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

Time Integration of the Chemistry of Combustion Processes in Industrial Furnaces, using Julia

Thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfilment of the requirements
for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by
Jochem Jelle van der Meer
Delft, Nederland
June 2023

BSc thesis APPLIED MATHEMATICS

"Time Integration of the Chemistry of Combustion Processes in Industrial Furnaces, using Julia"

Jochem Jelle van der Meer

Delft University of Technology

Thesis committee
Dr. Domenico J.P. Lahaye
Dr. Neil V. Budko

June, 2023
Delft

Abstract

Combustion is and remains for the foreseeable future an essential chemical process. The physical and mathematical modelling of such processes can help to optimise the design and operation of combustion furnaces which is critical for fuel efficiency. Given the combined complexity and interaction of the gas flow dynamics and the reaction processes however, such modelling can become time consuming and demanding when it is comes to computational capacity.

It is for this reason that alternative modelling and computational techniques are of interest. This paper serves as a thorough introduction to modelling this process using Chemical Reactor Networks (CRN) – which is known to be a relatively efficient model and computational approach.

It provides a step-by-step explanation of the CRN approach, as well as a hands-on implementation for a One-Step Mechanism, ie. a combustion process involving only a single stage oxidisation of the fuel. It also introduces the reader to the industry standard CHEMKIN format and the GRI 3.0 data-base, and investigates the possibility of incorporating the state-of-the-art GRI 3.0 database into Chemical Reactor Networks.

Following on from this work it is recommended to validate CRN based results against experimental data and modelling results using different techniques such as Computational Fluid Dynamics to gauge both accuracy and computational efficiency.

Acknowledgements

Many thanks go to Dr. Lahaye, who never shied away from the long meetings many of which ate into overtime. They always left me feeling highly motivated regarding both the subject matter and the project.

Contents

1 Introduction

Approximately 90% of the total energy production (including electricity) globally, is realized using combustion processes [15]. Despite the growth in renewable energies hydrocarbon fuels are expected to remain very relevant for many applications due to some their unique properties. Chief amongst them being their high energy density.

Given the increasing costs of combustion fuels and requirements to reduce carbon footprint, the further optimisation of combustion processes therefore remains very important. One way to identify and test possibilities for optimisation is through physical experiments using model systems, or even at industrial scale. Such experiments are however costly, time consuming, and combustion furnaces may not be readily available.

Instead physical and mathematical modelling can be applied which if validated, can overcome many of the limitations of physical experiments. Combustion processes are however notably complex involving the interdependent effect of flow of gasses and combustion products through the furnace, and the complex and often multi-stage combustion, chemical reaction processes, themselves. As a consequence such modelling can become very demanding in terms of computational power and time.

It is for that reason that alternative and more simple physical and mathematical modelling techniques are being investigated. One such technique is Chemical Reactor Networks (CRN), the subject of this research and paper. All models have limitations, CRN however has a main advantage when it comes to its computational efficiency relative to other common models. This applies especially when compared to Computational Fluid Dynamics.

In order to help progress the evaluation and eventual adaptation of CRN this paper provides a thorough introduction to subject. This is deemed useful because the subject of CRN and Combustion, can be a confusing topic as it draws on such a wide range of various disciplines namely Differential Equations, Graph Theory, Chemistry, Linear Algebra, and Numerical Methods.

In order to maximise our understanding of CRN and in particular the explicit control of the model, this study has developed some of the key algorithms used to construct the networks itself, instead of relying on using standard packaging from libraries such as NetworkDynamics.jl. The functioning of this CRN approach is tested and evaluated with a one-step mechanism for the combustion of methane.

Finally, we will also discuss how the approach can be extended to incorporate GRI 3.0, the importance of Chemical Kinetics, and how to handle GRI. Which enables the deployment of more complex processes than the one-step mechanism.

2 Construction of Chemical Reactor Networks, an Introduction

Chemical Reactor Networks are constructed by splitting up a problem (furnace, kiln, etc) spatially. Each section can then be represented by a node in a graph/network. Where each section is connected to the next via edges between the respective nodes. On each of these nodes we apply a model for simulating the chemistry that occurs within. We can then carry out calculations upon the network as a whole covering both the chemistry for all nodes and the flow between them.

In order to further the reader's understanding we will make use of a visual aid. Take for example this cement kiln:

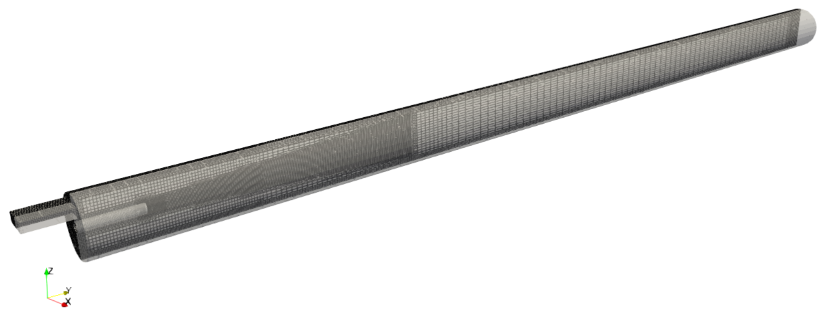


Figure 1: Rendering of Cement Kiln, courtesy of Dr. Lahaye

We can split this kiln into multiple, say 4, sections as follows:

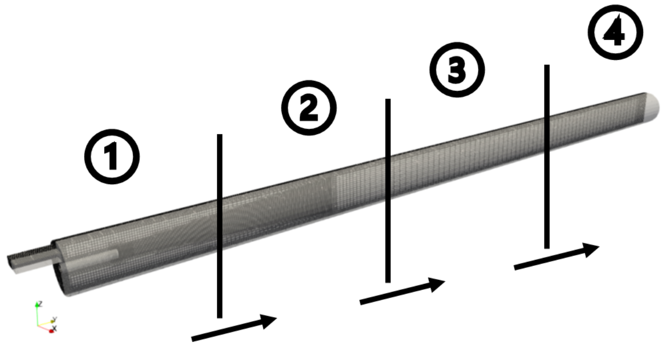


Figure 2: Rendering of Cement Kiln, courtesy of Dr. Lahaye, divided into 4 sections

From this we can extract our graph of nodes and edges:

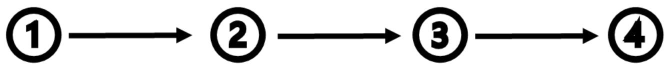


Figure 3: Graph of the Cement Kiln

Now we have a graph representing the cement kiln, such that the nodes represent a section of the kiln and the edges represent a connection between the sections. With a network such as this we can apply chemistry to each of the nodes and use the edges to connect the sections.

This is the essence of Chemical Reactor Networks.

2.1 Chemical Reactions on Nodes of the Network

First we must introduce some very basic terminology, namely Chemical species, or species, simply put this term is commonly used term to encompass molecules, atoms, ions and radicals.

For each node we wish to know the molar concentration of the various species which will be denoted using the common notation $[\cdot]$, the aforementioned is defined such that $[X_k]$ is the number of moles of species k per unit volume[15].

Each i-th node therefore will have a corresponding vector of molar concentrations:

$$u_i = \begin{bmatrix} [X_1] \\ \vdots \\ [X_{n_s}] \end{bmatrix}, \text{ where } n_s \text{ is the number of species.}$$

Off course we would also like to know how the concentration of the various species change over time. This, off course, is governed by differential equations:

$$\frac{d[X_k]}{dt} = RHS, \text{ for arbitrary } k$$

We will discuss the specifics of the RHS later on, see section 4 and section 5, but for now let's not get bogged down with the details.

For each i-th node we can again make a corresponding vector, this time containing the changes in concentrations:

$$\dot{u}_i = \begin{bmatrix} \frac{d[X_1]}{dt} \\ \vdots \\ \frac{d[X_{n_s}]}{dt} \end{bmatrix}, \text{ where again } n_s \text{ is the number of species.}$$

This results in two lists of vectors, namely:

$$u_1, u_2, \ldots, u_{n_n} \text{ and } \dot{u}_1, \dot{u}_2, \ldots, \dot{u}_{n_n}, \text{ where } n_n \text{ is the number of nodes.}$$

These lists of vectors can then be combined into two larger vectors as follows:

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_{n_n} \end{bmatrix}, \dot{u} = \begin{bmatrix} \dot{u}_1 \\ \vdots \\ \dot{u}_{n_n} \end{bmatrix}$$

Here u and $\dot{u}$ are associated with the entire network, with u_i and $\dot{u}_i$ being associated with the i-th node.

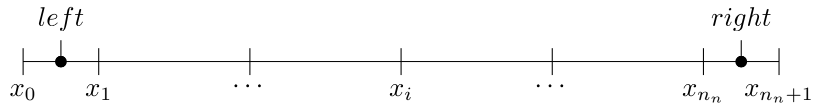
2.2 Diffusion across Edges of Network

In order to connect the different nodes of the network according to the edges between them we must allow for the diffusion of chemical species. This is governed by Fick's Second Law of Diffusion[2]:

$$\frac{\partial[X_k]}{\partial t} = D\frac{\partial^2[X_k]}{\partial x^2}, \text{ for arbitrary } k$$

Where D is a diffusivity coefficient which may be a function of temperature and can differ per species. For now we will simply set $D = 1$, see section 6. Conveniently and unsurprisingly Fick's Second Law is analogous to the Heat Equation, a common problem for which many spatial discretization methods are known. Here we will be drawing upon the field of Numerical Methods[10].

We can visualise the spatial aspect of our problem (for example the kiln we mentioned earlier) as a line, with a node at each x_i:



Where x_0 and x_{n_n+1} are virtual points, because in reality we only have n_n nodes. With regards to our kiln example earlier, these virtual points represent the space outside of the kiln.

Because we are modelling a sealed furnace there should be no flow of species out of the ends of the furnace. Meaning there should be no flow from point x_1 through $left$ to x_0, nor from x_{n_n} through $right$ to x_{n_n+1}. This will give us the following (cell-centered Neumann) boundary conditions[10]:

$$\frac{\partial[X_k]}{\partial x}(left) = 0, \frac{\partial[X_k]}{\partial x}(right) = 0, \text{ for arbitrary } k$$

We can now proceed to spatially discretise the diffusion equation. A common approach is FDM (Finite Differences Method).

For ease of notation we will use $y = [X_k]$ for arbitrary species k, and $y_i = y(x_i)$.

We start by constructing Taylor polynomials around x_i in order to derive the second-order central difference quotient:

$$y_{i+1} = y_i + h\frac{dy}{dx}(x_i) + \frac{h^2}{2!}\frac{d^2y}{dx^2}(x_i) + \frac{h^3}{3!}\frac{d^3y}{dx^3}(x_i) + \mathcal{O}(h^4)$$

$$y_{i-1} = y_i - h\frac{dy}{dx}(x_i) + \frac{h^2}{2!}\frac{d^2y}{dx^2}(x_i) - \frac{h^3}{3!}\frac{d^3y}{dx^3}(x_i) + \mathcal{O}(h^4)$$

$$\implies y_{i-1} + y_{i+1} = 2y_i + h^2 \frac{d^2y}{dx^2} + \mathcal{O}(h^4)$$

$$\implies \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} = \frac{d^2y}{dx^2} + \mathcal{O}(h^2)$$

Where h represents the (normalised) distance between nodes and thus between sections, since we are using uniform distance for all edges we will use $h = 1$. The standard notation for the numerical solution is u_i. Conveniently due to the set up of our model in 2.1 and because we designed y_i for arbitrary species, we are able to use the same u_i, by switching from element-wise (species wise) to vector-wise notation. Resulting in the following set of equations:

$$u_{i-1} - 2u_i + u_{i+1} = f_i, \text{ for } i = 1, \ldots, n_n \text{ where } f_i = \tfrac{\partial u_i}{\partial t}$$

With similarly discretised Boundary Conditions:

$$\begin{gathered} \tfrac{\partial [X_k]}{\partial x}(left) = 0 \implies -u_0 + u_1 = 0 \\ \tfrac{\partial [X_k]}{\partial x}(right) = 0 \implies -u_{n_n} + u_{n_n+1} = 0 \end{gathered}$$

Note however that the equations for $i = 1, n_n$ contain the virtual points with index $i = 0, n_n + 1$:

$$u_0 - 2u_1 + u_2 = f_1, \; u_{n_n-1} - 2u_{n_n} + u_{n_n+1} = f_{n_n}$$

In order to assert our Boundary Conditions and remove these virtual points we will substitute their discretised versions into these equations. Resulting in the following set of equations:

$$\begin{gathered} -u_1 + u_2 = f_1 \\ u_{i-1} - 2u_i + u_{i+1} = f_i, \text{ for } i = 2, \ldots, n_n - 1 \\ u_{n_n-1} - u_{n_n} = f_{n_n} \end{gathered}$$

We can then use this to construct a matrix-vector system for arbitrary species k which will look as follows:

$$[\dot{X_k}] = \underbrace{\begin{bmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & & \ddots & & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{bmatrix}}_{A} \begin{bmatrix} [X_k]\Big|_{node\ 1} \\ \vdots \\ [X_k]\Big|_{node\ n_n} \end{bmatrix} \tag{1}$$

Thus if our system were to consist of only one species, rather than n_s species we could write our system simply as:

$$\dot{u} = Au \tag{2}$$

Those familiar with Graph Theory will notice that the matrix A seems very familiar. In fact $A = -L$, where L is the Laplacian matrix of the graph. This is because the discrete Laplacian operator, also known as the Laplacian matrix, is the discretised version of the standard continuous Laplacian operator, i.e. its the continuous Laplacian operator re-defined for a discrete grid/graph. And the operator used in Fick's Second Law of Diffusion, $\frac{\partial^2}{\partial x^2}$, is simply the Laplace operator in one dimension. For a more thorough explanation of this connection with Graph Theory please refer to [21]. Fortunately the Laplacian matrix of a graph is common terminology which is often implemented in software dealing with graphs, including the Graphs.jl package [7] which we will be using. For formal definitions from the field of Graph Theory please refer to [1].

Now we will rewrite the above system (1), in order to set up a system for the entire network.

Note that $u = \begin{bmatrix} u_1 \\ \vdots \\ u_{n_n} \end{bmatrix}$ where $u_i = \begin{bmatrix} [X_1]\Big|_{node\ i} \\ \vdots \\ [X_{n_s}]\Big|_{node\ i} \end{bmatrix}$ are column vectors.

Consider u written horizontally such that the column-vectors are unstacked and laid out, as follows

$$\tilde{u} = unstack(u) = \begin{bmatrix} u_1 & u_2 & \dots & u_{n_n} \end{bmatrix} = \begin{bmatrix} u_1[1] & u_2[1] & \dots & u_{n_n}[1] \\ u_1[2] & u_2[2] & \dots & u_{n_n}[2] \\ \vdots & \vdots & \dots & \vdots \\ u_1[n_s] & u_2[n_s] & \dots & u_{n_n}[n_s] \end{bmatrix}$$

Where $u_i[k] = [X_k]\Big|_{node\ i}$ denotes k-th element of u_i.
Then

$$\tilde{u}^T = \begin{bmatrix} u_1[1] & u_1[2] & \dots & u_1[n_s] \\ u_2[1] & u_2[2] & \dots & u_2[n_s] \\ \vdots & \vdots & \dots & \vdots \\ u_{n_n}[1] & u_{n_n}[2] & \dots & u_{n_n}[n_s] \end{bmatrix}$$

For ease of notation we will call this matrix U and we'll let

$$U_k = \begin{bmatrix} [X_k]\Big|_{node\ 1} \\ \vdots \\ [X_k]\Big|_{node\ n_n} \end{bmatrix} = \begin{bmatrix} u_1[k] \\ \vdots \\ u_{n_n}[k] \end{bmatrix}$$

Then

$$U = \begin{bmatrix} U_1 & U_2 & \dots & U_{n_s} \end{bmatrix} = \tilde{u}^T$$

is of the following form:

$$\begin{array}{c|c}
 & species1\downarrow \quad species2\downarrow \quad \dots \quad species\ n_s\downarrow \\
\begin{array}{l} node1\rightarrow \\ node2\rightarrow \\ \vdots \\ node\ n_n\rightarrow \end{array} & \left(\begin{array}{cccc} & & & \\ & & & \\ & & & \\ & & & \end{array}\right)
\end{array}$$

Such that

$$AU = \begin{bmatrix} AU_1 & AU_2 & \dots & AU_{n_s} \end{bmatrix}$$

is a matrix multiplication for the entire network.
If we use $\widehat{u_i[k]} = \widehat{[X_k]}\Big|_{node\ i}$ to denote the effect of the matrix multiplication on k-th element of u_i, then

$$AU = \begin{bmatrix} \widehat{u_1[1]} & \widehat{u_1[2]} & \dots & \widehat{u_1[n_s]} \\ \widehat{u_2[1]} & \widehat{u_2[2]} & \dots & \widehat{u_2[n_s]} \\ \vdots & \vdots & \dots & \vdots \\ \widehat{u_{n_n}[1]} & \widehat{u_{n_n}[2]} & \dots & \widehat{u_{n_n}[n_s]} \end{bmatrix}$$

Of the form
$$\begin{array}{lcccc}
 & species1\downarrow & species2\downarrow & \dots & species\ n_s\downarrow \\
node1\rightarrow & \widehat{u_1[1]} & \widehat{u_1[2]} & \dots & \widehat{u_1[n_s]} \\
node2\rightarrow & \widehat{u_2[1]} & \widehat{u_2[2]} & \dots & \widehat{u_2[n_s]} \\
\vdots & \vdots & \vdots & \dots & \vdots \\
node\ n_n\rightarrow & \widehat{u_{n_n}[1]} & \widehat{u_{n_n}[2]} & \dots & \widehat{u_{n_n}[n_s]}
\end{array}$$

We can then rewrite this into same the form of u and $\dot{u}$ by taking the rows and stacking them vertically.

$$stack(AU) = \begin{bmatrix} row_1(AU) \\ row_2(AU) \\ \vdots \\ row_{n_n}(AU) \end{bmatrix} = \begin{bmatrix} \widehat{u_1[1]} \\ \widehat{u_1[2]} \\ \vdots \\ \widehat{u_1[n_s]} \\ \vdots \\ \widehat{u_{n_n}[1]} \\ \widehat{u_{n_n}[2]} \\ \vdots \\ \widehat{u_{n_n}[n_s]} \end{bmatrix}$$

So we finally derive our network wide system for multiple species.

$$\dot{u} = stack(AU) \tag{3}$$

3 Diffusion only

In this section we will investigate the effects of the diffusion without any chemical reactions taking place. This will be showcased on various graphs. For each graph we will give the Initial Conditions (henceforth IC) of a high concentration for one or multiple species in one or multiple nodes. To implement the Diffusion-only Mechanism the following (pseudo-)code is used to define the Differential Equation which is then solved using DifferentialEquations.jl.

