pgftransparentblack
transparentpgftransparent
natural
tikz@axis@topgraytikz@axis@middlegray!50!whitetikz@axis@bottomwhite
natural
tikz@ballblue
natural
tikz@radial@innergraytikz@radial@outerwhite

Dissertation

Francesco Tamburi

October 2023

Contents

Chapter 1

Intro

1.1 Positron Annihilation Lifetime Spectroscopy

Positron annihilation as a means of investigating lattice imperfections traces its roots back to the late 1960s, when the link between positron lifetimes and lattice vacancies was first proposed. [**?**] As the lifetime of a positron is a function of the electron density at the annihilation site, and lattice defects (e.g. open vacancies) represent a local decrease in electron density, longer lifetimes are observed for positrons trapped in these defects. [**?**] The presence of multiple positron lifetimes within a sample material, thus, indicates the presence of lattice defects within that material.

Positron Annihilation Lifetime Spectroscopy (PALS) is the study of these lifetimes, with the aim of determining the number and types of defects present in a sample material. While other techniques to study lattice defects exist, the main advantage of PALS is the high sensitivity of the technique to open-volume defects.[1] This property, along with the non-destructive nature of the technique, has led to its application in various fields of study.[**?**] [**?**]

1.1.1 The Positron Lifetime

Weakly radioactive isotopes, such as ^{22}Na, are commonly used to generate positrons. As part of its β^+ decay, ^{22}Na simultaneously produces a positron and a 1.27 MeV γ photon. Upon encountering the sample material, positrons quickly lose their kinetic energy in a process known as "thermalization".

The thermalization process isn't uniform, occurring via different mechanisms depending on the sample material and energy of the positron. At high energies, for example, atomic ionization is a dominant process, while at low energies (< 1 eV in metals and ~ 1 eV in semiconductors), phonon scattering is the most important process. Regardless of specific mechanism, however, thermalization occurs within the first few picoseconds of a positron lifetime.

After thermalization, the positron is free to diffuse through the sample material. During the diffusion process, the wavefunction of the positively charged positron spreads out into a highly delocalized Bloch state, primarily concentrated in the interstitial space between atomic nuclei in the material. It's in this phase that the positron spends the bulk of its lifetime and where defect trapping predominantly occurs.[2] How far a positron diffuses determines how many atoms are

[1] Thorough comparisons of defect sensitive techniques, as well as their respective advantages and disadvantages, are available in literature, but detailed analysis of these is beyond the scope of this project.

[2] Some trapping can occur during the thermalization process. The effect of this on positron lifetimes, however, has been proven to be negligible.

probed for positron traps, and thus how sensitive the PALS process is to defects.

Trapping occurs when a defect is encountered and a localized positron state forms at the defect. The rate at which this occurs depends on trapping rate κ, and is proportional to the defect concentration C. Specifically

$$\kappa = \mu C, \tag{1.1}$$

where μ is termed the positron trapping coefficient, and is constant for a given defect.

Annihilation can occur in the delocalized state or the localized trapped state, producing two 511 keV γ-photons. In both cases, lifetimes are usually on the order of hundreds of picoseconds.

1.1.2 Positronium

Positrons can also enter a third state, however, where, instead of delocalizing into the bulk or being trapped in a defect, the positron enters a bound state with an electron, forming an exotic atom known as positronium (Ps). Annihilation lifetimes for positronium depend on the spin alignment between the two particles and are around 140ps for the less common, parallel spin, ortho-Ps, or 1-5ns for the more common, antiparallel spin, para-Ps.

1.1.3 Measuring Lifetimes

Positron lifetimes can be determined experimentally by measuring the time interval between the emission of gamma photons that signal the production and annihilation of the positron. The experimental setup for doing so begins with a coupled scintillator and photomultiplier, which converts the γ-rays into analog electrical pulses. These are then processed by discriminators, distinguishing the 1.27 MeV (in the case of ^{22}Na) from the 511keV pulses. The former is used as a "start" signal to begin charging a capacitor, and the latter stops the charging process. The capacitor acts as a sort of "electronic stopwatch", converting the time delay into an amplitude signal. To ensure a linear time-amplitude relationship, the stop signal is delayed by a fixed amount. Each signal is stored in the memory of a multichannel analyzer, where the channel numbers represent time and every count represents a single annihilation event. The resulting lifetime spectrum contains more than 10^6 annihilation events.

1.1.4 The Spectrum

A lifetime spectrum contains $k+1$ lifetime components, corresponding to k different defect types and the positron lifetime in the defect-free bulk (τ_b). From these we get the time-dependent positron decay spectrum $D(t)$ – the probability for a positron to still be alive at a given time t [**?**] – given by the expression

$$D(t) = \sum_{i=1}^{k+1} I_i \exp\left(-\frac{t}{\tau_i}\right), \tag{1.2}$$

where τ_i and I_i represent the lifetime and intensity of each component, respectively.[3] The resulting positron lifetime spectrum $N(t)$ is given by the absolute value of the time derivative of the positron decay spectrum $D(t)$[4]:

$$N(t) = \left|\frac{\mathrm{d}D(t)}{\mathrm{d}t}\right| = \sum_{i=1}^{k+1} \frac{I_i}{\tau_i} \exp\left(-\frac{t+t_0}{\tau_i}\right). \tag{1.3}$$

[3] When $k = 0$ (i.e. there are no defect components), $\tau_i = \tau_b$ (the bulk lifetime) and $I_i = 100\%$

[4] Due to the delay introduced in the time-amplitude conversion, $t = t + t_0$

The experimental spectrum is the convolution of the lifetime spectrum analytically described above and an Instrument timing Resolution Function (IRF). This resolution function is mainly determined by the scintillator+photomultiplier and takes the form of a sum of several Gaussian functions,

$$F(t) = \sum G_i(t), \tag{1.4}$$

with variations in the peak position, relative height and width of each Gaussian, $G_i(t)$. When the individual Gaussians are close enough to each other, $F(t)$ can be approximated by a single Gaussian.

The experimental spectrum thus takes the form

$$D_{exp}(t) = \int_{-\infty}^{+\infty} D(t - t')F(t')dt'. \tag{1.5}$$

In addition, experimental spectra also contain a level of background noise and contributions from positron annihilation within the source.

1.2 The Standard Trapping Model

The goal of PALS is to extract meaningful information relating to defects in a sample material. The Standard Trapping Model (STM) links positron lifetimes to the number of defect types and their concentration. In the STM, the spectrum is modeled using a series of differential equations, based on annihilation and trapping rates, called rate equations. Two key assumptions of the model are that

(i) Positron trapping during thermalization can be safely ignored,

(ii) Defects within the sample are homogeneously distributed.

1.2.1 Single Defect Trapping

The simplest trapping model assumes a single, open volume defect type, such as an atomic vacancy. After thermalization, positrons either annihilate in the defect free bulk at a rate $\lambda_b = 1/\tau_b$ or are trapped in an open-volume defect, with a trapping rate $\kappa_d \propto$ the defect concentration C. As the electron density within the defect is lower than that of the bulk, the defect annihilation rate $\lambda_d = 1/\tau_d$ must be smaller than λ_b.

This information is used to write the rate equations:

$$\frac{\mathrm{d}n_b(t)}{\mathrm{d}t} = -(\lambda_b + \kappa_d)n_b(t), \tag{1.6}$$

$$\frac{\mathrm{d}n_d(t)}{\mathrm{d}t} = -\kappa_d n_d(t) + \kappa_d n_b(t), \tag{1.7}$$

which describes the change in the number of positrons in the bulk (n_b) and the defect (n_d), repectively, at time t. As we have assumed no trapping during thermalization, if we define N_0 as the number of positrons at $t = 0$, we get our starting conditions, i.e. $n_b(0) = N_0$ and $n_d(0) = 0$. The solution to eq.s **??** and **??** gives us the decay spectrum of the positrons

$$D(t) = \frac{\lambda_b - \lambda_d}{\lambda_b - \lambda_d + \kappa_d} e^{-(\lambda_b + \kappa_d)t} + \frac{\kappa_d}{\lambda_b - \lambda_d + \kappa_d} e^{-\lambda_d t}. \tag{1.8}$$

Taking eq. **??**, and setting $k = 1$ (as we're dealing with a single defect type) we can make the appropriate subtitutions and get:

$$D(t) = I_1 \exp\left(-\frac{t}{\tau_1}\right) + I_2 \exp\left(-\frac{t}{\tau_2}\right), \tag{1.9}$$

where

$$\begin{aligned} \tau_1 &= \frac{1}{\lambda_b + \kappa_d} \quad , \quad \tau_2 = \frac{1}{\lambda_d}, \\ I_1 &= 1 - I_2 \quad , \quad I_2 = \frac{\kappa_d}{\lambda_b - \lambda_d + \kappa_d}. \end{aligned} \tag{1.10}$$

By taking the absolute value of the time derivative of the decay spectrum, in similar fashion to eq. **??**, we get the positron lifetime spectrum

$$N(t) = \left|\frac{\mathrm{d}D}{\mathrm{d}t}\right| = \frac{I_1}{\tau_1} \exp\left(-\frac{t}{\tau_1}\right) + \frac{I_2}{\tau_2} \exp\left(-\frac{t}{\tau_2}\right). \tag{1.11}$$

1.2.2 Observations

Looking closely at eq. **??**, we can see that there are two lifetime components in the spectrum. From eq. **??** we can observe that the trapping rate κ_d affects the first lifetime component and the relative intensity of the two components, but not the second lifetime component, which arises from positron annihilation within the defects.

This means that while τ_2, also called the defect-related lifetime $\tau_d = 1/\lambda_d$, is simply the positron lifetime within the defect, τ_1 is not the positron lifetime within the defect-free bulk $\tau_b = 1/\lambda_b$. As $\tau_1 \leq \tau_b$, τ_1 is also called the "reduced bulk" lifetime and as we increase the defect concentration (and thus increase the trapping rate κ_d) τ_1 and its associated intensity $I_1 \to 0$.

This limiting case, known as saturation trapping, happens when the defect concentration is high enough that the average spacing between defects is much smaller than the positron diffusion length. As a result, virtually all positrons are trapped, and the reduced bulk component disappears entirely from the spectrum.

1.2.3 Multiple defects

The single defect model can be easily extended in order to model more complex spectra, where more than one defect type is present. This is simply done by assuming no interaction between different defects and writing additional rate equations for each additional defect type. Thus, instead of having two rate equations, we have $k + 1$ equations, with $k =$ the number of defects, or

$$\frac{\mathrm{d}n_b}{\mathrm{d}t} = -\left(\lambda_b + \sum_{i=1}^{k} \kappa_i\right) n_b(t) \tag{1.12}$$

for the bulk and

$$\frac{\mathrm{d}n_{di}}{\mathrm{d}t} = \kappa_i n_b(t) - \kappa_i n_i(t) \tag{1.13}$$

for each defect type, with $i = 1, \ldots, k$.

1.3 Spectrum Decomposition and Project Aims

Decomposition of the lifetime spectrum is an example of an "inverse problem", where observations (the spectrum) are used to determine the causal factors that produced them (the components

that make up the spectrum). The complexity of these problems means there are limits on when and how well they can be solved.

