PlasmaDM

A Python framework for modeling, optimizing, and propagating uncertainty in plasma-surface kinetics.

Table of Contents

Introduction

PlasmaDM provides a flexible, extensible platform to simulate plasma-surface reaction kinetics, fit model parameters to experimental data, and quantify uncertainty via Monte Carlo error propagation. With symbolic ODE construction under the hood, PlasmaDM makes it easy to explore new reaction mechanisms and optimization strategies.

Theoretical formulation

Model Descritption and Observable

Now, we present a brief introduction to the theoretical formulation addressed on this paper. The general formulation of the system corresponds to:

$$\frac{d}{dt}\vec{y}(t) = \vec{F}(\vec{x}, \vec{y}(t); \theta), \;\; \vec{y}(t_0) = \vec{y}_0 \tag{1}$$

where

- $\vec{x}$: vector of the input experimental conditions (e.g. wall temperature, gas temperature, ...)
- $\vec{y}$: vector of species concentrations
- θ: set of parameters that governs the surface kinetics (e.g. energy activation barrier, desorption frequencies, ...)
- $F(\cdot)$: system of chemical equations that model the surface and it is obatined from the kinetics scheme

The steady-state regime, corresponds to the condition: $d\vec{y}/dt = 0$

In our system, we are interested in modeling what we can categorize into two different types:

- **Linear** reactions (matrix A):

$$\begin{gathered} B(g) + y_i \rightarrow y_j + C(g) \\ y_i \rightarrow y_j + C(g) \end{gathered} \tag{2}$$

They correspond to the mechanims such as adsorption, where a gas specie, $B(g)$, adhres to a surface kinetics specie, y_i, or desorption, the reverse mechanims

- **Bimolecular** reactions (tensor B):

$$y_i + y_j \rightarrow y_{i'} + y_{j'} + D(g) \tag{3}$$

This type correspond to mechanims like difusion or *LH*-recombination, where an atom diffuses along the surface is stabilishes a connection with other site

Using this mechanisms, we can develop , by using the surface scheme, a mesoscopic model that describes the average occupation of the surface by the different chemical species. However, the macroscopic obsrvable that can be measured corresponds to the recombination probability, γ, which is given by:

$$\gamma = \hat{T}(\vec{x}, \vec{y}^*(\vec{x}, \theta), \theta) \tag{4}$$

where $\hat{T}(\cdot)$ correspond to a operator that, by selecting the appropriate reactions from the surface kinetics scheme, projects the computed steady-state chemical concentrations, $\vec{y}^*$, into the scalar and observable quantity γ. More details and the derivation of such observable are presented on previously cited papers.

Optimization Problem

Having the physical simulator that allow us to compute the observable, γ, for all the input conditions, we intend to tackle the optimization problem that corresponds to find the best parameters of the simulator, θ, that describe better the experimental data. We can summarize the problem as:

$$\min_{\theta} J(\theta) = \sum_{i \in D} \left(\frac{\gamma_{exp,i} - \gamma_i(\theta)}{\gamma_{exp,i}} \right)^2 \tag{5}$$

where i iterates over all the input experimental conditions considered.

Error Propagation

We also intend to compute uncertainty of model predictions resulting from experimental uncertainties in input parameters. We assume the following set of independent random variables:

$$X = \{X_1, X_2, \ldots, X_n\} \tag{6}$$

with knowns pdfs $P(X_1), P(X_2), \ldots, P(X_n)$, that modulate input experimental conditions. If we assume that the inputs are independent, the joint distribution is given by:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i) \tag{7}$$

The deterministic relationship between inputs and observable, γ, is defined as:

$$\gamma = F(X_1, X_2, \ldots, X_n) \tag{8}$$

And so it defines the conditional probability $P(\gamma | X_1, \ldots X_n)$, which in our deterministic case is given by:

$$P(\gamma | X_1, \ldots, X_n) = \delta(\gamma - F(X_1, X_2, \ldots, X_n)) \tag{9}$$

The resulting PDF for γ is calculated as:

$$P(\gamma) = \int_{x_1} \ldots \int_{x_n} \delta(\gamma - F(x_1, \ldots, x_n)) P(x_1, \ldots, x_n) dx_1 \ldots dx_n \tag{10}$$

This integral is numerically estimated using the **Monte Carlo sampling method**.

