1. SVD assisted global analysis of time-resolved spectroscopic data

This chapter aims to briefly acquaint the reader with some aspects of time-resolved spectroscopy. In addition, a short overview is given on what a global fit is, how an SVD works and how it can assist the fitting procedure.

1.1. On the concept of time-resolved spectroscopy for slow and fast dynamics

Time-resolved spectroscopy is a measurement method commonly used to study processes occurring in a material as it settles from an out-of-equilibrium state back to its ground-state. By measuring and analysing the temporal behaviour of (optical) properties of such a material sample, e.g. its absorption coefficient $\alpha_{sample}(\lambda)$, one can infer information about the processes by which the material resettles.
As the dynamics of these processes go from slow to fast, and thus the temporal resolution has to increase, there is an important conceptual shift in the general time-resolved spectroscopy experiment. In the following, this shift is illustrated by detailing two different measurement setups :

- *slow* dynamics: the sample is illuminated with a probe light at all times. The camera measures the intensity of the transmitted probe light e.g. every $1\,ms$. At some time t_0 the sample is put into the out-of-equilibrium state (e.g. with a laser pulse). The camera then takes further measurements until the sample is in equilibrium again. Thus one receives a set of measurements of $\alpha_{sample}(\lambda)$ at different times t before and after the excitation. The set of measurements $\alpha_{sample}(\lambda, t)$ is called a time-resolved spectrum.

However, if the dynamics of the processes one wants to study are faster than $\mathcal{O}$(ns), this setup starts to fail, as conventional camera electronics may not be able to keep up with the required temporal resolution. Thus, if one would e.g. like to study molecular vibrations or electron transfer processes, whose time scales may be of the order of (fs-ps), then the experiment setup needs to be adapted (Bouduban 2019):

- *fast* dynamics: the sample is illuminated with the probe light only at certain times for a short period. (I.e. in pulses.) The camera, though, is constantly switched on and measures the transmitted intensities. When the sample is in equilibrium, the measured $\alpha_{sample}(\lambda)$ should not change (apart from fluctuations) no matter the time of incidence of the probe pulse on the sample. But, if again at some time t_0 the material is put into the out-of-equilibrium state (again e.g. via a laser pulse), then the measured $\alpha_{sample}(\lambda)$ depends on the time delay t_{delay} between t_0 and the time of incidence of the probe light. To get a time-resolved spectrum, one would repeatedly conduct the procedure of excitation and measurement of $\alpha_{sample}(\lambda)$, but at each iteration with a different t_{delay}. (One must observe that the sample is reset to its ground state before the renewed excitation and measurement of $\alpha_{sample}(\lambda)$. Else one can not assume that the measurements are independent of each other. Thus one should also ensure that the sample is neither permanently damaged by the probe or the excitation pulse. Also, the excitation and probe pulse need to be stable enough, so that the different measurements are comparable.)

In jargon, the above described procedure for measuring fast dynamics is often called *pump-probe*-spectroscopy.[1]
It is a common method to investigate processes occurring on very fast time scales (up to $\mathcal{O}$(fs) or even faster). The name is fitting, as the pulse at t_0 "pumps" energy into the material, intended to lead to excitation from its equilibrium state.
In a time-resolved spectrum of such a measurement one might not just record the probe intensities as transmitted by the excited sample. Instead, often, the probe transmission of the unexcited sample is subtracted, so that

$$\alpha_{sample}(\lambda, t) = \alpha_{pumped\,sample}(\lambda, t) - \alpha_{non-pumped\,sample}(\lambda, t). \tag{1}$$

[1]It could of course also be applied to slow dynamics.

Thus one gets a time-resolved spectrum containing the transmitted intensities as deviations from the transmission of the unexcited sample. Given such a measurement, one might be able to e.g. identify the main excited states (and their lifetimes) involved in the relaxation process of a material. More on that and also some more aspects on *pump-probe*-spectroscopy will be detailed in the following sections.

1.2. On pump-probe spectroscopy

There are (at least) two pulses used in a pump-probe-spectroscopy measurement: the pump pulse and the probe pulse. The first is intended to excite the material. To ascertain that the sample will be sufficiently excited, one usually measures the absorption spectrum of the unexcited material. Using that, a suitable mean wavelength λ_0 for the pump pulse can be determined.
As mentioned above, the goal of time-resolved spectroscopy is often to identify the main excited states (and their lifetimes) involved in the relaxation process of a material. This gets more and more complex the larger the number of states included. Therefore one usually wants a narrow spectral width of the pump pulse, as one intends to populate only a selected excited state in the sample. However, as the spectral width and the temporal width of a laser pulse anti-correlate, one often has to compromise between the two, as the temporal width of the pump pulse influences the temporal resolution of the measurement.
In contrast, the probe pulse should have a broad spectral range, because the processes triggered by the excitation might lead to an array of newly available energy states, which influence the absorption of the sample at different wavelengths.

As stated, the pump pulse is intended to increase the population of an excited state in the material. As this population decays back to its ground state level, a cascade of other populations (in states which are not (significantly) populated in the material when it is in equilibrium) can be triggered. See fig. 1.1 for an idealised illustration of a most simple such cascade.

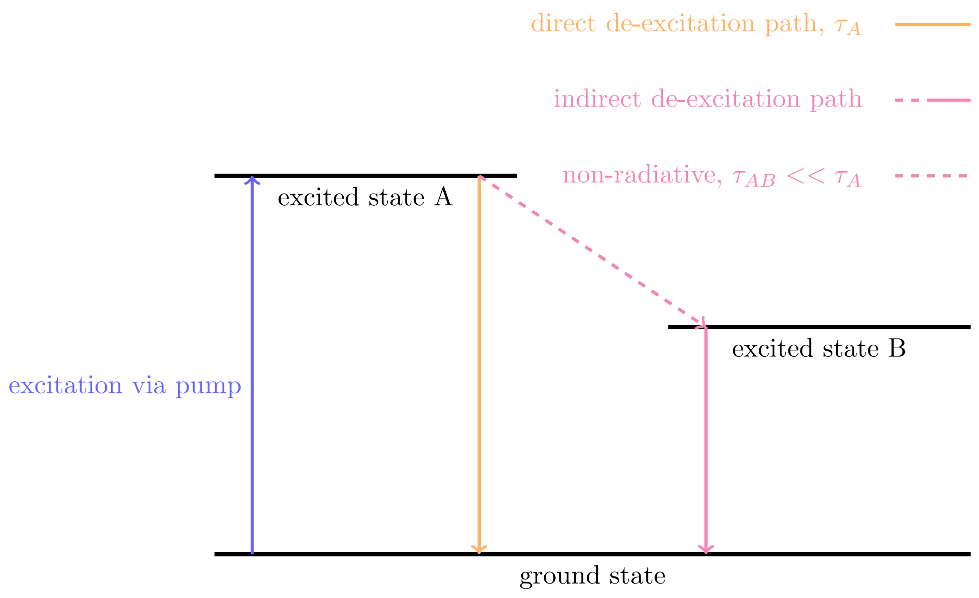

direct de-excitation path, τ_A
indirect de-excitation path
non-radiative, $\tau_{AB} << \tau_A$
excited state A
excited state B
excitation via pump
ground state

Figure 1.1 – Diagram of an idealised simple energy cascade in material that gets excited via a pump. Excited state B gets populated only through decays from excited state A.

If the energy levels in the material sample were as discrete as they are depicted in fig. 1.1, then the identification of the involved excited states would be rather straightforward: given that the probe pulse included both the wavelengths corresponding to the excited stated energies $\Delta E_A = E_A - E_{ground\,state}$ and

$\Delta E_B = E_B - E_{ground\,state}$, one would see very narrow (or even discrete) troughs in the time-resolved spectrum at these wavelengths. These troughs would correspond to stimulated emission signals from state A and B.
An estimate of τ_{AB} could then also be achieved: assuming, as is indicated in fig. 1.1, that the direct de-excitation from state A is slow compared to the non-radiative relaxation from A to B, then the troughs at $\lambda_A = \frac{h \cdot c}{\Delta E_A}$ in the measured $\alpha_{sample}(\lambda, t)$-spectrum could be fitted with an exponential to estimate τ_{AB}.

However, in samples that do not consist of a single molecule, there are generally more and no longer discrete energy levels present, which may also overlap. The identification of the principal involved states and the successive estimation of their lifetimes then becomes more complex.
This is because the contributions in a time-resolved-spectrum are then more difficult to separate from each other and assigning them to certain states becomes an intricate problem. The identification of the major processes now might require additional experiments as well as a more complex analysis of the data. The developed software can be used in such an analysis to, at first identify the number of relevant time constants and then estimate the values of these time constants with which the major populations accumulate or decay as the material resettles to its ground state. This is achieved by an inspection of the data's SVD, as well as a non-linear least-squares global fit.

1.3. What is a singular value decomposition, how it can be used in signal processing

In linear algebra, the SVD procedure is a generalisation of the eigenvalue decomposition. While the latter only works for diagonalisable, and therefore square ($n \times n$), matrices, the SVD is applicable to any matrix. Given a matrix M, one can decompose it as $M = U \times \Sigma \times V^*$, where $U \times \Sigma \times V^*$ constitutes the SVD. If M is $\epsilon\ \mathbb{R}$, it is guaranteed that U and V are also real orthogonal matrices and thus $V^* = V^T$. (Where $*$ and T represent complex conjugate and transpose respectively.)

