PyTeCK: a Python-based automatic testing package for chemical kinetic models

Kyle E. Niemeyer‡*

https://youtu.be/Ke3Ip25C4pY

Abstract—Combustion simulations require detailed chemical kinetic models to predict fuel oxidation, heat release, and pollutant emissions. These models are typically validated using qualitative rather than quantitative comparisons with limited sets of experimental data. This work introduces PyTeCK, an open-source Python-based package for automatic testing of chemical kinetic models. Given a model of interest, PyTeCK automatically parses experimental datasets encoded in a YAML format, validates the self-consistency of each dataset, and performs simulations for each experimental data point. It then reports a quantitative metric of the model's performance, based on the discrepancy between experimental and simulated values and weighted by experimental variance. The initial version of PyTeCK supports shock tube and rapid compression machine experiments that measure autoignition delay.

Index Terms—combustion, chemical kinetics, model validation

Introduction

Combustion simulations require chemical kinetic models to predict fuel oxidation, heat release, and pollutant emissions. These models are typically validated using qualitative, rather than quantitative, comparisons with limited sets of experimental data. Furthermore, while a plethora of published data exist for quantities of interest such as autoignition delay and laminar flame speed, most are not available in a standardized machine-readable format. Such data is commonly offered in poorly documented, nonstandard CSV files and Excel spreadsheets, or even contained in PDF tables or figures.

This work aims to support quantitative validation of kinetic models by:

1. Encouraging the use of a human- and machine-readable format to encode experimental data for combustion.
2. Offering an efficient, automated software package, PyTeCK, that quantitatively evaluates the performance of chemical kinetics models based on available experimental data.

Fundamental combustion experiments typically study the behavior of fuels in idealized configurations at conditions relevant to applications in transporation, aerospace, or power generation. These produce useful data for validating chemical kinetic models, which in turn can support simulations of more complex applications such as internal combustion or gas turbine engines. The autoignition delay of a fuel/oxidizer mixture represents the time required for the mixture to ignite (i.e., experience a rapid increase in temperature and corresponding consumption of fuel and oxidizer) after arriving at a specified initial state. Autoignition occurs in practical applications such as knock in spark-ignition engines or ignition in compression-ignition and gas turbine engines, and so ignition delay measurements provide useful validation measures for models aimed at capturing such phenomena. Other combustion experimental measurements—such as extinction in perfectly stirred reactors, species profiles in jet-stirred reactors, and laminar flame speeds—also provide useful information about fuel combustion characteristics, but these are not considered in this paper.

Ignition delay times are typically measured with two categories of experiments: shock tubes and rapid compression machines. In shock tubes, a diaphragm separates high-pressure gases from a lower-pressure mixture of fuel and oxidizer. Rupturing the diaphragm propagates a (compressive) shock wave into the fuel/oxidizer mixture, quickly increasing the temperature and pressure and leading to autoignition after a time delay. Chaos and Dryer [Chaos2010], and more recently Hanson and Davidson [Hanson2014], discuss shock tubes in more detail. In contrast, rapid compression machines, reviewed by Sung and Curran [Sung2014], emulate a single compression stroke in an internal combustion engine; the compression of a piston raises the temperature and pressure of a fuel/oxidizer mixture in a short period of time, after which ignition occurs. Shock tubes and rapid compression machines offer complementary approaches to measuring ignition delay times. Shock tubes can investigate a wide range of temperatures (600–2500 K) [Hanson2014], although problems with pre-ignition pressure rise occur at higher pressures and temperatures below around 1100 K [Petersen2009], [Chaos2010], while rapid compression machines can reach low-to-intermediate temperatures (600–1100 K) [Sung2014].

In this paper, I propose a data format for capturing results from experimental measurements of autoignition delay times. This paper also describes the components of PyTeCK (**Py**thon-based **te**sting of **c**hemical **k**inetic models), a software package that quantifies the performance of a chemical kinetic model in reproducing experimental ignition delays. This includes discussion of the experimental data parser, simulation framework, and solution post-processing. The paper also explains the theoretical basis for

* *Corresponding author: Kyle.Niemeyer@oregonstate.edu*
‡ *School of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University*

the models of shock tubes and rapid compression machines.

Implementation of PyTeCK

PyTeCK is packaged as a standard Python package using setuptools, and consists of three primary modules:

1. `parse_files` contains functions to read the YAML-encoded experimental data file using the `PyYAML` module. Smaller functions comprise this process to enable easier unit testing.
2. `simulation` contains the `Simulation` class and relevant functions for initiating, setting up, and running cases, and then processing the results.
3. `eval_model` uses the previous two modules to set up simulations based on experimental data, and then runs simulations in parallel using the `multiprocessing` module.

The next three sections explain the implementation of each primary module. PyTeCK also includes the module `detect_peaks`, based on the work of Duarte [Duarte2015], for detecting peaks in targeted quantities (e.g., pressure, temperature) to determine the ignition delay time. Supporting modules in PyTeCK include `exceptions` for raising exceptions while reading YAML files, `utils` that initializes a single Pint-based unit registry [Grecco2016], and `validation` that provides quantity validation functions.

