Dual Degree Project

A Software Tool for identifying Kinetic Models from Concentration Data

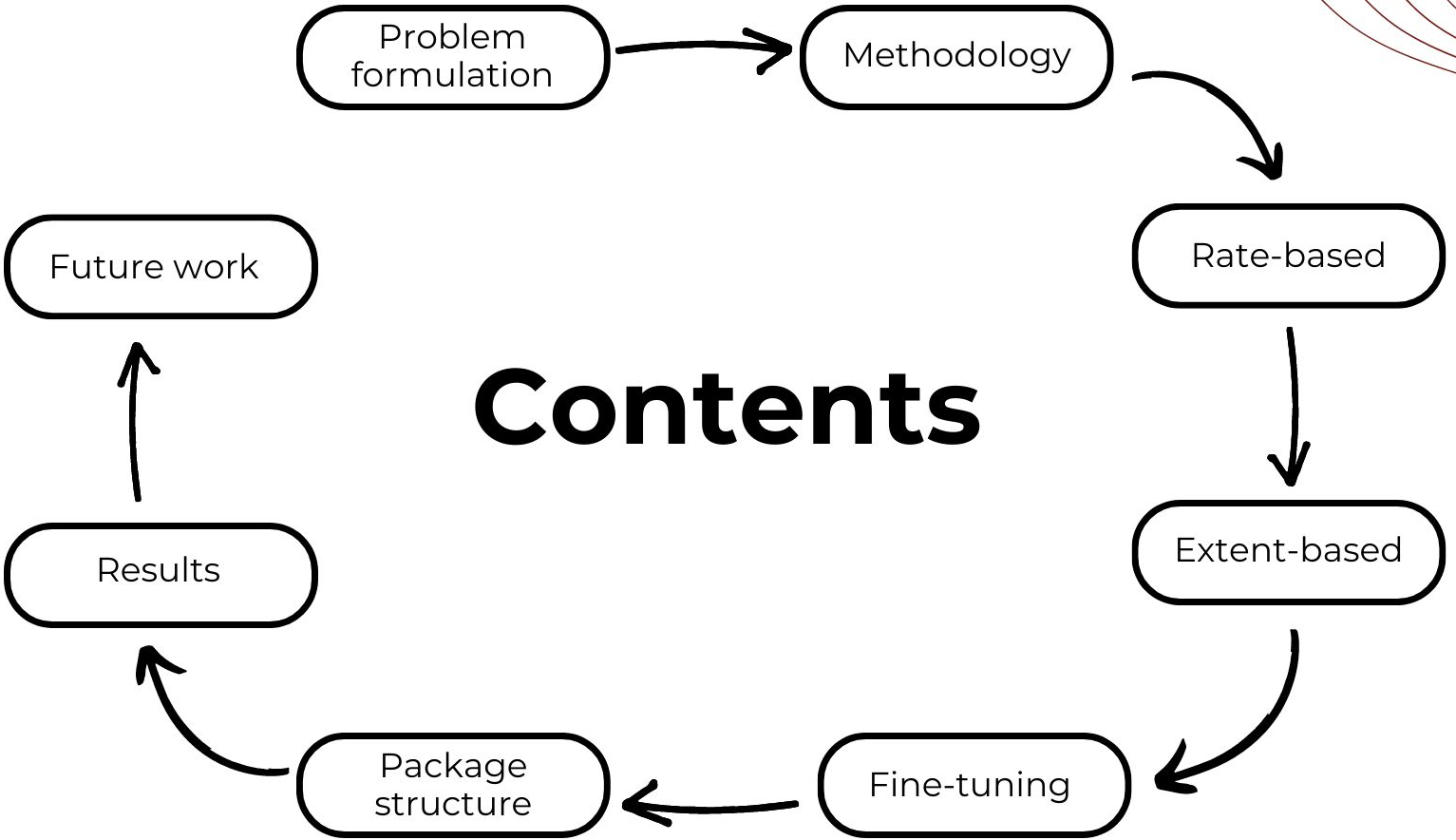
Vignesh Kumar S

B.Tech in Chemical Engineering, M.Tech in Data Science (Dual Degree)

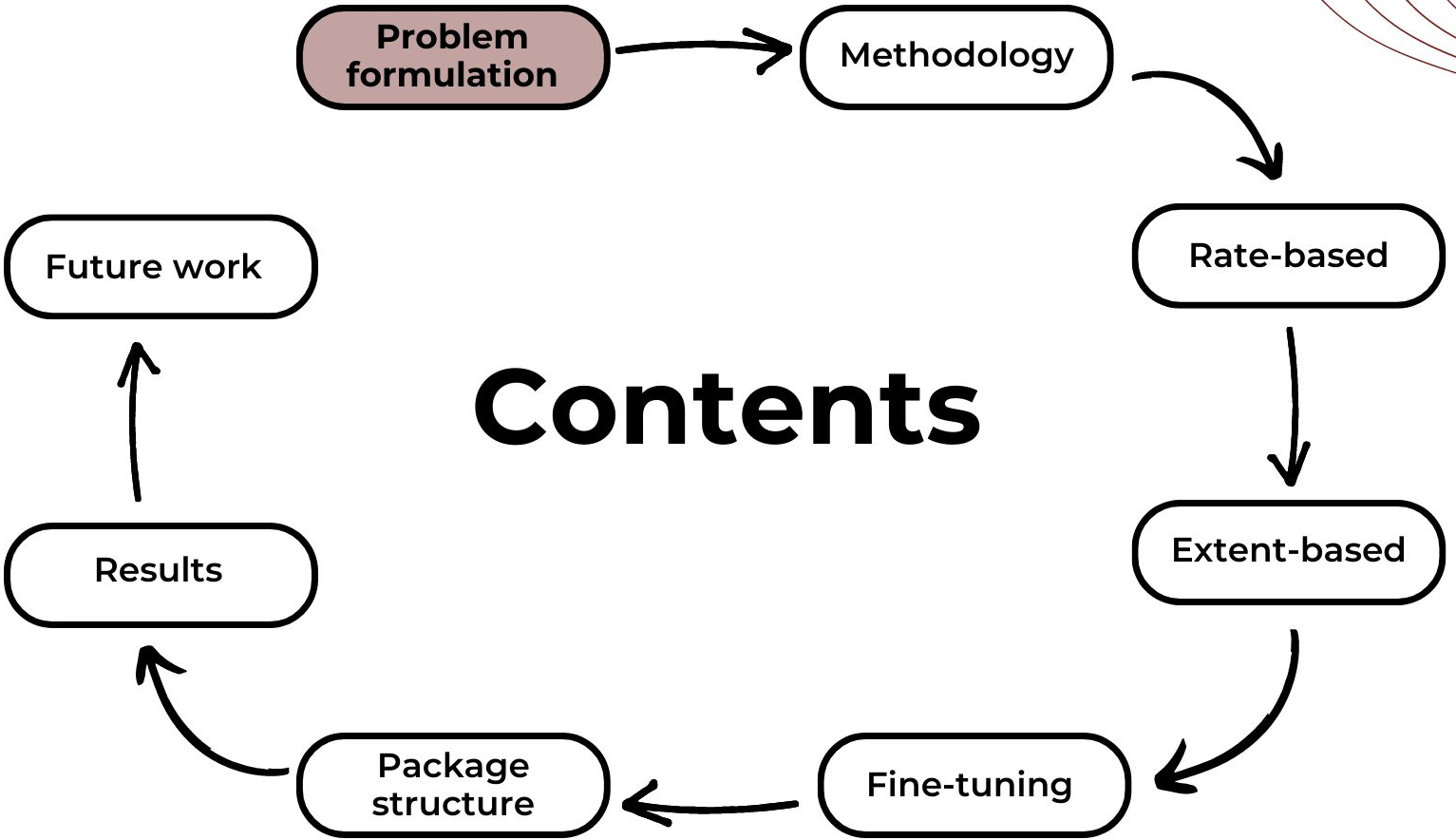
IIT Madras

27/05/23

Contents

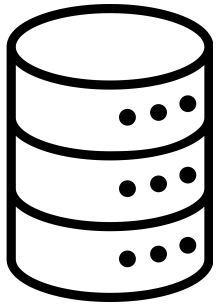

Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

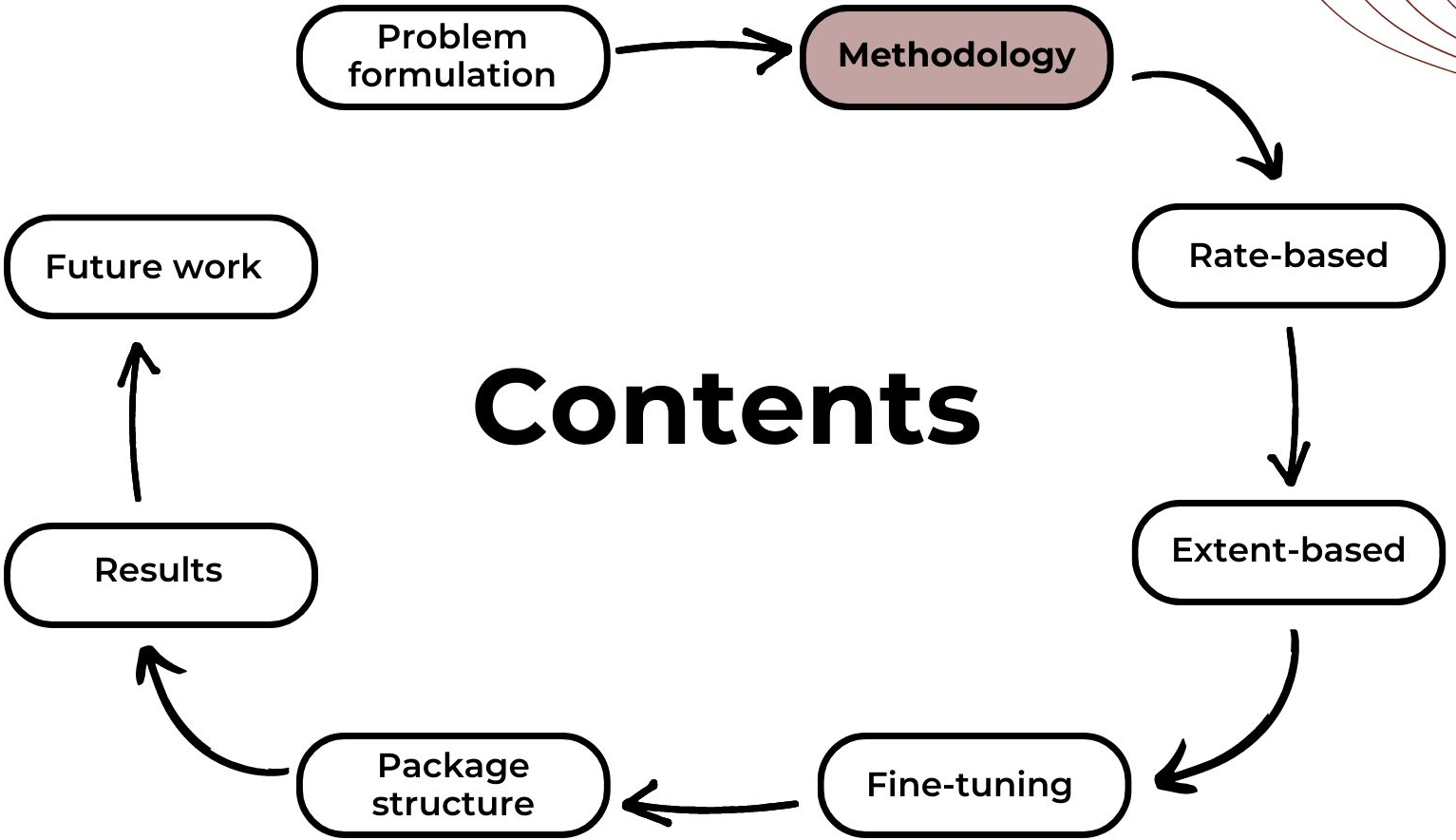
Problem Formulation:

