Comparing Chemical Kinetic Model Reduction Techniques Using pyMARS

By
Phillip O Mestas III

A THESIS

submitted to
Oregon State University
University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented April 1st, 2020
Commencement June 2020
AN ABSTRACT OF THE THESIS OF

Phillip O Mestas III for the degree of Honors Baccalaureate of Science in Computer Science presented on April 1st, 2020. Title: Comparing Chemical Kinetic Model Reduction Techniques Using pyMARS

Abstract approved:

Kyle Niemeyer

Accurately modeling the chemistry of conventional and alternative liquid transportation fuels in combustion technology is vital to predicting important quantities such as the burning rate, heat release, and pollutant emissions. However, incorporating detailed chemical kinetic models into reactive-flow simulations poses a significant challenge due to the associated high computational expense resulting from chemical stiffness and model size. Methods for reducing the size and complexity of kinetic models are needed to incorporate them into detailed multidimensional simulations. While numerous methods have been (and continue to be) developed, few objective comparisons have been made between methods; furthermore, few software implementations of reduction methods are openly available. To address both issues, I have developed the open-source software pyMARS (Python-based Automatic Reduction Software), which implements established reduction methods including directed relation graph (DRG), DRG with error propagation (DRGEP), path flux analysis, and sensitivity analysis. I will introduce and describe the design of pyMARS, and use it
to compare reduction methods using a wide range of hydrocarbon kinetic models. By running the different reduction methods with various inputs, I hope to gain insight on which algorithms tend to perform better under certain conditions.

Key Words: Chemical kinetic models, Model reduction, Reaction kinetics

Corresponding e-mail address: mestasp@oregonstate.edu
Honors Baccalaureate of Science in Computer Science project of Phillip O Mestas III presented on April 1st, 2020

APPROVED:

Kyle Niemeyer, Mentor, representing School of Mechanical, Industrial, and Manufacturing Engineering

Yeongjin Jang, Committee Member, representing School of Electrical Engineering and Computer Science

Jennifer Parham-Mocello, Committee Member, representing School of Electrical Engineering and Computer Science

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

Phillip O Mestas III, Author
Acknowledgements

This material is based upon work supported by the National Science Foundation under grant ACI-1535065, and the URSA Engage program at Oregon State University.
Contents

1 Introduction .......................................... 4
   1.1 Literature review .................................. 4
   1.2 Objectives ......................................... 6

2 Methods .............................................. 8
   2.1 Chemical Kinetics .................................. 9
   2.2 DRG method ........................................ 10
   2.3 DRGEP method ...................................... 12
   2.4 PFA method ......................................... 14
   2.5 Sensitivity analysis ................................. 16
   2.6 Reduction comparison ............................... 17

3 Implementation ...................................... 18
   3.1 Simulation Modules ................................. 19
      3.1.1 simulation .................................... 19
      3.1.2 sampling ...................................... 20
   3.2 Utility Modules ..................................... 21
   3.3 Algorithm Modules ................................. 22
   3.4 Software Engineering Practices .................... 24

4 Results and discussion ............................. 26
5 Conclusions
1 Introduction

Studying combustion gives us a better understanding of complex systems such as forest fires. It also allows us to design more efficient technology such as engines. To study combustion computationally, we create models that describe chemical systems and use them to perform simulations. These models contain detailed information about all of the chemical species and reactions in a system. Performing high-fidelity reactive-flow simulations requires accurate, detailed chemical kinetic models. Detailed models that describe systems such as gasoline and jet fuel surrogates tend to be very large [1]. Due to the complexities of these systems, the size of a detailed model used for simulation can contain hundreds to thousands of independent species and thousands to tens of thousands of individual reactions. For example, Westbrook et al. [2] and Herbinet et al. [3] developed models for large $n$-alkanes and a biodiesel blend surrogate with over 2000 and 3000 species, respectively. However, the size and detail of these models come with a heavy computational cost. Performing multidimensional simulations becomes unreasonable with chemical kinetic models of these sizes; thus, methods of reducing cost are necessary [1, 4].

1.1 Literature review

One solution to this problem is reducing the size and complexity of the chemical kinetic model. Reduced mechanisms relieve the burden of computational costs while still maintaining accuracy—if developed correctly [5]. Graph theory-based skeletal
reduction algorithms attempt to remove reactions and species from detailed models by representing the relationships between species on a graph, and traversing the graph from given target species to determine species that can be removed. The directed relation graph (DRG) method is a popular example of this [6]. Although DRG has been effective in reducing the size of chemical kinetic models, it has flaws such as only considering first-generation relationships between species. Methods such as directed relation graph with error propagation (DRGEP) [7, 8] and path flux analysis (PFA) [9] attempt to improve upon the DRG algorithm by accounting for factors that it ignores. Global pathway selection is a more recent algorithm developed as the next generation of PFA [10].

