Master Project

Development of algorithms for the analysis of experimental data obtained by equilibrium and time-resolved absorption spectroscopy

–

Author
Mario Asensio Franco

Supervisors
Milagros Medina Trullenque
José Carlos Ciria Cosculluela

Master in Biophysics and Quantitative Biotechnology
Year 2024-2025

INDEX

INDEX

INDEX

1. Abstract

Proteins in general, and enzymes in particular, are indispensable for supporting the development and reproduction of all forms of living beings. In addition to their key roles as biomolecules of life, scientists also investigate how to enhance their potential in biotechnological synthesis, bioremediation, or food industries as well as their biomedical applicability. Nonetheless, their rational use for those applications can only happen after a deep knowledge about their interplay with other molecules and the reactions occurring as a consequence of such interactions.

In this line, the main objective of this project is to develop software capable of analyzing experimental data generated by differential spectroscopy and multiwavelength time-resolved absorption spectroscopy using stopped-flow. Differential spectroscopy is useful for studying proteins because it captures subtle changes in absorbance of particular chromophores due to structural or environmental changes, allowing monitoring ligand binding and conformational changes. Multiwavelength time-resolved absorption spectroscopy using stopped-flow provides real-time kinetic data across multiple wavelengths, enabling researchers to study rapid reactions, identify intermediates and understand processes such as enzyme catalysis and ligand interactions. They offer information about protein/enzyme behavior and particular reaction mechanisms.

Currently, most of the software used to analyze results obtained from these techniques is mainly proprietary. This poses some issues: there is a license fee (which hinders their use in teaching), it is linked to the equipment manufacturer (generates dependence), the software is not open-source (it cannot be adapted to particular or evolving necessities) and tend to be not user-friendly.

Therefore, as an open-source alternative, three different software were here developed: DAIPProLi (Determination of Affinity of Interaction Parameters between Protein and Ligand), KiPaD (Kinetic Parameters Determination) and AKiPa (Analysis of Kinetic Parameters). DAIPProLi is aimed to analyze data from differential spectroscopy, whereas KiPaD and AKiPa are both aimed to analyze data from multiwavelength time-resolved absorption spectroscopy using stopped-flow. They are openly accessible to the community through the unizar-flav GitHub repository (https://github.com/unizar-flav).

All these tools display similar performance to that of proprietary software and solve the issues aforementioned. Furthermore, as they are open-source software, they allow students and young researchers to familiarize themselves with the numerical methods of analysis employed in the particular studied methodologies. The software produced here will be immediately used in practicums for students in the Informatics and Biophysics courses of the Biotechnology degree program at the University of Zaragoza in the coming years.

2. Introduction

2.1 Thermodynamic and kinetic parameters for processes involving proteins and enzymes

Every aspect of living organisms is ruled by biochemical reactions, in which proteins and enzymes play major roles through the interaction with other biomolecules. To understand how proteins and enzymes perform their functions, researchers usually evaluate their *in vitro* ability to interact with particular metabolites (including small molecules, but also other proteins, nucleic acids or lipids). Researchers also analyze reactive processes, including biochemical or catalytic reactions, with the aim of determining the thermodynamic and kinetic parameters describing such processes. In particular, interaction parameters determined under equilibrium situations, such as dissociation and association constants (respectively, K_d and K_a), provide relevant information regarding the affinity of a ligand and a particular receptor. On the other hand, pre-steady state kinetics measurements allow researchers to determine kinetic parameters in the form of "rate constants" for the interconversion of species along a reactive process, as well as to obtain information about particular properties of intermediate species produced during the reaction. Therefore, research to determine interactions involving proteins has increasingly gained interest in the scientific community, being sustained by nearly one million published articles on the topic (**Figure 1**).

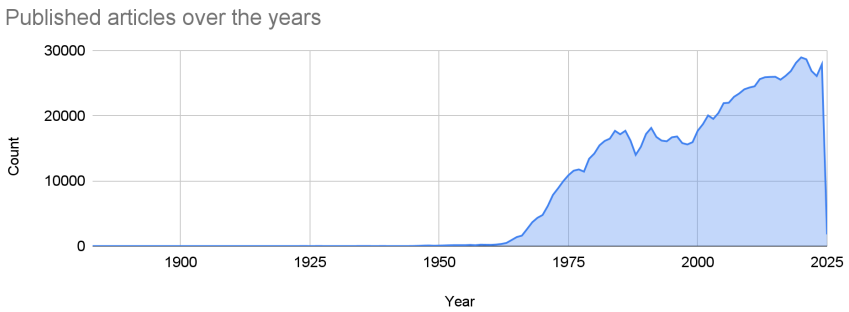


Figure 1: Number of articles available on PubMed over the years as retrieved when using the following keywords: protein & thermodynamic | kinetic. It shows the rising interest of the scientific community on these topics. Source: PubMed search engine, accessed January 13, 2025.

Absorption spectroscopy provides a straightforward method for qualitatively analyzing changes in the absorption of particular chromophores when their spectroscopic properties become altered due to changes in their environment (as for example upon interaction with another molecule) or to the conversion into another compound (as for example in a reaction where they get transformed into a different absorbing species or when they suffer an oxido-reduction process) (Martínez-Júlvez et al., 2001; Medina et al., 2001; Pérez-Amigot et al., 2019). Due to the universality and accessibility of absorption spectroscopy, these methods are widely used in the evaluation of such interactions and kinetic parameters. Usually, these methods generate extensive sets of spectra for multiple experimental conditions and replicates, rapidly producing large datasets that must be evaluated. The complexity is further magnified by the potential need to evaluate various models for each potential process, making manual data analysis challenging and time-consuming even with the aid of spreadsheets. Most of these evaluations also require non-linear fit analysis which require specialized software mostly provided by profit companies, in many cases related to a particular experimental set up. Typically, they are licensed on an individual basis. Moreover, they promote a captive market: their use for processing data obtained with equipment from different companies is not warranted.

Therefore, most students and new scientists in the field have limited access to such software tools, in a world where the generated data keeps increasing and needing for analysis. Finally, those tools are in many cases “dark boxes”, the mathematical framework supporting them is far from transparent, and prevents the user from understanding the algorithm, optimizing, generalizing and adapting it to their evolving needs.

The programs developed in this project allow analyzing data obtained by techniques such as differential spectroscopy and stopped-flow spectrophotometry. The following sections provide a quick introduction to such techniques.

2.2 Differential spectroscopy experiments

Differential spectroscopy is a powerful technique for quantitatively analyzing spectroscopic changes in the absorbance spectrum of an absorbing protein or ligand. When a receptor (for example the protein) with a characteristic absorbance spectrum binds to a ligand, this spectrum is susceptible to undergo subtle changes that can be magnified by recording the difference spectra between the complex and free protein (Bunaciu et al., 2013).

This technique can be used in equilibrium studies to compare the absorbance of a receptor solution under different conditions, such as varying ligand concentrations. The method allows researchers to track changes in protein conformation, stability, and interactions, as they relate to equilibrium parameters, such as association (K_a) or dissociation (K_d) constants (Medina et al., 2001; Piubelli et al., 2000; Sancho and Gómez-Moreno, 1991).

A differential spectrum is the difference between two absorbance spectra. There are two ways of obtaining them: Indirectly, by subtracting mathematically one absolute spectrum to another. Directly, by using a spectrophotometer with double beam, in which one of the compounds is added to the reference cuvette and the other compound in the sample cuvette.

Differential spectroscopy has the advantage of detecting very small changes in absorbance in systems with a great amount of background absorbance. A differential spectrum has the following characteristics (Medina et al., 1998):

1. Negative absorbance values can appear.
2. The maxima and minima may be shifted, and the extinction coefficients differ from those of the absolute absorption peaks.
3. There are points of zero absorbance which correspond to the wavelengths at which both absolute spectra absorb the same. These wavelengths are called isobestic points (Chemistry (IUPAC), n.d.).

For differential spectroscopy to be used, distinct spectroscopic properties must exist between the free and bound states of the receptor. Moreover, the ligand should ideally present different spectroscopic properties regarding the receptor, so it does not interfere with the measurements. In this ideal scenario, the spectroscopic signal (Y) will be proportional to the concentration of the receptor-ligand complex (RL) in solution:

$$Y \propto [RL]$$

Spectra are compared before and after perturbations to observe how the absorption of the chromophore changes as the environment shifts. By analyzing these differences, it is possible to determine equilibrium interaction parameters.

One approach to determine the binding affinity between a protein and a ligand is direct titration, where the concentration of one species remains more or less constant (**receptor**), while the other species (**ligand**) concentration is gradually increased. Note that strictly speaking, the concentration of the receptor does not actually remain constant (it gets diluted as titration progresses), but experiments are usually designed to minimize this dilution. By monitoring changes in the absorbance spectrum, populations of free receptor, free ligand, and receptor-ligand complexes can be tracked. It is important to highlight that the **receptor** is typically the molecule that absorbs in the studied wavelength range, and can be either the protein or the ligand.

One application of this method is the determination of the K_d of the Apomyoglobin-heme complex. The myoglobin is the protein present in the cardiac and skeletal muscle and its function is to facilitate oxygen diffusion from the blood towards the muscle tissue. This activity of the myoglobin revolves around its prosthetic group the heme molecule, which is non-covalently bound, and is the one in charge of binding the oxygen. Since the heme group is not covalently bound, the myoglobin has two states apomyoglobin, in which the heme group is not present, and the holomyoglobin, which has its heme group non-covalently bound (Vanek and Kohli, 2024). As such, we want to measure the dissociation constant that rules the equilibrium between these two states (Gómez-Moreno Sanz and Sancho Sanz, 2003, pp. 463–468).

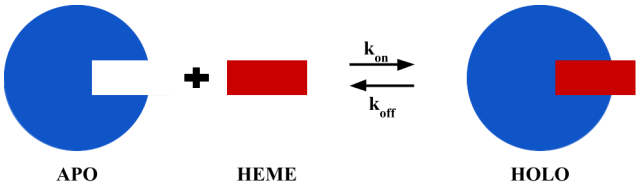


Figure 2: Binding equilibrium between the **apomyoglobin (APO)** and the **heme group (HEME)** to form the complex **holomyoglobin (HOLO)**.

This association model can be written as shown in **Figure 2**,where the association reaction rate is $v_a = k_{on}[APO][HEME]$, and the dissociation reaction rate is $v_b = k_{off}[HOLO]$ (k_{on} and k_{off} are the first order kinetic constant of the association and dissociation reactions, respectively). At the equilibrium, these rates balanced out, and thus the following will become true:

$$k_{on}[APO][HEME] = k_{off}[HOLO]$$

The concentrations' ratio of the different species at the equilibrium will be determined by the ratio of the kinetic constant. This ratio will define the association equilibrium constant K_a and its inverse the dissociation constant K_d:

$$K_a = \frac{1}{K_d} = \frac{k_{on}}{k_{off}} = \frac{[HOLO]}{[APO][HEME]}$$

In this particular case, we focus on the dissociation constant [Eq. 1].

$$K_d = \frac{k_{off}}{k_{on}} = \frac{[APO][HEME]}{[HOLO]} \quad \text{[Eq. 1]}$$

The Gibbs free energy of the association process (ΔG_a), which represents the energy released upon binding (also interpreted as the minimum energy needed to dissociate), can be calculated from the equilibrium association constant (K_a):

$$\Delta G_a = -RT \ln K_a$$

where R is the gas constant (R = 1.98 cal mol^{-1} K^{-1}) and T is the temperature in Kelvin. The value of ΔG_a determines the stability of the HOLO complex and describes the mutual affinity of the interacting components. In this example, the difference spectra upon heme binding to apomyoglobin can be directly determined using a double-beam spectrophotometer and two cuvettes, each initially filled with 1 ml of a solution of the receptor. The heme group, which generates a stronger signal than the apoprotein, serves as the receptor.

These cuvettes are placed on their respective sites (blank and sample) inside the spectrophotometer and the baseline performed. Initially, as both cuvettes have the same content they absorb the same and thus the difference spectrum is flat at zero. Once the baseline has been obtained, we take the sample cuvette and perform consecutive additions of the ligand (apomyoglobin) and do the same with buffer to the blank cuvette[1], recording the differential spectra after each addition. The volume added at each step must be carefully decided, and previous experience is required in order to determine the appropriate sequence.

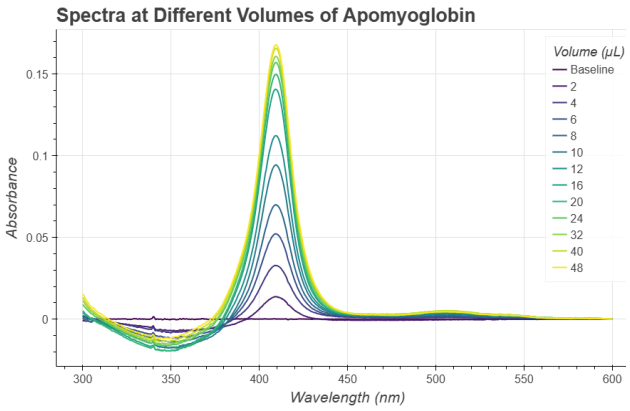


Figure 3: Evolution of differential spectra upon titration of a heme solution with increasing concentrations (as volume added) of apomyoglobin. The legend indicates the total accumulated volume of apomyoglobin after stepwise addition to the heme sample cuvette in µL. The spectra have been normalized so that Abs_{600} is equal to zero. There is a maximum at 409.5 nm and a minimum at 348.0 nm. Initial concentration of heme group in sample and blank cuvettes was [HEME]=0.91 µM and the concentration of the apomyoglobin stock solution was [APOMYOGLOBIN]=56.11 µM. The figure was generated using the Kd_ORD1.csv file that contains experimental derived data located in the Practice_data_sample folder of the DAIPProLi GitHub repository, along with the DAIPProLi script.

During the assay, it is necessary to keep track of the absorbance difference between the minimum (at around 355 nm, in this particular case) and maximum (at around 410 nm) of the spectrum (ΔAbs) (Kundu et al., 2015). This difference increases as larger amounts of the holoprotein are formed (**Figure 3**). When the equilibrium is reached, no more changes are expected to be observed.

$$\Delta Abs = Abs_{410} - Abs_{355}$$

[1] This step is necessary to keep the concentration of the receptor (heme group) in both cuvettes equalized.

Once the ΔAbs for each volume addition has been obtained, the data fitting can be performed to calculate the dissociation constant (K_d). For this particular fitting, a complex formation with stoichiometry 1:1 is assumed (Gómez-Moreno Sanz and Sancho Sanz, 2003, pp. 463–468).

Let Δε be the extinction coefficient of the perturbation associated to the complex formation and considering Lambert-Beer law:

$$\Delta Abs = l \cdot \Delta\varepsilon \cdot \Delta c \qquad \text{[Eq. 2]}$$

where ΔAbs and Δc are the perturbation absorbance (differential spectrum) and the change in complex concentration after each addition, respectively.

The concentrations of the protein ligand and the complex change with each addition. They are related to those of [APO], [HOLO] and [HEME] by:
$[P]_T$ = [APO] + [HOLO]
$[L]_T$ = [HEME] + [HOLO]
[PL] = [HOLO]

Equation 1 can be expressed as:

$$K_d = \frac{([P]_T - [PL])\,([L]_T - [PL])}{[PL]} \qquad \text{[Eq. 3]}$$

Solving for [PL]:

$$[PL] = \frac{[P]_T + [L]_T + K_d - \sqrt{([P]_T + [L]_T + K_d)^2 - 4[P]_T[L]_T}}{2} \qquad \text{[Eq. 4]}$$

Equation 4 is the only sensible solution to Equation 3. The other one, with a '+' sign in front of the square root of the numerator, has no physical meaning because concentration can never be a negative value. A '+' sign leads to a concentration of [PL] greater than those of [P] and [L]. This implies negative values for the concentrations of [APO] and [HEME], which obviously is physically meaningless.

Therefore, by combining equations [Eq. 2] and [Eq. 4], the dependence of the spectroscopic signal, ΔAbs, on the total concentration of apomyoglobin added at each titration step can be established.

$$\Delta Abs = l \cdot \Delta\varepsilon \cdot [PL] = l \cdot \Delta\varepsilon \cdot \left(\frac{[P]_T + [L]_T + K_d - \sqrt{([P]_T + [L]_T + K_d)^2 - 4[P]_T[L]_T}}{2} \right) \qquad \text{[Eq. 5]}$$

$[P]_T$ and $[L]_T$ are known at every step, from their initial concentrations ($[P_0]$ and $[L_0]$), the initial volume (V_0) and the added volume (v):

$$[P]_T = \frac{[P_0] \cdot v}{V_0 + v} \quad \text{[Eq. 6]} \qquad [L]_T = \frac{[L_0] \cdot V_0}{V_0 + v} \quad \text{[Eq. 7]}$$

Therefore, [Eq. 5] is a function of type:

$$y = f(C; x, \beta)$$

Where the following elements are identified:

1. Function f models the Physics of the system (Equation 5) and those which establish the relationship between [P], ad [L], with their initial concentrations, the initial volume and the added volume (Equations 6 and 7).
2. y is the dependent variable, which is the measured observable (ΔAbs, the differential absorbance).
3. x is the independent variable, whose value changes under the experimenter's control (v, the total volume added up to each titration step).
4. β is the set of parameters whose values the experimenter aims to work out: $\beta \equiv \{K_d, \Delta\varepsilon\}$ K_d, and Δε, being respectively the dissociation constant and the extinction coefficient associated to the perturbation linked to the formation of the complex HOLO formation of the complex.
5. C is the set of physical magnitudes that remain constant throughout the experiment $C \equiv \{[APO]_0, [HEME]_0, V_0, l\}$, namely the initial concentrations of APO and HEME, the initial volume in the cuvette and the optical pathlength of the cell.

The aim is to compute the values of the parameters that yield the best fit to the model. This process begins by determining a first estimation for the values of the parameters, to subsequently refine them using least-square fitting methods (Sancho and Gómez-Moreno, 1991).

2.3 Stopped-Flow Spectrophotometry with Photodiode Array Detection in the Presteady State

Stopped-flow (SF) is a spectroscopic technique used in the study of kinetics and mechanisms of fast chemical reactions over timescales of milliseconds to seconds, being often used to study biochemical processes, especially those involving catalytic processes mediated by enzymes or protein-ligand interactions, during the establishment of the equilibrium condition (Ferreira and Medina, 2021; Sørlie et al., 2000).

The SF instrument is a rapid mixing device where two solutions, such as an enzyme and a substrate, are quickly mixed together under highly controlled conditions into the observation cell, where they are illuminated by an observation light source. The change in a selected optical property as a function of time, can then be measured by the use of an adequate detector (usually absorbance at a given wavelength, absorbance with photodiode array, fluorescence, …). Particularly, the coupling of this technique with photodiode array detection, allows capturing time evolution of multiple wavelengths simultaneously, enabling detailed spectral analysis in real time. This combination is very convenient, as it allows tracking quick absorbance changes associated with interaction and reactive processes, even enabling the identification of intermediate species that only exist briefly during the reaction process (Ferreira and Medina, 2021).

For many systems, such kinetics in the time-scale of milliseconds occur before the system reaches the equilibrium, therefore, they are known as measurements in the pre-steady state. This approach reveals early mechanistic details, such as the initial interaction of reacting species or the formation of transient intermediates, that might not be detectable during the steady state phase. Therefore, SF spectrophotometry, with its high temporal resolution, is essential in studying these rapid changes and enables the analysis of reaction rates and intermediate species (Ferreira and Medina, 2021; Johnson, 1992, pp. 12–14; Sørlie et al., 2000). Photodiode array detection enhances this capability by providing a full absorbance spectrum at each timepoint, which allows the monitoring of specific chromophores or intermediates as the reaction progresses. Together, these techniques help unravel underlying reaction mechanisms, including enzyme-substrate and protein-ligand interactions (as well as reactive processes occurring as consequence of their interactions), by tracking dynamic concentration changes of reactants, products, and intermediates in real time (Ferreira and Medina, 2021). The identification of specific intermediates is possible, provided that such species exhibit distinct spectral properties.

One common application of SF spectrophotometry is enzyme kinetics, where the binding of a substrate to an enzyme, the formation of an enzyme-substrate complex and the catalytic reaction itself can be studied in detail. By analyzing these data, pre-steady state kinetics rate constants for the overall, and even individual, process can be obtained under different experimental set-ups (for example increasing the substrate concentration) (Ferreira and Medina, 2021; Johnson, 1992).

2.3.1 Determination of observed kinetic rates and species' spectroscopic properties

One particular application of this method, taken as working example in this study, is the determination of the observed kinetic rates (k_{obs}) and spectroscopic properties of the intermediate species' involved in the reduction of ferredoxin-$NADP^+$ reductase from *Brucella ovis* ($BoFPR_{ox}$) by its NADPH substrate (**Figure 4**) (Ferreira and Medina, 2021; Pérez-Amigot et al., 2019).

The absorbance is measured for n_λ different values of the wavelength (λ) and n_t timepoints (t). Thus, the measurements can be represented by n_t x n_λ matrix. An example of such data is shown in **Figure 4**.

At this point, there is no information about the inner workings of the reaction model that rules the reduction of FPR_{ox} by NADPH. Therefore, a method is needed to determine the number of spectroscopic active species involved in this process. Singular Value Decomposition (SVD) (refer to section 4.2.1) can be used to obtain a model-free estimation of such number. However, Mathematics is not self-sufficient for ultimately resolving the question. Experience and knowledge of the Chemistry of the process comes into play to complete the SVD treatment to yield well-founded educated hypotheses.