PyTeCK relies on well-established scientific Python software tools. These include NumPy [vanderWalt2011] for large array manipulation, SciPy [Jones2001] for interpolation, Pint [Grecco2016] for interpreting and converting between units, PyTables [Alted2002] for HDF5 file manipulation, Cantera [Goodwin2016] for chemical kinetics, and `pytest` [Krekel2016] for unit testing. Travis-CI [Travis2016] also provides continuous integration testing.

PyTeCK is available under an open-source MIT license via a GitHub repository [Niemeyer2016b]. It can be installed using setuptools by downloading the source code files and executing `python setup.py install`. More mature versions of PyTeCK will be distributed on PyPI (Python Package Index).

Parsing ChemKED files

The PyTeCK module `parse_files` parses experimental data encoded in the ChemKED (**chem**ical **k**inetics **e**xperimental **d**ata) format proposed by this paper. ChemKED builds on XML-based `ReSpecTh` of Varga et al. [Varga2015a], [Varga2015b]—which in turn builds on the PrIMe data format [Frenklach2007], [You2012], [PrIMe2016]—but is written in YAML instead of XML. While XML is a powerful markup language, YAML offers a number of advantages: parsers and libraries exist for most programming langauges, it supports multiple data types and arrays. YAML files are also intended for data and more readable by humans, which allows easier composition and could encourage adoption.

The code block below shows a complete example of an autoignition dataset for an hydrogen/oxygen/argon (`H2/O2/Ar`) mixture, taken from Figure 12 (right) of Chaumeix et al. [Chaumeix2007]:

```yaml
---
file-author:
    name: Kyle E Niemeyer
    ORCID: 0000-0003-4425-7097
file-version: (1, 0)
reference:
    doi: 10.1016/j.ijhydene.2007.04.008
    authors:
        - name: N. Chaumeix
          ORCID:
        - name: S. Pichon
          ORCID:
        - name: F. Lafosse
          ORCID:
        - name: C.-E. Paillard
          ORCID:
    journal: International Journal of Hydrogen Energy
    year: 2007
    volume: 32
    pages: 2216-2226
    detail: Fig. 12., right, open diamond
experiment-type: Ignition delay
apparatus:
    kind: shock tube
    institution: CNRS-ICARE
    facility: stainless steel shock tube
common-properties:
    pressure: &pres
        value: 220
        units: kilopascal
    composition: &comp
        - species: H2
          InChI: 1S/H2/h1H
          mole-fraction: 0.00444
        - species: O2
          InChI: 1S/O2/c1-2
          mole-fraction: 0.00566
        - species: Ar
          InChI: 1S/Ar
          mole-fraction: 0.9899
    ignition-type: &ign
        target: pressure
        type: d/dt max
datapoints:
    - temperature:
        value: 1164.48
        units: kelvin
      ignition-delay:
          value: 471.54
          units: us
      pressure: *pres
      composition: *comp
      ignition-type: *ign
    - temperature:
        value: 1164.97
        units: kelvin
      ignition-delay:
          value: 448.03
          units: us
      pressure: *pres
      composition: *comp
      ignition-type: *ign
    - temperature:
        value: 1264.2
        units: kelvin
      ignition-delay:
          value: 291.57
          units: us
      pressure: *pres
      composition: *comp
      ignition-type: *ign
    - temperature:
        value: 1332.57
        units: kelvin
      ignition-delay:
          value: 205.93
          units: us
      pressure: *pres
      composition: *comp
      ignition-type: *ign
    - temperature:
        value: 1519.18
```

```
      units: kelvin
    ignition-delay:
        value: 88.11
        units: us
    pressure: *pres
    composition: *comp
    ignition-type: *ign
```

This example contains all the information needed to evaluate the performance of a chemical kinetic model with five data points. The file also includes metadata about the file itself, as well as reference information. While these elements, including `file-author`, `file-version`, and the entries in `reference`, are not required by PyTeCK, a valid ChemKED file should include this information for completeness. The elements necessary for PyTeCK include the type of experiment given by `experiment-type` (currently limited to `Ignition delay`), the `kind` of apparatus used to measure ignition delay (`shock tube` or `rapid compression machine`), and then a list of experimental `datapoints` given as associative arrays with necessary information. Mandatory elements of each entry in"datapoints" include the initial `temperature`, `pressure`, and mixture `composition`, as well as the experimental `ignition-delay` and `ignition-type` (means by which PyTeCk detects ignition, discussed in more detail later). All quantities provided include a magnitude and units, which Pint [Grecco2016] interprets. Since many experimental datasets hold certain properties constant (e.g., composition, pressure) while varying a single quantity (e.g., temperature), a `common-properties` element can describe properties common to all `datapoints`, using an arbitrary anchor label (e.g., `&pres` above for the constant pressure). Each data point then refers to the common property with a reference (`*pres`). However, every data point should still contain the complete information needed to reproduce its conditions; the `common-properties` element is used for convenience.