Brute-force reduction methods can find unimportant species that are missed by these algorithms, but they require more error calculations, so they take significantly longer to run. As a result, these methods are typically applied on a model that has already been reduced by another reduction algorithm. Examples of this type of reduction method include sensitivity analysis [4, 11–13], and iterative screening and structure analysis (ISSA) [14]. Another common way of reducing model size is to group common species into one representative, removing the need to track all similar species. Pepiot-Desjardins and Pitsch developed an isomer lumping method and showed this approach to be effective [15]. The chemistry-guided reduction (CGR) method also uses chemical lumping to reduce the size of detailed models [16].

Although many effective reduction algorithms have been developed, few studies
have examined which of these methods is the most effective in practice. Niemeyer et al. [4] compared the results of the DRG, DRGEP, DRG-aided sensitivity analysis (DRGASA), and DRGEP-aided sensitivity analysis (DRGEPSA) on two different models and found that DRGEP and DRGEPSA performed better than DRG and DRGASA in both cases. When presenting the PFA method, Sun et al. [9] showed that PFA performed the same or better than DRG on two different models across a wide range of initial conditions. Tosatto et al. [17] looked at the impact of various methods for calculating connection weights, and their influence on the outcome of the reduction. In addition, they noted that the reduced models produced by DRG and DRGEP were very different in terms of the species that they selected for reduction, and that the on-the-fly version of DRG performed significantly better than the global version. Gao et al. [10] built on the PFA method to develop the global pathway selection algorithm, which they showed to outperform PFA and DRGEPSA, and showed similar results to DRGASA in the test case used. The past studies applied the methods of interest to one or two models, and generally did not perform their analysis over a wide range of models. Thus, one open question is: which common model reduction method is most effective across a broad range of models?

1.2 Objectives

In this study, I will compare popular kinetic model reduction methods using models of different sizes. Each reduction algorithm will be used on various models with the
same initial conditions and constraints, and I will quantify each method’s effectiveness using the number of species that can be removed without exceeding a specified error limit.

To perform these reductions, I will use pyMARS: Python-based Model Automated Reduction Software, an open-source software package I helped develop [18, 19]. pyMARS supports various model reduction techniques documented in literature by making use of the Cantera [20] library for handling chemical kinetics. pyMARS is written in Python and released openly under the MIT license. The source code can be found on GitHub: [github.com/Niemeyer-Research-Group/pyMARS](http://github.com/Niemeyer-Research-Group/pyMARS)

pyMARS currently supports four primary methods: DRG [6], DRGEP [7, 8], PFA [9], and sensitivity analysis (SA) [4, 11–13]. These techniques use ideas from graph theory to determine which species should be eliminated from the model. They represent the relationships between the species in the model as a directed graph, with the edge weights representing the dependency of each species on another. The DRG method retains species that remain connected to certain target species in the graph after trimming unimportant edges based on a small threshold value. PFA is similar to DRG, but also takes into account “second-generation” pathways when determining the strength of a relationship between species. The DRGEP method considers geometric propagation of dependence through indirect interactions, and quantifies a species’ overall importance to the target based on the product of edge weights over the graph path. For each graph-based method, pyMARS starts with a
low cutoff threshold value, produces a trial reduced model, and iterates over increasing
threshold values until reaching a user-specified maximum error. Error is (currently)
determined by comparing simulated autoignition delay times between the original and
reduced models under specified conditions.

pyMARS also supports the sensitivity analysis method [11], which removes species
individually based on the error induced by their removal. This method is computa-
tionally expensive when performed alone, but can instead be used to complement a
graph-based reduction method: the DRGASA and DRGEPSA methods [4, 12, 13].

2 Methods

pyMARS currently implements the previously developed DRG, DRGEP, and PFA
methods for model reduction [6, 7, 9]. In addition, sensitivity analysis [11] can be
used by itself or following other methods to remove additional species [4]. (DRG-aided
sensitivity analysis is referred to as the DRGASA method [12, 13], while combining
DRGEP with sensitivity analysis is the DRGEPSA method [4].)

The currently supported methods work iteratively: remove one or more species,
evaluate the error in autoignition delay with respect to the original mechanism to de-
termine if the candidate model performs within the specified error limit, then attempt
to remove the next set of species and repeat. Although DRG and DRGEP have many
similarities, the methods differ most by how they use the graph to determine which
species should be removed from the model. Notably, DRG uses a cutoff threshold to
trim unimportant graph edges directly connecting species (followed by a graph search
to identify the species reachable from a target), while DRGEP uses a cutoff threshold
to remove unimportant species based on their overall importance metric. PFA uses
the same graph search algorithm as DRG, but uses a different method for calculating
the weights of the edges on the graph.

