Photochemical & Photobiological Sciences
https://doi.org/10.1007/s43630-023-00460-y

ORIGINAL PAPERS

Pyglotaran: a lego-like Python framework for global and target analysis of time-resolved spectra

Ivo H. M. van Stokkum[1] · Jörn Weißenborn[1] · Sebastian Weigand[1,2] · Joris J. Snellenburg[1]

Received: 16 April 2023 / Accepted: 12 July 2023

Abstract

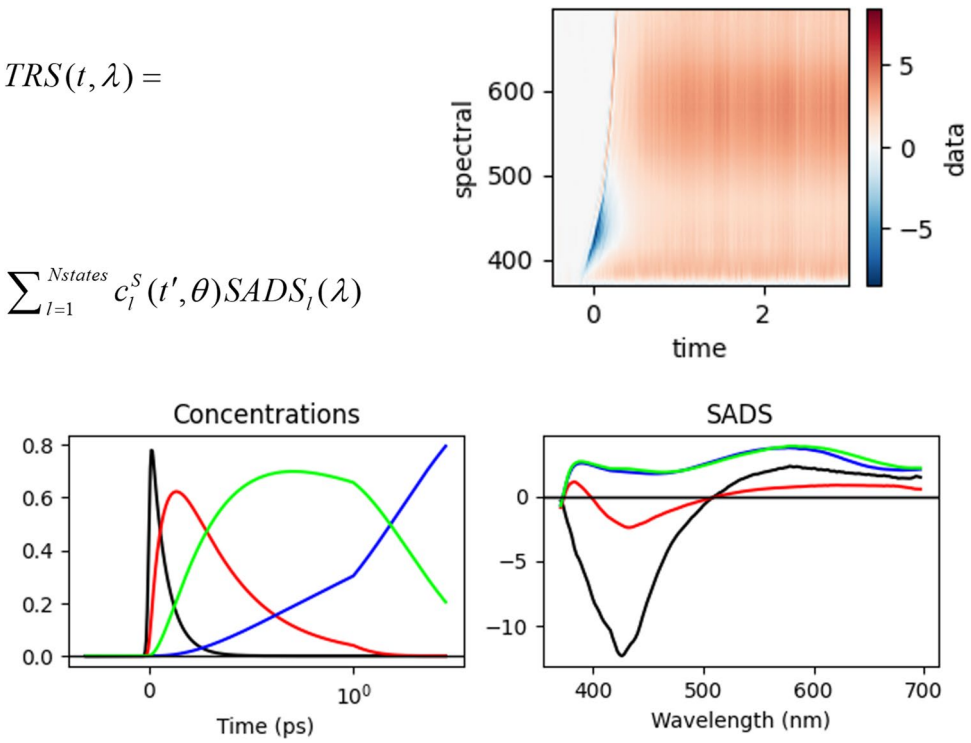
The dynamics of molecular systems can be studied with time-resolved spectroscopy combined with model-based analysis. A Python framework for global and target analysis of time-resolved spectra is introduced with the help of three case studies. The first study, concerning broadband absorption of intersystem crossing in 4-thiothymidine, demonstrates the framework's ability to resolve vibrational wavepackets with a time resolution of ≈10 fs using damped oscillations and their associated spectra and phases. Thereby, a parametric description of the "coherent artifact" is crucial. The second study addresses multi-chromophoric systems composed of two perylene bisimide chromophores. Here, pyglotaran's guidance spectra and lego-like model composition enable the integration of spectral and kinetic properties of the parent chromophores, revealing a loss process, the undesired production of a radical pair, that reduces the light harvesting efficiency. In the third, time-resolved emission case study of whole photosynthetic cells, a megacomplex containing ≈500 chromophores of five different types is described by a combination of the kinetic models for its elements. As direct fitting of the data by theoretical simulation is unfeasible, our global and target analysis methodology provides a useful 'middle ground' where the theoretical description and the fit of the experimental data can meet. The pyglotaran framework enables the lego-like creation of kinetic models through its modular design and seamless integration with the rich Python ecosystem, particularly Jupyter notebooks. With extensive documentation and a robust validation framework, pyglotaran ensures accessibility and reliability for researchers, serving as an invaluable tool for understanding complex molecular systems.

To be submitted to a special collection of scientific papers in the "Photochemical and Photobiological Sciences" in honor of Fred Brouwer https://www.springer.com/journal/43630.

Extended author information available on the last page of the article

Published online: 31 July 2023

Graphical abstract



Keywords Photosynthesis · Target analysis · Fluorescence · Transient absorption · Ultrafast spectroscopy

1 Introduction

Time-resolved spectroscopy is widely used in photochemistry and photobiology for investigating the dynamic properties of complex systems [1–4]. The global and target analysis methodology has been developed to model multidimensional datasets from these systems [5–9]. Here, *global* refers to a simultaneous analysis of all measurements; whereas, *target* refers to the applicability of a particular target model [10]. Fred Brouwer inspired the early developments of global and target analysis [11–13]. Several tools for global and target analysis exist in the public domain [14–21]. While some of these tools provide, as pyglotaran, a modular and Python-based approach, they have different foci for the analysis. The aim of this paper is to introduce pyglotaran [22], a lego-like Python problem-solving environment [23] for global and target analysis of time-resolved spectra. Several new features that are now becoming publicly available will be demonstrated with the help of an in depth presentation of two recent transient absorption case studies [24, 25], namely the usage of guidance spectra and the parametric description of the "coherent artifact" (CA) [2, 26]. Vibrational wavepackets [27, 28] will be described with the help of damped oscillations [26, 29]. Finally, in a third case study, the time-resolved emission of whole photosynthetic cells can be described by the contributions of all the megacomplexes present, thus resolving the energy transfer pathways [30] with the help of spectral area constraints [31]. These case studies explore systems containing 1, 2, or 3, and ≈500 chromophores, with compartmental models [32] describing 5 or 6 spectrally distinct species or states. The temporal resolution ranges from ≈10 fs to ≈10 ps.

The paper is organized as follows: We assume the reader is familiar with the basics of global and target analysis [7], with some foundational information included in the Supplementary Information (SI) Jupyter notebooks [33]. In the methods section, we summarize the methodological advancements since 2004, which are then illustrated through the case studies in the results and discussion section. Next, we present the design of the pyglotaran problem-solving environment and discuss software engineering aspects and future developments. In the results and discussion section, we provide a concise description of the salient analysis results, with extended descriptions

available in the SI Jupyter notebooks. Interested readers can download the Jupyter notebooks and pre-processed data to reproduce all results. Pyglotaran extends the FAIR data principles [34] by not only enabling users to share their data, pre-processing steps, and analysis results, but also by open-sourcing its software analysis tools and providing Jupyter notebooks for full reproducibility. To demonstrate this commitment to transparency and reproducibility, readers can download the Jupyter notebooks and pre-processed data associated with this paper to reproduce all the presented results, thus fostering a more collaborative and robust scientific community.

2 Methods

Crucial to virtually all global and target analysis is the superposition principle, expressing that the response of a complex system can be described by a linear superposition of the contributions from several components. We will first consider transient difference absorption spectroscopy, and then time-resolved emission spectroscopy. The superposition principle is schematically depicted in Fig. 1 for a single data set, that will be explained in detail in the first case study. The data matrix (first row) shows a complex spectral evolution (second row) with damped oscillations (third row) and a pronounced coherent artifact straddling time zero (fourth row, cf. the trace at 600.5 nm in the first row). The fifth row depicts the analysis of the remaining structure in the residual matrix of the fit.

In broadband absorption spectroscopy [35, 36], the evolution of the ground- and excited-state vibrational wave packets created by the short laser pulse is described with a superposition of damped oscillations. The amplitude of a damped oscillation $\cos(\omega_n t)\exp(-\gamma_n t)$ as a function of the detection wavelength constitutes a damped oscillation-associated spectrum ($DOAS_n(\lambda)$) with an accompanying wavelength-dependent phase $\varphi_n(\lambda)$ [29] (cf. row three of Fig. 1). When the vibrational evolution can be considered independently from the electronic evolution (Born–Oppenheimer approximation), we arrive at a superposition of the electronic and vibrational contributions to the time-resolved spectrum (TRS):

$$\begin{aligned} TRS(t,\lambda) = &\sum\nolimits_{l=1}^{Nstates} c_l^S(t',\theta) SADS_l(\lambda), \\ &+ \sum\nolimits_{n=1}^{Nosc} \cos(\omega_n t' - \varphi_n(\lambda))\exp(-\gamma_n t') DOAS_n(\lambda), \\ &+ \sum\nolimits_{m=0}^{2} i^{(m)}(t) IRFAS_m(\lambda) + residual(t,\lambda), \end{aligned}$$

where N_{states} electronically excited states are present in the system, with populations $c_l^S(t)$(superscript S stands for species), and species' spectral properties, the species associated difference spectra ($SADS_l(\lambda)$) (cf. row two of Fig. 1). The populations are determined by an unknown compartmental model [32], that depends upon the unknown kinetic parameters θ. In the target analysis constraints on the *SADS* are needed to estimate all parameters θ and $SADS_l(\lambda)$. t' indicates that the actual model function still must consider the instrument response function (IRF) by means of convolution. The next term describes the coherent artifact, with a weighted sum of the zeroth, first and second derivative of the IRF $i(t)$ [2] (cf. row four of Fig. 1). Finally, the residual represents the part of the data that is not described by the parameterized model (cf. row five of Fig. 1).

For every wavelength, the matrix formula for this superposition model is given by

$$\begin{aligned} TRS = &C^S(\theta,\mu,\Delta)\cdot SADS + \mathrm{Cos}(\omega,\gamma,\mu,\Delta)\cdot A \\ &+ \mathrm{Sin}(\omega,\gamma,\mu,\Delta)\cdot B + IRF(\mu,\Delta')\cdot IRFAS, \end{aligned}$$

where the matrix C^S consists of columns $c_l^S(t)$. A Gaussian shaped IRF is used, with parameters μ for the time of the IRF maximum and Δ for the full width at half maximum (FWHM) of the IRF. The matrices $\mathrm{Cos}(\omega,\gamma,\mu,\Delta)$ and $Sin(\omega,\gamma,\mu,\Delta)$ contain the damped oscillations, and the matrices A and B comprise their amplitudes. To limit the number of free parameters, we assume wavelength independence of the eigenfrequency ω_n and of the damping rate γ_n.

The final term, which describes the coherent artifact, contains a matrix $IRF(\mu,\Delta')$ with as columns the $i^{(m)}(t)$. The SADS and IRFAS and also the amplitudes A and B are unconstrained conditionally linear parameters, that can be implicitly solved for (per wavelength) using the variable projection algorithm [37, 38].

When the IRF width Δ is larger than ≈150 fs the damped oscillations will be virtually averaged out, and for every wavelength, the matrix formula for this superposition model reduces to [40]:

$$TRS = C^S(\theta,\mu,\Delta)\cdot SADS + IRF(\mu,\Delta)\cdot IRFAS.$$

In time-resolved emission spectroscopy, the IRF is generally much wider than 150 fs, and the presence of a possible scatter component can be described by the IRF shape (i.e., the zeroth derivative). The species' spectral properties are called the Species Associated Spectra ($SAS_l(\lambda)$) and we have for every wavelength (disregarding a possible scatter component)

$$TRS = C^S(\theta,\mu,\Delta)\cdot SAS.$$

Since emission cannot be negative, the SAS are *nonnegative* conditionally linear parameters, that can be implicitly solved for (per wavelength) [39, 40].

In most experiments, the location of the maximum of the IRF is wavelength dependent. This so-called dispersion can

$$TRS(t,\lambda) =$$

$$\sum_{l=1}^{Nstates} c_l^S(t',\theta) SADS_l(\lambda)$$

$$+\sum_{n=1}^{Nosc} \cos(\omega_n t' - \varphi_n(\lambda))$$
$$\exp(-\gamma_n t')$$
$$DOAS_n(\lambda)$$

$$+\sum_{m=0}^{2} i^{(m)}(t) IRFAS_m(\lambda)$$

$$+residual(t,\lambda)$$

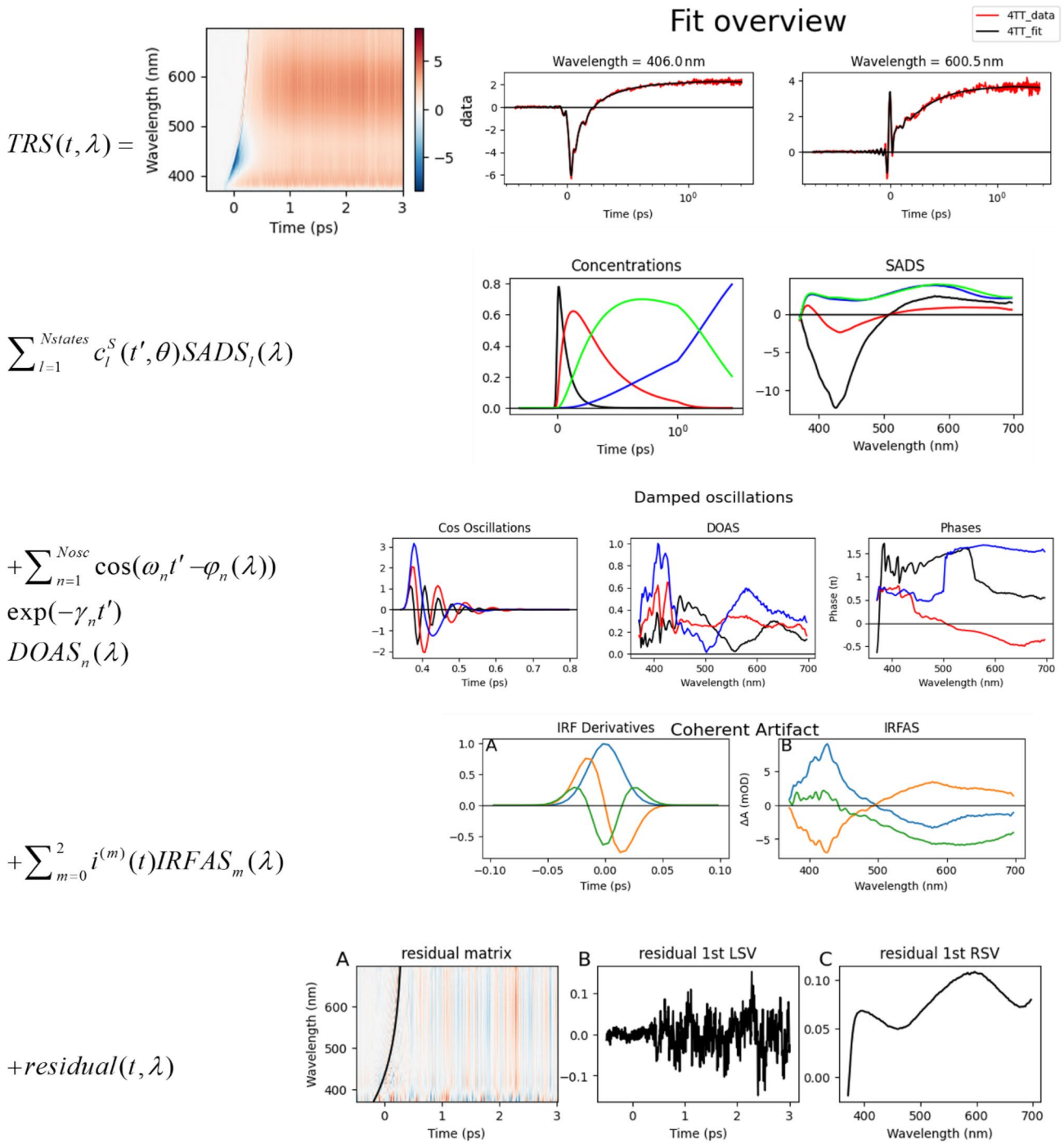


Fig. 1 Example of a time-resolved difference absorption Spectrum and fit thereof (row 1), with the help of a compartmental model (row 2), the damped oscillations (row 3), a "coherent artifact" (CA) (row 4) and the residual (row 5). Further explanation in text

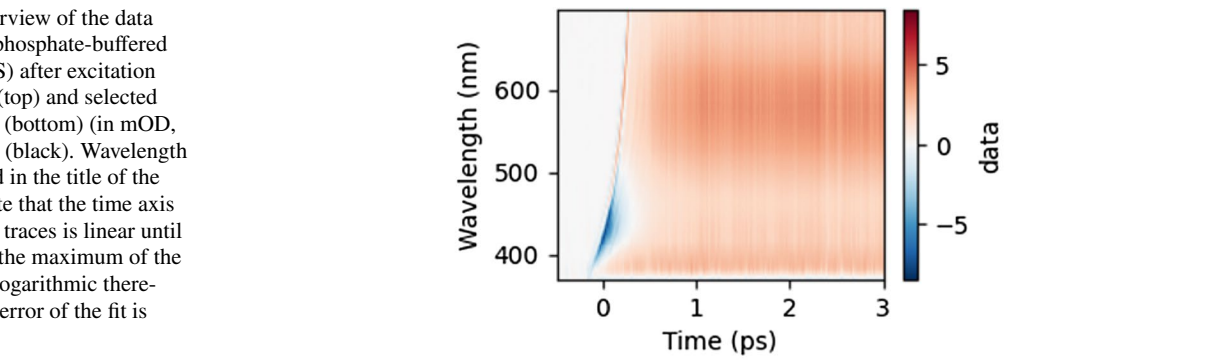
well be described by a polynomial function of the wavelength [7] or of its reciprocal, the wavenumber [2]. This introduces, typically, 1–3 "nuisance" parameters. Moreover, the whole kinetic model must be recomputed for every wavelength, which greatly increases the computation time. An independent experiment that involves a coherent artifact with dispersion must also be modeled with the above formulas [26].

3 Results and discussion

3.1 Broadband absorption case study of intersystem crossing in 4-thiothymidine

In a recent study, it was demonstrated that coherent vibrational modes promote the ultrafast internal conversion and intersystem crossing in thiobases [25]. Here, we will present the target analysis of 4-thiothymidine (4TT). The structure and steady-state spectra of 4TT are shown in Figure S1. The IRF width of ≈35 fs FWHM enables the resolution of damped oscillations that can be assigned to vibrational modes. The coherent artifact is clearly visible in the trace at 600.5 nm in Fig. 2, with large contributions of the IRF derivatives (Fig. 3).

We employ a sequential kinetic scheme with four states which we tentatively name S2, S1, hot T1', and T1, i.e., we assume the kinetic scheme S2 → S1 → hot T1' → T1. The populations and estimated SADS are shown in Fig. 4. The kink at 1 ps in Fig. 4A results from the time axis being linear until 1 ps, and logarithmic thereafter [41].