Lifetime decomposition is commonly done by using some sort of least-squares method to fit a model function to an experimental spectrum. One disadvantage of this approach is that the number of lifetime components in the model function must be estimated before performing the fit. Working around this limitation often means, then, starting with a single component fit and increasing the number of components until the variance of the fit stops decreasing.

The aim of this project is to investigate the fitting process at its limits, observing where the process fails and how it fails.

Chapter 2

Method

This project consists of a series of trials aimed at evaluating different aspects of the fitting process. Each trial involves first simulating and fitting multiple spectra, then analysing the results of the fits. The former was accomplished using the PALSfit software package and the associated PALSsim program, while for the latter custom Python scripts were written to group the fits together and create relevant plots, using the Matplotlib library.

Each trial can be further grouped into one of three categories. The first investigates how changing the lifetimes and intensities of a two lifetime spectrum affects the fit. The second looks at experimental parameters, in particular what can be changed about PALS experiments to improve fitting performance. The third focuses on modeling spectra that closer reflect real experimental spectra.

In this chapter is a description and overview of the tools used to generate, fit and analyse the spectra that make up each trial.

2.1 The PALSfit Bundle

PALSfit is a Windows based software package, developed for the analysis of positron lifetime spectra, based on a non-linear least squares approach. Included are two modules, POSITRONFIT and RESOLUTIONFIT, which are used to deconvolve the lifetime components and resolution functions (respectively) from the lifetime spectra. As the scope of the project limited to lifetime component analysis, only the former will be used. PALSfit Version 3, or more simply PALSfit3, is the latest version of this package, which this project is aimed at evaluating. All subsequent references to "PALSfit" are to this specific version of the software.

Included with PALSfit is the PALSsim program, which generates lifetime spectra based on user input. PALSsim makes it possible to quickly generate spectra with full control over relevant parameters such as lifetimes, resolution function, background noise, etc. All spectra to be analysed in the project were produced using PALSsim.

2.1.1 PALSsim

As can be seen in Fig.**??**, PALSsim groups user input data into three sections: Spectrum parameters, Resolution function and Lifetime components.

In the *Lifetime components* section, the number of components in the spectrum and their associated values can be modified. Only the *Lifetimes* and *Intensities* fields will be used for each component throughout this project, as we'll assume each lifetime component to be discrete. Up

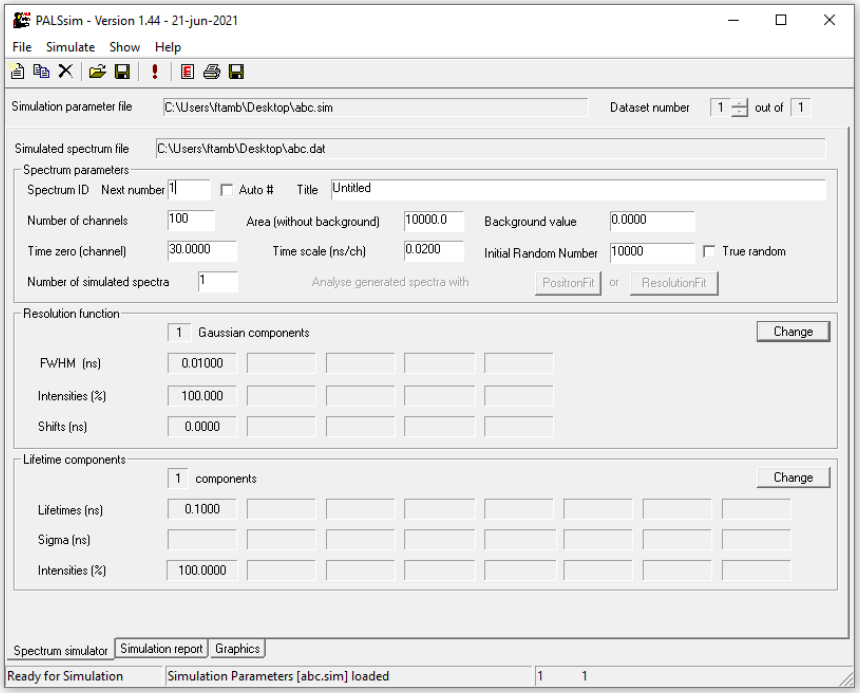


Figure 2.1: PALSsim window

to eight components can be simulated, but the maximum number of components used in the project will be three.

The *Resolution function* section deals with the instrument resolution function of the simulated spectrum. The resolution function is given by a sum of up to five Gaussian functions $G(t)$ with the width, relative height and peak position of each Gaussian component being controlled by the *FWHM*, *Intensities* and *Shifts fields* respectively.

For the majority of this project, a three gaussian resolution function will be used, with the appropriate parameters taken from an experimental spectrum and indicated in Table **??**. For a visual representation of the IRF, see Fig. **??**.

The variables found within the *Simulation parameters* section affect the spectrum as a whole. Here we find parameters controlling the number of channels, time per channel, number of counts, t_0 and background noise. Relevant parameters and default values used are as follows:

- Number of channels: 1000
- Area (without background): 5650000
- Background value: 8.5
- Time zero (channel): 1400
- Time scale (ps/ch): 5

All other parameters were left unchanged.

PALSsim generates five files every time a new spectrum is simulated. The file with the .dat extension contains the simulated spectrum, organized as a table of values for each channel. A .sim file stores the simulated values, allowing them to be reloaded into the program. A simulation report is generated every time a simulation is ran that contains data about the spectrum and saved in a .out file. Lastly two control files with the .pfc and .rfc extensions are generated to be read by PALSfit. Loading one or the other control file into PALSfit determines whether the POSITRONFIT or RESOLUTIONFIT module is used to analyse the spectrum (.pfc for the former and .rfc for the latter).

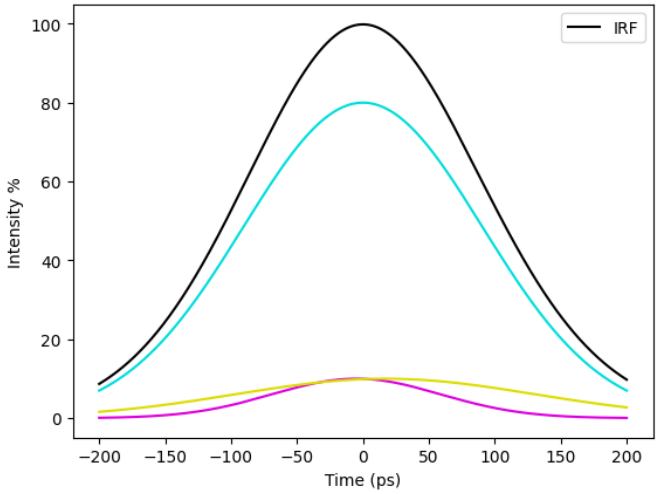


Table 2.1: IRF values

FWHM (ps)	Intensity (%)	Shift (ps)
213.2	80	0
150.2	10	-5
265.7	10	17

Figure 2.2: Visual representation of the IRF, generated using Python and Matplotlib

2.1.2 PALSfit

Opening a .pfc file with PALSfit should automatically load the spectrum and all relevant fitting values. The *Spectrum setup* window can be opened by clicking any of the two *Change* buttons in the *Spectrum* tab (selected by default when the program is opened). Here the *Default Ranges* button can be clicked to ensure that the appropriate range is fitted and the spectrum can be saved to the control file by checking the relevant checkbox.

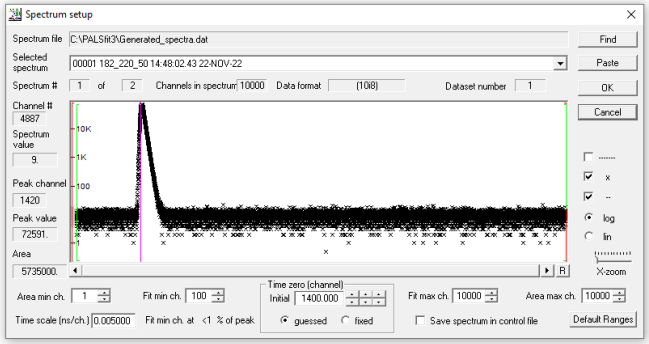


Figure 2.3: Spectrum setup window

Seven other tabs are present at the bottom of the program. The *Resolution function* tab shows a graphical representation of the resolution function used during fitting, and allows the user to

modify the Gaussian components that make up the IRF. In the *Lifetime and Corrections* tab, the number of lifetime components and the initial values used in fitting them can be modified. Meanwhile, the *Text output* tab displays the result of the last fit. All other tabs can be safely ignored, at least in the context of this project.

A fit is performed by pressing the *Analyse* button at the top of the window or pressing the F5 key. This generates a series of files in an output folder, contained in the same directory as the .pfc file. Of relevance are the .out file and .csv file that contain the results of the fit.

2.2 Python Scripts

Custom Python scripts were developed for the following purposes:

2.2.1 Grouping fit results

As each trial involves generating and fitting multiple spectra, the results of multiple fits had be collected into a single document for analysis. To this end, files were grouped into batches and subfolders containing spectra to be analysed together. Once generated and fitted, the fit results for each subfolder were contained in a single output folder. Two Python scripts were initially used to collect the data into a single file with all relevant information for plotting, that later were consolidated into one. The first script joined the multiple .csv files produced by PALSfit and the second added information on the original, simulated values of the lifetime components to the file.

2.2.2 Plotting fit data

All plots were produced using Python library Matplotlib. These can be grouped into a few different types, with an overview provided below as a reference to aid in interpretation.

An example of the first type, used for two-lifetime spectra, can be seen in Figure **??**. On the x-axis is always the simulated value of τ_2. On the y-axis is either τ_1, with the (fixed) simulated value of τ_1 indicated with a grey dashed line, or the difference between the fitted and simulated values of τ_2, with the grey line at $y = 0$. Each datapoint represents a different spectrum. For each simulated τ_2, there are three datapoints, indicating the relative intensities of the two lifetime components, marked in the legend. Error bars are given by the standard deviation calculated by PALSfit.

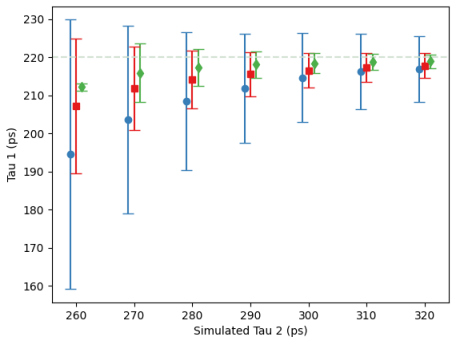


Figure 2.4: $\tau_1 = 220$ ps (fixed), $\tau_2 = 260$-320 ps

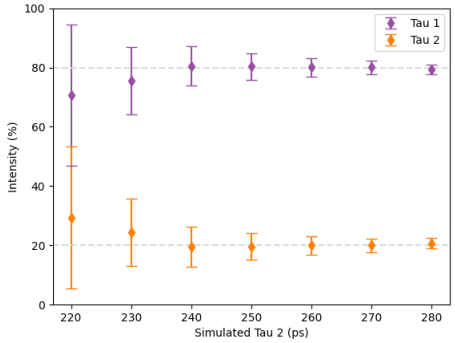


Figure 2.5: $I_1 : I_2 = 80\% : 20\%$, $\tau_2 = 260$-320 ps

Closely related is the second type, shown in Figure **??** and also used for two-lifetime fits. The simulated τ_2 is still on the x-axis, but lifetime intensity is on the y-axis, instead. The grey lines trace the simulated values for I_1 and I_2 and each spectrum is represented by two vertically stacked points, representing the fitted values of I_1 and I_2, as indicated in the legend.