2.1 Chemical Kinetics

The chemical kinetic models used to describe combustion systems define the rates of
change for the concentrations of all species present. Each reaction in the system has a
reaction rate that represents the rate at which its participating species are consumed
or produced. These reaction rates are defined as a function of the concentration of
the reactants in the reaction, as well as other factors including temperature. For
example, in the reaction $\text{CO}_2 + \text{C} \longrightarrow 2\text{CO}$, the participating species are $\text{CO}_2$, $\text{C}$,
and $\text{CO}$. The reaction rate depends on the product of the concentrations of $\text{CO}_2$ and
$\text{C}$, and this rate matches the rate at which $\text{CO}_2$ and $\text{C}$ are consumed, while $\text{CO}$ is
produced at twice this rate. All of the species production/consumption rates together
represent a system of ordinary differential equations that define the model’s behavior.

Simulations can be performed with the models by integrating these differential
equations in time to find the time-varying species concentrations. However, these
operations can be computationally expensive, particularly for larger kinetic models,
which can involve hundreds to thousands of species—meaning a system of ordinary
differential equations of the same size—interacting through thousands or tens of thousands of reactions. Simulations of practical combustion systems and fire become impractical for larger kinetic models, but we can achieve much more feasible simulation times by removing unimportant species and reactions from such large models.

2.2 DRG method

Once the program has started with the DRG method selected, autoignition simulation(s) are run to determine thermochemical state data used to calculate direct interaction coefficients. These coefficients represent the dependency of one species on another species. The direct interaction coefficient of species A on species B is defined by Lu and Law [6] as $r_{AB}$:

$$r_{AB} \equiv \frac{\sum_{i=1}^{I} |\nu_{A,i} \omega_i \delta_{B_i}|}{\sum_{i=1}^{I} |\nu_{A,i} \omega_i|}$$

where $I$ is the number of reactions in the model, $\nu_{A,i}$ is the stoichiometric coefficient of species A in the $i$th reaction, and $\omega_i$ is the overall reaction rate of the $i$th reaction. The direct interaction coefficients are used as the weights on a directed graph of the species in the system. One graph is created for each set of initial conditions presented using the tools available in the NetworkX library [21]. An edge is only included on
the graph if its weight is greater than a specified threshold value ($\epsilon$): $r_{AB} \geq \epsilon$.

Once the graph is created, a depth-first search is performed from each of the target species. If any species is not reached through the search, it is removed from the model for that threshold value, along with any reactions in which it participates. The program uses an iterative process to determine the maximum amount of species that can be removed before reaching the allowed error level (e.g., 10% in autoignition delay for the specified conditions). Iteration occurs by increasing the threshold value. The program starts with a threshold value of 0.01. This value is repeatedly divided by 10 until the error introduced is below the allowed limit. The error percentage is calculated based on autoignition delay by creating a reduced model and running the autoignition simulation with that model. The autoignition delay from that simulation is compared to the autoignition delay from the original model to determine the error percentage. When a starting threshold value has been determined, the threshold value will be slightly increased and the error will be calculated again. Once the error has gone beyond the allowed level, the iteration will stop. All species not connected to the target species on the final graph are removed from the model, and all reactions involving any of those species are removed as well.

Lu and Law [22] showed the DRG method to be efficient, especially on large models. Target species selection for the DRG method is important, but not vital because all of the species that should be maintained will be connected. Since the method uses a depth first search to determine what will be used in the reduced
model, all of the species that should be kept will be as long as one of the target species is in that group. For example, if one chooses the (one or more) fuel species, all species necessary for representing fuel oxidation should be reachable.

2.3 DRGEP method

pyMARS also supports the DRGEP method for model reduction. Although its implementation is similar to that of DRG, the two methods vary in the way the graph is created and searched. Instead of making a new graph to search for each threshold level, the DRGEP method requires only one graph per initial condition before iterating, and uses them to make a dictionary to determine what should be removed. Direct interaction coefficients are calculated similarly to DRG. Details of the calculations are given by Pepiot-Desjardins and Pitsch [7] and Niemeyer and Sung [8]:

\[
 r_{AB} = \frac{\sum_{i=1}^{I} \nu_{A,i} \omega_i \delta_{Bi}}{\max(P_A, C_A)}, \tag{3}
\]
where

\[
\delta_{B_i} = \begin{cases} 
1, & \text{if the } i\text{th elementary reaction involves species B,} \\
0, & \text{otherwise,} 
\end{cases}
\]  

\[
P_A = \sum_{i=1}^{I} \max(0, \nu_{A,i} \omega_i), \tag{5}
\]

\[
C_A = \sum_{i=1}^{I} \max(0, -\nu_{A,i} \omega_i). \tag{6}
\]

