFlowController(upstream, r, mdot=mass_flow_rate)

        v = ct.PressureController(r, downstream, master=m, K=1e-5)

        # Prepare the solver
        sim = ct.ReactorNet([r])
        sim.rtol = 1e-5
        sim.atol = 1e-8
        sim.set_max_time_step(1)
        sim.max_err_test_fails = 100

        # define time, space, and other information vectors
        z = (np.arange(n_steps) + 1) * dz
        t_r = np.zeros_like(z)  # residence time in each reactor
        u = np.zeros_like(z)
        t = np.zeros_like(z)
        states = ct.SolutionArray(r.thermo)

        # iterate through the network
        for i in range(n_steps):
            # Set the state of the reservoir to match that
            # of the previous reactor
            solu.TDY = r.thermo.TDY
            # update mass flow rate according to convection-diffusion
            if(i != 0):
                m.set_mass_flow_rate(condif[i]*solu.density*area)
            upstream.syncState()
            # integrate the reactor forward in time until
            # steady state is reached
            sim.reinitialize()
            sim.advance_to_steady_state()
            # compute velocity and transform into time
            u[i] = condif[i] / r.thermo.density / area
            t_r[i] = r.mass / mass_flow_rate
            t[i] = np.sum(t_r)
            # write output data
            states.append(r.thermo.state)

        return z, states

    # Plot the results

    maxO2inj = 100
```

```
for i in range(maxO2inj):
    if (i%10 == 0):
        z, staat = enrich_network(0.5, i/100)  #lean
        plt.plot(z, staat.T, label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='upper right')
plt.title('Oxygen-enriched lean combustion')
plt.xlabel('z [m]')
plt.ylabel('Temperature [K]')
plt.show()

for i in range(maxO2inj):
    if (i%10 == 0):
        z, staat = enrich_network(1, i/100)  #stoichiometric
        plt.plot(z, staat.T, label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='upper right')
plt.title('Oxygen-enriched stoichiometric combustion')
plt.xlabel('z [m]')
plt.ylabel('Temperature [K]')
plt.show()

for i in range(maxO2inj):
    if (i%10 == 0):
        z, staat = enrich_network(2, i/100)  #rich
        plt.plot(z, staat.T, label='${i}% enriched oxygen$'.format(i=i))
plt.legend(loc='upper right')
plt.title('Oxygen-enriched rich combustion')
plt.xlabel('z [m]')
plt.ylabel('Temperature [K]')
plt.show()
```

Appendix B

Correspondance Cantera User Group

As the increase in temperature of oxygen-enhanced combustion was not understood at first, it was decided to ask help online. The following correspondence aided greatly for explaining the results of the model.

```
Jori:
"Hello all,

Nearing the end of my bachelor's thesis for mathematics, I ran
into a problem. I am modelling combustion inside a rotary kiln,
where I have worked the past months to make several models of
varying difficulty. For all these models methane and air were
propelled into an IdealGasReactor using a MassFlowController.
What I was trying to research is what happens to the temperature
in the reactor if I were to increase the percentage of oxygen
present in air. So before the air flows into the reactor it is
injected with oxygen (oxygen-enriched combustion). As an example,
consider 25% O2 with 75% N2 instead of 21% O2 with 79% N2
flowing through the inlet into the reactor.Now as a mathematician
I am not very knowledgable about the chemics part. I am trying to
make sure the same amount of mol O2 and CH4 enters the reactor as
before, which is important because otherwise the temperature
could increase as result of more oxygen being available. The
code snippet I'm using for it right now is found below.

My questions:
- Is this the correct way to use mdot in order to keep
    the same mol ratio? my molweight_l is increasing as i
    increase injection, which seems logical as O has a higher
    molecular weight than N.
- Is there a way to check the mol ratio entering a reactor?
- I have found 2 approaches: density or mean_molecular_weight, are both valid?


(Python 3.7.7 with Cantera 2.4.0)

WITHOUT injection:

solu = ct.Solution('2step_CH4.cti')      #custom file for 2 step
                                         #methane reaction mechanism
solu.TPX = 300, 101325, 'CH4:1'
molweight_m = solu.mean_molecular_weight/1000   #kg/mol
```

```
air = ct.Solution('air.cti')
composition = 'O2:0.21, N2:0.79'
air.TPX = 300, 101325, composition
molweight_l = air.mean_molecular_weight/1000       #kg/mol

[...]

mfc_m = ct.MassFlowController(..., ..., mdot = molweight_m)
mfc_l = ct.MassFlowController(..., ..., mdot = molweight_l*2/0.21

WITH injection:

injection = 5
solu = ct.Solution('2step_CH4.cti')     #custom file for 2 step
                                        #methane reaction mechanism
solu.TPX = 300, 101325, 'CH4:1'
molweight_m = solu.mean_molecular_weight/1000   #kg/mol

air = ct.Solution('air.cti')
composition = 'O2: ' + str(0.21 + injection) + ', N2:' + str(0.79-injection)
air.TPX = 300, 101325, composition
molweight_l = air.mean_molecular_weight/1000       #kg/mol

[...]

mfc_m = ct.MassFlowController(..., ..., mdot = molweight_m)
mfc_l = ct.MassFlowController(..., ..., mdot = molweight_l*2/(0.21+injection)

"
```

Where the user "Bryan Callaway" responded with the following message:

```
Bryan Callaway:
"I'm having some trouble understanding your problem, but think that
you're talking about a reducing kiln, i.e. one with a deliberate
surplus of natural gas so that combustion is incomplete (this
would account for "the temperature could increase as a result of
more oxygen being available", which would not be the case in the
bulk gas for a lean burn system).  It seems that you want to hold
the ratio of CH4 to O2, and possibly the flow rate of CH4, constant.
If this is the case, then as you enrich with oxygen, you would
want to reduce the flow rate of air to keep the total oxygen flow
constant, yes?  So the effect of oxygen enrichment would in fact
be reduced flow of nitrogen.  You have some fixed oxygen flow, and
nitrogen flow is .79/.21=3.76 times the portion of oxygen flow
delivered from air.  The mass flow of O2+N2 is the mass flow of O2
(constant with a constant fuel rate) plus the mass flow of N2 (readily
calculated from the mol flow rate of N2 just described).

Letting enrichment scale from 0 (all air) to 1 (all oxygen), you could write:
charge.TPX=(T, P, {'O2':1, 'N2':((1-enrichment)*3.76)})

If you want the mass flow of enrichment oxygen, that will simply be enrichment*O2.

It's easy then to define corresponding mass flow rates for O2
(a constant) and N2 (a constant times 1-enrichment), and set the
total mass flow as the sum of the two.

If you're adjusting the methane flow rate, then you would scale
the mass flow of O2+N2 and enrichment O2 with that flow rate.

Note that with these constraints, as enrichment increases (and the
flow of nitrogen decreases) the temperature will increase because
```

```
the same chemical energy will be spread across a smaller total
thermal mass of gas.

While we're here, can you clarify why chemical kinetics are necessary
for your problem?  Unless you're doing some elaborate 3-D CFD with
chemical kinetics, I would expect an equilibrium assumption to be
adequate to find temperatures in this case.

If my understanding of your problem is incorrect, please advise.

Thank you,

Bryan
"
```