Additionally, if M is a $m \times n$ matrix, then

- Σ is a $m \times n$ rectangular diagonal matrix, i.e. all it's entries σ_{ij} with $i \neq j$ are zero. The diagonal entries σ_{ii} (or σ_i) represent the *singular values* of matrix M. The singular values are real and non-negative numbers.
- U is a $m \times m$ orthogonal matrix. The columns of U make up the *left singular vectors* of M.
- V is a $n \times n$ orthogonal matrix. The columns of V, and thus the rows of V^T, make up the *right singular vectors* of M.
- the rank of M can be determined by counting the number of non-zero singular values. (However, when calculating the SVD via a computer, one does usually not receive (singular) values that are truthfully zero. This is due to the finite precision of computations using *floats*. Thus one often has to set a cut-off below which one treats a (singular) value as zero.)
- it is possible to choose the decomposition such that the singular values in Σ are in descending order.

Now a brief excursion to physics: As (van Stokkum, Larsen, and van Grondelle 2004) have described it, "the spectroscopic properties of a mixture of components are a superposition of the spectroscopic properties of the components weighted by their concentration". (The reader could equate the here mentioned *components* with the *populations* of excited states from section 1.2.) Thus they conclude that a noise-free time-resolved spectrum can be represented as a superposition of the spectral contributions of the n_{comp} different components

$$\psi(\lambda, t_{delay}) = \sum_{i}^{n_{comp}} \varepsilon_i(\lambda)\, c_i(t_{delay}) \tag{2}$$

where $c_i(t_{delay})$ and $\varepsilon_i(\lambda)$ denote, respectively, the concentration (at time delay t_{delay}) and spectrum of component i. (One should consider that equation (2) assumes that a complete separation of time and wavelength properties is possible. I.e. that, as time moves on, the spectral contribution of each component (or population) only concerns it's amplitude (Cannizzo et al. 2008).)
For a real measurement, resulting in data matrix $\Psi(\lambda, t_{delay})$, equation (2) can be amended so that it now also includes noise contributions Ξ:

$$\Psi(\lambda, t_{delay}) = \psi(\lambda, t_{delay}) + \Xi(\lambda, t_{delay}) = \sum_i^{n_{comp}} \varepsilon_i(\lambda)\, c_i(t_{delay}) + \Xi(\lambda, t_{delay}). \tag{3}$$

While one would want to only measure the *signal* contribution $\psi(\lambda, t_{delay})$, undisturbed by any noise, so that one could also make undisturbed inferences about the components/populations within the excited sample, reality constricts us to measuring $\Psi(\lambda, t_{delay})$. However, another aspect of the SVD can help to separate the noise contributions from the signal.

And with that, back to linear algebra: Using the *outer product*, one can also compartmentalise a matrix M as

$$M = \sum_i A_i = \sum_i \sigma_i \cdot U_i \otimes V_i, \tag{4}$$

where σ_i, U_i and V_i are the singular values and left and right singular vectors respectively, and where the A_i are matrices computed from σ_i, U_i and V_i. This compartmentalisation of M is common in signal processing, as the inspection of its constituents (A_i, σ_i, U_i and V_i) allows to categorise which singular values and vectors contain signal information and which correspond to measurement noise. (Assuming that the measurement noise is only stochastic normally-distributed noise with a mean value of zero and, compared to the signal, small amplitudes, the A_i, U_i and V_i of the noise contributions should not display any meaningful structure and their corresponding singular values should be small. Both of these facts would enable the skilled analyst to distinguish them from signal contributions. This may, however, no longer be the case when the measurement also contains systematic noise, which might exhibit well defined structures with large corresponding singular values. In that case the categorisation into signal components and noise components can be difficult.)
By considering only the n_{comp} (estimated) signal components one can reconstruct a reduced matrix M_{SVD}:

$$M_{SVD} = \sum_i^{n_{comp}} A_i = \sum_i^{n_{comp}} \sigma_i \cdot U_i \otimes V_i. \tag{5}$$

M_{SVD} should, if one has identified a sufficient (and correct) set of singular values and vectors, contain all the relevant signal data, but overall lack the measurement noise contribution. Thus, the inspection of the differences between the reduced data matrix Ψ_{SVD} and the complete data matrix Ψ, tells one whether the selected set of components needs to be adjusted. Ideally, the difference matrix $\Delta\Psi_{SVD}$ no longer displays any significant structure. (Or, if large non-normally distributed noise components have been identified, the remaining structures in $\Delta\Psi_{SVD}$ can be fully assigned to those components.) Thus the SVD can be utilised to identify the number of signal components. This number is then taken as the number of independently resolvable decay-time-constants in a global fit (Ruckebusch et al. 2012).

Another aspect one should note is the similarity of the equations (2) and (5). Assuming now that the noise is small, this allows us to project the data on to the first n_{comp} vectors U_i and V_i^T, which should then contain the "decoupled" spectral and temporal contributions of signal component i (van Stokkum, Larsen, and van Grondelle 2004).
Like this one saves computational resources in a global fit, where one fits the selected right singular vectors V_i to both gain information about the time scales of the the major populations in the excited sample and

then uses the selected U_i to reconstruct the population specific impact on the change in absorption (or other measurement variable) of the sample.

(In the software, the python method *scipy.linalg.svd* (The-SciPy-community 2021) is used to compute the SVD of data matrices.)

1.4. The SVD assisted global fit

As mentioned, usually, the goal of the analysis of time-resolved spectroscopic data is the identification and parametrisation of the major processes occurring in the excited material as it regresses to it's initial equilibrium state. The final result of the analysis would then be a model of the complete de-excitation mechanism. As this can be a complex task, one typically proceeds in three steps (Ruckebusch et al. 2012):

1. exploratory analysis: to get a first insight about the number of populations involved in the de-excitation process and their respective time-scales. Often conducted as a *global analysis* with minimal assumptions. (If sufficient information about the correct model is already present, this step may be skipped.)
2. postulating and testing photo-physical models of the whole de-excitation mechanism. Based on the insights gained in step 1 and possibly other measurements or theory. This is termed *target analysis.*
3. model validation to find the model(s) that best describe(s) the data.

This way, a skilled analyst may be able to adequately unravel the de-excitation processes induced in the material of interest. In the following, steps 1 and 2 and their implementation in the software will be further illuminated.

By *global analysis* one often refers to the simultaneous analysis of all measurements (van Stokkum, Larsen, and van Grondelle 2004). Thus a *global fit* of time-resolved spectroscopic data could be interpreted as some sort of fit using the whole data matrix of section 1.2 all at once. However, in the *SVD assisted global fit* one utilises the SVD to reduce the complexity of the fit by separating what one judges to be the signal components from the measurement noise, as described in section 1.3. Then the fit is performed with this reduced set of data.

But, in the context of time-resolved spectroscopy, *global analysis* is additionally associated with another notion: it relates to an analysis wherein only few assumptions about the processes occurring in the excited material are made. Therefore the fit function in a global fit is kept very general. This is in contrast to a *target analysis.* Therein the analyst presumes to know to some degree how the different species in the excited material interact with each other. Thus a more particular fit function is defined, to reflect these presumptions.

The assumptions at the base of the SVD assisted *global* fit implemented in the software are very simple:

- there are n_{comp} time-constants to be identified.
- the components do not interact with each other. I.e. the components do not interconvert.
- the components decay or accumulate exponentially.

Thus, given a set of n_{comp} selected right singular vectors V_i and singular values σ_i, a set of n_{comp} fit functions is implemented as

$$f(a_{ij}, \tau_j)_i = \sum_{j=1}^{n_{comp}} a_{ij} \cdot \exp\left(-\frac{t}{\tau_j}\right) \tag{6}$$

so that each right singular vector V_i can be fitted as

$$\sigma_i \cdot V_i = f(a_{ij}, \tau_j)_i, \tag{7}$$

where the V_i are weighted according to their corresponding singular values.
The fit function f_i is handed the fit parameters a_{ij}, τ_j, which respectively act as amplitudes and decay constants for the n_{comp} exponentials in f_i. As one can see, the τ_j are shared parameters among the fit functions, i.e. they are the same for all f_i, while the a_{ij} are individual parameters. (Shared parameters are sometimes also called *global* fit parameters. But for the sake of clarity, this renewed use of "global" is avoided herein.)

The *minimize* function implemented in the *lmfit*[2] package by (Matt Newville et al. 2021) then assures that the residuals

$$r_i = \sigma_i \cdot V_i - f(a_{ij}, \tau_j)_i \tag{8}$$

are simultaneously minimised.

With the resulting amplitudes a_{ij} and the selected left singular vectors U_i one can then construct a set of so-called *decay associated spectra* (DAS) as

$$DAS_j = \sum_{i=1}^{n_{comp}} a_{ij} \cdot U_i. \tag{9}$$

The DAS_j then represents the spectral dependence of time constant τ_j throughout the spectral window of the measurement (Bouduban 2019).