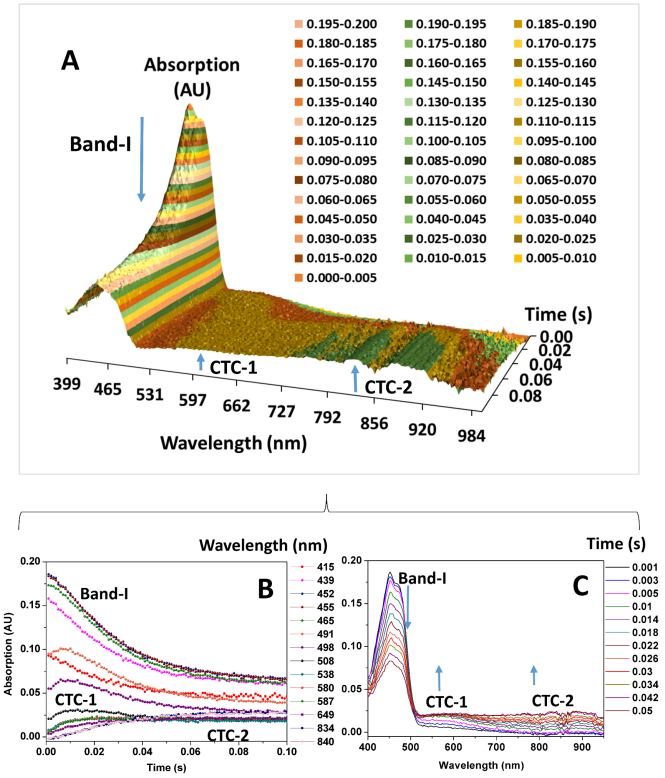


Figure 4. Spectral evolution of the reduction of a bacterial ferredoxin-NADP⁺ reductase (FPR) by a 1.5-fold excess of NADPH after their mixing in a SX-17MV stopped-flow equipment (*Applied Photophysics Ltd., Cambridge, UK*) coupled to a PDA detector with the X-SCAN software. **(A)** 3D plot showing the evolution of the protein spectral characteristics as a function of time and wavelength upon the processes of interaction with the NADPH and hydride transfer from the coenzyme to the FAD cofactor. **(B)** 2D plot showing the evolution of selected wavelengths along the reaction. **(C)** 2D plot showing the spectra of the reaction mixture at selected times. Data collected in 25 mM Tris/HCl pH 7.4 at 6 °C. The initial absorption increase in the 580 region envisages formation of an intermediate species that in this particular case correlates with the formation of a FPR_{ox}:NADPH charge transfer complex (CTC-1). Subsequently it is observed the absorption decrease in the Band-I (452 nm) characteristic of oxidized FAD, concomitantly with the appearance of a new broad band in the 800-900 nm region characteristic of a FPR_{rd}:$NADP^+$ charge transfer complex (CTC-2). Figure taken from (Ferreira and Medina, 2021).

Performing SVD will return a vector with the singular values[2] ordered in descending order of magnitude. This only provides a list of singular values and their magnitudes, but does not specify which ones are the most relevant (Significant Singular Values, SSVs). The number of SSVs is what can be used as a model-free indicator of the linearly independent components (spectroscopic species (*absorbers*) in this case) present in the original data.

In some cases, just visualizing the magnitudes of the singular values might allow determining the SSVs fairly well, even though it is subjective. This raises the issue of selecting a mathematical method to determine the number of SSVs to reduce personal bias.

The methods used in this project will be explained in section 4.2.2.

Once we have mathematically determined the number of SSVs, it is still necessary to choose the result of one method, and at this point is when information, knowledge, and hypothesis about the studied system comes into play. In the example broached, the oxidized form of FPR was mixed with NADPH (the reducing agent), and the result of the enzymatic reaction is that FPR gets reduced and NADPH oxidized into $NADP^+$, probably through the formation of some intermediate species:

$$FPR_{ox} + NADPH \rightleftharpoons \ldots \rightleftharpoons FPR_{red} + NADP^+$$

[2] The value of the singular value can be understood as "amount" of information of the dataset that it explains.

Therefore, there will be at least 2 SSVs. If one of the methods determines that there are, for example, 20 SSV, this result is not wrong, but it may be that the threshold used is not appropriate for the "shape" of the data. In this particular example, it was determined that there were 3 spectroscopic species involved in the reaction, in other words, 3 SSVs.

Having determined the number of SSVs, a matrix approximation can be performed. A matrix approximation is a reconstruction of the original data using only the largest singular values, the SSVs. The aim of the matrix approximation is to reduce the noise of the data, thus capturing the essence of the reaction.

Thereafter, it is necessary to propose a reaction model and to initialize the experimental parameters (pathlength of the cuvette, initial concentrations and estimates of the rate constants) into a fitting software. Considering the example at hand, the model has to include 3 spectroscopic species (A, B and C). Moreover, since it is a stopped-flow experiment the time window is small and reverse reactions are not likely to occur since we are under pre-steady-state conditions, therefore, we can assume the following expected reaction model:

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$

As a result, the proposed model has two rate constants (k_1 and k_2), for which an initial estimation must be inputted for the sake of fitting of the model.

Thus, the steps of the fitting routine will be the following:

First, define the expected kinetic model that rules de reaction as a set of Ordinary Differential Equations (ODEs). In this particular example, they will be:

$$\frac{dA}{dt} = -k_1[A] \qquad \text{[Eq. 8]}$$

$$\frac{dB}{dt} = k_1[A] - k_2[B] \qquad \text{[Eq. 9]}$$

$$\frac{dC}{dt} = k_2[B] \qquad \text{[Eq. 10]}$$

Using the estimation of k_1 and k_2, numerical integration at the timepoints evaluated in the experiments is then performed. As a result of the numerical integration, an "estimation" of the concentration of each species at each timepoint is obtained (Concentration profile).

Then using the "estimated" concentration profile, the determination of the characteristic spectra of the spectroscopic species (the extinction coefficients for each absorber/species at every wavelength evaluated) can be performed using the Beer-Lambert Law in the following fashion:

$$Abs_{\lambda,t} = l \cdot \sum_x \left(C_x(t) \cdot \varepsilon_x(\lambda) \right) \qquad \text{[Eq. 11]}$$

Where:

- **$Abs_{\lambda,t}$** is the absorbance at each wavelength (λ) and timepoint (t).
- **l** is the optical pathlength of the cell where the absorbance measurement is being recorded.
- **$C_x(t)$** is the concentration of each species (x) at each timepoint (t).
- **$\varepsilon_x(\lambda)$** is the extinction coefficient of each species (x) at each wavelength (λ).

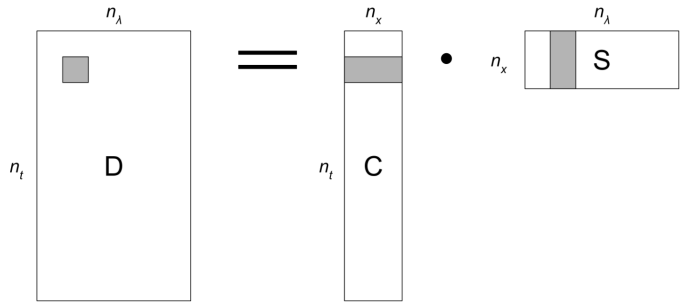


Figure 5: Representation of the Beer-Lambert law in matricial form for a multiwavelength time evolution data set (D is the Absorbance data, C the concentration profile and S the spectroscopic species spectra). Note that despite not being represented (due to the value typically being the unity), the product of the matrices C and S must be multiplied by the optical pathlength of the cell.

Where:

- D is the experimental absorbance matrix with dimensions n_t x n_λ.
- C is the concentration profile with dimensions n_t x n_x (where n_x is the number of species considered in the model).
- S contains the characteristic spectra of spectroscopic species (extinction coefficients of each species at each wavelength) with dimensions n_x x n_λ.

Solving for the S matrix from **Figure 5** the estimation of the extinction coefficients of each species at each wavelength is obtained. Now having an estimate of the C and S matrix, an estimation of the absorbance (D' matrix) at each timepoint (t) and at each wavelength (λ) can be obtained by simply performing the operation displayed in **Figure 5**.

Finally, a model approximation of the D matrix (D' matrix) is obtained, which is then compared to the original D matrix in the Levenberg-Marquardt minimization algorithm. The algorithm will optimize the parameters (β) in order to minimize the following:

$$\beta^* = argmin_{\{\beta\}} \sum_i \left[y_i - f(x_i;\beta)\right]^2 = \sum_i [y_i - \hat{y}_i]^2 \qquad \text{[Eq. 12]}$$

Where:

- y_i is the experimental absorbance
- $\hat{y}_i$ is the model approximation of the absorbance

Therefore, the model is a function of type:

$$y = f(C; x, \beta)$$

Where the following elements are identified:

1. Function f models the Physics of the system (Equations 8-10 and Equation 11).
2. y is the dependent variable, which is the measured observable ($Abs_{\lambda,t}$, the absorbance measured at different times and wavelengths).
3. x is the independent variable, whose value changes under the experimenter's control x≡{{t},{λ}} {t} being the timepoints at which the absorbance is measured and the {λ}, the wavelength at which the absorbance is measured).
4. β is the set of parameters whose values the experimenter aims to work out: β≡{k_n} namely the kinetic rates constants k_n.
5. C is the set of physical magnitudes that remain constant throughout the experiment C≡{n_S, {$[S]_0$}, l}, namely the number of species, their initial concentrations and the optical pathlength of the cell.

The purpose is to compute the values of the parameters that yield the best fit to the model. In order to do so, the process begins by determining a first estimation for the values of these parameters.

Once the fitting routine is completed, it will provide the following:

- Concentration profile of the "absorbers".
- Calculated spectra of the "absorbers".
- Observed rate constants for the transformation between species (k_1 and k_2) according to the proposed reaction model.

To conclude, the process must come to the assessment of the quality of fitted data by:

- Checking the lack of systematic deviations from residuals plots at different wavelengths and times (beware that matrix approximation might unveil the intrinsic noise of the equipment performing the measurements).
- Inspection of the calculated spectra.
- Consistency among the number of SSV with the final fit model.

2.3.2 Deriving interaction and limiting rate constants from the analysis of observed rate constants (k_{obs})

k_{obs} values obtained as above-mentioned are of interest to determine binding constant and limit reaction rates. In order to achieve this, it is necessary to determine k_{obs} in a range of ligand/substrate concentrations ([L])[3]. The determined k_{obs} might be linearly dependent, hyperbolically dependent, or independent on the [L]. These features are highly relevant as they provide kinetic parameters for the processes being studied (**Figure 6**).

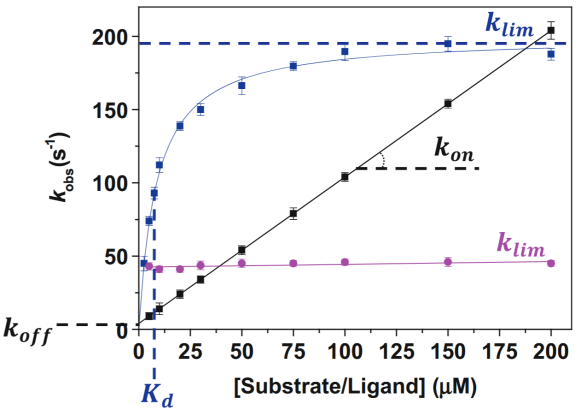


Figure 6: Evolution of k_{obs} on the Ligand concentrations. Linear dependent data in black, hyperbolically dependent data in blue and independent data in magenta. Figure taken from (Ferreira and Medina, 2021).

Linear dependent data (black line in **Figure 6**) indicates that the process corresponds to a binding event or a reactive process that does not require a previous formation of a complex of the reactants. In this case, the fitting of these data to [Eq. 13] would provide the **ligand association** (k_{on}) and **dissociation** (k_{off}) **rate constants**.

[3] Note that for each reaction step, it involves a kinetic rate constant. In the reaction model proposed above, it involves three species connected through two reaction steps and thus two different kinetic rates constants, k_1 and k_2. Therefore, when determining binding constant and reaction rates there will be $k_{1\text{-}obs}$ and $k_{2\text{-}obs}$, and each of them will be analyzed independently.

Data showing a saturation profile (blue line in **Figure 6**) suggests ligand association followed by a process as a consequence of such interaction (such as an electron transfer process or a reorganization process). Non-linear fitting such as described in [Eq. 14] will allow determining the **dissociation constant** of the interaction process, K_d, and the subsequent **limiting rate constant**, k_{lim}, for the reaction process. There are other equations which allow determining reverse rate constant or inhibition constant. However, they are beyond the scope of this project, but if required it would be possible to easily implement them in the AKiPa algorithm detailed later on.

Lastly, k_{obs} values independent of ligand concentration (magenta line in **Figure 6**) relate directly with the limiting rate constant of the measured process (k_{lim}) (Ferreira and Medina, 2021).

$$k_{obs} = k_{on} \cdot conc + k_{off} \quad \text{[Eq. 13]} \qquad k_{obs} = \frac{conc \cdot k_{lim}}{conc + K_d} \quad \text{[Eq. 14]}$$

3. Objectives

The primary goal of this project is to develop an open-source software repository for the analysis of spectroscopic data dealing with the determination of interaction and kinetic parameters involving proteins. They will be available to students and researchers on a free access platform (GitHub). To achieve this goal, the following specific objectives were pursued in this study:

1. Implementation of a user-friendly algorithm, named DAIPProLi, for the determination of the dissociation constant (K_d) and of the extinction coefficient of the spectroscopic change ($\Delta\varepsilon$) from experimental data obtained by differential spectroscopy.
2. Implementation of a user-friendly algorithm, named KiPaD, for the determination of kinetic parameters, observed rate constants (k_{obs}), and spectroscopic properties of intermediate species, from data obtained from multiwavelength time-resolved absorption spectroscopy using stopped-flow.
3. Implementation of a user-friendly algorithm, named AKiPa, for the analysis of pre-steady state kinetic data obtained from analysis of stopped-flow data, namely k_{obs} at different ligand concentrations, to determine the interaction and kinetic parameters of the observed process: dissociation constant (K_d) and limiting rate constant for reactive process (k_{lim}).
4. Adaptation of the created algorithms to protocols used in practical lessons in the degree of Biotechnology.

4. Methods and Materials

To tackle the analysis of the above indicated experimental data we will rely on Google Colab, as the environment to write the script's code in Python and to make it accessible to the users, and on five mathematical resources.

4.1 Python environment

Python is the programming language chosen for the scripts developed during this Master Project. Python is a versatile and powerful open source programming language known for its simplicity, readability, and ease of learning, making it perfect for projects for both beginners and experts. Released in 1991 by Guido van Rossum, Python has efficient **high-level data structures** (It abstracts from the complexity of hardware details, meaning that the code is closer to natural language and is easier for humans to write and understand it). It also provides a simple but effective approach to **object-oriented programming** (a type of computer programming in which programs are composed of objects which communicate with each other, which may be arranged into hierarchies, and which can be combined to form additional objects, modular programming). Moreover, python's elegant syntax and **dynamic typing** (it assigns the type of the variable at the runtime based on the variable's value at the time, in order words, it allows to overwrite variables), together with its **interpreted nature** (the code is executed line by line, with the program being translated (by the interpreter) and executed simultaneously), make it an ideal language for scripting for both beginners and experienced users. ("Definition of OBJECT-ORIENTED PROGRAMMING," n.d.; "Dynamic typing - MDN Web Docs Glossary," 2023; "The Python Tutorial," n.d.; Murat, 2023)

Furthermore, Python is widely used by the scientific community and has plenty of modules that enhance its capability to handle a diverse range of actions. This makes it an interesting platform for programming since it allows sharing and running the script easily. Also, it is supported by Google Colab. The user does not need to have Python installed, just needs to have a Google account, and Google Colab will provide access to a remote computer that will run the script. Several Python functions, modules, and libraries have been used in the present study.

4.1.1 NumPy library

NumPy (Numerical Python) is a powerful Python library for numerical computing. It provides efficient, high-performance tools for working with large, multidimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these data structures. It provides the tools for organizing, exploring and analyzing scientific data. As an example of its relevance, NumPy had an important role in the software used in the discovery of gravitational waves and the first imaging of a black hole (Harris et al., 2020).

4.1.2 Matplotlib library

Matplotlib is a comprehensive Python library for creating static, animated, and interactive visualizations. In other words, it is a library for making 2D plots of arrays in Python. Moreover, it allows creating simple plots with few commands, and is highly customizable, being a core tool for Python data visualization (Hunter, 2007).

4.1.3 Pandas library

Pandas is a powerful Python library designed for data manipulation and analysis. It provides flexible and efficient tools for working with structured data (even with large datasets), such as table and time series, making it adequate for data science workflows (McKinney, 2010).

4.1.4 Bokeh library

Bokeh is a Python library for creating interactive visualizations for modern web browsers (plots are rendered as HTML files, making it simple to share plots while maintaining their interactive functions). It allows building more aesthetically pleasing graphics than Matplotlib. It is designed to work smoothly with large datasets and offer powerful tools for building interactive plots (such as zooming, panning, and hover tools) (“Bokeh documentation,” 2024).

4.1.5 funcionesGenerales module

funcionesGenerales is a module containing the necessary functions to feed the main function used in this project, `procesa`. The `procesa` function aim is to perform a least-squares minimization using the Levenberg-Marquardt algorithm. It is a very flexible function which requires the following inputs to perform the minimization process:

- The model (function) to which to perform the minimization. The model describes the process evaluated (Chemical or Physical process).
- The initial estimation of the parameters.
- The names of the initial parameters to be optimized (the ones allowed to vary), and which ones must remain constant. This allows the user to fit their values at different steps.
- The dependent variable obtained “experimentally” (the measured observable).
- The independent variable (whose value is controlled by the experimenter) and other parameters needed for the model function to work (magnitudes that remain constant throughout the measurement).
- Optionally, bounds can be added, so the modelled data does not exceed certain values.

The function `procesa` will then return the optimized parameters along with their standard deviation calculated through the Hessian matrix, the regression coefficient (R^2) and details about the fitting process in a nested dictionary.

4.1.6 Other Python resources

4.1.6.1 csv module

The csv module implements functions to read and write tabular data in CSV format. The CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. The *reader* and *writer* objects of the csv module read and write sequences, respectively. (“csv — CSV File Reading and Writing,” 2024).

4.1.6.2 zipfile module

The zipfile module provides tools to create, read, write, append and list a ZIP file. The ZIP file format is a common archive and compression standard without losing data (“zipfile — Work with ZIP archives,” 2024) . One interesting use of creating a ZIP file is to combine all outputs into a single file, which makes handling results more efficiently and manageable.

4.1.6.3 svd function

The svd function, in the scipy.linalg module, performs SVD of the desired matrix. It can take multiple parameters, but to work only needs one, the matrix (a) upon which to perform SVD. This function factorizes the matrix *a* into two unitary matrices *U* and V^T, and a 1-D array *s* of singular values (real, non-negative) so that: $a = U \cdot S \cdot V^T$ where *S* is a diagonal matrix (whose diagonal entries are *s*) ("svd — SciPy v1.14.1 Manual," 2024).

4.1.6.4 LinearRegression class

The LinearRegression class is in the sklearn.linear_model module. It fits a linear model with coefficients w= (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation ("LinearRegression," 2024).

4.1.6.5 datetime class

The datetime class, in the datetime module, provides the current date and time at the moment it is invoked ("datetime — Basic date and time types," 2024).

4.1.6.6 Axes3D class

The Axes3D class, in the module mpl_toolkits.mplot3d from Matplotlib, provides tools for creating and editing 3D plots in Matplotlib ("mpl_toolkits.mplot3d.axes3d.Axes3D — Matplotlib 3.9.3 documentation," 2024).

4.2 Mathematical framework

4.2.1 Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)

Multiwavelength time-resolved absorption data obtained by stopped-flow spectroscopy contains information about the number of species which can be considered an "*absorber*[4]" (Cochran et al., 1980). To estimate the minimum number of *absorbers,* Principal Component Analysis (PCA) is usually used. PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower dimensional space, preserving as much of the data's variance as possible (explained variance). It works by finding the principal components that capture the most variance in the data. PCA is a particular case of Singular Value Decomposition (SVD), which is a matrix factorization technique (Bhatt, 2023). Therefore, we can compute PCA through SVD. SVD can then be understood as a generalized version of the eigen decomposition, because the former can be done with any matrix, whereas the latter can only be done with square matrices. As such, for any given dataset there is a unique set of singular values, similar to a square matrix and its eigenvalues.

Let's assume a matrix D of size n_t x n_λ (where n_t is the number of timepoints and n_λ is the number of wavelengths evaluated). The matrix D is decomposed as shown in Equation 15 and **Figure 7**.

$$D = U \cdot \Sigma \cdot V^T \qquad \text{[Eq. 15]}$$

[4] The term **absorber** may not refer to a single *species* but several. This can be further clarified by talking in terms of populations. Let's assume we have three absorbers, and three species, each absorber will be populated by all species, however, one would be the most significant.

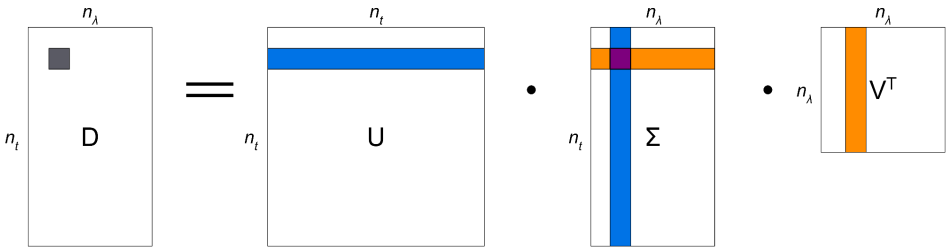


Figure 7: Visual representation of Equation 15.

where:

- U is a n_t x n_t matrix of left singular vectors: its columns are the eigenvectors of $D \cdot D^T$.
- Σ is a n_t x n_λ diagonal matrix of singular values: the square root of eigenvalues of both $D \cdot D^T$ and $D^T \cdot D$.
- V^T is a n_λ x n_λ transposed matrix of right singular vectors: the columns of V (rows of V^T) are the eigenvectors of $D^T \cdot D$.