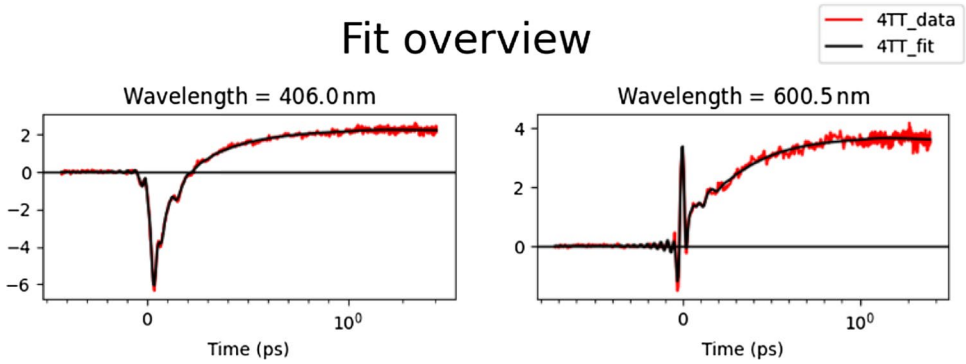


Fig. 2 Overview of the data of 4TT in phosphate-buffered saline (PBS) after excitation at 330 nm (top) and selected time traces (bottom) (in mOD, red) and fit (black). Wavelength is indicated in the title of the panels. Note that the time axis of the time traces is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter. Rms error of the fit is 0.12 mOD

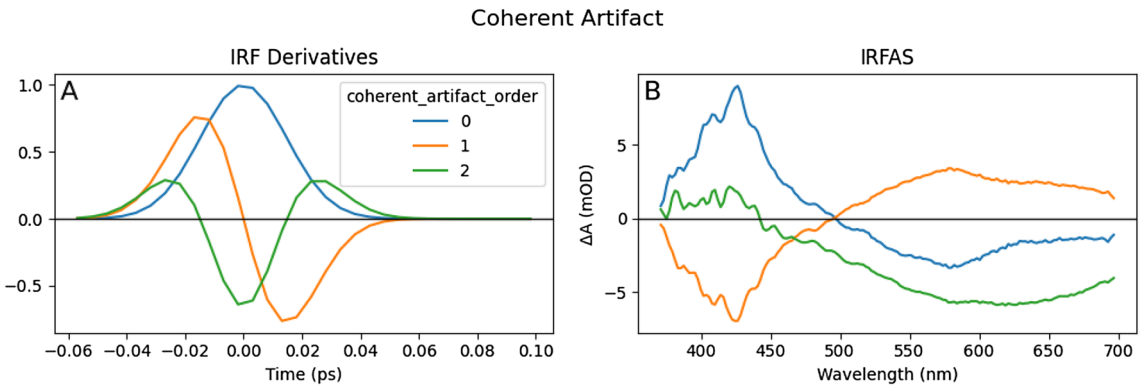


Fig. 3 Coherent artifact of 4TT in PBS after excitation at 330 nm. (A) 0th, 1st, and 2nd derivative of the IRF (blue, orange, green) which possessed a FWHM Δ' of 35 fs. (B) scaled IRFAS. Scaling of the IRFAS is such that the product of the IRFAS and the IRF derivative is the contribution to the fit. Thus, the blue IRFAS has the largest contribution to the fit

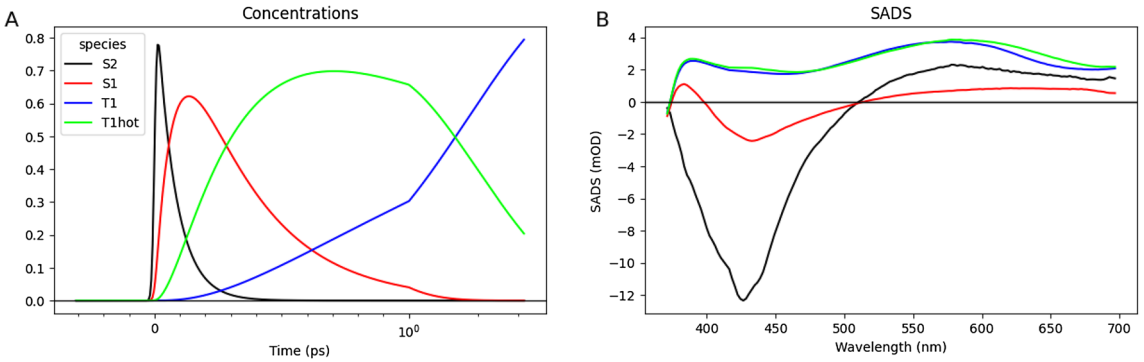


Fig. 4 Populations (A) of the kinetic scheme S2→S1→hot T1'→T1 and estimated SADS (B, in mOD) resulting from the sequential kinetic analysis of 4TT in PBS. Note that the time axis in A is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter. Estimated lifetimes: 74 fs (black), 284 fs (red), 1.76 ps (green), and long lived (blue)

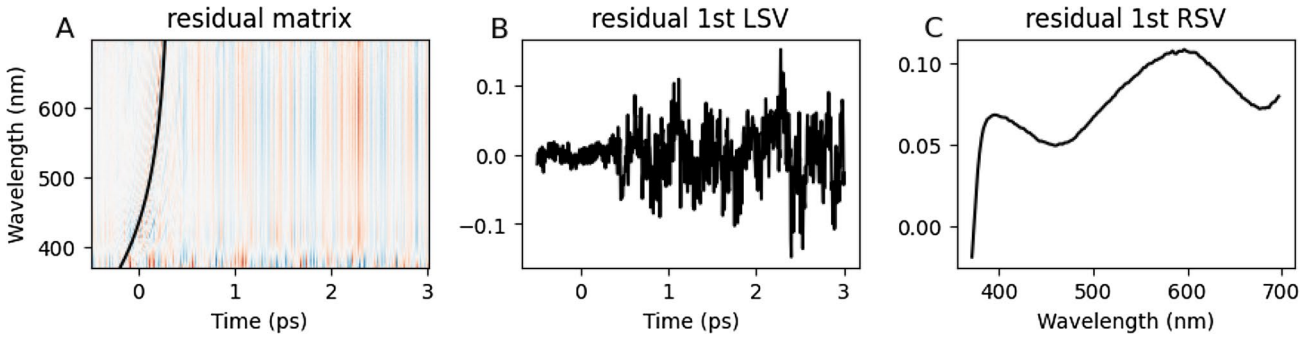


Fig. 5 First left (**B**) and right (**C**) singular vectors resulting from the singular value decomposition (SVD) of the residual matrix (**A**). The black line in (**A**) represents the location of the maximum of the IRF, which is described by a third-order polynomial function of the wavenumber

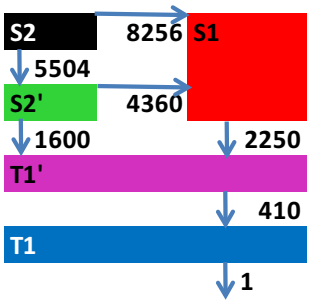


Fig. 6 Kinetic scheme used for the target analysis of 4TT in PBS. All rate constants in ns^{-1}. Key: lifetimes: 73 fs ($S_2(1\pi\pi^*)$, black), 168 fs ($S'_2(^1\pi\pi^*)$, green), 444 fs ($S_1(^1n\pi^*)$, red), 2.4 ps ($T_1'(^3\pi\pi^*)$, magenta) and long lived ($T_1(^3\pi\pi^*)$, blue)

Four damped oscillations have been used, one of which was time-reversed and is attributed to the coherent artifact. The DOAS and phases of the other three damped oscillations are shown in Figure S2.

Although at first glance the fit looks satisfactory (Fig. 2, bottom) and the rms error of 0.12 mOD is small, the residual matrix of the fit still shows some structure, cf. the vertical lines in Fig. 5A. These lines can tentatively be attributed to pump laser intensity fluctuations. This suggests refining the analysis and correct for these laser intensity fluctuations by estimating whether the residual spectrum at a certain time is proportional to the data. If so, the data can, thus, be corrected. This is demonstrated in Figure S3. The rms error decreases to 0.065 mOD, and the residuals show virtually no more structure (Figure S4).

It is difficult to interpret the S1 SADS, red in Fig. 4B, it still contains substantial stimulated emission around 430 nm, suggesting that it is a mixture of relaxed S2 and S1. Therefore, we adopted the kinetic scheme in Fig. 6 which contains five states, S2, relaxed S2', S1, hot T1' and T1. Associated with these states, there are five lifetimes. Several kinetic schemes have been tested, until we arrived at the scheme of Fig. 6 with well interpretable SADS that are in accordance with the theory [25]. The precision of the estimated parameters is reported in the result object (Figure S5). The relaxed S2' SADS (green in Fig. 7) is free from triplet (blue in Fig. 7) features. The S1 SADS (red) shows two ESA bands and a small SE around 430 nm.

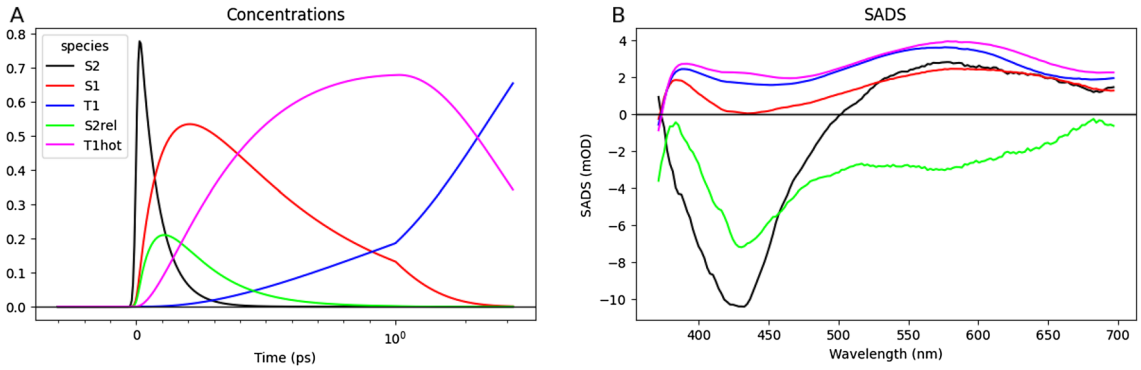


Fig. 7 Populations (**A**) of the target kinetic scheme from Fig. 6 and estimated SADS (**B**, in mOD) resulting from the target analysis of 4TT in PBS. Note that the time axis in A is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter

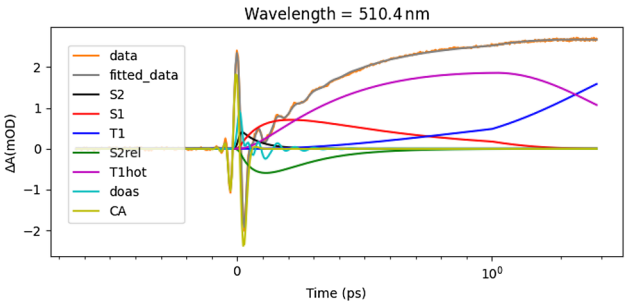


Fig. 8 Decomposition of the 4TT in PBS data at 510.4 nm (in mOD, orange, and the fit, gray). CA is the sum of the IRFAS and CA damped oscillations (decay rates > 50/ps), *doas* is the sum of the four damped oscillations with decay rates < 30/ps. Note that the time axis is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter

The contribution of the damped oscillations to the fit and the quality of the fit are demonstrated for the 510 nm data in Fig. 8. Note that the main decay of the damped oscillations (turquoise) corresponds to the main decay of S2 (black), suggesting that these oscillations live on (relaxed) S2. In the refined target analysis four DOAS have been resolved, the vertical lines in Fig. 9 indicate nodes concomitant with a phase jump in the 458 cm^{-1} DOAS (red, at 415 and 460 nm), in the 213 cm^{-1} DOAS (blue, at 393, 429, and 479 nm), and in the 847 cm^{-1} DOAS (green, at 526 nm). These results are further discussed in [25].

3.2 Transient absorption case study of the chromophoric systems rc and rcg

The primary event in molecule-based light energy conversion systems is light harvesting. We studied perylene bisimide-calix[4]arene multichromophoric systems composed of two different types of perylene bisimide (PBI) chromophores, red (**r**), and green (**g**) PBIs (named after their colors as solids) connected by calix[4]arene (**c**) [24, 42]. Figure 10A depicts the chemical structure of the supramolecular system **rcg**, and the absorption and emission properties of the parent chromophores **rc** and **gc** are shown in Fig. 10B. Due to the excellent overlap of the **rc** emission and the **gc**

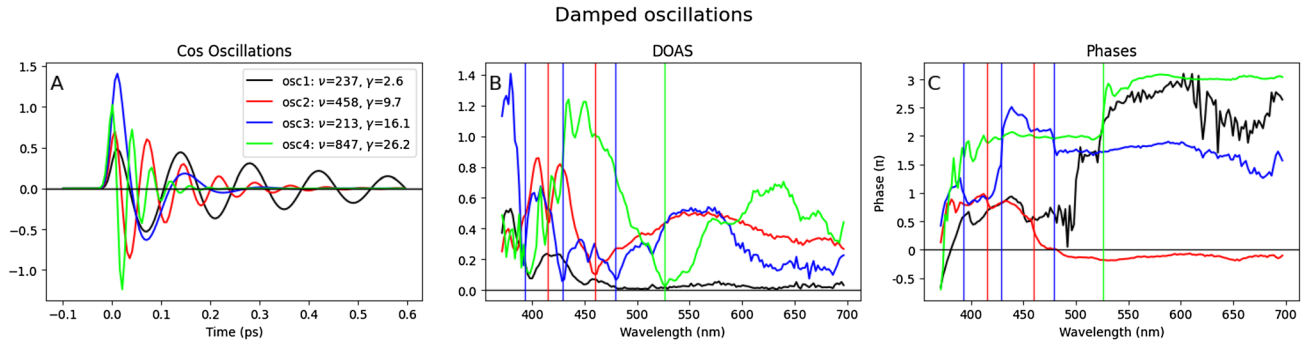


Fig. 9 Overview of the estimated DOAS and phases. **A** Cosine oscillations with frequencies $\overline{\nu}n$ (in cm^{-1}) (where n is the DOAS number) and damping rates γ (in ps^{-1}) written in the legend at the left, using the appropriate color. Scaling of the DOAS is such that the product of the DOAS and the damped oscillation is the contribution to the fit. **B** Estimated DOAS. (C) Estimated phase profiles of the DOAS. The vertical lines in (**B**, **C**) indicate nodes concomitant with a phase jump

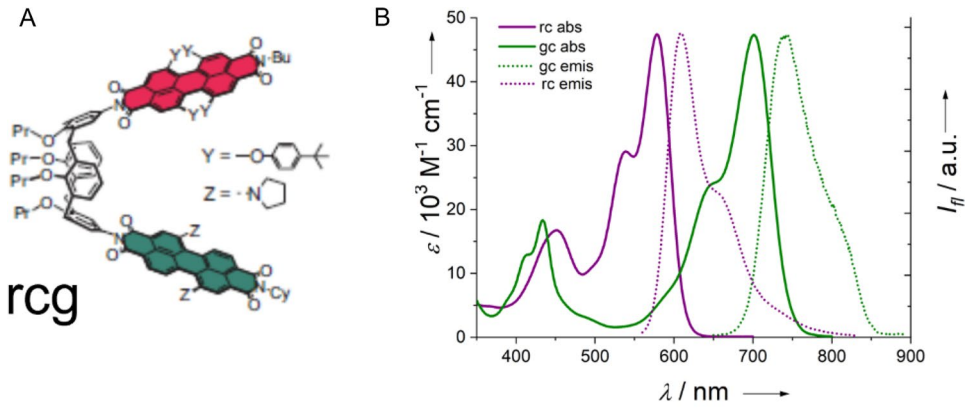


Fig. 10 **A** Chemical structure of the supramolecular system **rcg**, figure adopted from [43]. **B** UV/Vis absorption (red, solid) and fluorescence emission spectra (red, dotted) of tetraphenoxy-substituted (red) PBI compound **rc**; UV/Vis absorption (green, solid) and fluorescence emission spectra (green, dotted) of dipyrrolidino-substituted (green) PBI compound **gc**. All spectra are taken in CH_2Cl_2

absorption (dotted red and solid green lines in Fig. 10B) and the close proximity fast Förster excitation energy transfer (EET) is found after excitation of the **r** moiety [42]. However, the **rc** chromophore has inherent dynamic properties, which must be considered in a target analysis.

An important part of the data analysis is the pre-processing of the raw data. In the **rc** Jupyter notebook in the supplementary information, it is demonstrated how a global analysis is used to demonstrate the presence of a pre-zero baseline in the data (Figure S6, Figure S7), and how this baseline can then be estimated (Figure S8) and subtracted from the raw data. Throughout this manuscript, we refer to pre-processed data.

After 530 nm excitation of **rc** in CH_2Cl_2, the coherent artifact is well described by the wavelength-dependent $IRF(\mu, \Delta') \cdot IRFAS$ (Fig. 11). Representative traces demonstrating the excellent quality of the fit are shown in Fig. 12.

The extensive spectral evolution of **rc** can be described with four excited states $\mathbf{r_1} \rightarrow \mathbf{r_2} \rightarrow \mathbf{r_3} \rightarrow \mathbf{r_4} \rightarrow$ ground state, resulting in four rate constants $k_{r2,r1}, k_{r3,r2}, k_{r4,r3}, k_{r4}$ (where we use the $k_{to,from}$ convention, and k_{r4} denotes the decay rate to the ground state) and four **rc**-SADS. This sequential scheme (Fig. 13A) neglects the branching decay to the ground state of the first three states, since k_{r4} is much smaller than the other three rate constants. As a refinement one could add this decay channel, assuming a rate of decay to the ground state k_{r4} for all four states. The perylene red chromophore shows a strong spectral evolution in time, especially from 550 to 750 nm (Fig. 13B).

