

# GRADUATE STUDENT SOFTWARE PROJECT: KINETICS KALCULATOR

Friday, December 6<sup>th</sup>, 2024  
Nathaniel Corley

# BACKGROUND — ENZYME KINETICS EXPERIMENTS

In experiment contexts, researchers often want to understand the kinetics of an enzyme-catalyzed reaction.

The most common method to evaluate the **activity** of an enzyme is by conducting so-called “kinetics experiments”, where wet lab scientists empirically determine **Michaelis Menton (MM) constants** that describe catalyzed conversion of substrate to products

But these experiments are **burdensome** — not only from a wet lab perspective, but also from a computational perspective — and error-prone

$$\dot{P} = k_{cat} \frac{E_T (S_T - P)}{K_M + S_T - P},$$

*Michaelis Menton Equation*

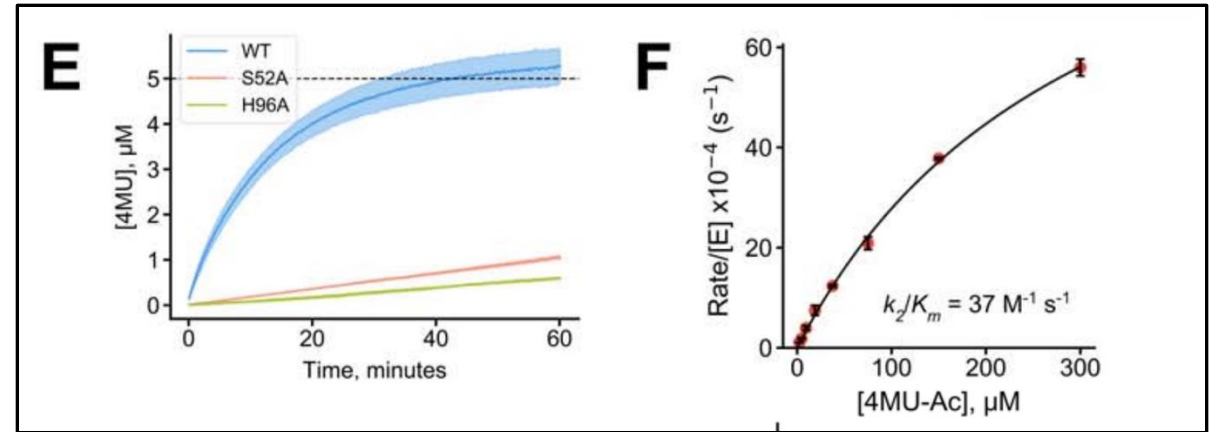
## Sources

Choi, B., Rempala, G.A. & Kim, J.K. Beyond the Michaelis-Menten equation: Accurate and efficient estimation of enzyme kinetic parameters. *Sci Rep* 7, 17018 (2017). <https://doi.org/10.1038/s41598-017-17072-z>

# BACKGROUND

Data cleaning, background noise reduction, curve fitting, and many other steps aren't automated and are often done in ad-hoc Jupyter notebooks

Creation of simple figures (e.g., the examples “E” and “F” shown to the right) often **take far longer** than necessary



Example kinetics figures from *Computational Design of Serine Hydrolases*

## Sources

Anna Lauko, Samuel J. Pellock, Ivan Anischanka, Kiera H. Sumida, David Juergens, Woody Ahern, Alex Shida, Andrew Hunt, Indrek Kalvet, Christoffer Norn, Ian R. Humphreys, Cooper Jamieson, Alex Kang, Evans Brackenbrough, Asim K. Bera, Banumathi Sankaran, K. N. Houk, and David Baker, "Computational Design of Serine Hydrolases," bioRxiv (2024), <https://doi.org/10.1101/2024.08.29.610411>.

# RESEARCH QUESTION / PROBLEM STATEMENT

How can we build a **general-purpose toolkit** that enzyme experimentalists who are comfortable in Jupyter Notebooks can use to accelerate their enzyme kinetic analysis workflow?

Based on conversations with multiple experimentalists, this software toolkit should be:



**Opt-in;** e.g., users can start analysis and any part of the pipeline, rather than requiring them to run the whole thing



**Customizable** to any enzymatic reaction



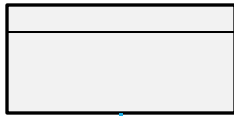
**Quick, clear, and efficient,** where users who are comfortable with basic python can easily access the API to generate the analysis and plots that they need

## Sources

Conversations with Saman Salike and Andrew Hunt, 2024

# USE CASES & METHODOLOGY

**Target User:** Experimentalist who is comfortable with Jupyter notebooks Python libraries.



Structured data (according to specified *data dictionary*)

- **Label experimental conditions/replicates** (e.g., if multiple cells with the same conditions)
- **Convert absorbance values to concentrations** using a “standard curve” (*calibrated to that instrument*)
- **Plot concentration vs. time** for each condition and **subset** to a well-behaved time interval
- **Calculate initial rates** for each condition by fitting a linear model; validate results
- **Adjust rates for background activity** (by subtracting out “negative control” cell values)
- **Plot Michaelis Menton curve** and calculate corresponding kinetic constants

