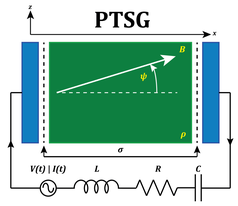
KGMf – Model class and Boltzmann Equation Solver

Janez Krek[1], Guy Parsey[1], John Verboncoeur[1]

[1]Michigan State University ({krek,parseygu,johnv}@msu.edu)



Motivation

The Kinetic Global Model framework (KGMf), a Python-based framework developed at Michigan State University, offers users great flexibility in defining various system parameters, that are defined as system variable dependent functions or as a constants - including the electron energy distribution function (EEDF). The KGMf was used to simulate microwave assisted jet flame [1], multi-phase chemistry [2], and rare gas metastable laser reaction kinetics [3].

Model class

Model class is "wrapper" class around current functionality of KGMf - making the use of the KGMf simpler for new users and keeping the same functionality (and flexibility) for advanced users. Currently supported ways of use Model class are:

- using KGMf provided script (runkey file):
  Running simulation for `10,000` steps from $t = 0$ to $t = 10^{-4}$s using reaction file `RF.txt` and simulation file `SF.txt`. Plot with species densities will be displayed on the screen.
  ```
  /home/nice-user/codes/kgm-framework/runkey_model.py -rf RF.txt -sf SF.txt -run 0 1e-4 1000
  ```
- user created running script:

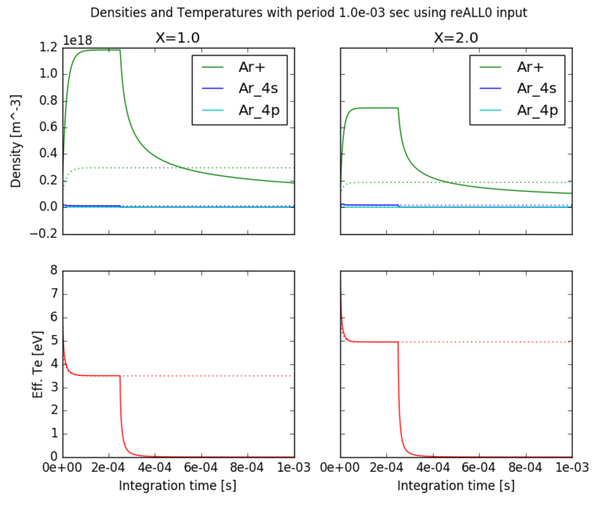
```
import sys
KGMf_root_dir = "/home/nice-user/kgm-framework"
# add path to KFM src directory if it is not already in system path
sys.path.append(KGMf_root_dir) if KGMf_root_dir not in sys.path else None

# relative to being inside of the kgm-framework directory
from src.KGM_user.model import Model
# define model with reaction and simulation data input files
m = Model("RF.txt", "SIF.txt")
# define simulation running time (start, end) and number of steps
m.simulationTime(0.0, 1e-4, 10000)
m.run()                 # run the simulation
m.show()                # show prepared plots (densities)
```

Boltzmann equation solver

KGMf enables the use of arbitrary defined EEDF - defined at the beginning of a simulation either as a constant value or as a system variable dependent function, e.g. $f = f(\epsilon, \alpha, \beta)$ or at least electron energy $f(\epsilon)$. This offers certain degree of flexibility in describing a system for simulation, but in some cases the assumed EEDF does not offer adequate description of the system (plots on the right displays results from two different assumed EEDFs: left for Maxwell and right for Druyvesteyn distribution). To fully describe the system, the EEDF (or better, reaction rates) have to be computed in each simulation step and that could be achieved using:

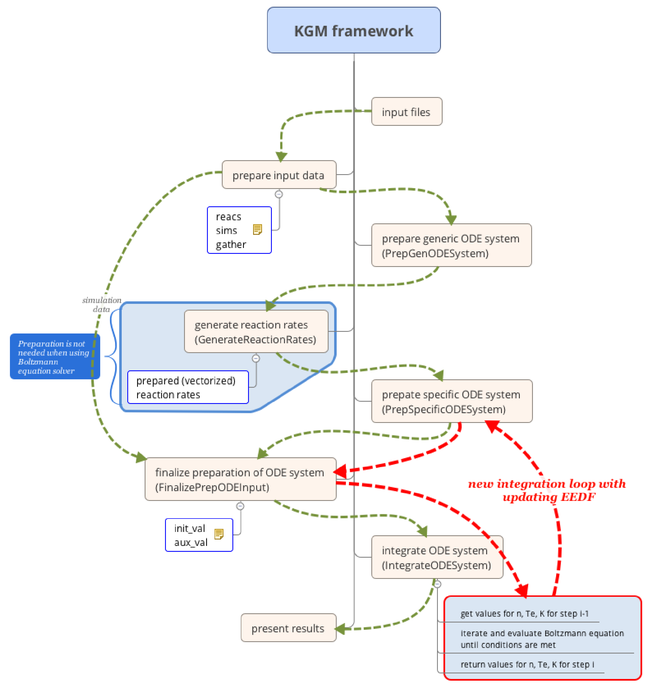
- Boltzmann equation solver
- Monte-Carlo method



Updated flowchart

Regardless of the method used, simulation time will be much longer as currently used optimizations will not be beneficial (parameter space of time-dependent EEDF would be too large to benefit from pre-computed splines).
With implemented Boltzmann solver (presently we are plan of using BOLOS [4] because of the language similarity with KGMf), there are two major changes in KGMf flowchart (as indicated on flowchart):

- no pre-computing of reaction rates
- densities, Te and rates are computed in each simulation step



Proposed method of computing reaction rates does not limit the implementation of used method - the computation of self-consistent system could be done using Boltzmann equation solver (single-, two- or multi-term approximation [5, 6, 7]). Selection of the method is (almost) arbitrary and easily upgradable in future.

Future work

Future work on field of better user friendliness of KGMf, following will be pursued:

- enable run of "batch runs" using Model class
- enable saving/loading intermediate configuration/state of the system from user Model class
- add flexibility in forming custom output, either as images and/or raw values for (external) post-processing

On field of updating core functionality of KGMf, updates will be done in following areas:

- add, test and verify Boltzmann equation solver (Bolos like)
- check and add other methods (e.g. Monte Carlo) for computing reaction rates computation to provide self-consistent system
- optimizing Boltzmann equation solver to utilize multi-core/multi-thread

[1] G. Parsey, Y. Güçlü, J. Verboncoeur and A. Christlieb, ICOPS, doi: 10.1109/PLASMA.2013.6634762 (2013)
[2] G. Parsey, Y. Güçlü, J. Verboncoeur and A. Christlieb, ICOPS, doi: 10.1109/PLASMA.2014.7012415 (2014)
[3] G. Parsey, J. Verboncoeur, A. Christlieb and Y. Güçlü, ICOPS, doi: 10.1109/PLASMA.2015.7179543 (2015)
[4] A. Luque, https://pypi.python.org/pypi/bolos (2004)
[5] G. Petrov, R. Winkler, J. Phys. D: Appl. Phys. 30 (1997) 53–66
[6] G. L. Braglia, M. Diligenti, J. Wilhelm, R. Winkler, Il Nuovo Cimento D, February 1990, Volume 12, Issue 2, pp 257–277
[7] R. Winkler, J. Wilhelm, Il Nuovo Cimento D, July 1990, Volume 12, Issue 7, pp 1005–1014