The first target analysis of **rcg** considers that the 530 nm also directly excites the **g** moiety (≈12% of the **r** absorption, Fig. 10B) and to the **rc**-kinetic scheme the four EET to **g** rate constants $k_{g,r1}, k_{g,r2}, k_{g,r3}, k_{g,r4}$ are added. By plotting the first left and right singular vectors resulting from the singular value decomposition (SVD) of the residual matrix, we can identify systematic patterns in the residuals, which may indicate potential issues with the model. Here, the fit using this

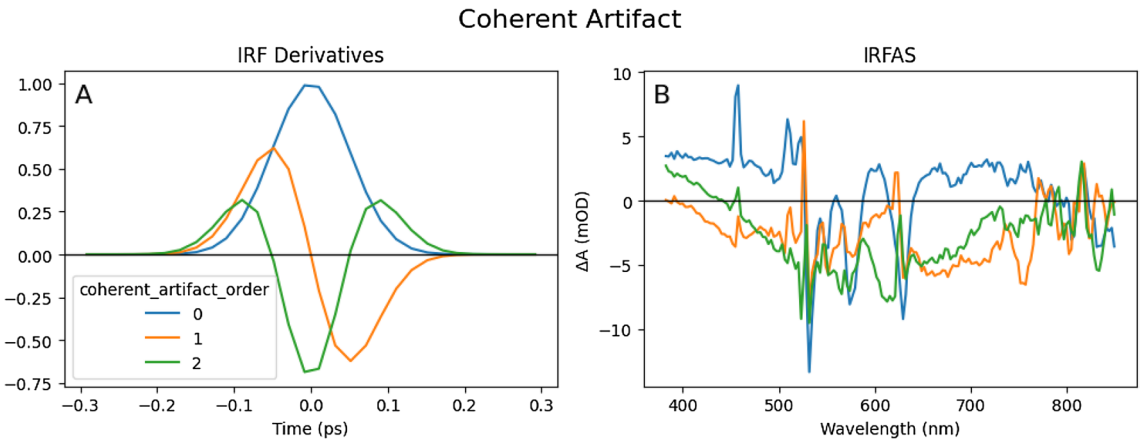


Fig. 11 Coherent artifact of **rc** in CH_2Cl_2. **A** 0th, 1st, and 2nd derivative of the IRF (blue, orange, green) which possessed a Full Width at Half Maximum (FWHM) of 119 fs. **B** scaled IRFAS. Scaling of the IRFAS is such that the product of the IRFAS and the IRF derivative is the contribution to the fit. Thus, the blue IRFAS has the largest contribution to the fit. It shows large amplitudes straddling 530 nm, the excitation wavelength. In addition, Raman scattering is visible as negative peaks at 580 and 636 nm, and positive peaks at 460 and 512 nm

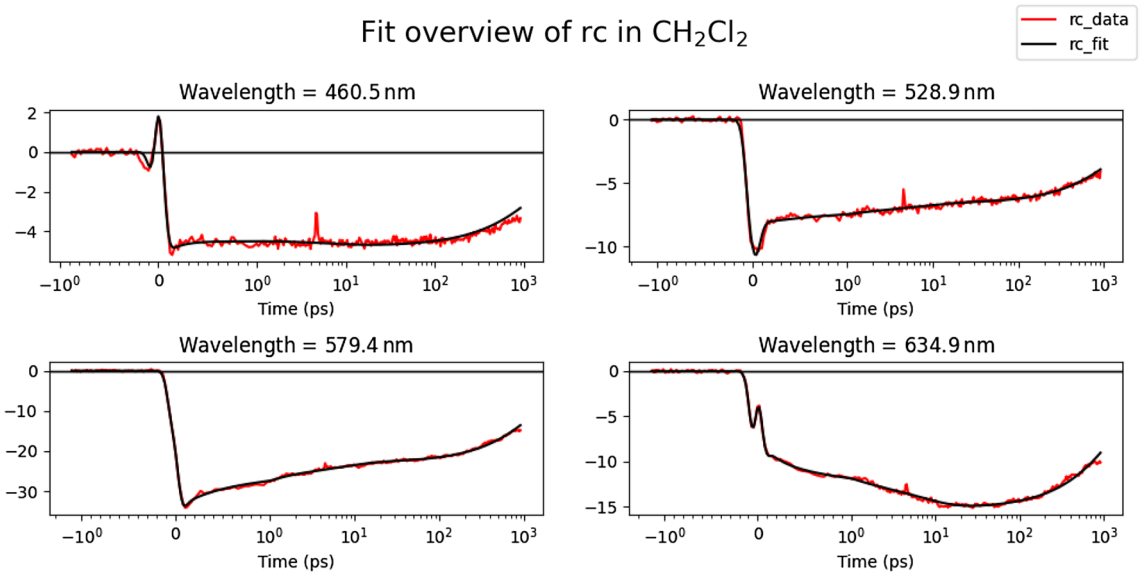


Fig. 12 Selected time traces of **rc** in CH_2Cl_2 after excitation at 530 nm data (in mOD, red) and fit (black). Wavelength is indicated in the title of the panels. Note that the time axis is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter. Rms error of the fit is 0.33 mOD

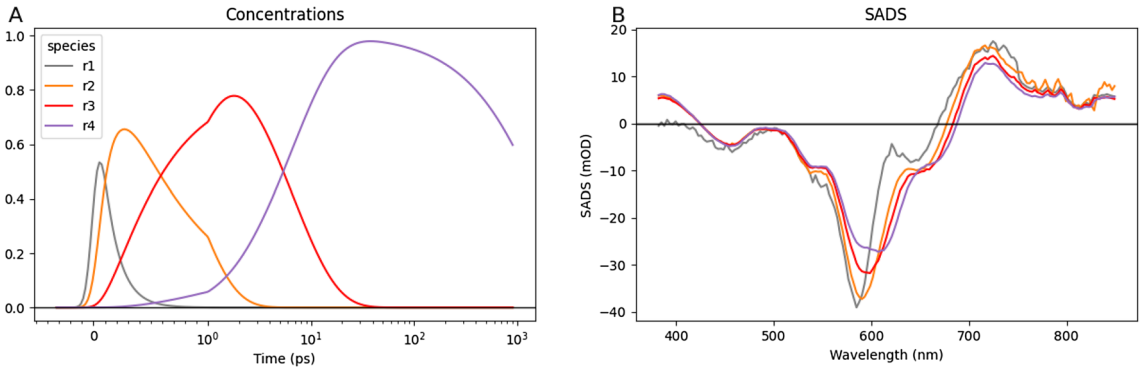


Fig. 13 Populations **A** and SADS (**B**, in mOD) of the four sequential compartments of **rc** in CH_2Cl_2. Key: gray, orange, red, purple: successively relaxed $\mathbf{r}^*$ states. Note that the time axis in (**A**) is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter

model is unsatisfactory since the left and right singular vectors of the residual matrix (Fig. 14A, B) show large trends, especially during the first 10 ps around 590 and 780 nm.

Therefore, a loss process is introduced: the formation of a radical pair state, called **rcgRP,** from $\mathbf{r_1}$ or $\mathbf{r_2}$, which requires two new rate constants $k_{rcgRP,r1}, k_{rcgRP,r2}$. Thus, the **rcg** system can be described by the four **rc**-SADS, the **g**-SADS and the **rcgRP**-SADS, and the twelve rate constants: $k_{r2,r1}, k_{r3,r2}, k_{r4,r3}, k_{r4}; k_{g,r_1}, k_{g,r_2}, k_{g,r_3}, k_{g,r_4}; k_{rcgRP,r1}, k_{rcgRP,r2}$ and the decay rates to the ground state $k_g, k_{\mathbf{rcgRP}}$. This kinetic scheme is schematically depicted in Fig. 15B and the differential equation is shown in Figure S10.

The main results from the **rc** in CH_2Cl_2 experiment are the four rate constants (Fig. 13A) and the four SADS (Fig. 13B). The four rate constants are used in the kinetic scheme of **rcg** (Fig. 15B). The estimated **rc**-SADS are used to guide the SADS of the $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$, $\mathbf{r_4}$ species in the target analysis of **rcg** by adding the **rc**-SADS as data to be fitted using the **rcg-**SADS. This is demonstrated in Fig. 16, where the dashed lines indicate the fit. The formulas for the fitting of the guidance SADS are shown in Figure S10. The usage of the guidance spectra allows for some flexibility, to accommodate small differences in the experimental conditions or the wavelength calibration or the white light of the probe, when the experiments have been performed on different days. It circumvents the more complicated simultaneous analysis of multiple datasets by selectively adding only the relevant information, i.e., here the four **rc**-SADS and the

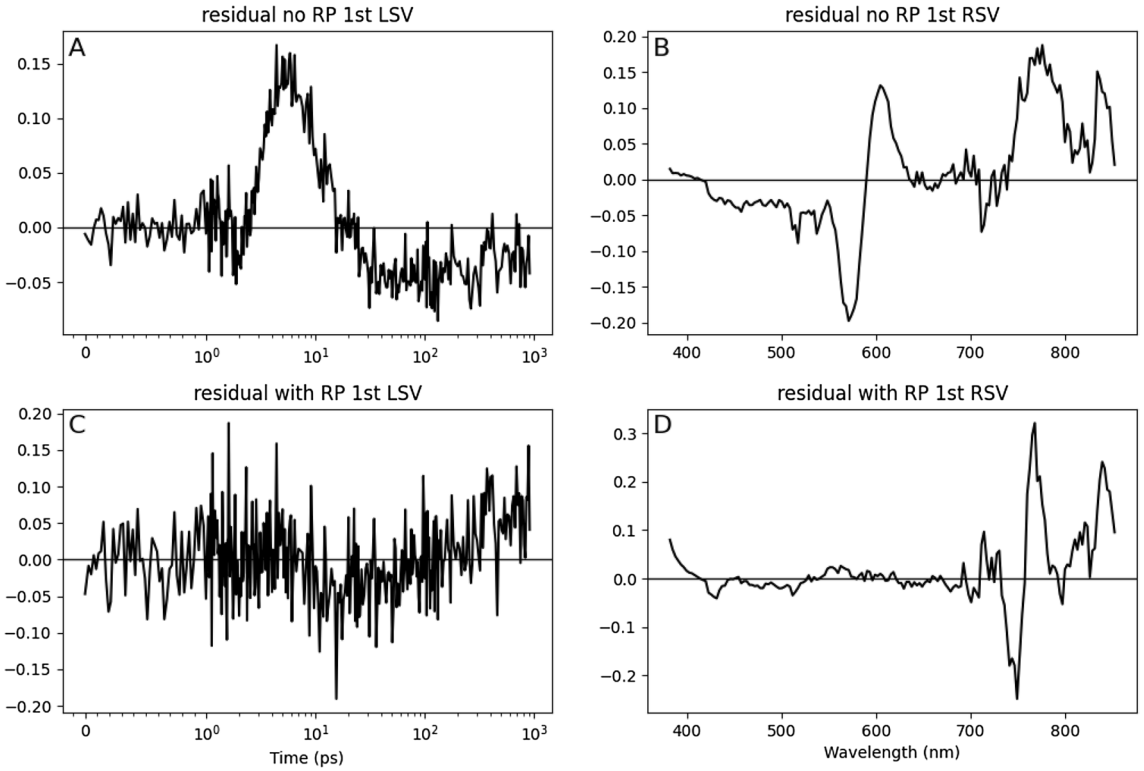


Fig. 14 First left and right singular vectors resulting from the singular value decomposition (SVD) of the residual matrix of **rcg** in CH_2Cl_2 resulting from a target analysis using a kinetic scheme without (**A**, **B**) or with (**C**, **D**) the **rcg** radical pair state. Note that panels (**A**, **B**) show large trends, whereas panels (**C**, **D**) show only small trends. The rms error of the fit decreases from 0.27 to 0.23 mOD. Note that the time axis in (**A**, **C**) is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter

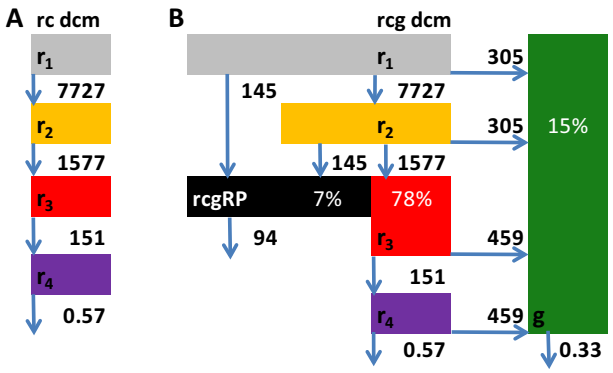


Fig. 15 Kinetic schemes used for the simultaneous target analysis of **rc** (**A**) and **rcg** (**B**) in CH_2Cl_2. All rate constants in ns^{-1}. Key: gray, orange, red, purple: successively relaxed $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$, $\mathbf{r_4}$; dark green: **g**; black, **rcgRP**: **rcg** radical pair. Vertical arrows indicate relaxation of an excited state, or decay. Horizontal arrows represent energy transfer to **g**

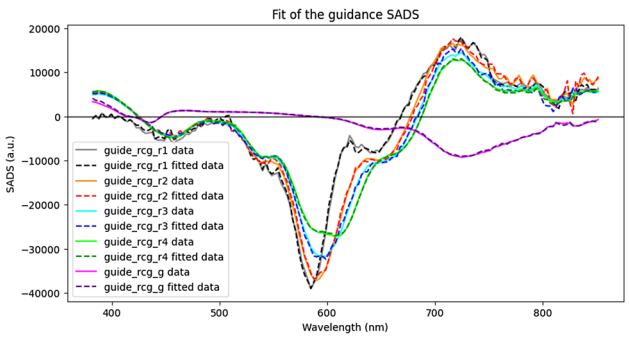


Fig. 16 Fit of the guidance SADS used in the target analysis

g-SADS. Without the guidance SADS it would have been impossible to take the **r*** spectral evolution properly into account in the **rcg** target analysis, since, e.g., the population of $\mathbf{r_4}$ is very small (purple in Fig. 17A).

rcg shows the typical spectral evolution of the **r** chromophore, as well as EET to **g** (see the 730 nm bleach in the **g** SADS) and $\mathbf{r}^{-\cdot}$ formation, characterized by the 575 nm bleach and the 780 nm absorption (black SADS). The trichromophoric systems **rcgcr** and **gcrcg** (Figure S9) can be analyzed with slightly modified versions of this kinetic scheme resulting in the concentrations depicted with dotted and dashed lines in Fig. 17A. Note that in **gcrcg** which contains two accepting chromophores, the **g** population (dashed green) rises faster, but also that of the **rcgRP** (dashed black),

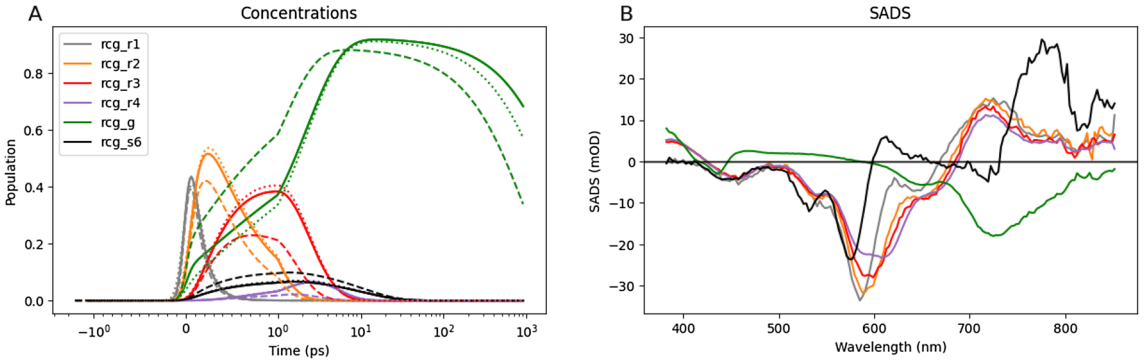


Fig. 17 Populations (**A**) of the target kinetic schemes from Fig. 15B (for **rcg**) and estimated SADS (**B**, in mOD) of the simultaneous refined target analysis of **rcg** (solid), **rcgcr** (dotted) and **gcrcg** (dashed) in CH_2Cl_2. Key: gray, orange, red, purple: successively relaxed $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$, $\mathbf{r_4}$ states; dark green: **g**; black, **rcgRP**: **rcg** radical pair. Note that the time axis in (**A**) is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter

cf. also the amplitude matrices in the **gcrcg** Jupyter notebook in the SI. These results are further discussed in [24].

3.3 Time-resolved emission case study of whole photosynthetic cells

The time-resolved emission spectrum of whole photosynthetic cells contains the contributions from all the pigment–protein complexes present. In cyanobacteria the phycobilisome (PB) is the light harvesting antenna, which contains phycocyanin (PC) and allophycocyanin (APC) pigments that absorb the light between 400 and 650 nm. The excitations of the antenna pigments are efficiently transferred to the chlorophyll-containing photosystems (PS) I and II [44]. Megacomplexes consisting of PB, PSI, and PSII have been demonstrated [45]. ΔPSI mutants which lack PSI [46] have been used as a model system to study the properties of the PB–PSII megacomplex [30]. PSII shows different properties when the reaction center (RC) is in the open or in the closed state [47]. To model the EET rates in a whole cell, we, thus, distinguish three basic types of megacomplexes (Figure S11): PB–PSII with PSII in the open or the closed state and non-transferring PB. Free PSII, not receiving PB input, cannot be distinguished from the PSII in a PB–PSII megacomplex. The minimal kinetic scheme of PB consists of ten compartments [48]: three core cylinders with a connected rod. Each rod consists of PC640 (cyan) and PC650 (blue) compartments, the top cylinder contains 24 APC660 pigments (magenta), the two basal cylinders consist of disks with only APC660 pigments (red) and disks with APC660 (orange) and APC680 pigments (black). APC680 is the terminal emitter that transfers the PB excitations to PSII. The biexponential decay of the PSII dimer emission is described by an equilibrium of a Chl *a* compartment (PSII open, green) with a radical pair (RP) compartment [47, 49]. Spectral equality constraints are employed linking the SAS of the

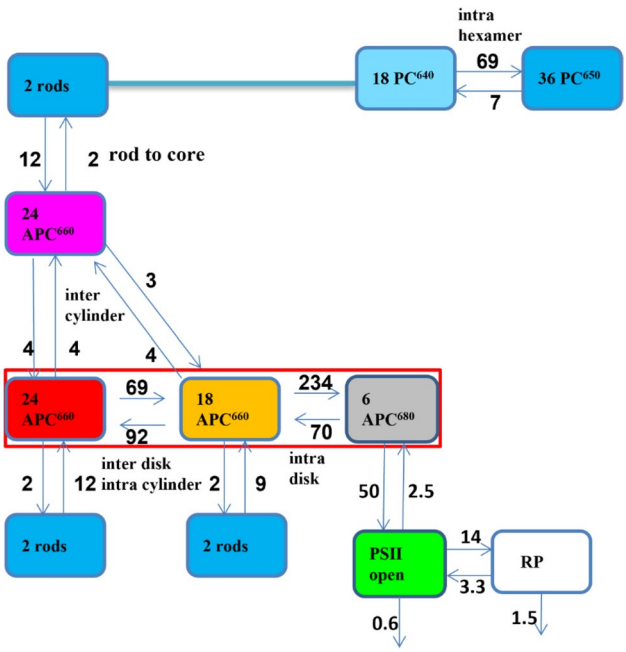


Fig. 18 Minimal kinetic scheme of the PB–PSII complex with RCs open at room temperature (RT). Key: PC640 (cyan), PC650 (blue), APC660 (red), APC680 (black), and PSII Chl a (green). Functional compartmental model, with a zoom out of a rod consisting of three lumped hexamers in the upper right. The magenta APC660 compartment represents the top cylinder. The red rectangle indicates the two basal cylinders. All microscopic rate constants are in ns^{-1}. The common decay rate constant for excited PC and APC states of 0.78 ns^{-1} has been omitted for clarity

PC640, PC650, APC660 compartments. Thus, together with APC680 and PSII there are only five different SAS. The RP SAS is by definition zero. Spectral area constraints [31] have been used to estimate the equilibria. The parameters of the PB model have been taken from [48].