Finally, using these DAS and the resulting decay constants τ_j, a reconstructed data matrix can be computed as

$$\Psi_{fit} = \sum_{j=1}^{n_{comp}} DAS_j \cdot \exp\left(-\frac{t}{\tau_j}\right). \tag{10}$$

The goodness of the fit results can be judged by comparing Ψ_{fit} with the original data matrix Ψ. Again, it can be helpful to inspect the difference matrix $\Delta\Psi_{fit} = \Psi - \Psi_{fit}$. Given that the correct set of components has been selected, this difference matrix should display only those structures that the analyst deems to come from systematic noise, as in section 1.3. Though one may be biased, the analyst should also judge the fit results by their experience. As is often the case, the fit procedure is dependent on the initial parameter values. So it can be useful to conduct the fit repeatedly with different initial conditions.

With the gained information, one can progress to step 2. This is very similar to step 1, and the two steps can also be conducted simultaneously. The one difference between the steps is the form of the fit functions. While in the global analysis the components, aka the exponentials in f_i, do not interact with each other, they are allowed to do that in a target analysis. Also, the analyst may introduce new exponentials, e.g. a phosphorescence contribution with a (fixed) decay constant τ_{ph}. To illustrate this, here is a possible target model fit function given three selected components:

$$f_i(a_{ij}, \tau_j) = a_{i1} \exp\left(-\frac{t}{\tau_1}\right) + a_{i2}\left(\exp\left(-\frac{t}{\tau_2}\right) - \exp\left(-\frac{t}{\tau_1}\right)\right) + a_{i3} \exp\left(-\frac{t}{\tau_{ph}}\right) \tag{11}$$

[2]The *lmfit* package allows for different fit methods applied in the *minimize* function. In this program the fit method denoted with 'leastsq' is used by default. This is an implementation of the Levenberg-Marquardt algorithm. If desired, the user can select another fit method from a set. This may be quite useful, as sometimes the default fit method does not converge, while another one (the main one that was tested is denoted with 'least_squares') may converge given the same data and initial fit parameter values.

where it is assumed that the second component is additionally populated via the first component, and that the third component decays comparatively slow with e.g. $\tau_3 = \tau_{ph} = \infty$.
The analyst can then compare the results of the fit using different models with the original data as in the above described global fit.

While the main focus of the software is the global analysis, a small feature has been introduced, which allows the user to conduct a fit with a self-defined fit function such as the one in equation (11). However, the practical use of this feature remains to be tested, as the feature relies on parsing of user input, which can be a tricky thing.

2. Program tutorial: a model analysis of some simple simulation data

This section is intended as a guide to the software. By going through the analysis steps for some simple simulation data, the reader can get to know the layout and major functionalities of the program.

The reader should note that the functioning of some widgets may not be explicitly declared as they are sometimes deemed self-explanatory. Also, though far from claiming to have covered everything, the program is equipped with many verbose tooltips and error messages, which should, hopefully, enable the user to navigate the program if something is unclear or goes wrong.

2.1. The simulated data

The simulated data is intended as a rough approximation of the resulting data as recorded in a pump-probe-spectroscopy measurement of a non-descript material as introduced in section 1.2. Thus the simulation computes an initial "excitation" whose magnitude at increasing times after the excitation is "measured". A positive (negative) magnitude could be interpreted as a higher (lower) transmission of the excited material at a certain wavelength when compared to the unexcited material.

The initial "excitation" consists of a number of Gaussians, i.e. a number of spectral "components". Added to the pure Gaussians is some pseudo-random noise, which is intended to imitate (to some degree) the fluctuations in the excitation process:

$$\text{excitation}(\lambda) = \sum_i^2 \text{comp}_i(\lambda) = \sum_i^2 \left(a_i \cdot \exp\left(-\frac{(\lambda - \mu_i)^2}{2 \cdot \sigma_i^2} \right) + \mathrm{p_{overall}} \cdot \mathrm{p_{pump}} \cdot X \sim \mathcal{N}(0,\, 2) \right) \tag{12}$$

Here the a_i, μ_i and σ_i represent the amplitudes, peak position and standard deviations of the Gaussians. $p_{overall}$ and p_{pump} are used as parameters to configure the scale of the noise contribution in the component Gaussian. (There are two noise scale parameters included, as this facilitates the configuration of multiple simulations, which is useful for the following sections.) The pseudo-random noise is drawn from a Normal distribution.

The component $comp_i$ then represents an array of intensities at each wavelength λ for which it is computed. It decays exponentially with decay constant τ_i. Thus the "measured" magnitude m of the "excitation" at each wavelength λ and at time t is computed as:

$$m(\lambda, t) = \sum_i^2 \left(\text{comp}_i(\lambda) \cdot \exp\left(-\frac{t}{\tau_i} \right) + \mathrm{p_{overall}} \cdot \mathrm{p_{probe}} \cdot X \sim \mathcal{N}(0,\, 2) \right) \tag{13}$$

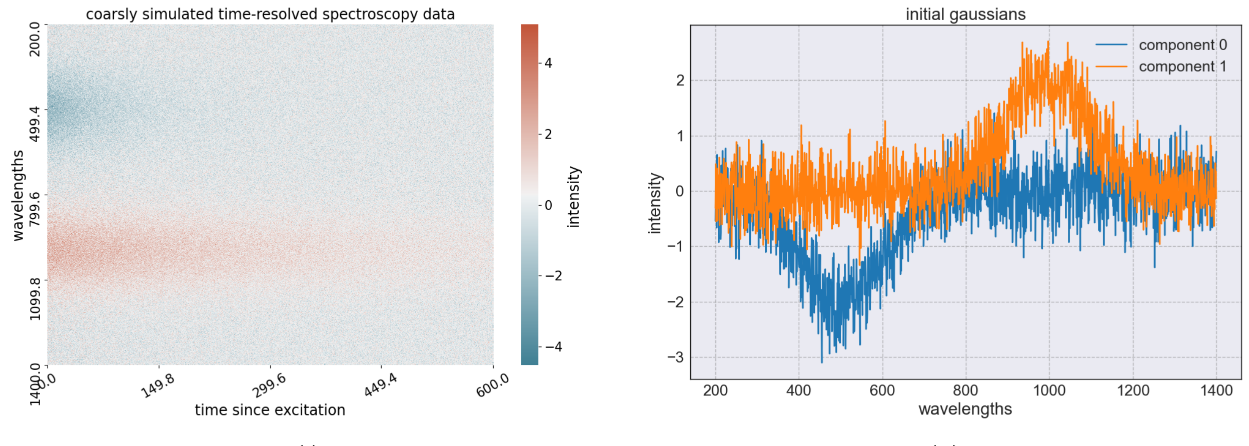
where again some pseudo-random noise is added (and scaled via $p_{overall}$ and p_{probe}) to represent the fluctuations in the measurement process.

In order to somewhat more accurately replicate a pump-probe-spectroscopy measurement, the initial "excitation" is generated anew for each "measurement". I.e. the noise on the initial Gaussians is different for each "measurement".

For this introductory chapter, the simulation contains two components with the following parameters:

- amplitudes a_0, a_1: -2, 2
- peak positions μ_0, μ_1: 500, 1000
- standard deviations σ_i: 100
- decay constants τ_0, τ_1: 100, 400
- p_{pump}, p_{probe}: 2.0
- $p_{overall}$: 0.1

A graphical representation of the simulated data can be seen in fig. 2.1



(i) (ii)

Figure 2.1 – Simulated idealised time-resolved spectrum and an example of some distorted initial Gaussians as used in the simulation.

The simulation is implemented in the `SimulateData\sim_MultiExcitation.py` file, the configuration is done in `..\configFiles\wavelength_overlap\overlap_0.ini`.

2.2. Data file format

The program expects a data file to be in the format depicted in table 1. The cell marked with x is the first value in the file and it will not be used in any computation. It should be filled with a placeholder value like e.g. 0.

At the moment only `.txt` or `.csv` files are supported. In the `.txt` file the values should be delimited with some white-space characters, as the program splits the values on white-space, if the `.txt` file type is detected. (In `.csv` files the values should of course be delimited by commas.)

For convenience sake, data files should be stored in the `DataFiles` folder, which already exists in the GitHub repository (i.e. which will be copied when downloading the code). Other folders can also be accessed from within the program, this just requires more clicks.

x	*wavelengths*				
time	data	data	data	data	data
	data	data	data	data	data
	data	data	data	data	data
	data	data	data	data	data
	data	data	data	data	data

Table 1 – The value in the cell marked with x will not be used in any computation. It should be filled with a placeholder value. One row of data represents the intensities measured at each wavelength at the row's corresponding time (delay).

2.3. Inspection of the data and its SVD in the GUI

As already mentioned, the package can be downloaded from the *GitHub* repository: `https://github.com/claudioHerger/TA_analyis_GUI`. Once in it's destination folder, the main program (`TA_analysis_GUI.py`) can be started from a terminal via the OS specific *python* command. (*python .\TA_analyis_GUI.py* in Windows when the current working directory is the mentioned destination folder.)

Once the program has loaded the window seen in fig. 2.2 is displayed. Use button set ⓪ in fig. 2.2 to choose and inspect a data file. The data can be displayed by clicking the *show original data*-button. (If no file has been selected via the *change data file*-button, a corresponding file dialog window is opened.)