Modeling ignition in shock tubes or RCMs may require more elements to capture effects not accounted for by the simplest models. Under certain conditions that lead to longer ignition delay times, shock tubes can exhibit pressure rise before ignition. This is typically expressed in the literature with a constant pressure rise rate at a fraction of the initial pressure (with units of inverse time), and ChemKED files encode this as an item in the associative array describing an experimental data point:

```
pressure-rise:
    value: 0.10
    units: 1/ms
```

Later versions of PyTeCK will support specifying a pressure-time history directly, although these are not commonly published in the shock tube literature.

Simulations of RCM experiments commonly provide a volume-time history to capture nonideal pre- and post-ignition heat losses, as well as effects due to the compression stroke. This data can be provided with experimental `datapoints` in ChemKED as a list of lists, with the `column` index and units identified:

```
volume-history:
    time:
        units: s
        column: 0
    volume:
        units: cm3
        column: 1
    values:
        - [0.00E+000, 5.47669375000E+002]
        - [1.00E-003, 5.46608789894E+002]
```

The PyTeCK `tests` directory [Niemeyer2016b] contains more examples of ChemKED files for shock tube and RCM experiments.

The function `parse_files.read_experiment()` takes a ChemKED-format file as input, and returns a dictionary with the necessary information to perform simulations of the experimental data points. The `parse_files.get_experiment_kind()` and `parse_files.get_datapoints()` functions perform important checking of input information for consistency and validity of quantities via the `validation` module. For example, after detecting the specified initial temperature, `get_datapoints()` checks the correct dimensionality of units and range of magnitude (in this case, that the units are consistent with Kelvin and that the magnitude is greater than zero),

```
validation.validate_gt('temperature',
                       case['temperature'],
                       0. * units.kelvin
                       )
```

where the `validation.validate_gt()` function—borrowed heavily from Huff and Wang's PyRK [Huff2015], [Huff2015b]—is

```
def validate_gt(value_name, value, low_lim):
"""Raise error if value not greater than lower
limit or wrong type.

Parameters
----------
value_name : str
    Name of value being tested
value : int, float, numpy.ndarray, pint.Quantity
    Value to be tested
low_lim : type(value)
    ``value`` must be greater than this limit

Returns
-------
value : type(value)
    The original value

"""
try:
    if not validate_num(value_name, value) > low_lim:
        msg = (value_name + ' must be greater than ' +
               str(low_lim) + '.\n'
               'Value provided was: ' + str(value)
               )
        # RuntimeError used to avoid being caught by
        # Pint comparison error. Pint should really
        # raise TypeError (or something) rather than
        # ValueError.
        raise RuntimeError(msg)
    else:
        return value
except ValueError:
    if isinstance(value, units.Quantity):
        msg = ('\n' + value_name +
               ' given with units, when variable '
               'should be dimensionless.'
               )
        raise pint.DimensionalityError(value.units,
                                       None,
                                       extra_msg=msg
                                       )
    else:
        msg = ('\n' + value_name +
               ' not given in units. Correct '
```

```
                'units share dimensionality with: ' +
                str(low_lim.units)
                )
            raise pint.DimensionalityError(None,
                                           low_lim.units,
                                           extra_msg=msg
                                           )
    except pint.DimensionalityError:
        msg = ('\n' + value_name +
               ' given in incompatible units. Correct '
               'units share dimensionality with: ' +
               str(low_lim.units)
               )
        raise pint.DimensionalityError(value.units,
                                       low_lim.units,
                                       extra_msg=msg
                                       )
    except:
        raise
```

The `read_experiment()` function also checks that necessary parameters are present, and also for consistency between input parameters based on the particular experiment type being modeled. For example, an input ChemKED file describing a shock tube experiment cannot include `compression-time` or `volume-history` elements.

After parsing and checking the simulation parameters, the `parse_files.create_simulations()` function creates a list of `Simulation` objects.

Autoignition simulation procedure

Once `parse_files.create_simulations()` initializes a list of `Simulation` objects, the member function `setup_case()` prepares each object to perform a simulation by initiating the governing equations that model shock tubes and rapid compression machines. These equations are briefly described next.