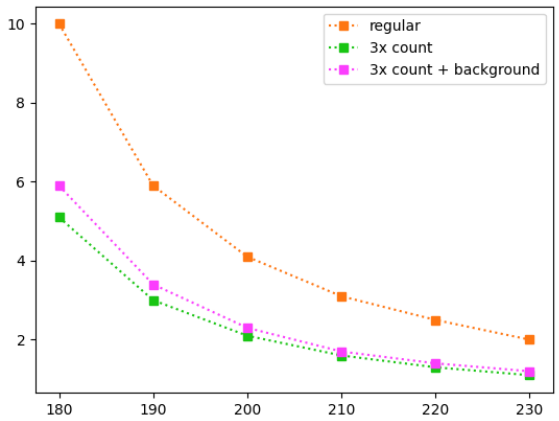


Figure 2.6: Std. dev. τ_1, $I_1 : I_2 = 50\% : 50\%$

In the third type a single fit result is being tracked on the y-axis, across the same intensity ratio, and with the x-axis representing (again) simulated τ_2. Figure **??** is an example where the variable being tracked is the standard deviation of τ_1, the intensity ratio is 50%-50%, and each colored line represents a different number of counts in the spectrum, indicated in the legend. Like the first type, each datapoint corresponds to an individual spectrum. Each spectrum is made up of two lifetime components, as in the previous types.

2.2.3 Batch simulation and fitting

A command-line version of PALSfit called PATFIT is available for use, found on the PALSfit website, as an auxiliary program. By using a Python library for interfacing with command-line programs, a script was developed that uses PATFIT to fit multiple spectra. This script then takes the .csv files produced by these fits and groups them into a single file. A separate script was also developed that allows for multiple .sim files to be generated, which can then be simulated using PALSsim. Using these two scripts still requires the user to simulate each .sim file individually, then link the spectrum file and adjust the fitting ranges on PALSfit. This can be tedious, however, and a single script to automate the entire process could be both useful and possible to code, though beyond the scope of this project.

Chapter 3

Trials and Results

3.1 Lifetime separation

The aim of the first trial was to investigate how the separation between lifetime components affected the fit. PALS spectra containing two lifetime components were generated, corresponding to a two defect material in saturation trapping. The first lifetime, τ_1, remained fixed at 180 ps while the second lifetime, τ_2, decreased from 280 ps to 220 ps, in 10 ps intervals.

For each pair of lifetimes, intensity was varied to simulate three diferent relative defect concentrations. The values chosen for the first lifetime intensity I_1 were 20%, 50% and 80%, with $I_2 = 100\% - I_1$.

Expectations are that:

(i) The overall fit improves as lifetime separation, $\tau_1 - \tau_2$, increases.

(ii) The fit improves for a given lifetime component as the intensity of that component increases.

In Figure **??** is the fit for the first lifetime component τ_1. Overall results seem to be mixed. As would be expected, the precision of the fit, indicated by the size of the error bars, visibly improves as τ_2 (and therefore the lifetime separation) increases. For any given value of τ_2 on the x-axis, a noticeable increase in both precision and accuracy can be observed as the intensity I_1 of the first component increases. This also in agreement with expectations.

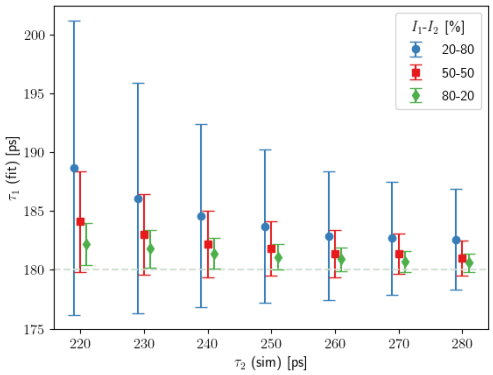


Figure 3.1: $\tau_1 = 180ps$, $\tau_2 = 220\text{-}280ps$

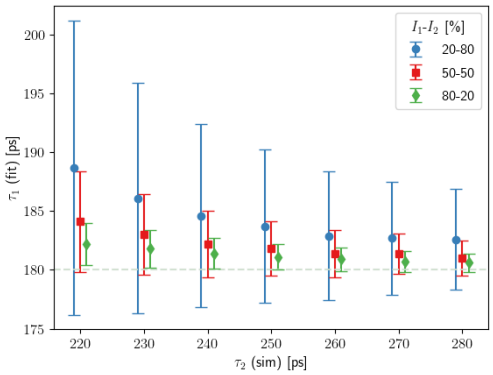


Figure 3.2: $\tau_1 = 180ps$, $\tau_2 = 250\text{-}270ps$

Contrary to expectations, however, is the trend in accuracy.

Using PALSSIM, a series of spectra containing two positron lifetimes, τ_1 and τ_2, were generated with a three gaussian instrument resolution function (see Table **??** and Figure **??**). τ_1 was kept fixed at 180ps and τ_2 varied from 220-280ps, in 10ps intervals. For each value of τ_2, three spectra were generated with the relative intensities of τ_1 and τ_2 set to 20%-80%, 50%-50% and 80%-20%. All the resulting spectra were then analyzed using PALSFIT, in order to evaluate how well the program could extract the two lifetimes, and their respective intensities, from each simulated spectrum.

In Figure **??**, we can observe how the values of τ_1 outputted by the program change, as the simulated value of τ_2 (on the horizontal axis) increases and the relative intensities (indicated by color and shape of marker) vary. Zooming in to the τ_2 = 250-270ps range, we can see in Figure **??** that, for these values in particular, PALSFIT struggles to determine τ_1 in the 50-50 and 80-20 case.

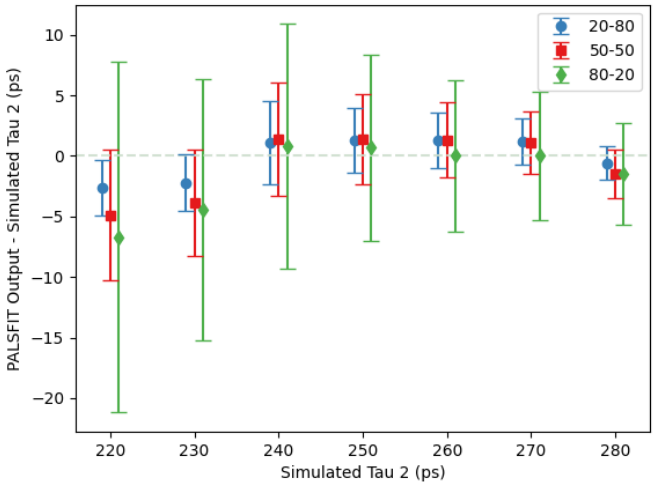
Unlike for τ_1, where the simulated value of the lifetime is fixed, the simulated value of τ_2 (our point of comparison) changes in between spectra. To make the data easier to visualize, rather than the fitted value of τ_2, the difference between the result and the original is plotted instead (see Figure **??**). In the figure we can see that, aside from the 20-80 spectrum for $\tau_2 = 220$, the software performs better than for τ_1.

The error bars represent the standard deviation of – and thus the confidence of the program in – the lifetimes. From them, we see two factors that affect the size of the bars. The first is the relative intensities of the two intensities. In fact, in Figure **??**, which plots τ_1, the error bars are the smallest for the 80-20 data, where the shorter lifetime is more intense, and in Figure **??**, tracking τ_2, the opposite is the case and we have the 20-80 data is most precise. Observing all the figures, we see the second factor: as the time interval between the two lifetimes increases, the size of the error bars decrease.

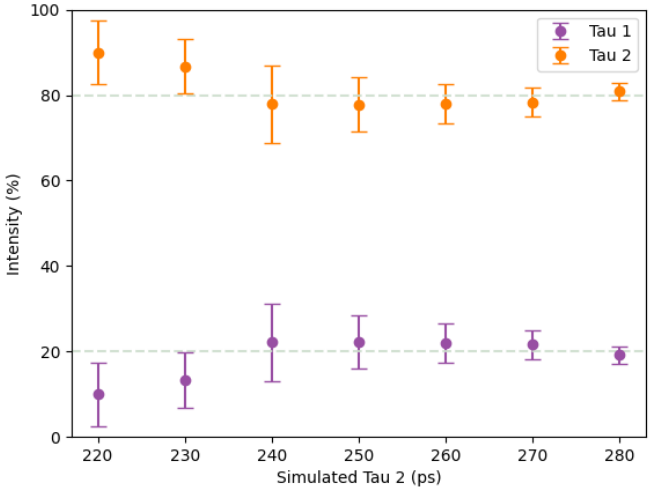
In Figures **??** - **??** the fitted intensities of the two lifetimes are plotted against simulated τ_2, for each combination of simulated intensities. In the figures, we see that when the relative intensity of τ_2 was greater or equal to τ_1, then PALSFIT was able to calculate all the appropriate lifetime intensities. However, when the intensity of τ_2 was set to 80%, shown in Figure **??**, the software was unable to output the correct intensities for the first two simulated values of τ_2. Looking at our error bars, we can see the same relationship between their size and the lifetime separation mentioned earlier.

On the whole, PALSFIT seems to have performed well, but not perfectly. As the first lifetime, τ_1, was kept constant at 180ps, the next step would be to change τ_1 and see how that affects our results.

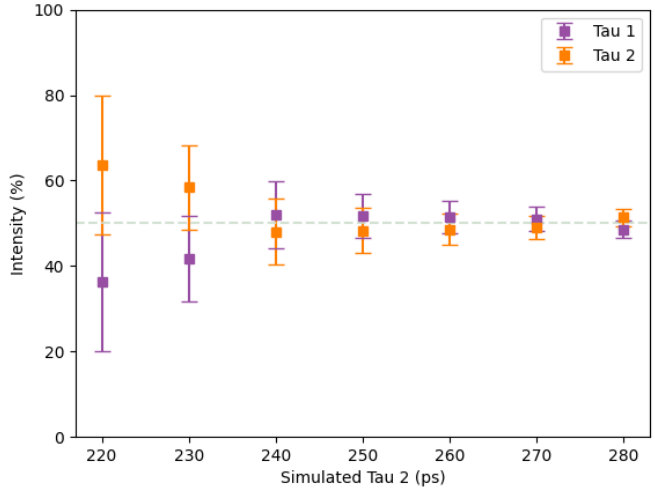
Figure 3.3



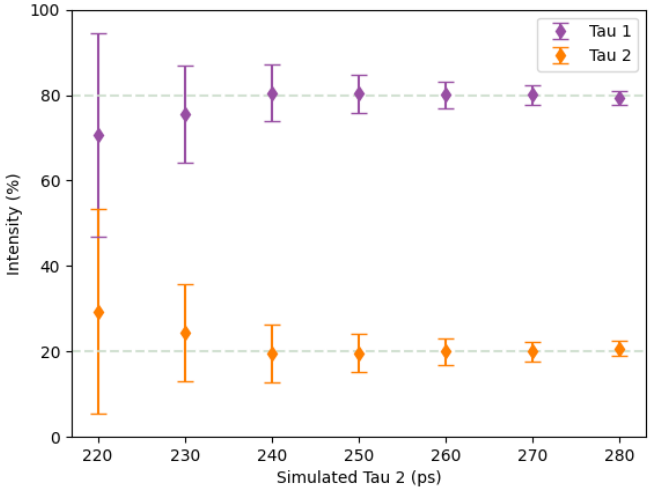
(a) τ_2 difference



(b) $\tau_1 = 20\%, \tau_1 = 80\%$



(c) $\tau_1 = 50\%, \tau_2 = 50\%$



(d) $\tau_1 = 80\%, \tau_2 = 20\%$

3.2 Modifying τ_1

A similar procedure was performed for τ_1=150 and τ_1=220. To keep the relative time interval the same in between batches, the corresponding spacing between τ_1 and the τ_2 range was kept consistent. For τ_1 = 150, this meant a τ_2 range of 190-250ps, and for τ_1 = 220, this meant a corresponding τ_2 range of 260-320ps.