The simulation data used in this tutorial is in the file `wavelength_overlap\overlap0.txt`. After selecting the file in the file dialog, a heat map depicting the data gets displayed on the GUI. By comparing fig. 2.1i and fig. 2.3 one can see, that the complete data matrix is displayed. This could be changed using the *change matrix bounds*-button, which allows the user to set specific bounds for the wavelengths and time delays. This way the data matrix can be downsized in the wavelength and or time step dimension.[3] By again employing the *show original data*-button, a heat map of the downsized data matrix would be produced. Until new bounds are set (or until the data file is changed, in which case the bounds are reset, so that by default the complete matrix is used again), this downsized data matrix is also used for any subsequent analysis.

By clicking the *inspect SVD*-button (fig. 2.3), a window is opened to inspect the singular values and vectors of the data matrix (corresponding to the set start time value). In this window the first ten (weighted) singular vectors, and the first 50 singular values can be plotted. These numbers are chosen as they should suffice to differentiate between signal and noise components in most cases.

For the simulated data, the first two singular values are the largest (see fig. 2.4). Also, the first two singular vectors (both left and right) are the ones displaying most structure. Thus these two components are selected for the further analysis steps.

[3]This might be useful if one wants to cut some wavelengths or time steps from the data matrix.

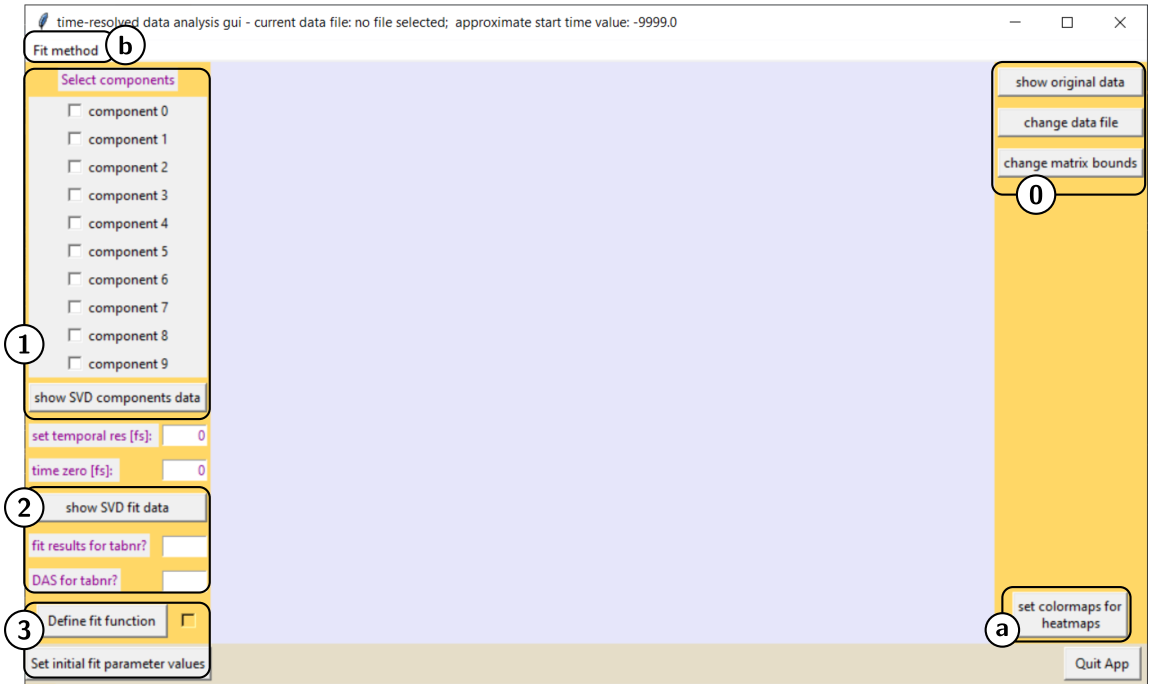


Figure 2.2 – ⓪ Display data via the *show original data*-button. Change the data file via the *change data file*-button. Change the data matrix dimensions via the *change matrix bounds*-button.
① Select SVD components for construction of the reduced matrix Ψ_{SVD} (equation (5)) via *show SVD components data*-button. Ψ_{SVD} and its difference from the original data matrix will be displayed as heat maps. Selected components will also be used for the fit.
② Start a fit using the selected components via the *show SVD fit data*-button. Once a fit has been successfully conducted, the Ψ_{fit} matrix (equation (10)) and its difference from the original data matrix will be displayed. Use the *fit results for tabnr*-entry to display a window containing exhaustive information on the fit results. Use the *DAS for tabnr*-entry to inspect the resulting DAS.
③ Define a fit function via the corresponding button. Once it has been successfully parsed, check the checkbox next to the button to use the defined function in the fit. (Checkbox checked: user defined fit function will be used. Checkbox unchecked: default fit function will be used.) Use the *Set initial fit parameter values*-button to do just that. If a fit does not converge, it can be helpful to set different start values. (The parsed user defined fit function will be stored to a file, so too will be the initial fit parameter values.)
ⓐ The *set colormaps for heat maps*-button can be used to change the color schemes for the heat map of the data matrices.
ⓑ Select the fit method from a drop-down menu.

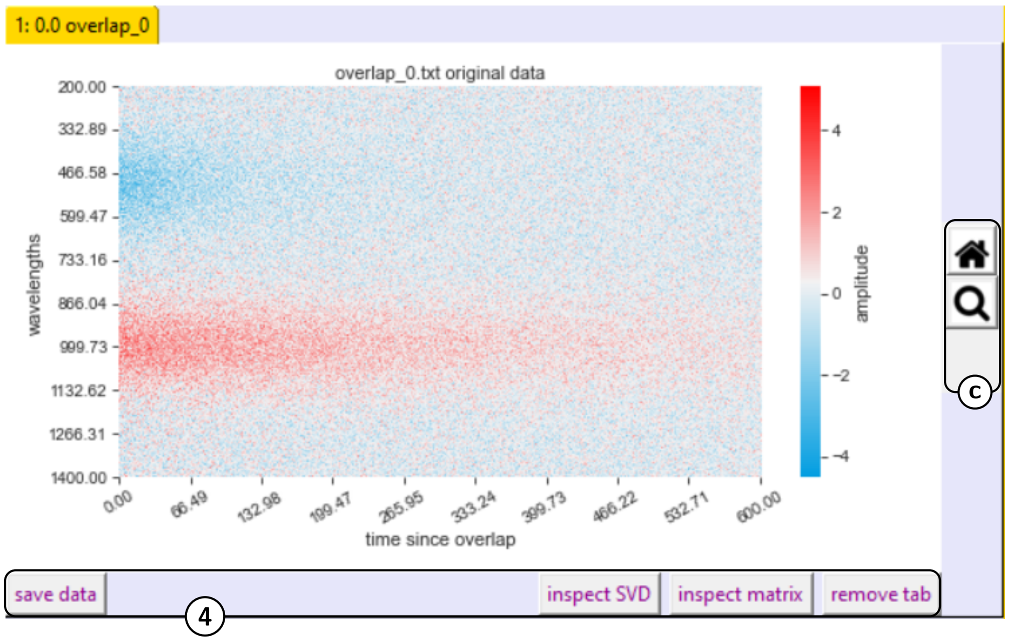


Figure 2.3 – ④ Save the figure and its corresponding matrix data to the `ResultData`-folder (within the `DataFiles`-folder) via the *save data*-button. Open a window to inspect the SVD of the data matrix via the *inspect SVD*-button. Inspect the data matrix row-by-row and column-by-column in a window via the *inspect matrix*-button. Remove the tab via the *remove tab*-button.
ⓒ These buttons can be used to zoom into the heatmap and then restore the initial heatmap again.