- Knowledge of reaction kinetics crucial, when unavailable, essential to determine from measured data without much experiments
- PyKineMod, a software package tool to determine kinetic models from concentration data
- Input data required for the software (with shape) are:
  - **N**: Stoichiometry Matrix $(R * S)$
  - **Mw**: Molecular weight Matrix $(S * S)$
  - **V**: Volume of the reactor (const, or $(T,)$ shape)
  - **Winhat**: Inlet composition matrix $(P * S)$
  - **uin**: Mass Flow rate of P inlets $(P * 1)$ or $(P * T)$
  - **uout**: Mass Flow rate of outlet const or $(T,)$
  - **n0**: Initial number of moles in the reactor $(S * 1)$



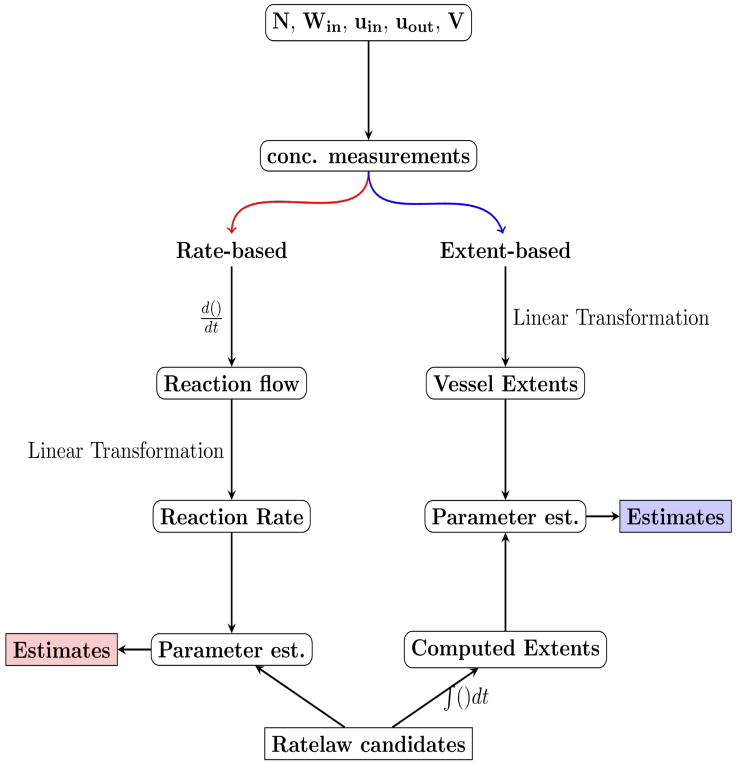
- where S = number of species, R number of reactions, T timesteps, and P inlets

Contents

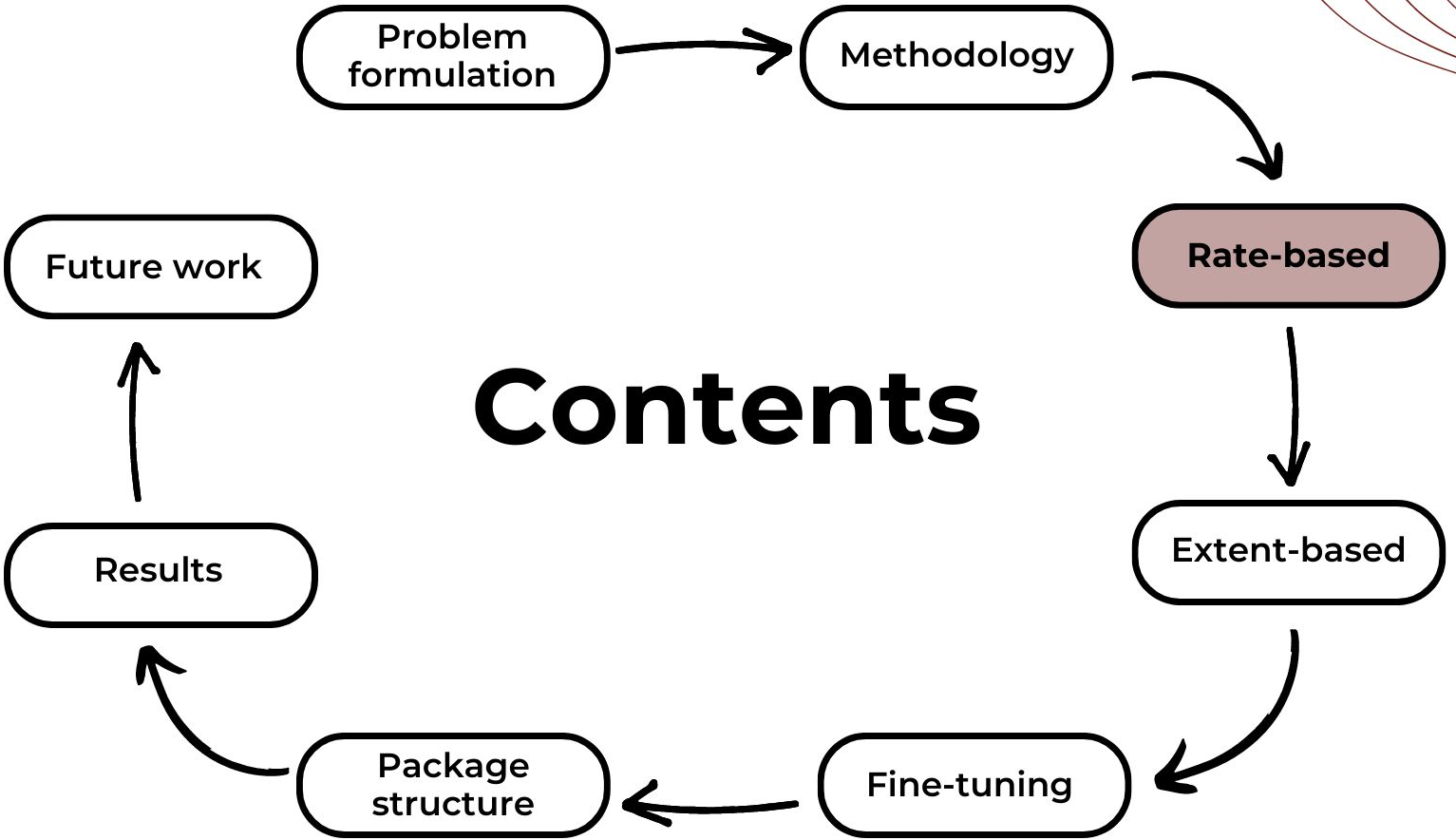

Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Methodology:

- Incremental approach decomposes the identification task into subproblems. Candidates are solved independently without looking into other subproblems avoiding combinatorial time complexity. Hence it is a computationally effective way.
- The package tool supports two methods: Rate-based and Extent-based methods



Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Rate-Based Method

- Rate-based method:

$$\hat{\theta}_i = \min \Sigma_{h=0}^{H} (r_i(t_h) - r_i(\mathbf{c}(t_h), \theta_i))^2$$

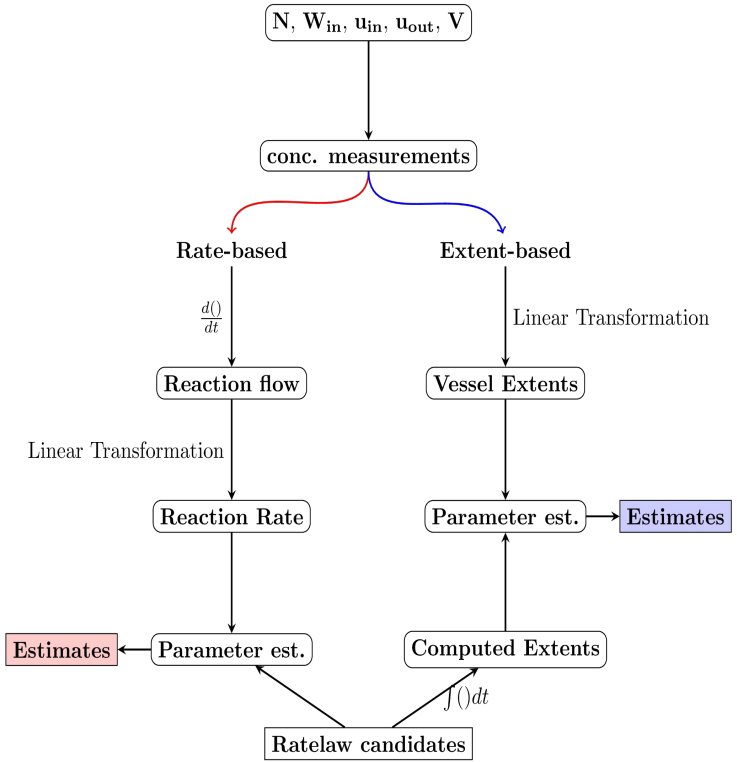
$$\text{s.t. } \mathbf{r}(t_h) = (N^T)^{+} \mathbf{f}(t_h) V^{-1}(t_h)$$

$$\mathbf{f}(t_h) = \frac{d\mathbf{n}}{dt}(t_h) - W_{in} u_{in}(t_h) + \frac{u_{out}(t_h)}{m(t_h)} \mathbf{n}(t_h) \quad (1)$$

$$m(t_h) = \mathbf{1}_S^T \mathbf{M}_W \mathbf{n}(t_h)$$

$$\text{(or)} \quad \dot{m} = \mathbf{1}_p^T \mathbf{u}_{in} - u_{out}, \quad m(0) = m_0$$