More about the theoretical foundation and detailed methodlogies introduced here can be found on the following paper:

- Pedro Viegas *et al.* 2024 *Plasma Sources Sci. Technol.* **33** 055003,
- V. Guerra, "Analytical Model of Heterogeneous Atomic Recombination on Silica," *IEEE Transactions on Plasma Science*, vol. 35, no. 5, pp. 1397–1403, Oct. 2007.
- José Afonso *et al* 2024 *J. Phys. D: Appl. Phys.* **57** 04LT01

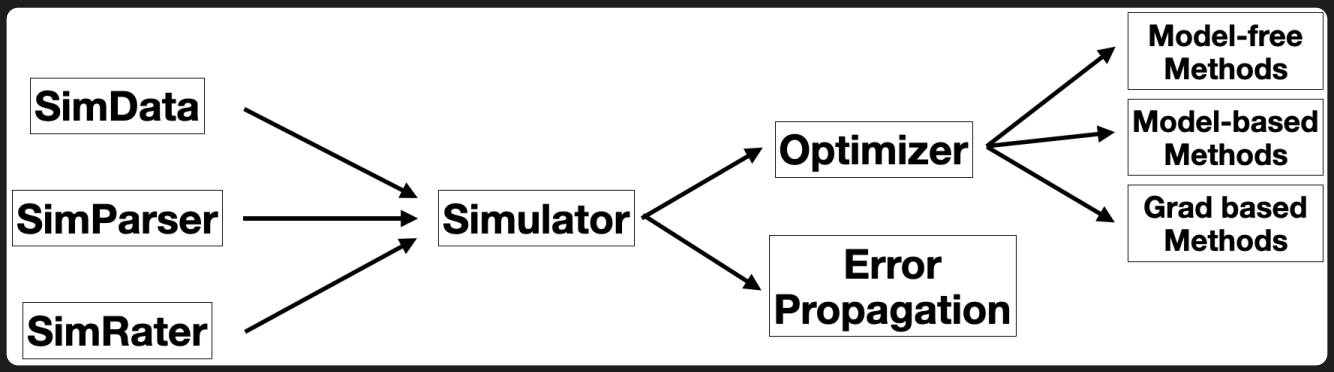
Project Structure

The code is written in *Python* and utilizes *Numpy, PyTorch, h5py, SciPy, and pathos.multiprocessing* packages.

The project is composed of the following classes and methods that perform the following roles:

- The `SimData` class picks all the input experimental data, measures output observables from different source files, and creates a data buffer with the proper data structure used in the code;
- The `SimParser` class creates the symbolic representation of the system of chemical equations using Sympy directly obtained from the provided surface kinetics scheme (through a JSON file);
- The `SimRater` class calculates the rates for each chemical equation provided in the surface kinetic scheme for all experimental conditions.

- The `Simulator` class leverages the `SimData` , `SimParser` , and `SimRater` classes and computes steady-state surface chemical concentrations and the output observable using Scipy numerical methods (ODE solvers)
- The `Optimizer` class modifies the simulator hyperparameters provided on the surface kinetics scheme that is chosen to be optimized and computes the objective loss.
- The `Model-free Methods` , `Model-based Methods` and `Grad-based methods` classes enable us to solve the optimization problem using either model-free or model-based algorithms.
- The `ErrorPropagation` class allows us to propagate errors from the input experimental conditions to the output observable.



Details Concerning the Implementation :

Simulator:

The surface kinetics scheme is provided through a JSON file, which also includes the default model parameters, θ, the rates used on each chemical equation and set of reactions that directly participate in the observable computation. An example of application is presented on folder tests.

Optimization:

For optimization we use three different approaches:

- Model-free methods: Our exploration revolves mainly around the differential evolution algorithm. Examples of application are presented on the `study_opt_model_free` folder
- Gradient based methods: Leverage on PyTorch, we compute the gradients efficiently (when compared to the numerical counterpart), which we provide for minimization algorithm used. Examples of application are presented on `study_opt_grad_based` folder. Considering the $J(\theta)$ and $\gamma_m(\theta)$ where m corresponds to an input experimental condition, we define:

$$\begin{aligned} J\left(\theta\right) &= \frac{1}{|D|}\sum_{m\in D}\left(\frac{\gamma_m(\theta)-\gamma_{exp,m}}{\gamma_{exp,m}}\right)^2 \\ \gamma_m(\theta) &= \sum_k [T_1(\theta)]_k y^*(\theta)_k + \sum_{kl}[T_2(\theta)]_{kl} y^*(\theta)_k y^*(\theta)_l \end{aligned} \tag{11}$$

where we have that:

$$\vec{F}(\vec{x},\vec{y}^*(\vec{x},\theta);\theta) = \vec{0} \tag{12}$$

As a result the derivatives are given by:

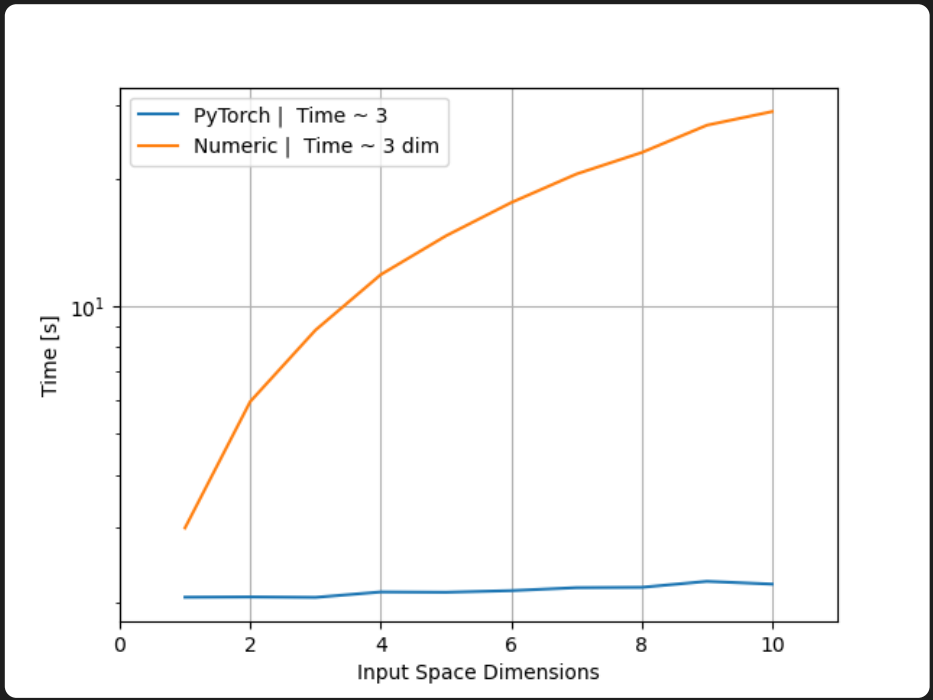
$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{1}{|D|}\sum_{m\in D}\frac{2}{\gamma^2_{exp,m}}(\gamma_m(\theta)-\gamma_{exp,m})\frac{\partial \gamma_m(\theta)}{\partial \theta_i} \tag{13}$$

$$\frac{\partial \gamma(\theta)}{\partial \theta_m} = \sum_k \left(\frac{\partial}{\partial \theta_m}[T_1(\theta)]_k\right) y^*_k + \sum_{ln}\left(\frac{\partial}{\partial \theta_m}[T_2(\theta)]_{ln}\right) y^*_l y^*_n + \sum_k [T_1(\theta)]_k \frac{\partial y^*_k}{\partial \theta_m} + \sum_{ln}\left[T_2(\theta)+T_2^T(\theta)\right]_{ln} y^*_l \frac{\partial y^*_n}{\partial \theta_m} \quad ($$

Using the **Implicit Differentiation** Theorem:

$$\begin{aligned} \frac{dF_i}{d\theta_m} = 0 \implies \frac{\partial F_i}{\partial \theta_m} + \sum_n \frac{\partial F_i}{\partial y_n}\frac{\partial y_n}{\partial \theta_m} = 0 \\ \Leftrightarrow [\partial_\theta F]_{im} + \sum_n [\partial_y F]_{in}[\partial_\theta y]_{nm} = 0 \\ \implies [\partial_\theta F] + [\partial_y F].[\partial_\theta y] = \vec{0} \end{aligned} \tag{15}$$

where we have that: $[\partial_\theta F] \in \mathbb{R}^{\#F \times \#\theta}$, $[\partial_y F] \in \mathbb{R}^{\#F \times \#y}$ and $[\partial_\theta y] \in \mathbb{R}^{\#y \times \#\theta}$. Since, $[\partial_y F]$ is not a square matrix, this system of equations is solved in the least-square sense.



- Model-based methods: To be explored (Surrogate model with GP, Active Learning, ...)

Furthemore, the implementation allows multiprocessing.