A composition state vector Φ defines the thermochemical state of a general chemical kinetic system:

$$\Phi = \left\{T, Y_1, Y_2, \ldots, Y_{N_{\text{sp}}}\right\} ,$$

where T is the temperature, Y_i is the mass fraction of the ith species, and N_{sp} is the number of species represented by the chemical kinetic model. A system of ordinary differential equations advances this thermochemical state when modeling both experimental types, derived from conservation of mass and energy:

$$\frac{d\Phi}{dt} = \left\{\frac{dT}{dt}, \frac{dY_1}{dt}, \frac{dY_2}{dt}, \ldots, \frac{dY_{N_{\text{sp}}}}{dt}\right\} . \tag{1}$$

The derivative terms in Equation (1) come from the conservation of energy

$$\frac{dT}{dt} = \frac{-1}{c_v}\left(\sum_{i=1}^{N_{\text{sp}}} e_i \frac{dY_i}{dt} + p\frac{dv}{dt}\right) \tag{2}$$

and conservation of mass

$$\frac{dY_i}{dt} = \frac{1}{\rho} W_i \dot{\omega}_i \quad i = 1, \ldots, N_{\text{sp}} , \tag{3}$$

where c_v is the mass-averaged constant-volume specific heat of the mixture, e_i is the internal energy of the jth species in mass units, v is the specific volume of the mixture, and $\dot{\omega}_i$ is the overall molar production rate of the ith species.

PyTeCK relies on Cantera [Goodwin2016] for handling most chemical kinetics calculations. Cantera is an open-source software library that provides tools for solving problems related to chemical kinetics, thermodynamics, and transport processes. The core of Cantera is written in C++, but it provides interfaces for Python and Matlab. PyTeCK uses a Cantera [Goodwin2016] `ReactorNet` object to solve the system given by Equation (1), by connecting `IdealGasReactor` and `Reservoir` objects separated by a `Wall`. The `Wall` may or may not be moving, depending on whether the modeled system has varying or constant volume, respectively.

The simplest way to model both shock tubes and RCM experiments is by assuming an adiabatic, constant-volume process. In this case, I simplify Equation (2) by assuming $\frac{dv}{dt} = 0$, and the `Wall` is initialized with `velocity=0`:

```
self.wall = ct.Wall(self.reac, env, A=1.0, velocity=0)
```

This approach does not account for either preignition pressure rise observed in some shock tube experiments [Chaos2010], [Hanson2014] or heat loss in RCMs [Sung2014]. RCM volume histories are typically provided directly, but publications describing shock tube experiments with observed preignition pressure rise usually instead give a constant pressure-rise rate $\frac{dP}{dt}$. This is incorporated into Equation (2) by determining an associated preignition pressure history $p(t)$:

$$p(t) = p_0 + \int_0^{t_{\text{end}}} \frac{dp}{dt} dt , \tag{4}$$

where p_0 is the initial pressure and t_{end} the time interval of interest (typically the ignition delay time). The function `simulation.sample_rising_pressure()` actually constructs this pressure history, which is then used to construct a volume history $v(t)$ assuming isentropic compression:

$$v(t) = v_0 \left.\frac{\rho_0}{\rho(t)}\right|_{s_0} , \tag{5}$$

where v_0 is the initial volume, ρ is the density, ρ_0 is the initial density, and s_0 is the specific entropy of the initial mixture.

The varying volume of the system is handled by assigning the `velocity` attribute of the `ReactorNet`'s '`Wall` to one of two classes: `VolumeProfile` when volume history is provided

```
self.wall = ct.Wall(
    self.reac, env, A=1.0,
    velocity=VolumeProfile(self.properties)
    )
```

and `PressureRiseProfile` when pressure-rise value is specified

```
self.wall = ct.Wall(
    self.reac, env, A=1.0,
    velocity=PressureRiseProfile(
        mechanism_filename, initial_temp,
        initial_pres, reactants,
        self.properties['pressure-rise'].magnitude,
        self.time_end
        )
    )
```

PyTeCK needs more details about the chemical kinetic model and initial conditions to initialize the `PressureRiseProfile` object, and specifically to construct the discrete volume-time history via Equations (4) and (5) using the `simulation.create_volume_history()` function. Objects of both classes contain the derivative of volume dv/dt, which PyTeCK obtains by numerically differentiating the volume history via `simulation.first_derivative()`. This

function uses `numpy.gradient()` to calculate second-order central differences at interior points and second-order one-sided differences (either forward or backward) at the edge points. When called, the `VolumeProfile` or `PressureRiseProfile` object returns the derivative of volume at the specified time (i.e., the velocity of the `Wall`), using `numpy.interp()` to interpolate as needed.

After each `setup_case()` prepares a `Simulation` object, the `run_case()` member function actually runs each simulation. PyTeCK prepares and runs each simulation independently to allow the use of `multiprocessing` workers to perform these steps in parallel (if desired), as described in the next section. When running a simulation, PyTeCK creates an HDF5 file and opens it as a PyTables [Alted2002] table, then performs integration steps until it reaches the desired end time (set as 100 times the experimental ignition delay). At every timestep, `run_case()` saves the time and information about the current thermochemical state (temperature, pressure, volume, and species mass fractions) to the HDF5 table. The Cantera `ReactorNet.step()` function performs a single integration step, selecting an appropriate time-step size based on estimated integration error. Internally, `step()` uses the CVODE implicit integrator [Cohen1996], part of the SUNDIALS suite [Hindmarsh2005], to advance the state of the `IdealGasReactor` contained by the `ReactorNet`.