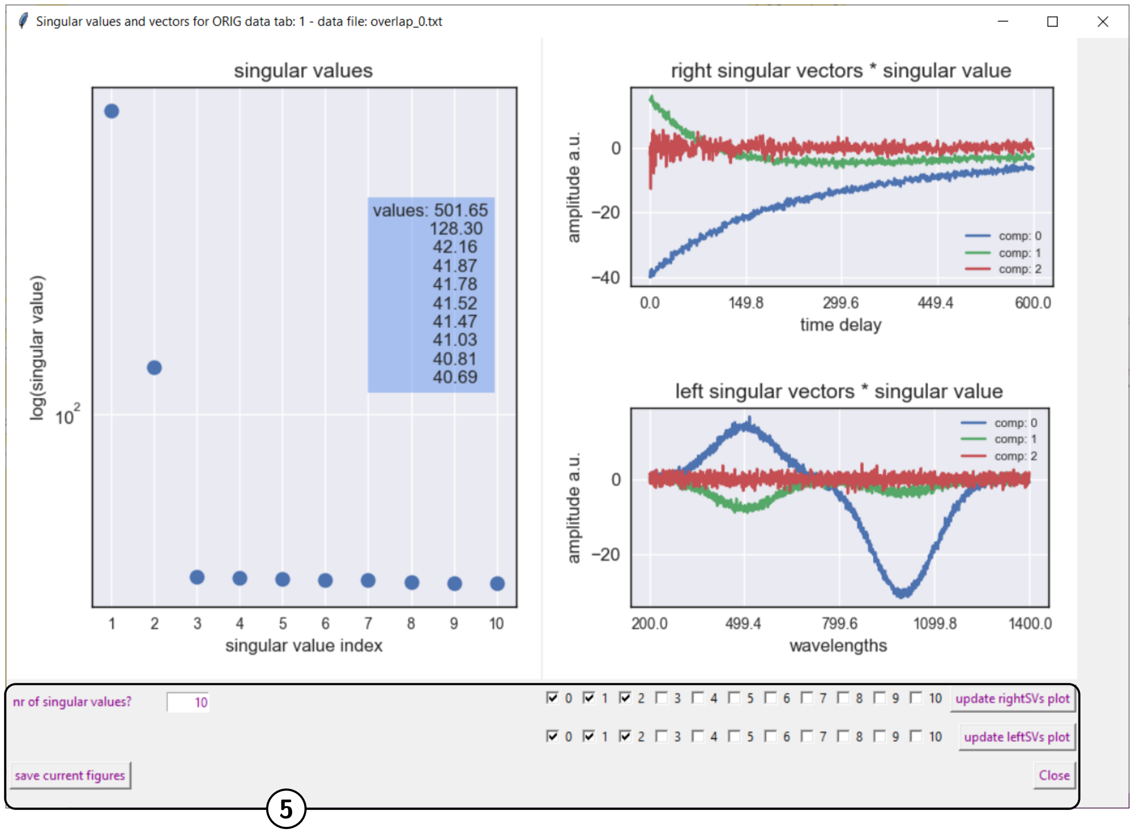
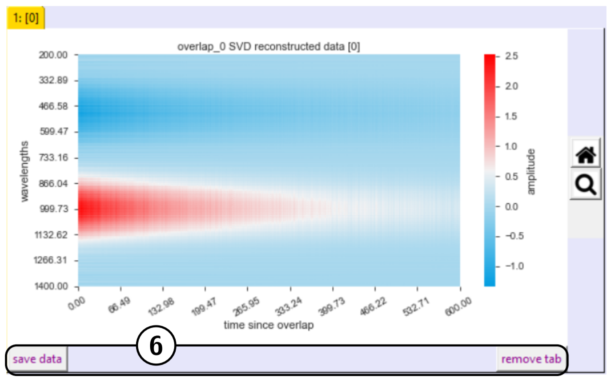


Figure 2.4 – ⑤ Save currently displayed figures via the *save current figures*-button (figures are saved in the same folder as the data saved via *save data* button in fig. 2.3). Set the number of displayed singular values via the *nr of singular values?*-entry (hit Return key to update singular values plot). Choose a set of singular vectors to inspect and display them via the *update rightSVs plot*-button and*update leftSVs plot*-button.

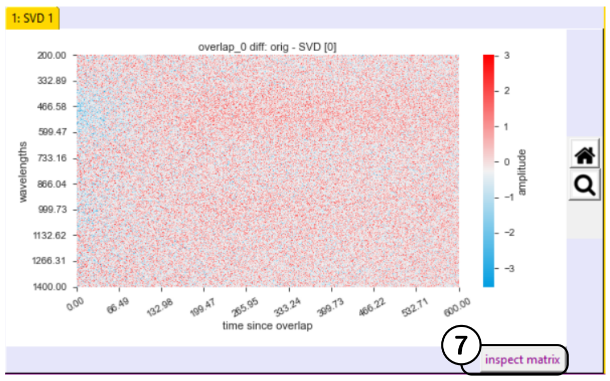
2.4. Comparing different SVD component sets

Using the checkboxes in ① from fig. 2.2, a set of SVD components can be selected. By clicking the *show SVD component data*-button the reduced matrix Ψ_{SVD} (equation (5)) can then be viewed. To facilitate comparison of Ψ_{SVD} with the original data, the difference matrix $\Delta\Psi_{SVD}$ is also displayed.

For illustrations sake, the Ψ_{SVD} and $\Delta\Psi_{SVD}$ of the simulated data using only the $0th$ component are displayed in fig. 2.5.



(i) Ψ_{SVD}

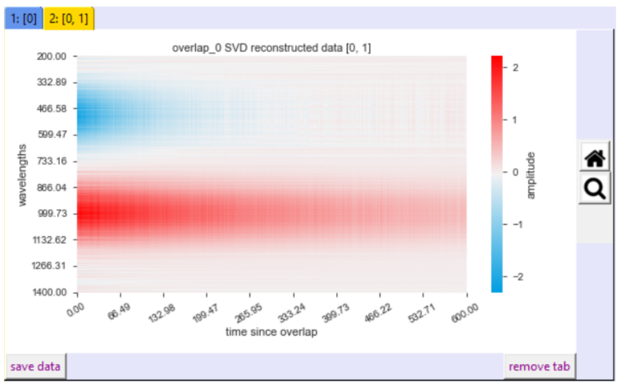


(ii) $\Delta\Psi_{SVD}$

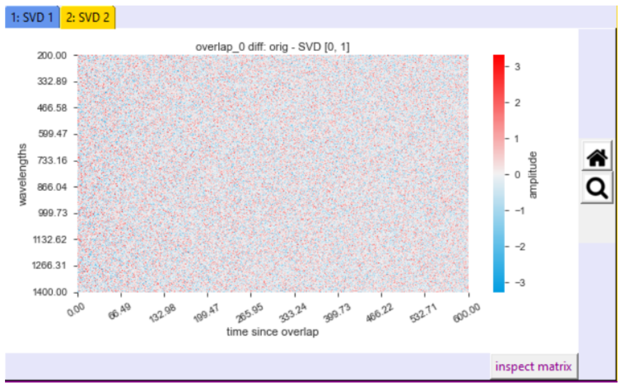
Figure 2.5 – The reduced matrix using only the $0th$ component of the simulated data in fig. 2.3.
⑥ Save this plot and corresponding data to file or remove the tab from the GUI, which will also remove the tab displaying $\Delta\Psi_{SVD}$.
⑦ Same as in fig. 2.3 but this time the $\Delta\Psi_{SVD}$ can be inspected instead of the actual data matrix Ψ.

Although difficult to see, the $\Delta\Psi_{SVD}$ matrix shown in fig. 2.5ii still displays some structure around the mean wavelengths of both simulated Gaussians. (Using the *inspect matrix*-button in ⑦, the $\Delta\Psi_{SVD}$ could further be inspected row-by-row (i.e. wavelength-by-wavelength) and column-by-column.)

In contrast, when selecting both the $0th$ and $1st$ SVD component for the construction of Ψ_{SVD} and $\Delta\Psi_{SVD}$, this small remaining structure disappears, as can be seen in fig. 2.6.
Thus, fitting the data with those two components selected seems appropriate.



(i) Ψ_{SVD}

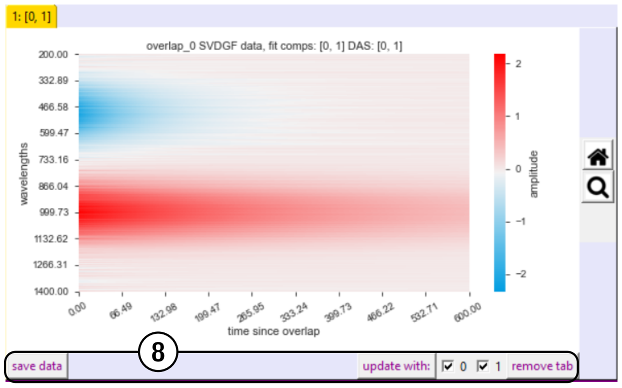


(ii) $\Delta\Psi_{SVD}$

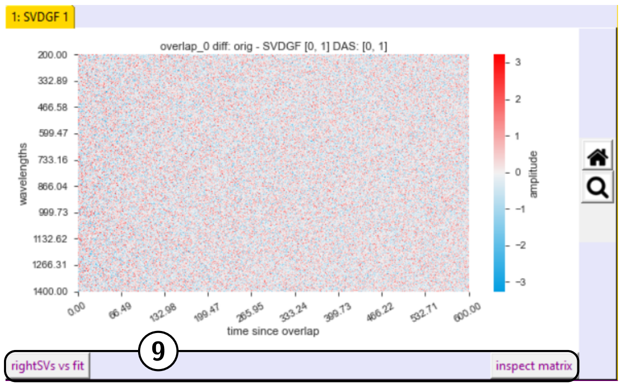
Figure 2.6 – The reduced matrix using the $0th$ and $1st$ component of the simulated data in fig. 2.3.

2.5. Performing a fit

Using the *show SVD fit data*-button in ② a fit can be conducted. (Note that in this section the default initial fit parameter values were used, i.e. all $\tau_i = 50$ and all $a_{ij} = 0.7$.) By default generic fit functions are applied to the weighted right singular vectors (equations (6) and (7)). Again, two heat maps are produced: one for Ψ_{fit} the other depicting $\Delta\Psi_{fit}$.
As one can see in fig. 2.7, the simulated data is well reproduced by the fit, as no clear structure remains in $\Delta\Psi_{fit}$.