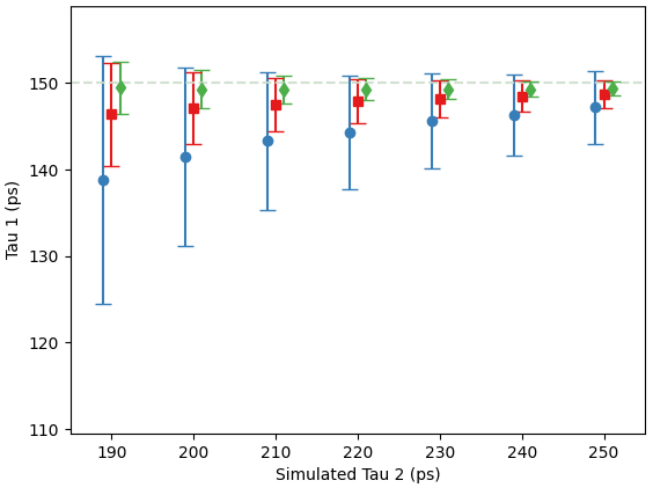
This was first done for τ_1 = 150. As can be seen in Figure **??**, the results for this batch are all within error, with both accuracy and precision getting better as the lifetime separation increases, in line with what would be expected.

The other batch, meanwhile, where τ_1 was set to 220ps, was not as successful. The results can be seen in Figure **??**, but in general, PALSFIT struggled with fitting the lowest values for τ_2 and the error bars are noticably larger. The program even gives nonsensical results when τ_1 = 260ps and the relative τ_1-τ_2 intensity is set to 80%-20%, as can be seen in Figures **??**, **??** and **??**.

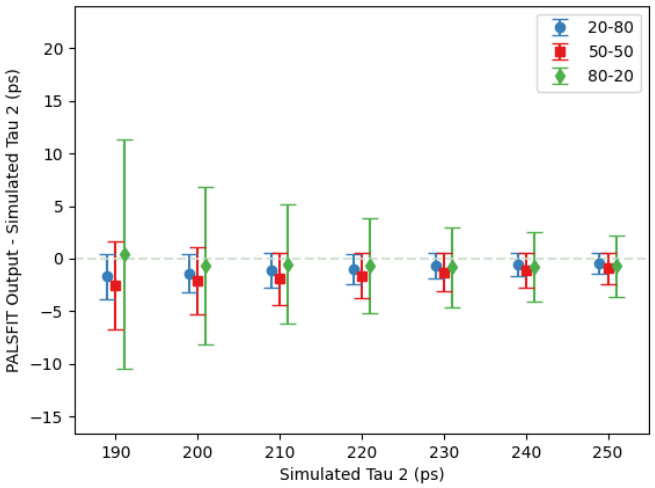
In order to compare all three datasets, the absolute difference between the fitted and simulated values of each relevant variable, τ_1, τ_2 and intensity was plotted. Additionally, plots for their respective standard deviations were generated. The resulting plots are represented in Figures **??**-**??**. Two general observations are apparent from analysis of these figures:

The first is that the τ_2 = 180ps data seems much more scattered than the other two datasets, which both follow a relatively clear increase in precision and accuracy as the lifetime separation increases. The second is that as τ_1 decreases, PALSFIT seems better able to resolve the two lifetimes, with both the difference from the true value and the size of the error bars being the smallest when τ_1 = 150ps.