Finally, a call to the `process_results()` member function determines the autoignition delay by opening the saved simulation results. The method by which it detects ignition depends on the target and type specified in the input ChemKED file. Target quantities include pressure, temperature, and mass fractions of commonly used species such as the OH and CH radicals (as well as their excited equivalents OH^* and CH^*). `process_results()` detects ignition by finding the location of either the maximum value of the target quantity (e.g., `type: max`) or the maximum value of the derivative of the quantity (e.g., `type: d/dt max`):

```
# Analysis for ignition depends on type specified
if self.ignition_type == 'd/dt max':
# Evaluate derivative
target = first_derivative(time, target)

# Get indices of peaks
ind = detect_peaks(target)

# Fall back on derivative if max value doesn't work.
    if len(ind) == 0 and self.ignition_type == 'max':
        target = first_derivative(time, target)
        ind = detect_peaks(target)

    # Get index of largest peak
    # (overall ignition delay)
    max_ind = ind[np.argmax(target[ind])]

    # add units to time
    time *= units.second

    # Will need to subtract compression time for RCM
    time_comp = 0.0
    if 'compression-time' in self.properties:
        time_comp = self.properties['compression-time']

    ign_delays = time[
        ind[np.where((time[ind[ind <= max_ind]] -
        time_comp) > 0)]
        ] - time_comp

    # Overall ignition delay

    if len(ign_delays) > 0:
        ign_delay = ign_delays[-1]
    else:
        ign_delay = 0.0 * units.second
    self.properties[
        'simulated ignition delay'
        ] = ign_delay
```

using the `detect_peaks.detect_peaks()` function [Duarte2015].

Evaluation of model performance

The approach used by PyTeCK to report performance of a chemical kinetic model is adapted from the work of Olm et al. [Olm2014], [Olm2015], and briefly discussed by Niemeyer [Niemeyer2016].

The function `eval_model.evaluate_model()` controls the overall evaluation procedure, given the required and optional parameters:

- `model_name`: a string with the name of the Cantera-format chemical kinetic model file (e.g., CTI file)
- `spec_keys_file`: a string with the name of a YAML file identifying important species
- `dataset_file`: a string with the name of a file listing the ChemKED files to be used, which gives the filenames in a newline delimited list
- `model_path`: a string with the directory containing `model_name`. This is optional; the default is `'models'`
- `results_path`: a string with the directory for placing results files. This is optional; the default is `'results'`
- `model_variant_file`: a string with the name of a YAML file identifying ranges of conditions for variants of the kinetic model. This is optional; the default is `None`
- `num_threads`: an integer with the number of CPU threads to use to perform simulations in parallel. This is optional; the default is the maximum number of available threads minus one

A few of these parameters require greater explanation. The chemical kinetic model, also referred to as "chemical reaction mechanism", needs to be provided in Cantera's CTI file (CanTera Input file) format [Goodwin2016]. This file contains a description of the elements, species (including names, molecular composition, and thermodynamic property data), and reactions (including reversibility, stoichiometry, Arrhenius rate parameters, third-body species efficiencies, and pressure dependence). Although the use of the CTI format in the literature has increased recently, often models are instead available in the older Chemkin format [Kee1996]. Such files can be converted using the Cantera-provided utility `ck2cti`.

PyTeCK needs the species key YAML file `spec_keys_file` because different chemical kinetic models internally use different names for species. PyTeCK interprets these names to set the initial mixture composition, and potentially identify a species target to detect ignition. This file contains entries (for multiple model files, if desired) of the form:

```
---
model_name:
    H2: "H2"
    O2: "O2"
    Ar: "AR"
```

where the key indicates the internal PyTeCK species name and the value is the name used by the model. In this case, the

necessary species names are consistent with the names used internally by PyTeCK, other than the capitalization of argon (`AR`). Names will likely differ for other kinetic models; for example, internally `nC7H16` represents the species *n*-heptane, while other models may use `C7H16`, `C7H16-1`, or `NXC7H16`, for example. PyTeCK's internal naming convention for key species is given by the `SPEC_KEY` and `SPEC_KEY_REV` dictionaries in the `utils` module, and can be obtained by calling `utils.print_species_names()`. For correct results the species name keys given in the `spec_keys_file` file only need to match names of species in the ChemKED files.

The `model_variant_file` YAML file is needed in certain (uncommon) cases where the chemical kinetic model needs manual changes to apply to different ranges of conditions (such as pressure or bath gas). In other words, different versions of the CTI file need to be created for accurate performance under different conditions. This file may contain entries of the form:

```
---
model_name:
    bath gases:
        N2: "_N2"
        Ar: "_Ar"
    pressures:
        1: "_1atm.cti"
        9: "_9atm.cti"
        15: "_15atm.cti"
        50: "_50atm.cti"
        100: "_100atm.cti"
```

where the keys are extensions added to `model_name`, in order of `bath gases` and then `pressures`, and the values represent the extensions to the base filename given by `model_name`. For models that need such variants, all combinations need to be present in the `model_path` directory. As an example, the kinetic model of Haas et al. [Haas2009] for mixtures of *n*-heptane, isooctane, and toluene, which I term `Princeton-2009`, has certain reactions that require rate parameters to be changed manually for different bath gases and pressure ranges. For a case with nitrogen as the bath gas and at pressures around 9 atm, the resulting file name would be `Princeton-2009_N2_9atm.cti`.