# DESIGN

Take as input a **dataframe of experimental outputs**, structured according to a pre-defined data dictionary

well	value	serial_number	plate_number	sample_number
I1	0.328341	21062324	1	1
I1	0.335198	21062324	1	1
I1	0.342055	21062324	1	1
I1	0.349892	21062324	1	1
I1	0.357729	21062324	1	1

**Initialize class** that maintains the dataframe as state, with class methods that perform relevant calculations

```
class KineticsKCalculator:
    def __init__(
        self,
        data_path: Path | PathLike,
        standard_curve_parameters: dict | None = None,
    ):
        """
        Initialize the KineticsKCalculator by loading the data from the specified file path.

        Args:
            data_path (Path | PathLike): The path to the data file to load. Supported file formats include:
                - CSV: .csv
                - Excel: .xls, .xlsx
                - JSON: .json
                - Parquet: .parquet
                - Feather: .feather
                - HDF5: .hdf
                - Pickle: .pkl
            standard_curve_parameters (dict | None): A dictionary containing the parameters of the standard curve.
                For example, {"slope": 2, "y_intercept": 1}, if using a linear standard curve of the form  $y = mx + c$ .
            NOTE: Currently only linear standard curves are supported.
            epsilon (float): A small value to add to the negative control values to avoid division by zero.

        Returns:
            None: The KineticsKCalculator object is initialized with the loaded data.
        """
        # Convert to a Path, if necessary
        self.data_path = Path(data_path)
```

**Interact** with the library via a Jupyter Notebook

```
calculator = KineticsKCalculator(
    data_path=kinetics_data_path,
    standard_curve_parameters=experiment_specific_standard_curve,
)
```

(DEMO)

# PROJECT STRUCTURE



**ruff** formatting (automatic linting)



**sphinx** documentation (generated from docstrings)



**pip**-installable via `pyproject.toml`  
(vs. legacy `setup.py`)



comprehensive test coverage with  
**pytest**

Nathaniel Corley and Nathaniel Corley feat: working full-featured version prior to docum... 0960276 · 1 hour ago 4 Commits		
docs	docs: component and functional specifications	3 weeks ago
examples	feat: working full-featured version prior to documentation	1 hour ago
kinetics_kalculator	feat: working full-featured version prior to documentation	1 hour ago
tests	feat: working full-featured version prior to documentation	1 hour ago
.DS_Store	feat: working full-featured version prior to documentation	1 hour ago
.gitignore	initial commit	2 months ago
LICENSE	initial commit	2 months ago
Makefile	chore: repository setup	2 months ago
README.md	initial commit	2 months ago
__init__.py	feat: working full-featured version prior to documentation	1 hour ago
pyproject.toml	feat: working full-featured version prior to documentation	1 hour ago

### Kinetic Calculator

CONTENTS:

- kinetics\_kalculator
  - kinetics\_kalculator package
    - Submodules
      - kinetics\_kalculator.kinetics\_kalculator module
      - kinetics\_kalculator.utils module
        - `add_rate_column()`
        - `adjust_rates_for_background()`
        - `calculate_michaelis_menten_constants()`
        - `convert_to_concentration_using_linear_standards()`
        - `filter_by_time_range()`
        - `fit_line()`

Module contents

kinetics\_kalculator / kinetics\_kalculator package [View page source](#)

## kinetics\_kalculator package

### Submodules

#### kinetics\_kalculator.kinetics\_kalculator module

```
class kinetics_kalculator.kinetics_kalculator.KineticsCalculator(data_path: Path | PathLike | DataFrame | None, standard_curve_parameters: dict | None = None) [source]
```

Bases: `object`

```
adjust_rates_for_background(rate_column: str, sample_type_column: str = 'sample_type', negative_control: str = 'negative_control', remove_negative_controls: bool = True) [source]
```

Adjust the rates in the DataFrame by subtracting the provided background value. Adds a new column containing the adjusted rates, `{rate_column}_minus_background`, to the DataFrame.

Parameters:

- `rate_column` (`str`) – The name of the column containing the rates to adjust.
- `sample_type_column` (`str`) – The name of the column containing the sample type information. Defaults to `"sample_type"`.
- `negative_control` (`str`) – The value to use as the negative control. Defaults to `"negative_control"`.
- `remove_negative_controls` (`bool`) – Whether to remove the negative control rows after adjusting the rates. Defaults to `True`.

Returns: The DataFrame is modified in-place.

Return type: `None`



# CHALLENGES/ISSUES

## Lessons learned and Challenges

**What I as a programmer thought would be useful wasn't always useful.** For example, I initially coded an automatically pipeline that ran every step sequentially. But it turns out, the experimentalists who I talked to wanted to execute the pipeline step-by-step, visualizing outputs, to ensure everything looked right

**Building general pipelines that don't depend strictly on the previous steps is hard.** It's easy enough to force users to confine to a particular path, but flexibility requires more forethought in terms of structure and design.

---

## Future directions

Support for **more nuanced kinetics models**, such as Michaelis Menton kinetics with the Hill coefficient

**Directly parse experimental outputs**, rather than requiring users to first format their data into a well-defined dataframe

Structure code as a **functional API** if users prefer non-object-oriented programming