Algorithm 1: Diffusion only

```
$u\big|_{t=initial} = IC$;
**Function** *Differential Equation*
    $U = transpose(unstack(u))$;
    $\dot{u} += stack(AU)$;
**end**
```

We may consider our diffusion successful if for each graph we achieve:

- For any given species its concentration should be the same in each node.
- No matter is created or destroyed, because there is no chemistry and no flow out of the network.

The latter can be checked simply by taking the sum of the final (post-diffusion) concentration across the nodes, which should then be equal to the sum of the initial concentration across the nodes.

$$\forall k : \sum_{i=1}^{n_n} [X_k]\Big|_{node=i,t=initial} = \sum_{i=1}^{n_n} [X_k]\Big|_{node=i,t=final} \tag{4}$$

3.1 Graph 1

The first graph we will showcase is a simple straight line with 6 nodes. It looks as follows.

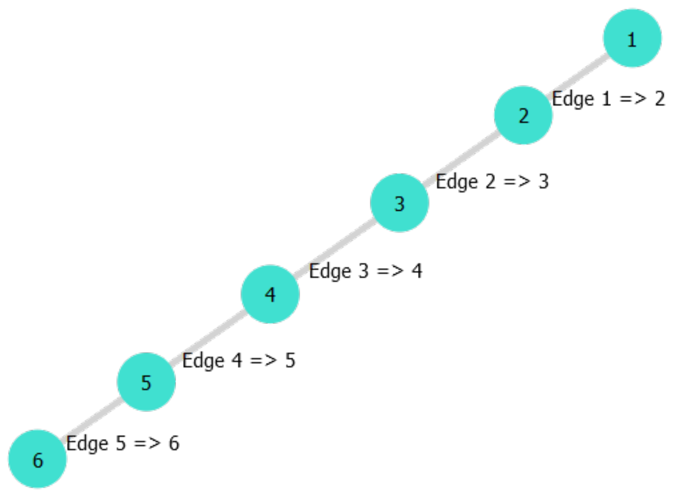


Figure 4: Graph 1 rendered using GraphPlot.jl

This image was generated using the very handy package GraphPlot.jl[13]. When building graphs with the Graphs.jl package it's always useful and good practice to visually double check that you've build the desired graph with a plot of the graph, this will prevent a great deal of frustration.

We will be using the following IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1
- $[O_2] = 1$, initial value for O_2 in node 2
- $[O_2] = 1$, initial value for O_2 in node 4

The results for each species are displayed below. The labels indicate the corresponding node.

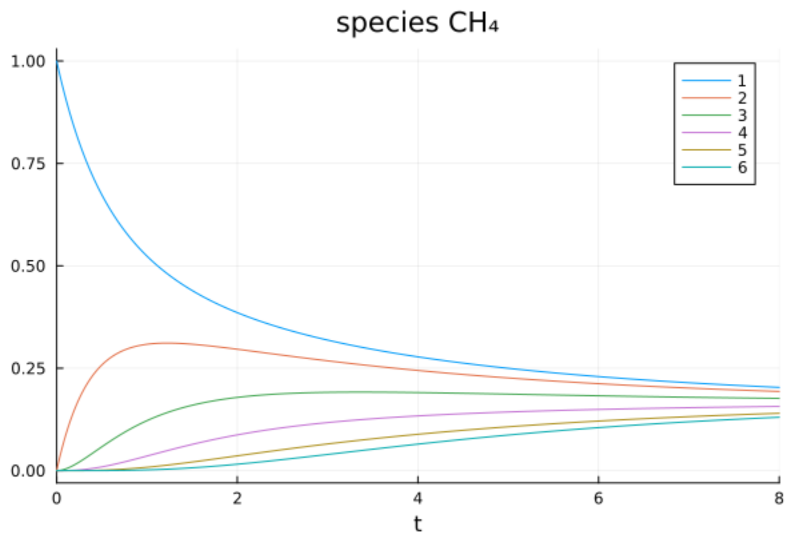


Figure 5: Concentrations of CH_4

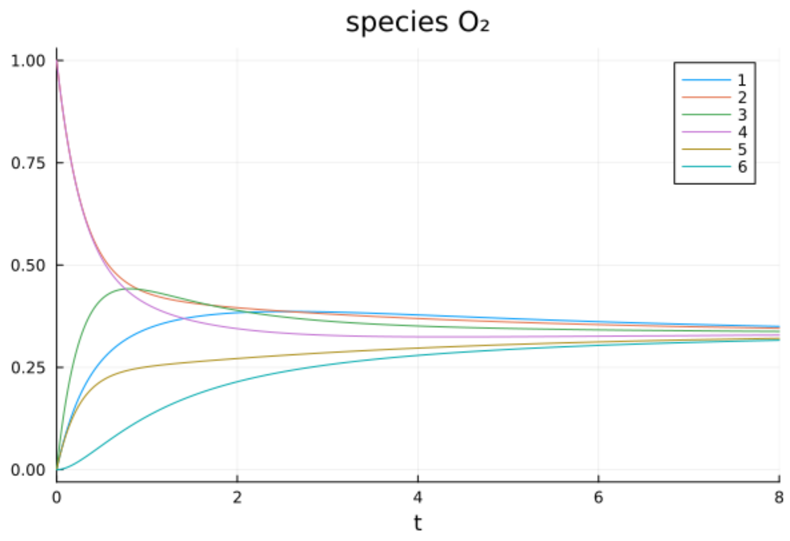


Figure 6: Concentrations of O_2

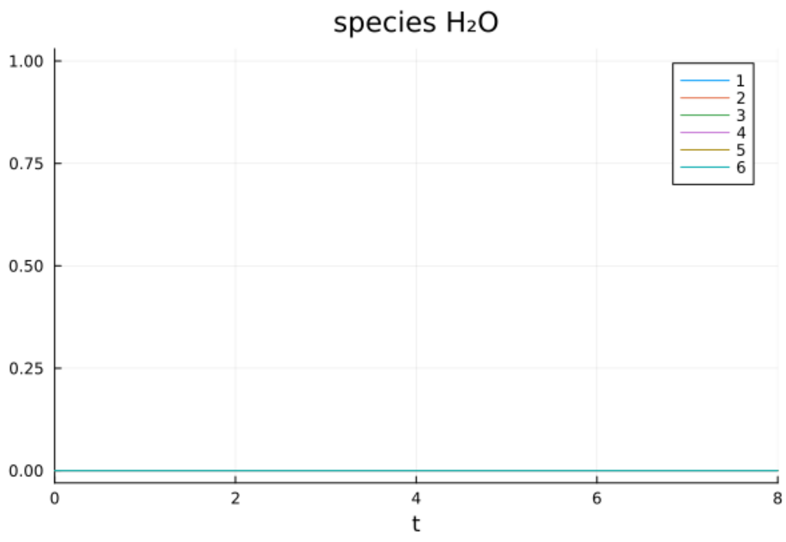


Figure 7: Concentrations of H_2O

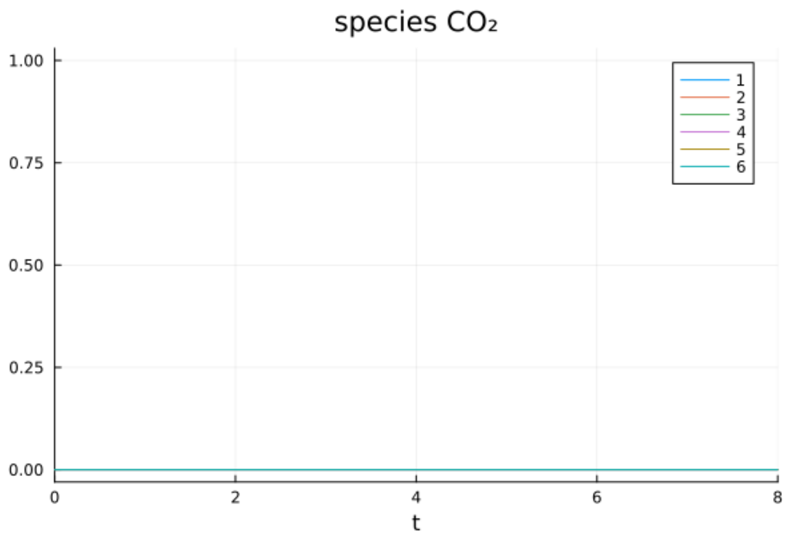


Figure 8: Concentrations of CO_2

As we can see in Figure 5 and Figure 6 we get a nice even diffusion of CH_4 and O_2 respectively. Where for both species the concentrations equalise across all the nodes. Thus satisfying our first criteria.

Also, no matter is created or destroyed. Thus satisfying our second criteria. This is observed very easily for each species. To wit: for CH_4 we get a concentration of $\frac{1}{6}$ for each node, with a total of 6 nodes: $\frac{1}{6} \cdot 6 = 1$ which is the same total as our IC.

The above graphs are species focused, for a node focused graph please see appendix B

3.2 Graph 2

The second graph we will showcase is a loop with 6 nodes. It looks as follows.

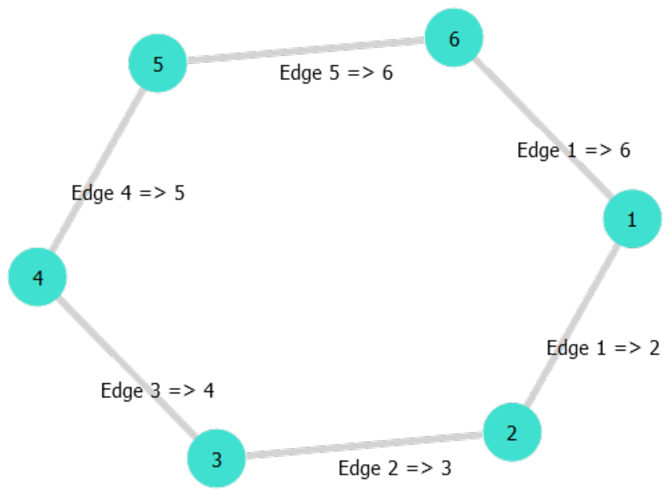


Figure 9: Graph 2 rendered using GraphPlot.jl

Using the same IC as for Graph 1 we get the following results.

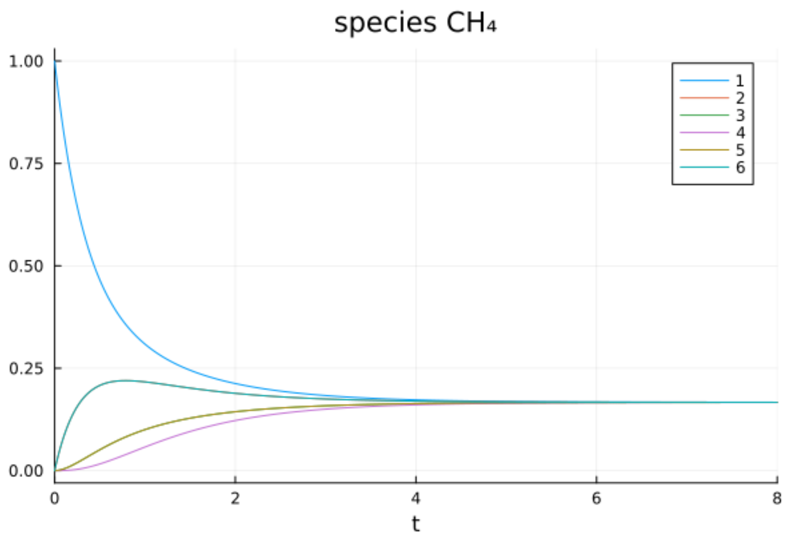


Figure 10: Concentrations of CH_4

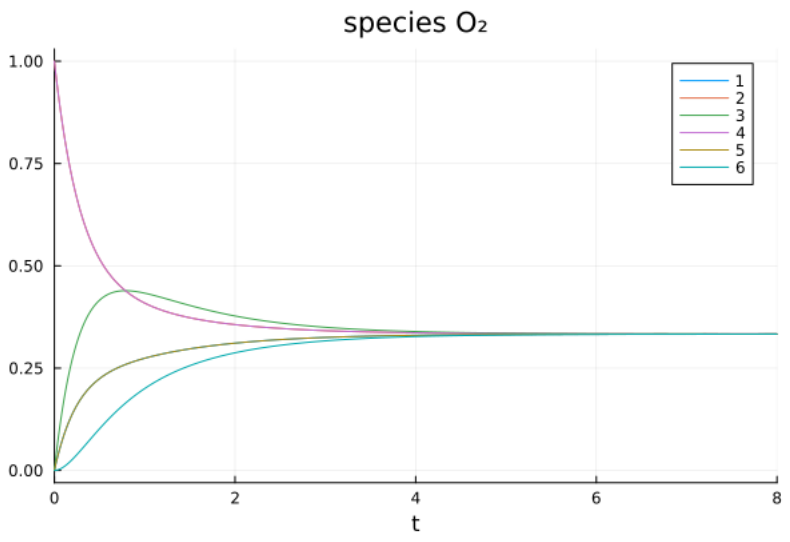


Figure 11: Concentrations of O_2

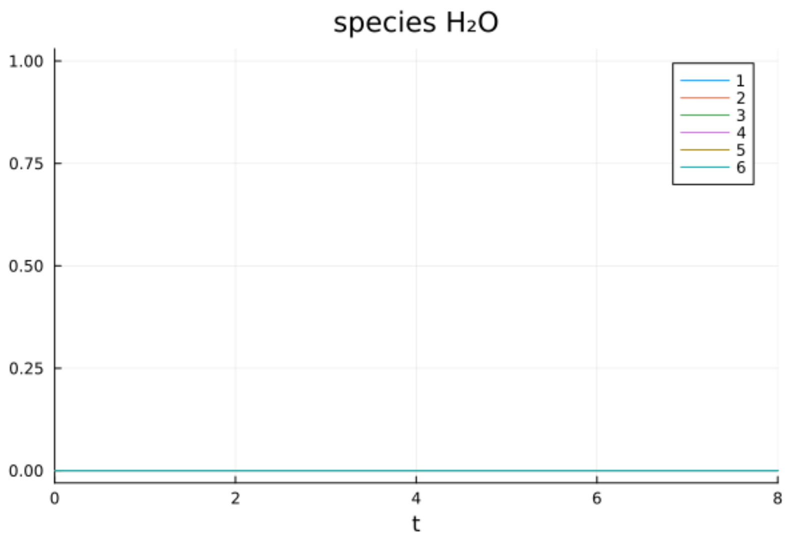


Figure 12: Concentrations of H_2O

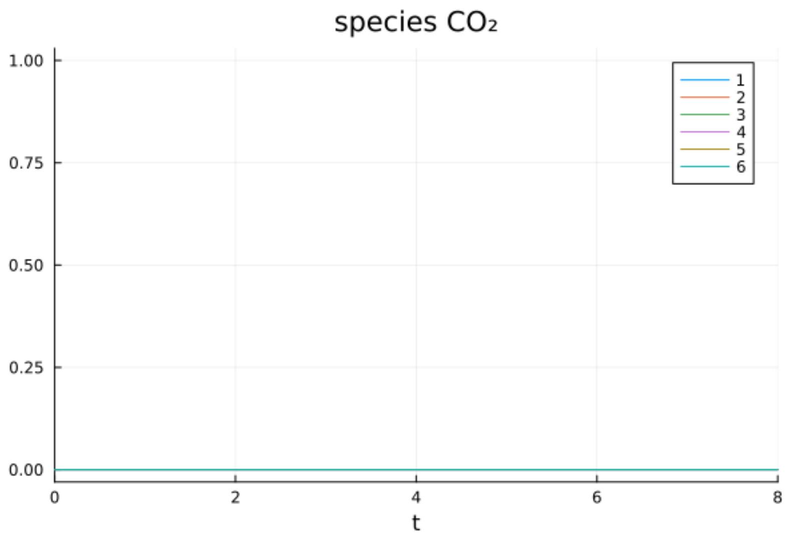


Figure 13: Concentrations of CO_2

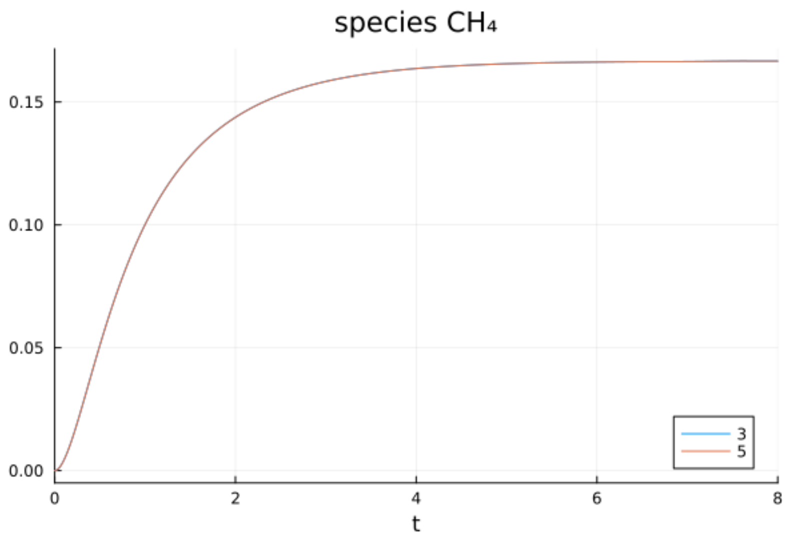


Figure 14: Concentrations of CH_4, overlap node 3 and 5

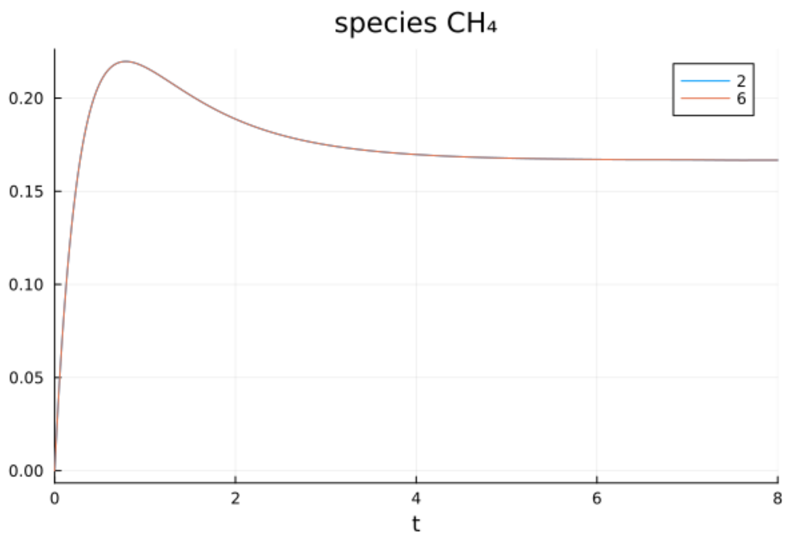


Figure 15: Concentrations of CH_4, overlap node 2 and 6

As we can see in figure 10 and 11 we get a nice even diffusion of CH_4 and O_2 respectively. And again the total amount of matter stays the same.