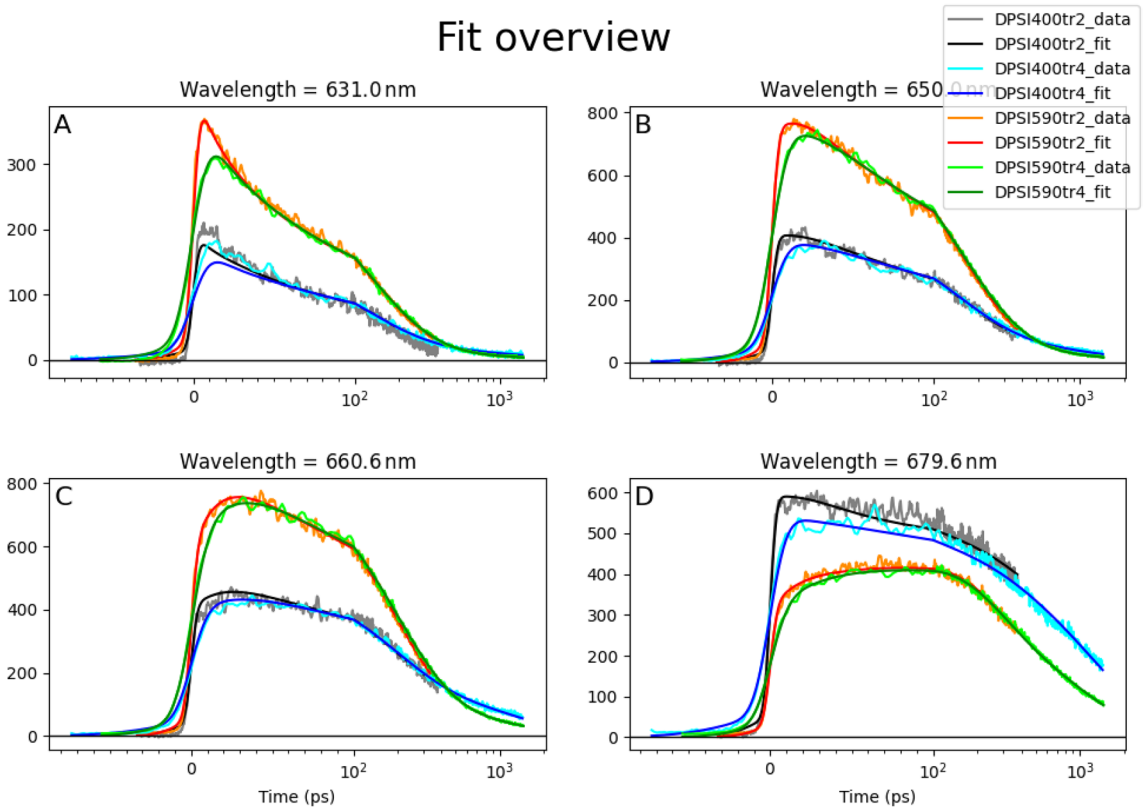


Fig. 19 Selected time traces of the emission at 4 wavelengths (indicated in the title of the panels) after 400 or 590 nm excitation at RT. Key: 400 TR2 (gray), 400 TR4 (cyan), 590 TR2 (orange), 590 TR4 (green). Black, blue, red, and dark green lines indicate the simultaneous target analysis fit of the four data sets. Note that the time axis is linear until 100 ps and logarithmic thereafter. Note also that each panel is scaled to its maximum. Overall rms error of the fit was 8.15

To collect enough information four experiments have been done, preferentially exciting the PB (590 nm excitation) or the PSII Chla (400 nm excitation), with a shorter or longer time range (TR2, IRF ≈7 ps FWHM and TR4, IRF ≈18 ps FWHM). Representative traces demonstrating the excellent quality of the fit are shown in Fig. 19. From the simultaneous target analysis of the four experiments, the rate of energy transfer from PB to PSII can be estimated (Fig. 18), together with the SAS of the five different species (Fig. 20), and the fractions of the different complexes (Table 1). They are megacomplex scaling parameters of the model, cf. the **dPSI** Jupyter notebook in the SI.

The PB–PSII megacomplex contains ≈500 chromophores of five different types PC640, PC650, APC660, APC680 and Chl *a* (in PSII). The properties of the estimated SAS (Fig. 20B, D) are in agreement with the literature [47, 50]. The main finding of this case study is the rate of EET from APC680 to PSII in vivo of 50 ns^{-1}. Pyglotaran enables the combination of the kinetic models for PB [50] and for PSII [47, 49], cf. Figure 18. These results are further discussed in [30].

3.4 Conclusion from the case studies

Since with these complex systems, it is unfeasible to directly fit the data by a theoretical simulation, our global and target analysis methodology provides a useful 'middle ground' where the theoretical description and the fit of the experimental data can meet (Fig. 1). In the first case study, we employed DOAS and IRFAS to describe the vibrational wavepackets. Theoretical chemistry computations then complemented the interpretation of the target analysis results [25]. In the second case study, computations [42] confirmed that the estimated energy transfer rates from the red to the green chromophore are in agreement with the Förster resonance energy transfer mechanism. Such computations are not possible with the PB–PSII megacomplex which contains ≈500 chromophores. Here, the functional compartmental model (Fig. 18) is the best possible theoretical description.

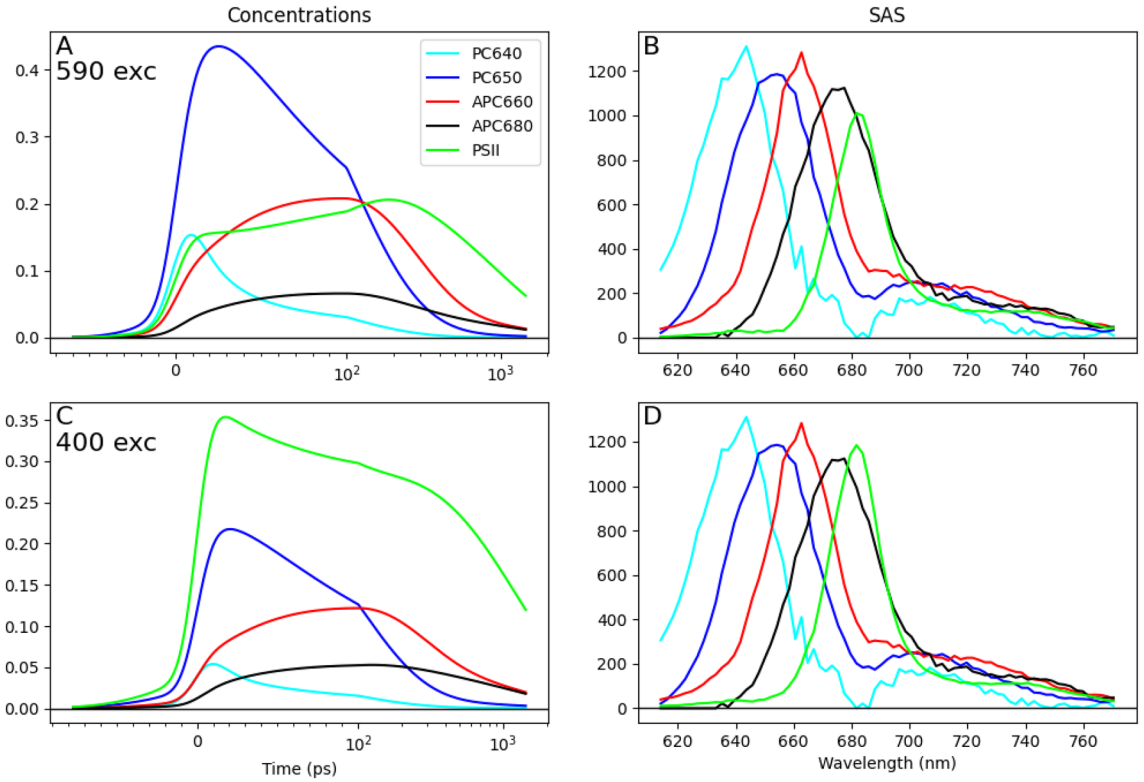


Fig. 20 Target analysis of the PB–PSII complex at RT. Total concentrations and SAS estimated after 590 (**A**, **B**) or 400 (**C**, **D**) nm exc. Key: PC640 (cyan), PC650 (blue), APC660 (red), APC680 (black), and PSII Chl a (green). Note that the time axis is linear until 100 ps and logarithmic thereafter

Table 1 Estimated fractions of the different complexes in the experiments at RT in the four experiments (in acquisition order)

	590 exc	590 exc	400 exc	400 exc
	TR2 (%)	TR4 (%)	TR2 (%)	TR4 (%)
PB–PSII open	84	82	51	50
PB–PSII closed	12	10	28	33
Non-transferring PB	4	8	21	17

4 Design of the lego-like problem-solving environment pyglotaran

The design of pyglotaran is based upon the well-known cycle of scientific discovery *model specification–parameter estimation–model validation* [16]. This cycle is illustrated in Fig. 21 and summarized for the **rc** and **rcg** case study in Table 2.

4.1 Model specification

The model specification in pyglotaran is designed to be lego-like, allowing for the easy declaration and reuse of building blocks. The core of pyglotaran is the modeling language which is a declarative domain-specific language (DSL) that is designed to describe the behavior of systems in terms of their states and how they interact with one another, in a modular and composable manner. A DSL enables a user unfamiliar with scientific modeling, computing, and programming to express the analysis of complex systems without detailed knowledge about the interiors of pyglotaran. Pyglotaran is a very general implementation of separable problems, which enables usage beyond the kinetic models presented here. The DSL is split up into two parts, the parameter definitions and the model definitions referencing parameters from the parameter definitions by their name. Using the DSL, pyglotaran functions as an engine that interprets the model and parameter definitions and applies them to fit the data. To reduce the mental load for users and simplify the translation between the kinetic model and its description, pyglotaran allows and encourages to use meaningful and verbose names both in the model and in the parameter definitions (Figure S12). The DSL is further detailed in the SI section *Modeling language* and illustrated in the Jupyter notebooks. The plain text model description feature allows for the use of version control software (e.g., git), ensuring that all changes are tracked and recorded.

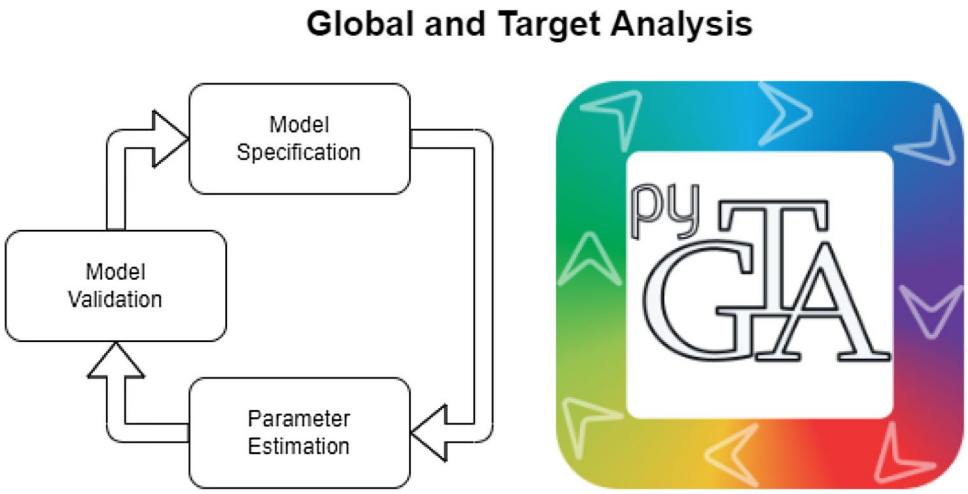


Fig. 21 At the left, the model specification–parameter estimation–model validation cycle of scientific discovery. The logo on the right-hand side symbolizes the crucial role of pyglotaran in this cycle

Table 2 The scientific discovery cycle of the **rc** and **rcg** case study

Model specification–parameter estimation	Model validation
rc raw data, **introducing** two elements: the sequential scheme with 4 compartments (4 SADS) and the coherent artifact (3 IRFAS), and the dispersion of the location of the maximum of the IRF	Residual analysis indicates the pre-zero baseline (Figure S5, Figure S6), after correction for this baseline resulting in the (pre-processed) **rc** data
rc data target analysis with spectral equality constraints to regularize the first SADS	Residual analysis indicates satisfactory residuals (Fig. 12, Fig. 14) and interpretable SADS (Fig. 13), IRFAS (Fig. 11) and kinetic parameters (Fig. 15A)
First target of the **rcg** system, introducing a kinetic scheme with 5 compartments where each **r** compartment transfers energy to **g**. New elements are the four guidance spectra estimated from **rc** (Fig. 16)	Residual analysis indicates large trends in the first left and right singular vectors of the residual matrix (Fig. 14A,B)
Second target of the **rcg** system, introducing a 6^{th} compartment (**rcgRP**) (Fig. 15B)	Residual analysis indicates no more significant trends in the first left and right singular vectors of the residual matrix (Fig. 14C,D). However, this single data set results in a noisy **rcgRP** SADS
Simultaneous target analysis of **rcg, gcrcg,** and **rcgcr** (three multi-chromophoric systems, with analogous kinetic schemes for the **gcrcg** and **rcgcr** systems) results in more robust estimation of the **rcgRP** SADS	Residual analysis indicates that the full linking of *all* the SADS results in a suboptimal fit of all data
Refined simultaneous target analysis of **rcg, gcrcg,** and **rcgcr** with linking of the **rcgRP** SADS only, results in a nice **rcgRP SADS** (Fig. 17)	Residual analysis indicates excellent fit of all data, with satisfactory residuals and interpretable SADS (Fig. 17), IRFAS and kinetic parameters. All kinetic schemes are internally consistent

4.2 Parameter estimation

The parameter estimation distinguishes nonlinear parameters (nlp) and conditionally linear parameters (clp). The clp can be either nonnegative (with SAS) or unconstrained (with SADS, A, B, IRFAS) [29]. In the model definition, this is specified using "residual_function: non_negative_least_squares" and "residual_function: variable_projection", respectively. To link the clp across multiple datasets, the option "link_clp: True" can be used.

Spectral relations between the clp can be specified, constraints to zero, and penalties based upon the area of the SA(D)S. For a relation between the clp of two components one would add to the model specification, e.g.:

```
clp_relations:
  - source: s1
    target: s2
    parameter: rel.r1
    interval: [[0, 1000]]
```

Here, the clp of s1 and s2 are related with a scaling parameter (rel.r1) over the interval from 0 to 1000.

For a zero-constraint, one would add to the model specification, e.g.:

```
clp_constraints:
  - type: zero
    target: s12
    interval: [[1, 1000]]
```

Here, the clp for the s12 component are forced to be zero in the interval from 1 to 1000.

For penalties based upon the (difference) in area of the SA(D)S [31] one would add to the model specification, e.g.:

```
clp_penalties:
  - type: equal_area
    source: s11
    source_intervals: [[1, 1000]]
    target: s1
    target_intervals: [[1, 1000]]
    parameter: area.PS2
    weight: 0.1
```

Here, the difference in the area of the clp (which are the SAS of components s11 (PS2) and s1 in this case) in the interval between 1 and 1000 is penalized, where the area of the s1 SAS is scaled with the parameter `area.PS2`. The penalty itself is scaled with a weight of 0.1 in this case before adding it to the residual vector that affects the minimization process.

The examples given above are all taken from the ΔPSI mutant emission case study where they can be studied in context. Since the clp constraints decrease the amount of the free clp parameters, they are very important in the target analysis [7, 31].

The starting values for the nonlinear parameters can be specified in two ways. Initially, with the help of a parameters.yml file analogous to model.yml file described in the SI. After optimization a csv file of the estimated nonlinear parameters can be written, which can then more easily be modified in model refinement, and subsequently be used as the new starting values for the nonlinear parameters.

The actual parameter estimation process employs the nonlinear least squares function *scipy.optimize.least_squares* [51], which is based upon an optional optimization algorithm, and is demonstrated in the Jupyter notebooks in the SI. After the fit, summary statistics are computed, most importantly the rms error of the fit and the *t*-values of the estimated nonlinear parameters (Figure S5).

4.3 Model validation

The model validation process in the pyglotaran framework is essential for ensuring that the generated models are accurately capturing the underlying dynamics of the molecular systems. This process involves a series of steps, which we outline below, along with relevant figures and supplementary information that can be found in the Jupyter notebooks provided in the SI.

1. Plotting overlays of data and fits: Visual inspection of the fitted model against the experimental data is the first step in assessing the quality of the model (Fig. 2, Fig. 12, Fig. 19). This comparison helps to identify any significant deviations between the model's predictions and the observed data.
2. Analyzing residuals: Examining the matrix of residuals for each dataset provides valuable insight into the model's performance. By plotting the first left and right singular vectors resulting from the singular value decomposition (SVD) of the residual matrix (Fig. 14), researchers can identify systematic patterns in the residuals, which may indicate potential issues with the model.
3. Inspecting *t*-values of estimated parameters: After validating the residuals, it is essential to check the *t*-values of the estimated parameters (Figure S5). Ideally, *t*-values should be larger than two, indicating that the parameters are statistically significant.
4. Assessing the scientific interpretability of nonlinear parameters (nlp) and conditionally linear parameters (clp): Once the fit's residuals and estimated parameters are deemed acceptable, researchers must evaluate whether the obtained nlp and clp are scientifically interpretable. This process often marks the beginning of a new round in the scientific discovery cycle (Fig. 21), where researchers refine their models and hypotheses based on the insights gained from the analysis.

4.4 Reporting and conclusion

An essential aspect of the scientific process, not fully covered by Fig. 21 is the effective reporting and communication of the results. Clear and concise reporting of the models, their parameters, and validation outcomes is crucial for the broader scientific community to understand, evaluate, and build upon the findings. In this context, the Jupyter notebook-style interface of pyglotaran proves to be highly advantageous.

The Jupyter notebook-style interface, typically implemented through Jupyter notebooks, allows researchers to combine code, output, visualizations, and descriptive text in a single, interactive document. This format greatly facilitates the reporting process by enabling researchers to:

1. Document their work in a transparent and reproducible manner: The Jupyter notebook interface makes it easy to share the complete analysis pipeline, from data pre-

Table 3 Feature comparison of glotaran + TIMP and pyglotaran

	Glotaran + TIMP	pyglotaran
User interface	Glotaran is the GUI for the R-Package TIMP	Use through external tools (e.g., VS Code, Jupyter notebooks)
Usability	GUI; drag and drop modeling, double-click to plot	Scripts (notebooks) and text-based modeling files lab journal-like experience of working
Core functionality	Global and target analysis Limited support for multiple datasets Basic kinetic modeling Basic parameter relations	Global and target analysis Designed to support multiple datasets Advanced kinetic modeling with DOAS, IRFAS and guidance spectra Mathematical expressions in parameter relations
Integration	Self-contained; limited import/export functionality Only access to a subset of internal data	Easy to integrate with the entire Python ecosystem Direct access to all internal data
Plotting	Limited number of plots with very limited customizability	Fully customizable plots and easy access to the raw data to make non-standard plots
Development	Limited maintenance only	Active development
Extendability	Not easily extendable; needs changes to Glotaran (Java) and TIMP (R) and the communication layer (RServe)	Designed to be extendable via a plugin system Python knowledge required
Support	No support, only critical bugs Slow response, Email	Active community GitHub issue tracker

processing to model validation, with colleagues and collaborators.

2. Visualize and explain their results: Pyglotaran's integration with popular Python plotting libraries, such as matplotlib [52], allows for the creation of compelling and informative visualizations. Researchers can seamlessly incorporate these visualizations into the Jupyter notebook, alongside explanations of their significance and interpretation.
3. Collaborate and share their findings: Jupyter notebooks can be easily shared with collaborators, who can then review, modify, or extend the analysis. Moreover, the Jupyter notebook format is conducive to sharing research findings in online repositories or supplementary materials, allowing for greater visibility and accessibility of the results.