A graph is created for each initial condition using all of the direct interaction coefficients and species. The path coefficient of a path on the graph can be calculated by multiplying all of the weights of the edges on that path. This value will decrease or stay the same with each new edge added to the graph because all edge weights are equal to or less than 1. The overall interaction coefficient for each species is defined as the greatest path coefficient between a species and a target species. To determine the overall interaction coefficient (OIC) for each species, a search is preformed from each target species on each graph to determine the greatest possible path to each other species. This search is done using a modified version of Dijkstra’s algorithm\,[23] that finds the greatest possible path to each node when taking the products of the edge weights. Because all direct interaction coefficients are less than one, this algorithm can be used. Wagner\,[24] showed many of the optimizations that can be made to Dijkstra’s algorithm and Niemeyer and Sung\,[8] established that this implementation of Dijkstra’s algorithm is the best graph searching algorithm for DRGEP.
specifically. The NetworkX implementation of Dijkstra’s algorithm was modified to fit the algorithm needed for pyMARS. The greatest path to each species from any target species on any of the graphs is stored in a dictionary. These values represent each species OIC. Species with an OIC below the threshold value are removed from the model. The threshold value is iterated in the same way as in the DRG method. Each species’ OIC is dependent on what target species are used. Because of this, target species selection is vital for the DRGEP method; the distance a species has from the target species will impact its OIC.

2.4 PFA method

The final graph-based method currently implemented in pyMARS is PFA. Although the graph search and iteration phases are the same as DRG, it differs in the way the direct interaction coefficients are calculated. Unlike DRG, the PFA algorithm accounts for second-generation pathways between species when calculating their direct interaction coefficients. The values are calculated using the production and consumption fluxes for each species as shown in equations 5 and 6 respectively, and the production and consumption fluxes that are caused by species B for each species A as defined by
Sun et al. \cite{9} as follows:

\[
P_{AB} = \sum_{i=1}^{I} \max(0, \nu_{A,i}\omega_{i}\delta_{Bi}) \quad \text{and} \quad (7)
\]

\[
C_{AB} = \sum_{i=1}^{I} \max(0, -\nu_{A,i}\omega_{i}\delta_{Bi}) , \quad (8)
\]

where \(\delta_{Bi}\) is the same as used above. The first-generation interaction coefficients are calculated using these values:

\[
r^{\text{pro-1st}}_{AB} = \frac{P_{AB}}{\max(P_{A}, C_{A})} \quad \text{and} \quad (9)
\]

\[
r^{\text{con-1st}}_{AB} = \frac{C_{AB}}{\max(P_{A}, C_{A})} , \quad (10)
\]

where “pro” represents production and “con” represents consumption. Second-generation interaction coefficients are derived from the products of the first-generation interaction coefficients along second-generation pathways:

\[
r^{\text{pro-2nd}}_{AB} = \sum_{M_{i} \neq A,B} \left( r^{\text{pro-1st}}_{AM_{i}} r^{\text{pro-1st}}_{M_{i}B} \right) \quad \text{and} \quad (11)
\]

\[
r^{\text{con-2nd}}_{AB} = \sum_{M_{i} \neq A,B} \left( r^{\text{con-1st}}_{AM_{i}} r^{\text{con-1st}}_{M_{i}B} \right) . \quad (12)
\]
Finally, the first- and second-generation interaction coefficients are combined to determine the direct interaction coefficient:

\[ r_{AB} = r_{AB}^{\text{pro-1st}} + r_{AB}^{\text{con-1st}} + r_{AB}^{\text{pro-2nd}} + r_{AB}^{\text{con-2nd}}. \] (13)

These values are used in the same way as in DRG to determine which species are removed from the detailed model. A graph is built with edges between all species pairs that have a direct interaction coefficient greater than the threshold value: \( r_{AB} \geq \epsilon \).

Then, a depth-first search is performed from the target species, and all species not found by the search are removed from the model.

### 2.5 Sensitivity analysis

Sensitivity analysis can be performed after either the DRG or DRGEP method to further reduce the model. First, a set of species in limbo must be defined using a user-provided cutoff threshold value: \( \epsilon^* \). The previously used reduction algorithm must be performed with \( \epsilon^* \) as the threshold value. Species that remain after the first stage of reduction (e.g., DRG, DRGEP) but would be removed at this threshold constitute the “limbo” set. The limbo species are then removed from the reduced model one by one, and the error induced by their removal is calculated. The species B is then assigned an error measure:

\[ \delta_B = |\delta_{B,\text{ind}} - \delta_{\text{red}}|, \] (14)
where $\delta_{B,\text{ind}}$ is the error of the reduced model without species B and $\delta_{\text{red}}$ is the error between the reduced model from the first stage of reduction [4]. The species are then removed from this model one at a time in increasing order of these values. The final reduced model is the last model in this series that does not exceed the given error limit.