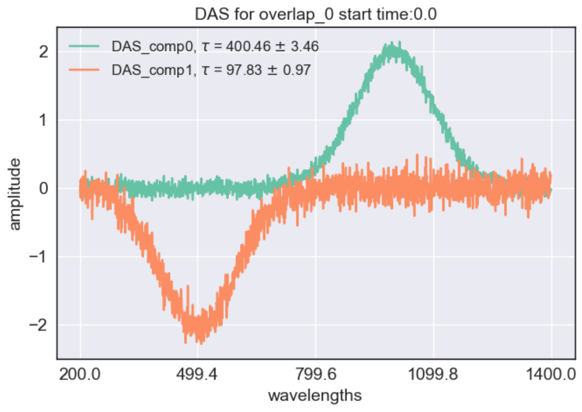
(i) Ψ_{fit}



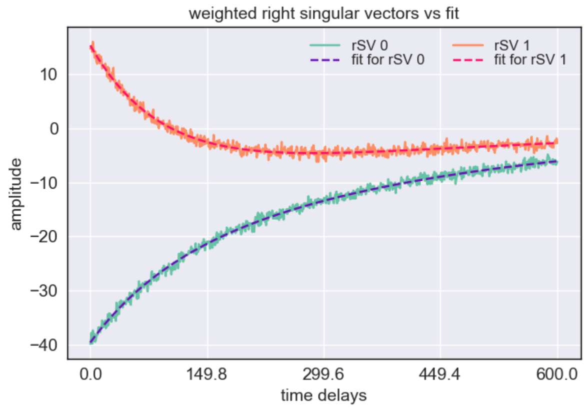
(ii) $\Delta\Psi_{fit}$

Figure 2.7 – The matrix resulting from the fit procedure using the 0*th* and 1*st* component of the simulated data in fig. 2.3.
⑧ Save this plot and corresponding data to file or remove the tab from the GUI, which will also remove the tab displaying $\Delta\Psi_{fit}$. Additionally, using the checkboxes and the *update with*-button, the user can update this plot and the one displaying $\Delta\Psi_{fit}$ by including only a selection of *DAS*.
⑨ use the *rightSVs vs fit*- button to open a window within which the weighted right singular vectors are plotted together with their reconstruction via the resulting fit parameters (see fig. 2.8ii for such a plot).

The resulting *DAS* can be inspected using the *DAS for tabnr?*-entry ②: simply enter the number of the tab that displays Ψ_{fit} and press the Return key. Also useful may be to check how the weighted right singular vectors are reconstructed by the resulting fit parameters. This can be done with the *rightSVs vs fit*-button in ⑨. In fig. 2.8 these fit results can be seen.
By comparing fig. 2.8i with fig. 2.1ii one can see that the *DAS* well reproduce the initial Gaussians used in the simulation. Additionally, fig. 2.8ii shows that the right singular vectors are also well reproduced by the fit.
All the fit results and further information can be inspected in the GUI by using the *fit results for tabnr?*-entry in the widget-set ②. The displayed text file would also be stored when saving the fit data using the *save data*-button in ⑧.



(i)



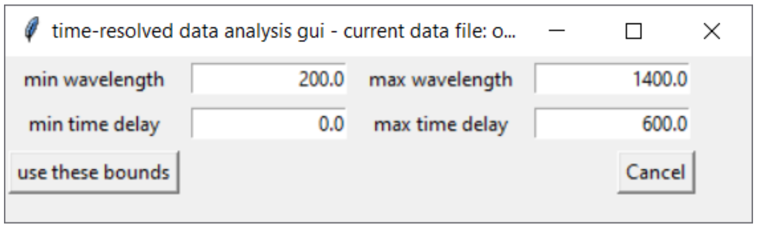
(ii)

Figure 2.8 – The resulting *DAS* and the weighted right singular vectors as well as their reconstructions using the resulting fit parameters in equation (6).

2.6. Further information on additional GUI widgets

2.6.1. Changing the data matrix dimensions

Using the *change matrix bounds*-button in ⓪ with the data file `wavelength_overlap\overlap0.txt` selected, the window depicted in fig. 2.9 is opened. There the minimum and maximum values for the time delays and or wavelengths can be set. By clicking the *use these bounds*-button, the entered values are evaluated and if valid they are applied from now on to downsize the data matrix. Whenever the data file is changed, the bounds are reset so that by default the complete data matrix is used until bounds for this data file are entered. When opening the window, the displayed values in the entries correspond to the maximum and minimum time delay/wavelength values of the corresponding data file.


time-resolved data analysis gui - current data file: o...

min wavelength	200.0	max wavelength	1400.0
min time delay	0.0	max time delay	600.0

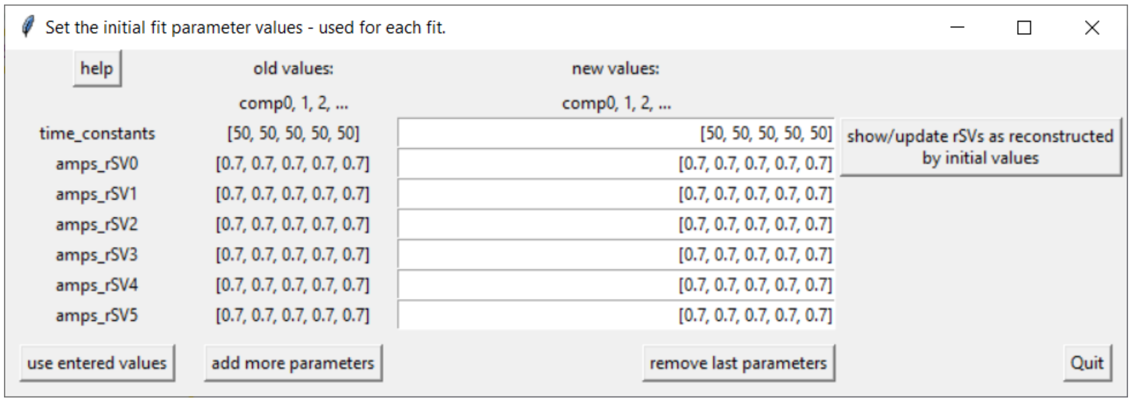
Figure 2.9 – Window to change the matrix dimensions.

2.6.2. Setting initial values for the fit parameters

Using the *Set initial fit parameter values*-button in ③, the window seen in fig. 2.10 can be opened. There the user may define a set of initial values for the fit parameters. The fit procedure (in `FunctionsUsedByPlotClasses/get_SVDGFit_parameters.py`) initialises n_{comps} decay constants τ_j and $n_{comps} \cdot n_{comps}$ amplitude parameters a_{ij} (n_{comps} = the number of components selected in ①). This is true if the generic fit function is used (6) as well as when any user defined fit function is applied[4]. Thus the initial parameter values set must at least contain that number of τ_j and a_{ij}. If it does not, the fit procedure will inform the user about that error and initialise the fit parameters to default values ($\tau_j = 50$, $a_{ij} = 0.7$).
The values entered in the text fields should be formatted as shown in fig. 2.10 and any value must be

[4]While all those parameters are used in the former, for the latter, any τ_j can be left out and any a_{ij} can be suppressed.

convertible to a float. There are some measures taken to ensure correct parsing of the user entered values, but no guarantee for completeness is given.
If correctly entered, the set of initial values is saved to a file formatted as a python dictionary (`configFiles/Initial_fit_parameter_values.txt`). Upon loading the program, this file is evaluated. If the evaluation is successful, the initial values used in the last session can be reused, else, the program will fallback to default values and the contents of the file will be overwritten once the user defines new values.


Set the initial fit parameter values - used for each fit.
help
show/update rSVs as reconstructed by initial values
use entered values
add more parameters
remove last parameters
Quit

	old values:	new values:
	comp0, 1, 2, ...	comp0, 1, 2, ...
time_constants	[50, 50, 50, 50, 50]	[50, 50, 50, 50, 50]
amps_rSV0	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]
amps_rSV1	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]
amps_rSV2	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]
amps_rSV3	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]
amps_rSV4	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]
amps_rSV5	[0.7, 0.7, 0.7, 0.7, 0.7]	[0.7, 0.7, 0.7, 0.7, 0.7]

Figure 2.10 – Window to set initial values for the parameters used in the fit procedure. Using the *show/update rSV as reconstructed by initial values*-button a window similar to fig. 2.8ii can be opened to check whether the initial values are very far off from a possible solution.