The Jupyter notebook-style interface plays a pivotal role in streamlining the reporting process, fostering collaboration, and promoting transparency in the scientific discovery process. By enabling researchers to effectively communicate their findings, pyglotaran not only contributes to the advancement of time-resolved spectroscopy analysis but also supports the broader scientific community in uncovering new insights and understanding of complex molecular systems.

4.5 Software engineering: the pyglotaran ecosystem

The development of pyglotaran [22] is standing on the shoulders of giants in multiple ways. On the one hand, it benefits from decades of knowledge and lessons learned by interacting with users of predecessor software TIM [23], TIMP [16], and Glotaran [18]. On the other hand, it relies on battle-proven Python scientific libraries like scipy [51], numpy [53], numba [54] and xarray [55], instead of trying to reinvent the wheel.

To ensure efficient development and high-quality code, several key practices have been implemented in the development of the pyglotaran [22] ecosystem (see also the SI section *Software development*):

1. **Version control**: Managed through Git and GitHub, using the GitHub flow model and branch protection to manage changes and ensure code quality. This enables multiple developers to collaborate on the codebase simultaneously while maintaining version history and control over changes.
2. **Organization**: All development happens within the Glotaran organization on GitHub, which, besides the new Python projects, also contains the legacy projects Glotaran [18] and TIMP [16]. These legacy projects are still maintained but not further developed. The most notable components of the pyglotaran ecosystem include pyglotaran extras, pyglotaran examples, and pyglotaran validation, which together form the basis for the pyglotaran validation framework.
3. **Code structure**: Code is organized using packages and modules, making it easier to navigate and manage the codebase.
4. **Quality assurance**: Linters, formatters, and type checkers are used to catch errors and enforce consistency in the code.

5. **Continuous Integration and Delivery (CI/CD)**: These processes automate the building and testing of the software, ensuring that the code is always in a working state (Figure S15, Figure S16, Figure S17, Figure S18).
6. **Documentation**: Automated documentation, both generated and manually curated, is provided to facilitate understanding of the codebase and help new users get up to speed quickly.
7. **Dependency management**: Automated dependency updates ensure that the software remains up-to-date with the latest libraries/frameworks and potential problems are discovered early.
8. **Deployment**: Handled through PyPI and Conda-forge, making it easier for users to install and use the software.

Overall, these practices ensure that pyglotaran is a high-quality, well-maintained software package that is efficient to develop, test, and use.

4.6 Glotaran versus pyglotaran

The pyglotaran project was developed based on the lessons learned with glotaran + TIMP and its support. While the learning curve of pyglotaran is steeper it provides a lot more capabilities, extendability and customizability. Due to its text-based nature, it can in principle be integrated into a GUI. Table 3 contains a feature comparison of glotaran + TIMP and pyglotaran.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s43630-023-00460-y.

Acknowledgements We thank Sergey Laptenok for help with designing the more modern logo features, critical reading, and helpful discussion. We thank Artur Nenov for helpful discussion.

Data availability statement The Jupyter notebooks and the preprocessed data can be downloaded from https://github.com/glotaran/pyglotaran-releasepaper-supplementary-information/releases, so that the reader can reproduce all results.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Holzwarth, A. R. (1995). Time-resolved fluorescence spectroscopy, in *Methods in Enzymology*. *Academic Press, 246*, 334–362.
2. Kovalenko, S. A., Dobryakov, A. L., Ruthmann, J., & Ernsting, N. P. (1999). Femtosecond spectroscopy of condensed phases with chirped supercontinuum probing. *Physical Review A, 59*, 2369–2384.
3. vandeVen, M., Ameloot, M., Valeur, B. & Boens, N. (2005) Pitfalls and Their Remedies in Time-Resolved Fluorescence Spectroscopy and Microscopy, *Journal of Fluorescence*, **15**, 377–413.
4. Berera, R., van Grondelle, R., & Kennis, J. T. M. (2009). Ultrafast transient absorption spectroscopy: Principles and application to photosynthetic systems. *Photosynthesis Research, 101*, 105–118.
5. Beechem, J. M., Ameloot, M., & Brand, L. (1985). Global and Target Analysis of Complex Decay Phenomena. *Instrumentation Science & Technology, 14*, 379–402.
6. Holzwarth, A. (1996) Data analysis of time-resolved measurements, in *Biophysical Techniques in Photosynthesis*, eds. J. Amesz and A. Hoff, Kluwer Academic Press, Dordrecht, pp. 75–92.
7. van Stokkum, I. H. M., Larsen, D. S., & van Grondelle, R. (2004). Global and target analysis of time-resolved spectra. *Biochimica Et Biophysica Acta, 1657*, 82–104.
8. van Stokkum, I. H. M., Larsen, D. S., & van Grondelle, R. (2004). Erratum to "Global and target analysis of time-resolved spectra." *Biochimica Et Biophysica Acta, 1658*, 262–262.
9. Ruckebusch, C., Sliwa, M., Pernot, P., de Juan, A., & Tauler, R. (2012). Comprehensive data analysis of femtosecond transient absorption spectra: A review. *Journal of Photochemistry and Photobiology C: Photochemistry Reviews, 13*, 1–27.
10. Arcioni, A., & Zannoni, C. (1984). Intensity deconvolution in fluorescence depolarization studies of liquids, liquid crystals and membranes. *Chemical Physics, 88*, 113–128.
11. van Stokkum, I. H. M., Brouwer, A. M., van Ramesdonk, H. J. & Scherer, T. (1993) Multiresponse parameter estimation and compartmental analysis of time resolved fluorescence spectra: Application to conformational dynamics of charge-separated species in solution. *Proc. Kon. Ned. Akad. v. Wetensch.,* **96**, 43–68.
12. Hoff, W. D., Van Stokkum, I. H. M., Van Ramesdonk, H. J., Van Brederode, M. E., Brouwer, A. M., Fitch, J. C., . . . Hellingwerf, K. J., (1994). Measurement and Global Analysis of the Absorbency Changes in the Photocycle of the Photoactive Yellow Protein from Ectothiorhodospira-Halophila, *Biophysical Journal*, **67**, 1691–1705.
13. van Stokkum, I. H. M., Scherer, T., Brouwer, A. M., & Verhoeven, J. W. (1994). Conformational dynamics of flexibly and semirigidly bridged electron donor-acceptor systems as revealed by spectrotemporal parametrization of fluorescence. *Journal of Physical Chemistry, 98*, 852–866.
14. Beechem, J. M. (1989). A second generation global analysis program for the recovery of complex inhomogeneous fluorescence decay kinetics. *Chemistry and Physics of Lipids, 50*, 237–251.
15. Dioumaev, A. K. (1997). Evaluation of intrinsic chemical kinetics and transient product spectra from time-resolved spectroscopic data. *Biophysical Chemistry, 67*, 1–25.

16. Mullen, K. M., & van Stokkum, I. H. M. (2007). TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements. *Journal of Statistical Software, 18*, 1–46.
17. van Wilderen, L. J. G. W., Lincoln, C. N., & van Thor, J. J. (2011). Modelling Multi-Pulse Population Dynamics from Ultrafast Spectroscopy. *PLoS ONE, 6*, e17373.
18. Snellenburg, J. J., Laptenok, S. P., Seger, R., Mullen, K. M., & van Stokkum, I. H. M. (2012). Glotaran: A Java-based Graphical User Interface for the R-package TIMP. *Journal of Statistical Software, 49*, 1–22.
19. Slavov, C., Hartmann, H., & Wachtveitl, J. (2015). Implementation and Evaluation of Data Analysis Strategies for Time-Resolved Optical Spectroscopy. *Analytical Chemistry, 87*, 2328–2336.
20. Müller, C., Pascher, T., Eriksson, A., Chabera, P., & Uhlig, J. (2022). KiMoPack: A python Package for Kinetic Modeling of the Chemical Mechanism. *The Journal of Physical Chemistry A, 126*, 4087–4099.
21. Uhlig, J. (2022). KiMoPack - Open source tool for the analysis of transient spectral data. https://doi.org/10.5281/zenodo.6049186
22. Weißenborn, J., Snellenburg, J. J., Weigand, S. & van Stokkum, I. H. M. (2022) *pyglotaran: a Python library for global and target analysis,* https://doi.org/10.5281/zenodo.4534043
23. van Stokkum, I. H. M., & Bal, H. E. (2006). A Problem Solving Environment for interactive modelling of multiway data. *Concurrency and computation: Practice and experience, 18*, 263–269.
24. van Stokkum, I. H. M., Wohlmuth, C., Würthner, F., & Williams, R. M. (2022). Energy transfer in supramolecular calix[4]arene—Perylene bisimide dye light harvesting building blocks: Resolving loss processes with simultaneous target analysis. *Journal of Photochemistry and Photobiology, 12*, 100154.
25. Teles-Ferreira, D. C., van Stokkum, I. H. M., Conti, I., Ganzer, L., Manzoni, C., Garavelli, M., . . . de Paula, A. M. (2022). Coherent vibrational modes promote the ultrafast internal conversion and intersystem crossing in thiobases, *Physical Chemistry Chemical Physics*, **24**, 21750–21758.
26. van Stokkum, I. H. M., Kloz, M., Polli, D., Viola, D., Weißenborn, J., Peerbooms, E., . . . Kennis, J. T. M. (2021). Vibronic dynamics resolved by global and target analysis of ultrafast transient absorption spectra, *The Journal of Chemical Physics*, **155**, 114113.
27. Dobryakov, A. L., Kovalenko, S. A., & Ernsting, N. P. (2003). Electronic and vibrational coherence effects in broadband transient absorption spectroscopy with chirped supercontinuum probing. *The Journal of Chemical Physics, 119*, 988–1002.
28. Dobryakov, A. L., Kovalenko, S. A., & Ernsting, N. P. (2005). Coherent and sequential contributions to femtosecond transient absorption spectra of a rhodamine dye in solution. *The Journal of Chemical Physics, 123*, 044502.
29. van Stokkum, I. H. M., Jumper, C. C., Snellenburg, J. J., Scholes, G. D., van Grondelle, R., & Malý, P. (2016). Estimation of damped oscillation associated spectra from ultrafast transient absorption spectra. *The Journal of Chemical Physics, 145*, 174201.
30. Acuña, A. M., Van Alphen, P., Van Grondelle, R., & Van Stokkum, I. H. M. (2018). The phycobilisome terminal emitter transfers its energy with a rate of (20 ps)–1 to photosystem II. *Photosynthetica, 56*, 265–274.
31. Snellenburg, J. J., Dekker, J. P., van Grondelle, R., & van Stokkum, I. H. M. (2013). Functional Compartmental Modeling of the Photosystems in the Thylakoid Membrane at 77 K. *The Journal of Physical Chemistry B, 117*, 11363–11371.
32. Godfrey, K. (1983). *Compartmental models and their application*. Academic Press.
33. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., . . . Willing, C. (2016). Jupyter Notebooks - a publishing format for reproducible computational workflows, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, eds. F. Loizides and B. Schmidt, IOS Press, pp. 87–90.
34. Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., . . . Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship, *Scientific Data*, **3**, 160018.
35. Liebel, M., & Kukura, P. (2013). Broad-Band Impulsive Vibrational Spectroscopy of Excited Electronic States in the Time Domain. *The Journal of Physical Chemistry Letters, 4*, 1358–1364.
36. Liebel, M., Schnedermann, C., Wende, T., & Kukura, P. (2015). Principles and Applications of Broadband Impulsive Vibrational Spectroscopy. *The Journal of Physical Chemistry A, 119*, 9506–9517.
37. Golub, G. H. & LeVeque, R. J. (1979). *Extensions and uses of the variable projection algorithm for solving nonlinear least squares problems*, Proc. of the 1979 Army Numerical Analysis and Comp. Conf., ARO Report 79-3, pp. 1–12.
38. Nagle, J. F. (1991). Solving complex photocycle kinetics - theory and direct method. *Biophysical Journal, 59*, 476–487.
39. Lawson, C. L., & Hanson, R. J. (1974). *Solving Least Squares Problems*. Prentice Hall.
40. Mullen, K. M., & van Stokkum, I. H. M. (2009). The variable projection algorithm in time-resolved spectroscopy, microscopy and mass spectrometry applications. *Numerical Algorithms, 51*, 319–340.
41. Satzger, H., & Zinth, W. (2003). Visualization of transient absorption dynamics – towards a qualitative view of complex reaction kinetics. *Chemical Physics, 295*, 287–295.
42. Hippius, C., van Stokkum, I. H. M., Gsanger, M., Groeneveld, M. M., Williams, R. M., & Würthner, F. (2008). Sequential FRET processes in calix[4]arene-linked orange-red-green perylene bisimide dye zigzag arrays. *Journal of Physical Chemistry C, 112*, 2476–2486.
43. Hippius, C. (2007). Multichromophoric Arrays of Perylene Bisimide Dyes - Synthesis and Optical Properties; Multichromophore Perylenbisimidkaskaden - Synthese und optische Eigenschaften, PhD Thesis, Universität Würzburg, Fakultät für Chemie und Pharmazie, 2007.
44. Tian, L., van Stokkum, I. H. M., Koehorst, R. B. M., Jongerius, A., Kirilovsky, D. & van Amerongen, H. (2011). Site, Rate, and Mechanism of Photoprotective Quenching in Cyanobacteria, *Journal of the American Chemical Society*, **133**, 18304–18311.
45. Liu, H., Zhang, H., Niedzwiedzki, D. M., Prado, M., He, G., Gross, M. L., & Blankenship, R. E. (2013). Phycobilisomes Supply Excitations to Both Photosystems in a Megacomplex in Cyanobacteria. *Science, 342*, 1104–1107.
46. Shen, G., Boussiba, S., & Vermaas, W. F. (1993). Synechocystis sp PCC 6803 strains lacking photosystem I and phycobilisome function. *The Plant Cell, 5*, 1853–1863.
47. Tian, L., Farooq, S., & van Amerongen, H. (2013). Probing the picosecond kinetics of the photosystem II core complex in vivo. *Physical Chemistry Chemical Physics, 15*, 3146–3154.
48. van Stokkum, I. H. M., Gwizdala, M., Tian, L., Snellenburg, J. J., van Grondelle, R., van Amerongen, H., & Berera, R. (2018). A functional compartmental model of the Synechocystis PCC 6803 phycobilisome. *Photosynthesis Research*, 135, 87–102. https://doi.org/10.1007/s11120-017-0424-5
49. van Stokkum, I., (2018). Systems biophysics: Global and target analysis of light harvesting and photochemical quenching in vivo, in *Light Harvesting in Photosynthesis*, eds. R. Croce, R. van Grondelle, H. van Amerongen and I. van Stokkum, CRC Press, Boca Raton, ch. 20, pp. 467–482.
50. van Stokkum, I. H. M., Gwizdala, M., Tian, L., Snellenburg, J. J., van Grondelle, R., van Amerongen, H., & Berera, R. (2018). A

functional compartmental model of the Synechocystis PCC 6803 phycobilisome. *Photosynthesis Research, 135*, 87–102.
51. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python, *Nature Methods*, **17**, 261–272.
52. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science and Engineering, 9*, 90–95.
53. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy, *Nature*, **585**, 357–362.
54. Lam, S. K., Pitrou, A. & Seibert, S. (2015). Numba: a LLVM-based Python JIT compiler, presented in part at the Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, Austin, Texas.
55. Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*. https://doi.org/10.5334/jors.148

Authors and Affiliations

Ivo H. M. van Stokkum[1] · Jörn Weißenborn[1] · Sebastian Weigand[1,2] · Joris J. Snellenburg[1]

✉ Ivo H. M. van Stokkum
i.h.m.van.stokkum@vu.nl

[1] Department of Physics and Astronomy and LaserLaB, Faculty of Science, Vrije Universiteit Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

[2] Institut für Festkörperphysik, Technische Universität Berlin, Hardenbergstr. 36, 10623 Berlin, Germany

Supplementary Information

pyglotaran: a lego-like Python framework for global and target analysis of time resolved spectra

Ivo H.M. van Stokkum[a], Jörn Weißenborn[a], Sebastian Weigand[a,b], Joris J. Snellenburg[a]

[a]Department of Physics and Astronomy and LaserLaB, Faculty of Science, Vrije Universiteit Amsterdam, De Boelelaan 1081, 1081 HV, Amsterdam, The Netherlands

[b]Institut für Festkörperphysik, Technische Universität Berlin, Hardenbergstr. 36, 10623 Berlin, Germany

The Jupyter notebooks and the preprocessed data can be downloaded from https://github.com/glotaran/pyglotaran-release-paper-supplementary-information/releases, so that the reader can reproduce all results.

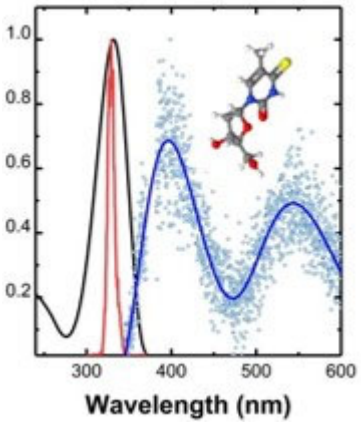


Figure S 1. The structure of 4TT (inset) with its normalized absorption spectrum (black curve), the normalized pump pulse spectrum (red curve), and normalized photoluminescence spectrum (blue curve is the fit and blue dots are the data) obtained pumping the sample at 330 nm. Figure adopted from [1].

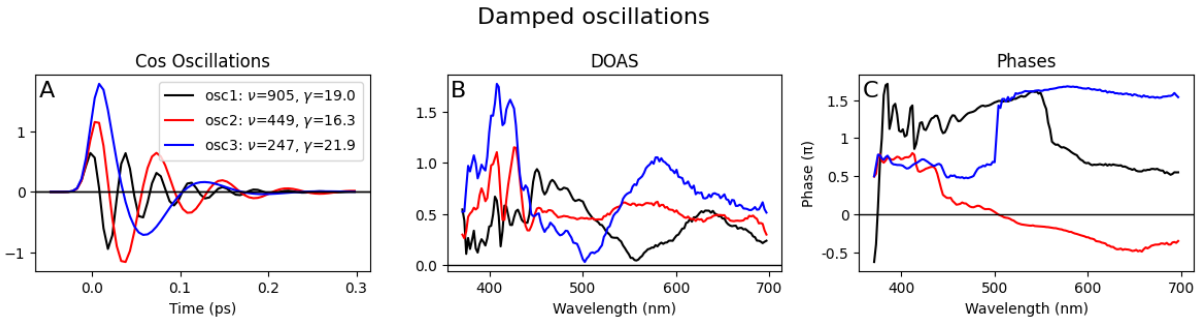


Figure S 2. Overview of the estimated DOAS and phases. (A) Cosine oscillations with frequencies $\bar{\nu}n$ (in /cm) (where n is the DOAS number) and damping rates γ (in 1/ps) written in the legend at the left, using the appropriate color. Scaling of the DOAS is such that the product of the DOAS and the damped oscillation is the contribution to the fit. (B) Estimated DOAS. (C) Estimated phase profiles of the DOAS.