The reader may have noticed that in Figure 10 only 4 lines are visible yet we have 6 nodes. This is because the lines for node 3 and 5 overlap, as can be seen in figure 14. And the lines for node 2 and 6 overlap, as can be seen in figure

15. This occurs because node 2 and 6 are equidistant from node 1 into which the methane is injected. Likewise node 3 and 5 are also equidistant from node 1. Thus they will fill with methane at equal rates.

Likewise for oxygen we can see from the below figures 16 and 17 that the lines for nodes 2 and 4 overlap, and the lines for nodes 1 and 5 overlap, respectively.

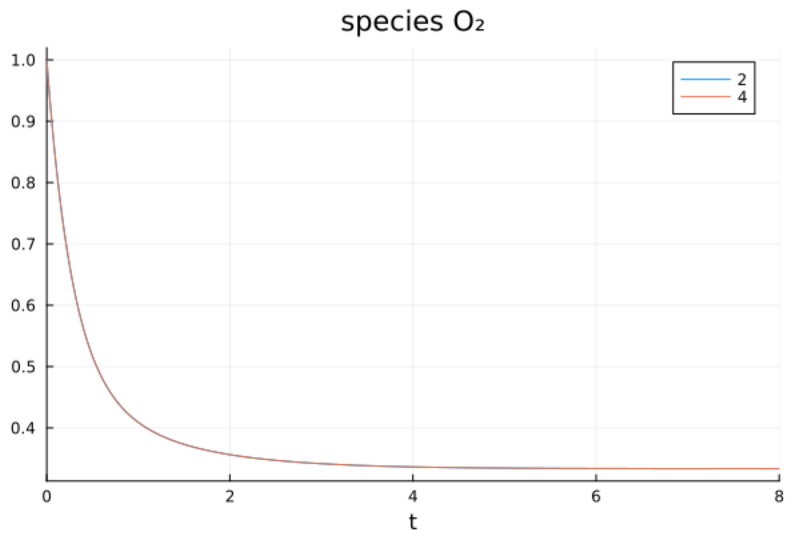


Figure 16: Concentrations of O_2, overlap node 2 and 4

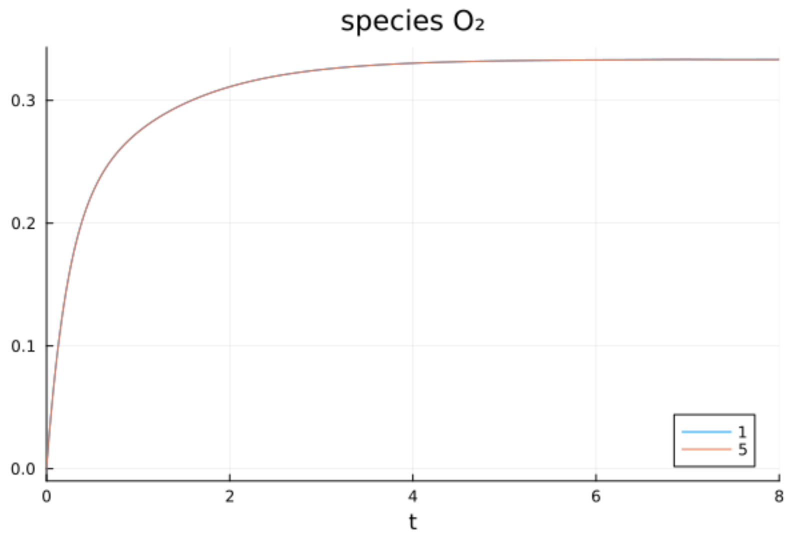


Figure 17: Concentrations of O_2, overlap node 1 and 5

The above graphs are species focused, for node focused graphs please see appendix B

3.3 Graph 3

For our third graph we will use a standard generator from the Graph.jl[7] package in order to produce a more complicated graph. The generator used is the barabasi-albert function with 12 nodes, an average degree of 3, and random seed 42.

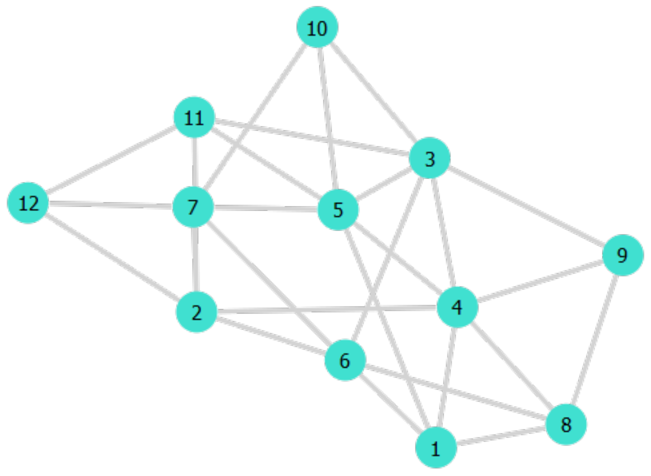


Figure 18: Graph 3 rendered using GraphPlot.jl

In order to reduce the number of graphs for us to look at we will use the following IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1

Which provides the following results.

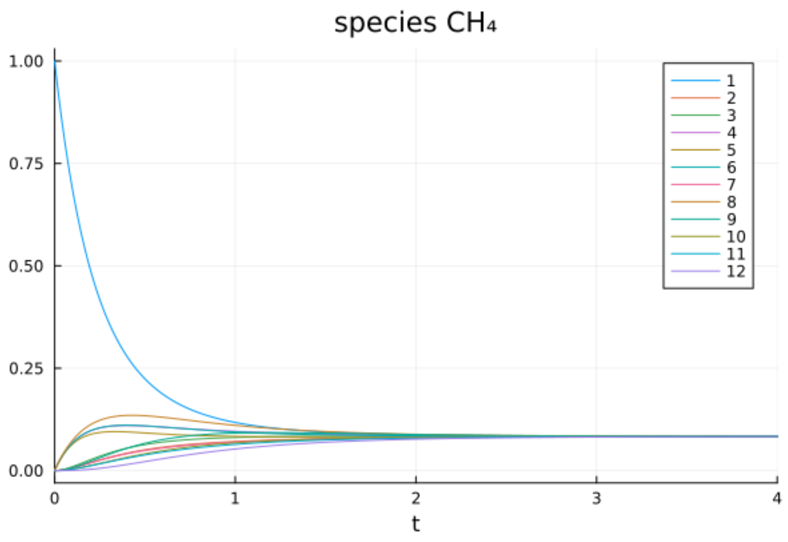


Figure 19: Concentrations of CH_4

Again we get a nice even diffusion across all nodes and the total amount of matter is maintained. For the additional graphs please refer to appendix B.

3.4 Very large graphs

Here we will investigate diffusion on very large graphs. Specifically we will compare calculation times for diffusion when using time-integration and a linear solver. Julia provides a native standard library called LinearAlgebra [9] it contains the backslash operator, so called because of its syntax "\" , which performs linear solution.

However we must first derive the linear system which will we solve. We will use only one species, say CH_4, with IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1

Recall the diffusion system for one species (2):

$$\dot{u} = Au$$

Diffusion will have finished when $\dot{u} = 0$, thus we want to solve

$$\dot{u} = Ax = 0, \text{ where } x = u\big|_{t=final} \tag{5}$$

We can use (4) to incorporate our IC as follows.

$$\sum_{i=1}^{n_n} [CH_4]\Big|_{node=i,t=initial} = \sum_{i=1}^{n_n} [CH_4]\Big|_{node=i,t=final}$$

$$\implies 1 = \sum_{i=1}^{n_n} [CH_4]\Big|_{node=i,t=final} = \sum_{i=1}^{n_n} x_i$$

Which we can incorporate into the system we need to solve.

$$\dot{u} = \underbrace{\begin{bmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & & \ddots & & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}}_{\bar{A}} \begin{bmatrix} x_1 \\ \vdots \\ x_{n_n} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}}_{q}$$

$$\dot{u} = \bar{A}x = q$$

To generate these graphs we will again use the barabassi-albert graph generator with an average degree of 3 and random seed 42. A varying number of nodes will be used in increasing orders of magnitude to up the computational load. The results are tabulated in Table 1.

		Time required (s)	
Graph	Number of nodes	Linear solve ”\”	DifferentialEquation.jl
4	1,000	0.1750141	1.1304552
5	10,000	41.8191324	159.6922803
6	100,000	unsuccessful	528.5795513

Table 1: Time requirements diffusion

As we can see the Backslash operator is able to deliver much faster performance than time-integration using DifferentialEquations.jl. However due to memory requirements the former is unsuccessful when we reach Graphs of the order of 100,000 nodes. For the related visuals refer to Appendix B.

4 One-Step Mechanism

In this section we will look at a simple chemical reaction for the combustion of methane, a so-called One-Step Mechanism.

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O$$

Here Methane and Oxygen react forming Carbon-dioxide and Water. The graph upon which we will implement this looks as follows:

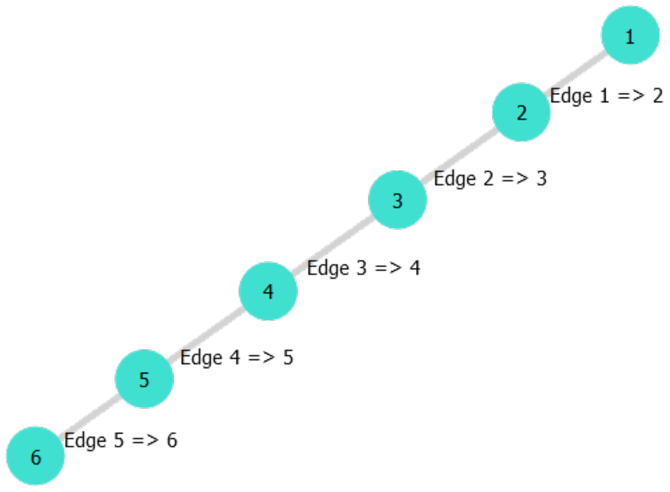


Figure 20: Graph 1 rendered using GraphPlot.jl

Now that we have our graph and we have our species and chemical reaction, the next step is to determine the reaction rate. This is governed by the following aspects:

1. Arrhenius' Law
2. Concentration levels
3. Stoichiometric Coefficients

Arrhenius Law [15] is an empirical law which states that the (forward) reaction rate coefficient k is given by:

$$k = AT^{\beta} exp(-\frac{E}{RT})$$

Where

- k is the (forward) reaction rate coefficient
- A is a pre-exponential constant
- T is the temperature
- β is the temperature exponent
- E is the activation energy
- R is the ideal gas constant.

When concentrations of CH_4 and O_2 are higher more collisions between them will occur thus increasing the chances of a reaction taking place. To take this into account the Arrhenius' law must be combined with the concentration levels of Methane and Oxygen to determine the overall (forward) reaction rate, as follows[15]:

$$rate = k \cdot [CH_4] \cdot [O_2]$$

Because this reaction is not an elementary reaction we must also take into account the associated reaction order for the two species,[11].

$$rate = k \cdot [CH_4]^1 \cdot [O_2]^{0.5}$$

In order to determine the rate at which each *individual* species is used-up or produced we must regard their Stoichiometric coefficients[15]. The Stoichiometric coefficient of a given species is simply the coefficient of said species in the equation for the relevant reaction. For our One-Step Mechanism we have 1 molecule of Methane reacting with 2 molecules of Oxygen, resulting in 1 molecule of Carbon-dioxide and 2 molecules of Water. Thus the Stoichiometric coefficients for CH_4, O_2, CO_2 and H_2O are 1,2,1,2 respectively. Combining these with our overall reaction rate we can determine the rate for each individual species as follows:

$$\begin{gathered}\frac{d[CH_4]}{dt} = -1 \cdot rate \\ \frac{d[O_2]}{dt} = -2 \cdot rate \\ \frac{d[CO_2]}{dt} = 1 \cdot rate \\ \frac{d[H_2O]}{dt} = 2 \cdot rate\end{gathered}$$

To implement the One-Step Mechanism the following (pseudo-)code is used to define the Differential Equation which is then solved using DifferentialEquations.jl.

Algorithm 2: One-Step Mechanism

```
coefficients = $[-1, -2, 1, 2]$
**Function** *Differential Equation*
    **for** $i \in 1 : n_n$ **do**
        $rate = k \cdot [CH_4]_i \cdot [O_2]_i$;
        **for** $j \in 0 : n_s - 1$ **do**
            $\dot{u}[i \cdot n_s - j]$ =coefficient[end -j]$\cdot rate$;
        **end**
    **end**
    $U = transpose(unstack(u))$;
    $\dot{u}+ = stack(AU)$;
**end**
```

4.1 Results

The One-Step Mechanism was implemented using the parameter values[11] found in Table 2.

Parameter	Value
A	1.1e10
T	1e3
β	0
E	2e4
R	8.3145

Table 2: Parameter values for the One-Step Mechanism [11]

And using the following IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1
- $[O_2] = 1$, initial value for O_2 in node 2
- $[O_2] = 1$, initial value for O_2 in node 4

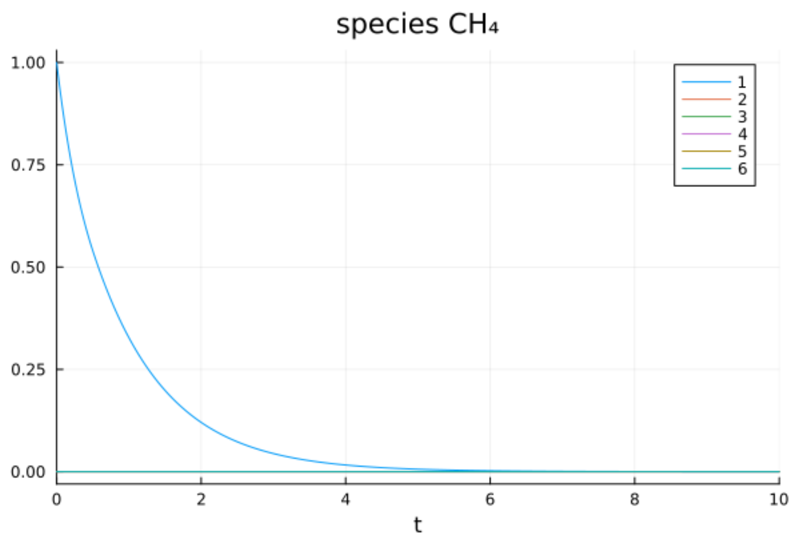


Figure 21: Concentrations of CH_4

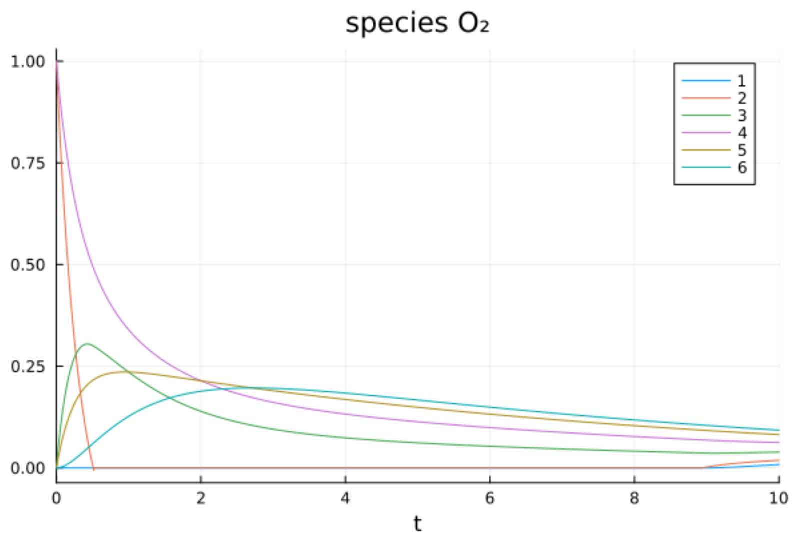


Figure 22: Concentrations of O_2

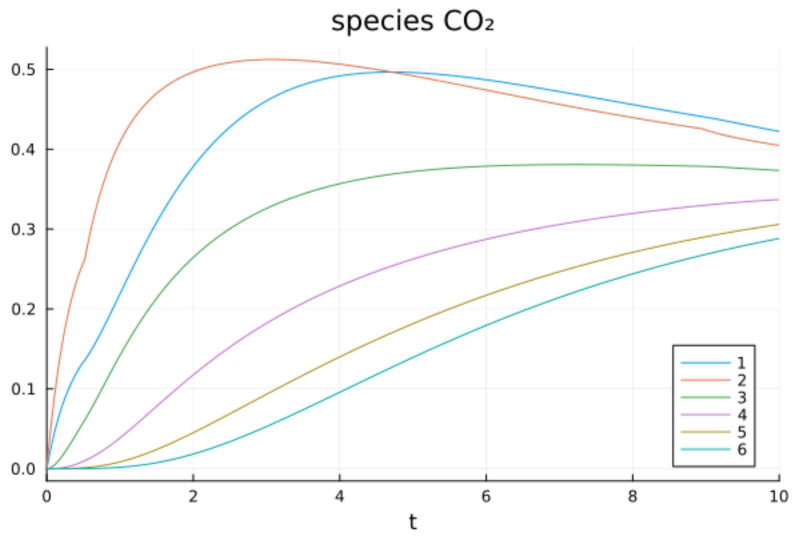


Figure 23: Concentrations of CO_2

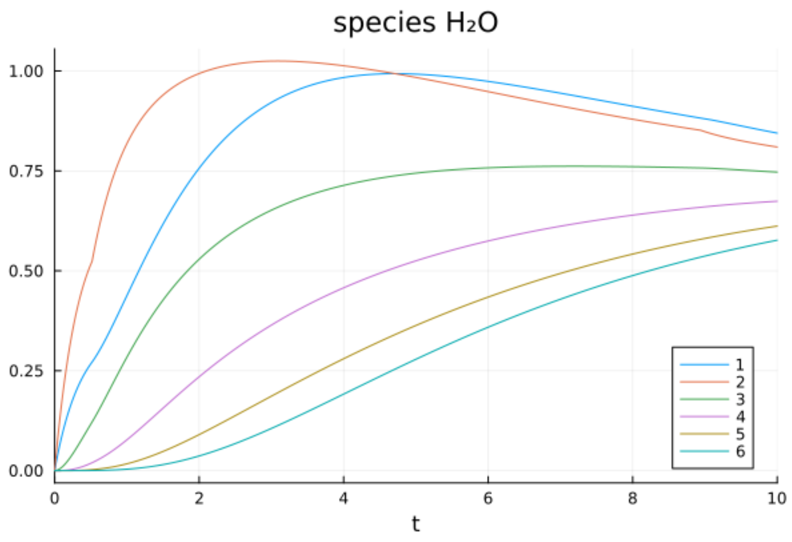


Figure 24: Concentrations of H_2O

5 GRI 3.0 database

GRI 3.0 is a state of the art database containing all manner of data concerning combustion reactions and the species involved[18].