This is the reason SVD can be used for dimensionality reduction. Any dataset with n_t rows and n_λ columns can be reduced into smaller subsets, which are the most relevant to explain the data. Therefore, the result is a matrix with a lower rank that is said to approximate the original matrix. In order to obtain an approximate matrix, an SVD operation is performed on the original data, selecting the ***k*** larger singular values in Sigma (Σ) along with corresponding rows of U and columns of V^T. Thus, D can be approximated with a lower-rank matrix D' as:

$$D' = U_k \cdot \Sigma_k \cdot V_k^T \qquad \text{[Eq. 16]}$$

Here, U_k, Σ_k and V^T_k are truncated versions of the original matrices, keeping the top ***k*** elements. As a result, D' is a compressed approximation of A, capturing its most significant characteristics while reducing the noise and less significant dimensions (Brownlee, 2019; Falini, 2022).

In summary, performing SVD allows determining the number of important directions in a data set, that is the number of *absorbers* present, and to reduce the background noise and the data's dimensions, speeding calculations.

4.2.2 Determination of the Significant Singular Values (SSVs)

Once SVD is performed, a list of singular values, whose values represent their "relevance" to explain the behavior of the data, is produced. This information can intuitively be used as an indicator of the number of relevant directions in data, in other words the number of *absorbers*, being a model-free estimation of the number of absorbers. However, this cut-off approach is subjective. Therefore, there is a need for a method to choose the ***k***-largest singular values (SSVs) which represent *most* of the variation of the data objectively (or at least more objectively). The ***k***-largest singular values also refer to the principal components of the data. In this project, three methods have been implemented to determine the number of SSVs: scree plot (modified), broken-stick and entropy-based.

4.2.2.1 Scree plot with fit method

A scree plot is a simple representation of singular values in a Cartesian coordinate system. The number of SSVs is typically picked at the "elbow" point, where the curve's slope changes significantly. This approach is effective when there are large differences between the largest and smallest singular values, as shown in **Figure 8**. However, its main drawback is that the cut-off point is defined visually, and thus it is rather subjective (Falini, 2022).

To reduce subjectivity, the cut-point is based on how many singular values fit a straight line with a regression coefficient (R^2) above a threshold (e.g., R^2=0.9). The R^2 value measures the goodness-of-fit of the singular values to a line, with 1 indicating a perfect fit. If adding another singular value causes R^2 to drop below the threshold, the previous number of singular values is selected. While this method improves in objectivity, it still requires the user to define the threshold. However, if R^2 falls below 0.9 the fit can be considered as "not good enough" and, thus, using values inferior to 0.9 is non-advisable. Moreover, this method only works if there is a "sharp" decrease in the value of the singular values, if the datasets do not present such behavior another method should be used for selecting the SSVs.

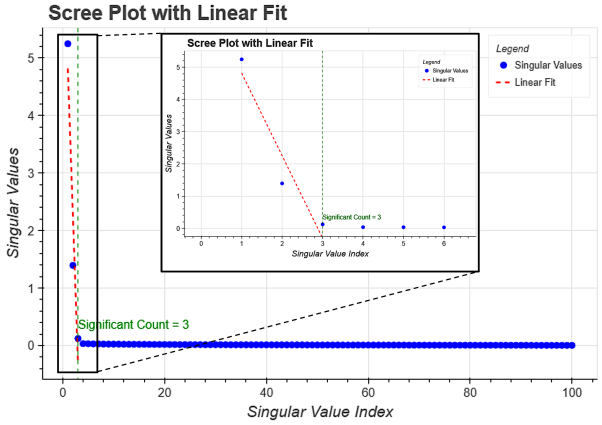


Figure 8: Scree plot with linear fit as the cut-off mechanism to select the SSVs. This plot is created by the function scree_plot_with_fit in the KiPaD script. The data used for this plot are present in the Sample_data folder in the KiPaD repository in GitHub (file used: BoFPR+60NADPH00071_t.csv) which contains data from previously published results (Pérez-Amigot et al., 2019).

4.2.2.2 Broken-stick model method

The broken-stick model is based on the following analogy. Imagine a stick of unit length broken randomly into r pieces. The lengths of these pieces represent the expected proportions of variance attributed to the components in a completely random system (Cangelosi and Goriely, 2007; Falini, 2022). These proportions can be computed mathematically and compared to the squared singular values of the data. The main points are:

- The broken-stick model predicts the expected distribution of squared singular values under random variance allocation.
- Squared singular values larger than the expected values from the broken-stick distribution are considered significant and worth retaining, and likely to represent relevant structure in the dataset.

Mathematical Formulation

The expected length of the ***k***-th largest segment of a stick randomly broken into r segments is:

$$b_k := \frac{1}{r}\sum_{i=k}^{r}\frac{1}{i} \qquad \text{[Eq. 17]}$$

Where:

- r: Total number of singular values (or components)
- k: Rank of the segment or singular value
- b_k: Expected value of the ***k***-th largest singular value under the broken-stick model

Note, the total sum of all b_k values is 1. Therefore, in order to compare the b_k values to the squared singular values (whose total sum is not 1 and thus not directly comparable to the b_k values), there is a need to scale the b_k values or normalize the squared singular values. In this project it was decided to normalize the squared singular values, because the main aim is to compare proportions of variance explained, while absolute variance values are not crucial for the analysis.
After squaring the singular values (σ^2_k), they are normalized (Normalized σ^2_k) by their total sum as shown in Equation 18:

$$Normalized\ \sigma^2_k = \frac{\sigma^2_k}{\sum_{k=1}^{r} \sigma^2_k} \qquad \text{[Eq. 18]}$$

Procedure:

1. Perform SVD to obtain the singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$.
2. Square the singular values.
3. Normalize the squared singular values by their total sum, as shown in Equation 18.
4. Generate the Broken-Stick Distribution, computing the expected values b_k for each rank k using [Eq. 17].
5. Compare the normalized squared singular values to the Broken-Stick Model. It retains the normalized squared singular values (Normalized σ^2_k) that lie above their corresponding b_k. These represent the SSVs or significant components of the data.

The comparison can be visualized by plotting the normalized squared singular values (Normalized σ^2_k) against the expected values from the broken-stick model (b_k) (**Figure 9**). Points above the broken-stick curve are deemed significant. The logic behind this method is that the broken-stick model assumes a random variance distribution. Thus, singular values above this baseline indicate structure in the dataset (not noise).

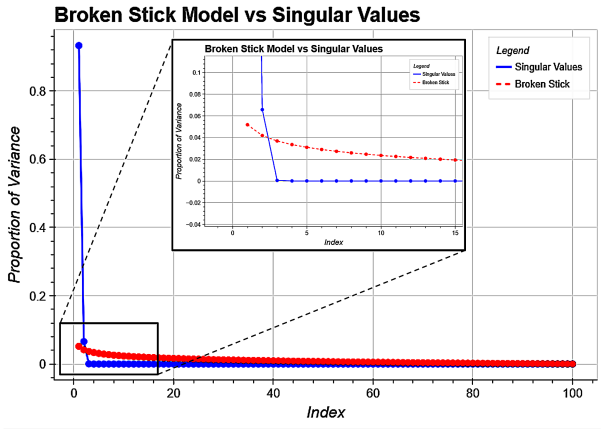


Figure 9: Graphical representation of the normalized broken-stick distribution (red) and the normalized singular values (blue). This plot is created by the function broken_stick_model in the KiPaD script. The data used for this plot are present in the Sample_data folder in the KiPaD repository in GitHub (file used: BoFPR+60NADPH00071_t.csv) which contains data from previously published results (Pérez-Amigot et al., 2019).

4.2.2.3 Entropy-based Selection method

The entropy-based selection employs a measure called **entropy** to figure out how many of the largest singular values (***k***) are needed to explain the most important patterns or structures in a dataset. It is a way to decide which parts of the data are most relevant and how many of them must be kept.

Entropy (*E(X)*) measures the degree of uncertainty or the spread of probability across different states, X are all the states available and x each singular state. It is defined as:

$$E(X) = -\sum_{x} p(x) log(p(x)) \qquad \text{[Eq. 19]}$$

For a dataset *D*, entropy can quantify how evenly distributed the dataset's singular values (σ_i) are. This SVD-based entropy is expressed as:

$$E(D) = -\frac{1}{log(r)} \sum_{j=1}^{r} f_j log(f_j) \qquad \text{[Eq. 20]}$$

with f_j ("normalized" singular values) being:

$$f_j = \frac{\sigma^2_j}{\sum_{i=1}^{r} \sigma^2_i}$$

The **entropy-based selection** method uses entropy to decide how many of the largest singular values to keep. **Entropy** measures how evenly the singular values contribute to the dataset's structure:

- **High entropy:** All singular values contribute equally, indicating no clear dominant structure in the dataset.
- **Low entropy:** A few singular values contribute the most, suggesting that the dataset's structure can be simplified.

Since [Eq. 20] uses the normalized squared singular values, the entropy *E(D)* is within the range [0,1].

To choose the number (***k***) of SSVs:

1. Determine the entropy using [Eq. 20], using the normalized singular values.
2. Find the smallest ***k*** such that the cumulative entropy reaches a set percentage of the total. This means a threshold must be set (e.g., 70% or 85%)

By selecting ***k*** SSVs, this method ensures that enough singular values are retained to capture the dataset's essential structure while discarding noise or redundant dimensions (Falini, 2022).

4.2.3 Least squares minimization, Levenberg–Marquardt algorithm

The Levenberg-Marquardt algorithm is a robust method for optimizing parameters in non-linear models. It combines features from two common optimization techniques: the Gauss-Newton method and the gradient descent.

Problem. Let the model be fitted to the data being

$$\hat{y} = f(x_1, x_2, \dots, x_m; \beta_1, \beta_2, \dots, \beta_k) = f(x, \beta) \qquad \text{[Eq. 21]}$$

where x are the independent variables, β is a set of parameters and ŷ is the expected value of the dependent variable y. While performing the fitting of the model to the experimental dependent variable, the problem in question is computing the estimates of the parameters β which will minimize

$$\beta^* = argmin_{\{\beta\}} \sum_i [y_i - f(x_i; \beta)]^2 = argmin_{\{\beta\}} \sum_i [y_i - \hat{y}_i]^2 \qquad \text{[Eq. 22]}$$

where $\hat{y}_i$ is the value of y predicted by [Eq. 21] at the *i*-th data point.

In order to perform this minimization of a non-linear model and to obtain estimates of the parameters β the Levenberg-Marquardt algorithm combines the Gauss-Newton method and the gradient descent method, in a way that compensate each other drawbacks:

- **Gauss-Netwon method:** its approach is to linearize [Eq. 22] around the current estimate using a Taylor expansion. Its advantages are: fast convergence and optimized for least squares. However, as drawbacks it has divergence issues (it may not minimize properly) and computationally is costly, since solving each interaction requires calculating the transpose of a matrix (jacobian) and multiplying it by itself.
- **Gradient descent method:** it minimizes [Eq. 22] by updating iteratively the parameters β in the direction of the **negative gradient** of the function, which leads to the steepest descent. Its advantages are its straightforwardness, flexibility, and the fact that it always minimizes. As negative points, it shows slow convergence (requires many iterations, especially near the minimum) and the probability to get stuck in local minima.

The Levenberg-Marquardt interpolates both these methods in such a way that the final result is an algorithm with fast convergence and very good at minimizing. However, it still does not guarantee a global minimum. The parameter update, $\delta\beta$, is computed as:

$$\delta\beta = \left(P^T P + \lambda I\right)^{-1} P^T \left[y - \hat{y}\right] \qquad \text{[Eq. 23]}$$

where:

- *P* is the Jacobian matrix of the function f: $P[i, j] = \partial f_i / \partial f_j$.
- P^TP is the curvature approximation from Gauss-Newton.
- λI is a regularization term that determines the update direction.

The role of λ is:

- If λ>>1: the diagonal regularization term dominates, and the update is done employing the gradient descent method.

$$\delta\beta = P^T(y - \hat{y}) \qquad \text{[Eq. 24]}$$

- If λ<<1: The algorithm behaves like the Gauss-Newton Method.

$$\delta\beta = (P^T P)^{-1} P^T(y - \hat{y}) \qquad \text{[Eq. 25]}$$

The Levenberg-Marquardt is an iterative algorithm. In each iteration, calculates a new value of β (β = β + $\delta\beta$). Therefore, the optimization process us ruled by λ:

1. If the current step increases the error (increases [Eq. 22]): it increases λ making the algorithm behave more like gradient descent [Eq. 24].
2. If the current step reduces the error (reduces [Eq. 22]): it decreases λ making the algorithm behave more like Gauss-Newton to accelerate convergence (near solution) [Eq. 25]

This dynamic adjustment ensures the algorithm remains stable while converging efficiently. In simpler terms, the Levenberg-Marquardt algorithm uses the gradient descent [Eq. 24] for the initial interactions until the algorithm determines it is near the minimum, and then switches to the Gauss-Newton [Eq. 25] method to accelerate the convergence (Marquardt, 1963).

4.2.4 Runge-Kutta (RK4) method

The Runge-Kutta method of order 4 (RK4) is one of the most commonly used numerical techniques to solve ODEs of the form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0$$

Where:

- *t* is the independent variable (e.g., time).
- *y* is the dependent variable (the solution we want to obtain, e.g., concentration at each timepoint in the case described in section 2.3.1 and section 5.2).
- *f(t, y)* gives the rate of change of *y* at any point (e.g., [Eqs. 8-10] in section 2.3.1).

RK4 performs stepwise integration starting at y_n (known value at time t_n), it estimates y_{n+1} (the value at time $t_{n+1} = t_n + h$, h is the step size) by averaging the derivatives calculated at different points within the interval. It computes y_{n+1} by calculating the weighted average of four slopes:

- k_1: The rate of change at the start of the interval.

$$k_1 = hf(t_n, y_n) \qquad \text{[Eq. 26]}$$

- k_2: The rate of change at the midpoint, using k_1 to estimate *y*.

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \qquad \text{[Eq. 27]}$$

- k_3: Another midpoint evaluation, using k_2 for the estimate.

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \qquad \text{[Eq. 28]}$$

- k_4: The rate of change at the end of the interval, using k_3.

$$k_4 = hf(t_n + h, y_n + k_3) \qquad \text{[Eq. 29]}$$

The next value (y_{n+1}) is therefore calculated as:

$$y_{n+1} = y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \qquad \text{[Eq. 30]}$$

The error associated with the RK4 method is proportional to the fifth power of the time step (h^5), so this method is called fourth order. This makes it more accurate than simpler methods like Euler's or the midpoint method when using the same step size (Press et al., 1992).

4.2.5 Pseudo-inverse method

The pseudo-inverse of a matrix, also called the **Moore-Penrose inverse**, is a generalization of the inverse of a matrix for situations where the matrix is not square or invertible. It allows solving linear systems of equations and performing matrix operations when the **standard inverse** does not exist.

In our case, it allows us to solve systems of linear equations where the number of equations (m) and the number of variables (n) are not the same: overdetermined (m>n) and underdetermined (m<n) systems (When m=n it behaves like the standard inverse) (Rallabandi, 2023).

To compute the pseudo-inverse in this project, the `numpy.linalg.pinv` function has been used. It calculates the generalized inverse of a matrix using its SVD and including all large singular values ("numpy.linalg.pinv — NumPy v2.1 Manual," n.d.).

The pseudo-inverse is used to invert C matrix (with dimensions n_t x n_x), its exact solution (P), with the notation shown in **Figure 5**, is:

$$P = (C^T C)^{-1} C^T \qquad \text{[Eq. 31]}$$

P has dimensions n_x x n_t. Thus:

$$D = l \cdot C \cdot S$$

can be rearranged to solve for S:

$$P \cdot D = l \cdot P \cdot C \cdot S \quad \rightarrow \quad P \cdot D = l \cdot S \quad \rightarrow \quad \frac{P \cdot D}{l} = S$$

The definition of pseudo-inverse proposed in [Eq. 31] can be applied to the C matrix as it an overdetermined system and all its columns will always have full column rank if the following conditions are satisfied: independent species, valid reaction model, and matrix obtained numerically (not experimentally).

5. Results

5.1. DAIPProLi algorithm

DAIPProLi (Determination of Affinity of Interaction Parameters between Protein and Ligand) is a python script aimed to analyze data generated by differential spectroscopy experiments, in particular those that involve a receptor and its ligand in a stoichiometry 1:1. This analysis involves first the determination of wavelengths (λ) of the maximum and minimum absorbance for each differential spectrum obtained along a titration experiment of the receptor (at known concentration and volume) with increasing volumes of the ligand from a concentrated stock at known concentration. Subsequently, ΔAbs is obtained by performing $Abs_{\lambda=maximum}$- $Abs_{\lambda=minimum}$ after each titration point. Finally, ΔAbs and volumes of ligand added along with the proposed model are used to perform a least squares minimization with Levenberg-Marquardt algorithm to fit it to the experimental data, ΔAbs and volumes of ligand. This is used to determine the dissociation constant, K_d, and the extinction coefficient of the perturbation associated with complex formation, Δε. The output of the script is:

1. The dataset uploaded with the differential absorbance spectrum at each titration point (table).
2. The experimental ΔAbs at each titration point (table).
3. The modelled ΔAbs at each titration point (table).
4. A 2D plot of the spectra at different titration points, from the experimental data.
5. A 3D plot of the spectra at different titration points, from the experimental data.
6. A 2D plot of the ΔAbs at different titration points, from the experimental data along with the fitting.
7. A table with the fitting results that includes the initial estimations for the parameters as well as their final values, obtained after fitting, the regression coefficient and other parameters informing about the goodness of fit.

The script contains 4 original functions, and it is divided into 11 sections or Google Colab cells.

5.1.1 Functions

5.1.1.1 leeFichero

```
leeFichero(nombrFich, colValLabel='Wavelength', intercaladas=True, separador=',', column_inter="Name")
```

The function `leeFichero` reads the uploaded CSV file containing the experimental data to Google Colab and formats the data inside it into a pandas DataFrame compatible with the following functions. The file is assumed to contain absorbance values obtained through differential spectroscopy, arranged as a 2D array, where the rows represent the difference absorbance (ΔAbs) at each wavelength and the columns represent the difference absorbance at each added volume. The functions return a DataFrame with the absorbance values. The column names and row names of the DataFrame represent, respectively, the volumes added at each titration step and the wavelengths at which absorbance was measured.

This function takes 5 arguments, from which two of them have default values and do not require manipulation (unless the format of the data uploaded changes).

Parameters:

- `nombrFich` **(string)**: this argument takes the name of the CSV file uploaded to Google Colab. In order words, it is the name of the file to read.
- `colValLabel` **(string, optional)** : The label in the file indicating the start of the main data columns (default is `'Wavelength'`). This argument acts as a keyword. The function skips the contents of the file previous to the appearance of this keyword, and stores the information following it in the returned DataFrame.
- `intercaladas` **(bool, optional)**: It determines whether to process intercalated data columns (default is `False`) or not. When set to `True`, the function records only the values at odd-indexed positions in each data row (reference for the first row that contains `colValLabel`) and skip certain rows according to the argument `column_inter` as shown in **Table 2**. If set to `False`, all the data in the file will be recorded starting from the `colValLabel`.
- `separador` **(string, optional)**: The delimiter that separates values in the file (default is `','`). This is a string argument in which we determine the punctuation mark that delimits the columns in the CSV file. It has a default value, although it is kept editable just in case.
- `column_inter` ***(string, optional)***: The label indicating the start of intercalated column headers. Useful for capturing additional header information from specific rows (default is `"Name"`). It is only used when `intercaladas` is `True`. It has a default value that depends on the format of the uploaded CSV file. It searches for the row in which this string is present and obtains all the values in its row as column names for the DataFrame output.

Returns:

- `df` **(DataFrame)**: The data organized in a DataFrame, where columns are labeled according to the detected or assigned column names. The DataFrame contains the absorbance values for different values of the added volume and wavelength. The column names are the added volumes and the row names are the wavelengths. If the file is empty or does not adhere to the expected structure (characterized by the values of arguments `intercaladas` and `column_inter`) the function returns `None`.

Notes:

- This function is designed to handle data files with variable header rows and intercalated data.
- Column names can be inferred from the presence of `colValLabel` or `column_inter`
- When `intercaladas` is set to `True`, rows with two or more data entries will alternate values, and if `"Baseline"` appears in the headers, it will be removed from the columns.

If `intercaladas` is set to `False`, `leeFichero` expects the data to be arranged in the CSV (**input**) format shown in **Table 1**.

Table 1. Expected input table when `intercaladas=False`.

Table 1: Expected input table of the experimental data when `intercaladas=False`. The output is the table as is. That is, when `intercaladas=False` the input and output format is the same.

Wavelength (nm)	Baseline	2	4	6	…
300	-0.0002	0.0004	0.0031	0.0034	…
300.5	-0.0001	0.0004	0.0029	0.0034	…
301	-0.0002	0.0002	0.0027	0.0032	…
…	…	…	…	…	…

If `intercaladas` is set to `True`, `leeFichero` expects the data to be arranged in the CSV format shown in **Table 2**.

Table 2. Expected input table when `intercaladas=True`.

Table 2: Expected input table of the experimental data when `intercaladas=True`. The output is the table with the format as displayed in Table 1.

…	…	…	…	…	…	…	…	…	…
SAMPLES									…
Name	Baseline		0		2		4		…
…	…	…	…	…	…	…	…	…	…
	Wavelength (nm)	Abs	Wavelength (nm)	Abs	Wavelength (nm)	Abs	Wavelength (nm)	Abs	
	300	-0.015	300	-0.0002	300	0.0004	300	0.0031	…
	300.5	-0.015	300.5	-0.0001	300.5	0.0004	300.5	0.0029	…
	301	-0.015	301	-0.0002	301	0.0002	301	0.0027	…
…	…	…	…	…	…	…	…	…	…

In both cases, the output is a DataFrame with similar structure of the data disposition of the file expected of the function `leeFichero` when `intercaladas=False` (**Table 1**).