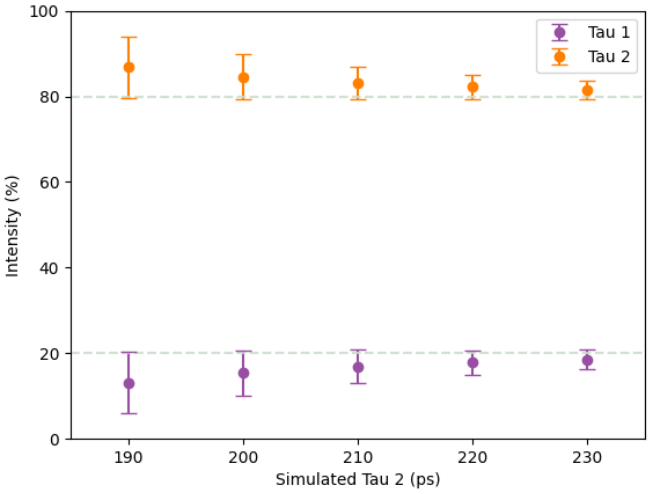
Figure 3.4



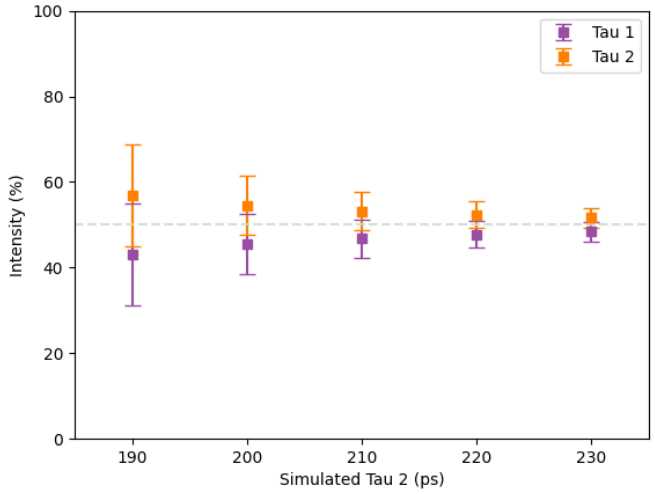
(a) fixed $\tau_1 = 150ps$



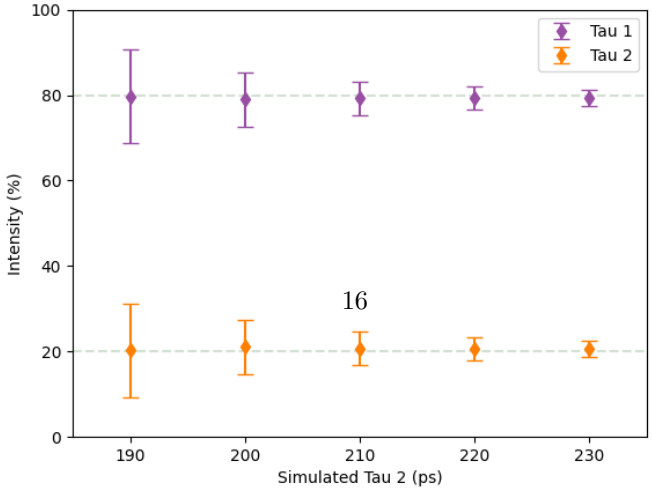
(b) τ_2 difference



(c) $\tau_1 = 20\%, \tau_2 = 80\%$

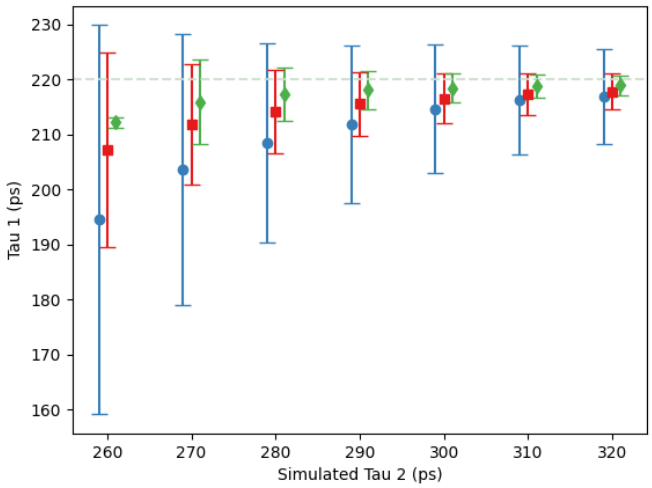


(d) $\tau_1 = 50\%, \tau_2 = 50\%$

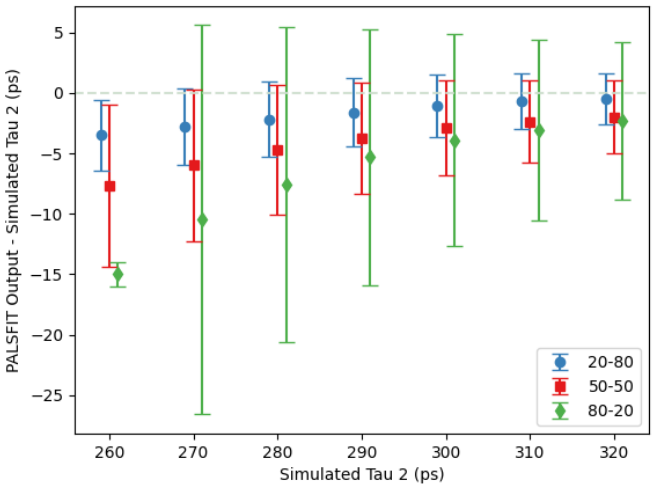


(e) $\tau_1 = 80\%, \tau_2 = 20\%$

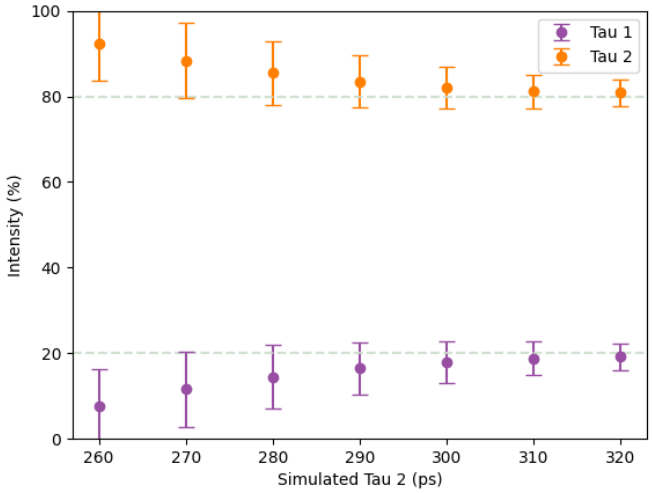
Figure 3.5



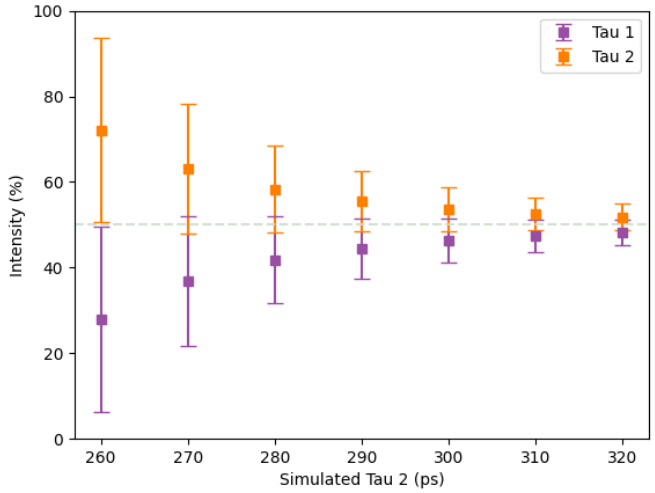
(a) fixed $\tau_1 = 220ps$



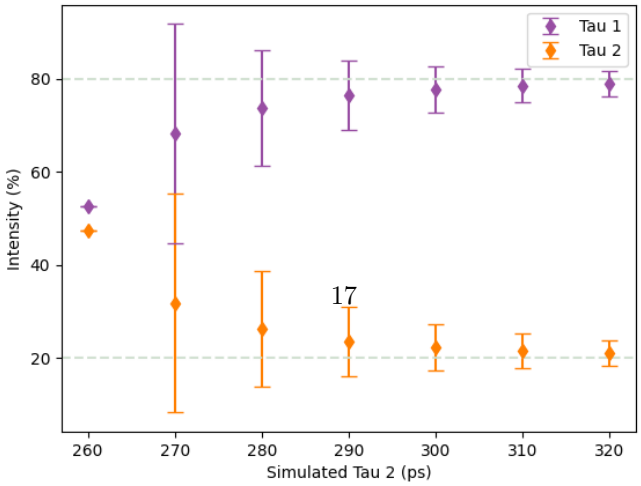
(b) τ_2 difference



(c) $\tau_1 = 20\%, \tau_2 = 80\%$

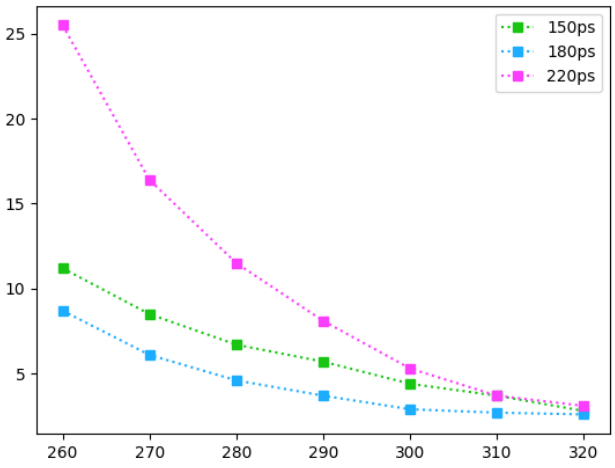


(d) $\tau_1 = 50\%, \tau_2 = 50\%$

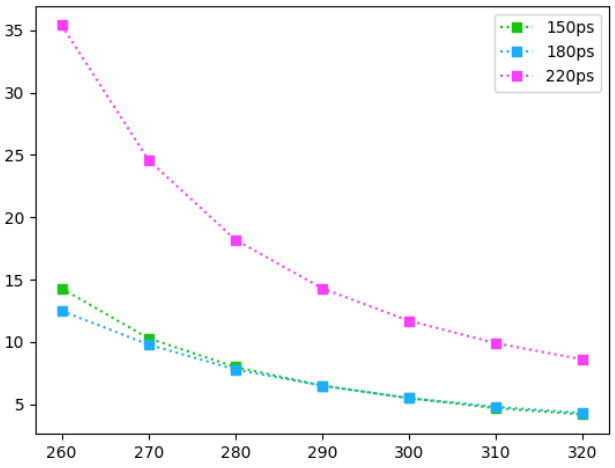


(e) $\tau_1 = 80\%, \tau_2 = 20\%$

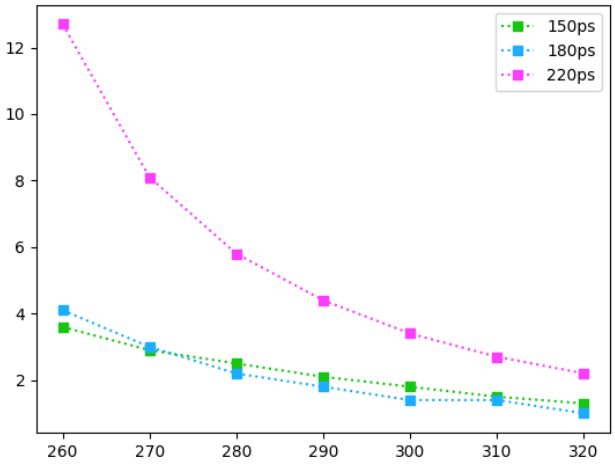
Figure 3.6: τ_1, rows = 20-80, 50-50, 80-20, columns = diff, std dev



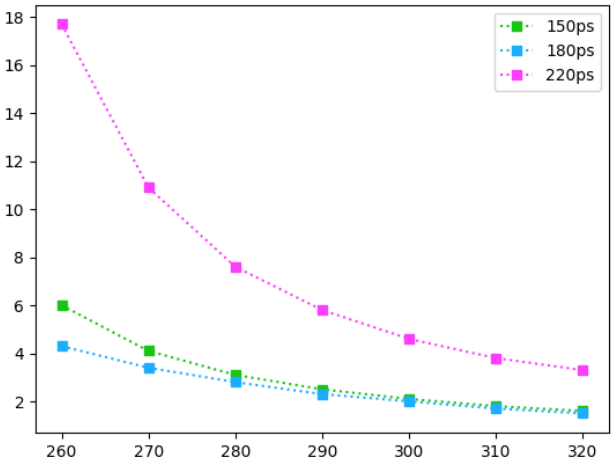
(a)



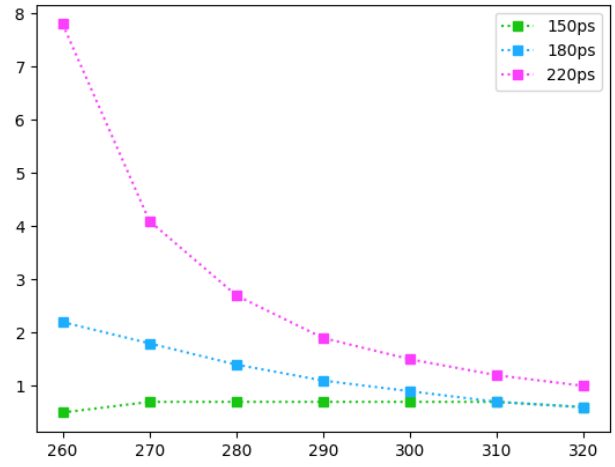
(b)



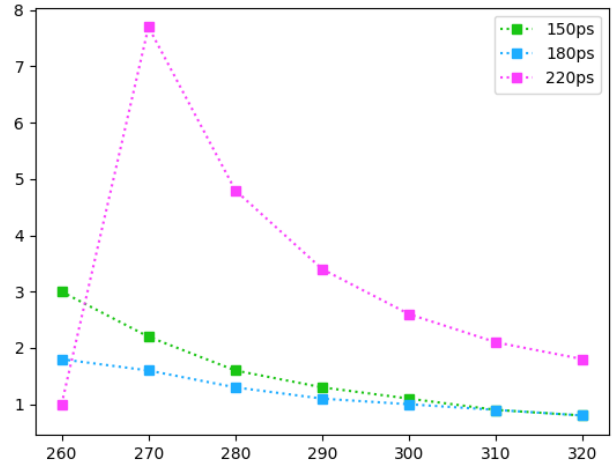
(c)



(d)

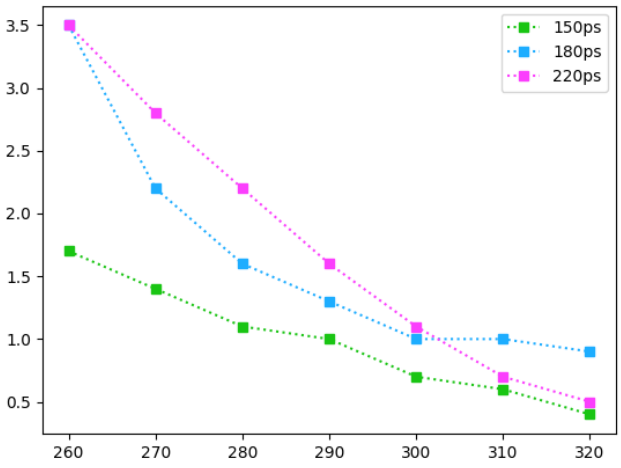


(e)

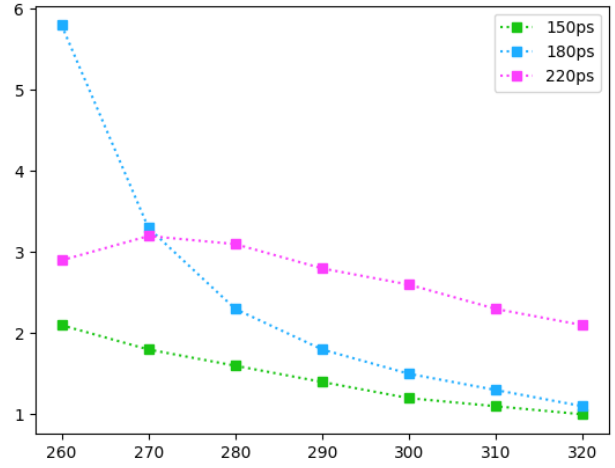


(f)

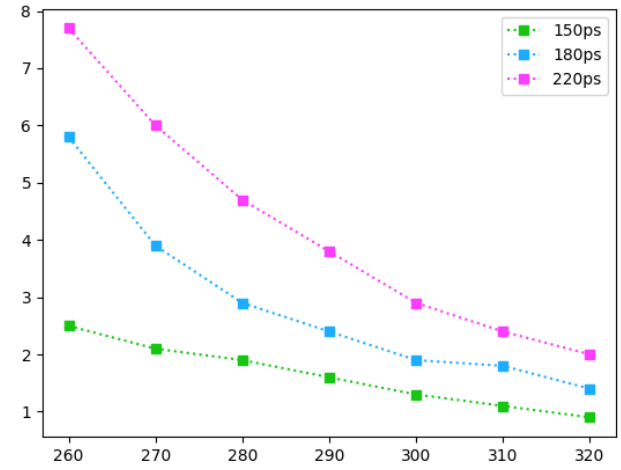
Figure 3.7: τ_2, rows = 20-80, 50-50, 80-20, columns = diff, std dev



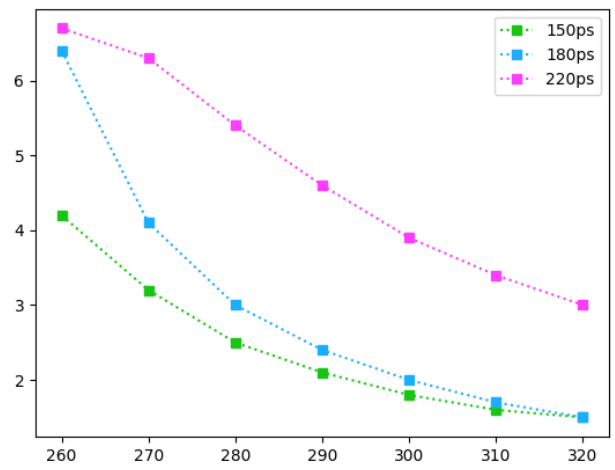
(a)