5.1 Chemical Kinetics

In highschool students are taught, as with most matters relating to the sciences, a simplified version of combustion reactions. For example, the simplest combustion reaction possible is that of Hydrogen:

$$2H_2 + O_2 \rightarrow 2H_2O$$

We are taught that Hydrogen and Oxygen react thus producing Water. In reality however there are many more elementary reactions that occur in between, even for such a simple reaction as the combustion of hydrogen we get the following list of elementary reactions that actually occur[18]:

- H2+O2=OH+OH
- H2+OH=H2O+H
- H+O2=OH+O
- O+H2=OH+H
- H+O2+M=HO2+M
- H+O2+O2=HO2+O2
- H+O2+N2=HO2+N2
- OH+HO2=H2O+O2
- H+HO2=OH+OH
- O+HO2=O2+OH

- OH+OH=O+H2O
- H2+M=H+H+M
- O2+M=O+O+M
- H+OH+M=H2O+M
- HO2+H=H2+O2
- HO2+HO2=H2O2+O2
- H2O2+M=OH+OH+M
- H2O2+H=H2+HO2
- H2O2+OH=H2O+HO2

The more complex the fuel the more complex the reaction and the more elementary reactions we get. If we then wish to look at the combustion of natural gas, which is a mix of multiple fuels, well you can just imagine how complicated it gets. But in fact you will not have to use your imagination, you can simply look at the GRI 3.0 database which was designed for this very purpose, it contains the 53 species and 325 elementary reactions involved in the combustion of natural gas using air (rather than simply pure oxygen). This is also much closer to industry application where natural gas is used rather than just methane, and air rather than pure oxygen (which is rather expensive, whilst air is free).

On top of this, these elementary reactions greatly affect the speed at which the combustion occurs both in reality and in our model. This will affect how closely our model can mimic reality with regards to the rate at which concentrations of species change, which also affects diffusion (e.g. there can be no diffusion if a species is already used-up), and the rate at which temperature changes (see section 6). Where thermochemistry deals only with the initial and final states, chemical kinetics deals with the speed at which chemistry proceeds.

This is the essence of Chemical Kinetics.

5.2 CHEMKIN format

GRI 3.0 contains a grimech30 file which is the chemical-kinetics file, and a thermo30 file which is the thermodynamic file[18]. They are stored in the CHEMKIN format (specifically CHEMKIN-II), the relevant parts of which will now be explained.

The grimech30 file will look as follows:

```
ELEMENTS
O  H  C  N  AR
END

SPECIES
H2      H       O       O2      OH      H2O     HO2     H2O2
C       CH      CH2     CH2(S)  CH3     CH4     CO      CO2
...(etc)...
END

!THERMO
! Insert GRI-Mech thermodynamics here or use in default file
!END

REACTIONS
REACTIONS  KJOULES/MOLE  MOLECULES
2O+M<=>O2+M                             1.200E+17   -1.000        .00
...(etc)...
END
```

Figure 25: grimech30 example

As you can see in figure 25 the file contains:

1. An elements block

   Which contains a list of the chemical elements involved

2. A species block

   Which contains a list of the 53 species involved

3. A thermo block

   This where the thermo30 file is inserted

4. A reactions block

   Which contains a list of the 325 reactions.

The reaction block has 4 columns, which contain the following pieces of information, in order from left to right:

i. The chemical reaction equation

ii. The Arrhenius' Law parameter A

iii. The Arrhenius' Law parameter β

iv. The Arrhenius' Law parameter E

The thermo30 file look as follows:

```
THERMO
O                 L 1/90O   1              G   200.000  3500.000  1000.000    1
2.56942078E+00-8.59741137E-05 4.19484589E-08-1.00177799E-11 1.22833691E-15    2
2.92175791E+04 4.78433864E+00 3.16826710E+00-3.27931884E-03 6.64306396E-06    3
-6.12806624E-09 2.11265971E-12 2.91222592E+04 2.05193346E+00                   4
O2                TPIS89O   2              G   200.000  3500.000  1000.000    1
3.28253784E+00 1.48308754E-03-7.57966669E-07 2.09470555E-10-2.16717794E-14    2
-1.08845772E+03 5.45323129E+00 3.78245636E+00-2.99673416E-03 9.84730201E-06    3
-9.68129509E-09 3.24372837E-12-1.06394356E+03 3.65767573E+00                   4
...(etc)...
```

Figure 26: thermo30 example

As you can see in figure 26 the file contains an entry for each species. Where each entry is 4 lines (as is indicated on the right hand side). Lines 2,3 and 4 contain 14 numbers.

These numbers are used for the NASA polynomials:

$$\begin{aligned} \frac{Cp}{R} &= a_1 + a_2T + a_3T^2 + a_4T^3 + a_5T^4 \\ \frac{H}{RT} &= a_1 + a_2T/2 + a_3T^2/3 + a_4T^3/4 + a_5T^4/5 + a_6/T \\ \frac{S}{R} &= a_1\ln(T) + a_2T + a_3T^2/2 + a_4T^3/3 + a_5T^4/4 + a_7 \end{aligned} \tag{6}$$

We will be using a YAML formatted GRI file, which contains the information of the grimech30 and thermo30 files[19]. This way we can use the human-friendly YAML format[12], which we can parse using the YAML.jl package[3].

The gri30.yaml file will look as follows:

```
phases:
- name: gas
        thermo: ideal-gas
        elements: [O, H, C, N, Ar]
        species: [H2, H, O, O2, OH ...(etc)..]
        kinetics: gas
        transport: mixture-averaged
        state: {T: 300.0, P: 1 atm}
species:
- name: H2
        composition: {H: 2}
        thermo:
                model: NASA7
                temperature-ranges: [200.0, 1000.0, 3500.0]
                data:
                - [2.34433112, 7.98052075e-03, -1.9478151e-05, 2.01572094e-08,
                 -7.37611761e-12, -917.935173, 0.683010238]
                - [3.3372792, -4.94024731e-05, 4.99456778e-07,
                 -1.79566394e-10, 2.00255376e-14, -950.158922, -3.20502331]
                note: TPIS78
        transport:
                model: gas
                geometry: linear
                well-depth: 38.0
                diameter: 2.92
                polarizability: 0.79
                rotational-relaxation: 280.0
...(etc)...
reactions:
- equation: 2 O + M <=> O2 + M  # Reaction 1
        type: three-body
        rate-constant: {A: 1.2e+17, b: -1.0, Ea: 0.0}
        efficiencies: {H2: 2.4, H2O: 15.4, CH4: 2.0, CO: 1.75, CO2: 3.6, C2H6: 3.0,
                AR: 0.83}
...(etc)...
```

Figure 27: gri30.yaml example

As you can see in figure 27 the file contains:

1. A phases block

    Which contains a list of the chemical elements, and species involved.

2. A species block

Which contains a list of the 53 species involved. Where thermo contains the thermodynamic information of the respective species from the thermo30 file. With `data` containing the NASA polynomial coefficients for the respective species.

3. A reactions block

   Which contains a list of the 325 reactions. With the Arrhenius' Law parameter values for each reaction.

Occasionally there will be an entry in the reactions block which looks like this:

```
- equation: O + CO (+M) <=> CO2 (+M)  # Reaction 12
type: falloff
low-P-rate-constant: {A: 6.02e+14, b: 0.0, Ea: 3000.0}
high-P-rate-constant: {A: 1.8e+10, b: 0.0, Ea: 2385.0}
efficiencies: {H2: 2.0, O2: 6.0, H2O: 6.0, CH4: 2.0, CO: 1.5, CO2: 3.5,
        C2H6: 3.0, AR: 0.5}
```

Figure 28: gri30.yaml Reaction equation example

In Figure 28 we see that there are both Low Pressure and High Pressure parameter values. We will be using the low pressure as Industrial Furnaces generally do not operate using high pressures[5]. Note: M can be ignored when doing any calculation, it is a special dummy character symbolising all species.

5.3 Handling GRI

5.3.1 Extracting Stoichiometric Coefficients

Extract a vector of all species, we will call this vector: species.

$$species = \begin{bmatrix} "H2O" \\ "H" \\ "O" \\ \vdots \end{bmatrix}, \text{ a 53 element-vector}$$

Per convention for every reaction the species on the Left Hand Side (henceforth LHS) are called the reactant species and the species on the Right Hand Side (henceforth RHS) are called the product species.

$$reactants \rightleftarrows products, \text{ reversible reactions}$$
$$reactants \rightarrow products, \text{ non-reversible reactions}$$

For each reaction equation we extract the molar stoichiometric coefficients using string manipulation and put these into 2 vectors.

For example let's take reaction equation 1 in the GRI database: "2 O + M <=> $O2$ + M" This is split into a vector:

$$eq = \begin{bmatrix} "2" \\ "O" \\ "+" \\ "M" \\ "<=>" \\ "O2" \\ "+" \\ "M" \end{bmatrix}$$

Which in turn is split into 2 vectors, *eq_l* and *eq_r* for the LHS and RHS respectively:

$$eq\_l = \begin{bmatrix} "2" \\ "O" \\ "+" \\ "M" \end{bmatrix}, eq\_r = \begin{bmatrix} "O2" \\ "+" \\ "M" \end{bmatrix}$$

We extract the stoichiometric coefficient of every species in *eq_l* and *eq_r*. From *eq_l* we get 2 for species "O". From *eq_r* we get 1 for species "$O2$". This is done by parsing to Float64 the vector element at the index preceding that of a species in the vector.

These coefficients are put into two vectors, v_l_j and v_l_r for the LHS and RHS of the j-th reaction respectively. These vectors are of 53-elements (number of species) such that the coefficients are at the index of the species.

For our example with reaction equation 1: The only species in the LHS is "O" with coefficient 2. The index of species "O" in the *species* vector is 3. So we make a vector with 3rd entry equal to 2.

$$v\_l\_1 = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \text{ similarly for RHS } v\_r\_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

The above is done for all 325 reactions. These column vectors are then combined into into 2 matrices, matrix v_l and v_r for the LHS and RHS respectively.

$$v\_l = \begin{bmatrix} v\_l\_1 & v\_l\_2 & \dots & v\_l\_325 \end{bmatrix}, v\_r = \begin{bmatrix} v\_r\_1 & v\_r\_2 & \dots v\_r\_325 \end{bmatrix}$$

Such that *v_l* and *v_r* are each of the following form:

$$\begin{array}{c|c}
 & reaction1\downarrow \quad reaction2\downarrow \quad \ldots \quad reaction325\downarrow \\
\begin{array}{l} species1 \rightarrow \\ species2 \rightarrow \\ \vdots \\ species53 \rightarrow \end{array} & \left(\begin{array}{c} \\ \\ \\ \\ \end{array}\right)
\end{array}$$

The implementation for this is of the following structure:

Algorithm 3: Stoichiometric Coefficient Matrices *v_l* and *v_r*

```
Function Make matrices
    v_l = zeros($n_s$,$n_r$);
    v_r = zeros($n_s$,$n_r$);
    for $j \in 1 : n_r$ do
        Extract equation j from gri30.yaml ;
        split equation j into eq_l and eq_r;
        for $i \in 1 : n_s$ do
            if species $i \in$ eq_l then
                get index species of species i in eq_l;
                try to parse to Float64 eq_l[index-1]:
                    coeff = the parsed float
                catch e:
                    coeff =1
                end
                v_l[i,j]=coeff
            else if species $i \in$ eq_r then
                get index species of species i in eq_r;
                try to parse to Float64 eq_r[index-1]:
                    coeff = the parsed float
                catch e:
                    coeff =1
                end
                v_r[i,j]=coeff
            end
        end
    end
    return v_l, v_r
end
```

5.4 GRI 3.0 constant temp, forward rate only

5.4.1 In 1 node, for 1 reaction

For any reaction, say the *n*-th reaction, we extract the coefficients:

- A_k_n = pre-exponential term
- β_n = temperature exponent
- E_n = activation energy

Which are used to calculate Arrhenius' law:

$$k_n = A\_k_n \cdot T^{\beta_n} \cdot exp(-\frac{E_n}{RT}), \text{ for arbitrary } n \tag{7}$$

To calculate the (forward) rate we need to multiply by the concentrations of the reactant-species of the reaction.

How do we know a species is a reactant in the n-th reaction?
We look at the stoichiometric coefficient matrix v_l, if the entry for species j in reaction n is non-zero then species j is a reactant in reaction n. This entry, off course, is simply $v\_l[j, n]$.
We set: $rate_n = k_n$ then for every reactant species j we multiply the rate by $[X_j]$ such that the overall (forward) reaction rate of reaction n is given by:

$$rate_n = k_n \prod_j [X_j]$$

For reaction n we now have the overall (forward) rate. This can now be used to assign a value to the rate of change of concentration.
For every reactant species j in reaction n:

$$\frac{d[X_j]}{dt} = -(\text{stoichiometric coefficient}) \cdot rate_n = -v\_l[j, n] \cdot rate_n$$

For every product species j in reaction n:

$$\frac{d[X_j]}{dt} = +(\text{stoichiometric coefficient}) \cdot rate_n = +v\_r[j, n] \cdot rate_n$$

Note the difference in signs and the difference in stoichiometric coefficient matrices. Also note that this reaction rate not oly differs per species but is different for each reaction and each node.

5.4.2 For all nodes, for all reactions

Our implementation must repeat this for all nodes and reactions. To do so the following (pseudo-)code is used to define the Differential Equation which is then solved using DifferentialEquations.jl.

Algorithm 4: GRI 3.0

```
Function Differential Equation
    for $i \in 1 : n_n$ do
        for $i \in 1 : n_r$ do
            Extract: $A\_k_n, \beta_n, E_n$ ;
            $k_n = A\_k_n \cdot T^{\beta_n} \cdot exp(-\frac{E_n}{RT})$;
            $rate_n = k_n$;
            for $j \in 1 : length(v\_l[:, n])$ do
                if $v\_l[j, n]! = 0$ then
                    $rate = -v\_l[j, n] \cdot rate_n$
                end
            end
            for $j \in 1 : length(v\_l[:, n])$ do
                if $v\_l[j, n]! = 0$ then
                    $\dot{u}_i[j] + = -$(stoichiometric
                      coefficient)$\cdot rate_n = -v\_l[j, n] \cdot rate_n$
                end
                if $v\_r[j, n]! = 0$ then
                    $\dot{u}_i[j] + = +$(stoichiometric
                      coefficient)$\cdot rate_n = -v\_r[j, n] \cdot rate_n$
                end
            end
        end
    end
    $U = transpose(unstack(u))$;
    $\dot{u} + = stack(AU)$;
end
```

Where n_r is the number of reactions.

5.5 Results

We implement the GRI mechanism with forward reactions only, on the following graph.

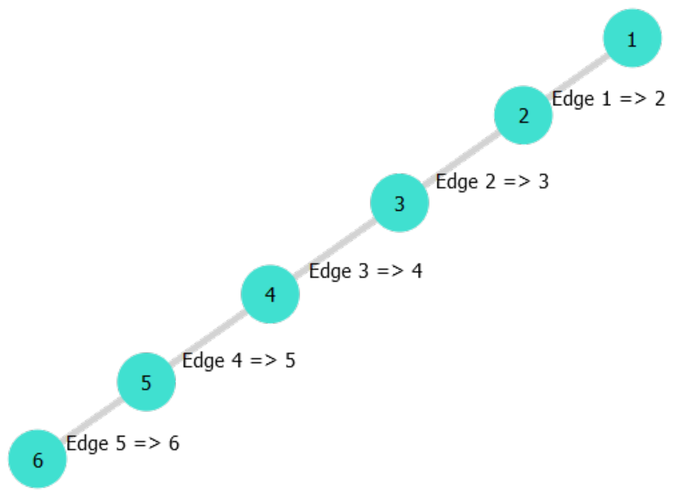


Figure 29: Graph 1 rendered using GraphPlot.jl

Using the following IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1
- $[O_2] = 1$, initial value for O_2 in node 2
- $[O_2] = 1$, initial value for O_2 in node 4

There are no forward reactions in the GRI 3.0 mechanism with just the two species CH_4 and O_2, we would therefore expect no reactions to occur. Thus only diffusion should be happening. Which is exactly what we can see in Figures 30 and 31.

