Kinetic Parameter Estimation Package Documentation

Indian Institute of Technology, Madras

June 6, 2023

Contents

List of Code Listings

1 Introduction

This documentation contains detailed description of all classes and methods in KineParEs package.

2 Methods in Package

par_estimate() class description

2.1 Input - __init__()

The __init__() method is the initializer/constructor of the par_estimate class. It is called when you create an instance of the class. This method sets up the initial state of the object and assigns values to its attributes.
Input data required (with shape):

Input required	Required Shape/Type
Absorbance Spectra	$(L \times M)$
Initial Concentration	$(S \times 1)$
Stoichiometric Matrix	$(R \times S)$
Residence Times	$(L \times 1)$
Wavelengths	$(M \times 1)$
Number of parameters	int
Model Type	Integer String
Kinetic Model	A function representing custom kinetic model

Table 1: Input data format

Input arguments:

- **A** : Absorbance Spectra $(L \times M)$
- **c0** : Initial Concentration $(S \times 1)$
- **N** : Stoichiometric Matrix $(R \times S)$
- **tau** : Residence Times $(L \times 1)$
- **lamdas** : Wavelengths at which we have spectra $(M \times 1)$
- **n** : Number of parameters in kinetic model (int)

- **model_type** : A integer to select from set of default models (string)
- **fun** : A function representing custom kinetic model

*fun argument can only be given if its not present in models library
*L - No. of residence times
*M - No. of wavelengths
*S - No. of species
*R - No. of reactions
Arguments of the __init__() method shown in table 1 and their descriptions are given below:

- **A**: This argument represents the absorbance spectra and is expected to be a 2D numpy array. It contains the absorbance data for different wavelengths and residence times.
- **c0**: This argument represents the initial concentrations of species and is expected to be a 1D numpy array. It contains the initial concentrations of different chemical species involved in the reactions.
- **N**: This argument represents the stoichiometric matrix and is expected to be a 2D numpy array. It describes the stoichiometry of the reactions by specifying the number of moles of every species in each reaction.
- **tau**: This argument represents the time intervals and is expected to be a numpy array. It defines the time points at which the spectra were measured.
- **lamdas**: This argument represents the wavelengths and is expected to be a numpy array. It contains the wavelength values at which the spectra were measured.
- **n**: This argument represents the number of parameters to estimate in kinetic model. It specifies the size of the parameter vector.
- **model_type**: This argument represents the type of kinetic model to be used. It can take values '1', '2', or 'custom' corresponding to different predefined models or a custom user-defined model. Here '1', '2' represent kinetic models of Lipase catalysed hydrolysis reaction and Wittig reaction.
- **fun**: This argument represents a custom user-defined function for the kinetic model. It is optional and only required if model type is set to 'custom'. If provided, this function will be used to calculate the reaction rates instead of the predefined models.

Code for creating a par_estimate class object is given below.

```
#importing KineParEs package
from KineParEs import *
#creating parameter estimation object
par=par_estimate(A,c0,N,tau,lamdas,n,"custom",fun)
```

It also performs some initial validation checks on the input parameters to ensure they meet the expected requirements, such as checking the shapes and dimensions of the arrays, matching sizes of species and concentrations, and verifying the validity of the model type and custom function.
Overall, the __init__() method sets up the necessary data and parameters for further analysis and estimation of parameters in chemical kinetic models based on the provided input.

2.2 Preprocessing Data - preprocess()

It is used to perform the preprocessing of the spectra data before further analysis. It also checks for data consistency (shapes and sizes) of input data given.

Checking for Negative Initial Concentrations: The function first checks if any initial concentration values (**c0**) are negative. If any negative values are found, an exception is raised to indicate that the initial concentration cannot be negative.

Removing Negative Spectra Values: The function checks the spectra array (**A**) to remove any negative values. It replaces negative values with zero while keeping non-negative values unchanged. This step ensures that the spectra data is valid and non-negative. Code for using the function is given below.

```
#checks input dimensions of all data given
#applies mentioned preprocessing techniques
par.preprocess()
```

By performing these preprocessing steps, the preprocess() function ensures that the input data is properly formatted, consistent, and ready for further analysis. It helps to eliminate any potential issues or inconsistencies in the data that could affect the accuracy of subsequent calculations and estimations.

2.3 Savitzky-Golay filter - savitzky_golay()

It applies the Savitzky-Golay filter to smooth the spectra data. The Savitzky-Golay filter is commonly used for data smoothing and noise reduction.