5.1.1.2 diff_absorbance

`diff_absorbance(df, method="Manual", debug=False, min_chosen=0, max_chosen=0)`

This function processes a DataFrame containing wavelength and absorbance data to calculate the difference in absorbance (ΔAbs) between specified minimum and maximum wavelengths for each column, typically corresponding to different sample volumes or conditions. In the DAIPProLi script, it takes the output from the function `leeFichero`.

Parameters:

- **`df` (DataFrame)**: The input data, where the first column represents wavelengths and the remaining columns represents measurements at different added volumes (conditions).
- **`method` (string)**: The method to determine the minimum and maximum wavelengths. It takes one of the following three arguments:
    - **“`Mean`”**: the mean of the wavelengths at minimum and maximum absorbance values of all conditions are calculated, and the nearest wavelengths in the dataset are selected.

    - `"Median"`: Uses the median of the wavelengths at minimum and maximum absorbance values of all conditions evaluated. Default is `"Median"`.
    - `"Manual"`: Uses `min_chosen` and `max_chosen` as the chosen wavelengths for the minimum and maximum absorbance, respectively.
- **`check_vals` (bool, optional):** If `True`, prints intermediate DataFrame showing the wavelength and absorbance values at the detected minimum and maximum values for each column. Default is `False`.
- **`min_chosen` (float, optional):** Only used when `method="Manual"`. Specifies the wavelength to be treated as the minimum for absorbance calculations. Default is `0`.
- **`max_chosen` (float, optional):** Only used when `method="Manual"`. Specifies the wavelength to be treated as the maximum for absorbance calculations. Default is `0`.

Returns:

- **`out_df` (DataFrame):** A DataFrame with the following columns:
    - `"Volume µl"`: The volumes derived from the original column names in the input DataFrame
    - `"ΔAbs"`: The calculated difference in absorbance between the "closest" maximum and minimum wavelength for each column.
    - `"min_wave (nm)"`: The wavelength used as the minimum for absorbance calculations.
    - `"max_wave (nm)"`: The wavelength used as the maximum for absorbance calculations.

Notes:

- **Absorbance Extraction:** The function extracts the absorbance columns (excluding the first column, which is assumed to be wavelengths) and processes each column individually.

5.1.1.3 shift_spectra

`shift_spectra_to_zero(datos, target_wavelength)`

The function adjusts all spectra so that their absorbance is zero at a given target wavelength. This is useful for standardizing spectra by removing offsets at a specific wavelength.

Parameters:

- **`data` (DataFrame):** Input data where the first column represents wavelengths (e.g., in nm), and the remaining columns contain absorbance values for different conditions or sample volumes.
- **`target_wavelength` (float):** The wavelength at which each spectrum will be shifted to zero. The function identifies the closest wavelength in `data` to this target and uses it as a reference point for shifting.

Returns:

- **`shifted_data` (DataFrame):** A DataFrame with the same structure as the input, where absorbance values have been shifted so that absorbance at the `target_wavelength` is zero for each spectrum. The first column remains the wavelength, and the remaining columns are adjusted absorbance values.

Notes:

- **Wavelength Matching:** The function finds the closest available wavelength in the `data` to the specified `target_wavelength`. This value is then used to retrieve absorbance values for each spectrum.
- **Shift Calculation:** The absorbance value at the target wavelength is subtracted from all absorbance values within each spectrum, effectively zeroing absorbance at the target wavelength.

5.1.1.4 binding_model

`binding_model(params, v)`

This function computes the predicted change in absorbance (ΔAbs) for a ligand binding model, as a function of sample volume. This model accounts for equilibrium conditions involving ligand concentration, receptor concentration, and dissociation constants. It is suitable for absorbance-based assays that study binding affinities.

Parameters:

- **`params` (dict):** a dictionary containing the following parameters:
  - **`L` (float):** Pathlength of the sample cell in cm, which affects the absorbance through the Lambert-Beer Law.
  - **`V0` (float):** Initial volume of the receptor solution in µL.
  - **`Lo` (float):** Initial concentration of ligand, in the stock solution, in µM.
  - **`Ro` (float):** Initial concentration of receptor in µM.
  - **`Kd` (float):** Dissociation constant for the ligand-receptor complex, in µM.
  - **`epsilon` (float):** Extinction coefficient of the ligand-receptor complex in L/(mol · cm)
- **`v` (array-like):** Array of volumes (µL) of ligand added to the receptor solution. This represents the varying conditions for which the change in absorbance is calculated.

Returns:

- **`deltaAbs` (array-like):** Calculated ΔAbs for each added ligand volume (v). This array represents the binding response curve based on the model parameters.

Notes

- **Total Ligand and Receptor Concentrations:**
  - **Lt:** Total ligand concentration after mixing, calculated as:

$$Lt = Lo\frac{V}{V0+v}$$

  - **Rt**: Total receptor concentration after mixing, calculated as:

$$Rt = Ro\frac{V0}{V0+v}$$

  - Lt and Rt corresponds to $[L]_T$ and $[P]_T$ respectively in [Eq. 5] with their definition being [Eq. 6] for $[P]_T$ and [Eq. 7] for $[L]_T$ in section 2.2.
- **Absorbance Calculation:**
  - ΔAbs is derived using the Lambert-Beer Law, adjusted for the equilibrium concentrations of ligand and receptor.
  - The function solves for the equilibrium concentration of the ligand-receptor under the assumption of a 1:1 binding model, as explained in section 2.2.

5.1.2 Sections

In this context, the term 'cell' or 'chunk' refers to a section of code within the Google Colab script. All figures in this section are screen captures (SCs) from the DAIPProLi Google Colab script, which is available on GitHub (https://github.com/unizar-flav/DAIPProLi). The data used in these figures comes from the file Kd_ORD1.csv, located in the Practice_data_sample folder of the DAIPProLi GitHub repository.

5.1.2.1 Environment

In this section, once the user clicks on the "play" button in the Environment section, as shown in **Figure 10**, the script loads the libraries, modules, and functions required by the script in the subsequent sections. This section also contains the functions described in section 5.1.1.

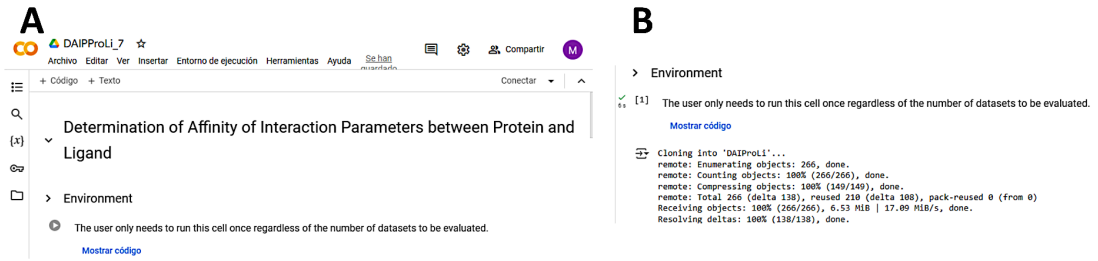


Figure 10: SCs of the **Environment** section. **(A)** Header of the DAIPProLi script in Google Colab. **(B)** Output of the section when executed.

5.1.2.2 Upload file

Here the user has to specify the format of the CSV file, explained in section 5.1.1.1 leeFichero. After that, the cell must be run and a message prompting the user to upload the file will appear (**Figure 11-A**). Once the file has been uploaded and read by the function `leeFichero`, the console will print a table with the loaded data (**Figure 11-B**).

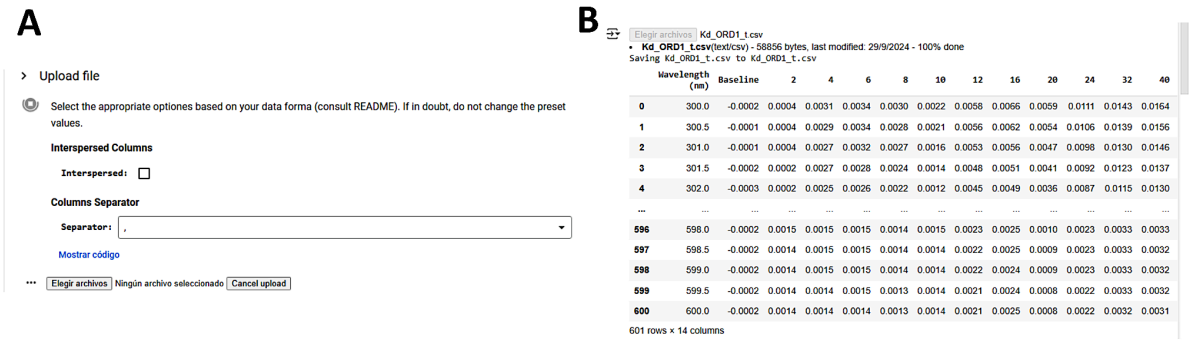


Figure 11: SCs of the **Upload file** section. **(A)** Input SC showing the areas to determine if data contains interspersed columns, and the column separator. Once run, it prompts the user to choose the file to upload. **(B)** Output of the section displaying a DataFrame of the file uploaded.

5.1.2.3 Absorbance Difference

The cell here allows the user to decide the method for which to identify the wavelengths for the maximum and minimum in the difference absorbance spectra dataset uploaded (**Figure 11-B**). The user may choose between three methods: **"Mean"**, **"Median"** and **"Manual"**:

- **"Mean"**: it uses the mean of the wavelengths where the minimum and maximum absorbance values of all conditions evaluated are found.
- **"Median"**: it uses the median of the wavelengths where the minimum and maximum absorbance values of all conditions evaluated are found.
- **"Manual"**: it uses `Minimum` and `Maximum` as the chosen wavelengths for the minimum and maximum absorbance, respectively (**Figure 12-A**).

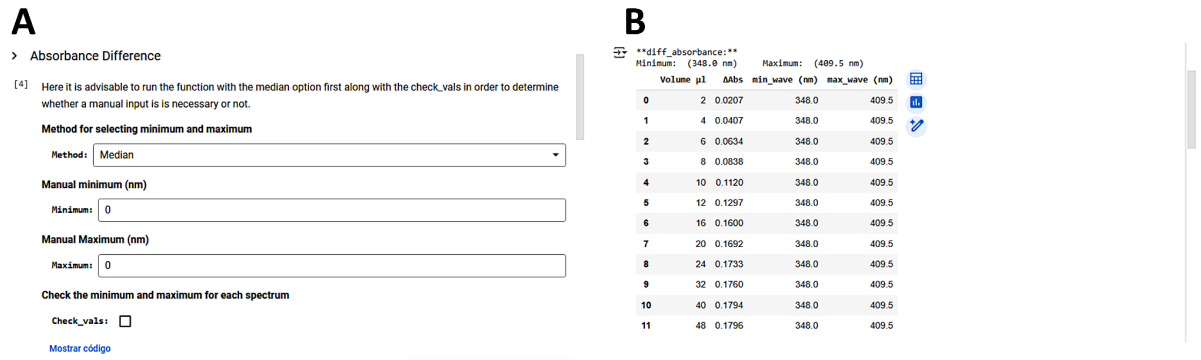


Figure 12: SCs of the **Absorbance Difference** section. **(A)** Input SC showing a dropdown menu for choosing the method for selecting the minimum and maximum, two areas for their manual input and a box (boolean) for checking intermediate results of the function `diff_absorbance`. **(B)** Output displaying a DataFrame with the ΔAbs for each added volume of ligand.

The default method is the **"Median"** as it is less influenced by extreme values. At the bottom of this cell there is a box, that by default is unchecked, which will provide the wavelengths at which the minimum and maximum were found at each spectrum contained in the data (according to what has been explained at section 5.1.1.2 diff_absorbance). In case the user has previous knowledge about the wavelengths where the minimum and maximum are located, the user can select the **"Manual"** method and input the values in the corresponding spot. In any method, after executing the cell, it will feed the input parameters into the `diff_absorbance` function. As output, it will print a table with the difference absorbance (ΔAbs) at each added volume of the ligand, along with the wavelength chosen for the minimum and maximum at each spectrum (**Figure 12-B**).

5.1.2.4 Spectra plot

This section plots the difference spectra contained in the file uploaded in 2D and 3D (**Figure 13-B,C**), and it allows the user to modify the titles, the axis, and the legend. Moreover, it allows the user to shift the spectra to zero at a given target wavelength using the function `shift_spectra` by just clicking on the box `Shift_spectra` (boolean, check inside the box means True) and inputting the desired `target_wavelength` (**Figure 13-A**).

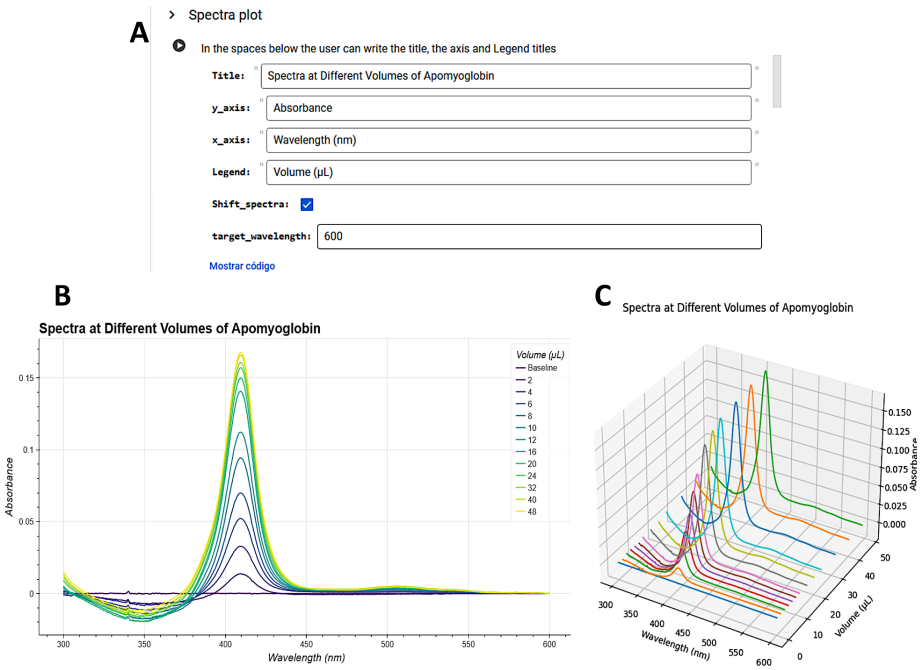


Figure 13: SCs of the **Spectra plot** section. **(A)** Input SC showing the areas for introducing names, target wavelength and checkbox that controls whether to shift the spectra to zero. **(B-C)** Outputs, showing the 2D **(B)** and 3D **(C)** plots of the spectra obtained after addition to the heme sample of increasing volumes of the Apomyoglobin stock solution. In **(B)** and **(C)** all spectra have been shifted to zero at 600 nm.

5.1.2.5 Plot ΔAbsorbance vs Volume (µL)

This section plots the absorbance difference (ΔAbs) (calculated in section Absorbance Difference) against volume of ligand added, allowing the user to personalize the titles, the axis, and the legend (**Figure 14**).

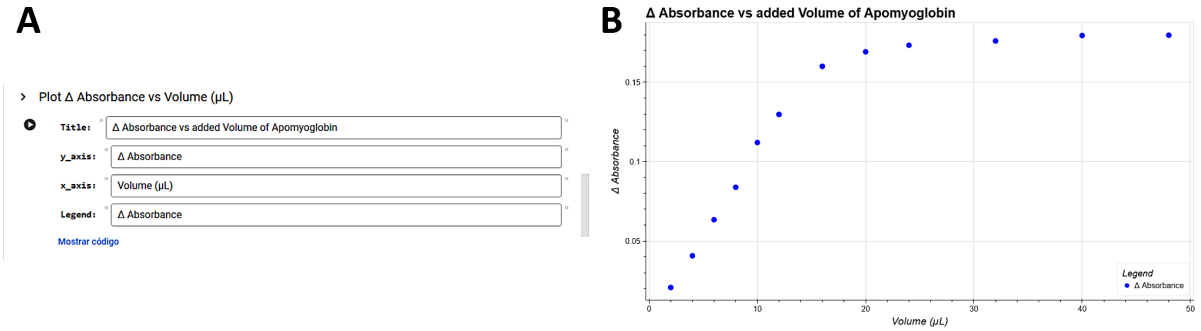


Figure 14: SCs of the **Plot ΔAbsorbance vs Volume (µL)** section. **(A)** Input SC showing the areas for introducing names for the title, axis, and Legend of the plot. **(B)** Output: A scatter plot showing experimental ΔAbs at each added volume of Apomyoglobin.

5.1.2.6 Model

This cell does not require any action except running it, as it contains the proposed model for binding between the receptor and the ligand (section 5.1.1.4 binding_model) (**Figure 15**). If another model requires to be studied, the user will need to rewrite the function `binding_model` according to the new requirements along with the Parameters section if needed. The `binding_model` is the function to minimize and is fed to `procesa`.

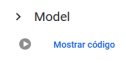


Figure 15: SC of the **Model** section, it contains the function `binding_model` which can be modified if required for fitting other models.

5.1.2.7 Parameters

In this cell the user has to initialize the parameters needed for the model and specify whether they must be optimized or not (check the boxes to fix the value inputted, if the value needs to be optimized leave its correspondent box unchecked) (**Figure 16-A**).

This section will classify the parameters into variable and fixed parameters to handle them properly when performing the fitting and minimization process. Once the cell has been executed, it will print the "list" (dictionary) of fixed and variable parameters (**Figure 16-B**). The parameters inputted will later be fed to the function `procesa` in the section Procesa.

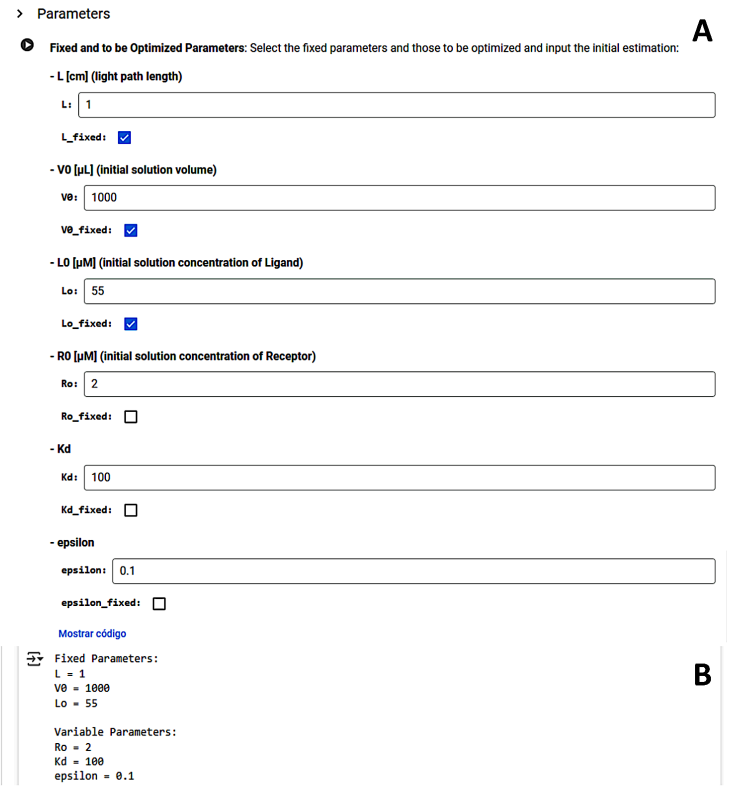


Figure 16: SCs of the **Parameters** section. **(A)** Input SC showing the areas for introducing the parameters and if their values are fixed (not optimized). **(B)** Output: Prints the inputted parameter values and indicates whether each parameter is fixed or variable during the fitting and minimization process.

5.1.2.8 Procesa

This cell calls the function `procesa` from the module funcionesGenerales, with the necessary arguments inputted. When executed, it performs a least-squares minimization through the Levenberg-Marquardt algorithm (**Figure 17-A**). The console will print the residuals at each iteration and when it finishes the minimization process it prints the optimized parameters (`parAjustados`), along with their standard deviation (`sd`) (**Figure 17-B**).

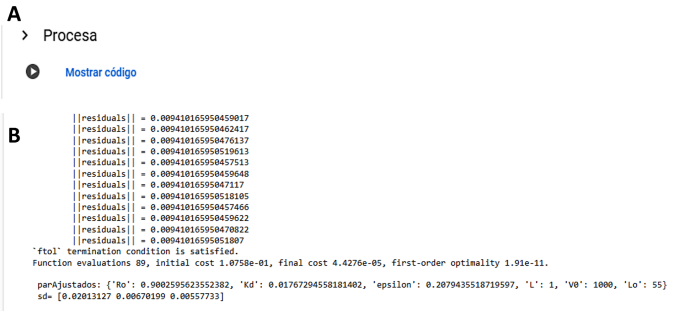


Figure 17: SCs of the **Procesa** section. **(A)** The `procesa` function is fed with the necessary arguments, such as indicated in section 4.1.5 funcionesGenerales. **(B)** Output: Prints the residuals at each iteration, and then prints the optimized parameters along with their standard deviation.

5.1.2.9 Plot ΔAbsorbance vs Volume (µL) with model fitting

This section plots the Absorbance difference (calculated in section Absorbance Difference) against Volume of ligand added from the experimental data (blue dots) along with the modelled data curve using the optimized parameters from `procesa` (red line). It also allows the user to personalize the titles, the axis, and the legend (**Figure 18**).