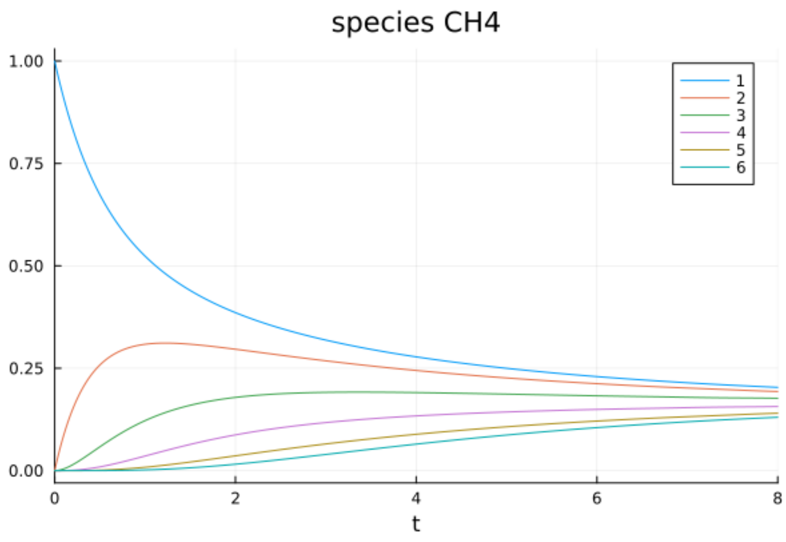


Figure 30: Concentrations of CH_4

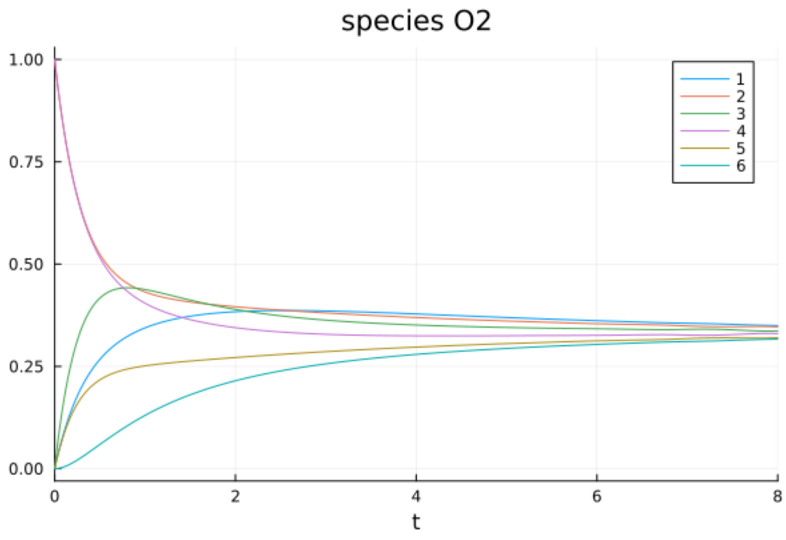


Figure 31: Concentrations of O_2

Let us now showcase GRI using species for which a forward reaction is in the GRI database. For example, one such reaction is

$$O + CH_4 \rightleftarrows OH + CH3$$

We will use the same graph as before. But this time we will use the following

IC:

- $[CH_4] = 1$ initial value for CH_4 in node 1
- $[O] = 1$, initial value for O in node 2
- $[O] = 1$, initial value for O in node 4

A myriad of species are produced in varying amounts via multiple different reaction. However it is immediately confirmed that the GRI mechanism is not designed to be used with only a forward reaction rate implemented. As you can see with the species below the amounts are off huge proportions, because the forward reaction rates are not tempered by backward reaction rates.

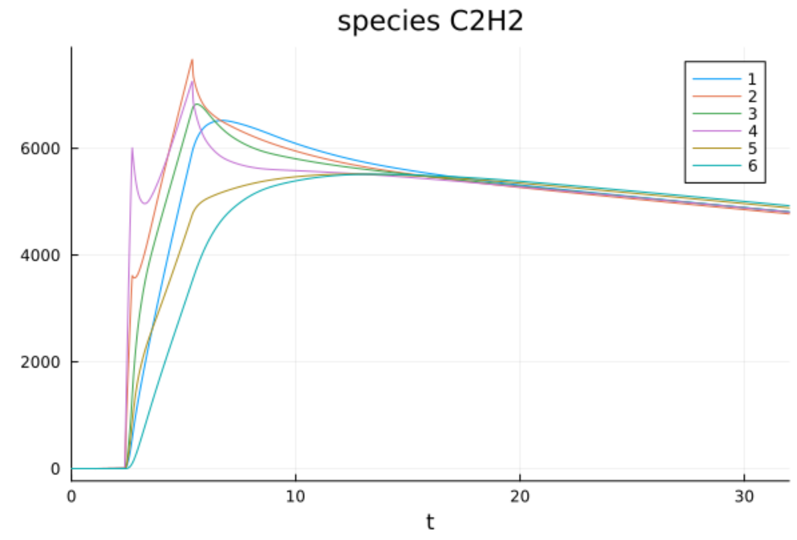


Figure 32: Concentrations of C_2H_2

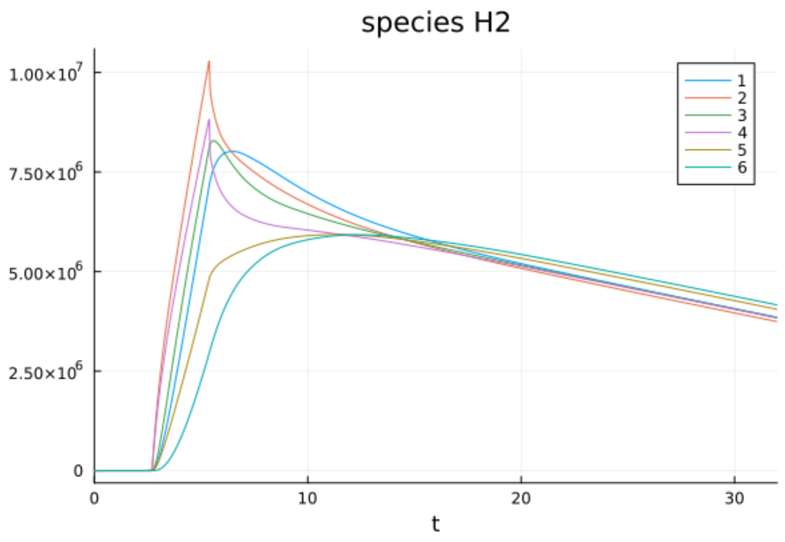


Figure 33: Concentrations of H_2

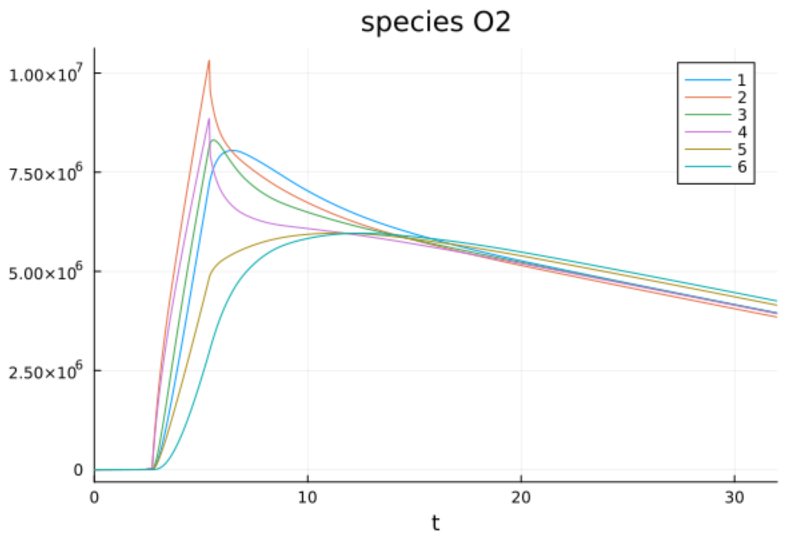


Figure 34: Concentrations of O_2

6 Future Expansions

This section pertains to matters for which the author regrettably did not have the required time to implement, due to the limited duration of a bachelor project.

6.1 Diffusivity coefficient

The diffusivity coefficient D may vary per gas and can be a function of temperature. There are a few different possible implementations which we will discuss in an increasing order of complexity.

Firstly one could take the diffusivity constant of the predominant gas in the furnace. Most practical burners operate using lean combustion, which is when the oxidizer is in excess. Meaning that the fuel-air filled furnace contains mostly air. Hence the properties such as the molecular weight, transport properties and heat capacities are not significantly changed compared to just air[15]. The diffusivity coefficient varies with a power of 1.5-1.8 of the temperature[4]. One could take a tabulated value, D_0, for the diffusion coefficient of air and use that to calculate D as follows:

$$D = D_0 \cdot T^{1.65}$$

Secondly, one could take D_0 as an average of the diffusivity coefficients of the various species weighted according to their concentration in each section (which will vary over time).

Thirdly, one could use the Chapman-Enskog Theory[4]. This theory gives the following equation

$$D = \frac{1.86 \cdot 10^{-3} T^{3/2} (1/M_1 + 1/M_2)^{0.5}}{p \sigma_{12}^2 \Omega}$$

Where

- p is the pressure in atmospheres
- M_i are molecular weights
- $\sigma_{12} = 0.5(\sigma_1 + \sigma_2)$ is the collision diameter
- Ω a complicated dimensionless integral which depends on the interaction between the two species

Here p can be taken constant, this is one of the assumptions of our model. Since for most deflagrations (subsonic combustion) p is almost constant[15].
For M_i and σ_i tabulated values can be used for the two most predominant gases in a section according to their concentrations (which will vary over time).
For Ω tabulated values can be used, or one could use

$$\Omega = T k_B / \epsilon_{12}$$

Where tabulated values can be found for Tk_B/ϵ_{12} for the two most predominant gases in a node according to their concentrations (which will vary over time).

Finally, one could repeat all of the above except with temperature as a variable rather than a constant.

6.2 GRI 3.0 backward reaction rates

Currently for GRI 3.0 only the forward reaction rates have been implemented. In order to complete the GRI model the backward reaction rates should also be implemented. There are a few different possible implementations which we will discuss in an increasing order of complexity.

Firstly, one could use pre-calculated values for the backwards reactions rates for each equation. These rates are calculated for specific values of T. Hence if one is using T as a constant then this would be a good option.

Secondly, one could use the same pre-calculated values when using T as a variable. In this case one could simply use the pre-calculated value for a temperature that is closest to T. One could also take some form of weighted average of the two pre-calculated values that are closest to T.

Thirdly, if we denote the forward and backward reaction rate for a single reaction by k_f and k_b respectively. Then we can denote the reaction rates for arbitrary reaction j by k_{fj} and k_{bj}. Where k_{fj} is calculated, as before, using Arrhenius' law as in Equation (7). The backward reaction rate can then be calculated using[15]

$$k_{bj} = \frac{k_{fj}}{(\frac{1}{RT})^{\sum_{i=1}^{n_s} v_{ij}} exp(\frac{\Delta S_j^0}{R} - \frac{\Delta H_j^0}{RT})} \tag{8}$$

Where

- $v_{ij} = v_{ij}^{''} - v_{ij}^{'}$ with $v_{ij}^{''}$ and $v_{ij}^{'}$ being the LHS and RHS stoichiometric coefficients for species i in reaction j, respectively
- ΔH_j^0 and ΔS_j^0 are the mass enthalpy and entropy changes for reaction j at reference temperature $T_0 = 298.15K$, which can be obtained from tabulations.

For all of the implementations of backward reaction rates we will require extra information, which unfortunately is not included in the GRI files. This makes implementing GRI a much more inconvenient affair, not due to computation or mathematical complexity but simply due to a lack of information. For a more about this issue please look at Appendix C.

6.3 Temperature

So far the temperature has been implemented as a constant value. Besides its inherent importance as a quantity to keep track of, it can also affect the reaction rates. Thus it would be a valuable addition to the model. Firstly, we must add T to the model in order to be able to track it. This can be done by adding an extra entry to every u_i as such:

$$u_i = \begin{bmatrix} [X_1] \\ \vdots \\ [X_{n_s}] \\ T_i \end{bmatrix}, \text{ where } T_i \text{ is the Temperature in node } i \tag{9}$$

Secondly we must model the temperature change over time. The simplest of the multiple equations which govern this[15] is:

$$\rho C_p \frac{dT}{dt} = \dot{w}_T^{'} + \frac{\partial}{\partial x_i}(\lambda \frac{\partial T}{\partial x_i}) + \dot{Q} + \rho \sum_{k=1}^{n_s} Y_k f_{k,i} V_{k,i}$$

Where

- ρ the density of the multi-species gas.
- C_p is the heat capacity
- $\dot{w}_T^{'}$ is heat release due to combustion.
- λ is the thermal conductivity of the gas.
- $\dot{Q}$ is a heat source term, e.g due to the electric spark used to ignite the gas.
- $\rho \sum_{k=1}^{n_s} Y_k f_{k,i} V_{k,i}$ is a convection term.

Here ρ can be calculated[15] using $\sum_{i=1}^{n_s} \rho_i$.
The heat capacity in this case is assumed to be equal for all gases, this is often used[15]. One could use the heat capacity of air because the fuel-air gas properties are not significantly changed compared to just pure air, this was discussed in section 6.1. The thermal conductivity can be treated similarly.
The heat source, due to being so short lived, is negligible.
For convection one would be well advised to first consider a simpler implementation. For example one could look at the amount of diffusion going into a node from its neighbour and use the temperature difference between them to calculate an increase or decrease in temperature.
We can calculate $\dot{w}_T^{'}$ [15]using

$$\dot{w}_T^{'} = -\sum_{i=1}^{n_s} \Delta h_{f,i}^0 \dot{w}_i$$

Where $\Delta h^0_{f,i}$ is the mass formation enthalpy of species i at temperature $T_0 = 298.15K$.

The molar formation enthalpies can be found at [20], which can then be converted to mass formation enthalpy. Another option is to use the thermo file to and NASA polynomials (6) to calculate the formation enthalpy.

7 Conclusion

Chemical Reactor Networks are a versatile way of modelling combustion processes. They are capable of handling both the diffusion and chemistry at the same time. The spatial discretisation used combines both Graph Theory and Numerical Methods in a very neat and organised manner.

The way in which the diffusion is handled is independent from the chemistry that occurs. Hence the single-step mechanism can be readily implemented and could be just as readily replaced by a two or four-step mechanism, without diffusion related complications. On top of this CRN is clearly able to handle diffusion on very large and complicated graphs without painful calculation times.

Unfortunately the GRI 3.0 files do not contain all of the necessary information which is required to produce a fully implemented CRN model of the GRI mechanism. However given enough time and persistence this information could be gathered, at which point it would be a seemingly simple task to complete the implementation.

8 Further practical resources

On top of the references that have been mentioned so far, there are a few honourable mentions. These are specifically of very practical value.

- The textbook [15] has an associated website[6] which also contains the lectures associated with the book. Lecture 1 provides a good introduction for the general topic of combustion and motivates the study thereof.
- One of the authors of [15], Thierry Poinsot, has also written an article[14] entitled "How to successfully fail in CFD". This article is written in an entertaining yet instructive manner and contains valuable reminders of mistakes to avoid when designing and implementing models.
- The lead developer of DifferentialEquations.jl, Dr. Christopher Rackauckas, has a very useful video[17] that serves as an introduction to DifferentialEquations.jl.

Bibliography

[1] Cesar O. Aguilar. *Graph Theory.* URL: `https://www.geneseo.edu/~aguilar/public/notes/Graph-Theory-HTML/ch4-laplacian-matrices.html` (visited on June 16, 2023).

[2] Robert W. Balluffi, Samuel M. Allen, and W. Craig Carter. *KINETICS OF MATERIALS.* 2005.

[3] Kevin Bonham. *YAML.jl: a flexible data serialization format that is designed to be easily read and written by human beings.* 2013. URL: `https://github.com/JuliaData/YAML.jl`.

[4] E.L. Cussler. *Diffusion: Mass Transfer in Fluid Systems.* Cambridge Series in Chemical Engineering. Cambridge University Press, 2009. ISBN: 9780521871211. URL: `https://books.google.ch/books?id=dq6LdJyN8ScC`.

[5] Xavier D'Hubert. *Latest burner profiles.* Global Cement Magazine. 2017. URL: `https://www.globalcement.com/magazine/articles/1018-latest-burner-profiles`.

[6] *eLearning @ Cerfacs - Online Courses.* URL: `https://elearning.cerfacs.fr/` (visited on June 23, 2023).

[7] James Fairbanks et al. *JuliaGraphs/Graphs.jl: an optimized graphs package for the Julia programming language.* 2021. URL: `https://github.com/JuliaGraphs/Graphs.jl/`.

[8] *Individual reactions.* URL: `http://combustion.berkeley.edu/gri-mech/data/frames.html` (visited on June 23, 2023).

[9] *Julia Standard Library LinearAlgebra.* URL: `https://docs.julialang.org/en/v1/stdlib/LinearAlgebra/` (visited on June 21, 2023).

[10] J. van Kan et al. *Numerical methods for Partial Differential Equations.* 2019. ISBN: 9789065624383.

[11] van der Linden L. "The parareal algorithm on the model for combustion of methane." In: *Delft University of Technology* (2020). URL: `http://resolver.tudelft.nl/uuid:6839f1d6-002a-4415-b68b-848505cf6ada`.

[12] Ingy döt Net et al. *YAML format: YAML Ain't Markup Language version 1.2.* 2021. URL: `https://yaml.org/` (visited on June 20, 2023).

[13] Hector Perez. *GraphPlot.jl: Graph visualization for Julia.* 2015. URL: `https://github.com/JuliaGraphs/Graphs.jl/`.

[14] Thierry Poinsot. *How to successfully fail in CFD.* 2020. URL: `https://www.cerfacs.fr/elearning/index.php?id=index-7` (visited on June 23, 2023).

[15] Thierry Poinsot and Denis Veynante. *Theoretical and Numerical Combustion, 3rd edition.* Addison-Wesley Professional, 2022.