Input arguments: The function takes two optional parameters: **window_size** and **order**. **window_size** specifies the size of the smoothing window, which is the number of adjacent points used for fitting curve. **order** specifies the order of the polynomial used in the fitting process. The resulting smoothed spectra are stored back in the spectra attribute. Code for using the function is given below.

```
#applies Savitzky-Golay smoothing filter to the data
par.savitzky_golay(window_size=5,order=3)
```

Applying the Savitzky-Golay filter helps reduce noise and variability in the spectra data. This can be beneficial for enhancing the quality of the data and improving the accuracy of subsequent analysis and modeling tasks.

2.4 Selecting wavelength range - subset_slice()

It allows selecting a subset of the spectra and corresponding wavelength range by removing a certain number of wavelengths from the start and end of the data.

Input arguments: The function takes two parameters: **a** and **b**. **a** specifies the number of wavelengths to remove from the start of the spectra. **b** specifies the number of wavelengths to remove from the end of the spectra. The function performs some input validation checks to ensure the slicing parameters are valid. It modifies the both spectra and lamdas attribute. Code for using the function is given below.

```
#selecting required part of spectrum
#removing first 100 and last 20 wavelengths
par.subset_slice(100,20)
```

By using the subset slice() function, you can focus on a specific range of wavelengths in the spectra data, which can be useful for isolating specific features or regions of interest. This allows for more targeted analysis and modeling based on the selected subset of the spectra.

2.5 Multiplicative Scatter Correction - MSC()

It performs Multiplicative Scatter Correction (MSC) on the spectra data. MSC is a preprocessing technique commonly used in spectroscopy to correct for

the effects of scattering and variations in baseline intensity across different spectra. Its is done in two steps given below.

STEP 1: Ordinary least squares / Regression

$$d_i = b_i d_{\text{ref}} + a_i + \epsilon_i \tag{1}$$

where, d_{ref} is the reference spectrum, given as (if not provided),

$$d_{\text{ref}} = \frac{\sum d_i}{N}$$

STEP 2: Shifting and Scaling

$$\mathbf{d}_{il,MSC} = (\mathbf{d_i} - \mathbf{a}_i) / \mathbf{b_i} \tag{2}$$

Input arguments: The function takes an optional parameter **dref** which represents the reference spectrum. If **dref** is not provided, the mean spectrum is used as the reference. The function updates spectra attribute with the corrected spectra. Code for using the function is given below.

```
#For applying MSC on spectra
par.MSC()
```

The MSC() function helps to improve the comparability and interpretability of the spectra data by reducing unwanted variations due to scattering and baseline effects. By applying MSC, the spectra are normalized, allowing for more accurate analysis and modeling.

2.6 Visualizing Spectra - spectrum_plot()

It is used to visualize the spectra data after or before preprocessing.

```
#For plotting given Absorption spectrum(A)
par.spectrum_plot()
```

The spectrum_plot() function provides a visual representation of the spectra data. It allows you to observe the absorbance values across different wavelengths, providing insights into the characteristics of the spectra. This plot can be useful for analyzing the spectral profiles and identifying any patterns or trends in the data.

2.7 Number of Reactions - variance_ratio()

It calculates the cumulative variance ratio explained by PC's and provides insights into the number of significant components or independent reactions present in the chemical reaction system.

The number of observable reactions can be determined by analyzing the rank of Absorbance matrix **A**, which can be derived solely from the spectral data. This is achieved through singular value decomposition (SVD) of A, where the number of dominant singular values indicate the rank of the matrix. By comparing the variance captured by these dominant singular values ($\mathbf{S_d}$) to the total variance captured by all singular values, the number of dominant singular values can be determined. The threshold for the percentage of variance captured by $\mathbf{S_d}$ is set to be assumed to be $\geq$ 98%.

$$\text{Var}\,(\mathbf{S_d}) = \frac{\sum_{i=1}^{S_d} \mathbf{S_i}}{\sum_{j=1}^{L} \mathbf{S_j}} \tag{3}$$

Code for using the function is given below.

```
'''For plotting cumulative variance explained
   against number of components for
   rank analysis of A to find maximum
number of reactions in reaction system'''
par.variance_ratio()
```

The variance_ratio() function provides a plot of the cumulative variance ratio, allowing you to determine the number of significant components or independent reactions in the reaction system. It helps in understanding the contribution of each principal component and determining the optimal number of components to consider for further analysis or modeling.