To determine the performance of a given model, `evaluate_model()` parses the ChemKED file(s), then sets up and runs simulations as described. A `multiprocessing.Pool` can perform simulations in parallel if multiple CPU threads are available, creating `simulation_worker` objects for each case. Then, `process_results()` calculates the simulated ignition delays.

PyTeCK reports the overall performance of a model by the average error function over all the experimental datasets:

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i \tag{6}$$

where N is the number of datasets and E_i is the error function for a particular dataset. A lower E value indicates that the model better matches the experimental data. The error function for a dataset E_i is the average squared difference of the ignition delay times divided by the variance of the experimental data:

$$E_i = \frac{1}{N_i}\sum_{j=1}^{N_i}\left(\frac{\log\tau_{ij}^{\text{exp}} - \log\tau_{ij}^{\text{sim}}}{\sigma(\log\tau_{ij}^{\text{exp}})}\right)^2, \tag{7}$$

where N_i is the number of data points in dataset i, τ_{ij} is the jth ignition delay value in the ith dataset, σ is the experimental variance, log indicates the natural logarithm (rather than base-10), and the superscripts "exp" and "sim" represent experimental and simulated results, respectively.

The experimental variance σ serves as a weighting factor for datasets based on the estimated uncertainty of results. This term reduces the contribution to E of a dataset with high variance, from discrepancies between model predictions and experimental data, compared to datasets with lower variance. Ideally, publications describing experimental results would provide uncertainty values for ignition delay results, but these are difficult to estimate for shock tube and rapid compression machines and thus not commonly reported. Thus, for now, PyTeCK estimates all variance values.

PyTeCK estimates the variance with the `eval_model.estimate_std_dev()` function, by first fitting a `scipy.interpolate.UnivariateSpline()` of order three (or less, if the fit fails) to the natural logarithm of ignition delay values for a given dataset (where results mainly vary with a single variable, such as temperature), and then calculating the standard deviation of the differences between the fit and experimental data via `numpy.std()`. PyTeCK sets 0.1 as a lower bound for the uncertainty in ignition delay time, based on the precedent set by Olm et al. [Olm2014], [Olm2015].

After calculating the error associated with a dataset using Equation (7) and the overall error metric for a model using Equation (6), `evaluate_model()` saves the performance results to a YAML file and returns the associated dictionary if `evaluate_model()` was called programmatically. If the `--print` command line option was given, or the `print_results` option set to `True` when calling `evaluate_model()`, then the results are also printed to screen.

Example Usage

This section provides an example of using PyTeCK to compare the performance of 12 chemical kinetic models for hydrogen oxidation [Niemeyer2016c] using a collection of experimental shock tube ignition delay data [Niemeyer2016d]. 54 data sets from 14 publications comprise this collection, with a total of 786 ignition data points. Both the set of models and ChemKED experimental data set are available openly via the respective references.

After installing PyTeCK [Niemeyer2016b], and placing the model and experimental data files in appropriate locations (`h2-models` and `h2-files`, in this example), each model can be evaluated by executing a command similar to `PyTeCK -m GRI30-1999.cti -k h2-model-species-keys.yaml -d h2-data-list.txt -dp h2-files -mp h2-models`, with the appropriate model name inserted in place of `GRI30-1999.cti`.

Figure 1 compares the performances of the 12 hydrogen models, showing both the average error function E as well as the standard deviation of E_i values across data sets. Lower error function values indicate better agreement with experimental data. While the actual values are not important for the current example, generally both the average and variation of error function decrease with publication year of the models---indicating an overall improvment of model fidelity with time. Although this example only considers subsets of both the models and experimental data of Olm et al.'s study [Olm2014], the results generally agree.

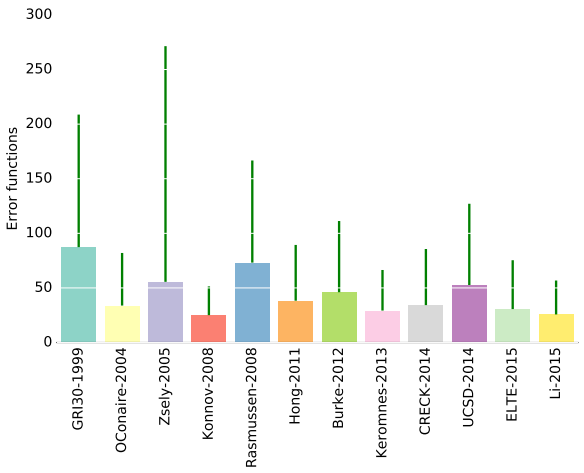

Fig. 1: *Average error functions, with standard deviations, for the 12 models of hydrogen oxidation. Models are arranged in order of publication, going from the oldest to the newest.*

Conclusions and Future Work

PyTeCK provides an open, Python-based framework for rigorously quantifying the performance of chemical kinetic models using experimental autoignition data generated from shock tube and rapid compression machine experiments. It can be used to compare models for describing the combustion of a given fuel and identify areas for improvement. Along with the software framework, this paper describes a new YAML-based data standard, ChemKED, that encodes experimental results in a human- and machine-readable manner.