[16] Jacob Quinn et al. *HTTP client and server functionality for Julia.* 2022. URL: `https://github.com/JuliaWeb/HTTP.jl` (visited on June 23, 2023).

[17] Chris Rackauckas. *Intro to solving differential equations in Julia.* 2018. URL: `https://www.youtube.com/live/KPEqYtEd-zY` (visited on June 23, 2023).

[18] Gregory P. Smith et al. *GRI 3.0.* URL: `http://www.me.berkeley.edu/gri_mech/` (visited on June 19, 2023).

[19] Ray Speth. *GRI 3.0 in YAML format.* 2019. URL: `https://gist.github.com/speth/c8bfcf0389beba4422a334cc21828692` (visited on June 20, 2023).

[20] *Thermodynamic Data at 298K.* URL: `http://combustion.berkeley.edu/gri-mech/data/thermo_table.html` (visited on June 23, 2023).

[21] Shubhendu Trivedi. *A Note on the Graph Laplacian.* Mar. 29, 2015. URL: `https://onionesquereality.wordpress.com/2015/03/29/a-note-on-the-graph-laplacian/` (visited on June 16, 2023).

A Julia implementation

Code 1: Building a Graph

```
using DifferentialEquations, Plots, Graphs, GraphPlot,
    YAML
using Cairo, Fontconfig, Compose

n_node=6
g2 = Graph(n_node)
add_edge!(g2,1,2)
```

```
add_edge!(g2,2,3)
add_edge!(g2,3,4)
add_edge!(g2,4,5)
add_edge!(g2,5,6)
nodelabel = collect(1:Graphs.nv(g2));
edgelabel =edges(g2)
display(gplot(g2,nodelabel=nodelabel,edgelabel=edgelabel,
     edgelabeldistx=1, edgelabeldisty=1));
draw(PNG("res_diffusion_only_graph1/graph1.png"),gplot(g2
    ,nodelabel=nodelabel,edgelabel=edgelabel,
    edgelabeldistx=1, edgelabeldisty=1))
```

Code 2: Diffusion only on a Generated Graph

```
n_node=100000
g2 = barabasi_albert(n_node,3,seed=42)
n_species = 1

u0 = zeros(n_species * n_node)
u0[1] = 1. # initial value CH4 in node 1

function myrhsode2!(du,u,p,t)
        du.=0
        for i in 1:n_node
                if u[i]<0
                        u[i]=0
                end
        end
        A = -laplacian_matrix(g2)
        U = reshape(u,(n_species,n_node))'
        du[:] += reshape(transpose(A*U),(n_species*n_node
            ,1))
end




# problem set-up: set the time span
tspan = (0.0,8.0)
# problem set-up: define the ODE problem
prob = ODEProblem(myrhsode2!,u0,tspan)

# perform time integration
@elapsed sol = solve(prob); #timing it
```

Code 3: Plotting for One-Step Mechanism

```
labels=["CH4" "O2" "CO2" "H2O"]
```

```

#plot of all species for each node!
for i in 1:n_node
        idxs=(i*n_species -3):1:(i*n_species);
        display(plot(sol,idxs=idxs,title= "node $(i)",
            labels=labels)); #chemistry labels
        savefig("res_diffusion_only_graph3/graph3_node_$(
            i).png")
end

#plot of all nodes for each species
for i in 1:n_species
        idxs=i:4:n_species*n_node
        idxs2= 1:1:n_node
        idxs2=string.(collect(idxs2))
        idxs2=reshape(idxs2,(1,n_node));
        display(plot(sol,tspan=(0.0,4.0),idxs=idxs,title=
             "species $(labels[i])",labels=idxs2));
        display(plot(sol,idxs=idxs,title= "species $(
            labels[i])"));
        savefig("res_diffusion_only_graph3/
            barabasi_albert_species_$(labels[i]).png")
end
```

Code 4: Performing One-Step Mechanism

```
E = 2e4 # activation energy
B = 0. # temperature exponent
A = 1.1e10 # pre-exponential factor
R = 8.3145 # gas constant
T = 1e3 # temperature in Kelvin
k = A * T^B * exp(-E / R / T) # rate coefficient
n_node = 6
n_species = 4
multiplier = [-1,-2,1,2]
u0 = zeros(n_species * n_node)
u0[1] = 1. # initial value CH4 in node 1
u0[2n_species-2] = 1. # initial value O2 in node 2
u0[4n_species-2] = 1. # initial value O2 in node 4
function myrhsode2!(du,u,p,t)
du.=0

u[u.<0].=0



A = -laplacian_matrix(g2)
U = reshape(u,(n_species,n_node))'
```

```

#chemistry for each section
for i in 1:n_node
rate = k * u[i*n_species -3]^1 * u[i*n_species -3+1]^0.5

for j in 0:(n_species -1)
        du[i*n_species-j]=multiplier[(end -j)]*rate
end
end
        du[:] += reshape(transpose(A*U),(n_species*n_node
            ,1))
end
# problem set-up: set the time span
#tspan = (0.0,8.0)
tspan=(0.0,10.0)
# problem set-up: define the ODE problem
prob = ODEProblem(myrhsode2!,u0,tspan)

# perform time integration
@elapsed sol  = solve(prob); #timing it
```

Code 5: Make Coefficient Matrices for GRI

```
gri = YAML.load_file("gri30.yaml")

phases = gri["phases"]
species=phases[1]["species"]
n_species = length(species)
u0 = zeros(length(species)*n_node)
reactions = gri["reactions"]


function make_v(gri)
phases=gri["phases"]
species=phases[1]["species"]
reactions = gri["reactions"]
v_l = zeros(length(species),length(reactions));
v_r = zeros(length(species),length(reactions));

for j in 1:length(gri["reactions"])
        equation2 = gri["reactions"][j]["equation"]
        eq=split(equation2);

        if findfirst(item->item=="<=>",eq)!== nothing
        middle = findfirst(item->item=="<=>",eq)
        elseif findfirst(item->item=="=>",eq)!==nothing
        middle=findfirst(item->item=="=>",eq)
        end

        eq_l = eq[1:middle-1]
        eq_r = eq[middle+1:end]
        for i in 1:length(species)
                if species[i] in eq_l
                index_species = findfirst(item->item==
                    species[i],eq_l)
                try
        if tryparse(Float64, eq_l[index_species-1]) !==
            nothing
        global coeff=parse(Float64,eq_l[index_species-1])
        else
        global coeff = 1
        end
        catch e
        global coeff =1
        end
v_l[i,j]=coeff
elseif species[i] in eq_r
index_species = findfirst(item->item==species[i],eq_r)

        try
                if tryparse(Float64, eq_r[index_species
                    -1]) !== nothing
                global coeff=parse(Float64,eq_r[
                    index_species-1])
                else
                global coeff = 1
                end
        catch e
        global  coeff =1
        end
        v_r[i,j]=coeff
        end
        end
        end
        return v_l,v_r
end

v_l,v_r= make_v(gri)
```

Code 6: Using GRI and plotting

```

n_species= length(species)
u0 = zeros(n_species * n_node)
```

```
#####IC
u0[14]=1 #CH4 in node 1
u0[n_species + 4]=1 #O2 in node 2
u0[3*n_species+4]=1 #O2 in node 4
#not actual values, but merely initializing the
    parameters
E = 2e4 # activation energy
B = 0. # temperature exponent
A_k = 1.1e10 # pre-exponential factor
R = 8.3145 # gas constant
T = 1e3 # temperature in Kelvin

v_l, v_r = make_v(gri)
function myrhsode2!(du,u,p,t)
    du.=0
  u[u.<0].=0
    A = -laplacian_matrix(g2)
U = reshape(u,(n_species,n_node))'
#chemistry for each section
for i in 1:n_node
for n in 1:length(reactions)
 try
global A_k = gri["reactions"][n]["rate-constant"]["A"]
global B = gri["reactions"][n]["rate-constant"]["b"]
global E = gri["reactions"][n]["rate-constant"]["Ea"]
catch e
global A_k = gri["reactions"][n]["low-P-rate-constant"]["
    A"]
global B = gri["reactions"][n]["low-P-rate-constant"]["b
    "]
global E = gri["reactions"][n]["low-P-rate-constant"]["Ea
    "]
end
 k = A_k * T^(B) * exp(-E / R / T) # rate coefficient;

   rate = k
    for j in 1:length(v_l[:,n])
   if v_l[j,n]!=0 #this checks that j is index of a
       relevant species for reaction n
    rate = rate*(u[i*n_species-n_species + j])

   end
   end
    for j in 1:length(v_l[:,n])
   if v_l[j,n]!=0
   du[(i-1)*n_species+j]+=-v_l[j,n]*(rate)
```

```
end
if v_r[j,n]!=0
du[(i-1)*n_species+j]+=v_r[j,n]*rate
end
                end
            end
        end
        du[:] += reshape(transpose(A*U),(n_species*n_node
            ,1));
end
tspan=(0.0,8.0)
# problem set-up: define the ODE problem
prob = ODEProblem(myrhsode2!,u0,tspan)

# perform time integration
@elapsed sol  = solve(prob); #timing it


## 1 plot per species
labels = species
for i in 1:n_species
idxs=i:n_species:n_species*n_node
idxs2= 1:1:n_node
idxs2=string.(collect(idxs2))
idxs2=reshape(idxs2,(1,n_node));
display(plot(sol,idxs=idxs,title= "species $(labels[i])
    ",labels=idxs2));
savefig("res_gri_ic1/gri_species_$(labels[i]).png")
end
```

Code 7: Potential GRI Web-Scraping Method

```
k_b_all = zeros(325);
for i in 1:325
        url ="http://combustion.berkeley.edu/gri-mech/
            data/reactions/$(i)"
        r = HTTP.get(url)
        str1=String(r.body)
        println(str1)
        str2=str1[findfirst("1000",str1)[1]:end]
        str3= str2[begin:findfirst("\n",str2)[1]-1]
        vec1 = split(str3)
        println(i)
        k_b_all[i]=parse(Float64,vec1[5])
end
```