- dn/dt is computed by fitting a spline and taking derivative under the hood

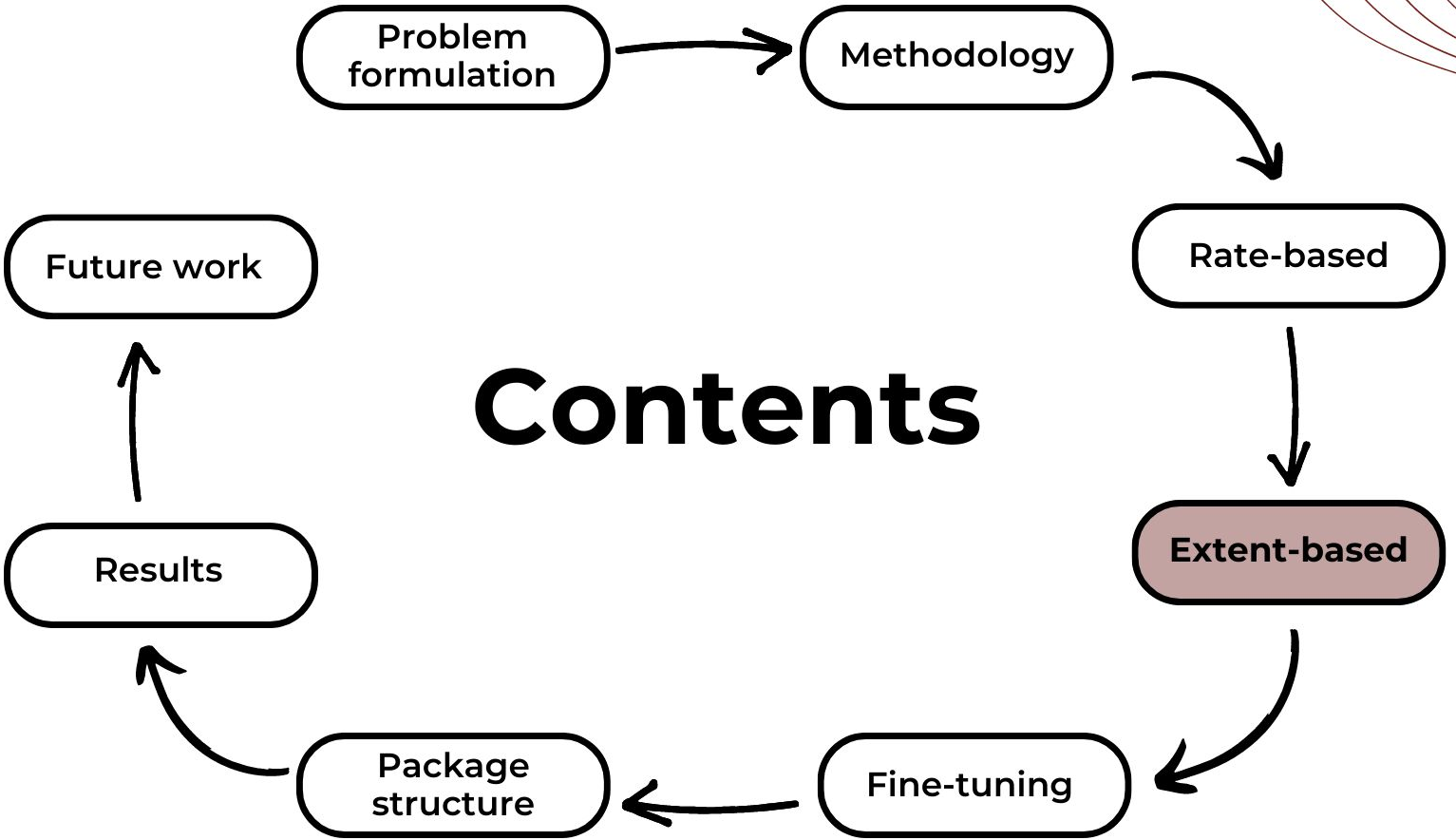


Rate-Based Method

- dn/dt is computed by fitting a smoothing spline and taking derivative under the hood
- Cubic smoothing spline estimate f, is defined as the minimizer of the penalized criterion:
  - $$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int_a^b \left(f''(x)\right)^2 dx$$

- f here is a cubic spline
- λ is smoothing parameter.
  - Controls the tradeoff between bias and variance of f.
  - λ = 0 is exact fit, λ -> ∞ is equivalent to linear regression
- λ is selected via **Cross Validation**
  - 70-30 split. Since it is a time series data, first and last element are ensured to be in train data

Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Extent-Based Method

- Transformation:

$$\mathbf{n} \rightarrow \begin{bmatrix} \mathbf{x_r} \\ \mathbf{x_{in}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{S_0^T} \\ \mathbf{M_0^T} \\ \mathbf{q_0^T} \end{bmatrix} \mathbf{n}$$

- Extent-based objective function:

$$\hat{\theta}_i = \min \Sigma_{h=0}^{H} (x_{r,i}(t_h) - x_{r,i}(\theta_i, t_h))^2$$

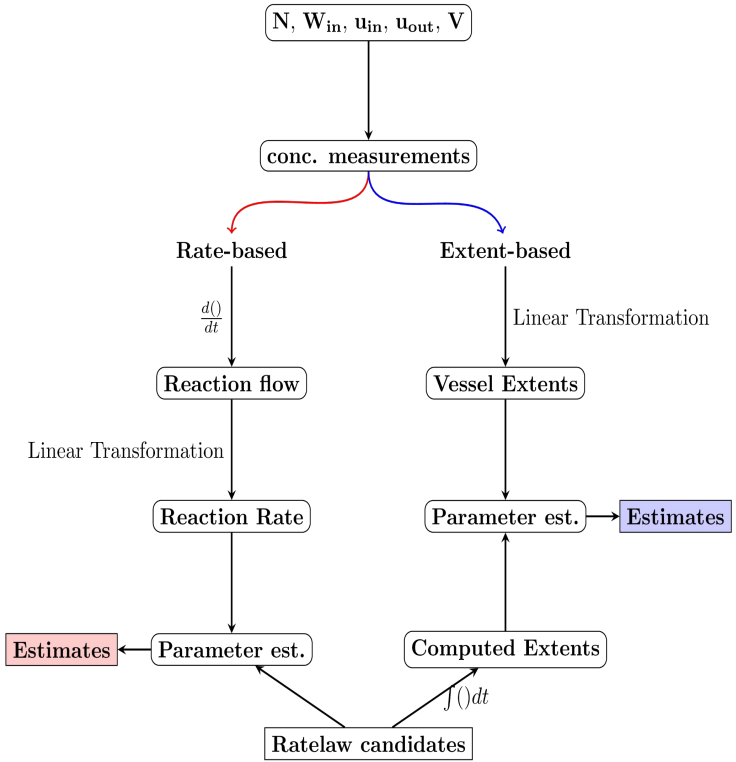
$$\text{s.t. } \mathbf{x}_r(t_h) = (S_0^T)\mathbf{n}(t_h); \quad S_0 \text{ through P3D alg.}$$

$$\dot{x}_{r,i}(\theta_i, t) = r_i(\mathbf{c}(t), \theta_i)V(t) - \frac{u_{out}(t)}{m(t)} x_{r,i}(\theta_i, t) \quad (1)$$

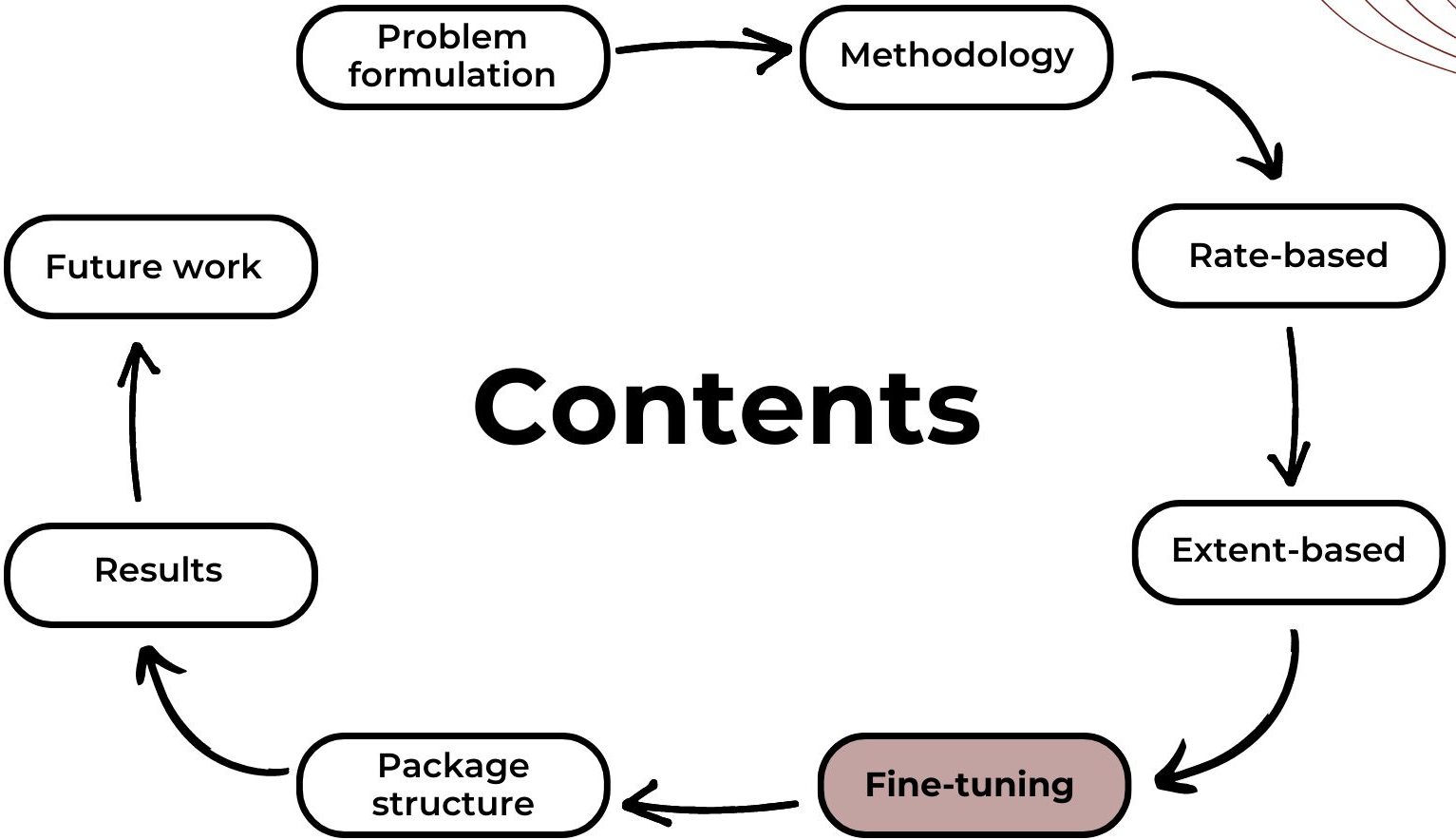
$$m(t_h) = \mathbf{1}_S^T \mathbf{M}_W \mathbf{n}(t_h)$$

$$\text{(or)} \quad \dot{m} = \mathbf{1}_p^T \mathbf{u}_{in} - u_{out}, \quad m(0) = m_0$$

- x_r can be interpreted as **extent of reaction**
- x_{in} can be interpreted as **extent of inlet flow**
- x_{out} =(1-λ) can be interpreted as **extent of outflow**



Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Fine-tuning

- Done using simultaneous approach
- Simultaneous objective function:

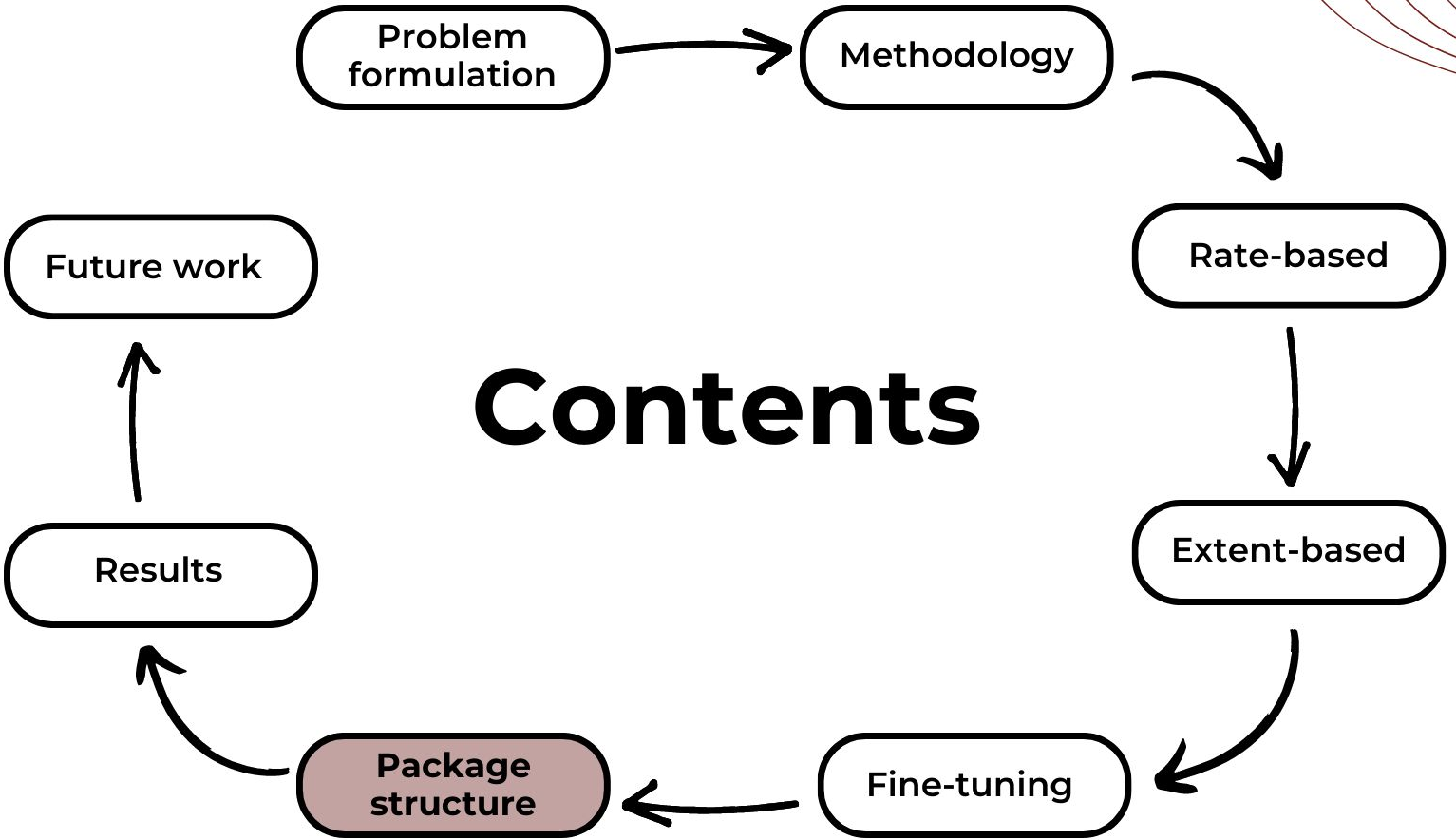
$$\sum_{i=1}^{R}\sum_{j=1}^{j=T_H}\left((C_{ij} - C(\theta_i, t_j))^2\right)$$

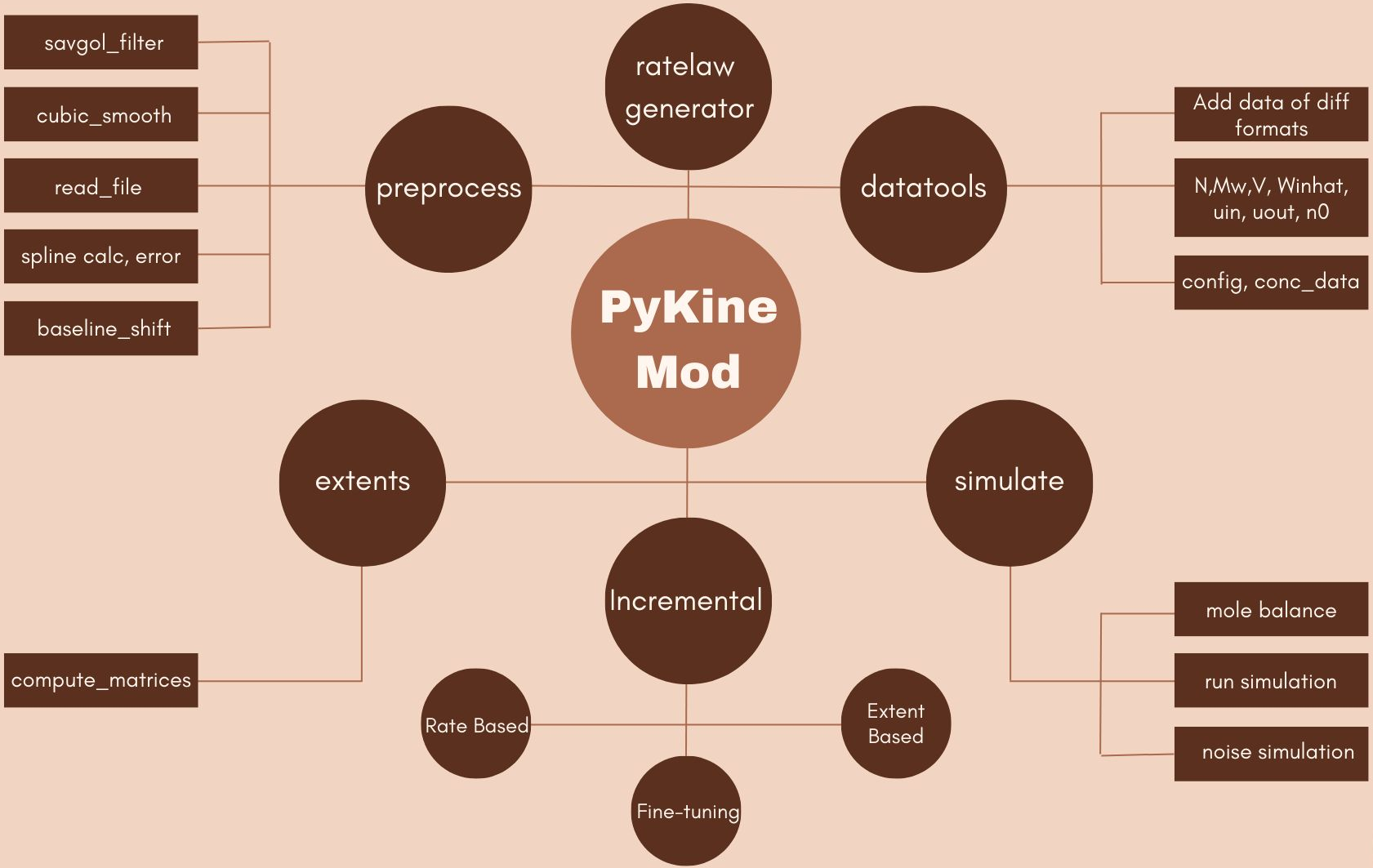
$$\dot{n}(t) = N^T V(t)\mathbf{r}(t) + \mathbf{W_{in}}\mathbf{u_{in}}(t) - \frac{u_{out}(t)}{m(t)}\mathbf{n}(t),\ \mathbf{n}(0) = \mathbf{n_0}$$

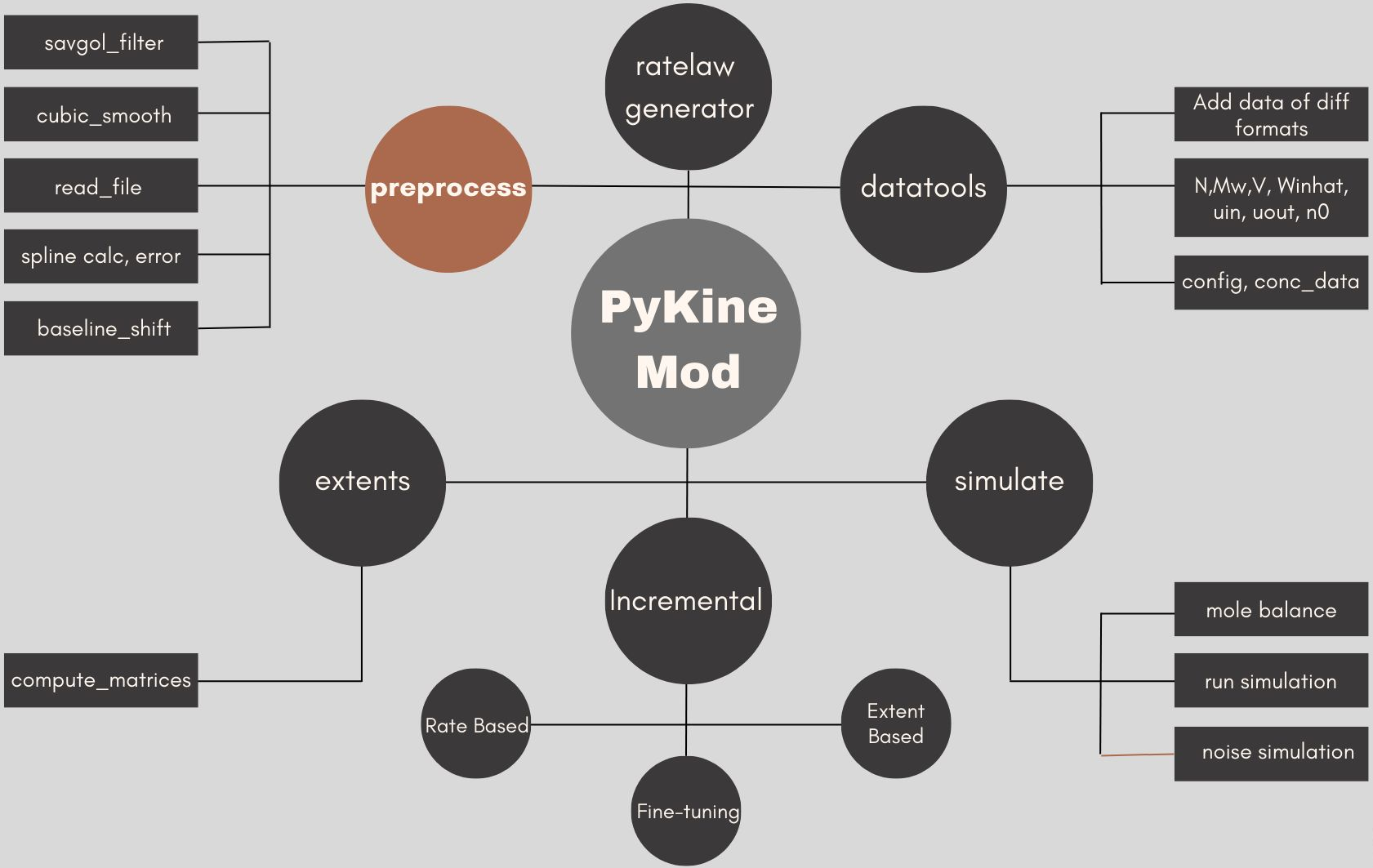
- By choosing different combinations of ratelaw, C(θ,t) is obtained simultaneously. Minimising with respect to the parameters, the estimates are obtained.
- The combination of ratelaw for which the loss is minimum is chosen the best
- Initial guess is given from incremental identification method.
- Only top contenders are chosen for this.
- Much faster and efficient