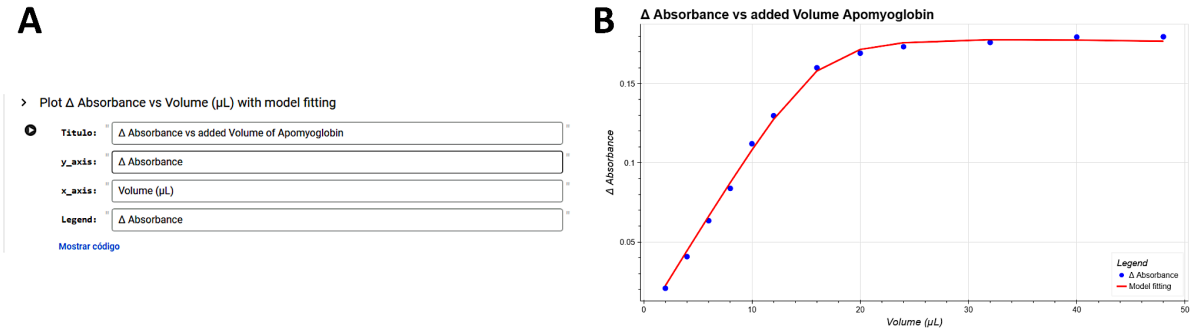


Figure 18: SCs of the **Plot ΔAbsorbance vs Volume (µL) with model fitting)** section. **(A)** Input SC showing the areas for introducing names for the title, axis, and Legend of the plot. **(B)** Output: A scatter plot showing experimental ΔAbs at each added volume of Apomyoglobin (blue dots) along with their modelled ΔAbs at each added volume of Apomyoglobin (red line).

5.1.2.10 Export results

This last cell saves and exports all the inputted and generated data. The datasets are saved in CSV format and the plots are saved in HTML and PNG (**Figure 19-A**). All of these items are downloaded into a ZIP archive, which the user can name (**Figure 19-B**). Nonetheless, this name will always include as a suffix the date and time at which this cell was run. The outputs are those described above (5.1 DAIPProLi).

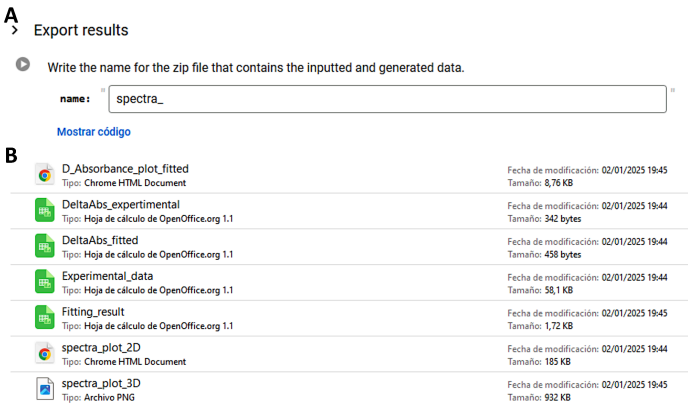


Figure 19: SCs of the **Export results** section. **(A)** Input SC showing the area for introducing the name of the ZIP archive exported by this section when executed. **(B)** SC of the inside of the ZIP archive generated.

5.2 KiPaD algorithm

KiPaD (Kinetic Parameters Determination) is a Python script designed to calculate the observed rate constants (k_{obs}) and spectroscopic properties of intermediate reaction species from data obtained through multiwavelength time-resolved absorption spectroscopy measurements, using stopped-flow techniques. The script employs SVD to identify the principal components, or SSVs, present in the dataset, providing a model-free estimation of the number of spectroscopically active species involved in the reaction under study.

Once the number of species is determined, SVD is applied again to perform dimensionality reduction on the original data, reducing noise while preserving relevant information. Thus, SVD is employed both to estimate the number of species (SSVs) and to enhance the dataset by reducing noise.

The resulting SSVs and "denoised" data are then used to propose a kinetic model, enabling the determination of k_{obs} values for the different steps of the overall process and the estimation of the spectra of the relevant spectroscopic species (absorbers) involved.

From this point onward, the script will focus on "deconstructing" the original absorbance matrix (D, with dimensions n_t x n_λ) into three components: a concentration profile matrix (C, with dimensions n_t x n_x), a spectroscopic species spectra matrix (S, with dimensions n_x x n_λ), and a residual matrix (E, with dimensions n_t x n_λ), which is the experimental error (**Figure 20**).

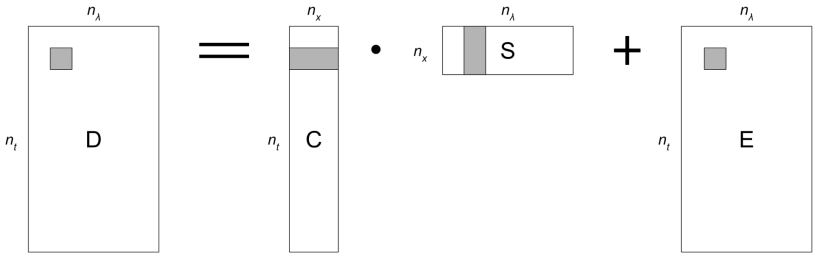


Figure 20: Graphic representation of the end result of the KiPaD algorithm. The algorithm estimates the Concentration profiles (C matrix) and the spectra (S matrix) of the relevant species along the reaction (D matrix contains the absorbance values). Their product gives rise to a modelled absorbance (D'), and the subtraction of D' matrix to the D matrix results in the experimental error (E matrix). Where $\mathbf{n_t}$: number of timepoints; $\mathbf{n_\lambda}$: number of wavelengths; and $\mathbf{n_x}$: number of spectroscopic species. Note that despite not being represented (due to the value typically being the unity), the product of the matrices C and S must be multiplied by the optical pathlength of the cell.

To obtain the concentration profile (C) and spectroscopic species spectra (S) matrices from the absorbance matrix (D), the user has to input the expected number of spectroscopic species and to initialize the parameters required by the proposed model. The algorithm of the model first estimates a concentration profile (C), using the initial estimates for the kinetic rates, by numerical integration (using RK4) of the system of ODEs that rules the model. Once an estimate for the concentration profile (C) is obtained, it is then used along with the experimental absorbance (D) to determine an estimate for the spectral properties of the species (S). Then the model algorithm reconstructs the absorbance matrix (D') with the estimated C and S matrices. Then in the minimization step `procesa` performs:

$$\beta^* = argmin_{\{\beta\}} \sum_i [d_i - d'_i]^2 \qquad \text{[Eq. 32]}$$

through the Levenberg-Marquardt algorithm, explained in section 4.2.3. Since both absorbance data are in matrix form they need to be flattened to pass them to `procesa` for optimization (thus the different notation in [Eq. 32] d_i and d_i' are respectively the flattened the experimental absorbance and modelled absorbance): instead of a matrix of n_t x n_λ a vector with $n_t \cdot n_\lambda$ elements is used. After the minimization process, `procesa` provides the optimized rate constants along with their estimated standard deviation and additional data about the fitting process. The output of this script is:

1. **Original experimental data**: The uploaded absorbance dataset.
2. **Denoised experimental data**: The approximated (denoised) absorbance dataset determined through SVD.

3. **Modelled data**: The modelled absorbance dataset obtained using the optimized kinetic rates in the reaction model proposed.
4. **Residuals (Denoised - Modelled data)**: The residuals from the original uploaded absorbance dataset and the modelled absorbance dataset.
5. **Residuals (Original - Modelled data)**: The residuals from the approximated absorbance dataset and the modelled absorbance dataset.
6. **Concentration profile**: The concentration profile matrix obtained from the model using the optimized kinetic rates.
7. **Spectroscopic species spectra**: The spectra of the spectroscopic species matrix obtained from the model using the optimized kinetic rates.
8. A table with the fitting results that includes the initial parameters inputted, their homologous adjusted, the standard deviation of the parameters adjusted, the regression coefficient and other parameters that informs about the goodness of fit.

The script contains 13 original functions, and it is divided into 12 sections or cells in the Google Colab.

5.2.1 Functions

5.2.1.1 lee_espectro

`lee_espectro(nombrFichs, tag="_t", skip_rows=0)`

This function processes multiple CSV files containing spectral data, arranges them into a single DataFrame according to the timepoint (from smaller to bigger), and chooses a representative file name (using the argument `tag`) to keep track of the data being analyzed, particularly for labeling plots. The files are assumed to contain absorbance values arranged as a 2D array, where the rows represent the absorbance at each wavelength and the columns represent the difference absorbance at each timepoint.

Parameters

- **`nombrFichs` (list)**: A list of CSV file names to be read and processed.
- **`tag` (string, optional)**: The substring that determines the representative filename for the combined DataFrame that results from the function. Default is **`"_t"`**.
- **`skip_rows` (int, optional)**: The number of rows to skip at the beginning of each CSV file when reading. Default is `0`.

Returns

- **`df (DataFrame)`**: The combined and sorted DataFrame containing all the spectral data from the files in `nombrFichs` (which in this script are those files the user uploads when prompted).
- **`main (string or None)`**: The name of the representative file from the list, specifically the first file containing the substring tag. Returns `None` if no such file exists.

Notes

- This function was created with the idea that all the files to be read are related to a single reaction studied over different time windows.
- The function cannot handle files containing duplicate timepoints under any circumstances, as it will combine and overwrite them with the latest uploaded file.

5.2.1.2 create_plot

`create_plot(df, Title, x_axis, y_axis, Legend, width=1200, height=700)`

This function creates an interactive plot from a dataset with multiple lines (series), allowing the user to zoom and hover around the plot. It uses Bokeh to create the plot, and it includes a toggle button for displaying or hiding the legend.

Parameters:

- **`df` (DataFrame):** The input data, where each column represents a series to be plotted, and the index serves as the x-axis values.
- **`Title` (string)**: The title of the plot.
- **`x_axis` (string)**: Label for the x-axis.
- **`y_axis` (string)**: Label for the y-axis.
- **`Legend` (string)**: Title of the legend.
- **`width` (int, optional)**: Width of the plot in pixels. Default is `1200`.
- **`height` (int, optional)**: Height of the plot in pixels. Default is `700`.

Returns

- **`column (Bokeh layout)`**: A Bokeh object containing the plot and the toggle button controlling the legend visibility.

Notes

- This function is designed to be compatible with any number of series to be plotted (lines), all of them with different colors. This is achieved by dividing the RGB spectrum into the number of series to be plotted, and thus assigning "evenly spaced" color to each series.
- The toggle button is a necessity when the number of series is large (more than 10 series of data), to hide it to make the plot more legible.
- It is highly compatible with any dataset, as long as it is a DataFrame that contains numbers column and index names.

5.2.1.3 scree_plot_with_fit

`scree_plot_with_fit(singular_values, threshold, width=800, height=600)`

This function draws a scree plot to visualize singular values and identifies SSVs by performing a linear fit. The threshold inputted for the regression coefficient (R^2) determines when the linear fit is no longer acceptable. The outputs are the number of SSVs and a Bokeh plot.

Parameters

- **`singular_values` (array-like)**: an array of singular values obtained from SVD.
- **threshold (float)**: a threshold value (between 0 and 1) for the regression coefficient (R^2). This determines the cutoff for SSVs.
- **`width` (int, optional)**: Width of the plot in pixels. Default is `800`.
- **`height` (int, optional)**: Height of the plot in pixels. Default is `600`.

Returns

- **`sol` (dict):**
  - `"SSVs"`: The number of SSVs identified by the linear regression method.
  - `"plot"`: A Bokeh object containing the scree plot and a toggle button for hiding or displaying the legend.

Notes

- Refer to section 4.2.2.1 for further information about the method scree plot with fit.

5.2.1.4 broken_stick_model

`broken_stick_method(singular_values, width=800, height=600)`

This function implements the Broken-Stick model to determine the number of SSVs by comparing their normalized proportions to their respective randomized distributions, known as the "broken stick" model. The output is the number of SSVs and a Bokeh plot.

Parameters:

- **`singular_values` (array-like):** An array of singular values obtained from SVD.
- **`width` (int, optional)**: Width of the plot in pixels. Default is `800`.
- **`height` (int, optional)**: Height of the plot in pixels. Default is `600`.

Returns:

- **`sol` (dict):**
  - **"`SSVs`"**: The number of SSVs identified by the linear regression method.
  - **"`plot`"**: A Bokeh object containing the singular values and the broken stick model plot, and a toggle button for hiding or displaying the legend.

Notes:

- Refer to section 4.2.2.2 for further information about the normalized broken-stick model.

5.2.1.5 entropy_selection

`entropy_selection(singular_values, entropy_theshold)`

This function uses an entropy-based approach to identify the number of SSVs from a given array. The normalized entropy of singular values is calculated to determine their importance, and a cumulative threshold is applied to identify the most relevant ones.

Parameters:

- **`singular_values` (array-like):** An array of singular values obtained from SVD.
- **`entropy_threshold` (float)**: A threshold value (between 0 and 1), which represent the percentage of cumulative entropy needed to consider a singular value significant.

Returns:

- **`significant_componets` (int):** The number of SSVs. Returns 0 if no singular values meet the threshold.

Notes

- Refer to section 4.2.2.3 for further information about the entropy selection method.

5.2.1.6 matrix_approximation

`matrix_approximation(A, n)`

This function approximates a given matrix *A* by retaining only the top *n* singular values (SSVs) from its SVD. It reconstructs a simplified version of the matrix that captures the most significant parts of its structure.

Parameters:

- **`A` (array-like):** The original matrix to be approximated. Usually it is a 2D NumPy array
- **`n` (int):** The number of SSVs to retain in the approximation.

Returns:

- **`A_approx` (array-like):** The approximated version of matrix *A*, reconstructed using the top *n* singular values.

Notes:

- This function's input and outputs are NumPy arrays, unlike most of the functions of this script that work with DataFrames. This is due to the function `svd` not being compatible with DataFrames.
- The output of this function, the approximated matrix A, is a dataset with reduced 'noise.' This is because the approximation retains only the “important” information, discarding other sources of variability, considered as noise.

5.2.1.7 kinetic_model_matrix

`kinetic_model_matrix(n_species, k_vals)`

This function creates a matrix that represents the kinetic model proposed, which is governed by ODEs. The elements of the matrix are the k_{obs} values that define the rates of inflow and outflow between species.

Parameters:

- **`n_species` (int)**: The number of species in the system. This determines the dimensions of the output matrix.
- **`k_vals` (dict):** A dictionary with the rate constants. Their keys follow the following naming rules:
  - `kn`: Rate constant for the forward reaction from species n to species n+1.
  - `k_n`: Rate constant for the reverse reaction from species n+1 to species n.

Returns:

- **`ode_matrix` (2D NumPy array)**: A square matrix with size `n_species x n_species`, where each entry is the kinetic rate constants contained in the `k_vals` dictionary.

Notes:

- This function works with any number of species and rate constants. However, it is programmed for reactions with forward and reverse reactions involving the next and previous species.

Let us consider a reaction involving n species, and let C_i and k_{ij} be the concentration of the i-species and the rate constant ruling the $C_i \rightarrow C_j$ reaction ($i, j \in [0, n-1]$). In general:

$$\frac{dC_i}{dt} = -\left(\sum_{j=0}^{n-1} k_{ij}\right) C_i + \sum_{j=0}^{n-1} k_{ji} C_j$$

Note that $k_{ii}=0$ for all values of i. The most general kinetic model can be described by the matrix equation:

$$\frac{d\,s}{dt} = As$$

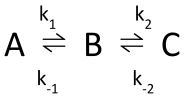
where **s** is an n-component column vector whose i-component is C_i, and **A** is a n x n matrix given by

$$A_{ij} = -\delta_{ij} \sum_{l=0}^{n-1} k_{il} + k_{ji}$$

δ_{ij} being the Kronecker delta (δ_{ij}= 1 if i=j). We note that the sum of the values of each column is 0.

Example:

Let us imagine the reaction in study involves three spectroscopic species. The expected kinetic model[5] would be:



and it is ruled by the following first order ODEs:

$$\frac{dA}{dt} = -k_1 \cdot A + k_{-1} \cdot B$$

$$\frac{dB}{dt} = k_1 \cdot A - (k_{-1} + k_2) \cdot B + k_{-2} \cdot C$$

$$\frac{dC}{dt} = k_2 \cdot B - k_{-2} \cdot C$$

This system of ODEs can be arranged in matricial form, as shown in **Figure 21** which is the output of this function.

$$\begin{pmatrix} -k_1 & k_{-1} & 0 \\ k_1 & -(k_{-1}+k_2) & k_{-2} \\ 0 & k_2 & -k_{-2} \end{pmatrix}$$

Figure 21: Matricial arrangement of the system of ODEs.

5.2.1.8 deriv_conc

`deriv_conc(connc, t, ks_matrix)`

This function computes the rate of change in concentration of the species at a specific timepoint based on a matrix of rate constants (the one obtained using the function `kinetic_model_matrix`). It arranges the derivative of concentrations for a system of species governed by ODEs.

Parameters:

- **`conc` (array-like):** An array representing the concentrations of all species at a given time *t*. Each element corresponds to the concentration of a specific species.
- **`t` (float)**: The timepoint at which the derivative of the concentrations is calculated. This argument is not used, but it is necessary for the ODE solver function.
- **`ks_matrix` (2D NumPy array):** The matrix containing the reaction rates between species. The matrix encodes (this matrix is created by the `kinetic_model_matrix`):
  - Diagonal entries: Net outflow rates for each species.
  - Off-diagonal entries: Inflow rates from other species.

Returns:

- **`np.ndarray` (vector):** This vector contains the derivative of the concentrations of all species at the specified time *t*. Each element is the rate of change of concentration of a specific species.

5.2.1.9 solve_conc_profile

`solv_conc_profile (k_vals, f_deriv, Conc_0, t)`

This function computes the concentration profiles of species over time for a reaction system using a 4th-order Runge Kutta (`RK4`) numerical integration method (from the module funcionesGenerales). It supports working with time steps of unequal length (making possible to combine data of different time frames and different time steps) and initial conditions.

Parameters:

- **`k_vals` (dict):** A dictionary with the reaction rate constants for the system.

[5] Taking the general description aforementioned note that: $A=C_0$, $B=C_1$, $C=C_2$, $k_1=k_{01}$, $k_{-1}=k_{10}$, $k_2=k_{12}$, $k_{-2}=k_{21}$.

- **`f_deriv` (function**): A function that computes the derivatives of the concentrations for all species. In this script, it is expected to use the function `deriv_conc`.
- **`Conc_0` (dict):** Initial concentrations of species as key-value pairs:
    - Keys: species names (e.g., 'A', 'B').
    - Values: Initial concentrations (numeric values).
- **`t` (array-like)**: An array or list of timepoints at which the concentrations should be determined. The timepoints can be non-uniformly spaced.

Returns:

- **`df` (DataFrame):** A DataFrame containing the concentration profiles of all species at the timepoints in argument `t`. Each row contains the concentrations of the different species at a given timepoint. Each column contains the concentration of one species at the different timepoints.

Notes:

- This function will solve the ODE system specified in `kinetic_model_matrix` by numerical integration of the rate of change (`deriv_conc`) using RK4.

5.2.1.10 species_spectra

`species_spectra(k_vals, f_deriv, Conc_0, t, abs, pathlength, method, Lower_bound, min_value)`

This function calculates the extinction spectra (molar absorptivity) of species in a reaction system based on their concentration profiles over time and spectroscopic data. The calculation can be performed using three different methods: **Explicit**, **Implicit**, and **Pseudo-inverse.**

Parameters:

- **`k_vals` (dict):** Dictionary of reaction kinetic constants.
- **`f_deriv` (function):** Function that computes derivatives of concentrations over time, passed to the function `solv_conc_profile` (which is called inside this function).
- **`Conc_0` (dict):** Initial concentrations of the species in the system.
- **`t` (array-like)**: Timepoints over which to compute concentrations profiles.
- **`abs` (DataFrame):** Experimental absorbance data. Each column corresponds to a wavelength, and each row corresponds to a timepoint.
- **`pathlength` (float)**: The optical pathlength of the measurement cell (in cm).
- **`method` (string)**: Method to calculate the extinction spectra for each species. Options:
    - **`"Explicit"`:** Calculates spectra based on explicit assumptions.
    - **`"Implicit"`:** Solves extinction coefficients using a system of equations.
    - **`"Pseudo-inverse"`:** Uses the pseudo-inverse of the concentration profile matrix.
- **`Lower_bound` (boolean):** Determines whether a lower bound is applied or not:
    - If `True`, the lower bound is applied when calculating the spectroscopic species spectra.
    - If `False`, the lower bound is not applied when calculating the spectroscopic species spectra.
- **`min_value` (float):** It is the value used for the lower bound. It is only used when `Lower_bound = True`.

Returns:

- **`result` (DataFrame):** A DataFrame with the calculated extinction spectra (molar absorptivity). The rows correspond to species, and columns correspond to wavelengths.

Notes:

Implicit method: It determines each spectroscopic species spectrum by focusing on the absorbance spectrum and concentration profile at each species' peak concentration, using the most relevant portions of the dataset. This process is illustrated in **Figure 22** which shows how the original absorbance (D) and concentration profile (C) are sliced to retain only the data corresponding to the peaks of each species' concentration (DRed and CRed respectively).

As we have the same number of timepoints (n_t) as number of species (n_x), the concentration profile matrix has become a square matrix, which allows computing its inverse and thus solving for the spectroscopic species spectra matrix (S) in the following manner:

$$S = l^{-1} \cdot CRed^{-1} \cdot DRed \qquad \text{[Eq. 33]}$$

Where CRed and DRed are reduced versions of matrices C and D, whose sizes are n_x x n_x and n_x x n_λ. They are defined in the following way:

$$DRed[i,j] = D[ti,j] \qquad CRed[i,j] = C[ti,j]$$

where ti ∈ [0, n_x - 1] is the index of the time point at which the i-th species reaches its maximum concentration. Thus, the extinction coefficients for the i-th species are given by:

$$\varepsilon_i(\lambda_j) = S[i,j] \qquad \lambda_j \text{ being the j-th wavelength value.}$$

However, Equation 33 has the "drawback" that it requires independence between rows in the concentration profile matrix (CRed), an unlikely feature in this particular scenario unless the kinetic parameters initialized are equal or similar between them.