### 2.6 Reduction comparison

To compare the different reduction methods, I used models of different sizes ranging from 53 species to 545 species. The models used were for

- methane (CH$_4$) [25],
- a toluene reference fuel, consisting of $n$-heptane, isoctane, and toluene (nC$_7$H$_{16}$, iC$_8$H$_{18}$, C$_6$H$_5$CH$_3$) [26],
- methyl butanoate (C$_5$H$_{10}$O$_2$) [27],
- ethyl propanoate (C$_5$H$_{10}$O$_2$) [28],
- propanol (C$_3$OH) [29], and
- 2,5-dimethylfuran ((CH$_3$)$_2$C$_4$H$_2$O) [30].

The execution time of all algorithms was recorded during the experiment. I ran all simulations on the same desktop setup with multithreading disabled. The inputs used for reduction were consistent across all models and methods. The primary fuel
species in each model were used as the target species for all reductions on that model. DRGEP reductions also used O\textsubscript{2} as an additional target species. A maximum error limit of 5\% was set for all reductions. Non-greedy sensitivity analysis was used with an $\epsilon^*$ value of 0.1 for DRGEP\textsc{sa} reductions and 0.2 for DRG\textsc{asa} reductions. For the autoignition simulations, initial pressures range from 1 atm to 21 atm in increments of 10 atm, initial temperatures range from 800 K to 1600 K in increments of 200 K, and equivalence ratios range from 0.5 to 1.5 in increments of 0.5. All cases used fuel–air oxidation, with air represented as 3.76:1 mixture of N\textsubscript{2} and O\textsubscript{2} and N\textsubscript{2}. The 2,5-dimethylfuran model is an exception to this rule, as it is made for high pressure systems. Its initial pressures range from 20 atm to 60 atm in increments of 20 atm. The methane model did not use initial conditions with a temperature of 800 K, because it would not ignite for these conditions.

3 Implementation

\textsc{pyMARS} is implemented in Python and uses the Cantera library \cite{Cantera} for handling chemical kinetics and storing information about models. It also relies on the \textsc{NetworkX} library for handling the graphs used in the algorithms \cite{NetworkX}. Key components of the software package are separated into various modules. When using \textsc{pyMARS}, the user provides an input file via the command line with the -i option. Command-line arguments are parsed using \texttt{argparse}. Input files are stored in the YAML format and consist of the name of the model to be reduced, target species for the model,
species to be retained during the reduction, the algorithm to be run, the maximum error threshold, sensitivity analysis information, and autoignition simulation initial conditions. The YAML file is parsed into a dictionary, and the input is checked for errors in the `parse_inputs()` function. If all assertions are passed, the main function is called. In this function, the inputs are examined, and the main function of the module for the corresponding reduction algorithm is called. If the input file is correctly configured for a sensitivity analysis, it is performed after the initial reduction.

### 3.1 Simulation Modules

Simulations must be run using the original and reduced models to generate data needed to determine what information to remove from models and evaluate reduced model performance. These operations are performed by the sampling and simulation modules in pyMARS. Simulations are performed using the Cantera suite for chemical kinetics [20].

#### 3.1.1 simulation

The `simulation` module contains the simulation class which interfaces with Cantera to perform the needed operations. To instantiate the class, the caller must provide the model to be simulated, along with properties of the simulation such as the initial conditions and timestep information. Before running a simulation, the `setup_case()` function must be called to create the objects within the class from the properties
provided at initialization. The run_case() function can then be used to run the previously set up simulation. Detailed information from each timestep will be written to disk in the HDF5 file format. After the simulation has been completed, the process_results() function can be run to retrieve the information from this file. If the full detailed information is not needed, the calculate_ignition() function can be used to quickly determine just the autoignition delay. This function runs much faster because it does not write the detailed information to the disk.

3.1.2 sampling

While the simulation module uses the interface provided by Cantera to run simulations, the sampling module uses the simulation module to run multiple simulations for the same model in parallel. It contains the InputIgnition class, which holds input information for a single case of an autoignition simulation. The parse_ignition_inputs() function takes in the model and initial conditions in a dictionary, performs error checking on the inputs, and creates the input object if all assertions are passed.

Two of the most important functions in the module are the ignition_worker() and simulation_worker() functions. Each takes a simulation object as input, runs the simulation, and outputs the relevant data. The simulation_worker() function performs a detailed simulation, while the ignition_worker() function performs the quicker autoignition delay simulation without writing data. The sample() and sample_metrics() functions each take a list of input conditions, and create a list
of simulation objects out of them. They then create a pool of threads based on the
user supplied input for the number of threads, and map the corresponding worker
function over the list of simulations in parallel. The sample function uses the de-
tailed `simulation_worker()`, while the `sample_metrics()` function uses the faster
`ignition_worker()`. Both functions first check to see if the output of the simulation
is already stored on the disk, and use the previous output if so. In addition, the
module also holds the `calculate_error()` function which is used to compare the
outputs of simulations to calculate the error percentage introduced by the reduced
model. Other classes exist within the module so that it can be extended further to
support perfectly stirred reactor and laminar flame simulations as well as autoigni-
tion simulations, but they are not yet implemented. These simulations will be slower
than autoignition, but provide a more accurate picture of the error introduced by the
reduced model.

