Computational tutorials with Cantera

1. INTRODUCTION

Cantera is an open-source suite of object-oriented software tools for problems involving chemical kinetics, thermodynamics, and transport processes. Cantera automates the chemical kinetic, thermodynamic, and transport calculations so that the users can efficiently incorporate detailed chemical thermo-kinetics and transport models into their calculations, such as time-dependent reactors, reactor networks and one-dimensional reacting flows. It also utilizes object-oriented concepts for robust yet flexible phase models, and algorithms are generalized so that users can explore different phase models with minimal changes to their overall code. It is currently used for applications including combustion, detonations, electrochemical energy conversion and storage, fuel cells, batteries, aqueous electrolyte solutions, plasmas, and thin film deposition. Cantera can be used from Python and Matlab, or in applications written in C/C++ and Fortran 90. The use of the Python interface is the most common among users as it offers most of the features of the core C++ code in a flexible environment with excellent documentation.

The main objective of this tutorial is to introduce you to the use of Cantera with the Python interface and explore some of the important lessons taught (or will be) in this summer school. It is highly recommended to visit the official Cantera website (https://cantera.org/) and explore the documentation, the tutorials, the user's group and for the more advanced ones, the source code.

At the time of this tutorial, the most recent stable version of Cantera is 2.4.0.

2. GET STARTED

Part 1. Anaconda platform

The most straightforward way to run Cantera with Python is through the Anaconda platform. Anaconda is an open-source platform to perform easy data-science operations using Python and R. With Anaconda it is possible to set up and link a whole set of libraries in a few minutes, which makes it ideal for installing Cantera and interfacing it with Python. In this tutorial, we are using the Windows operating system but the same guidelines apply for Linux and Mac OS X.

To get started:

- Go the search bar and type 'Anaconda Prompt'. If nothing appears, search for conda.exe and run this executable first.
- Wait for a black terminal to appear and then type the following command:

  ```
  conda install --channel cantera cantera ipython matplotlib scipy spyder
  ```

  Use the 'Enter' key or type 'y' where necessary to complete the installation. This will download and install Cantera and all the necessary libraries to the 'base' profile of Anaconda. In the future you might consider installing more libraries or create a new profile to install another Cantera version or combination of libraries.
- Go again to the search bar and now type 'Anaconda Navigator'. Wait for the software to start. The Navigator allows you to interactively switch between profiles and easily call Python interpreters or other applications.
- **Launch** the application 'Spyder'. You will see that the window is separated in three main parts: (i) a text editor where you can **drag & drop** a Python script, (ii) an IPython console that allows you to directly run Python code (similar to MATLAB/Octave) and (iii) an additional sub-window that allows you to see the files in the current directory or the loaded variables. Note the toolbar at the top of the window and the '**Run**' button (shortcut: F5). You can use this to run a script.

- To test if Cantera is successfully installed, type the following command on the IPython console:

  import cantera

  If no error messages appear, then everything is ready. You can now start running the scripts provided in this tutorial.

Part 2. Some basics

When opening the provided Python scripts, note the syntax and the commands structure. Groups of commands are aligned, and parent groups are separated from their children via space or tab (it doesn't matter how many spaces or tabs). Be mindful of extra spaces that may misalign commands of the same group.

Cantera is an object-oriented software, which means that it consists of objects that have attributes and methods associated with it (note the use of '.' to define property). To understand this, create a new script with Spyder (shortcut: Ctrl+N) and copy the following code which performs an equilibrium calculation:

```
# Import cantera package and name it ct
import cantera as ct
# Create an instance representing a gas-phase mixture.
# Use a pre-installed chemical kinetic mechanism and properties ('gri30.xml')
gas = ct.Solution('gri30.xml')
# Show all the available attributes and methods associated with the object 'gas'.
help(gas)
# Define the temperature (in K) and pressure (in Pa) of the mixture.
gas.TP = 300, 1e5
# Define the composition of this mixture (X for mole fractions and Y for mass fractions).
# Note that Cantera automatically rescales the composition.
gas.X = 'CH4:1.0, O2:1.0, N2:3.76'
# Calculate the equilibrium composition for constant enthalpy (H) and pressure (P) at adiabatic
conditions.
gas.equilibrate('HP')
# Show the new composition and its properties
gas()
```

As shown above, to perform combustion-related calculations with Cantera, we need a chemical kinetic mechanism together with thermodynamic and transport properties. Usually, these can be found in Chemkin format and should then be converted to a format suitable for Cantera (.cti or .xml). Check the documentation for conversion instructions. For this tutorial, these files are already provided in .cti format for convenience.

3. TUTORIALS

3.1 Chemical kinetics calculations

Part 1. Ignition delay time

Objectives
To use the Python and Cantera libraries to predict ignition delay times (IDTs) for a constant volume reactor, for a given fuel-air mixture, using the provided chemical kinetic mechanism (in Cantera format) over a range of initial temperature, pressure and composition conditions. IDTs for two fuels, n-heptane and n-butanol, will be compared over a range of temperatures in order to explore the presence of a Negative Temperature Coefficient (NTC) regime in each fuel.

Description
Simulations will be conducted for the fuels, n-heptane and n-butanol, in air over the temperature range 750-1100 K, at 20 bar initial pressure, under stoichiometric conditions. Ignition delay times for each fuel-air mixture will then be plotted with respect to 1000/T (as is common practice). This data is also saved to .txt files.