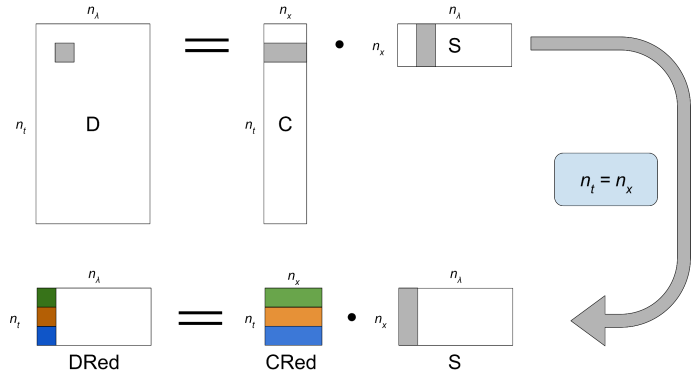


Figure 22: Representation of the Beer-Lambert law in matricial form before and after applying the assumptions of the implicit method. With the implicit method assumptions, observe that in this example n_t=n_x=3. Note that despite not being represented, the product of the matrices C and S must be multiplied by the optical pathlength of the cell. D is the original absorbance, C is the concentration profile, S is the spectroscopic species spectra Dred and CRed are the sliced versions of D and C.

Explicit method We assume that at t=0 only the first species (*C[0,k]= 0 ∀ k≠0*) , and at the final time measured only the last species is present (*C[n_t-1, k]= 0 ∀ k≠n_x-1*). Thus, using the Beer-Lambert law:

$$D[i,j] = l \cdot C[i,k] \cdot S[k,j]$$

Where:

- D[i,j]: are the absorbance at wavelength *j* at time *i*.
- l: is the optical pathlength.
- C[i,k]: are the concentrations for the species *k* at time *i*.
- S[k,j]: are the extinction coefficients for the species *k* at wavelength *j*.

The extinction coefficients for the first and last species can be determined in the following manner:

$$D[0,\ j]\ =\ l\ \cdot\ C[0,0]\ \cdot\ S[0,j] \qquad D[n_t\ -\ 1,\ j]\ =\ l\ \cdot\ C[n_t\ -\ 1, n_x\ -\ 1]\ \cdot\ S[n_x\ -\ 1, j]$$

Therefore, $S[0,j]$ and $S[n_x\ -\ 1, j]$ are:

$$S[0,j]\ =\ \frac{D[0,j]}{l\cdot C[0,0]} \qquad S[n_x\ -\ 1, j]\ =\ \frac{D[n_t-1, j]}{l\cdot C[n_t-1,\ n_x-1]}$$

To calculate the extinction coefficients of the remaining species, we restrict ourselves to values of time for which they reach maximum concentration:

$$SRed\ =\ l^{-1}\cdot CRed^{-1}\ \cdot\ DRed$$

where CRed and DRed are reduced versions of matrices with dimensions n_x- 2 x n_x- 2 and n_x- 2 x n_λ. They are defined in the following way:

$$DRed[i,j]\ =\ D[t_{i+1}, j]\ -\ C[t_{i+1},\ 0]\cdot S[0,j]\ -\ C[t_{i+1},\ nx\ -\ 1]\cdot S[n_x\ -\ 1, j]$$
$$CRed[i,k]\ =\ C[t_{i+1},\ k\ +\ 1]$$

where $t_i \in [1, n_t\text{-}1]$ is the index of the timepoint at which the i-th species reaches its maximum concentration. Thus, spectra of the i-th species are given by:

$$\varepsilon_0(\lambda_j)\ =\ S[0,\ j] \qquad \varepsilon_{n_x-1}(\lambda_j)\ =\ S[n_x\ -\ 1,\ j]$$
$$\varepsilon_i(\lambda_j)\ =\ SRed[i,\ j] \text{ for other values of i}$$

Pseudo-inverse Method: Refer to section 4.2.5 Pseudo-inverse.

5.2.1.11 Model_spectra

`Model_spectra(k_vals, f_deriv, Conc_0, t, abs, pathlength, original_data, method, Lower_bound, min_value, fitting = True)`

This function models spectroscopic data of a system using the concentrations and extinction coefficients of species calculated from kinetic and spectroscopic input data. It employs Lambert-Beer's law to predict absorbance and optionally prepares results for data fitting.

Parameters:

- **`k_vals` (dict):** Dictionary of reaction kinetic constants.
- **`f_deriv` (function):** Function that computes derivatives of concentrations over time, passed to the function `solv_conc_profile` (which is called inside this function).
- **`Conc_0` (dict):** Initial concentrations of the species.
- **`t` (array-like):** timepoints over which to compute concentrations profiles.
- **`abs` (DataFrame):** Absorbance data after denoising.
- **`pathlength` (float):** Optical pathlength of the measurement cell (in cm).
- **`original_data` (DataFrame):** Raw experimental absorbance data.
- **`method` (string):** Method for calculating extinction coefficients (`"Explicit"`, `"Implicit"` or `"Pseudo-inverse"`).
- **`Lower_bound` (boolean):** Determines whether a lower bound is applied or not:

    - If `True`, the lower bound is applied when calculating the spectroscopic species spectra.
    - If `False`, the lower bound is not applied when calculating the spectroscopic species spectra.
- **`min_value` (float):** It is the value used for the lower bound. It is only used when `Lower_bound = True`.
- **`fitting` (bool, optional):** Determines the output of the function:
    - If `True`, returns a flattened array of predicted absorbance values for fitting.
    - If `False`, returns a detailed dictionary of intermediate results.

Returns:

- If `fitting` is `True`: **`sol` (1D array):** Flattened array or predicted absorbance values (for fitting).
- If `fitting` is `False`: **`sol` (dict):** Detailed results containing:
    - **`params` (dict):** Reactions kinetic constants.
    - **`Conc_0` (dict):** Initial species concentrations.
    - **`pathlength` (float):** Optical pathlength.
    - **`n_species` (int):** Number of species in the system.
    - **`D_orig` (DataFrame):** Original experimental absorbance data.
    - **`D_approx` (DataFrame):** Denoised experimental absorbance data.
    - **`D_model` (DataFrame):** Modelled absorbance data.
    - **`C_matrix` (DataFrame):** Concentration matrix.
    - **`S_matrix` (DataFrame):** Extinction coefficient matrix.
    - **`residuals`:** Difference between `D_model` and `D_orig`.
    - **`residual_denoised`:** Difference between `D_model` and `D_approx`.

5.2.1.12 slice_dataset

`slice_dataset(df, t_start=None, t_end=None, wave_start=None, wave_end=None)`

The `slice_dataset` function allows basic slicing of a DataFrame (df) based on both row index (e.g., timepoints) and column labels (e.g., wavelengths). This is useful for quickly slicing the original dataset without the need for prior editing in a spreadsheet program (e.g., Excel or LibreOffice Calc).

Parameters:

- **`df` (DataFrame):** Input dataset in which **row indexes** are expected to be the timepoints, and **columns names** are expected to be the wavelengths.
- **`t_start` (float, optional):** Starting value for row slicing based on the DataFrame index. Default value: `None`.
- **`t_end` (float, optional):** Ending value for row slicing based on the DataFrame index. Default value: `None`.
- **`wave_start` (float, optional):** Starting value for column slicing based on the column labels. Default value: `None`.

  `wave_end` (float, optional): Ending value for column slicing based on the column labels. Default value: `None`.

Returns:

- **`df` (DataFrame):** the sliced version of the input DataFrame, containing only the rows and columns within the specified ranges.

Notes:

- There is a condition that controls whether the function is used or not, which is the boolean variable `Slicing`. If it is checked (`Slicing = True`) and values have been inputted for `t_start`…, a slicing operation will be performed, otherwise it won't.
- The input values are flexible in that if the provided value does not exactly match the row or column names, the function searches for the closest match.

5.2.1.13 create_dynamic_plot

`create_dynamic_plot(df1, df2, Title, x-axis, y-axis, Legend, df1_label, df2_label, width=1200, height=700)`

This function is designed to create an interactive plot using **Bokeh**, a powerful Python library for creating interactive visualizations. This function allows users to visualize data from two DataFrames (`df1` and `df2`) side by side, toggle the visibility of specific series (using a dropdown menu) and legend (using a button) dynamically.

Parameters:

- **`df1` (DataFrame):** Contains the experimental dataset to be plotted. Each column represents a data series.
- **`df2` (DataFrame):** Contains the modelled dataset aligned with **`df1`**. Each column represents a data series.
- **`Title` (string):** Title of the plot.
- **`x_axis` (string):** Label for the x-axis.
- **`y_axis` (string):** Label for the y-axis.
- **`Legend` (string):** Title of the legend.
- **`df1_label` (string):** Label to distinguish `df1` DataFrame.
- **`df2_label` (string):** Label to distinguish `df2` DataFrame.
- **`width` (int, optional):** Width of the plot in pixels. Default is `1200`.
- **`height` (int, optional):** Height of the plot in pixels. Default is `700`.

Returns:

- **`column` (Bokeh layout):** A Bokeh object containing: the plot, the dropdown menu to toggle between data series, and the toggle button controlling the legend visibility.

Notes:

- The dropdown menu contains the column names as labels for ease to identify the series.

5.2.2 Sections

All figures in this section are SCs from the KiPaD Google Colab script, available on GitHub (https://github.com/unizar-flav/KiPaD). The data used in these figures comes from the file BoFPR+60NADPH00071_t.csv, located in the Sample_data folder of the KiPaD GitHub repository. That folder contains data from previously published results (Pérez-Amigot et al., 2019).

5.2.2.1 Environment

In this section, the script loads the libraries, modules, and functions required for the following section of the script to work (**Figure 23**). This section also contains the functions described in section 5.2.1.

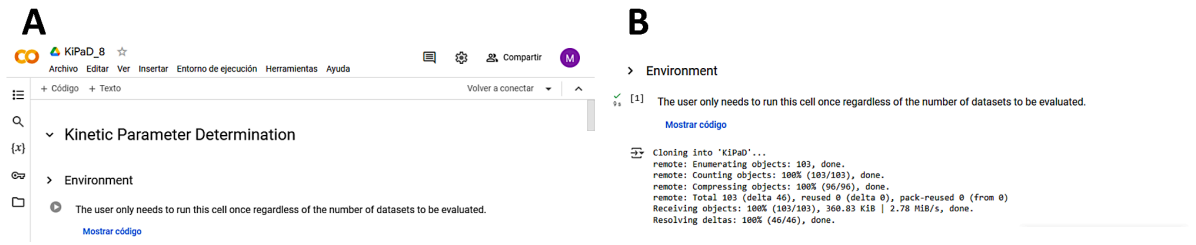


Figure 23: SCs of the **Environment** section. **(A)** Header of the KiPaD script in Google Colab. **(B)** Output of the section when executed.

5.2.2.2 Upload files

After executing this cell, a prompt appears asking the user for the files to be uploaded (**Figure 24**). The files are then read by the function `lee_espectro`, which supports multiple files providing they have distinct timepoints (t_m). For example:

1. A file with timepoints that ranges from 0.01 to 0.1 s in increments of 0.01 s.
2. A file with timepoints that ranges from 1 to 100 s in increments of 1 s.
3. A file with a single timepoint at 0.00 s.

The function `lee_espectro` merges the files into a single DataFrame sorting the data by timepoints in ascending order.

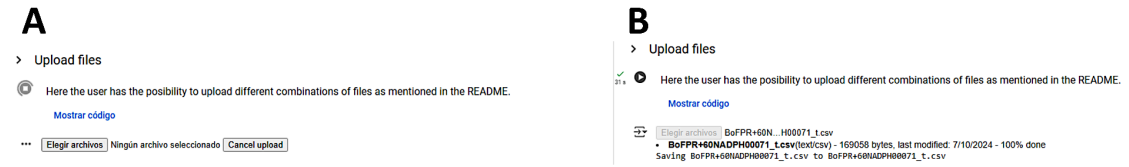


Figure 24: SCs of the **Upload files** section. **(A)** Input SC, once executed, it prompts the user to choose the file to upload. **(B)** Output of the section indicating the names of the files uploaded.

5.2.2.3 Slicing Dataset

This section allows the user to slice the dataset uploaded row and column-wise using the timepoints and wavelengths, respectively, to delimit the sliced dataset. To perform slicing, the user checks the `Slicing` box, and introduces the respective values as needed, which will then be provided to the `slice_dataset` function (**Figure 25-A**). Once executed, this cell will print the resulting DataFrame (**Figure 25-B**).

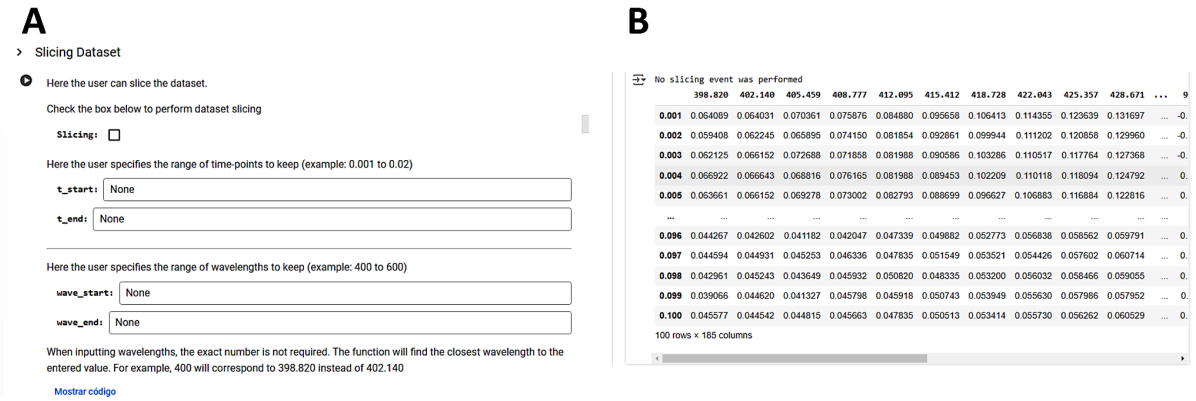


Figure 25: SCs of the **Slicing Dataset** section. **(A)** Input SC, showing four areas for entering the values for slicing the dataset, and the checkbox (boolean) to state whether to perform slicing or not. **(B)** Output of the section showing a message that indicates whether a slicing event occurred and the resulting DataFrame in either case.

5.2.2.4 Spectra plot

Executing this cell will generate two 2D plots: Absorbance *vs* Wavelength and Absorbance *vs* Time. The plots are displayed in tabs with their respective titles and are interactive, allowing zooming via box zoom or wheel zoom. Additionally, the legend visibility can be controlled by the button "Toggle Legend" which is below the plot (**Figure 26**). Users can also download the plots as PNG files. Both plots are generated by the function `create_plot`.

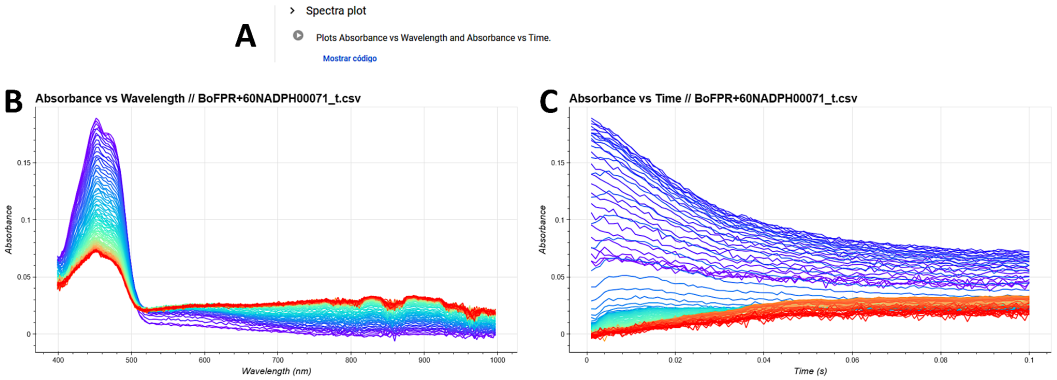


Figure 26: SCs of the **Spectra plot** section. **(A)** Input SC, once run, it creates two tabs containing each one a 2D plot. Outputs of the section: **(B)** Plot of Absorbance *vs* Wavelength and **(C)** Plot of Absorbance *vs* Time.

5.2.2.5 SVD and SSV Identification

In this section, SVD and identification of the SSVs takes place. The identification of the SSVs is estimated using three functions (**Figure 27-A**):

1. `scree_plot_with_fit`**:** Once the singular values are arranged in descending order, the algorithm selects all those that fit to a straight line passing through the first two ones. Points are considered to satisfactorily fit a line if the regression coefficient exceeds or equals a threshold value (default: `scree_plot_th` to `0.9`). It also generates the scree plot (**Fig. 27-B**).
2. `entropy_selection`**:** Evaluates the data's uncertainty explained by singular values. By selecting a threshold (from 0 to 1, default: `entropy_theshold` to `0.9`), the method identifies the smallest number of singular values needed to exceed this level, ensuring the specified uncertainty is explained.

3. `broken_stick_model`: Compares the singular values to a random "broken stick" distribution. SSVs are identified as those exceeding the corresponding values from this distribution, indicating significant components in the data. It also generates its corresponding plot (**Fig. 27-C**).

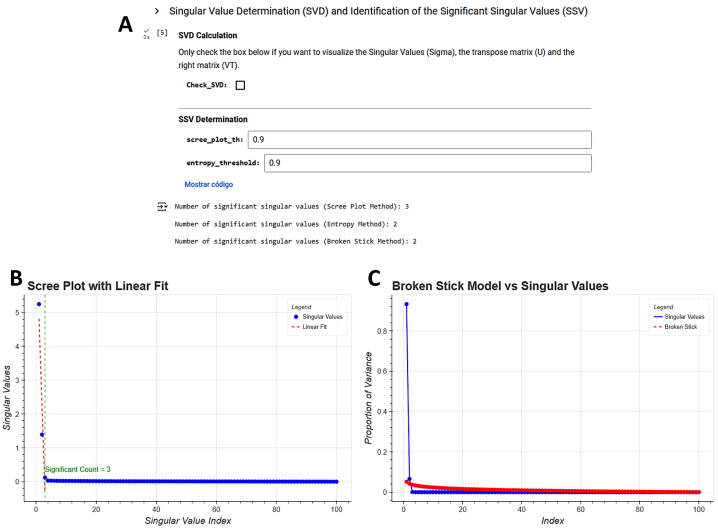


Figure 27: SCs of the **SVD and SSV Identification** section. **(A)** SC, once run, prints the result of each method, and it generates two tabs: one displaying the plot from the `scree_plot_with_fit` function and the other showing the `broken_stick_model`. Outputs of the section: **(B)** Plot of the Scree Plot with Linear Fit and **(C)** Plot of the Broken Stick Model *vs* Singular Values.

5.2.2.6 Dimensionality reduction and Matrix Approximation

In this section, the user is prompted to input the number of SSVs he wishes to select (based on the results from the previous section) (**Figure 28-A**). The script (`matrix_approximation`) will then approximate the original data using only the SSVs, which capture the primary variations in the data, effectively reducing noise (**Figure 28-B**). If the user does NOT want matrix approximation to happen, uncheck the `Answer` box.

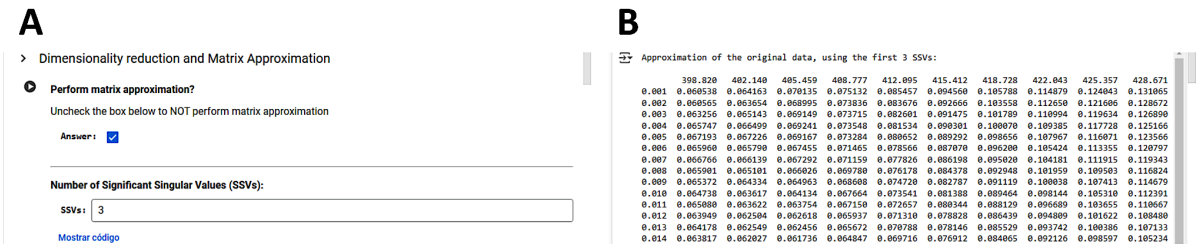


Figure 28: SCs of the **Dimensionality reduction and Matrix Approximation** section. **(A)** Input SC showing a checkbox to determine to perform matrix approximation or not and the number of SSVs used to perform matrix approximation. **(B)** Output of the section showing a message that indicates the number of SSVs used and displaying the approximated matrix using the number of SSVs inputted in **(A)**.

5.2.2.7 Approximated Spectra plot

This step repeats section 5.2.2.4 but uses the *denoised* data, generated in the Dimensionality reduction and Matrix Approximation section. The denoised data are plotted, with smoother lines reflecting reduced noise. Both plots are generated by the function `create_plot` (**Figure 29**).

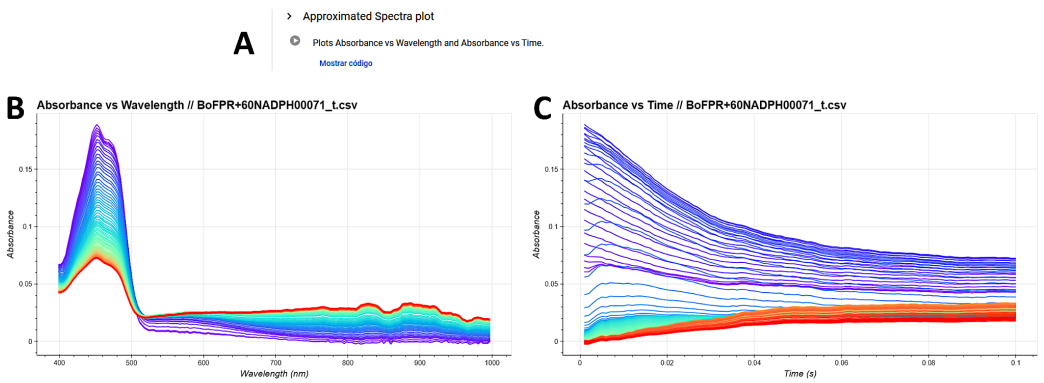


Figure 29: SCs of the **Approximated Spectra plot** section, it uses the denoised data. **(A)** Input SC, once run, it creates two tabs containing each one a 2D plot. Outputs of the section: **(B)** Plot of Absorbance *vs* Wavelength and **(C)** Plot of Absorbance *vs* Time.