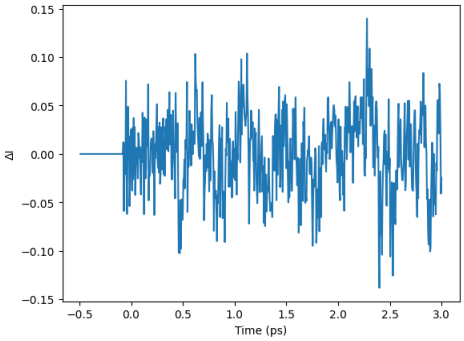


Figure S 3. Estimation of the laser intensity fluctuations responsible for the residual structure in Figure 5A by fitting the residual spectrum against the data. Further details can be found in the **4TT** Jupyter notebook.

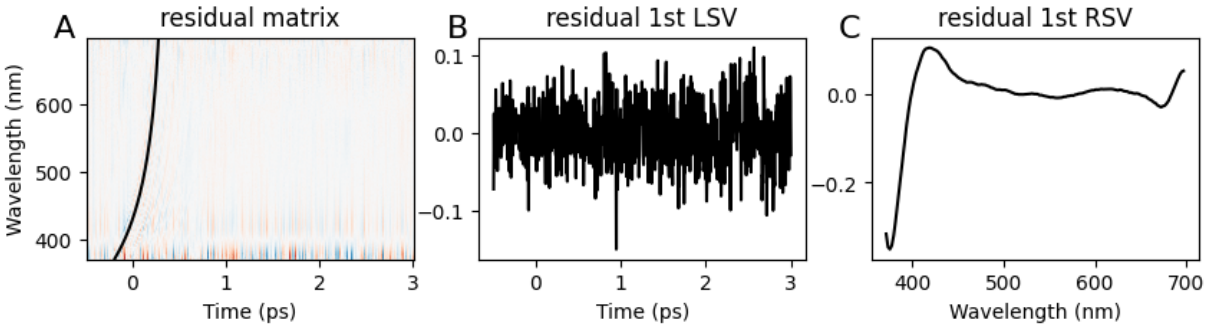
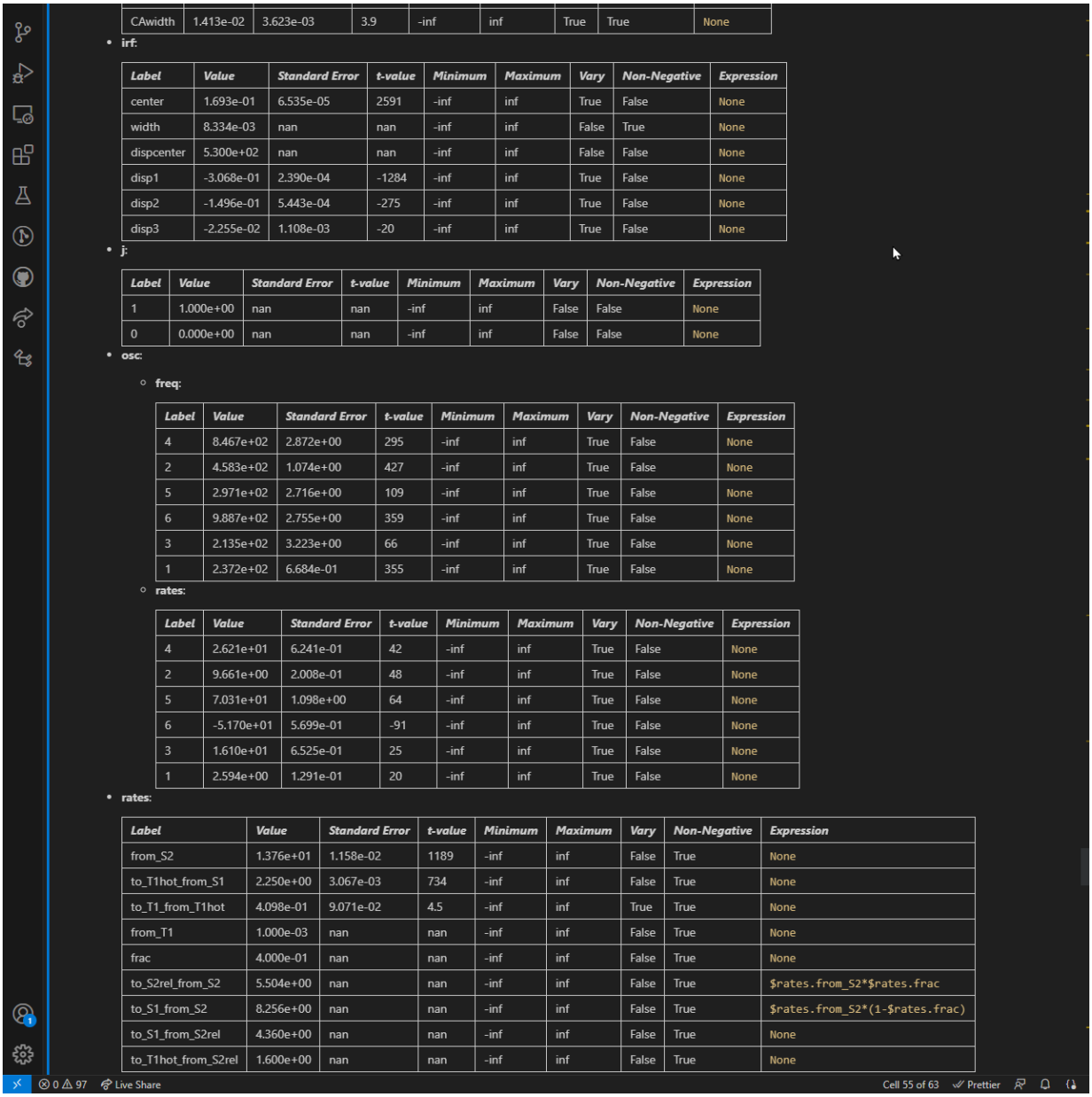


Figure S 4. First left (B) and right (C) singular vectors resulting from the singular value decomposition (SVD) of the residual matrix (A) after correcting the data for the laser intensity fluctuations. The black line in (A) represents the location of the maximum of the IRF, which is described by a third order polynomial function of the wavenumber.



CAwidth	1.413e-02	3.623e-03	3.9	-inf	inf	True	True	None

- **irf:**

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
center	1.693e-01	6.535e-05	2591	-inf	inf	True	False	None
width	8.334e-03	nan	nan	-inf	inf	False	True	None
dispcenter	5.300e+02	nan	nan	-inf	inf	False	False	None
disp1	-3.068e-01	2.390e-04	-1284	-inf	inf	True	False	None
disp2	-1.496e-01	5.443e-04	-275	-inf	inf	True	False	None
disp3	-2.255e-02	1.108e-03	-20	-inf	inf	True	False	None

- **j:**

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	1.000e+00	nan	nan	-inf	inf	False	False	None
0	0.000e+00	nan	nan	-inf	inf	False	False	None

- **osc:**
  - **freq:**

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
4	8.467e+02	2.872e+00	295	-inf	inf	True	False	None
2	4.583e+02	1.074e+00	427	-inf	inf	True	False	None
5	2.971e+02	2.716e+00	109	-inf	inf	True	False	None
6	9.887e+02	2.755e+00	359	-inf	inf	True	False	None
3	2.135e+02	3.223e+00	66	-inf	inf	True	False	None
1	2.372e+02	6.684e-01	355	-inf	inf	True	False	None

  - **rates:**

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
4	2.621e+01	6.241e-01	42	-inf	inf	True	False	None
2	9.661e+00	2.008e-01	48	-inf	inf	True	False	None
5	7.031e+01	1.098e+00	64	-inf	inf	True	False	None
6	-5.170e+01	5.699e-01	-91	-inf	inf	True	False	None
3	1.610e+01	6.525e-01	25	-inf	inf	True	False	None
1	2.594e+00	1.291e-01	20	-inf	inf	True	False	None

- **rates:**

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
from_S2	1.376e+01	1.158e-02	1189	-inf	inf	False	True	None
to_T1hot_from_S1	2.250e+00	3.067e-03	734	-inf	inf	False	True	None
to_T1_from_T1hot	4.098e-01	9.071e-02	4.5	-inf	inf	True	True	None
from_T1	1.000e-03	nan	nan	-inf	inf	False	True	None
frac	4.000e-01	nan	nan	-inf	inf	False	True	None
to_S2rel_from_S2	5.504e+00	nan	nan	-inf	inf	False	True	$rates.from_S2*$rates.frac
to_S1_from_S2	8.256e+00	nan	nan	-inf	inf	False	True	$rates.from_S2*(1-$rates.frac)
to_S1_from_S2rel	4.360e+00	nan	nan	-inf	inf	False	True	None
to_T1hot_from_S2rel	1.600e+00	nan	nan	-inf	inf	False	True	None

Figure S 5. Summary statistics of the optimized parameters, screenshot from the sequential_doas_4TT.ipynb Jupyter notebook.

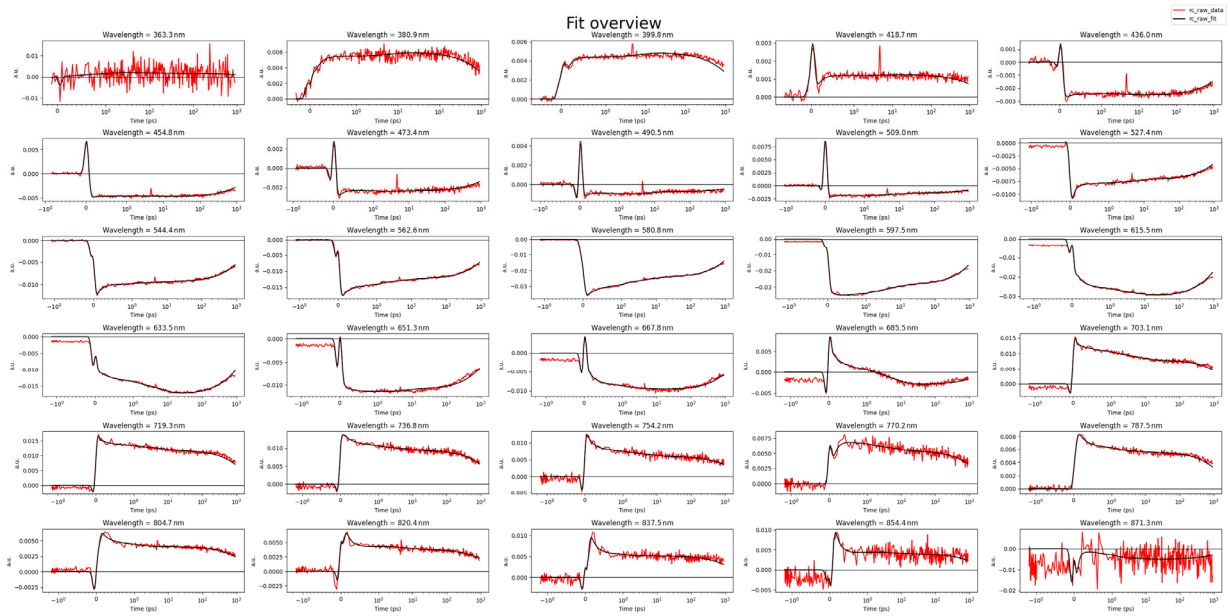


Figure S 6. Selected time traces of raw **rc** data in CH_2Cl_2 after excitation at 530 nm data (in mOD, red) and fit (black). Wavelength is indicated in the title of the panels. Note that the time axis is linear until 1 ps (after the maximum of the IRF), and logarithmic thereafter. Rms error of the fit is 0.59 mOD. Note the presence of the prezero baseline, especially with wavelengths 527, 597 nm, and higher.

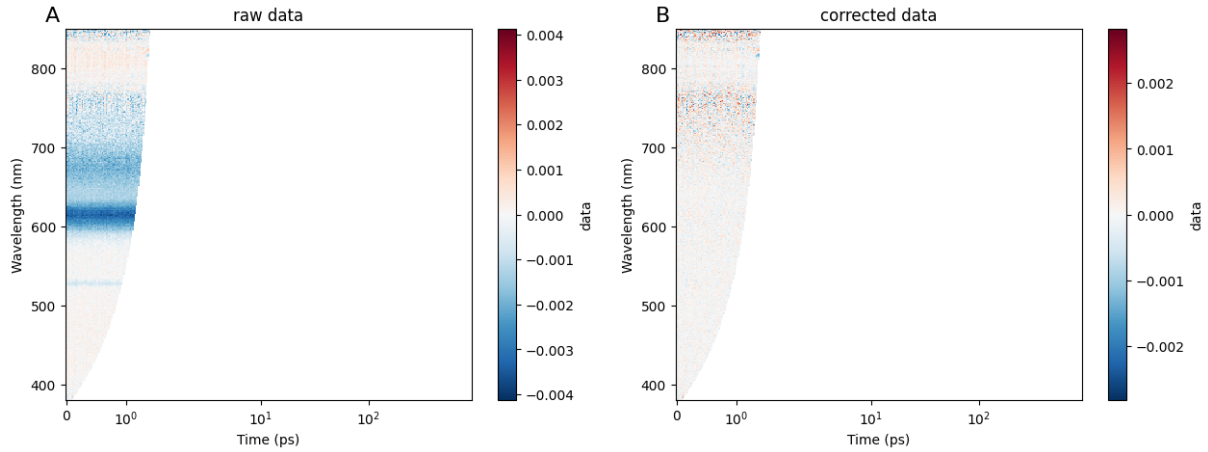


Figure S 7. Selection of the data until 0.2 ps before the center of the IRF, demonstrating the prezero baseline in the raw data (A), and the baseline corrected data (B).

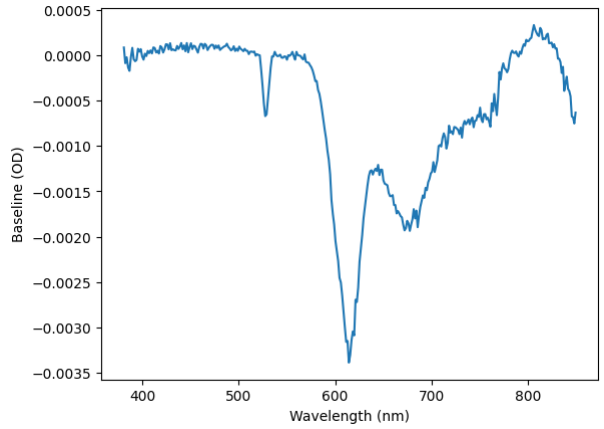


Figure S 8. Estimated prezero baseline.

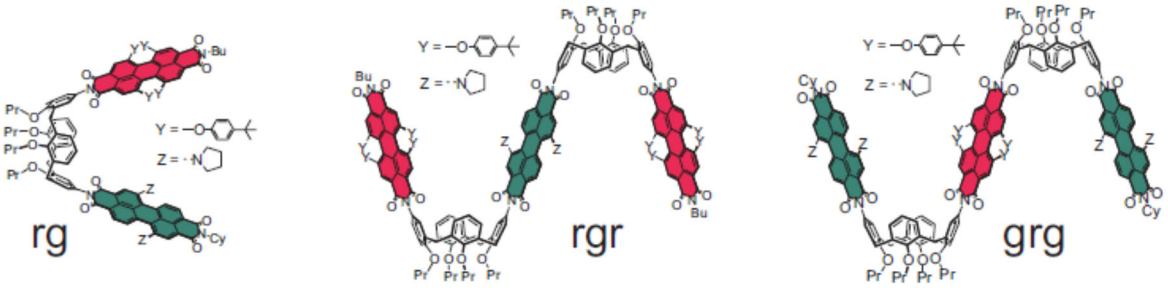


Figure S 9. Chemical structures of the supramolecular systems **rcg**, **rcgcr**, **gcrcg**. Figure adopted from [2]. Note that the **c** is omitted from the labels in the figure.

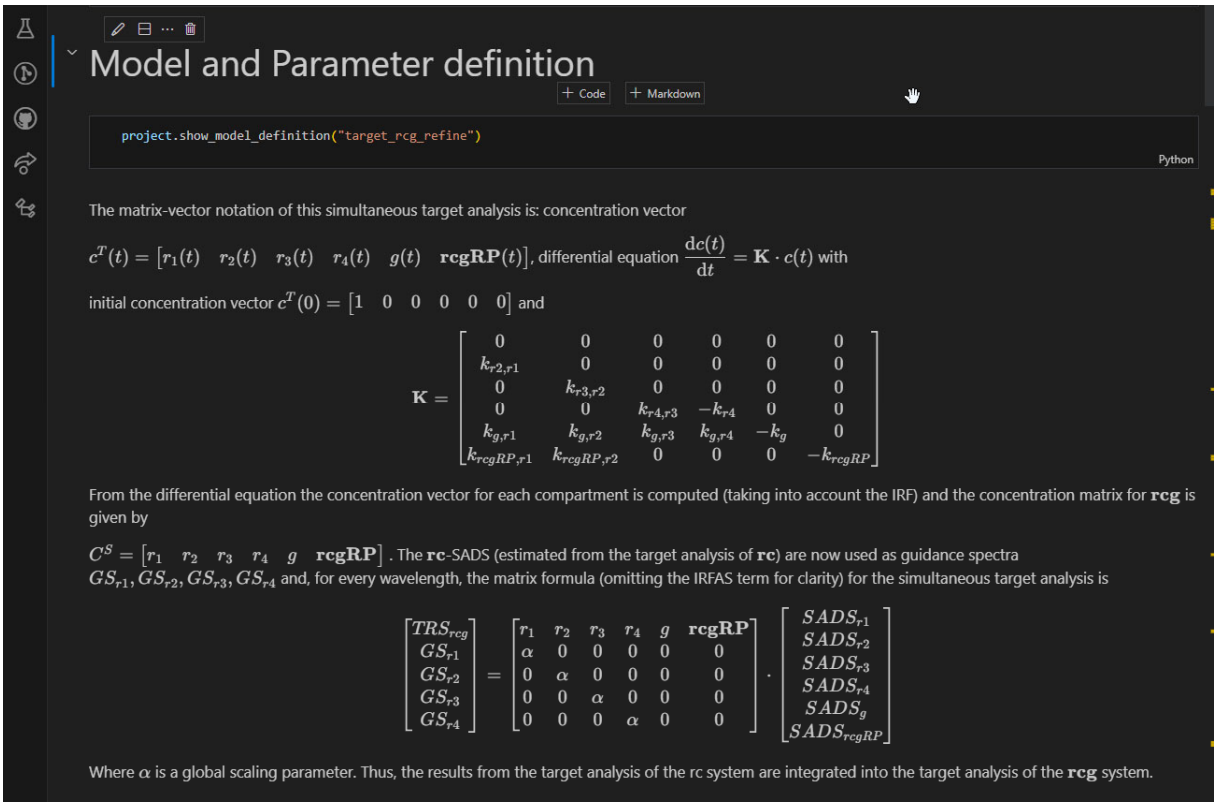


Figure S 10. Formulas for the fitting of the guidance SADS[3], screenshot from the target_rcg_compare_part2.ipynb Jupyter notebook.