B Diffusion only

B.1 Graph 1

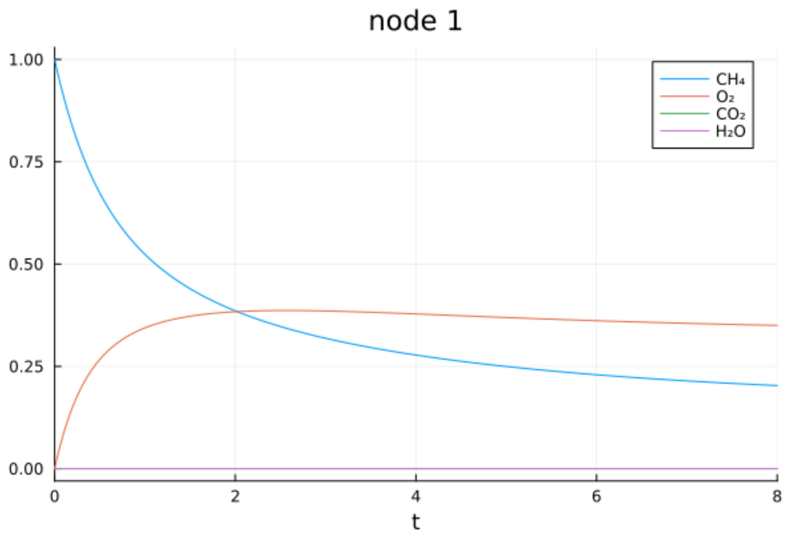


Figure 35: Graph 1 - Node 1

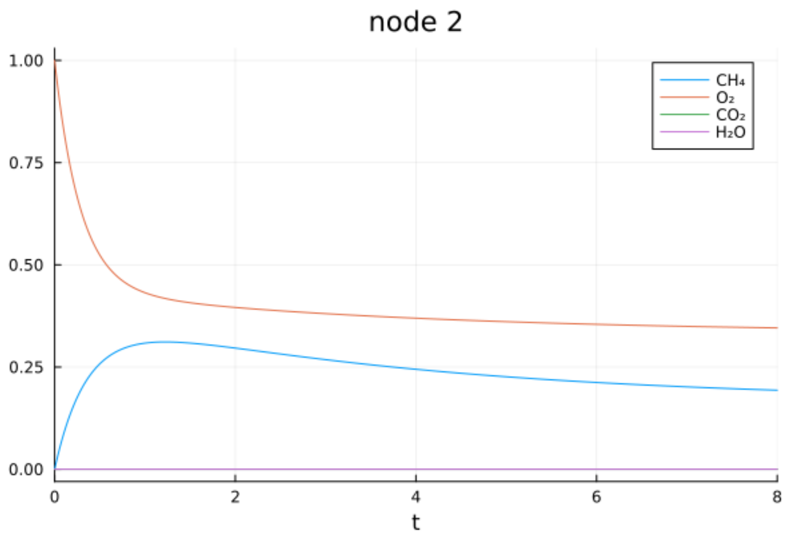


Figure 36: Graph 1 - Node 2

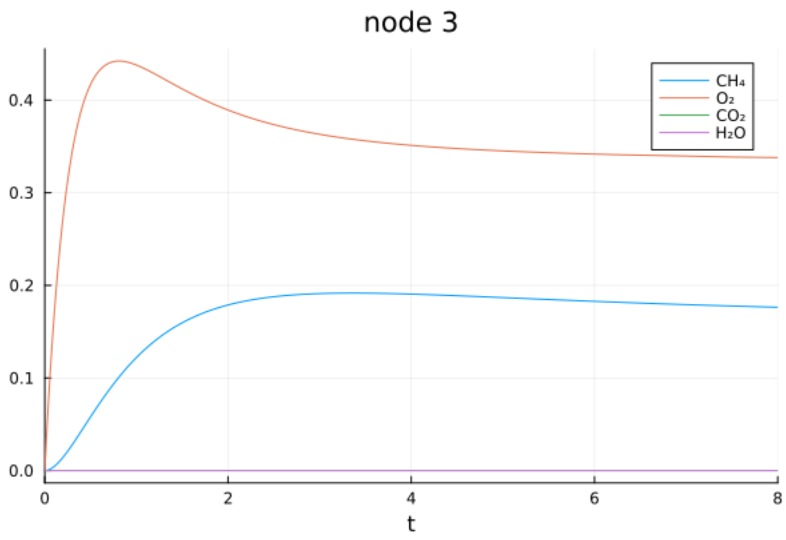


Figure 37: Graph 1 - Node 3

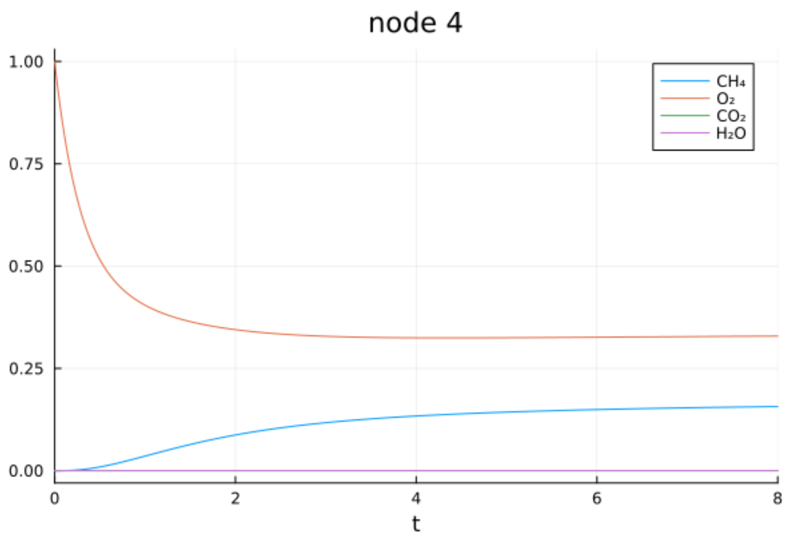


Figure 38: Graph 1 - Node 4

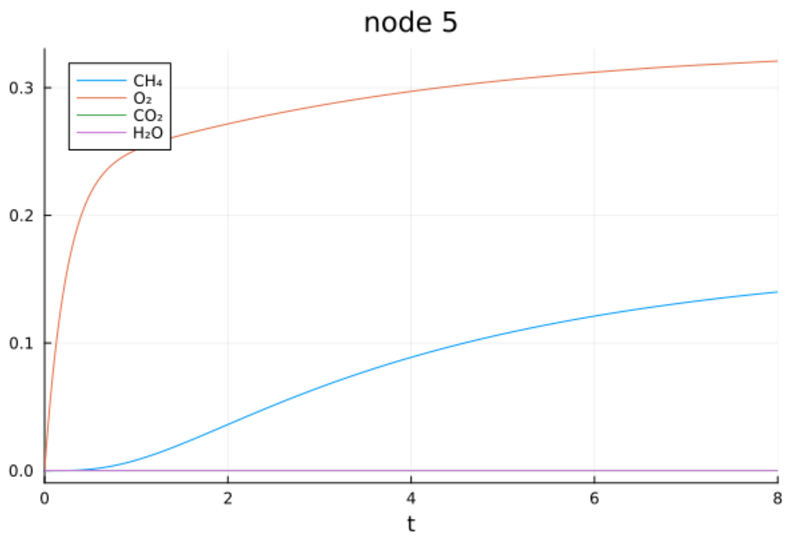


Figure 39: Graph 1 - Node 5

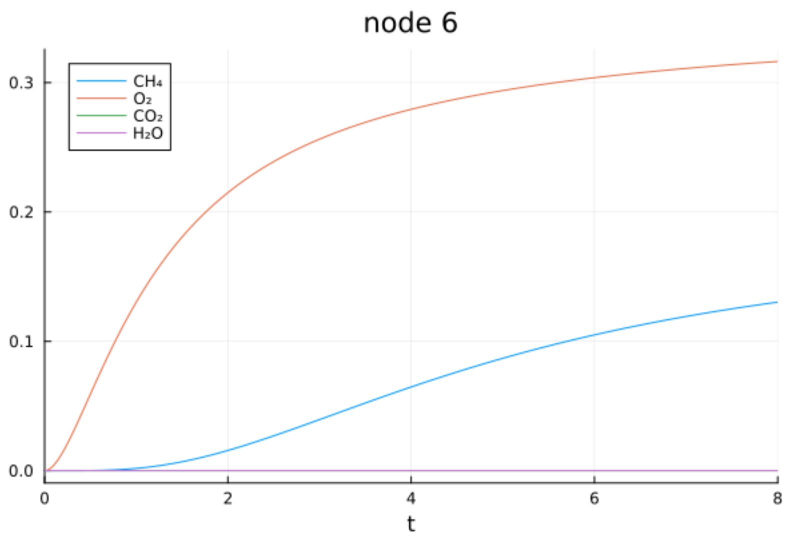


Figure 40: Graph 1 - Node 6

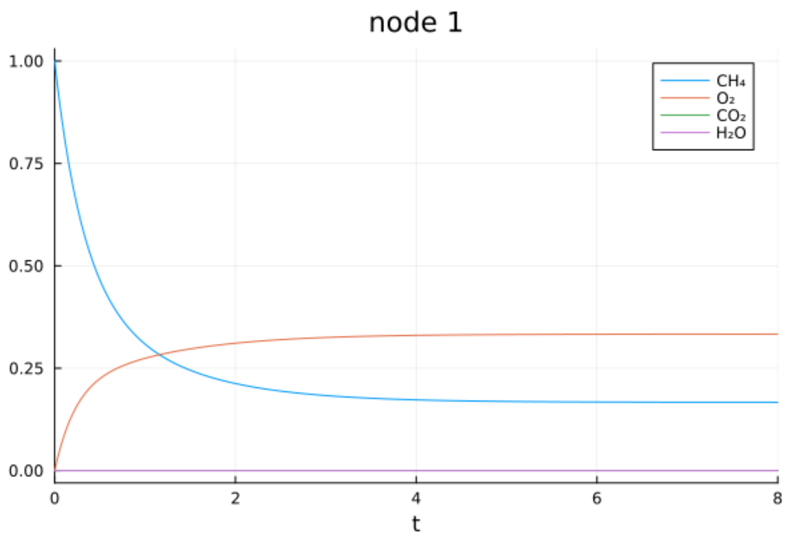


Figure 41: Graph 2 - Node 1

B.2 Graph 2

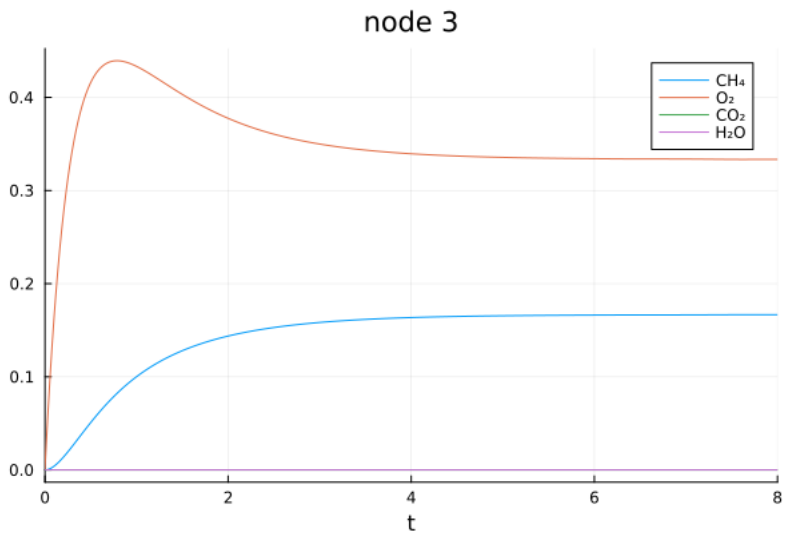


Figure 43: Graph 2 - Node 3

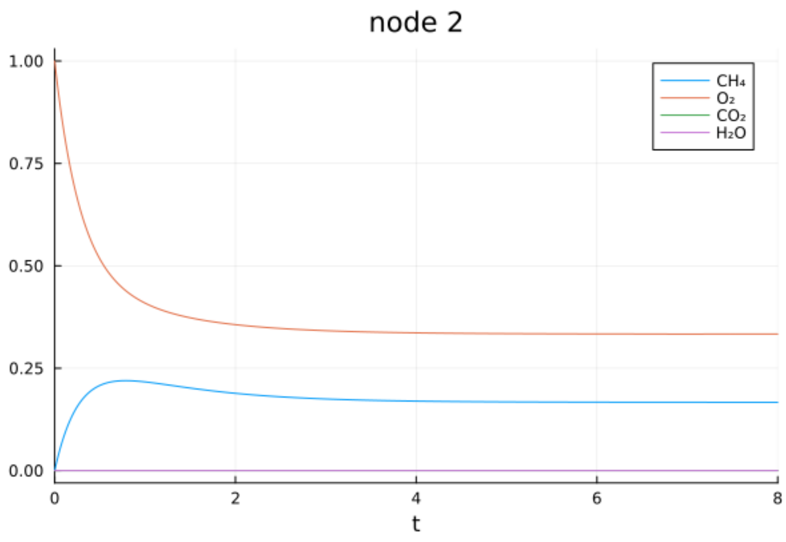


Figure 42: Graph 2 - Node 2

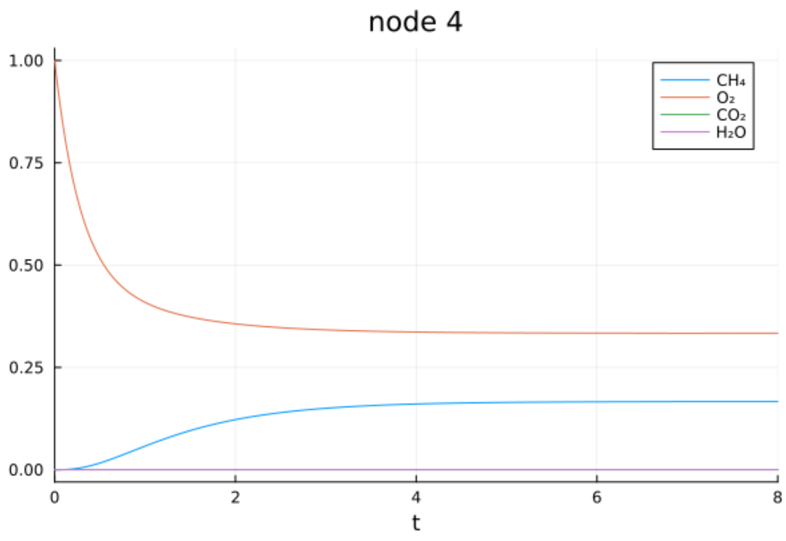


Figure 44: Graph 2 - Node 4

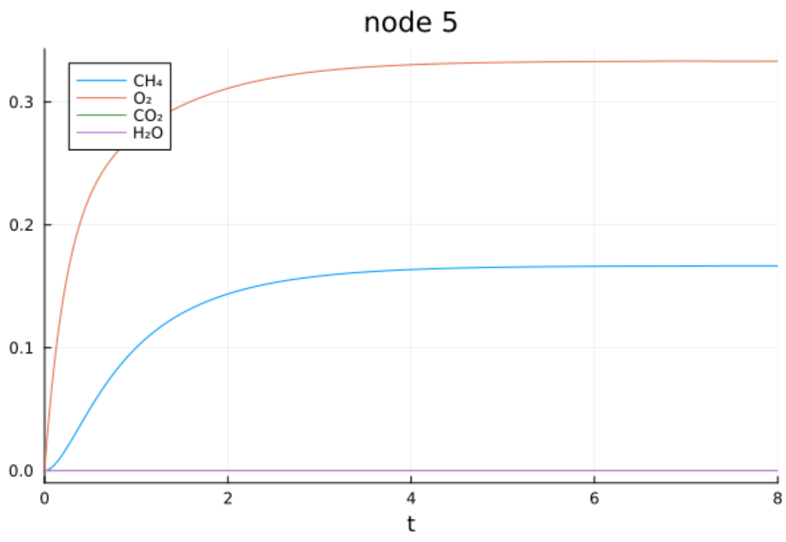


Figure 45: Graph 2 - Node 5

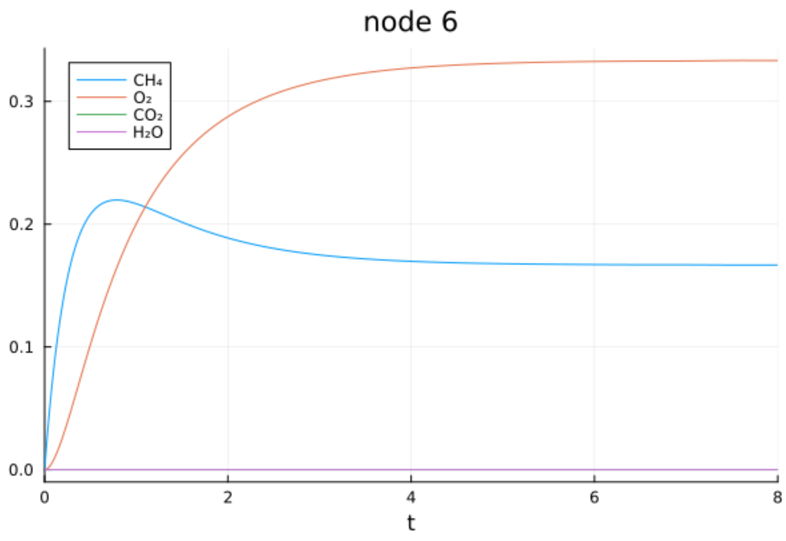


Figure 46: Graph 2 - Node 6

B.3 Graph 3

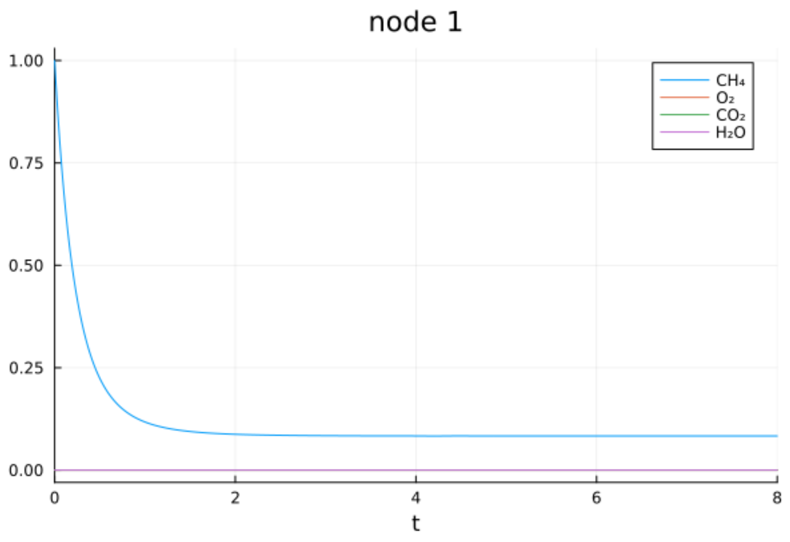


Figure 47: Graph 3 - Node 1

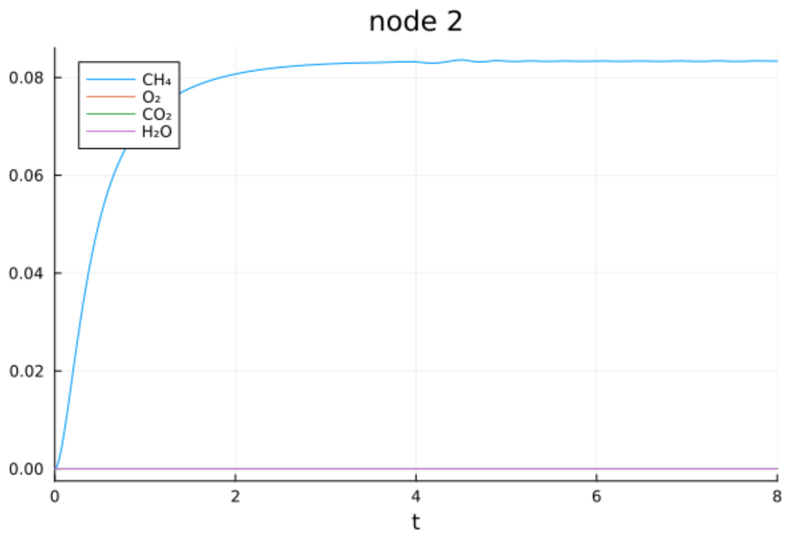


Figure 48: Graph 3 - Node 2

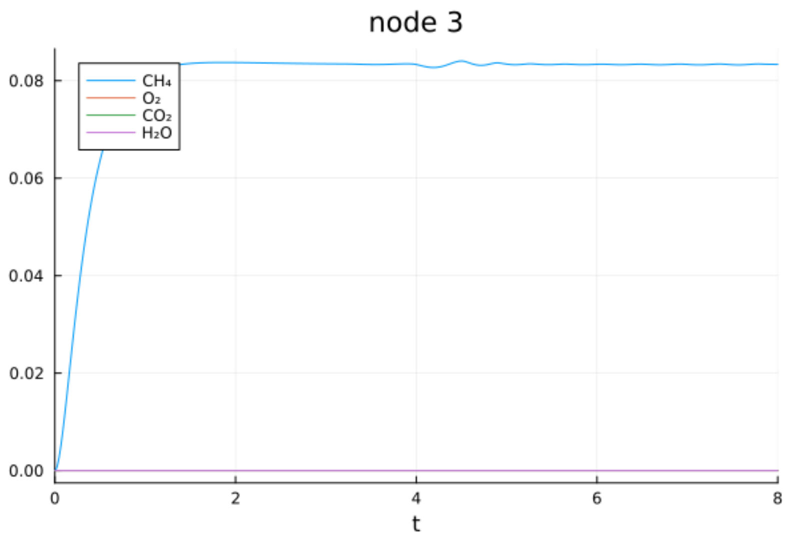


Figure 49: Graph 3 - Node 3

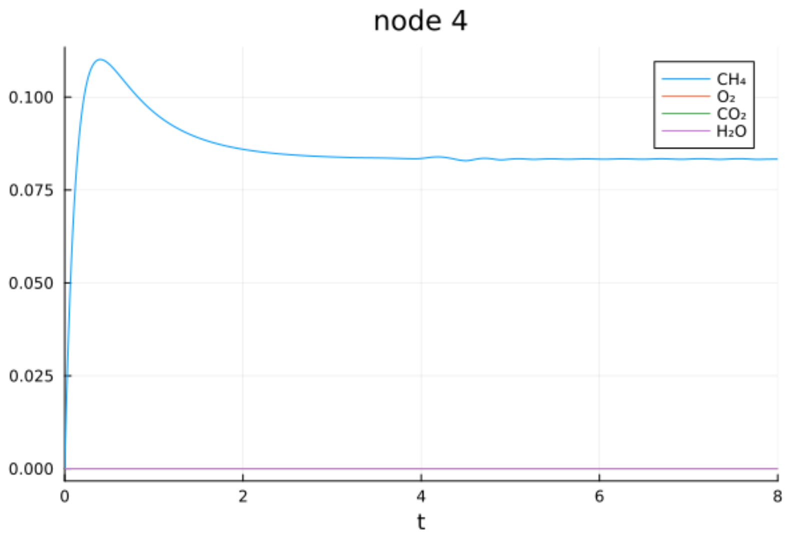


Figure 50: Graph 3 - Node 4

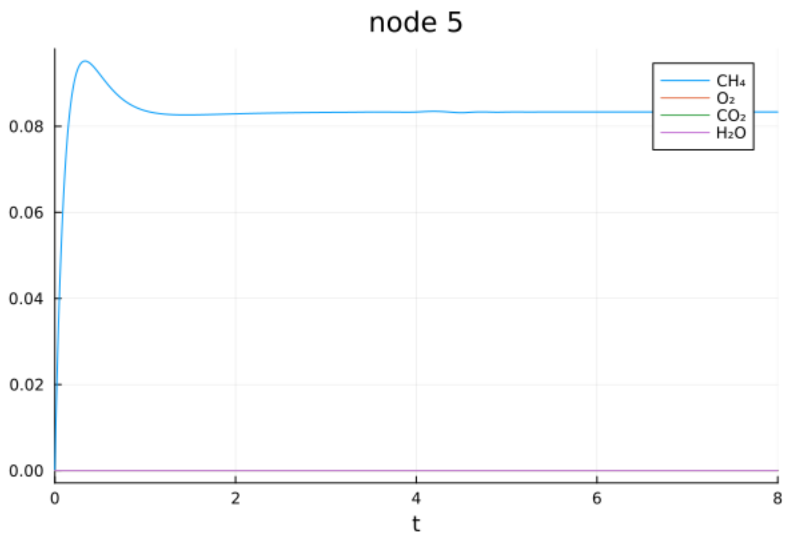


Figure 51: Graph 3 - Node 5

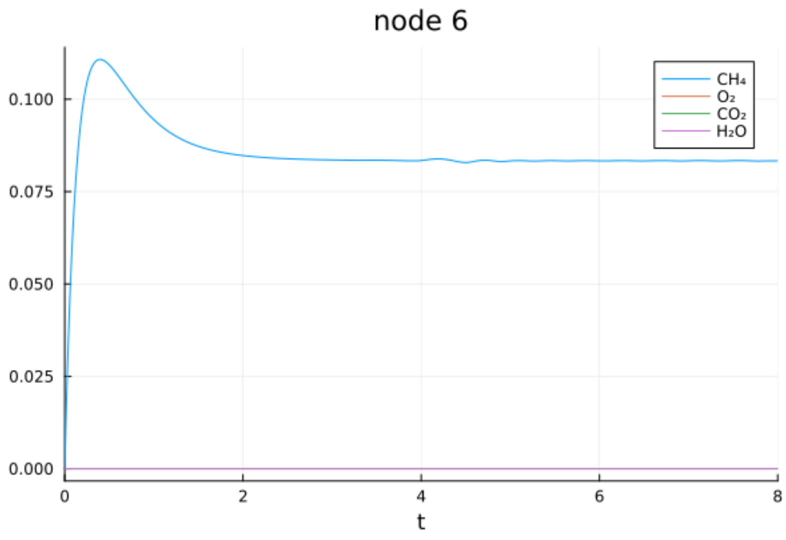


Figure 52: Graph 3 - Node 6

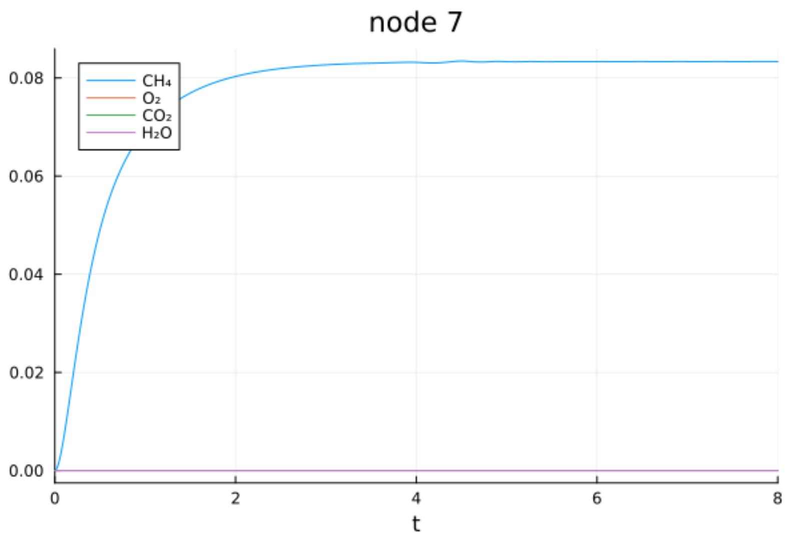


Figure 53: Graph 3 - Node 7

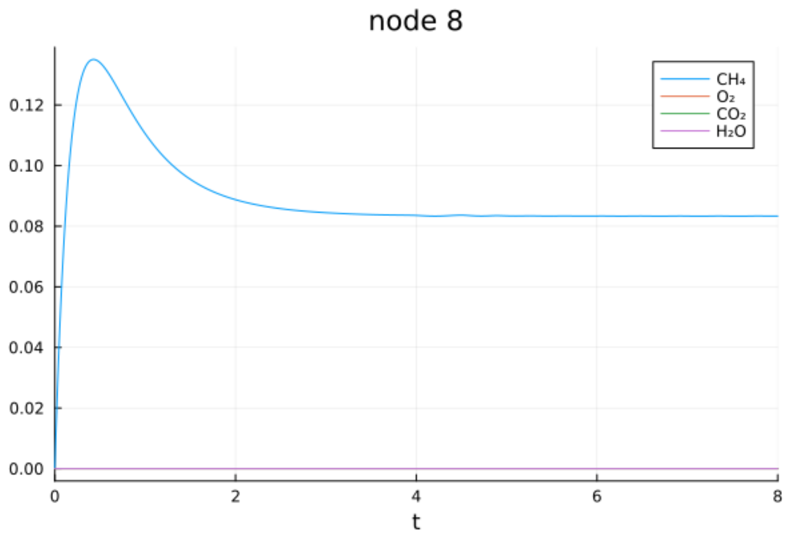


Figure 54: Graph 3 - Node 8

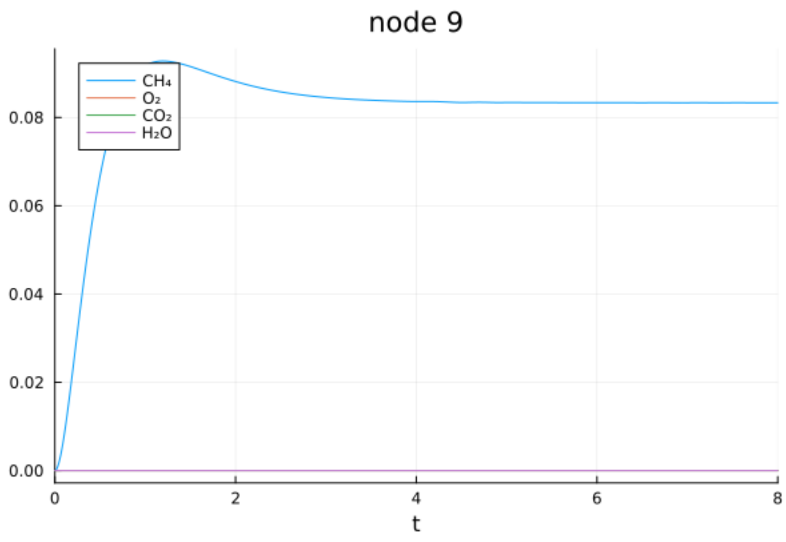


Figure 55: Graph 3 - Node 9

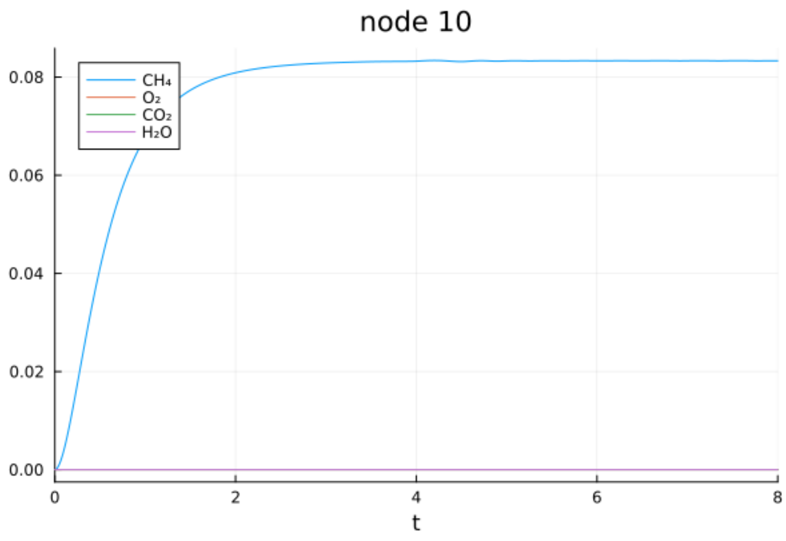


Figure 56: Graph 3 - Node 10

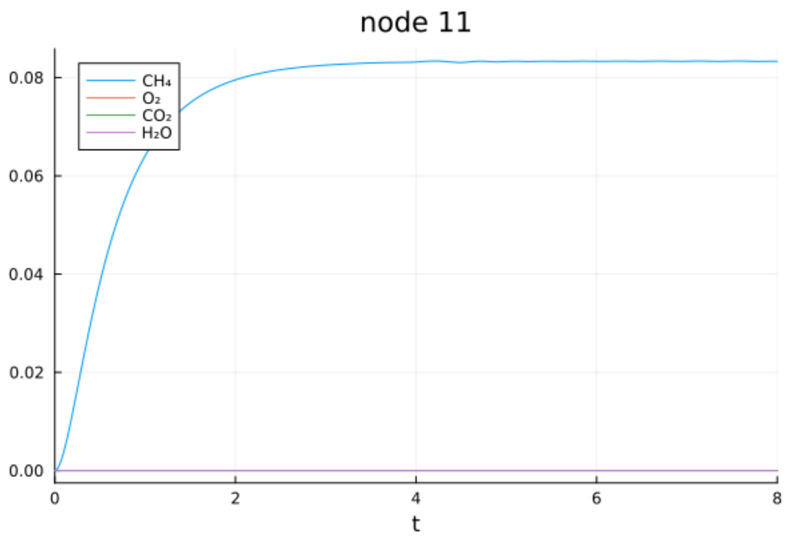


Figure 57: Graph 3 - Node 11

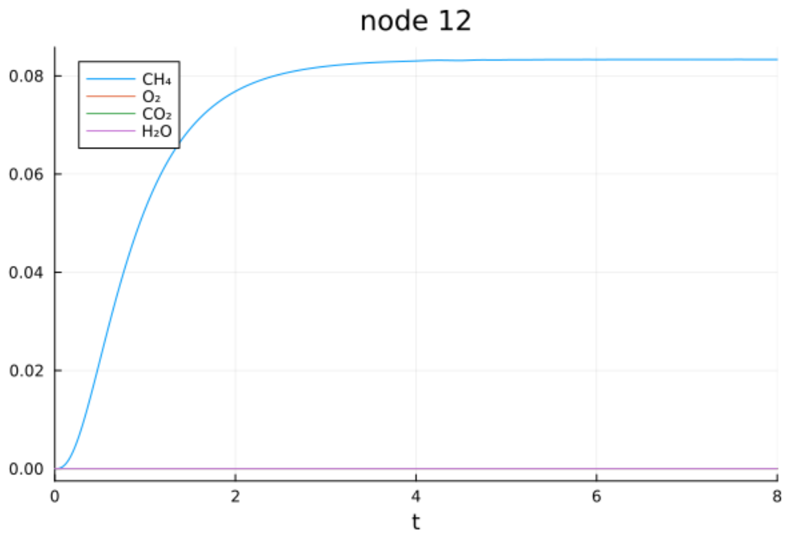


Figure 58: Graph 3 - Node 12

B.4 Graph 4

Generated with the barabassi-albert graph generator using an average degree of 3, and random seed 42. This time with a 1000 nodes.

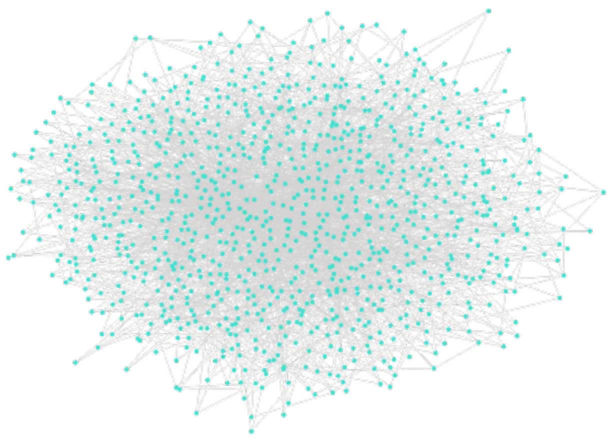


Figure 59: Graph 4 rendered using GraphPlot.jl

Solving the system linearly required 0.1750141 seconds.
Solving this using DifferentialEquations.jl required 1.1304552 seconds.

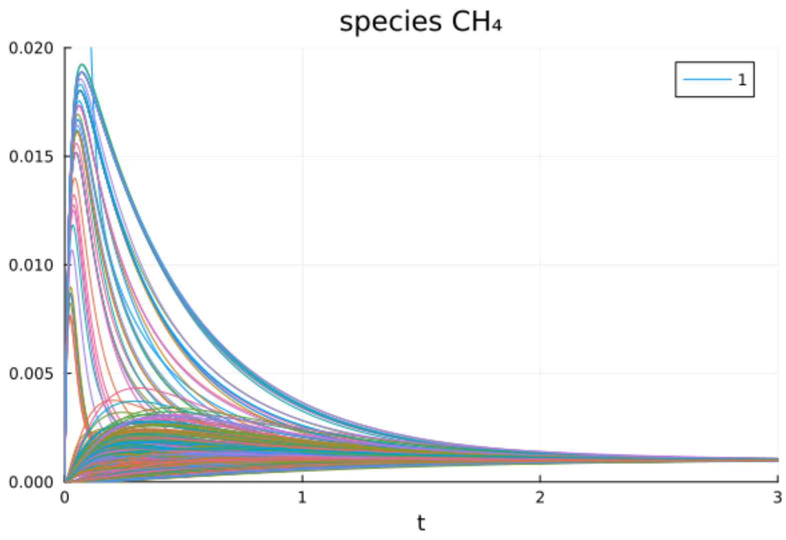


Figure 60: Concentrations of CH_4 on Graph 4

B.5 Graph 5

Again we use the barabassi-albert graph generator with an average degree of 3, and random seed 42. However this time we will use 10,000 nodes.

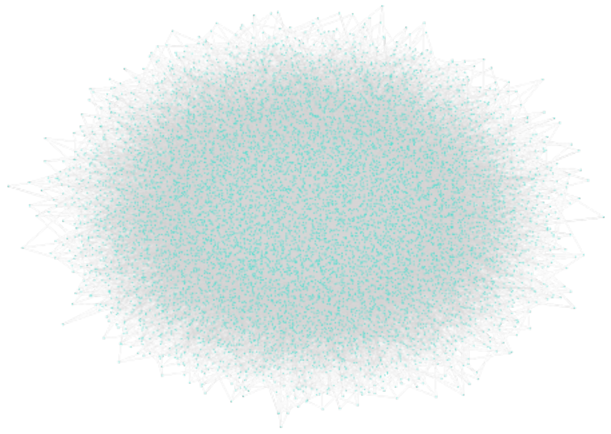


Figure 61: Graph 5 rendered using GraphPlot.jl

Solving the system linearly required 41.8191324 seconds. Solving this using DifferentialEquations.jl required 159.6922803 seconds.

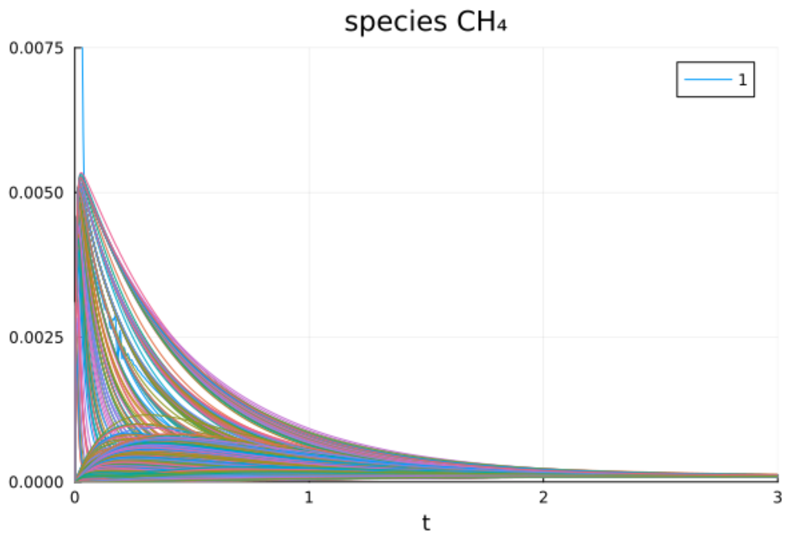


Figure 62: Concentrations of CH_4 on Graph 5

B.6 Graph 6

This time we use the barabassi-albert graph generator with, again, an average degree of 3 and random seed 42. But now with 100,000 nodes.

Solving this using DifferentialEquations.jl required 528.5795513 seconds.
Due to sheer size producing any plots was not possible.

C GRI, gathering data

The information required is available on the webpage [8]. Unfortunately this cannot be easily downloaded. The information appears to be loaded in after interacting with the webpage, so a simple http request will also prove unfruitful. However by reading the html code we discover that the reactions are available on their own individual web pages which have a url of the following format:
`http://combustion.berkeley.edu/gri-mech/data/reactions/i`
Where the i at the end denotes the number of the reaction, meaning that inserting $i = 1$ will provide the web page of the first reaction. This does not appear to be noted anywhere. The package HTTP.jl [16] will allow us to request the html of these webpages. We can then convert this to the String type in order to do string manipulation. If we do this for the first reaction and print the result it will look as follows.