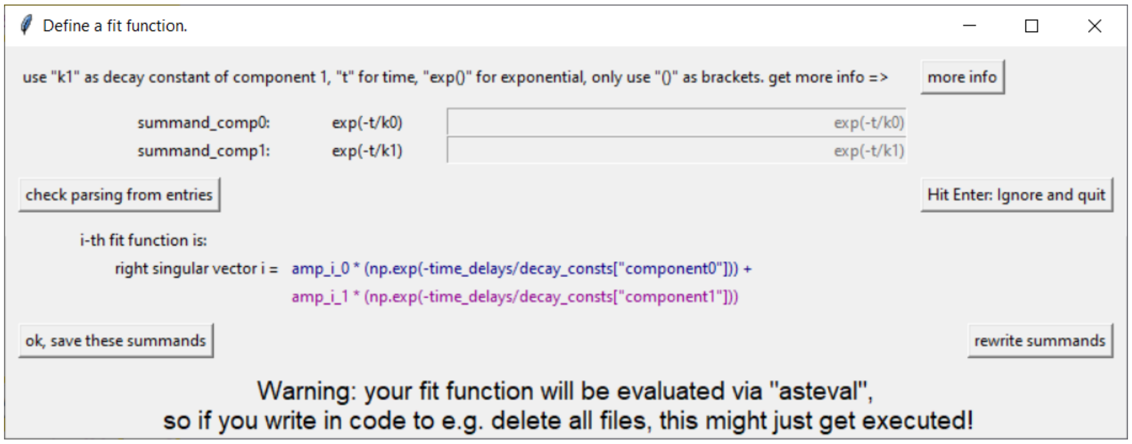
2.6.3. Defining and using a target model

The *Define fit function*-button in ③ opens a window similar to the one in fig. 2.11. Each fit function f_i (equation (6)) will still be a sum over the selected components (therefore the *summand_comp* labels in the window), but the form of these summands can be set by entering a (somewhat) mathematical statement into the entries. To evaluate the summands in the fit procedure, they need to be parsed to valid python code that refers only to variables that are known in the scope of a specific function within the fit procedure. Thus certain variables need to have a very particular name. Consequently some rules need to be followed when creating the summands:

- the decay constants τ_j are to be referred to as $k\#$, where # represents the number of the component. I.e. for τ_2 write $k2$ in the entry.
- the time (e.g. in an exponential decay) needs to be referred to as t. Thus any t in the entry string will be parsed to *time_steps*.
- an exponential function must be referred to as *exp*(). It will be parsed to *np.exp*(), i.e. *numpy.exp()*.
- Use of any other brackets than "()" will likely result in an error when evaluating the parsed code.

Depending on which components are checked in ①, the window to define the fit function will contain more summand labels and entries.
The parsing of the entry strings can be checked in the window via the *check parsing from entries*-button. The summands entered in fig. 2.11 (the entries are deactivated while checking the parsing) evaluate to the generic fit function using the 0-*th* and 1*st* component.
As written on the window, the parsed strings will be evaluated via the *asteval* (Matthew Newville 2021) python module. This is safer than using the python *eval* function, though care should still be taken to avoid any mishaps.
As with the initial fit parameter values, the parsed fit function is saved to a file (`configFiles/target_model_summands.txt`). The summand labels *summand_comp#* are the keys and the entry strings are the

values of a python dictionary.
In order to employ the user defined fit function, the checkbox next to the *Define fit function*-button must be checked. This can only be done once a fit function has been defined.



Define a fit function.

use "k1" as decay constant of component 1, "t" for time, "exp()" for exponential, only use "()" as brackets. get more info => more info

summand_comp0:	exp(-t/k0)	exp(-t/k0)
summand_comp1:	exp(-t/k1)	exp(-t/k1)

check parsing from entries

Hit Enter: Ignore and quit

i-th fit function is:

right singular vector i = amp_i_0 * (np.exp(-time_delays/decay_consts["component0"])) +
amp_i_1 * (np.exp(-time_delays/decay_consts["component1"]))

ok, save these summands

rewrite summands

Warning: your fit function will be evaluated via "asteval",
so if you write in code to e.g. delete all files, this might just get executed!

Figure 2.11 – Window to define a fit function.

2.6.4. Inspecting a matrix row-by-row and column-by-column

As already mentioned above, the GUI makes it possible to investigate the Ψ, $\Delta\Psi_{SVD}$ and $\Delta\Psi_{fit}$ matrices row-by-row and column-by-column in a separate window. This window can be opened via the *inspect matrix*-button in either ④, ⑦ and ⑨. The opened window to investigate Ψ depicted in 2.3 can be seen in fig. 2.12.
As established before, the data matrices resulting from time-resolved-spectroscopy are in the dimensions wavelength and time, i.e. $\Psi = \Psi(\lambda, t)$. Thus inspecting the matrix row-by-row actually means looking at the intensities measured at every timestep for a certain wavelength. In fig. 2.12 this can be done in the plot on the left. Using the slider below, the wavelength can be changed and the plot is redrawn. Correspondingly, inspection column-by-column means looking at the intensities measured at every wavelength for a certain timestep. Using the slider on the right, the timestep can be changed and the plot is redrawn.

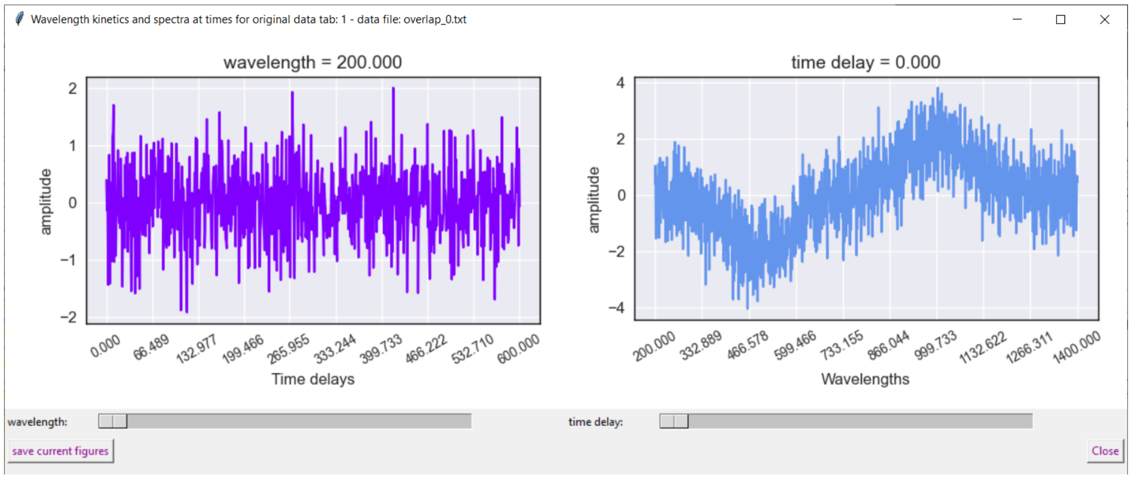
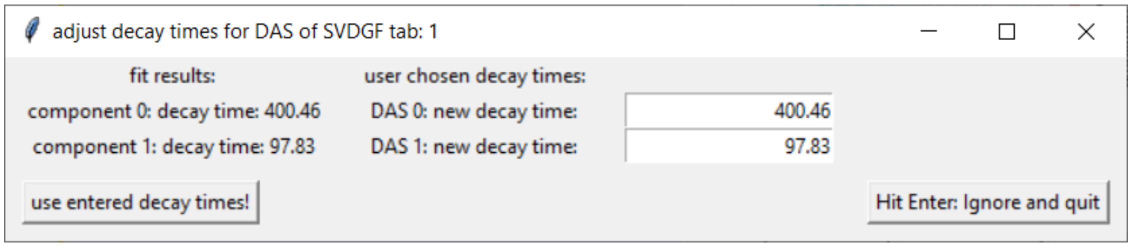


Figure 2.12 – Window to investigate a matrix row-by-row or column-by-column, i.e. wavelength-by-wavelength (on the left) and timestep-by-timestep (on the right).

2.6.5. Redrawing Ψ_{fit} with a selection of DAS only

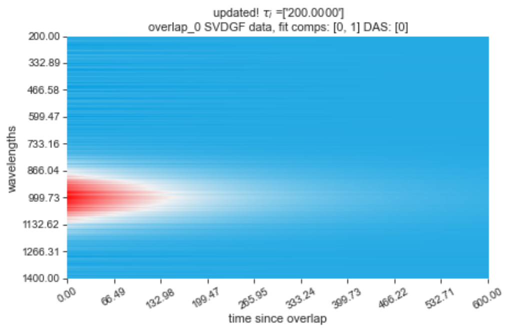
As mentioned before, for every component used in the fit, a DAS_j is summarised from the amplitudes a_{ij} and the left singular vectors U_i (equation (9)). To the DAS_j corresponds the fitted decay constant τ_j. Using these, the Ψ_{fit} is computed as a sum of exponential decays (equation (10)).
By default the matrix Ψ_{fit} is computed as a sum over all DAS. However, in some cases it might be useful to see how Ψ_{fit} looks with only a selection of the DAS_j used in equation (9). This is possible via the checkboxes and *update with:*-button in ⑧. There the user can make a selection of DAS to use in equation (9) via the checkboxes. Then the *update with:*-button opens a dialog window similar to the one in fig. 2.13.


adjust decay times for DAS of SVDGF tab: 1

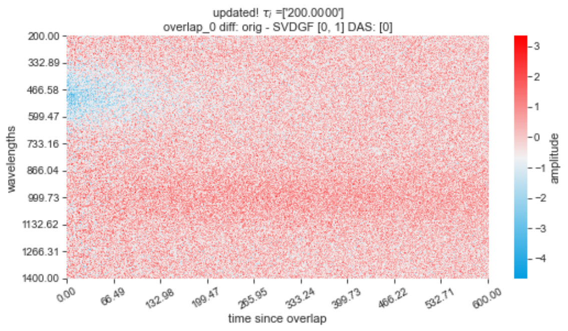
fit results:	user chosen decay times:	
component 0: decay time: 400.46	DAS 0: new decay time:	400.46
component 1: decay time: 97.83	DAS 1: new decay time:	97.83

Figure 2.13 – Window to set decay constants τ_j to use in computation of Ψ_{fit}.