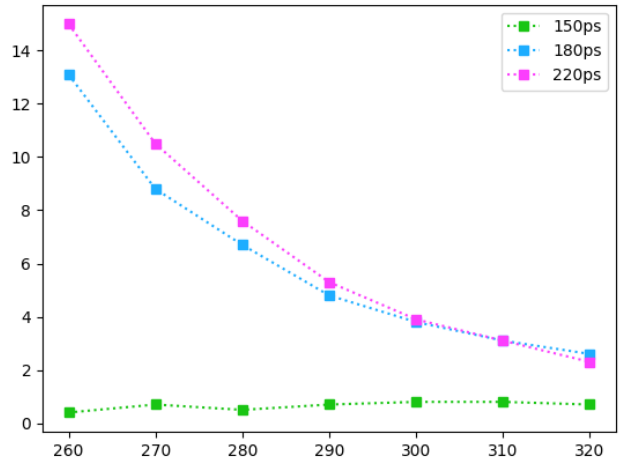
(b)



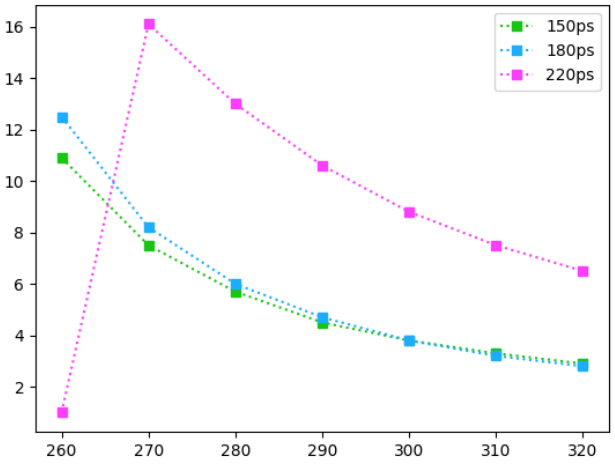
(c)



(d)

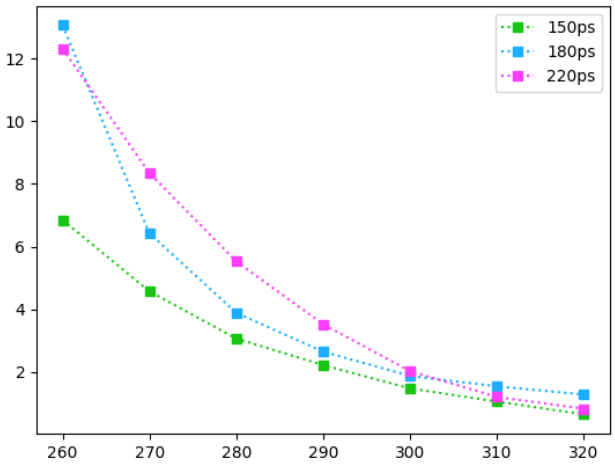


(e)

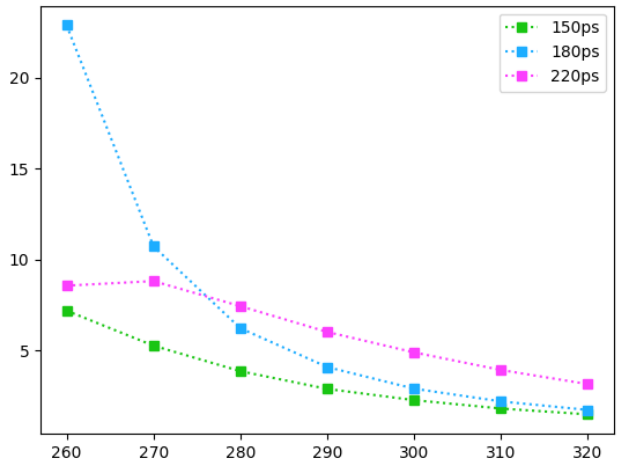


(f)

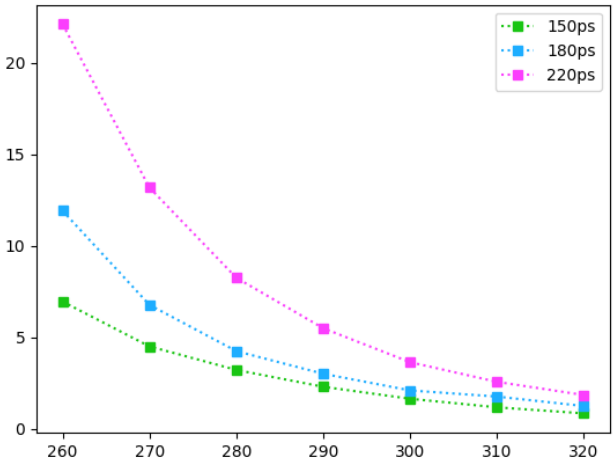
Figure 3.8: intensities, rows = 20-80, 50-50, 80-20, columns = diff, std dev



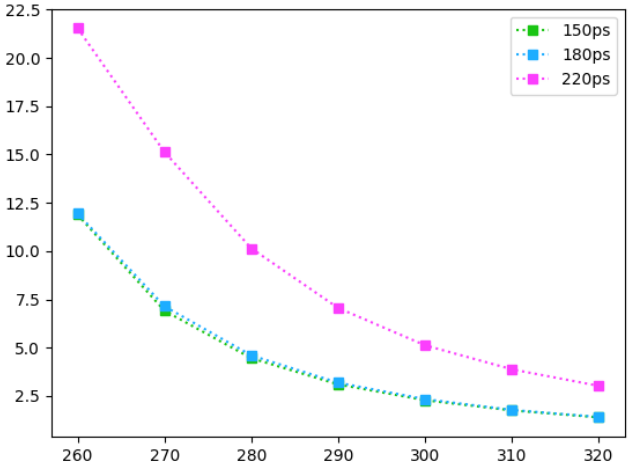
(a)



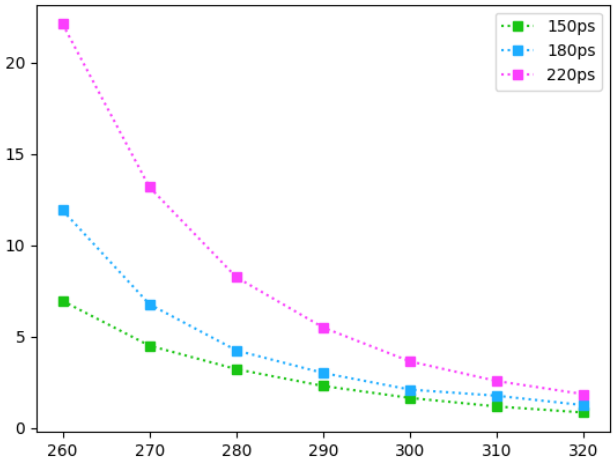
(b)



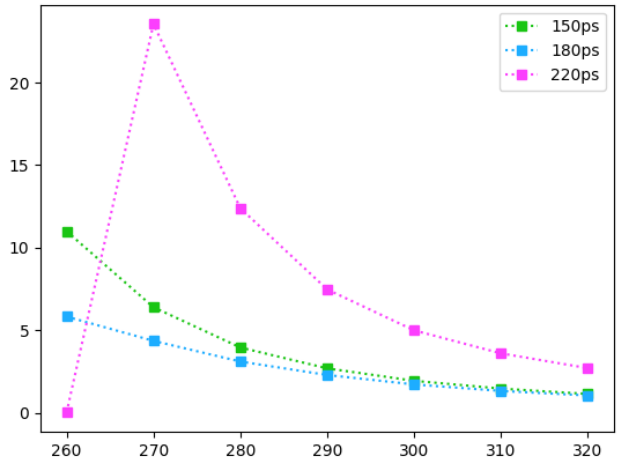
(c)



(d)



(e)



(f)

3.3 Single Gaussian IRF

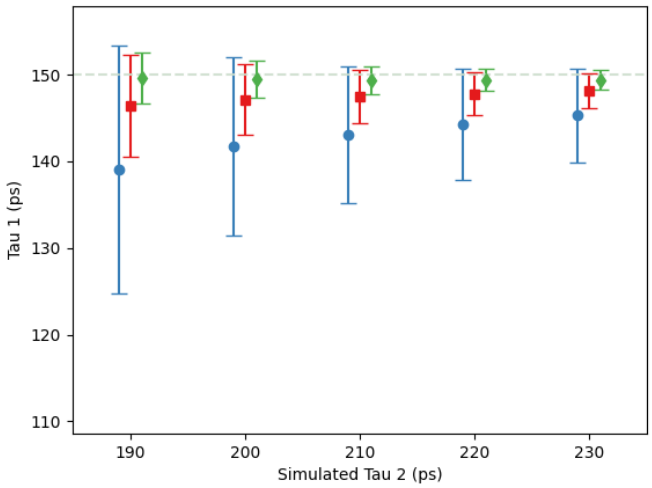
To investigate how the instrument resolution function affects the software, the three Gaussian resolution function in use so far can be simplified to a single Gaussian (refer to ...). As a sanity check, we can compare how the program performs with a single Gaussian resolution function versus the previous three Gaussian resolution function. Setting $\tau_1 = 150$ps, as this previously gave us the best results, we get the data in Figure **??**. We can then compare this to Figure **??** and see that the single Gaussian resolution function produces remarkably similar results, demonstrating the single Gaussian approximation to be valid.

With that established, spectra were generated and fitted for FWHM values of 220ps, 180ps, 150ps and 100ps, with τ_1 set to 150ps and τ_2 ranging from 180-230ps. In general, the trend we expect is the same increase in precision and accuracy with increasing τ_2 that we see previously, but also an increase in both with narrower resolution functions. We would see this as multiple, separate, decreasing trendlines, ordered so that those corresponding to wider resolution functions were always higher. The results we get are shown in Figures **??**-**??**, and we can analyse them in two parts.

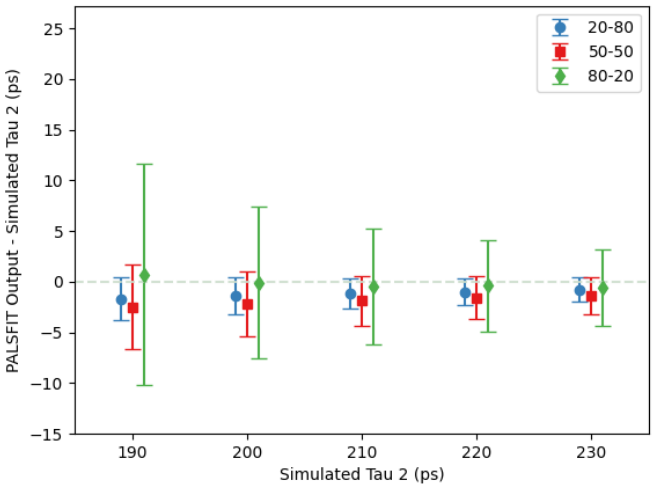
If we just look at the subfigures on the left ((a),(c) and (e) for each figure), which represent the difference between the fit and original data, we see that for the 20%-80% and 50%-50% data the data follows the expected trend. When the τ_1 intensity is set to %80, however, we notice that accuracy doesn't seem to follow the expected relationship with the width of the resolution function. In many cases we have an outright reversal of the predicted relationship and in each figure we have significant crossing between the trendlines.