```
...(irrelevant)...
------------------------------------------------------------------------
Temp      delta-S     delta-H      kf          kr         Keq
(K)    (cal/mol K)  (kcal/mol) ----(mol,cm3,s)-----   (cm3/mol)

------------------------------------------------------------------------


300       -28.0      -119.1    4.00E+14    3.57E-71    1.12E+85
500       -29.6      -119.7    2.40E+14    7.86E-37    3.05E+50
1000       -31.1      -120.8    1.20E+14    3.61E-11    3.32E+24
1500       -31.7      -121.5    8.00E+13    1.07E-02    7.49E+15
2000       -32.0      -122.0    6.00E+13    1.63E+02    3.67E+11
2500       -32.1      -122.4    4.80E+13    4.91E+04    9.79E+08
3000       -32.2      -122.7    4.00E+13    2.08E+06    1.92E+07
------------------------------------------------------------------------

...(irrelevant)...
```

It contains the desired information surrounded by some irrelevant information. Here kr denotes the backward (or "reverse") reaction rate.

However there are various exceptions for this format which makes the webscraping a bit more challenging.

For example, for some reactions the backward reaction rate column is not present, though one could use the other data to calculate it. Such as below for reaction 324. And as you can see there are also some minor issues with the html.

```
------------------------------------------------
Temp <font face="Symbol">D</font>S <font face="Symbol">D</font>H kf
(K)   (cal/mol K)  (kcal/mol)   (mol,cm3,s)
------------------------------------------------
298      31.3        -15.2       2.41E+13
300      31.3        -15.2       2.41E+13
400      31.7        -15.1       2.41E+13
500      31.8        -15.1       2.41E+13
1000      31.0        -15.7       2.41E+13
1500      30.3        -16.5       2.41E+13
2000      29.8        -17.4       2.41E+13
2500      29.3        -18.6       2.41E+13
3000      28.8        -19.9       2.41E+13
------------------------------------------------
```

But more than that, for some reactions the html is quite a mess. For example, for reaction 23:

```
---------------------------------------------------------------
&nbsp;&nbsp;&nbsp; Temp&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; delta-S&nbsp;&nbsp;&nbsp;&nbsp; delta-H&nbsp;&nbsp;&nbsp;...(etc)...
&nbsp;&nbsp;&nbsp;&nbsp; (K)&nbsp;&nbsp;&nbsp; (cal/mol K)&nbsp; (kcal/mol) ----(mol,cm3,s)-----&nbsp;&nbsp;&nbsp;
---------------------------------------------------------------

&nbsp;&nbsp;&nbsp;&nbsp; 300&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 7.2&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; ...(etc)...
&nbsp;&nbsp;&nbsp;&nbsp; 500&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 6.5&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; ...(etc)...
&nbsp;&nbsp;&nbsp; 1000&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 4.9&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;...(etc)...
&nbsp;&nbsp;&nbsp; 1500&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 3.8&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;...(etc)...
&nbsp;&nbsp;&nbsp; 2000&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 2.9&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;...(etc)...
&nbsp;&nbsp;&nbsp; 2500&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 2.2&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;...(etc)...
&nbsp;&nbsp;&nbsp; 3000&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 1.6&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;...(etc)...
---------------------------------------------------------------
```

As you can see collecting the information via web-scraping is not quite as simple as it would seem at first, due to the non-uniformity of the reaction web-pages. However given enough time it is surely possible.