5.2.2.8 Reaction Model Parameters

This section allows the user to input the relevant parameters for the proposed reaction model:

- **`n_species`**: number of species (e.g., the number of SSVs).
- **`pathlength`**: optical pathlength of the cuvette used.
- **`Lower_bound`:** Check this box if the user wants to set a lower bound for the spectroscopic species spectra.
- **`min_value`:** Input the value that will act as the lower bound for the spectroscopic species spectra, in the case the `Lower_bound` box is checked. Otherwise, it will do nothing.
- **Initial concentration of the species:** Initial concentrations for each species in the model.
- **Estimated kinetic rates:** Provides estimates for kinetic rates. For parameters to be optimized during fitting, uncheck the `k_fixed` box. This script is currently able to handle this reaction model (or simpler versions of it) (**Figure 30**). However, the script is easily upgradable to handle more species if needed.

These parameters are then fed to the function `procesa`.

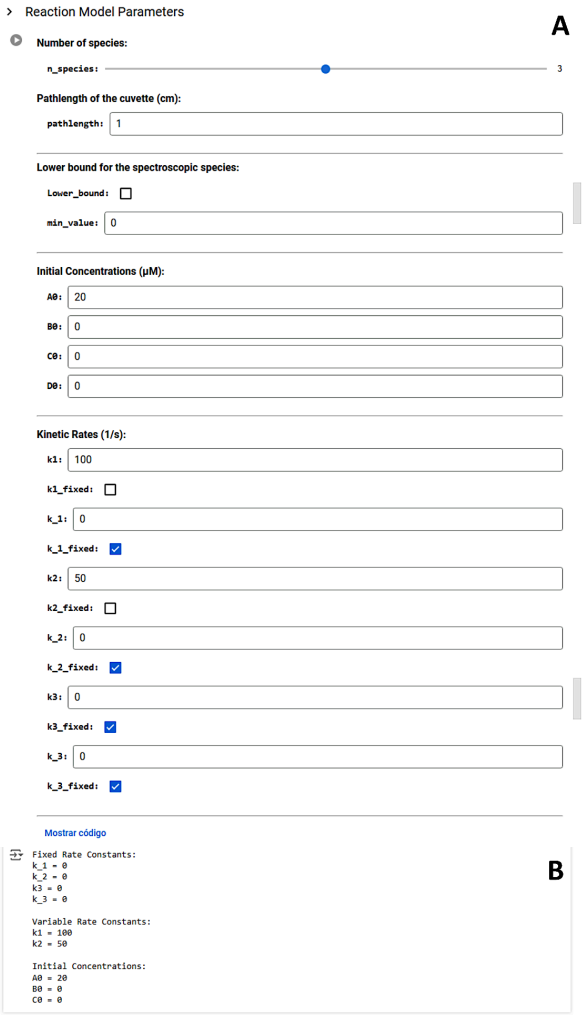


Figure 30: SCs of the **Reaction Model Parameters** section. **(A)** Input SC displaying the fields for entering parameters and specifying whether their values are fixed (not optimized). **(B)** Output: Displays the inputted parameter values and specifies whether each parameter is fixed or variable during the fitting and minimization process.

5.2.2.9 Procesa

This cell calls the function `procesa` from the module funcionesGenerales, with the required arguments inputted, and performs a least-squares minimization through the Levenberg-Marquardt algorithm of the model encoded in the function `Model_spectra`[6]. Before running the cell, the user has to choose their preferred method for estimating the spectroscopic species spectra using a dropdown menu (**Figure 31-A**).

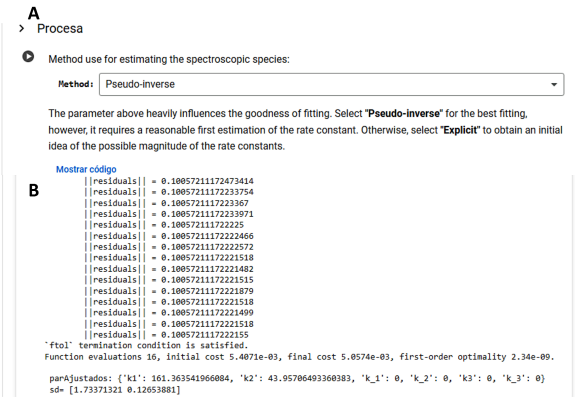


Figure 31: SCs of the **Procesa** section. **(A)** Input SC, showing a dropdown menu where the user selects the method to feed the function `Model_spectra`.**(B)** Output: Prints the residuals at each iteration, and then prints the optimized parameters along with their standard deviation.

- **Pseudo-inverse Method**: Recommended if the user has a rough estimation of the k_{obs} values.
- **Explicit Method**: Recommended if there is no prior information about the k_{obs} values. It provides some starting values that must be included again up in the Parameters section. When reaching this section again, select Pseudo-inverse Method to obtain the final fitting (Beware, this method can only be used if the assumptions for the Explicit method described in `species_spectra` are met).
- **Implicit Method**: It has issues when initializing kinetic parameters with the same value.

The console will print the residuals at each iteration and when it finishes the minimization process it prints the optimized parameters (`parAjustados`), along with their standard deviation (`sd`) (**Figure 31-B**).

5.2.2.10 Plots of Modelled data

This section of the script will plot[7] the modelled data (**Figure 32-A**): Absorbance *vs* Wavelength (**Figure 32-B**), Absorbance *vs* Time (**Figure 32-C**), spectroscopic species spectra (**Figure 32-D**), Concentration Profile (**Figure 32-E**), and **Residual Plots**:

- Absorbance *vs* Wavelength (Original Data): Original - Modelled absorbance (**Figure 32-F**).
- Absorbance *vs* Wavelength (Denoised Data): Denoised - Modelled absorbance (**Figure 32-G**).
- Absorbance *vs* Time (Original Data): Original - Modelled absorbance (**Figure 32-H**).
- Absorbance *vs* Time (Denoised Data): Denoised - Modelled absorbance (**Figure 32-I**).

[6] It depends on these functions: `kinetic_model_matrix`, `deriv_conc`, `solve_conc_profile` and `species_spectra`.

[7] All the plots in this section are generated by the function `create_plot`.

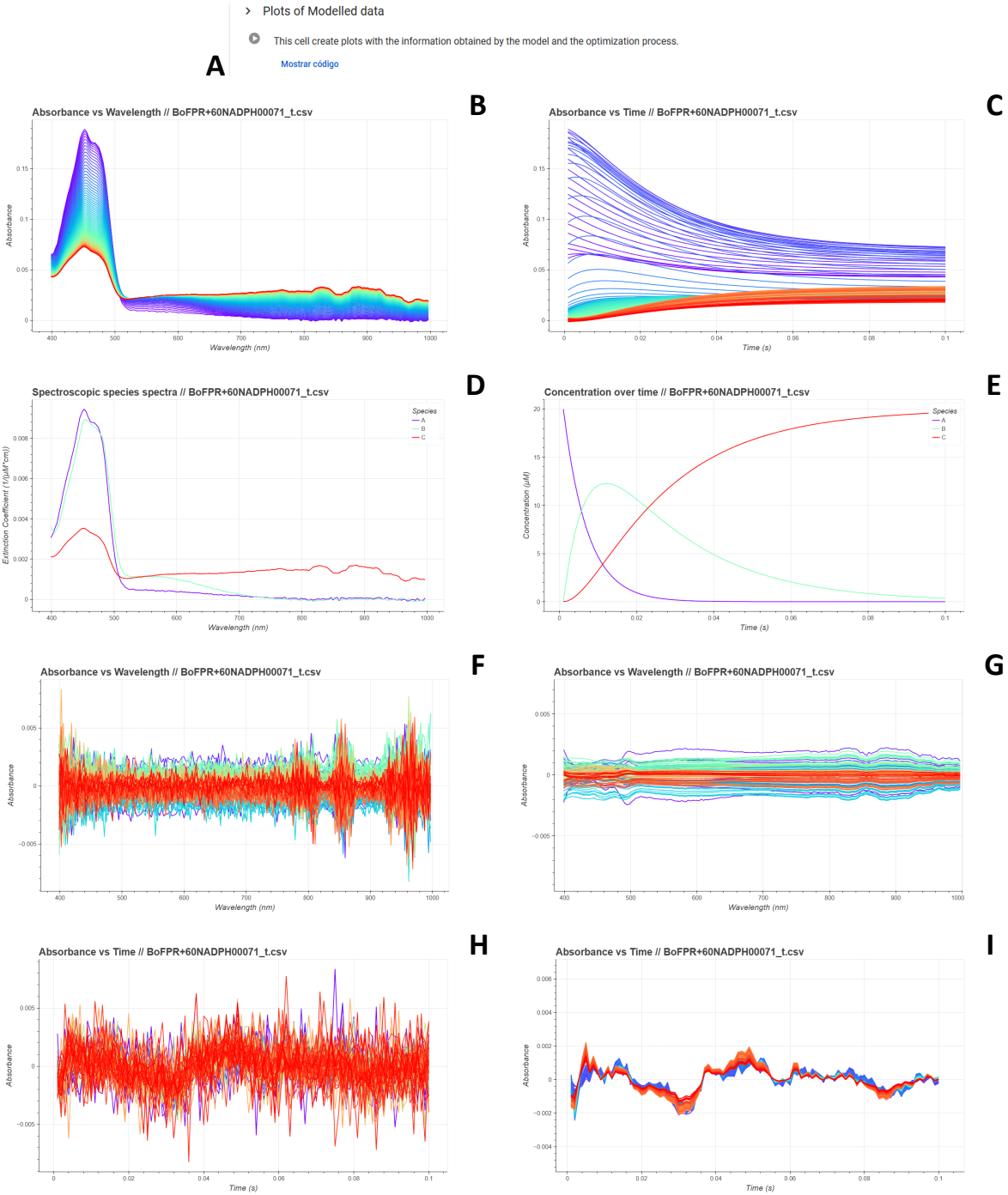


Figure 32: SCs of the **Plots of Modelled data** section, it uses plots the data generated by the model using the optimized parameters and the denoised absorbance data (except in **(F)** and **(H)**). **(A)** Input SC, once run, it creates eight tabs containing each one a 2D plot. Outputs of the section: **(B)** Modelled Absorbance vs. Wavelength plot, **(C)** Modelled Absorbance vs. Time plot, **(D)** Spectroscopic species spectra plot, **(E)** Concentration profile of spectroscopic species plot, **(F)** Residual plot (Original - Modelled Absorbance) for Absorbance vs. Wavelength, **(G)** Residual plot (Denoised - Modelled Absorbance) for Absorbance vs. Wavelength, **(H)** Residual plot (Original - Modelled Absorbance) for Absorbance vs. Time, and **(I)** Residual plot (Denoised - Modelled Absorbance) for Absorbance vs. Time.

5.2.2.11 Modelled and Experimental data comparison

This section plots similar plots to those in sections 5.2.2.4 and 5.2.2.7, however, in this case the plots will display only one series from two different datasets, showing the overlap between Experimental and Modelled data. Moreover, the user can choose between the Original and Denoised Experimental dataset in the `DF1_label` dropdown menu (**Figure 33-A**). Both plots are generated by the function `create_dynamic_plot`. The user may change the series displayed by using the dropdown menu that appears below the plots after executing this cell. This will enable visualizing the fitting of the experimental data by inspecting series by series (**Figures 33-B,C**).

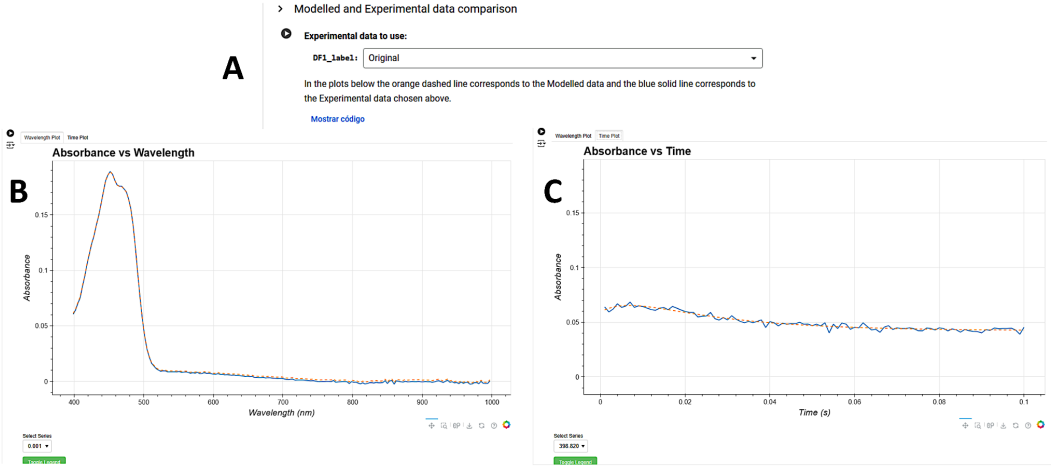


Figure 33: SCs of the **Modelled and Experimental data comparison** section. **(A)** Input SC, showing a dropdown menu where for selecting the experimental data to compare with the model (Original or Denoised). Once executed, it generates two tabs, each containing a 2D plot: Absorbance *vs* Wavelength and Absorbance *vs* Time. Outputs of the section: **(B)** SC displaying the tab with the Absorbance *vs* Wavelength plot (Modelled data as a blue solid line and Experimental data as an orange dashed line) and **(C)** SC displaying the tab with the Absorbance *vs* Time plot (Modelled data as a blue solid line and Experimental data as an orange dashed line).

5.2.2.12 Export results

This final section gathers all the inputted and generated data and saves them as a set of CSV files within a ZIP archive (which has as suffix the date and time of the moment the section was executed) (**Figure 34-A**). List of data is described in section 5.2 KiPaD algorithm (**Figure 34-B**).

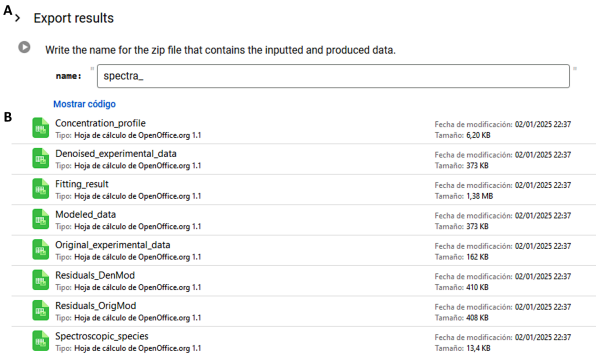


Figure 34: SCs of the **Export results** section. **(A)** Input SC showing the area for entering the name of the ZIP archive exported by this section when executed **(B)** SC of the inside of the ZIP archive generated.

5.3 AKiPa algorithm

AKiPa (Analysis of Kinetic Parameters) is a python script aimed to analyze pre-steady state kinetic data obtained from analysis of stopped-flow data (KiPaD), namely k_{obs} at different ligand/substrate concentrations ([L]), to determine the interaction and kinetic parameters of the observed process. The k_{obs} determined by stopped-flow can be linearly dependent, hyperbolically dependent, or independent on the ligand concentration ([L]). This script focuses mainly in solving the linearly and hyperbolicaly dependent cases as if the k_{obs} are independent of [L], k_{obs} will remain more or less constant regardless of [L] and thus $k_{obs} = k_{lim}$ at any [L].

$$k_{obs} = k_{on} \cdot conc + k_{off} \quad \text{[Eq. 13]} \qquad k_{obs} = \frac{conc \cdot k_{lim}}{conc + K_d} \quad \text{[Eq. 14]}$$

- **Linear fitting (Eq. 13):** it allows determining the ligand association and dissociation rate constants (k_{on} and k_{off} respectively).
- **Hyperbolic fitting (Eq. 14):** it allows determining the dissociation constant (K_d) and the limiting rate constant (k_{lim}).

The script contains 4 original functions, and it is divided into 7 sections.

5.3.1 Functions

5.3.1.1 basic_plot

`basic_plot(df, Title, x_axis, y_axis, width=1000, height=600)`

The `basic_plot` function draws a simpler scatter plot using **Bokeh**. It visualizes data from a DataFrame, displaying one column on the x-axis and another on the y-axis. It provides a quick visual exploration of the data, with the interactivity present in all plots created with Bokeh library.

Parameters

- **`df` (DataFrame):** The input data to be plotted. It assumes that the first column (`df.iloc[:,0]`) corresponds to the x-axis values, and the second column (`df.iloc[:,1]`) corresponds to the y-axis values.
- **`Title` (string)**: The title of the plot.
- **`x_axis` (string):** Label for the x-axis.
- **`y_axis` (string)**: Label for the y-axis.
- **`width` (int, optional)**: Width of the plot in pixels. Default is `1000`.
- **`height` (int, optional)**: Height of the plot in pixels. Default is `600`.

Returns

- **`p` (Bokeh figure)**: A Bokeh figure containing the scatter plot.

5.3.1.2 basic_plot_with_fit

`basic_plot_with_fit(df, y_model, Title, x_axis, y_axis, width=1000, height=600)`

The `basic_plot_with_fit` function generates a scatter plot of experimental data points from a DataFrame and overlays a line plot representing model-predicted data on the same graph. It provides a visual comparison between the experimental and modelled values, making it useful for evaluating the goodness of fit of the model proposed.

Parameters:

- **`df` (DataFrame):** The input data to be plotted. It assumes that the first column (`df.iloc[:,0]`) corresponds to the x-axis values, and the second column (`df.iloc[:,1]`) corresponds to the y-axis values.
- **`y_model` (array-like)**: The model-predicted y-values corresponding to the x-values in the first column of `df`.
- **`Title` (string)**: The title of the plot.
- **`x_axis` (string)**: Label for the x-axis.
- **`y_axis` (string)**: Label for the y-axis.
- **`width` (int, optional)**: Width of the plot in pixels. Default is `1000`.
- **`height` (int, optional)**: Height of the plot in pixels. Default is `600`.

Returns:

- **`p` (Bokeh figure):** A Bokeh figure containing the scatter plot of the experimental data and the line plot of the modeled data.

Notes:

- **Legend Labels:** The scatter plot is labeled as "Experimental data", and the line plot is labeled as "Modelled data" in the legend.
- **Style:** The scatter plot uses blue points, and the line plot uses a red line (experimental and modelled data respectively), making it easy to visually distinguish between data types.

5.3.1.3 linear_model

`linear_model(params, conc)`

The `linear_model` function assumes a linear relationship between a dependent variable (`k_obs`, k_{obs}) and an independent variable (`conc`, [L]) using parameters (`k_on` and `k_off`). The goal is to fit experimental data to a straight-line equation.

Parameters:

- **`params` (dict)**: A dictionary containing the following keys:
  - **`k_on` (float)**: The slope of the linear model, representing the rate of change with respect to `conc`.
  - **`k_off` (float)**: The y-intercept of the linear model.
- **`conc` (array-like or float)**: The independent variable (e.g., [L], ligand concentration) values for which the dependent variable (`k_obs`) is calculated.

Returns

- **`k_obs` (array-like or float)**: The computed dependent values based on the following linear equation: $k_{obs} = k_{on} \cdot [L] + k_{off}$

Notes

- Represents a model where the observed reaction rate (k_{obs}, dependent variable) depends linearly on the concentration [L].

5.3.1.4 hyperbolic_model

`hyperbolic_model(params, conc)`

The `hyperbolic_model` function calculates a hyperbolic relationship between a dependent variable (k_{obs}) and an independent variable (`conc`) using parameters (`Kd` and `k_lim`). The goal is to fit experimental data to a hyperbolic equation.

Parameters

- `params` **(dict)**: A dictionary containing the following keys:
  - `Kd` **(float)**: The dissociation constant ($K_d = k_{off}/k_{on}$), representing the value at which the independent variable (`conc`) at which the dependent variable reaches half of its limiting value.
  - `k_lim` **(float)**: The limiting rate value for the process. Maximum value of the dependent variable.
- `conc` **(array-like or float)**: The independent variable (e.g., [L], ligand concentration) values for which the dependent variable (`k_obs`) is calculated.

Results

- `k_obs` **(array-like or float)**: The computed dependent values based on the following equation: $k_{obs} = \frac{[L] \cdot k_{lim}}{[L] + K_d}$

Notes:

- The function describes a saturating response, where the k_{obs} approach a maximum value (k_{lim}) as the ligand concentration increases (`conc`).
- The function only considers the forward direction of the reaction process, but it can be modified in order for it to accommodate for the reverse reaction, using the following equation: $k_{obs} = \frac{[L] \cdot k_{lim}}{[L] + K_d} + k_{rev}$

5.3.2 Sections

All figures in this section are SCs from the AKiPa Google Colab script, which is available on GitHub (https://github.com/unizar-flav/AKiPa). The data used in these figures comes from the file Simulación datos AKiPa.csv, located in the Sample_data folder of the AKiPa GitHub repository.

5.3.2.1 Environment

In this section, the script loads the libraries, modules, and functions required for the following section of the script to work. This section also contains the functions described in section 5.3.1 (**Figure 35**).