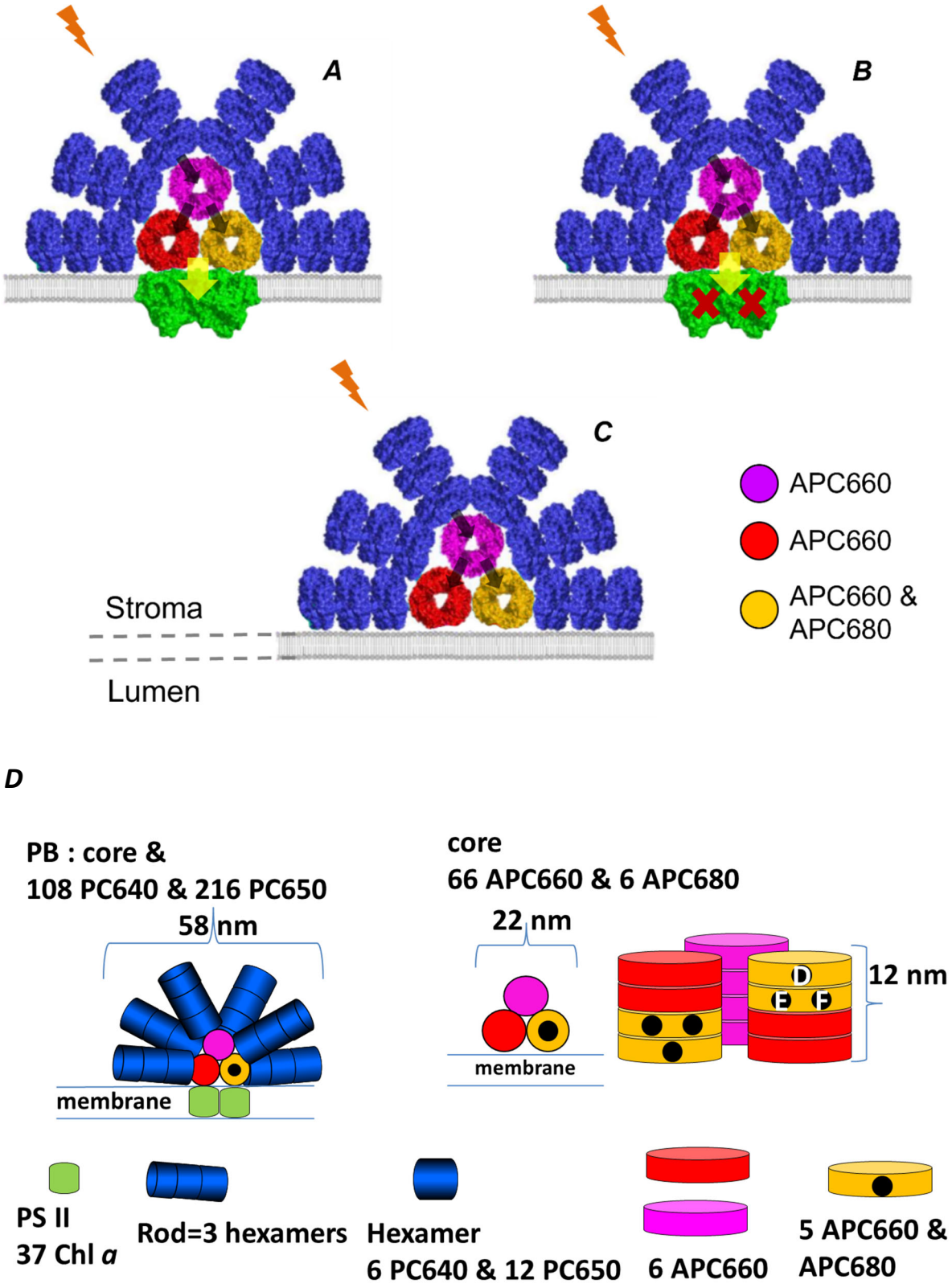


Figure S 11. Cartoon in side-view of possible fluorescent species in the *Synechocystis* ΔPSI mutant. Depicted are a PB-PSII complex with both PSII RCs open (*A*), both closed (*B*), and a PB that is not coupled to any PSII (*C*). *Key:* blue, rods consisting of three hexamers; top and basal core cylinders respectively in magenta, red and orange; green, PS II dimer. Dark arrows represent intra-PB EET; yellow arrows represent EET from the PB core to PSII. An "X" stands for a closed PSII RC. Panel D depicts the location of the different pigments in the structure. The letters D,E,F indicate the three different APC680 pigments. The approximate length for each subunit is based on [4]. Figure adopted from [5].

Modeling language

The core of pyglotaran is the modeling language which is a declarative Domain-Specific Language (DSL) that is designed to describe the behavior of systems in terms of their states and how they interact with one another, in a modular and composable manner. The DSL is split up into two parts, the parameter definitions and the model definitions referencing parameters from the parameter definitions by their name. Using the DSL pyglotaran functions as an engine that interprets the model and parameter definitions and applies them to fit the data.

Expressive user defined names

Traditionally Problem Solving Environment (PSE) like the pyglotaran predecessors Glotaran [6] and TIMP [7] use an index based description for models and parameters definitions. While this is very close to the mathematical description and implementation using matrixes and vectors, it puts a lot of mental load on the user who needs to keep track of what which index means and constantly translate back and forth between an index based description and the kinetic model describing the same physical system.

To reduce the mental load for users and simplify the translation between the kinetic model and its description pyglotaran allows and encourages to use meaningful and verbose names both in the model as well as in the parameter definitions (Figure S 12).

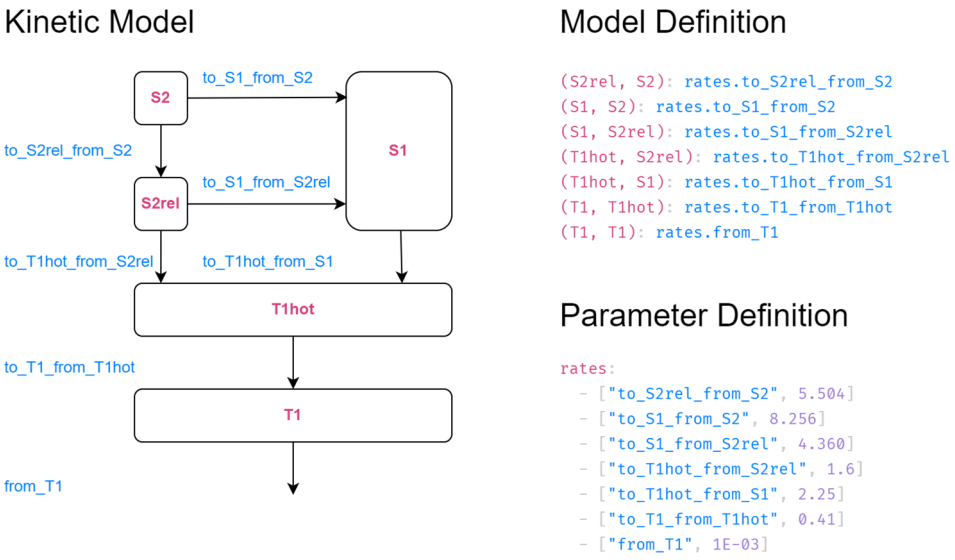


Figure S 12. Schematic illustrating the relation between the kinetic model (see also Figure 6) and the corresponding parts of pyglotaran model and parameter definitions for the 4TT case study.

Model definition structure

Since pyglotaran and its DSL are under active development and considering that the DSL can be extended via the pyglotaran plugin system it is more expedient to explain the core concepts in an abstract manner than to discuss the concrete syntax of the current model definition implementation.

In its core the model definition consists of two main parts: the model element definitions and the dataset definitions (Figure S 12). The model element definitions define reusable elements which can be combined for each dataset in the dataset definitions describing the data measured in that dataset. The parts of the model definition mostly follow a nested mapping (key-value pairs) pattern where the keys alternate between keywords defining the functionality and free user defined names which provide the meaning. This groups functionally similar parts together, with the most top-level key always being a keyword. Keywords can be categorized into required and optional keywords, where required keywords are mandatory for an element to provide its base functionality and optional keywords can further customize the functionality.

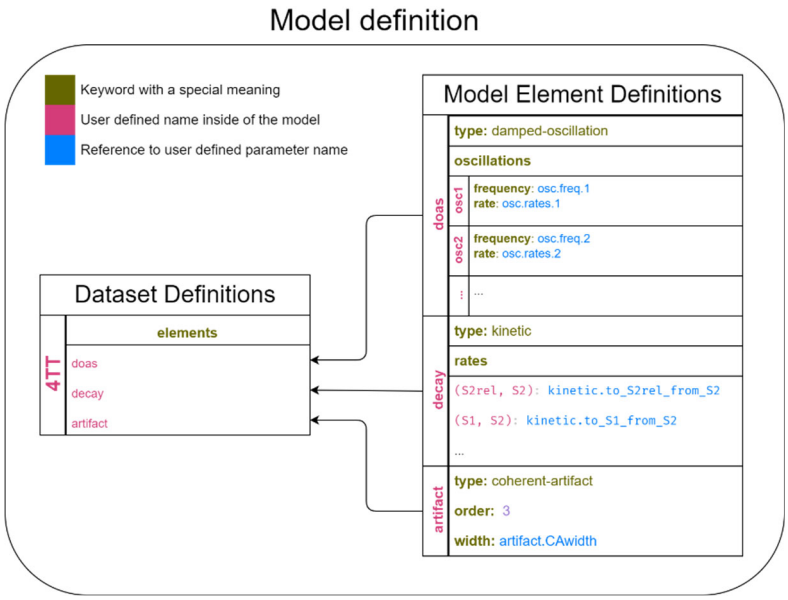


Figure S 13. Abstract schematic of the model definition structure illustrating the relation between model element definitions and dataset definitions as well as the nested alternating pattern of keywords and user defined names and references to parameter names.

The most notable required keyword of model elements is the *type* which defines the overall functionality of the element, and which other keywords can be used with it. While the value for the *type* is always a string, what values can be used for it depends on the installed plugins that define the value by which they are referred to in their implementation. Pyglotaran already comes with *built-in* model element plugins for the most common analysis needs in time-resolved spectroscopy which don't require an additional installation and can be used out-of-the-box.

What structure the value for a keyword key should have depends on the *type* of the element itself. The value of *oscillation* key in the *doas* element (Figure S 13) for example expects a nested mapping where the keys are user defined names for oscillations and the value is a mapping with the keywords *frequency* and *rates* as keys and the values being references to parameter names in the parameter definition for the corresponding values. While the value of *rates* key in the *decay* element expects a mapping where the keys have the form (*<user-defined-compartment-name>*, *<other-user-defined-compartment-name>*) in to-from-notation and the values reference the name of the corresponding transition rate parameter. Whereas the keys in the *artifact* element don't use a mapping at all but expect an integer value between 1 and 3 in case of the *order* key and a reference to the name of a parameter in case of the *width* key.

The reference implementation of the file format used for the model definition uses the YAML markup language which is designed to be intuitive and easy to read, using indentation to indicate nesting and a minimal use of punctuation. But which file formats are supported can easily be extended using the plugin system (described below) for model file reading and writing.

Parameter definition

The parameter definition connects the name a parameter is referred to with the value which will be used as the starting value in the optimization process and allows to define additional options for the parameter. The naming of the parameter is free to the user's desire with very few restrictions like that it cannot be a reserved keyword in the python programming language and that only ASCII characters are allowed. For a better structuring the parameter naming allows the creation of groups to bunch up parameters that contextually belong together making it easier to focus on those, while the group names provide the verbosity to determine what the intended usage of the parameter is. When

referring to a parameter the group it belongs to become part of its name with a dot (.) denoting the parent child relationship between parent group and child.

Two input styles for the parameter definition are supported: a nested input style using the YAML markup language and a flattened table like style input using the format CSV, xlsx or ods. The advantage in using the nested YAML syntax is that groups of parameters which belong contextually together are also visually grouped due to the indentation based YAML syntax, options can be applied to a whole group in a single line and the short name (name inside of a group) can be used which simplifies visual differentiation between parameters. The advantage of the table style is that it uses the full name of the parameter including the groups which makes it easier to look it up in the model definition and search for it.

If a parameter should not be changed during optimization the *vary* option can be set to *false*, which will exclude this parameter from the optimization. This is useful for parameters like the initial input to the compartments, to manually help the optimizer to get out of a local minimum when it gets stuck or to speed up optimization when focusing to improve a selected part of the model.

To set boundaries for the parameters that their values should not pass the *minimum* and *maximum* options can be used when using an algorithm in the optimization that supports boundaries.

The most powerful feature of the parameter definition is the usage of an *expression* allowing to define relations between parameters using equations. This can be used to enforce prior knowledge of a systems behavior, reducing the number of free parameters since this parameter is implicitly set to not be varied which consequently also improves the optimization.

Structure of the result and relation to the model definition

Optimization results generated by pyglotaran can be categorized into overall results for the whole analysis and per dataset results in the *data* attribute (Figure S 13). Each dataset result can be further categorized into a general section and one or more element dependent sections which depend upon the elements used in the dataset definition of that particular dataset inside the model definition.

The result for the overall analysis contains the optimized parameters, optimization history and the parameter history for the whole analysis, as well as optimization metrics like the degrees of freedom and root mean square error.

The per dataset results can be accessed by the same names that the datasets were referred to in the model definition. The general part consists of information which is independent of the elements used in the model definition such as the original data, the fitted data, and the residual. In addition, each model element used in the model definition for a dataset also adds information specific to the type of the element. The decay component of type kinetic for example adds information about the species concentration and species associated spectra (SAS). The names of the saved information are in general only dependent on the type of an element and not on the freely chosen name, an exception to the rule are elements whose type can occur multiple times per dataset in which case the name is suffixed with the name of the element in the model definition (e.g., a_matrix_decay for the 4TT example in Figure S 14).

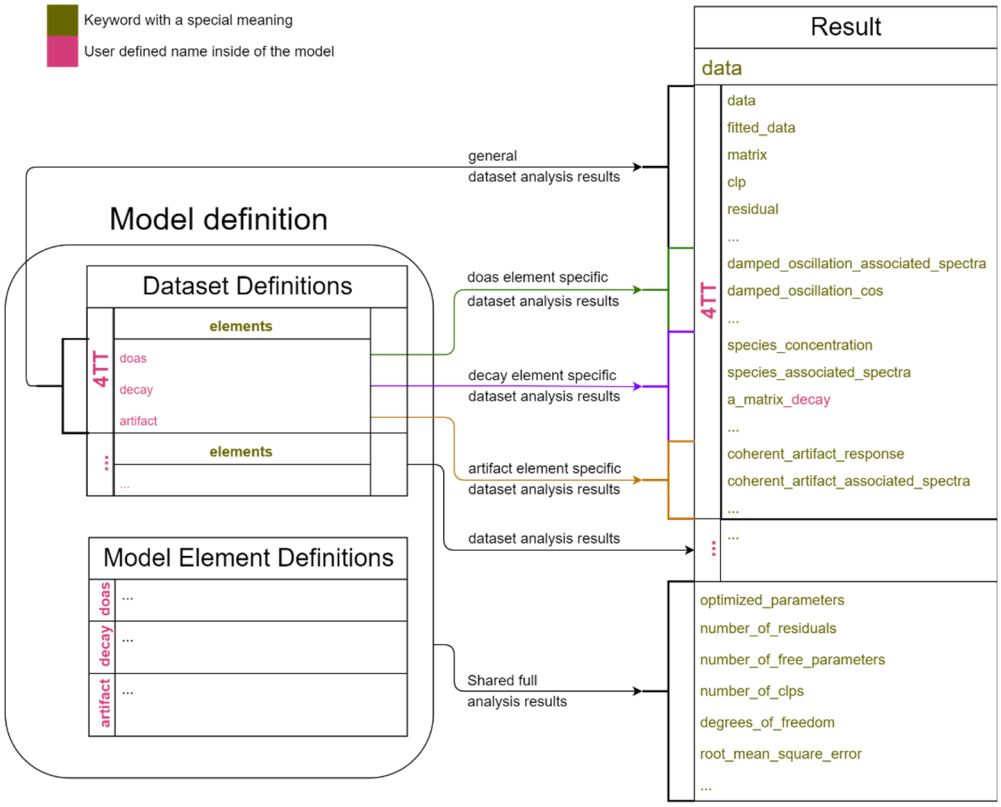


Figure S 14. Schematic of the general structure of a result object and its relation to the model definition. The information saved in the result can be categorized into overall results for the whole analysis and per dataset results. The per dataset results use the same name as the dataset in the model definition and can be categorized into general information independent of model elements and element specific information. The per dataset results include some redundant information, which could be calculated from other saved values (e.g., $fitted\_data = data - residual$). This is done for the convenience of the user, who can use the redundant information to plot the results in external plotting software without the need to compute those values and can be configured using *SavingOptions*.

Plugin system

Pyglotaran uses a plugin system for model elements and for reading and writing files. While the builtin plugins already provide the functionality needed by most users, the plugin system provides additional modularity, flexibility, extendibility, and the possibility for seamless integration with other software. The plugins can be created by third-party developers or by the users themselves and can provide new features and capabilities that are not included in pyglotaran itself. The file io plugins are split into two categories the ProjectIo plugins for the model, parameter definitions, and results and the DataIo plugins to read and write data. Pyglotaran already allows for many different input and output data formats (e.g. plain ASCII and NetCDF). To illustrate the use case, creating a DataIo plugin allows one to read and/or write new file formats without the need for an additional conversion step. Whereas creating a result saving plugin allows for integration with other software. By registering a file io plugin with the extension of a file format, pyglotaran is able to automatically determine which plugin to use when reading or writing a given file (on https://pyglotaran.readthedocs.io/, search for: *write own plugin*). To ensure that the right plugin is used when multiple plugins are registered under the same name the plugins are also registered with their fully qualified name (python import path) and the user can override which plugin to use (on https://pyglotaran.readthedocs.io/, search for: *using plugins*). Creating a model element plugin allows researchers to add new analysis capabilities as well as the ability to share the details of their research with the wider scientific community, allowing for reproducibility, easier collaboration, and improving scientific progress.

Software development

To ensure efficient development and high-quality code, several key practices have been implemented in the development of the pyglotaran ecosystem. Firstly, version control is managed through git and GitHub, using the GitHub flow model and branch protection to manage changes and ensure code quality. This allows multiple developers to collaborate on the codebase simultaneously while maintaining version history and control over changes.

All the development happens in the glotaran organization on GitHub, that besides the new python projects also contains the legacy projects TIMP[7], and Glotaran[6], which are still maintained but not further developed. The most notable components of the pyglotaran ecosystem are pyglotaran-extras [pygta-extras], pyglotaran-examples and pyglotaran-validation which together build the basis for the pyglotaran validation framework.

Code is structured using packages and modules, making it easier to organize and navigate through the codebase. Quality assurance is ensured using linters, formatters, and type checkers, which help to catch errors and enforce consistency in the code.

Continuous Integration and -Delivery (CI/CD) are utilized to automate the building and testing of the software, ensuring that the code is always in a working state. Automated documentation, both generated and manually curated, is also provided to facilitate understanding of the codebase and to help new users get up to speed quickly.

Automated dependency updates ensure that the software remains up to date with the latest libraries/frameworks and potential problems are discovered early. Deployment is handled through PyPI and conda-forge, making it easier for users to install and use the software.

Overall, these practices ensure that pyglotaran is high-quality, well-maintained software that is efficient to develop, test, and use.