Immediate plans for PyTeCK include better documentation generated by Sphinx [Brandl2016] and hosted on Read The Docs. Longer term plans for PyTeCK include extending support for other experimental types, including laminar flames and flow reactors, building in visualization of results, and creating an open database of ChemKED files for experimental data.

Acknowledgments

I thank Bryan Weber of the University of Connecticut for helpful discussions on the ChemKED format and an early review of this paper. I also thank Matt McCormick, Erik Tollerud, and Katy Huff for their helpful review comments.

Appendix

The following code snippet can be used to reproduce Fig. 1 using the produced by PyTeCK following the instructions given in the Example Usage section.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
from matplotlib.backends.backend_pdf import PdfPages
import brewer2mpl
import yaml

names = ['GRI30-1999', 'OConaire-2004', 'Zsely-2005',
         'Konnov-2008', 'Rasmussen-2008', 'Hong-2011',
         'Burke-2012', 'Keromnes-2013', 'CRECK-2014',
         'UCSD-2014', 'ELTE-2015', 'Li-2015'
         ]

ind = np.arange(len(names))

error_funcs = []
error_stds = []
for name in names:
with open(name + '-results.yaml', 'r') as f:
    results = yaml.load(f)
    error_func = results['average error function']
    std_dev = results['error function '
                      'standard deviation']
    error_funcs.append(error_func)
    error_stds.append(std_dev)

# colors for boxes
box_colors = brewer2mpl.get_map('Set3',
                                'qualitative',
                                len(names)
                                ).mpl_colors

fig, ax = plt.subplots()
yerr = [np.zeros(len(names)), error_stds]
ax.bar(ind, error_funcs, align='center',
       color=box_colors, linewidth=0,
       yerr=yerr, error_kw=dict(ecolor='g',
       lw=2, capsize=0)
       )

fmt = ScalarFormatter(useOffset=False)
ax.xaxis.set_major_formatter(fmt)

ax.set_ylabel('Error functions')
ax.set_xticks(ind)
ax.set_xticklabels(names, rotation='vertical')
ax.set_xlim([-0.5, ind[-1] + 0.5])
plt.subplots_adjust(bottom=0.25)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.yaxis.set_ticks_position('none')
ax.xaxis.set_ticks_position('none')
ax.grid(axis = 'y', color ='white', linestyle='-')