2.8 Concentration profile - conc_profile()

It is used to generate and plot the concentration profiles of species over time based on the provided parameters.
Input Parameters: The conc_profile() function takes a single parameter **pars**, which is an tuple or list of parameter values. Code for using the function is given below.

```
#For plotting concentration profile with estimated parameters
par.conc_profile(tuple(params_array))
```

The conc_profile() function allows you to visualize and analyze the concentration profiles of the chemical species over time based on the provided parameters. It helps in understanding the dynamics of the chemical reactions and their impact on the concentrations of the species involved.

2.9 Forward & Backward Exploratory Factor Analysis - forward_EFA() & backward_EFA()

It performs Forward Exploratory Factor Analysis (EFA) and Backward Exploratory Factor Analysis (EFA) on the spectra data respectively.

An EFA plot can reveal valuable information about the system being analyzed. It typically represents the evolution of eigenvalues over the course of an experiment (such as pH changes, etc.), indicating the emergence and disappearance of different components in the system. One can identify the number of significant components or factors in a system. The appearance of a new eigenvalue in the plot can signal the emergence of an additional component in the system.

EFA plots can also be used to detect changes in a system over time or under different conditions.

Input arguments: Both the function takes two parameters: **step** and **comp**. step specifies the increment in number of rows of data on which analysis is performed. comp specifies the number of top significant components to plot. The default value is 3. The resulting plot shows the trend of the log of eigen values vs the number of data increments. Code for using the function is given below.

```
#For Forward Evolving Factor analysis plot
par.forward_EFA(step=1,comp=7)
#For Backward Evolving Factor analysis plot
par.backward_EFA(step=1,comp=7)
```

The forward_EFA() and backward_EFA() functions provide a way to analyze the significant components or factors in the spectra data by plotting the log of eigen values against the number of data increments. These functions help in analysing at what wavelength range the product is getting formed/progress of the reaction.

2.10 Maximum reactions - rmax()

It calculates the maximum number of reactions that can be inferred from a given atomic matrix.

Input arguments: The rmax() function takes one parameter: **A**. **A** is a 2D numpy array representing the atomic matrix of the species present in the chemical system. Code for using the function is given below.

```
'''For finding maximum number of reactions
using Atomic matrix Am'''
```

```
par.rmax(Am)
```

The rmax() function is useful for determining the maximum number of reactions that can be estimated or identified based on the given atomic matrix. It provides insights into the system's complexity and helps in setting appropriate constraints or expectations when performing parameter estimation or model inference tasks.

2.11 Optimized Kinetic Parameters - optim()

It outputs the optimized parameter values and the square of Frobenius norm between measured and predicted absorbances.
Input arguments: The optim() function takes three main parameters: **niters**, **bounds**, and **algo**. **niters** is a number of randomized different initial guesses for parameters to find global optimum. **bounds** (optional) is a list or tuple containing the lower and upper bounds for each parameter. If not provided, the optimization is performed without bounds. **algo** (optional) specifies the optimization algorithm to use. The default method is 'Nelder-Mead', which is a simple and robust optimization algorithm. Other options include 'BFGS', 'CG', 'L-BFGS-B', etc.

The calibration-free approach for estimating parameters θ of the reaction system starts with the following assumptions

- Information about stoichiometric matrix **N**, initial concentration of species **c0** and reaction rate model structure **r(c,**θ**)** are available
- Minimum number of absorbing species is equal to R
- The spectral data and concentration are related by a linear model (Beer Lambert's law)

A constrained objective function is formulated which minimizes the Frobenius norm between measured and predicted absorbances as given as:

$$\begin{aligned} \min_{\theta} J_A, \quad J_A \equiv \|\mathbf{A} - \hat{\mathbf{A}}(\theta)\|_{\mathbf{F}}^2 &= \left\| \left[\mathbf{I_L} - \mathbf{C}(\theta)\mathbf{C}^{+}(\theta)\right] \mathbf{A} \right\|_{\mathbf{F}}^{\mathbf{2}} \\ \text{s.t.} \quad \frac{d\mathbf{c}(\theta)}{d\tau} &= \mathbf{N}^{\mathbf{T}}\mathbf{r}(\mathbf{c},\theta), \quad \theta \in \left[\theta^{\mathbf{L}}, \theta^{U}\right] \geq \mathbf{0} \end{aligned} \tag{4}$$