### 3.2 Utility Modules

In addition to the simulation and reduction related modules, pyMARS implements
various utility modules that are used by the reduction modules, and have other poten-
tial use cases as well. The `soln2cti` module takes a Cantera solution object as input,
and writes a corresponding CTI file that stores the module. Similarly, the `soln2ck`
module writes a Cantera solution object in the CHEMKIN format. Both modules
contain various helper functions that build specific sections of the corresponding out-
put file. The tools module contains a `convert()` function that can convert a file from the Cantera format to the CHEMKIN format by reading the Cantera file into a Cantera solution object, and using the `soln2ck` module to write the object into the CHEMIKIN file format. To convert from the CHEMKIN format to the Cantera format, the function calls Cantera’s built-in `ck2cti` tool. Additionally, the tools module contains a function called `compare_models()`, that performs a deep comparison of two Cantera solution objects and determines if the objects represent the same model.

The `reduce_model` module contains a function that will trim a list of species from a given model. A list of species that are in the model and not in the exclusion list will be generated. Also, a list of reactions is created that includes all reactions in the original model that do not include any species from the exclusion list. The reduced species and reaction lists are used to create a reduced Cantera solution object that represents the reduced model. This function is used frequently in all of the reduction algorithms.

### 3.3 Algorithm Modules

Each of the implemented algorithms has their own module that handles their functionality. All of the algorithms besides sensitivity analysis follow a similar pattern. The `run()` function is the main function of the algorithm that handles finding the optimal threshold value used in the reduction. First, it creates the Cantera solution object of the model, and uses the `sampling` module to sample key data from the
Then, it uses the `create_matrix()` function to create a matrix of direct interaction coefficients based on the corresponding algorithm. The program attempts to use a starting threshold value of 0.01 for the reduction, but decreases the threshold value if the error induced at this level is greater than the allowed level. Once a viable starting threshold value has been found, the program will perform a reduction at the starting threshold value and continue to increment the threshold value by the starting value until the error goes above the allowed limit. The last model that fell within the acceptable error range will be used as the final output of the reduction. Performing a reduction at a specific threshold value is done using the `reduce()` function. This function generates a list of species to be removed from the model, and uses the previously described modules to create a reduced version of the original model, run a simulation with it to record the autoignition delay, and calculate the error introduced by this reduced model. To determine what species should be removed from the model, the `trim()` function is used. For DRG and PFA, this function builds a graph with edges that fall above the threshold value, and does a depth-first search from the target species, returning all species that do not show up in this search as those that should be removed. For DRGEP, the `run()` function uses the direct interaction coefficients to build a dictionary of overall interaction coefficients. All species with overall interaction coefficients below the threshold value are chosen for removal. Since these algorithms are all very similar, they could have been implemented using the template design pattern. However, sensitivity analysis and most future algorithms will not
follow the same pattern, so this would have negative consequences when adding that functionality.

The `run()` function for sensitivity analysis is much different from the other algorithms. There is no threshold value to increment. Instead, a list of limbo species must be provided by the caller. This is a list of all species which are present in the reduced model produced by the previous algorithm, but would have been removed at a (given) higher threshold value (e.g., 0.1 or 0.2). That list is generated by the previous algorithms module. The error introduced by removing each limbo species is calculated using the `evaluate_species_errors()` function. While the error is below the allowed level, the species that introduced the lowest amount of error is committed as a removal. If the algorithm is set to “greedy” mode, all error values will be calculated using the `evaluate_species_errors()` function again before moving on to the next iteration. Once the error introduced crosses the allowed limit, the last reduced model that was within the allowed error limit will be used as the output. Similar to the current algorithms, each new method that is implemented in the future will get its own module.

### 3.4 Software Engineering Practices

pyMARS is an open-source software package that is available to the scientific community for use. We followed standard software engineering practices to improve our development process and the user experience. Using the pytest framework [31], we
created a testing module that contains both unit tests and integration tests. These tests are used with continuous integration to prevent changes that introduce bugs, saving future development time. In addition, we used Sphinx to automatically create a website documenting the code. The documentation helps contributors understand the code they are working with, and the examples and theory that we added help users get started with the program.