Development infrastructure and tooling

Each project in the pyglotaran ecosystem as well as pyglotaran itself are developed using professional state of the art software development technologies and practices, as well as an extensive set of different tests that need to be passed by each change as well as code reviews from the maintainer before it can be added to the project. The passing of those tests is enforced by using branch protection with requires each change to pass all CI/CD tests (see below) including the QA-tools (see below) and have at least one approving review.

Development workflow using GitHub

GitHub is a web-based platform that provides version control using git and collaboration features for software development projects. It allows developers to store their code repositories in the cloud, provides version control features to track changes to code over time, and includes collaboration features like managing issues, pull requests (PR), and code reviews. Additionally, it offers Continuous Integration and Continuous Delivery to automate building, testing, and deploying of code changes.

Development follows the GitHub branching model where each developer only works on their fork of the main repository and submits PRs to the main repository to get a change incorporated. After the changes are reviewed and all tests are passed these changes get squash merged into the main branch of the repository resulting in a clean linear history, where details of the changes and discussions can be looked up in the PR itself. The details on how to contribute to a project can be looked up in the contribution section of each project's documentation.

Quality assurance tools

Quality assurance tools (QA-tools) are an essential part of modern software development, and they play a critical role in improving the quality and maintainability of the code. In the Python development world, several quality assurance tools are widely used, such as linters, formatters, and type checkers.

Linters are tools that analyze code for potential errors, bugs, and style issues. They can identify common mistakes like syntax errors, undefined variables, or unused imports. Linters can enforce coding conventions and best practices, ensuring that code is consistent and easy to read. This type of tool can save developers significant amounts of time by catching issues early in the development cycle.

Formatters, on the other hand, are tools that help developers to enforce a consistent code style. They can help ensure that the code adheres to indentation, line length, and other formatting rules. In addition to enforcing consistency, formatters can also automatically fix common formatting issues, such as incorrect whitespace or inconsistent line breaks. The result is code that is easier to read and maintain.

Type checkers are another critical quality assurance tool in Python. They help ensure that code adheres to the specified types of variables, arguments, and return values. By catching type-related errors before runtime, type checkers can help developers avoid bugs and improve code quality. Type checkers can also improve code understanding by helping developers understand the flow of data and logic in their code.

By using quality assurance tools like linters, formatters, and type checkers, Python developers can ensure that their code is consistent, error-free, and adheres to best practices and conventions. This, in turn, can help improve the reliability, maintainability, and readability of code, and reduce the time and effort required for testing and debugging. Ultimately, the use of quality assurance tools helps to improve the overall quality of software development, making it more efficient and effective.

The pyglotaran development uses the pre-commit framework to manage the quality assurance tools and their versions. This ensures that each committed code change is up to standards by intercepting the commit if any of the checks fails, leading to a cleaner commit history and an easier review process.

Continuous-Integration and Continuous-Delivery

Continuous-Integration and -Delivery (CI/CD) is a set of software development practices that aim to improve the speed and quality of software delivery by automating the process of building, testing, and deploying software.

Continuous Integration (CI) is the process of regularly integrating code changes into a shared code repository, typically using tools like Git, and ensuring that the codebase is always in a working state. This involves running automated tests and checks on the codebase to catch errors and issues early on.

Continuous Delivery (CD) is a practice where developers automate the process of deploying code changes to production. With Continuous Delivery, every code change is automatically built, tested, and prepared for deployment. Once the code changes have passed all tests and quality checks, they are automatically deployed to a staging or testing environment. The code changes can then be manually promoted to the production environment later.

In the context of software builds and distributions, CI/CD involves setting up automated pipelines that build, test, and package software changes, and then automatically deploy them to target environments. This ensures that software changes are thoroughly tested and validated before they are released to end-users and helps to reduce the time and effort required to deliver software updates.

Software Documentation

The documentation of the projects consists of handwritten guides and API documentation generated from the source code using Sphinx. To autogenerate documentation for the source code Sphinx uses the *docstring* added by the developers which are also shown in editors to guide users on how to use each part of the software and explain what the configuration options and intended usage are. The documentation for the current development version and each release is hosted on Read the Docs, allowing online access to the documentation websites as well as downloads as PDF or epub.

Dependency updates

In contrast to classical compiled desktop applications that come with a fixed set of dependencies, python packages typically only define a minimal version requirement for dependencies to allow interoperability of multiple packages in an environment. On the one hand this allows users to use different packages without their requirements conflicting and breaking functionality. On the other hand, this makes it nondeterministic for developers which specific versions of dependencies the users have installed, and new releases of a dependency can possibly break functionality. Therefore, the development is done with the dependencies pinned to specific versions creating a deterministic and reproducible reference environment. It also allows to automate dependency updates using the update service *dependabot* which is integrated into github. Dependabot will create a pull request for each new release of a dependency which triggers the automated tests and ensures that the software works properly with the new version of the dependency or allows developers to disallow this specific version when installing the software and file a bug report/make create a fix for that dependency.

Release and Deployment

Releases are done using GitHub releases on the corresponding project which allow to have a rich description of the release itself and its changes, as well as automatically creating a git tag for the release which allows to download the source code of the project for that particular release. Creating the git tag then triggers the CI/CD workflow, runs all tests again and in the case of a package builds source and binary distributions of the package that are uploaded to Python Package Index (PyPI). The release on PyPI then gets picked up by the conda-forge bot which will create a PR to the corresponding conda-forge feedstock repository, which contains the build recipe to create a release on conda-forge. This allows for additional manual adjustments of the build recipe (e.g., changed dependencies) and ensures that the conda build is also tested before it is released.

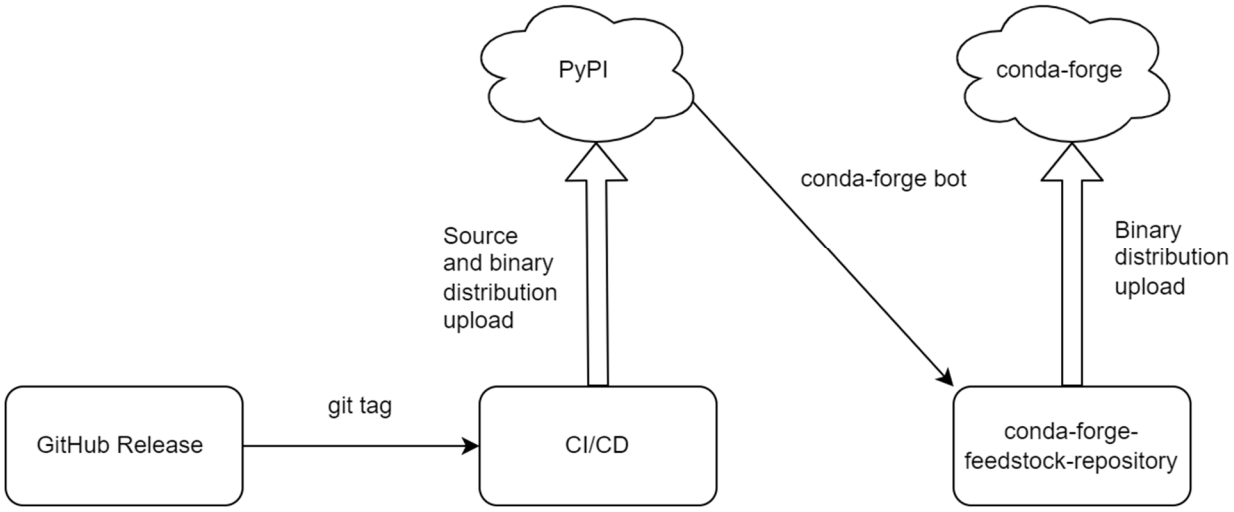


Figure S 15. Schematic of the automated release process of python packages. When creating a release on github a git tag is created which triggers the CI/CD workflow to run test and upload source and binary distributions to PyPI. The release on PyPI is then used to create a pull request to the corresponding feedstock repository updating the conda build recipe, test the created build and create a release to conda-forge.

Deprecation warnings

To allow improvements of pyglotaran as well as its DSL without breaking functionality for users pyglotaran uses deprecation warnings. Those warnings provide users with guidance on what changes they need to make to their code and/or model definition to ensure compatibility with the latest version. Users can take proactive steps to update their code, avoiding costly and time-consuming issues that may arise from using deprecated features. The clear and concise explanations of the changes required to ensure compatibility with future versions also allow for a gradual and incremental upgrade path of existing case studies done with older versions of pyglotaran by doing incremental

updates. The deprecation framework of pyglotaran was built for maximum development flexibility and its capabilities range from the renaming of functions, methods, and classes to moving source files to different locations with minimal development effort (see the documentation https://pyglotaran.readthedocs.io/en/latest/contributing.html#deprecations). To ensure that the provided deprecation warning is correct the existence of the new and old usage is verified in self-tests of the deprecation function. When a new release is prepared the deprecation functions raise errors if a deprecation is overdue, which ensures that the deprecated functionality is removed, and the source code is kept clean. A list of new deprecations and removal of deprecated functionality for each version can be found on the release-page (https://github.com/glotaran/pyglotaran/releases) for that version as well as the changelog in the documentation (https://pyglotaran.readthedocs.io/en/latest/changelog.html).

Ecosystem

While pyglotaran functions as the engine allowing to fit complex case studies in a way which wasn't possible before, its ecosystem provides convenience functions for plotting, documentation, and validation of the analysis result.

Pyglotaran-Extras

The pyglotaran-extras [**pygta-extras**] are a collection of very high-level helper functions for plotting and inspecting results from a parameter estimation using pyglotaran with a strong focus on time-resolved spectroscopy. One main purpose is to provide users with a quick and easy way to get feedback on the quality of their analysis without the need of detailed knowledge about the structure of the generated result dataset(s) and the usage of the xarray API [8]. The other purpose is to provide cross version compatibility to inspect result datasets. The plotting functionality focusses on providing a starting point for publication ready figures while allowing for the full customization and flexibility provided by matplotlib [9].

Pyglotaran-Examples

The pyglotaran-examples were historically grown as a development tool to validate results from pyglotaran manually against published and simulated results of its pre-decessors, which was mostly done by comparing plots. It still serves the purpose of validation using pyglotaran-validation and the comparison-results branch which is validated manually each time it is updated (e.g., to accommodate improvements or fix bugs). But now it also serves as usage documentation, demonstrating different features of the pyglotaran model language depending on which system is analyzed.

Validation

In the 0.5.0 release of pyglotaran the manual validation of results and testing of their consistency was extended by automatic tests which were later moved into its own repository pyglotaran-validation. This change allows for a better extendability in the future to cross validate pyglotaran against a wider range of other of software operating in the same domain. As well as decoupling the validation of generated results from the main project reducing the barrier for contributions. The validation itself does not use pyglotaran but operates on the data directly to prevent overlooking or even misinterpreting issues with the data it is supposed to validate due to circular logic.

Validation framework

The pyglotaran projects use GitHub Actions as their CI-CD platform. The automation pipelines are called workflows and are defined as plaintext files in yaml syntax inside the special folder *.github/workflows* inside of the repositories allowing to put them under version control and easily review changes to the infrastructure. Workflows consist of a definition of events on which the workflows should be executed and a definition of jobs that should be run. The jobs themselves define a list of steps to setup reproducible testing/build environments and run the necessary commands. Steps can either run commands in the shell or use predefined *Actions* to simplify more complex tasks and improve the readability of the workflow. Using the so-called *matrix strategy* (Figure S 17) allows to parametrize jobs and reduce duplications to for example run tests across different operating systems and python versions. For a

more dynamic creation of the matrix a custom step that generates the matrix can be used (Figure S 18). Having reproducible environments in which tests and builds are executed is crucial to guarantee that the software behavior is also reproducible and consistent.

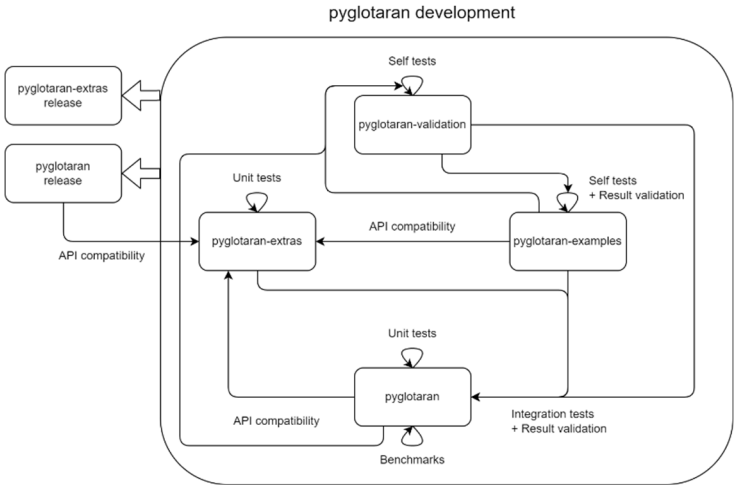


Figure S 16 Schematic overview how the different projects in the pyglotaran ecosystem interact with another using the CI. For example, if a change is to be made to pyglotaran the unit tests and benchmarks contained inside the pyglotaran repository as well as integration tests and result validation using pyglotaran-extras, pyglotaran-examples and pyglotaran-validation are triggered.

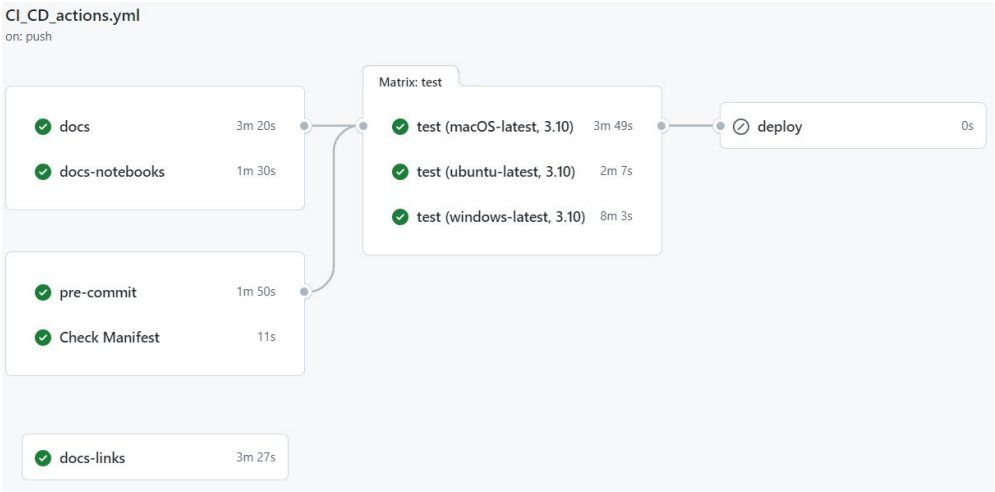


Figure S 17 Schematic illustration of the pyglotaran CI-CD workflow. First the QA-tools and manifest-check are run to ensure that the code is up to standards and all files that should be part of a release are in fact included, as well as tests that the documentation builds properly and does not contain dead links. Only after those tests have passed the unit tests are run for all supported python versions and operating systems. In case of a release all tests but the doc-links tests need to pass before a release on PyPI is created.

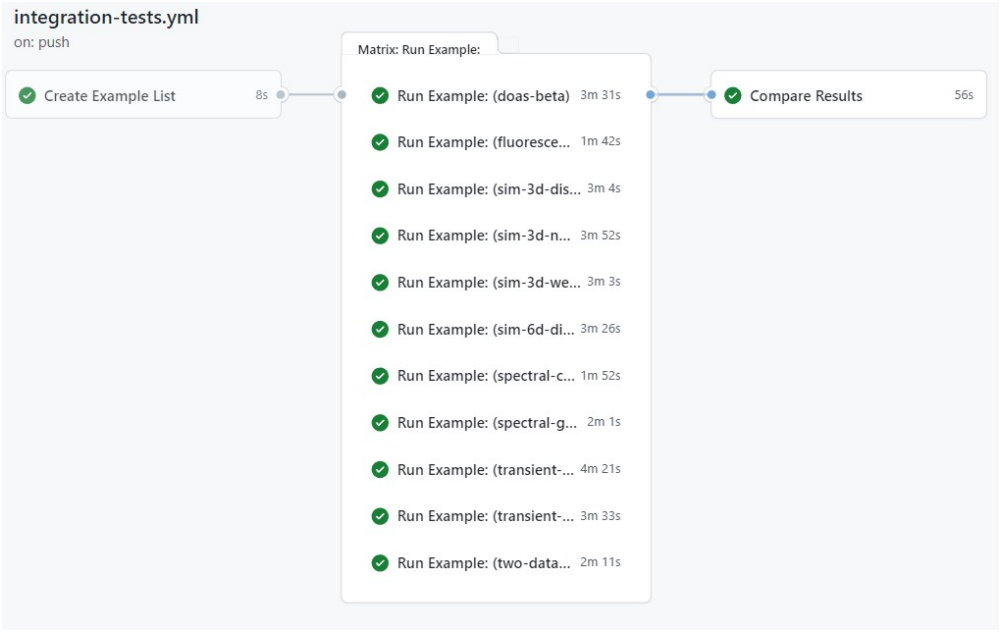


Figure S 18 Schematic illustration of the pyglotaran integration test workflow using github actions defined in the pyglotaran-examples and pyglotaran-validation repositories. The github action of the examples first generates a list of all available examples and then runs them in parallel and uploads the results to an artifact cache, those results are then downloaded and validated against the established and manually validated standards.

References

1. D. C. Teles-Ferreira, I. H. M. van Stokkum, I. Conti, L. Ganzer, C. Manzoni, M. Garavelli, . . . A. M. de Paula, Coherent vibrational modes promote the ultrafast internal conversion and intersystem crossing in thiobases, *Physical Chemistry Chemical Physics*, 2022, **24**, 21750-21758.
2. C. Hippius, PhD Thesis, Universität Würzburg, Fakultät für Chemie und Pharmazie, 2007.
3. I. H. M. van Stokkum, C. Wohlmuth, F. Würthner and R. M. Williams, Energy transfer in supramolecular calix[4]arene—Perylene bisimide dye light harvesting building blocks: Resolving loss processes with simultaneous target analysis, *Journal of Photochemistry and Photobiology*, 2022, **12**, 100154.
4. A. A. Arteni, G. Ajlani and E. J. Boekema, Structural organisation of phycobilisomes from Synechocystis sp strain PCC6803 and their interaction with the membrane, *Biochim. Biophys. Acta*, 2009, **1787**, 272-279.
5. A. M. Acuña, P. Van Alphen, R. Van Grondelle and I. H. M. Van Stokkum, The phycobilisome terminal emitter transfers its energy with a rate of (20 ps)–1 to photosystem II, *Photosynthetica*, 2018, **56**, 265-274.
6. J. J. Snellenburg, S. P. Laptenok, R. Seger, K. M. Mullen and I. H. M. van Stokkum, Glotaran: a Java-based Graphical User Interface for the R-package TIMP, *Journal of Statistical Software*, 2012, **49**, 1-22.
7. K. M. Mullen and I. H. M. van Stokkum, TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements, *Journal of Statistical Software*, 2007, **18**, 1 - 46.
8. S. Hoyer and J. Hamman, xarray: N-D labeled Arrays and Datasets in Python, *Journal of Open Research Software*, 2017, DOI: 10.5334/jors.148.
9. J. D. Hunter, Matplotlib: A 2D graphics environment, *Computing in Science and Engineering*, 2007, **9**, 90--95.