The dialog window displays the fitted decay constants τ_j and allows the user to enter new values for them. If the user wants to use the fitted τ_j values for the construction of Ψ_{fit}, pressing the Return key or the *Hit Enter: Ignore and quit*-button will close the window and redraw the heat map in fig. 2.7 with the Ψ_{fit} using the defined DAS selection with their original decay constants. Clicking the *use entered decay times!*-button redraws the Ψ_{fit} heat map using the defined DAS selection and lets the DAS decay with the user entered decay constants.
For convenience sake, whenever the Ψ_{fit} heat map is redrawn, so too is the heat map depicting $\Delta\Psi_{fit}$. The title string of these heat maps is rewritten to express the fact that these heat maps/data matrices have been tampered with, i.e. to express that the depicted data matrices are not direct fit results. This can be seen in fig. 2.14.



(i) Ψ_{fit}



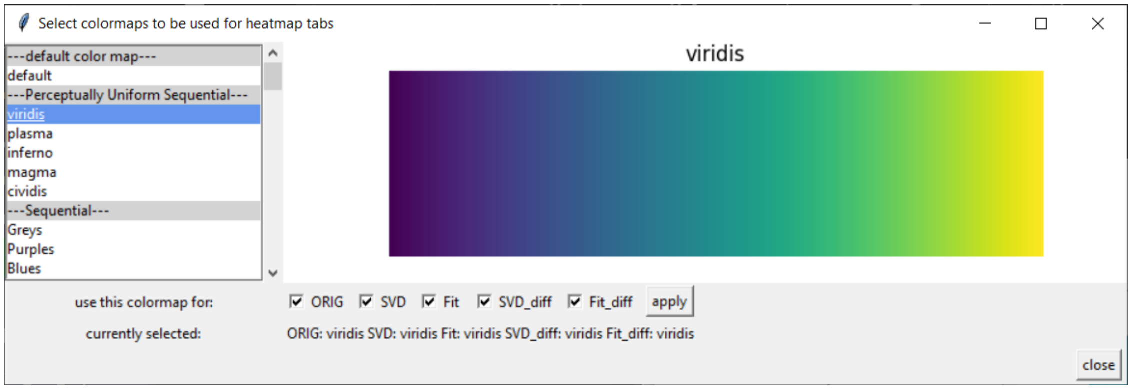
(ii) $\Delta\Psi_{fit}$

Figure 2.14 – The Ψ_{fit} and the $\Delta\Psi_{fit}$ with only the 0*th* DAS applied and with τ_0 changed from ≈ 400 to ≈ 200. The original figures can be seen in fig. 2.7

2.6.6. Changing the color-scheme for the heat maps

Via the *set colormaps for heatmaps*-button in ⓐ the window in fig. 2.15 is opened. In this window a colormap can be chosen by clicking on its name in the listbox. The selected colormap then gets displayed in the window. If the user likes a colormap, they can decide to which heat maps it should be applied to via

the checkboxes labelled *ORIG, SVD, Fit, SVD_diff and Fit_diff* and then clicking the *apply*-button. From then on, any newly created heatmap (whose checkbox was ticked) will be drawn with that colormap.
Below the checkboxes the current selection is displayed. Again, the current selection is saved to file (`configFiles/colormaps_for_heatmaps.txt`) and applied when restarting the program.



use this colormap for: ☑ ORIG ☑ SVD ☑ Fit ☑ SVD_diff ☑ Fit_diff apply

currently selected: ORIG: viridis SVD: viridis Fit: viridis SVD_diff: viridis Fit_diff: viridis

Figure 2.15 – Window to select the colormaps to be used for the heat maps of the data matrices.

2.7. Index: When will what data be saved to file

If some terminology is unclear, see other sections. Except for the configuration files, all saved data will be organised into sub-directories in `DataFiles/ResultData/...` by data file name, date, type (e.g. does it belong to original data heatmap tab or to SVD data heatmap tab etc.), selected components (where that applies) and then by the used start-time value (i.e. the minimum decay time of the complete or downsized data matrix), tab index, hour and minute of the creation of the data/plots etc.

2.7.1. Configuration files

The configuration files defining the colormaps to be used, the initial fit parameter values and the parsed target model summands will be stored to file whenever they are updated as described above. The folder is `configFiles/...` (See above.)

2.7.2. Original data heatmap tab

Clicking the *save data*-button in ④ will store

- a logfile (with date and time, data filename and the applied start-time value as well as the matrix bounds (if empty {} are indicated for the matrix bounds this means that the complete data matrix was used)),
- the data matrix and the corresponding heat map plot,
- the time delays (starting from applied start-time value) and wavelengths

to file.
When opening the SVD inspection window (*inspect SVD*-button in ④) and therein clicking the *save current figures*-button (⑤), the currently displayed singular values and vectors plots will be saved to file.
When opening the inspect matrix window (*inspect matrix*-button in ④) and therein clicking the *save current figures*-button, the currently displayed time trace/spectrum plots will be saved to file.

2.7.3. SVD components heatmap tabs

Clicking the *save data*-button in ⑥ will store

- a logfile (with date and time, data filename, the applied start-time value, the matrix bounds (if empty {} are indicated for the matrix bounds this means that the complete data matrix was used)) and the selected components),
- the Ψ_{SVD} and the $\Delta\Psi_{SVD}$ matrices with corresponding heat map plots,
- the data matrix,
- the time delays (starting from applied start-time value) and wavelengths,
- the full U and VT matrices,
- all the singular values,
- as well as three files with only the retained singular values and left and right singular vectors corresponding to the selected components

to file.
When opening the inspect matrix window (*inspect matrix*-button in ⑦) and therein clicking the *save current figures*-button, the currently displayed time trace/spectrum plots will be saved to file.

2.7.4. Fit heatmap tabs

Clicking the *save data*-button in ⑧ will store

- a logfile (with date and time, data filename the applied start-time value, the matrix bounds (if empty {} are indicated for the matrix bounds this means that the complete data matrix was used)), the selected components and a boolean to know whether or not a user defined target model was applied),
- the Ψ_{fit} and the $\Delta\Psi_{fit}$ matrices with corresponding heat map plots,
- a complete fit report as given by the *lmfit* package (also details the used initial fit parameter values),
- the data matrix,
- the time delays (starting from applied start-time value) and wavelengths,
- the retained singular values and left and right singular vectors,
- the computed DAS,
- if a user entered target model was used: the parsed summands,
- the $\Delta\Psi_{fit}$ as computed with the selected DAS (differs from $\Delta\Psi_{fit}$ if the plots were updated (via *update with*-button in ⑧)

to file.
When opening the inspect matrix window (*inspect matrix*-button in ⑨) and therein clicking the *save current figures*-button, the currently displayed time trace/spectrum plots will be saved to file.
When opening the window to inspect the fit vs right singular vectors (*rightSVs vs fit*-button in ⑨) and therein clicking the *save current figure*-button, the currently displayed figure will be saved to file.

2.7.5. Set initial fit parameters window

In this window one can configure the initial fit parameter values which will be stored separately, see above. But one can also inspect the fit functions as evaluated using the initial values as parameter values. (*show/update rSVs as reconstructed by initial values*-button) When saving the plot in the pop-up window,

- this plot,
- the entered initial fit parameter values,
- the right singular vectors of the selected components and their corresponding singular values

will be stored to file.
As it is possible to quickly update this plot, the final path will include the seconds of when the plot was saved.

References

Bouduban, Marine Eva Fedora (2019). "Charge carrier and exciton dynamics within hybrid lead halide perovskites of mixed composition and dimensionality". In: p. 154. DOI: `10.5075/epfl-thesis-9668`. URL: `http://infoscience.epfl.ch/record/269154`.

Cannizzo, Andrea et al. (2008). "Femtosecond Fluorescence and Intersystem Crossing in Rhenium(I) CarbonylBipyridine Complexes". In: *Journal of the American Chemical Society* 130.28. PMID: 18570416, pp. 8967–8974. DOI: `10.1021/ja710763w`. eprint: `https://doi.org/10.1021/ja710763w`. URL: `https://doi.org/10.1021/ja710763w`.

Newville, Matt et al. (Feb. 2021). *lmfit/lmfit-py 1.0.2*. Version 1.0.2. DOI: `10.5281/zenodo.4516651`. URL: `https://doi.org/10.5281/zenodo.4516651`.

Newville, Matthew (June 2021). *asteval*. Version 0.9.25. URL: `https://pypi.org/project/asteval/`.

Ruckebusch, C. et al. (2012). "Comprehensive data analysis of femtosecond transient absorption spectra: A review". In: *Journal of Photochemistry and Photobiology C: Photochemistry Reviews* 13.1, pp. 1–27. ISSN: 1389-5567. DOI: `https://doi.org/10.1016/j.jphotochemrev.2011.10.002`. URL: `https://www.sciencedirect.com/science/article/pii/S1389556711000748`.

The-SciPy-community (2021). *scipy.linalg.svd*. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.%20svd.html` (visited on 05/06/2021).

van Stokkum, Ivo H.M., Delmar S. Larsen, and Rienk van Grondelle (2004). "Global and target analysis of time-resolved spectra". In: *Biochimica et Biophysica Acta (BBA) - Bioenergetics* 1657.2, pp. 82–104. ISSN: 0005-2728. DOI: `https://doi.org/10.1016/j.bbabio.2004.04.011`. URL: `https://www.sciencedirect.com/science/article/pii/S0005272804001094`.