On top of improving the usability of pyMARS, we also optimized the operations it performs, making it produce reduced models much faster. Originally, we used standard Python arrays to do all calculations. This resulted in the reductions requiring significant runtimes. When running our comparison experiment, the DRG and DRGEP methods took multiple days to run on the bigger models, and the PFA method did not finish after two weeks of running. To fix this issue, we refactored the code to use NumPy arrays \[32\]. After these changes were made, all three algorithms finished in under a day on the largest model in our dataset. Additionally, the calculations for direct interaction coefficients are not described optimally in the papers that define them. Lu and Law showed that the DRG method can be implemented with a time complexity that is linear to the number of reactions in the detailed model \[22\]. This approach can be extended to DRGEP and PFA as well to reduce the time it takes to calculate the interaction coefficients. The ability to run autoignition cases in parallel was added to speed up reductions with a large set of initial conditions. These optimizations lead to a much faster program, which in turn provides our users with
4 Results and discussion

I used pyMARS to compare the effectiveness of the implemented model reduction techniques on models of varying sizes. The following shows and describes the results of the previously described experiment. Table 1 summarizes the results from applying each model to the various starting models. It compares the numbers of species in the starting models with the sizes of the reduced models produced by each reduction method, for the given error limit. The results from the DRG, DRGEP, and PFA methods should be compared, as these are comparable graph-based reduction methods, while the DRGASA and DRGEPSA methods should be directly compared since these both use sensitivity analysis. In all cases, DRGEP performed the same or better than DRG, and DRGEPSA performed the same or better than DRGASA. (Here, the goal of model reduction is to eliminate as many species as possible for a given error constraint.) In half of the cases, the difference was small or non-existent, and in the other half the difference was significant. In one case, DRGEP was able to remove as many as 38 more species than DRG. Generally, a big difference between DRG and DRGEP corresponds with a big difference between DRGASA and DRGEPSA. This likely occurred because DRGEP removed some groups of species that both DRG and DRGASA could not identify. For PFA, the results were inconsistent. In some cases, such as on the propanol model, it narrowly outperforms both...
DRG and DRGEP. However, it left significantly more species in the model than any of the other algorithms for the reductions performed on the 2,5-dimethylfuran model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Starting</th>
<th>DRG</th>
<th>DRGEP</th>
<th>PFA</th>
<th>DRGASA</th>
<th>DRGEPASA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRI Mech 3.0</td>
<td>53</td>
<td>47</td>
<td>35</td>
<td>44</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Toluene Reference Fuel</td>
<td>138</td>
<td>114</td>
<td>111</td>
<td>108</td>
<td>53</td>
<td>50</td>
</tr>
<tr>
<td>2,5-dimethylfuran</td>
<td>545</td>
<td>287</td>
<td>249</td>
<td>316</td>
<td>257</td>
<td>219</td>
</tr>
<tr>
<td>Ethyl propanoate</td>
<td>139</td>
<td>79</td>
<td>76</td>
<td>78</td>
<td>72</td>
<td>71</td>
</tr>
<tr>
<td>Propanol</td>
<td>237</td>
<td>138</td>
<td>138</td>
<td>135</td>
<td>106</td>
<td>103</td>
</tr>
<tr>
<td>Methyl butanoate</td>
<td>275</td>
<td>184</td>
<td>158</td>
<td>187</td>
<td>165</td>
<td>120</td>
</tr>
</tbody>
</table>

Table 1: Number of species in each starting model and remaining after applying the reduction methods, for a 5% error limit in ignition delay.

<table>
<thead>
<tr>
<th>Model</th>
<th>DRG</th>
<th>DRGEP</th>
<th>PFA</th>
<th>DRGASA</th>
<th>DRGEPASA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRI Mech 3.0</td>
<td>00:54</td>
<td>00:34</td>
<td>2:37</td>
<td>3:07</td>
<td>1:52</td>
</tr>
<tr>
<td>Toluene Reference Fuel</td>
<td>3:30</td>
<td>2:42</td>
<td>10:06</td>
<td>25:29</td>
<td>23:54</td>
</tr>
<tr>
<td>2,5-dimethylfuran</td>
<td>26:33</td>
<td>16:28</td>
<td>55:30</td>
<td>3:38:38</td>
<td>52:17</td>
</tr>
<tr>
<td>Ethyl propanoate</td>
<td>3:25</td>
<td>1:43</td>
<td>8:29</td>
<td>12:28</td>
<td>4:51</td>
</tr>
<tr>
<td>Propanol</td>
<td>5:40</td>
<td>5:29</td>
<td>14:06</td>
<td>48:39</td>
<td>32:47</td>
</tr>
<tr>
<td>Methyl butanoate</td>
<td>7:31</td>
<td>7:56</td>
<td>14:06</td>
<td>1:20:30</td>
<td>38:16</td>
</tr>
</tbody>
</table>

Table 2: Elapsed time (given in hours:minutes:seconds) while running each of the reduction methods on the same desktop configuration.