In the above Equation 2.11, $\mathbf{I}_L$ is an identity matrix of dimension $L \times L$, $\mathbf{C}^{\dagger}(\theta)$ is the Moore-Penrose pseudo inverse of **C** matrix, and $\|\cdot\|_F$ denotes the Frobenius norm. The second expression in the objective function J_A is obtained from the factorization of $\hat{\mathbf{A}}$ using Beer Lambert's law, i.e., $\hat{\mathbf{A}}(\theta) = \mathbf{C}(\theta)\mathbf{E} = \mathbf{C}(\theta)\mathbf{C}^{\dagger}(\theta)\mathbf{A}$. One should note that **C** is a function of both **r** and θ. It minimized

to find the best estimates of the unknown parameters θ of the kinetic model. Code for using the function is given below.

```
'''LB & UB for each parameter and number of iterations
(for different initial guess of parameters) are given
as input for optim function. This function outputs
final estimated parameters and resiudal value as tuple.'''
output=par.optim(bounds=((0.08,0.15), (0.01, 0.025)),niters=500)
```

It provides flexibility by accepting different optimization methods and handles both bounded and unbounded optimization problems. By utilizing this function, you can perform parameter estimation and optimization tasks efficiently.

2.12 Choosing Best Kinetic Model - fun_arr()

It outputs best kinetic model out of the given kinetic model list based on the lowest residual value.
Input arguments: The optim() function takes four main parameters: **arr**, **niters**, **bounds**, and **algo**. **arr** is the list of probable kinetic models (defined like python function). **niters** is a number of randomized different initial guesses for parameters to find global optimum. **bounds** (optional) is a list or tuple containing the lower and upper bounds for each parameter. If not provided, the optimization is performed without bounds. **algo** (optional) specifies the optimization algorithm to use. The default method is 'Nelder-Mead', which is a simple and robust optimization algorithm. Other options include 'BFGS', 'CG', 'L-BFGS-B', etc. Code for using the function is given below.

```
#Finds the best fir kinetic model to the data
par.fun_arr([model1,model2,....,model10],bounds=((0.08,0.15),
    (0.01, 0.025)))
```

The fun_arr() function is useful for selecting the most appropriate kinetic model from a given set of models based on their goodness-of-fit. It iterates through each model, performs parameter optimization, and compares the residuals to identify the model that provides the best fit to the experimental data.

models() class description
The models class represents a collection of kinetic models used for modeling chemical reactions. Detailed description of the models class is given below:

2.13 Input for Kinetic Models - __init__()

The __init__() method is the constructor of the models class.
Input arguments: **N** and **type**. **N** is a 2D numpy array that represents the stoichiometric matrix of the chemical reaction. **type** is a string that specifies the type or category of the kinetic model.

2.14 Library of Kinetic Models - model()

The specific model equations and parameters are implemented in the model() method. **Input arguments**: The model() method takes three parameters: C, t, and K. C is a numpy array representing the concentrations of the reactants or intermediates in the chemical reaction. t is a scalar or numpy array representing the time points at which the concentrations are evaluated. K is a list or numpy array representing the rate constants or parameters of the kinetic model.

The model() method calculates and returns the rate of change of concentrations based on the specified kinetic model type and the given concentrations and parameters. The specific model equations and calculations vary depending on the chosen model type. This function is a key component in the models class, enabling the evaluation and analysis of various reaction systems.

The models class provides a framework for defining and utilizing different types of kinetic models. By specifying the stoichiometric matrix (N) and the type of model (type), the class enables the calculation of reaction rates and simulations based on the chosen model.

3 Sample Datasets

3.1 Lipase catalyzed hydrolysis reaction

The selected reaction involves the hydrolysis of p-nitrophenyl acetate (pNPA) catalyzed by lipase. This process results in the formation of p-nitrophenol (pNP) and acetic acid. The dataset consists a total of 401 evenly spaced wavelengths, spanning from 200 *nm* to 600 *nm*, were used to record each spectrum at 15 different residence times.