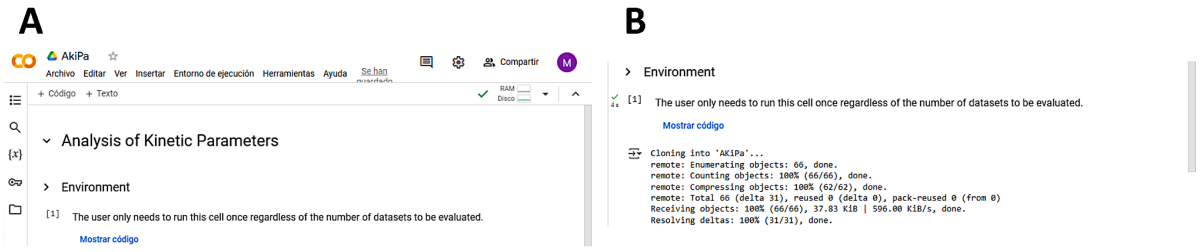


Figure 35: SCs of the **Environment** section. **(A)** Header of the AKiPa script in Google Colab. **(B)** Output of the section when executed.

5.3.2.2 Upload file

After executing this cell, a prompt will appear below asking the user for the CSV file to be uploaded (**Figure 36-A**). The structure of the CSV has to be the one displayed in **Table 3**.

[L] (µM)	k_obs (1/s)
10	115
35	135
…	…

Table 3: Data structure format the AKiPa script expects.

Once the file has been uploaded and read, the console will print a table with the data (**Figure 36-B**).

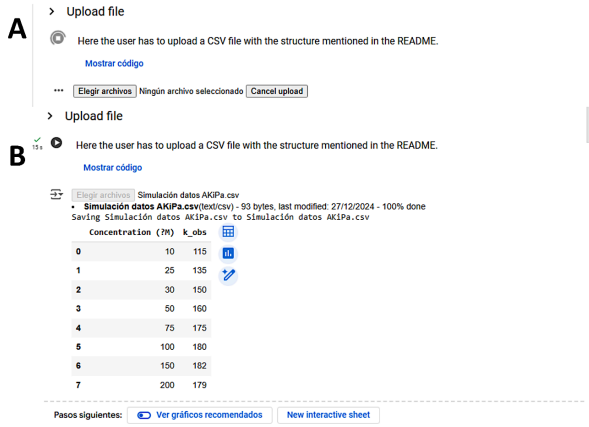


Figure 36: SCs of the **Upload file** section. **(A)** Input SC, once executed, prompts the user to choose the file to upload. **(B)** Output of the section displaying a DataFrame of the file uploaded.

5.3.2.3 Plot experimental data (k_obs)

This section plots the experimental determined k_{obs} values against [L]. The arrangement of data points in the plot provides insight into the appropriate fitting method, such as linear or hyperbolic fitting. The plot is generated by the function `basic_plot` (**Figure 37**).

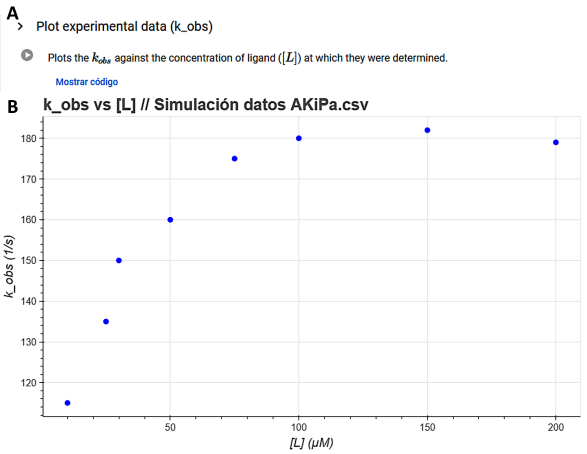


Figure 37: SCs of the **Plot experimental data (k_obs)** section. **(A)** SC of the cell. **(B)** Output: a scatter plot of experimental k_{obs} *vs* [L] showing hyperbolic dependence.

5.3.2.4 Parameters

In this section, the user has to select the model for the fitting (linear or hyperbolic) and then, if needed, initializes the parameters for the model selected (default: all are initialized to 1, to avoid indeterminations) and whether they are to be fixed or not. The parameters are divided into two sections according to the model they belong to (linear or hyperbolic) (**Figure 38-A**). The user only has to fill in the parameters corresponding to the chosen model (parameters corresponding to the other model will be disregarded) (**Figure 38-B**).

These parameters are then fed to `procesa` in the Procesa section.

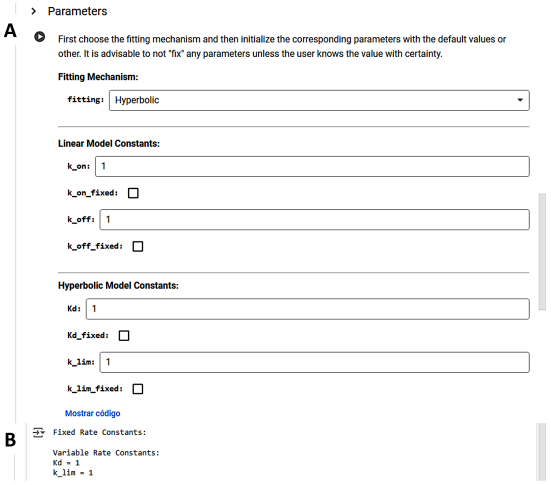


Figure 38: SCs of the **Parameters** section. **(A)** Input SC showing the areas for introducing the parameters and if their values are fixed (not optimized). **(B)** Output: Prints the inputted parameter values and indicates whether each parameter is fixed or variable during the fitting and minimization process.

5.3.2.5 Procesa

This section calls the function `procesa` from the module funcionesGenerales, with the necessary arguments inputted. When executed, it performs a least-squares minimization through the Levenberg-Marquardt algorithm (**Figure 39-A**). The console will print the residuals at each iteration and when it finishes the minimization process it prints the optimized parameters (`parAjustados`), along with their standard deviation (`sd`) (**Figure 39-B**).

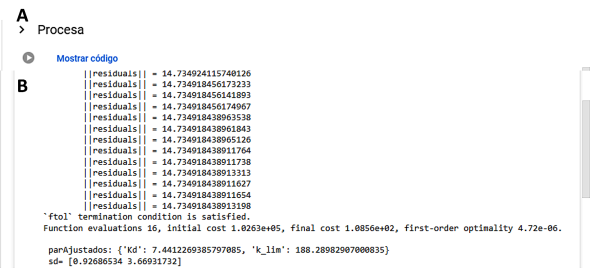


Figure 39: SCs of the **Procesa** section. **(A)** The `procesa` function is fed with the necessary arguments, such as indicated in section 4.1.5 funcionesGenerales. **(B)** Output: Prints the residuals at each iteration, and then prints the optimized parameters along with their standard deviation.

5.3.2.6 Plot experimental data (k_obs) with model fitting

The script plots the experimental k_{obs} at the assayed [L] (blue dots) alongside with the modelled data generated using the optimized parameters from `procesa` (red line). The plot is generated by the function `basic_plot_with_fit` (**Figure 40**).

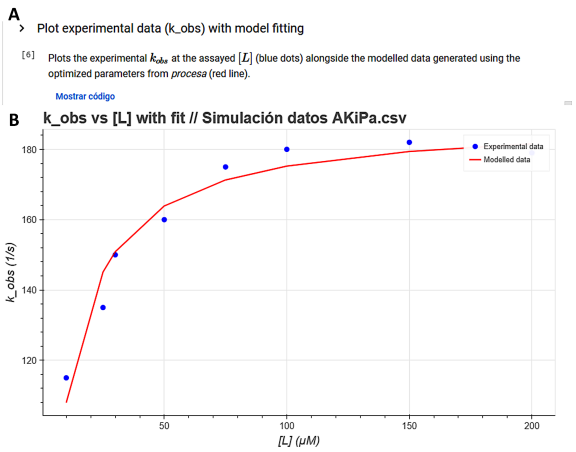


Figure 40: SCs of the **Plot experimental data (k_obs) with model fitting** section. **(A)** SC of the cell. **(B)** Output: a scatter plot of the experimental k_obs *vs* [L] (blue dots) and modelled k_obs *vs* [L] (red line).

5.3.2.7 Export results

This final section gathers all the inputted and generated data and saves them as a set of CSV files, and HTML (plots) within a ZIP archive (which has a suffix the date and time of the moment the section was executed) (**Figure 41**). List of data:

1. Original experimental data
2. Modelled data
3. Plot of the experimental k_{obs} at different ligand concentrations [L] along with the modelled k_{obs} at those same [L]
4. A table with the fitting results that includes the initialized parameters given by the user, their homologous after fitting, the standard deviation of the fitted parameters, the regression coefficient and other parameters informing about the goodness of fit.

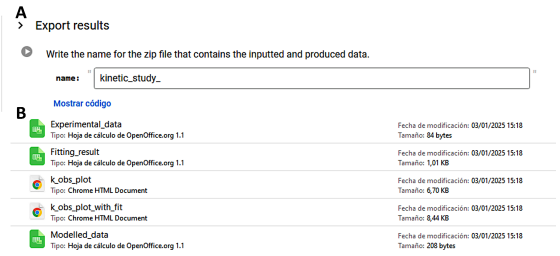


Figure 41: SCs of the **Export results** section. **(A)** Input SC showing the area for introducing the name of the ZIP archive exported by this section when executed **(B)** SC of the inside of the ZIP archive generated.

6. Discussion

The **DAIPProLi algorithm** is designed to analyze data from differential spectroscopy experiments for which there is not specialized software. At the University of Zaragoza, these data are typically analyzed using either Origin or Excel. While both of them are capable of analyzing them, Origin performs much better than Excel. However, Origin is a licensed program with a high installation fee, making it inaccessible to students and unsuitable for use as teaching material. Excel, though also a licensed program, is more affordable and freely available to students. Although, it requires creating custom templates for "rapid" use, and even then, the analysis remains time-consuming and cumbersome, with a less-than-optimal fitting function (Solver). In contrast, the DAIPProLi software, following the Model section (including it), can be tailored to the scientist's specified model. It only requires users to upload the data in the appropriate format and input the parameters. It then performs the analysis automatically and provides the results inside a downloadable ZIP archive. Moreover, as a Google Colab notebook, DAIPProLi is hosted on the unizar-flav GitHub repository, eliminating the need for installation and allowing access from any computer with an internet connection. DAIPProLi has already been tested during the 2024-2025 academic course by the students of the Biophysics course on the data obtained by themselves in the lab, with satisfactory results, outperforming Excel.

The **KiPaD algorithm** is designed to analyze data obtained through multiwavelength time-resolved absorption spectroscopy measurements, using stopped-flow techniques. Currently, there is no publicly available open-source software for such analysis. Licensed programs like Pro-Kinetist are available but come with significant drawbacks. These programs are often cryptic about their mathematical framework, compatible only with equipment from the same manufacturer due to proprietary data formats, and expensive, making them inaccessible to students. As a result, research groups with equipment from different manufacturers are forced to use multiple software programs, complicating data analysis. In light of these issues, KiPaD offers a user-friendly, open-source alternative with a transparent mathematical framework and data handling. KiPaD has been tested using data from previous published results (Pérez-Amigot et al., 2019), displaying a similar numerical performance than the proprietary software in the article aforementioned. KiPaD also offers better noise reduction options and different fitting methods.

The **AKiPa algorithm** is designed to analyze the observed kinetic rates obtained from the analysis of the data treated by KiPaD when using different concentrations of the substrate. There is no specialized software for this task, as the analysis can be done with Origin and Excel (with an optimal fitting). Nevertheless, AKiPa's advantages over alternative methods lie in its open-source nature and its ability to analyze data without prior adjustments. Users simply upload the file, input the parameters, and the analysis is done automatically. AKiPa has only been tested with simulated data, showing satisfactory results.

All these scripts have not only scientific interest for researchers carrying out differential spectroscopy or stopped-flow measurements, but they will also become of academic relevance. The DAIPProLi software is intended to be easily translated to a programming practicum for the students of the new course in Informatics that will shortly run in the Biotechnology degree program at the University of Zaragoza.

In addition, all these scripts will be used from academic year 2025-2026 in the course of Biophysics of the degree in Biotechnology at UNIZAR. The DAIPProLi software for analyzing experimental data obtained by the students in the lab. The KiPaD and AKiPa software are used to teach students about stopped-flow methodologies by helping them better understand the outputs of these experiments through the analysis of data provided in their lectures.

7. Conclusions

This master's project provides three freely available scripts relevant for analyzing experimental data to determine affinity and kinetic parameters, as well as for academic use in several subjects of the Biotechnology degree at the University of Zaragoza. All of them are hosted at the unizar-flav GitHub.

DAIPProLi is an open-source and user-friendly software for analyzing data from differential spectroscopy experiments. It determines the dissociation constant (K_d) and the extinction coefficient of the spectroscopic change ($\Delta\varepsilon$). Its simple and straightforward code makes it ideal for programming practicum, while providing a specialized and efficient solution for these analyses to researchers in the lab.

KiPaD is an open-source and user-friendly software for analyzing data obtained from multiwavelength time-resolved absorption spectroscopy using stopped-flow. It determines observed rate constants (k_{obs}), and spectroscopic properties of intermediate species. KiPaD offers an open-source alternative to licensed programs, with the added benefit of being much more transparent about its inner workings.

AKiPa is an open-source and user-friendly software for evaluating the dependence of observed rate constants (k_{obs}) of an enzymatic reaction on the substrate concentrations, allowing to determine kinetic parameters of the process observed. As DAIPProLi and KiPaD, AKiPa offers a specialized and fast solution for data analysis.

Bibliography

Bhatt, D., 2023. Singular Value Decomposition (SVD): A Powerful Matrix Factorization Technique. Medium. URL https://medium.com/@dbhatt245/singular-value-decomposition-svd-a-powerful-matrix-factorization-technique-19ab1690472d (accessed 9.3.24).

Bokeh documentation [WWW Document], 2024. . Bokeh. URL https://docs.bokeh.org/en/latest/index.html (accessed 12.2.24).

Brownlee, J., 2019. How to Calculate the SVD from Scratch with Python [WWW Document]. Mach. Learn. Mastery. URL https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/ (accessed 8.12.24).

Bunaciu, A.A., Hoang, V.D., Aboul-Enein, H.Y., 2013. Applications of Differential Spectrophotometry in Analytical Chemistry. Crit. Rev. Anal. Chem. 43, 125–130. https://doi.org/10.1080/10408347.2013.803357

Cangelosi, R., Goriely, A., 2007. Component retention in principal component analysis with application to cDNA microarray data. Biol. Direct 2, 2.

https://doi.org/10.1186/1745-6150-2-2
Chemistry (IUPAC), T.I.U. of P. and A., n.d. IUPAC - isosbestic point (I03310) [WWW Document]. https://doi.org/10.1351/goldbook.I03310
Cochran, R.N., Horne, F.H., Dye, J.L., Ceraso, J., Suelter, C.H., 1980. Principal component analysis of rapid scanning wavelength stopped-flow kinetics experiments on the liver alcohol dehydrogenase catalyzed reduction of p-nitroso-N,N-dimethylaniline by 1,4-dihydronicotinamide adenine dinucleotide. J. Phys. Chem. 84, 2567–2575. https://doi.org/10.1021/j100457a017
csv — CSV File Reading and Writing [WWW Document], 2024. . Python Doc. URL https://docs.python.org/3/library/csv.html (accessed 12.2.24).
datetime — Basic date and time types [WWW Document], 2024. . Python Doc. URL https://docs.python.org/3/library/datetime.html (accessed 12.3.24).
Definition of OBJECT-ORIENTED PROGRAMMING [WWW Document], n.d. URL https://www.merriam-webster.com/dictionary/object-oriented+programming (accessed 12.2.24).
Dynamic typing - MDN Web Docs Glossary: Definitions of Web-related terms | MDN [WWW Document], 2023. URL https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing (accessed 12.2.24).
Falini, A., 2022. A review on the selection criteria for the truncated SVD in Data Science applications. J. Comput. Math. Data Sci. 5, 100064. https://doi.org/10.1016/j.jcmds.2022.100064
Ferreira, P., Medina, M., 2021. Anaerobic Stopped-Flow Spectrophotometry with Photodiode Array Detection in the Presteady State: An Application to Elucidate Oxidoreduction Mechanisms in Flavoproteins, in: Barile, M. (Ed.), Flavins and Flavoproteins: Methods and Protocols. Springer US, New York, NY, pp. 135–155. https://doi.org/10.1007/978-1-0716-1286-6_9
Gómez-Moreno Sanz, C., Sancho Sanz, J., 2003. Estructura de proteínas., Ariel Ciencia. Ariel, Barcelona.
Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2
Hunter, J.D., 2007. Matplotlib: A 2D Graphics Environment. Comput. Sci. Eng. 9, 90–95. https://doi.org/10.1109/MCSE.2007.55
Johnson, K.A., 1992. 1 Transient-State Kinetic Analysis of Enzyme Reaction Pathways, in: Sigman, D.S. (Ed.), The Enzymes. Academic Press, pp. 1–61. https://doi.org/10.1016/S1874-6047(08)60019-0
Kundu, J., Kar, U., Gautam, S., Karmakar, S., Chowdhury, P.K., 2015. Unusual effects of crowders on heme retention in myoglobin. FEBS Lett. 589, 3807–3815. https://doi.org/10.1016/j.febslet.2015.11.015
LinearRegression [WWW Document], 2024. . Scikit-Learn. URL https://scikit-learn/stable/modules/generated/sklearn.linear_model.LinearRegression.html (accessed 12.3.24).
Marquardt, D.W., 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. J. Soc. Ind. Appl. Math. 11, 431–441. https://doi.org/10.1137/0111030
Martínez-Júlvez, M., Nogués, I., Faro, M., Hurley, J.K., Brodie, T.B., Mayoral, T., Sanz-Aparicio, J., Hermoso, J.A., Stankovich, M.T., Medina, M., Tollin, G., Gómez-Moreno, C., 2001. Role of a Cluster of Hydrophobic Residues Near the FAD Cofactor in Anabaena PCC 7119 Ferredoxin-NADP+Reductase for Optimal Complex Formation and Electron Transfer to Ferredoxin. J. Biol. Chem. 276, 27498–27510. https://doi.org/10.1074/jbc.M102112200
McKinney, W., 2010. Data Structures for Statistical Computing in Python. scipy. https://doi.org/10.25080/Majora-92bf1922-00a

Medina, M., Luquita, A., Tejero, J., Hermoso, J., Mayoral, T., Sanz-Aparicio, J., Grever, K., Gómez-Moreno, C., 2001. Probing the Determinants of Coenzyme Specificity in Ferredoxin-NADP+ Reductase by Site-directed Mutagenesis. J. Biol. Chem. 276, 11902–11912. https://doi.org/10.1074/jbc.M009287200

Medina, M., Martinez-Júlvez, M., Hurley, J.K., Tollin, G., Gómez-Moreno, C., 1998. Involvement of glutamic acid 301 in the catalytic mechanism of ferredoxin-NADP+ reductase from Anabaena PCC 7119. Biochemistry 37, 2715–2728. https://doi.org/10.1021/bi971795y

mpl_toolkits.mplot3d.axes3d.Axes3D — Matplotlib 3.9.3 documentation [WWW Document], 2024. URL https://matplotlib.org/stable/api/toolkits/mplot3d/axes3d.html#mpl_toolkits.mplot3d.axes3d.Axes3D (accessed 12.3.24).

Murat, 2023. Decoding the Debate: Interpreted vs. Compiled Programming Languages. Medium. URL https://muratakkan.medium.com/decoding-the-debate-interpreted-vs-compiled-programming-languages-b5551c2f0770 (accessed 12.2.24).

numpy.linalg.pinv — NumPy v2.1 Manual [WWW Document], n.d. URL https://numpy.org/doc/2.1/reference/generated/numpy.linalg.pinv.html (accessed 12.27.24).

Pérez-Amigot, D., Taleb, V., Boneta, S., Anoz-Carbonell, E., Sebastián, M., Velázquez-Campoy, A., Polo, V., Martínez-Júlvez, M., Medina, M., 2019. Towards the competent conformation for catalysis in the ferredoxin-NADP+ reductase from the *Brucella ovis* pathogen. Biochim. Biophys. Acta BBA - Bioenerg. 1860, 148058. https://doi.org/10.1016/j.bbabio.2019.148058

Piubelli, L., Aliverti, A., Arakaki, A.K., Carrillo, N., Ceccarelli, E.A., Karplus, P.A., Zanetti, G., 2000. Competition between C-terminal Tyrosine and Nicotinamide Modulates Pyridine Nucleotide Affinity and Specificity in Plant Ferredoxin-NADP+ Reductase *. J. Biol. Chem. 275, 10472–10476. https://doi.org/10.1074/jbc.275.14.10472

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992. Numerical Recipes in C: The Art of Scientific Computing, 2nd ed. Cambridge University Press.

Rallabandi, S., 2023. Moore-Penrose Pseudo-Inverse: The Genius in Simplicity. Medium. URL https://medium.com/@sreeku.ralla/moore-penrose-pseudo-inverse-the-genius-in-simplicity-887cf2872b61 (accessed 12.27.24).

Sancho, J., Gómez-Moreno, C., 1991. Interaction of ferredoxin-NADP+ reductase from *Anabaena* with its substrates. Arch. Biochem. Biophys. 288, 231–238. https://doi.org/10.1016/0003-9861(91)90189-P

Sørlie, M., Seefeldt, L.C., Parker, V.D., 2000. Use of Stopped-Flow Spectrophotometry to Establish Midpoint Potentials for Redox Proteins. Anal. Biochem. 287, 118–125. https://doi.org/10.1006/abio.2000.4826

svd — SciPy v1.14.1 Manual [WWW Document], 2024. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html (accessed 12.2.24).

The Python Tutorial [WWW Document], n.d. . Python Doc. URL https://docs.python.org/3/tutorial/index.html (accessed 12.2.24).

Vanek, T., Kohli, A., 2024. Biochemistry, Myoglobin, in: StatPearls. StatPearls Publishing, Treasure Island (FL).

zipfile — Work with ZIP archives [WWW Document], 2024. . Python Doc. URL https://docs.python.org/3/library/zipfile.html (accessed 12.2.24).