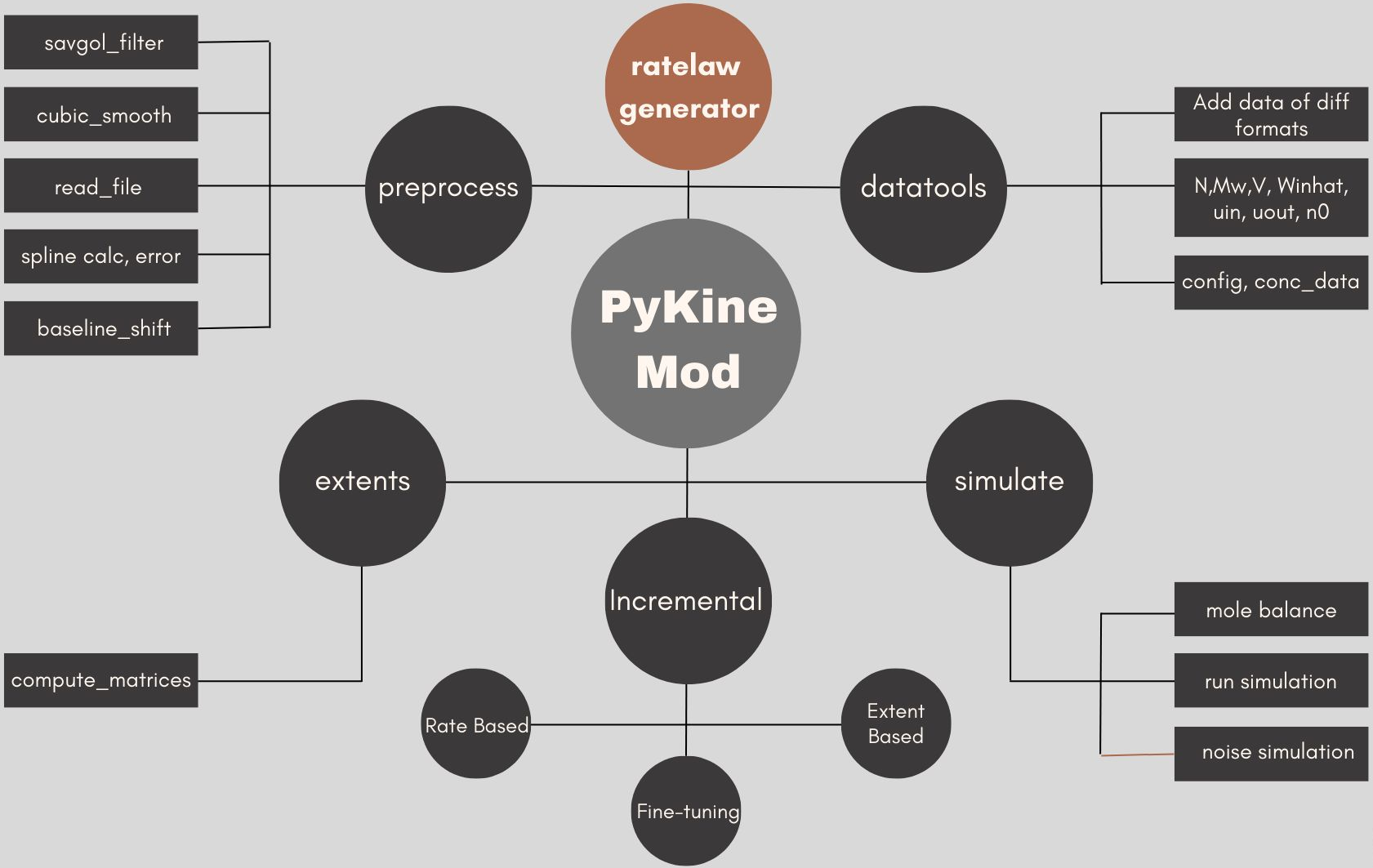
- Statistically optimal estimates
- Final model tuning does not require much computational effort because good initial guesses are available.

Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work


PyKine Mod
ratelaw generator
preprocess
savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
extents
compute_matrices
simulate
mole balance
run simulation
noise simulation
Incremental
Rate Based
Extent Based
Fine-tuning


savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
preprocess
ratelaw generator
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
PyKine Mod
extents
simulate
compute_matrices
Incremental
Rate Based
Extent Based
Fine-tuning
mole balance
run simulation
noise simulation


savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
preprocess
ratelaw generator
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
PyKine Mod
extents
simulate
compute_matrices
Incremental
Rate Based
Extent Based
Fine-tuning
mole balance
run simulation
noise simulation

Analyses the stoichiometric matrix, and generates ratelaw depending on type. If type is not mentioned, it tries a variety of options.

PL_gen (N,d,idx]):

- Uses recursion and hashmap to optimize
- Generates (N[0]+1)(N[1]+1).. size functions
- Stores these functions in dictionary
- These functions are of the power law form

Class rateLaw:

Attributes:

- Type
- Expression (str)
- Function

Class candidateRL:

Attributes:

- Stoic. List
- Type [Opt]

Generating rate law candidates:

: argument N: Stoichiometric Matrix
: argument type [optional]: Type of reaction:

- Reversible/Irreversible as of now

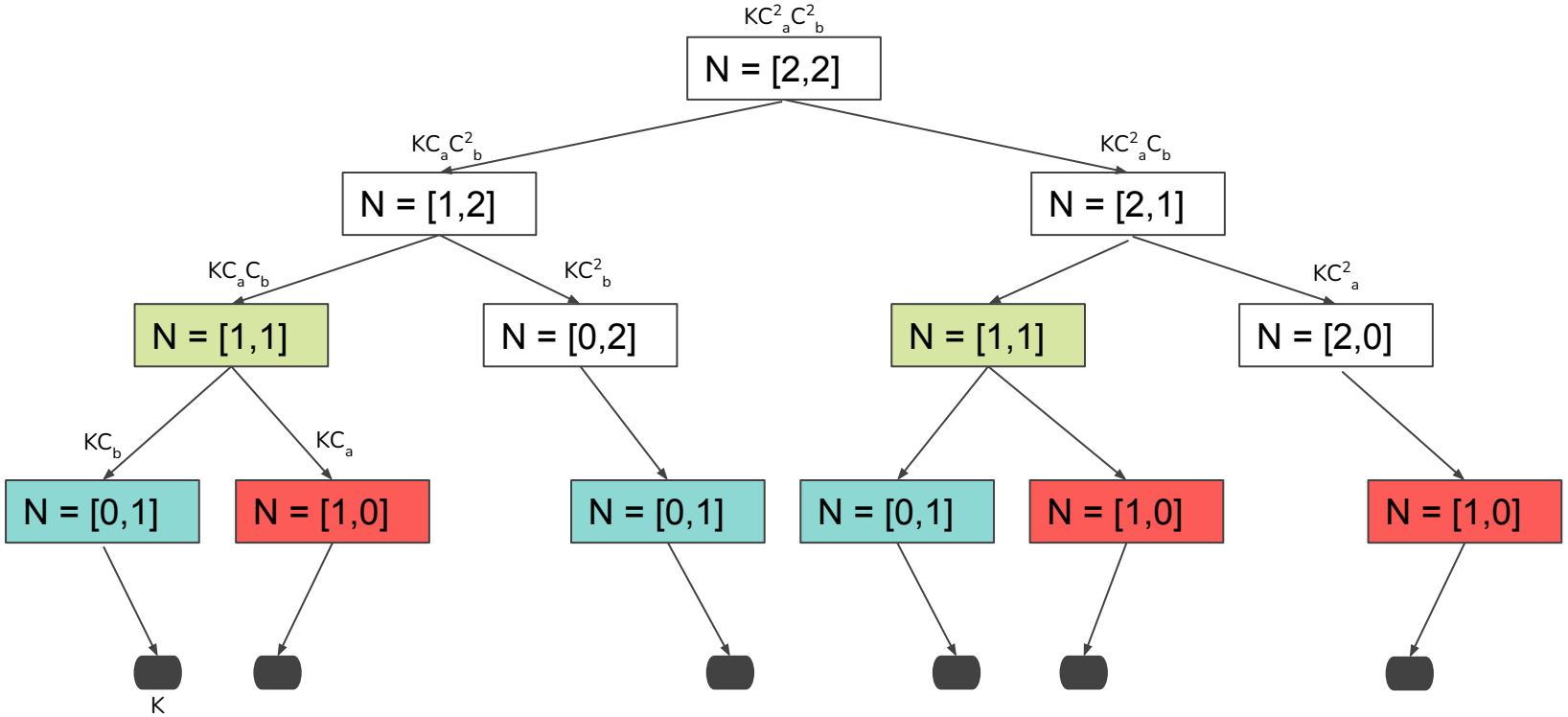
: returns a list of rate expressions (of class ratelaw):
Eg:
[RateLaw(type = "Irreversible", Expression = "K[0]")
*RateLaw(type = "Irreversible", Expression = "K[0]*C[1]^1")*
*RateLaw(type = "Reversible", Expression = "K[0]*C[0]^1- K[1]*C[3]^1")]*

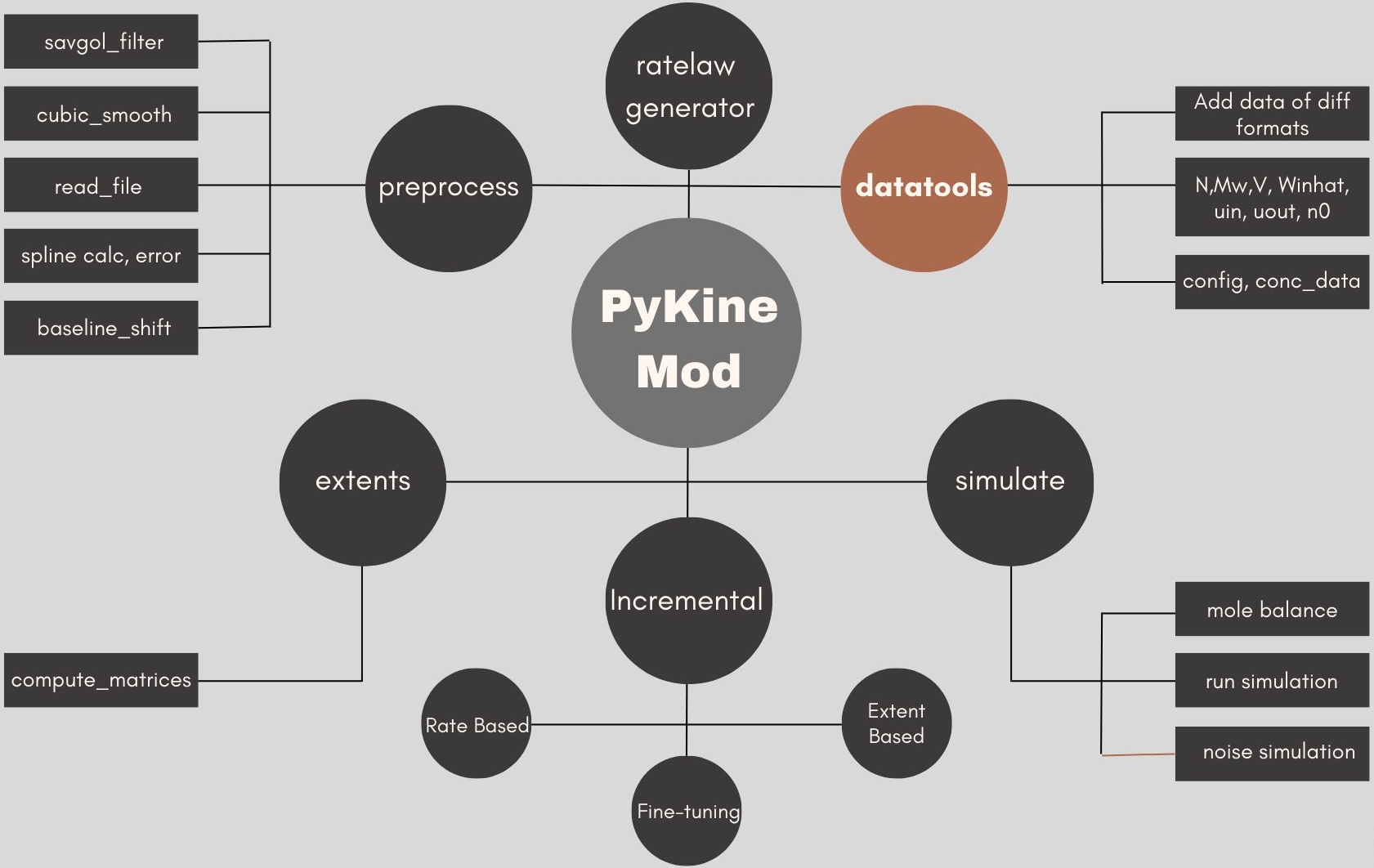
User can view the ratelaw and pickup the essential one appropriately. The object supports slicing, array like access.