Method
To run the constant volume auto-ignition simulation, open the script 'const_vol_ignition.py' in a Python interpreter, such as 'Spyder' or 'IDLE'. From the interpreter toolbar select 'Run' (shortcut: F5). The code should complete the ignition delay time simulations for n-heptane and n-butanol in approximately 5 minutes. Results and graphs will appear in the Python console.

The 'README' file contains further information about how the script is operating.

The script is commented so that you may open it in an editor if you wish to see how the code is written and try changing basic parameters such as equivalence ratio etc.

To investigate the influence of the stoichiometry of the fuel-air mixtures, simply change the equivalence ratio value on lines 38 and 44, for the n-heptane and n-butanol simulations, respectively.

Questions
1. Is there a difference between the fuels in terms of the presence of an NTC?
2. Does either of the fuels express an Arrhenius type behaviour?
3. What happens to the IDT profiles if the overall stoichiometry is changed, e.g. to leaner or richer conditions?
4. Can you link back to the lectures from this morning and explain the kinetic reasons for the differences between the fuels?

Part 2. Sensitivity analysis

Objectives
To carry out a local sensitivity analysis for the stoichiometric n-heptane in air case at the point of maximum 'OH' concentration at various temperatures to explore the chemistry within the different regions of the ignition delay diagram.

Description
Local 'OH' sensitivity analysis is performed for constant volume simulations, at the point of maximum 'OH' concentration, to provide an insight into the chemistry driving the ignition in each case. Peak OH is chosen since this has been shown to fairly closely match the outputs when a full Brute Force Analysis for predicted IDT is performed (see lectures). Sensitivities are normalised against the largest sensitivity value and the top 10 positive and negative (20 total) normalised 'OH' sensitivity values are plotted for each case and saved to output files (.csv format) with the respective reaction equations. The sensitivities have different signs depending on whether they are promoting or inhibiting reactivity.

The current version of the provided script will produce a local 'OH' sensitivity analysis of n-heptane in air at temperatures of 800, 950 and 1100 K, at 20 bar initial pressure, under stoichiometric conditions. This will highlight the changing chemistry of auto-ignition in the low-temperature, NTC and high-temperature regimes.

Method
To run the sensitivity analysis code, open the file 'sensitivity_analysis.py' in a Python interpreter, such as 'Spyder' or 'IDLE'. From the interpreter toolbar select 'Run' (shortcut: F5). This code will require a longer processing time than in 'Part 1', approximately 5-10 minutes per temperature condition. You may start to consider the questions below as you wait for the next temperature to complete. Results and graphs will appear in the Python console.

The Readme file contains further information about how the script is operating.

The script is commented so that you may open it in an editor if you wish to see how the code is written and try changing basic parameters such as equivalence ratio etc.

Questions

1. How do the main sensitive reactions change as you move from low to high-temperature conditions?
2. Do you see the presence of the typical low-temperature branching routes in the high-ranking sensitivities as discussed in the lectures? What are the key reactions that promote and inhibit reactivity?
3. At high temperature, do you still see an influence of primary fuel reactions or does the small molecule chemistry dominate?
4. In which region is the chemistry of HO_2 most important?

3.2 One-dimensional flame calculations: Freely-propagating premixed flat flames

Objectives

To use the Python and Cantera libraries to predict the laminar burning velocity of a given fuel-air mixture at ambient conditions and study the flame structure of laminar premixed flames. For this purpose, freely-propagating premixed flat flames are simulated using the provided chemical kinetic mechanism (in Cantera format) over a range of equivalence ratios. The laminar burning velocity of two fuels, methane and hydrogen, will be compared.

Description

Simulations will be conducted for the fuels, methane and hydrogen, in air over a pre-specified equivalence ratio range, at 1 bar pressure and 298 K initial temperature. Laminar burning velocities and maximum flame temperatures will then be plotted with respect to equivalence ratio (as is common practice). This data is also saved to .txt files. Flame structure results are saved to .mat files for further post-processing with MATLAB.

Method

To run the methane flame simulations, open the script 'freely_propagating_flame.py' in a Python interpreter, such as 'Spyder' or 'IDLE'. From the interpreter toolbar select 'Run' (shortcut: F5). The code should complete the laminar burning velocity simulations in approximately 5-10 minutes (depending on the fuel). Results and graphs of immediate interest will appear in the Python console. Results of flame structure will be saved in .mat files in the 'results' folder. MATLAB scripts are provided to post-process these files.

The 'README' file contains further information about how these scripts are operating.

The Python script is commented so that you may open it in an editor if you wish to see how the code is written and try changing basic parameters such as pressure etc.

To change the fuel, simply change the variable 'fuelname' on line 10 to 'H2'. The same chemical kinetic mechanism may be used. To investigate the influence of the pressure, simply change the pressure value on line 32.

Questions

1. Can you explain the location of the maximum laminar burning velocity in the case of methane?
2. Using the MATLAB post-processing scripts, show the flame structure and check if the radical species OH, O, CH or CH_2O good markers of the heat release rate? Try the same by multiplying the OH and CH_2O profiles. Do you think that this is useful from an experimental point of view?
3. Calculate the laminar burning velocity of hydrogen with respect to equivalence ratio. Modify the Python script for methane accordingly. Note that a different range of equivalence ratio might be required. Explain why we observe differences compared to methane.
4. Modify the provided Python script for sensitivity analysis with pressure at constant equivalence ratio. Can you find a relation of pressure to the laminar burning velocity of methane at stoichiometry in the 1-100 bar range? Hint: $S_L=a*p^b$