Looking at the subfigures on the right, representing the standard deviation of the fitting given by PALSFIT, we again see three outliers where the predicted relationship between precision and resolution function width is reversed, in Figures **??**, **??** and **??**, though in the case of **??**, this reversed relationship quickly reverts to the expected one as τ_2 increases.

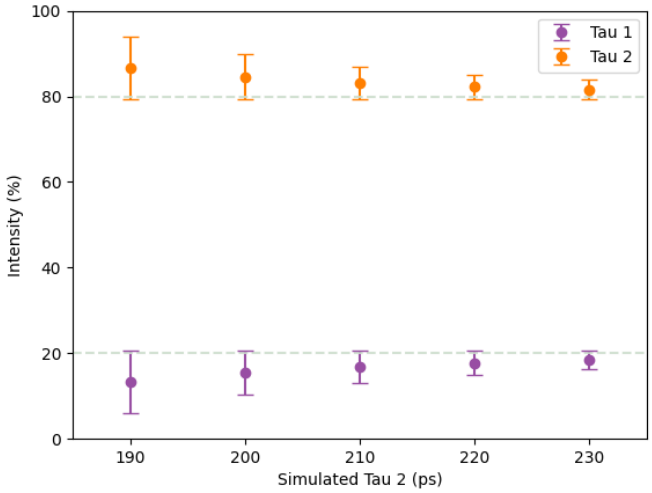
Figure 3.9



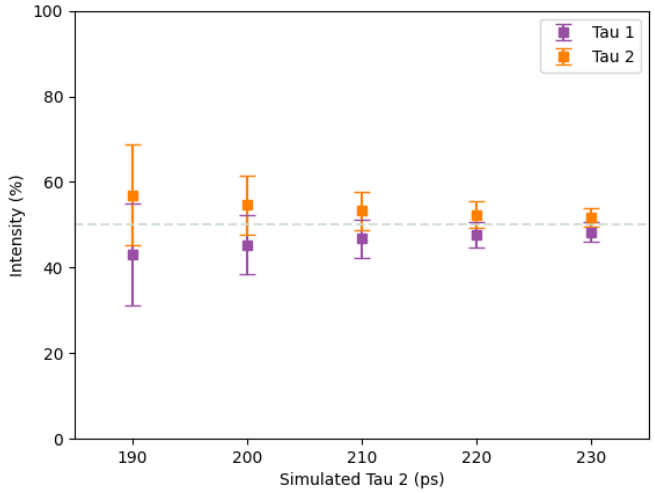
(a) fixed $\tau_1 = 150ps$



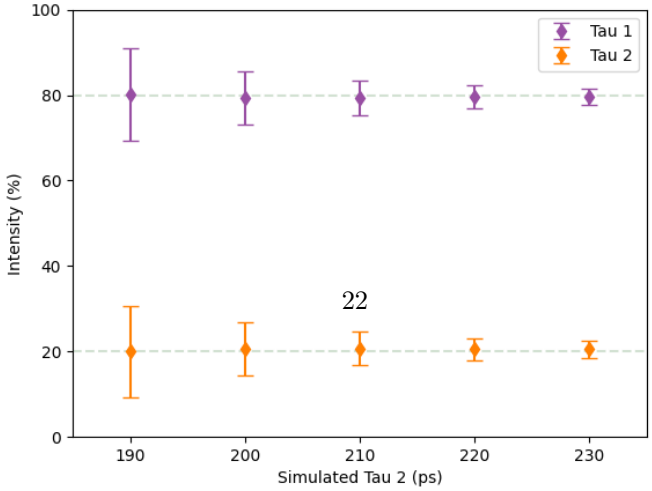
(b) τ_2 difference



(c) $\tau_1 = 20\%, \tau_2 = 80\%$

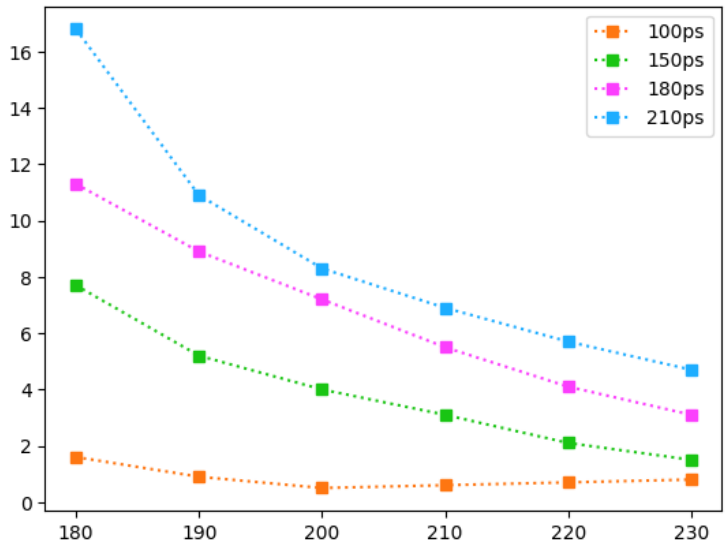


(d) $\tau_1 = 50\%, \tau_2 = 50\%$

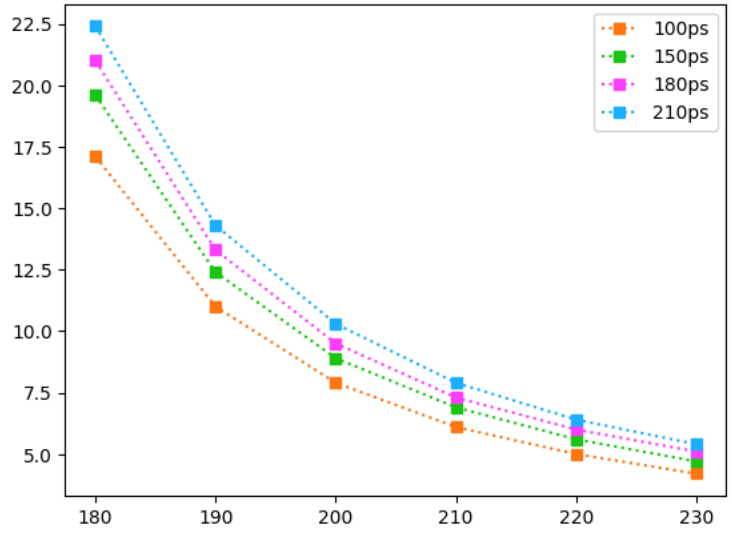


(e) $\tau_1 = 80\%, \tau_2 = 20\%$

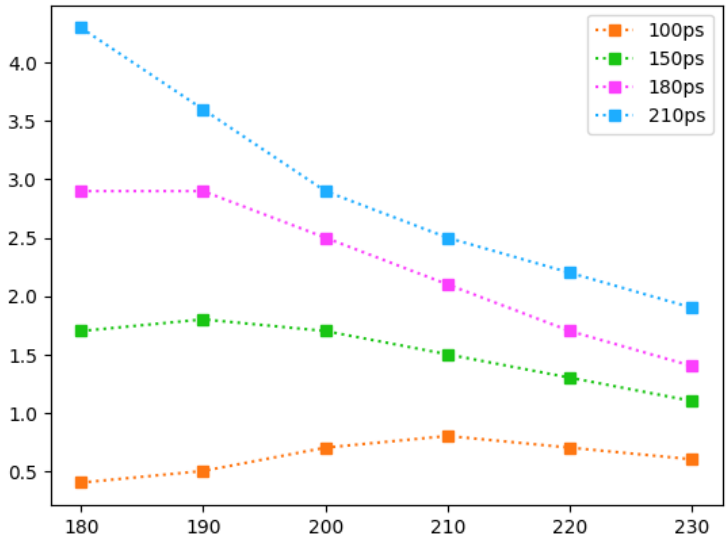
Figure 3.10: τ_1, rows = 20-80, 50-50, 80-20, columns = diff, std dev



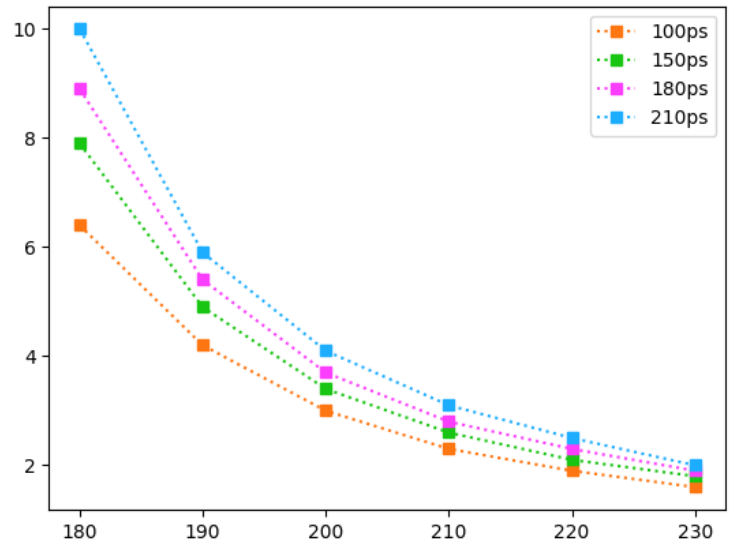
(a)



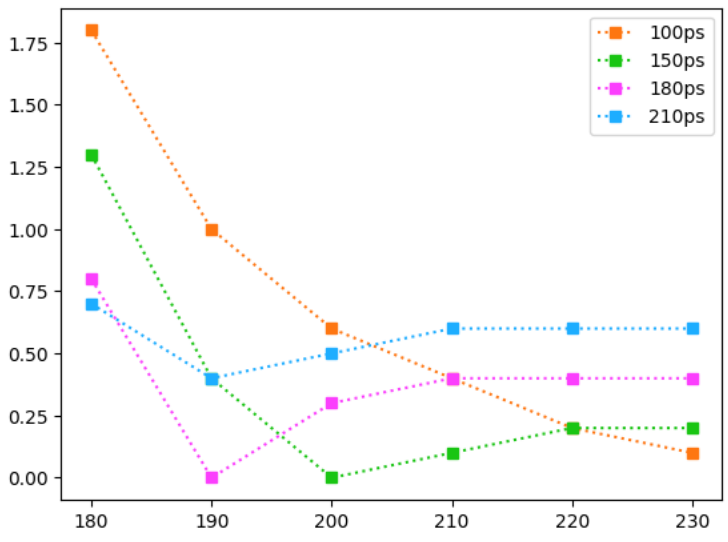
(b)