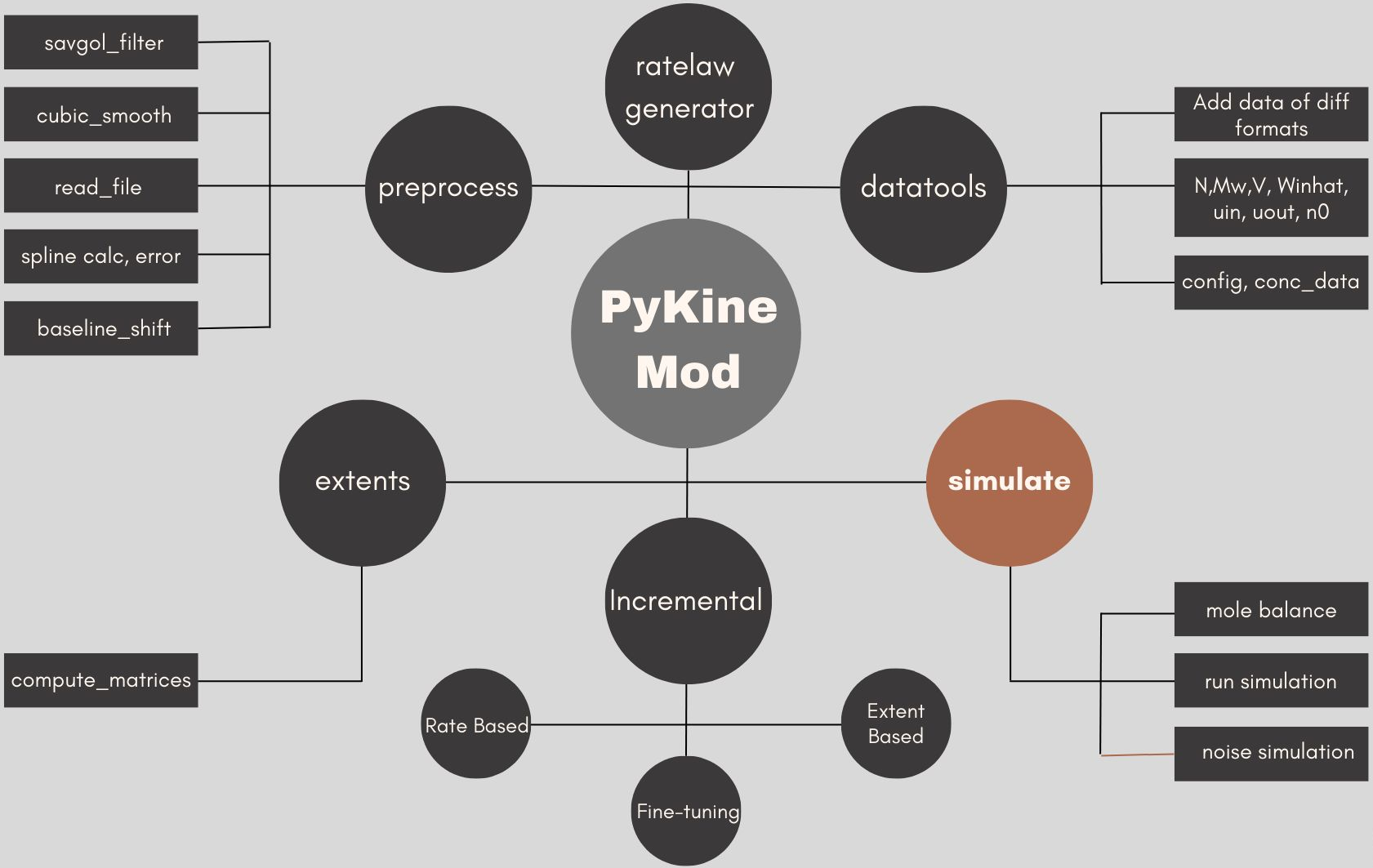
If N = [-a, -b, c, d]

Candidate Rate laws could be:
K1*(Ca^x)*(Cb^y) -
K2*(Cc^z)*(Cd^w)
form with

x<=a, y<=b, z<=c, w<=d

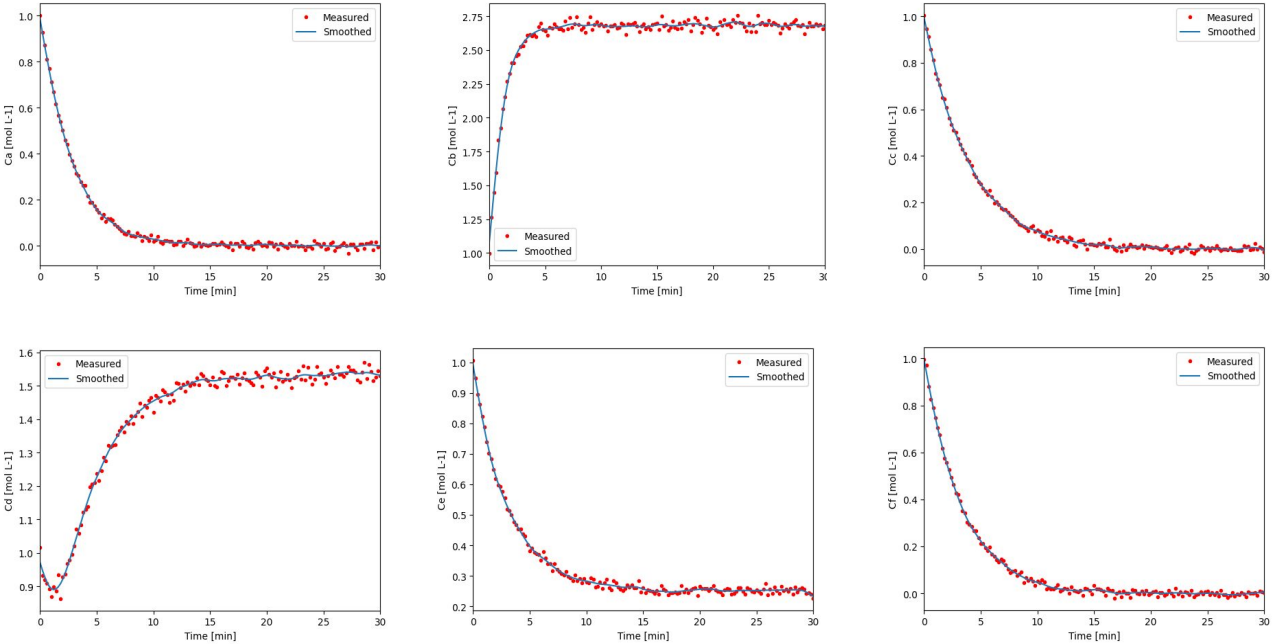

$KC^2_aC^2_b$
N = [2,2]
$KC_aC^2_b$
$KC^2_aC_b$
N = [1,2]
N = [2,1]
KC_aC_b
KC^2_b
KC^2_a
N = [1,1]
N = [0,2]
N = [1,1]
N = [2,0]
KC_b
KC_a
N = [0,1]
N = [1,0]
N = [0,1]
N = [0,1]
N = [1,0]
N = [1,0]
K


savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
preprocess
ratelaw generator
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
PyKine Mod
extents
compute_matrices
Incremental
Rate Based
Extent Based
Fine-tuning
simulate
mole balance
run simulation
noise simulation


savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
preprocess
ratelaw generator
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
PyKine Mod
extents
compute_matrices
simulate
Incremental
Rate Based
Extent Based
Fine-tuning
mole balance
run simulation
noise simulation

Simulation of Acetoacetylation of Pyrrole

$$\text{Measured} = \text{Simulated} + WN \sim N(0,\sigma^2)\ ;\ \sigma_s = (\alpha/100)c_s^{max}$$

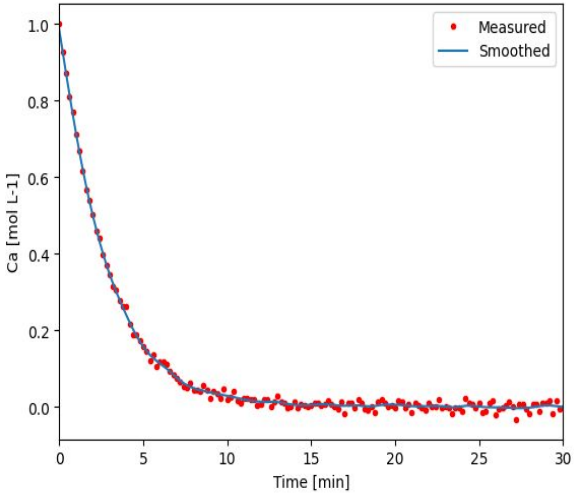


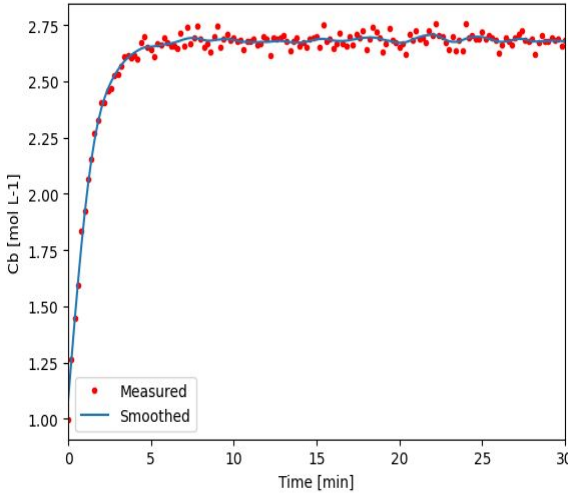
A + B -> C
B + B -> D
B -> E

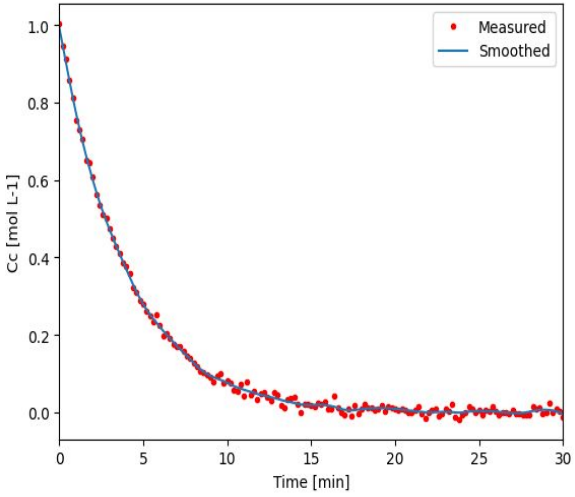
- A: Pyrrole
- B: Diketene
- C: acetoacetyl pyrrole,
- D = Dehydroacetic acid
- E is oligomer