plt.show()
```

References

[Alted2002] F. Alted, I. Vilata, and others. "PyTables: Hierarchical Datasets in Python," 2002–. http://www.pytables.org/

[Brandl2016] G. Brandl and others. "Sphinx: Python documentation generator," version 1.4.2, 2016. http://sphinx-doc.org/

[Chaos2010] M. Chaos, F. L. Dryer. "Chemical-kinetic modeling of ignition delay: Considerations in interpreting shock tube data," *Int. J. Chem. Kinet.*, 42:143–50, 2010. https://dx.doi.org/10.1002/kin.20471

[Chaumeix2007] N. Chaumeix, S. Pichon, F. Lafosse, and C.-E. Paillard. "Role of chemical kinetics on the detonation properties of hydrogen/natural gas/air mixtures," *Int. J. Hydrogen Energy*, 32:2216–2226, 2007. https://dx.doi.org/10.1016/j.ijhydene.2007.04.008

[Cohen1996] S. D. Cohen and A. C. Hindmarsh. "CVODE, A Stiff/Nonstiff ODE Solver in C," *Comput. Phys.*, 10:138–143, 1996. http://dx.doi.org/10.1063/1.4822377

[Duarte2015] M. Duarte. "Notes on Scientific Computing for Biomechanics and Motor Control," GitHub repository, 2015. https://GitHub.com/demotu/BMC

[Frenklach2007] M. Frenklach. "Transforming data into knowledge—Process Informatics for combustion chemistry," *Proc. Combust. Inst.*, 31:125–140, 2007. https://dx.doi.org/10.1016/j.proci.2006.08.121

[Goodwin2016] D. G. Goodwin, H. K. Moffat, and R L. Speth. "Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes," Version 2.2.1, 2016. http://www.cantera.org

[Grecco2016] H. E. Grecco. Pint version 0.7.2, GitHub repository, 2016. https://GitHub.com/hgrecco/pint

[Haas2009] F. M. Haas, M. Chaos, F. L. Dryer. "Low and intermediate temperature oxidation of ethanol and ethanol–PRF blends: An experimental and modeling study," *Combust. Flame*, 156:2346–2350, 2009. http://dx.doi.org/10.1016/j.combustflame.2009.08.012

[Hanson2014] R. K. Hanson, D. F. Davidson. "Recent advances in laser absorption and shock tube methods for studies of combustion chemistry," *Prog. Energy. Comb. Sci.*, 44:103–14, 2014. http://dx.doi.org/10.1016/j.pecs.2014.05.001

[Hindmarsh2005] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," *ACM Trans. Math. Software.*, 31:363–396, 2005. http://dx.doi.org/10.1145/1089014.1089020

[Huff2015] K. Huff and X. Wang. PyRK v0.2, Figshare, Feb 2015. http://dx.doi.org/10.6084/m9.figshare.2009058

[Huff2015b] K. Huff. "PyRK: A Python Package For Nuclear Reactor Kinetics," *Proceedings of the 14th Python in Science Conference*, 87–93, 2015. Editors: K. Huff and J. Bergstra.

[Jones2001] E. Jones, T. Oliphant, P. Peterson, et al. "SciPy: Open source scientific tools for Python," 2001–. http://www.scipy.org/

[Kee1996] R. J. Kee, F. M. Rupley, E. Meeks, and J. A. Miller. "CHEMKIN-III: A FORTRAN chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics," Sandia National Laboratories Report SAND96-8216, May 1996. http://dx.doi.org/10.2172/481621

[Krekel2016] H. Krekel. pytest version 2.9.1, GitHub repository, 2016. https://github.com/pytest-dev/pytest/

[Niemeyer2016] K. E. Niemeyer. "An autoignition performance comparison of chemical kinetics models for *n*-heptane," Spring 2016 Meeting of the Western States Section of the Combustion Institute, Seattle, WA, USA. 21–22 March 2016. https://dx.doi.org/10.6084/m9.figshare.3120724

[Niemeyer2016b] K. E. Niemeyer. PyTeCK version 0.1.0, Zenodo, 2016. https://dx.doi.org/10.5281/zenodo.57565

[Niemeyer2016c] K. E. Niemeyer. "Selected hydrogen chemical kinetic models," figshare, 2016. https://dx.doi.org/10.6084/m9.figshare.3482906.v1

[Niemeyer2016d] K. E. Niemeyer. "Hydrogen shock tube ignition dataset," figshare, 2016. https://dx.doi.org/10.6084/m9.figshare.3482918.v1

[Olm2014] C. Olm, I. G. Zsely, R. Pálvölgyi, T. Varga, T. Nagy, H. J, Curran, and T. Turányi. "Comparison of the performance of several recent hydrogen combustion mechanisms," *Combust. Flame* 161:2219–34, 2014. http://dx.doi.org/10.1016/j.combustflame.2014.03.006

[Olm2015] C. Olm, I. G. Zsely, T. Varga, H. J. Curran, and T. Turányi. "Comparison of the performance of several recent syngas combustion mechanisms," *Combust. Flame* 162:1793–812, 2015. http://dx.doi.org/10.1016/j.combustflame.2014.12.001

[Petersen2009] E. L. Petersen, M. Lamnaouer, J. de Vries, H. J. Curran, J. M. Simmie, M. Fikri, et al. "Discrepancies between shock tube and rapid compression machine ignition at low temperatures and high pressures," *Shock Waves*, 1:739–44, 2009. http://dx.doi.org/10.1007/978-3-540-85168-4_119

[PrIMe2016] "Process Informatics Model," http://primekinetics.org. Accessed: 29-05-2016.

[Sung2014] C. J. Sung, H. J. Curran, "Using rapid compression machines for chemical kinetics studies," *Prog. Energy Comb. Sci.*, 44:1–18, 2014. http://dx.doi.org/10.1016/j.pecs.2014.04.001

[Travis2016] Travis-CI. "travis-ci/travis-api," GitHub repository. Accessed: 30-May-2016. https://github.com/travis-ci/travis-api

[vanderWalt2011] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation," *Comput. Sci. Eng.*, 13:22–30, 2011. https://dx.doi.org/10.1109/MCSE.2011.37

[Varga2015a] T. Varga, T. Turányi, E. Czinki, T. Furtenbacher, and A. G. Császár. "ReSpecTh: a joint reaction kinetics, spectroscopy, and thermochemistry information system," Proceedings of the 7th European Combustion Meeting, Budapest, Hungary. 30 March–2 April 2015. http://www.ecm2015.hu/papers/P1-04.pdf

[Varga2015b] T. Varga. "ReSpecTh Kinetics Data Format Specification v1.0," 25 March 2015. http://respecth.hu/

[You2012] X. You, A. Packard, M. Frenklach. "Process Informatics Tools for Predictive Modeling: Hydrogen Combustion," *Int. J. Chem. Kinet.*, 44:101–116, 2012. https://dx.doi.org/10.1002/kin.20627