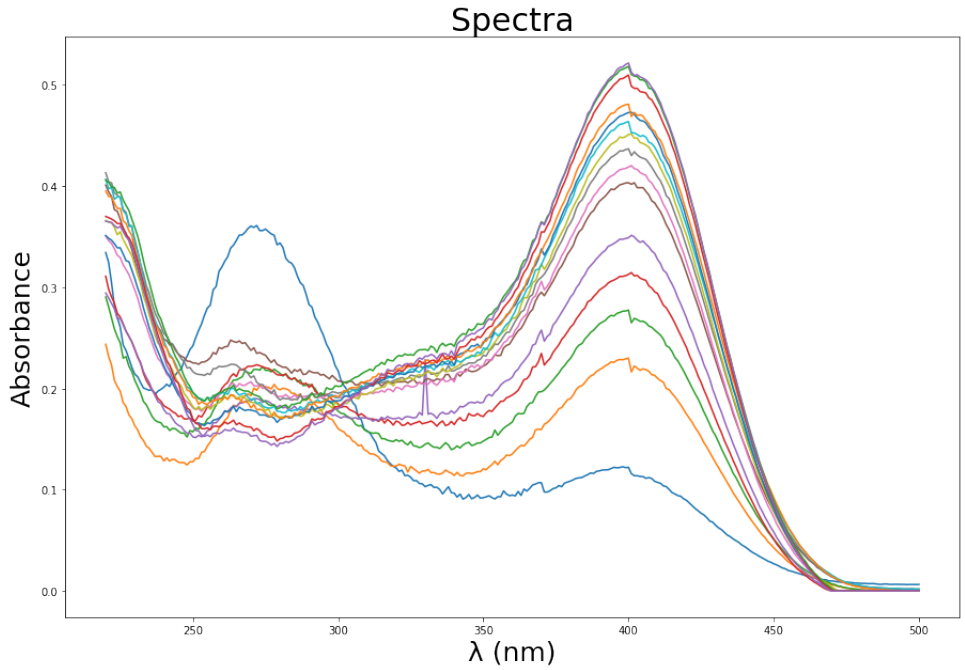

Figure 1: Absorbance spectra of Lipase catalysed hydrolysis reaction system

3.2 Wittig reaction

The synthesis of olefins which involves triphenylphosphonium ylide and aldehydes or ketones, using the Wittig reaction, has been extensively investigated in scientific literature. The dataset consists absorption spectra of the Wittig reaction between (4- nitrobenzyl)triphenylphosphonium bromide and benzaldehyde, with KOH serving as the catalyst. The reaction proceeds in the following manner: the Wittig salt reacts with KOH, resulting in the formation of an unstable ylide intermediate, along with KBr and water. The ylide intermediate then reacts with benzaldehyde, leading to the formation of trans and cis-nitrostilbene, triphenylphosphine oxide, and HBr. Additionally, in the presence of KOH, the Wittig salt undergoes hydrolysis, giving rise to the formation of triphenylphosphine oxide, p-nitrotoluene, and HBr. The dataset consists a total of 201 evenly spaced wavelengths, spanning from 200 *nm* to 600 *nm*, were used to record each spectrum at 15 different residence times.

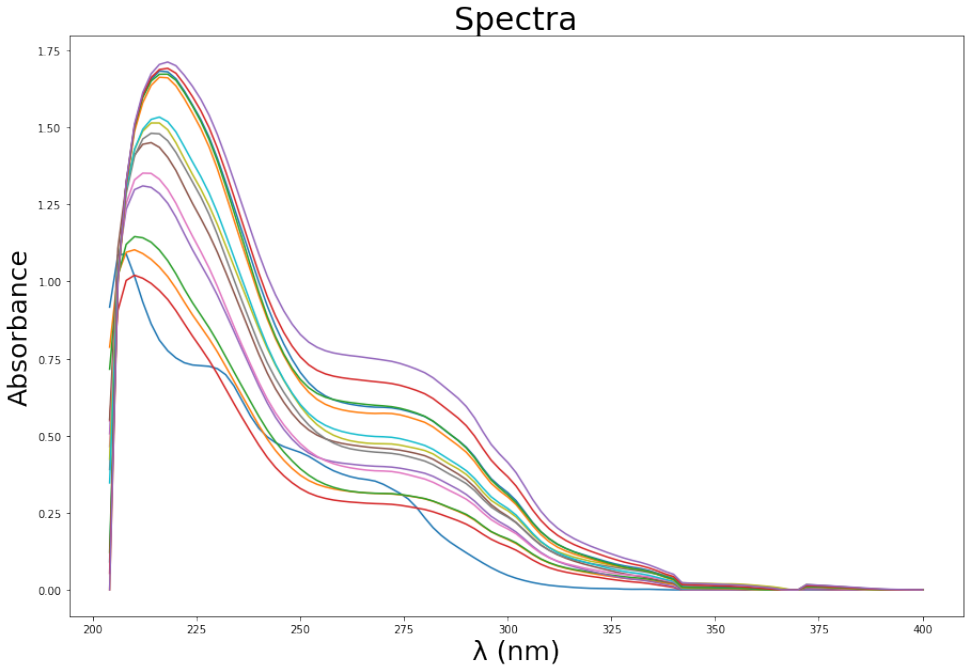

Figure 2: Absorbance spectra of Wittig reaction system