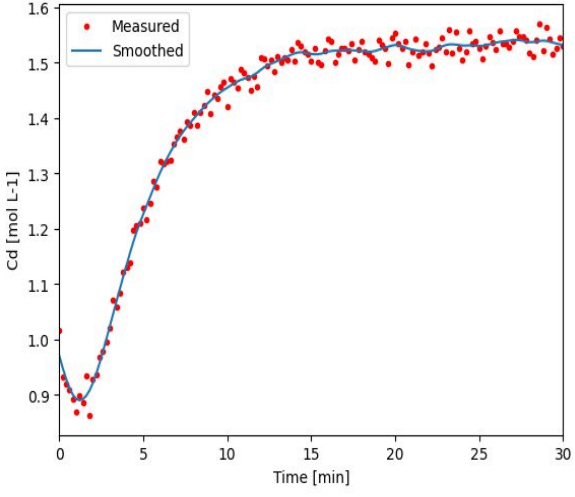
Simulation:

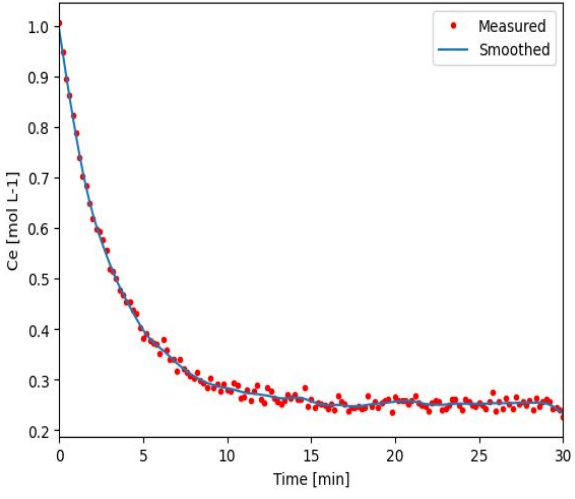
- 151 data points
- α is 1%

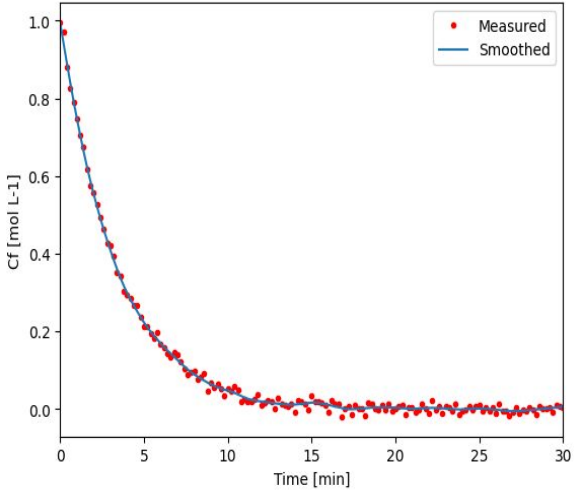

Measured
Smoothed
Ca [mol L-1]
Time [min]

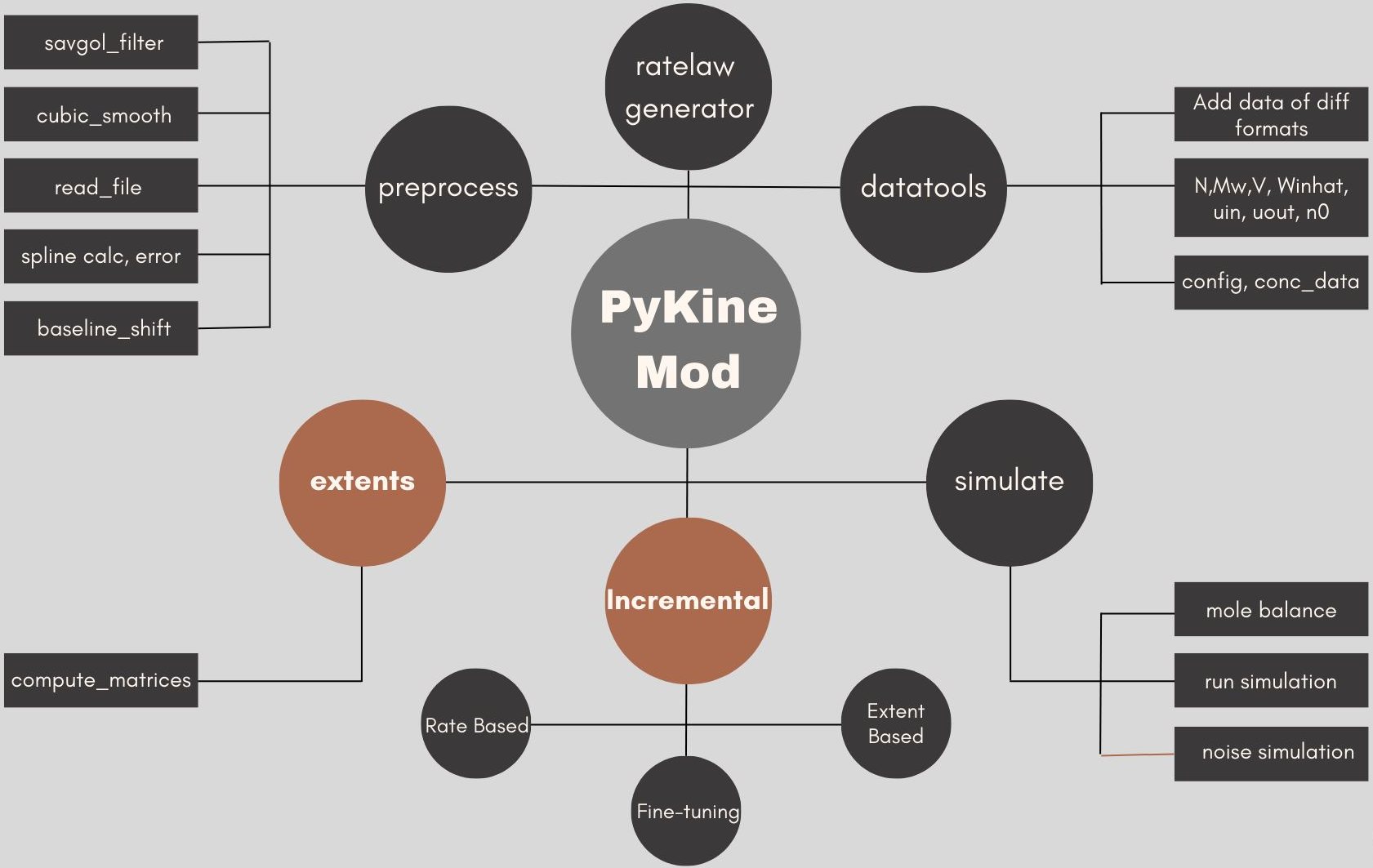

Measured
Smoothed
Cb [mol L-1]
Time [min]


Measured
Smoothed
Cc [mol L-1]
Time [min]


Measured
Smoothed
Cd [mol L-1]
Time [min]


Measured
Smoothed
Ce [mol L-1]
Time [min]


Measured
Smoothed
Cf [mol L-1]
Time [min]


savgol_filter
cubic_smooth
read_file
spline calc, error
baseline_shift
preprocess
ratelaw generator
datatools
Add data of diff formats
N,Mw,V, Winhat, uin, uout, n0
config, conc_data
PyKine Mod
extents
simulate
Incremental
compute_matrices
Rate Based
Extent Based
Fine-tuning
mole balance
run simulation
noise simulation

Package Structure

```
# Importing PyKineMod package
from PyKineMod import Incremental
# Creating Model Object
model = Incremental(N, Mw, V, Winhat, uin, uout, n0)
# Adding concentration data from csv file
model.add_concentration_data("aceto_pyrrole_basecase.csv",
    preprocess = 'baseline_shift')
```

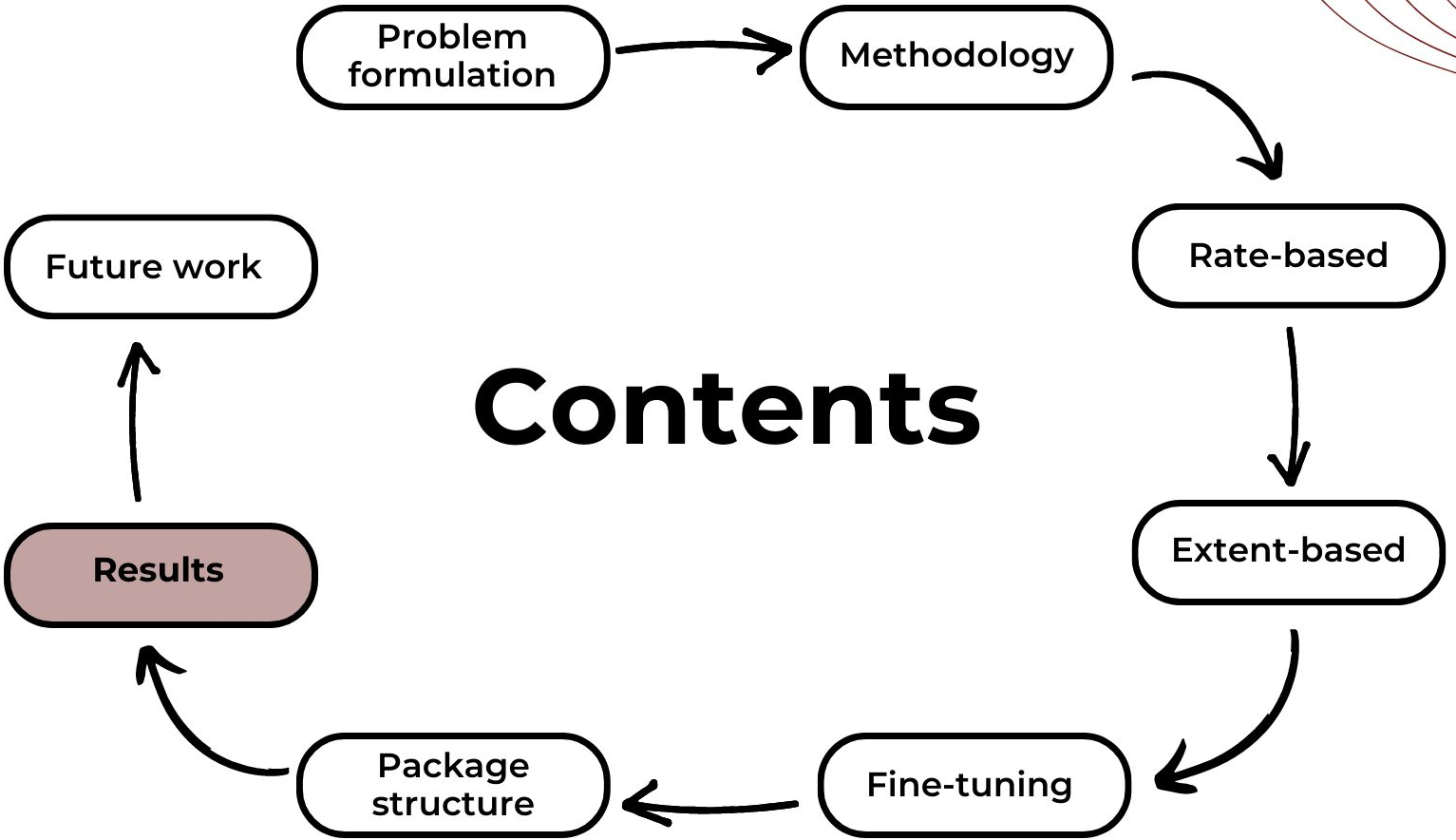
```
# Rate-based Method
res1 = model.estimate_parameters(candidates_list, method = '
    rate_based', conf_int = True, metric = 'aicc', plot =
    True, bootstraps = 1000)
# Extent-based Method
res2 = model.estimate_parameters(candidates_list, method = '
    extent_based', conf_int = True, metric = 'aicc', plot =
    True, bootstraps = 100)
```

```
# Generating Ratelaws from Stoichiometric Matrix
from ratelawgen import CandidateRateLaws
import numpy as np

# N is RxS Matrix
cand1 = CandidateRateLaws(N[0], type = "Irreversible") # For
    First Reaction
cand2 = CandidateRateLaws(N[1], type = "Reversible") # For
    second Reaction
cand3 = CandidateRateLaws(N[2], type = "Irreversible") # For
    Third Reaction

candidates_list = [cand1, cand2, cand3]
```

```
# Pythonic way of creating custom rate laws
def cand_ratelaw11(y,K):
    Ca, Cb, _, _, _, _ = y
    Ck = 0.5
    return K[0]*Cb*Ck
candidates_list = [[cand_ratelaw11]]
```