Table 2 shows the amount of time that each reduction took to run without running ignition cases for the detailed model. All reductions were performed on a desktop machine with the same configuration. Multithreading was disabled for all of these reductions. In all cases except for one, DRGEP executed faster than DRG. Both algorithms calculate the direct interaction coefficients in a time linear to the number of reactions in the original model. DRG uses depth-first search, so it is able to perform its traversal in a time that is linear to the number of species in the model plus the
number of reactions in the model. However, it must build and traverse the graph at each threshold value, while DRGEP only needs to build and traverse the graph once to create the dictionary of overall interaction coefficients. After that, DRGEP can determine the species to remove at each threshold in a time linear to the number of species in the model. The overhead of repeatedly building and searching the graph is likely what causes DRG to take longer. In the case where it was faster than DRGEP, it is possible that DRG found the optimal threshold value with far less attempts than DRGEP. PFA always finished slower than both DRGEP and DRG. This is to be expected, as PFA includes the same repeated graph building stage as DRG, and also calculates the direct interaction coefficients in a time proportional to the number of reactions in the model squared. This is the worst time complexity of the three algorithms. The increased time penalty of PFA does not seem to be made up for with smaller output models in most cases.

Comparing the models involving sensitivity analysis, DRGEP-SA outperformed DRGASA in all cases in terms of run time. This observation makes sense: DRGEP tends to produce reduced models that are smaller than those produced by DRG, so the sensitivity analysis stage likely has less species to check in limbo in DRGEP-SA than in DRGASA. In the case where DRGEP and DRG produced output models with the same size, DRGEP-SA still outperformed DRGASA in terms of time. This is likely because DRGEP-SA still identified less species in limbo in this case, despite both reduced models being the same size. The differences in timing seem to be more
pronounced in the larger models. All timing observations are consistent with the theoretical analysis of the algorithms.

Overall, the algorithms seem to be performing well by significantly reducing the amount of species in the model. For example, DRGEP was able to reduce the 2,5-dimethylfuran model from 545 species to 249 species, just 45 percent of its original size. In the cases where DRG and DRGEP were not as successful in reducing the model, adding in the sensitivity analysis stage made a huge difference. The toluene reference fuel model was reduced from 138 species to 114 species by DRG and 111 species by DRGEP, but DRGASA got it down to 53 species and DRGEPSA got it down to 50 species, all within the bounds of staying below a 5% error compared to the original model. In the case of the methane model (GRI Mech 3.0), DRG reduced it to 47 species and DRGEP reduced it to 35 species, but both DRGASA and DRGEPSA reduced it to 21 species. Cases like these likely occur because the algorithm identifies a key species for removal much earlier in the process than it should. Sensitivity analysis gets around this by skipping over that species.

5 Conclusions

I helped develop pyMARS, an open-source implementation of multiple chemical kinetic model reduction algorithms. Additionally, I used pyMARS to compare the effectiveness of the implemented reduction methods in practice. For the models and initial conditions used in this study, the DRGEP and DRGEPSA methods consistently
outperformed DRG and DRGASA, respectively. DRGEP is an improved version of DRG that includes enhancements to weight species with closer connections to the target species higher than those with less direct connections. This work suggests that the extensions made to DRG in DRGEP are leading to a significant improvement in performance. The results collected from the experiments are in line with our initial expectations. On the other hand, the results collected from the PFA reductions were inconsistent. When compared with DRGEP and DRG on the six models used, PFA was twice the best performer, twice the middle performer, and twice the worst performer. This suggests that PFA may be better suited for models with certain characteristics. More research is necessary to determine these characteristics. All of the timing results gathered from the experiment are in line with the expectations gathered from theoretical analysis of the algorithms.

Although the results align with our expectations, the current sample size is small. It has been challenging to find models without any formatting issues so that they can be used as input to pyMARS. In addition, memory constraints prevented me from running reductions on models in the thousands of species. I recommend testing reductions with different target species, error limits, and initial conditions to see if DRGEP continues to perform better in all situations. In the future, more methods will be added to pyMARS, making it easier to compare them to the other supported methods. The software package should also be improved by adding support for perfectly stirred reactor and laminar flame simulations to more accurately calculate induced
error. All of the currently supported algorithms are skeletal reduction methods. We plan to extend this to other categories such as isomer lumping reduction methods [33] or genetic algorithms [34]. This will make pyMARS a more robust and versatile software package, and allow researchers to collect more data and make stronger conclusions about the methods. Through the experiments I ran, I was able to get a direct comparison between the reduction algorithms implemented in pyMARS and provide insight regarding which algorithm will reduce models the furthest in practice. I also was able to help make pyMARS the first open source chemical kinetic model reduction package to be released. pyMARS supplies the community with open and freely available implementations of popular reduction methods, providing an easy way to improve the efficiency of chemical kinetic models and promoting transparency in future research.
References