- By separating into modules, it is easier to review code, work on preprocessing steps effectively, add functionality and debug
- Volume, uin, uout data can now be given as csv, excel, pandas dataframe input
- PyKineMod supports **time variation** in uin, uout, volume data.
- Intermediate times are interpolated to get the values. Options are provided to do different kind of interpolation (linear, nearest, quadratic etc.). Codes are parallelised while doing bootstraps

Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Results

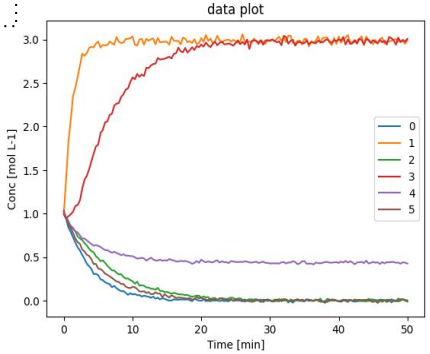
- Scenario 0:
  - Clean data, 151 points
- Scenario 1:
  - 1% noise, 151 points
- Scenario 2:
  - 10% noise, 151 points
- Scenario 3:
  - 1% noise, 21 points
- Scenario 4:
  - A conc is not available

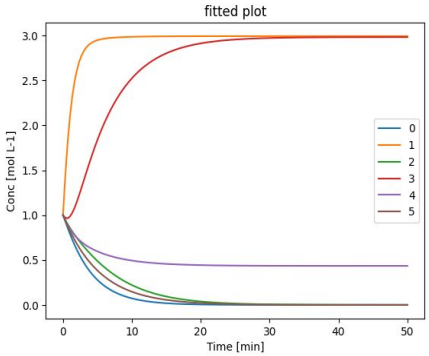
Scenario	Rate par.	RB ident.	RB est.	EB ident.	EB est.	FT ident	FT est.
0 Clean data	k1	True	0.0531	True	0.0529	True	0.0529
	k2	True	0.1280	True	0.1280	True	0.1279
	k3	True	0.0280	True	0.0280	True	0.0280
1 Base case	k1	False	0.0538	True	0.0511	True	0.0518
	k2	True	0.1279	True	0.1276	True	0.1278
	k3	False	0.0279	False	0.0275	True	0.0279
2 High noise	k1	False	0.0618	False	0.0449	True	0.0524
	k2	False	0.1252	True	0.1260	True	0.1251
	k3	False	0.0303	False	0.0298	True	0.0293
3 Fewer points	k1	False	0.0583	False	0.0333	False	0.0333
	k2	False	0.1371	False	0.1384	False	0.1384
	k3	False	0.0311	True	0.0285	True	0.0307

Scenario	Rate par.	EB ident.	EB est.	FT ident	FT est.
4 Missing A	k1	True	0.0547	True	0.0547
	k2	True	0.1278	True	0.1278
	k3	True	0.0279	True	0.0279

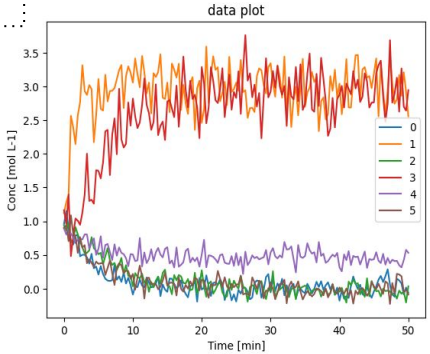
Results

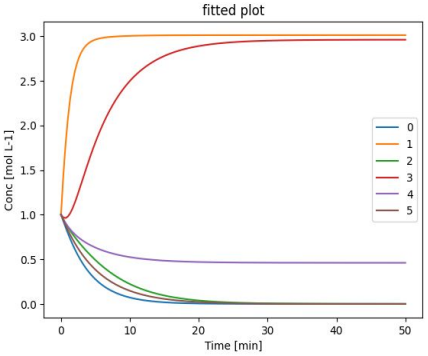
1



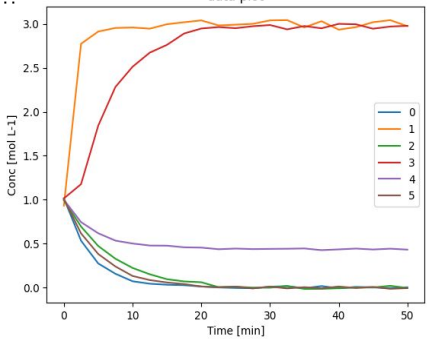


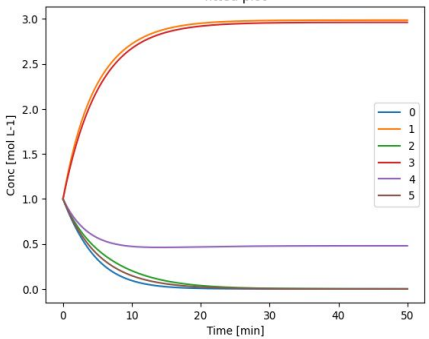
2



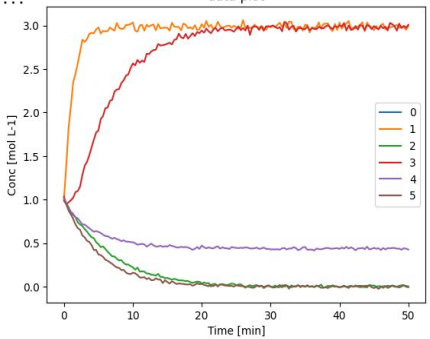


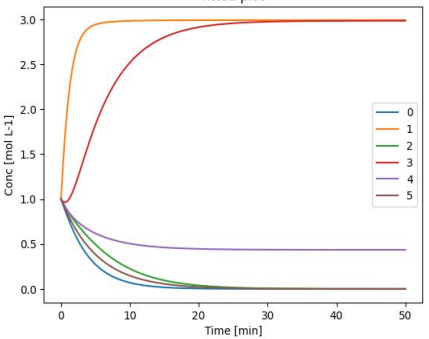
3



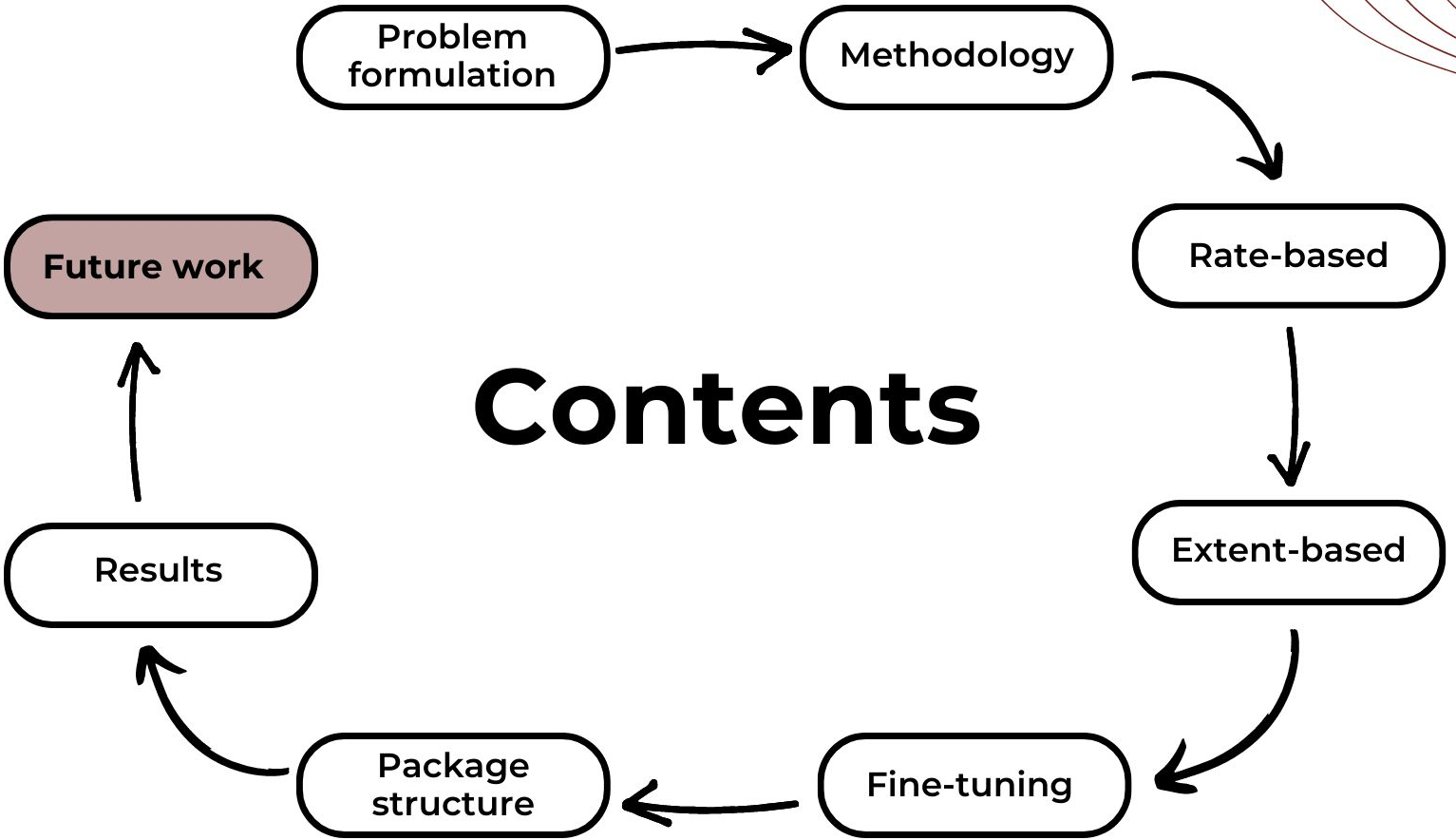


4





Contents


Problem formulation
Methodology
Rate-based
Extent-based
Fine-tuning
Package structure
Results
Future work

Future Work

- Combine kinetic identifiability with spectral data work
- Heuristic to pick top_k while finetuning
- Support for different kind of ratelaw generation
- Support kinetic model identification via density measurements
- Support for more preprocessing techniques
- Support various file formats and input formats in addition to current state
- Use probabilistic techniques to reach global optimum to improve convergence
- Implement bias-corrected (BC or BCa) to calculate confidence intervals to improve estimated CI range
- Support for various libraries for plotting options (currently uses plotly)
- Support for exporting results
- Creating an API to embed in a GUI

Thank you