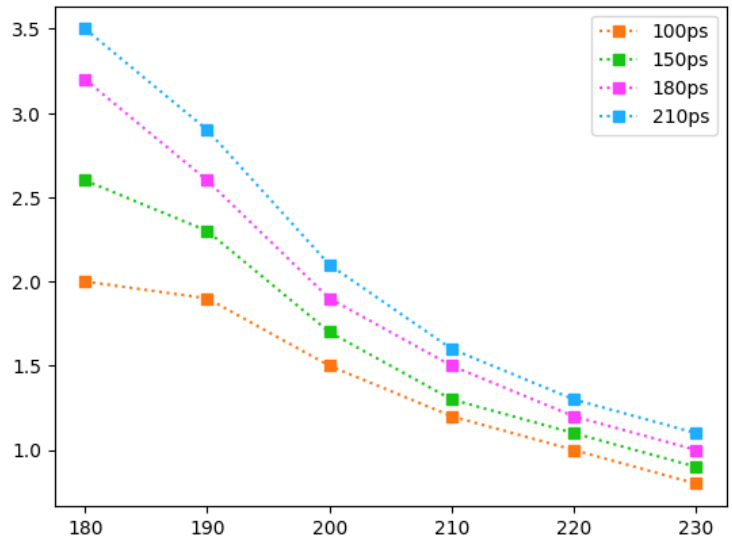
(c)



(d)

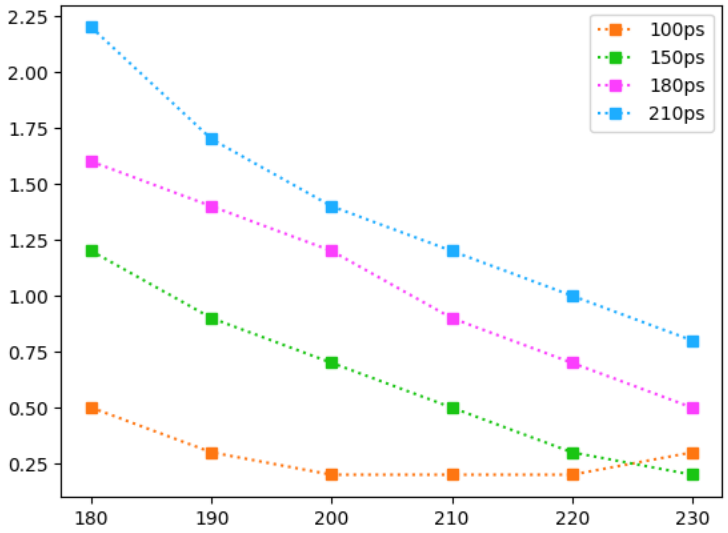


(e)

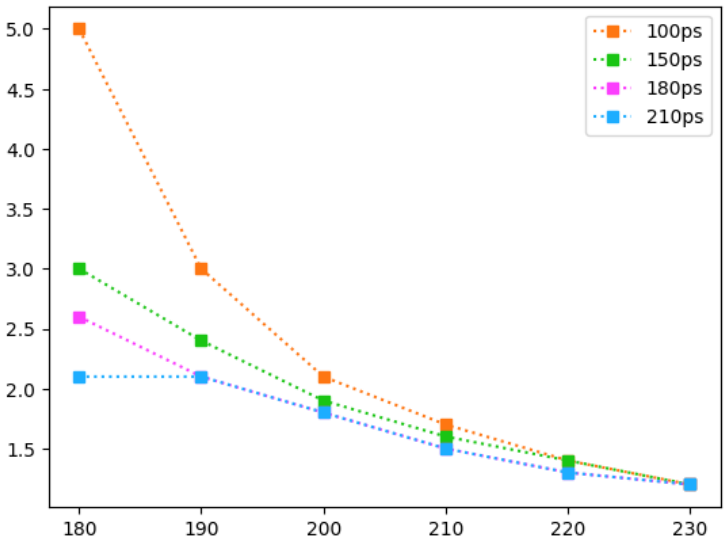


(f)

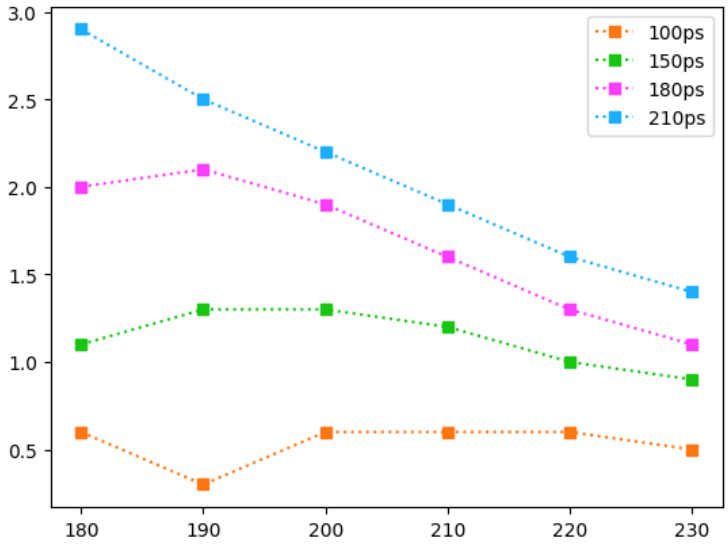
Figure 3.11: τ_2, rows = 20-80, 50-50, 80-20, columns = diff, std dev



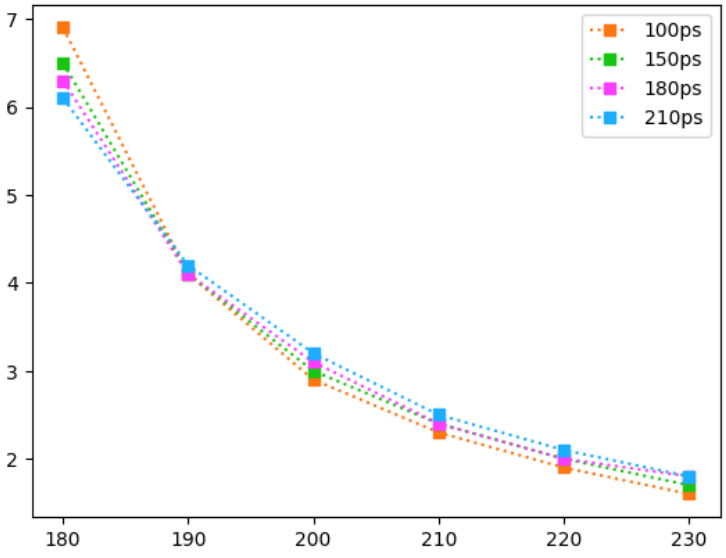
(a)



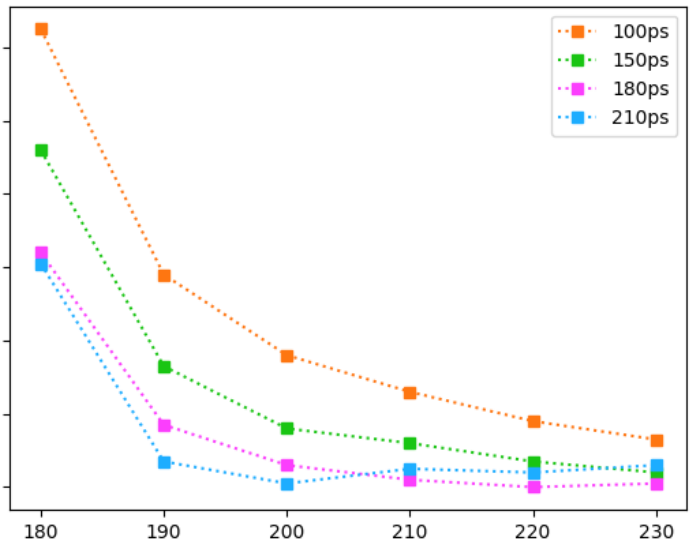
(b)



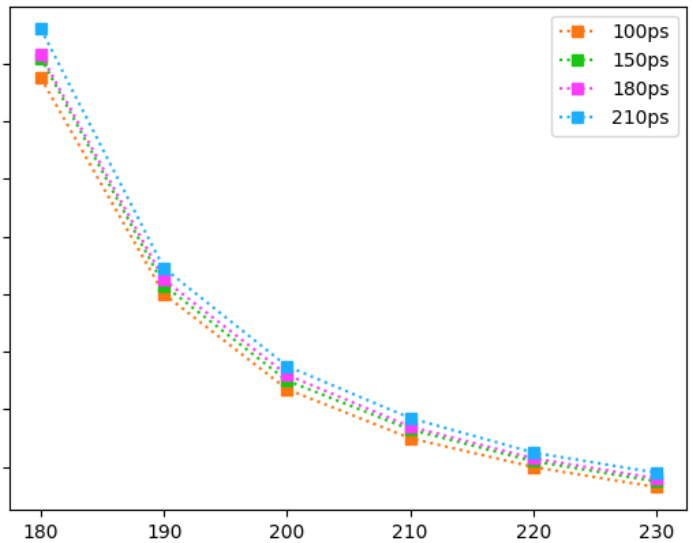
(c)



(d)

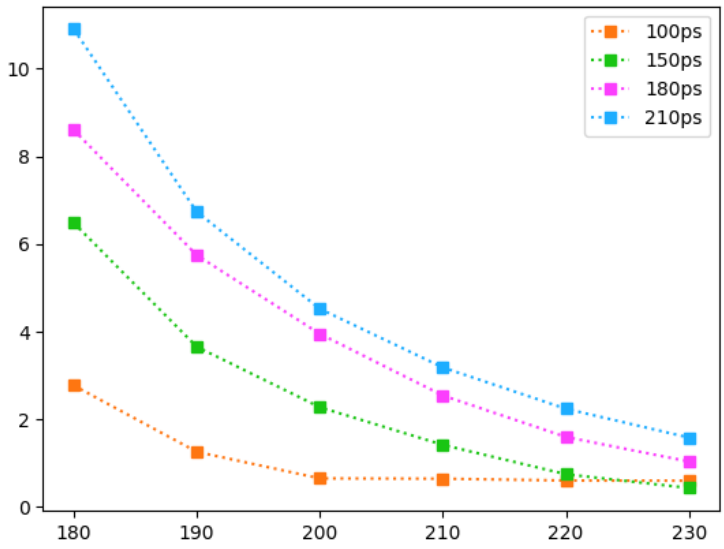


(e)

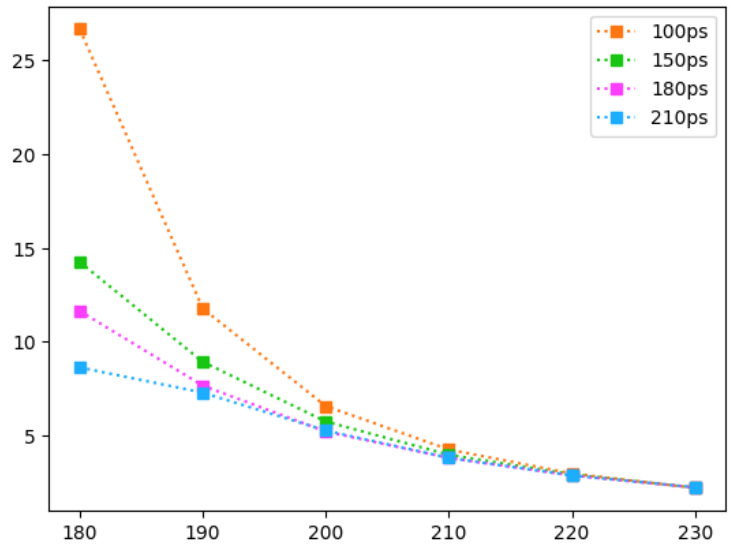


(f)

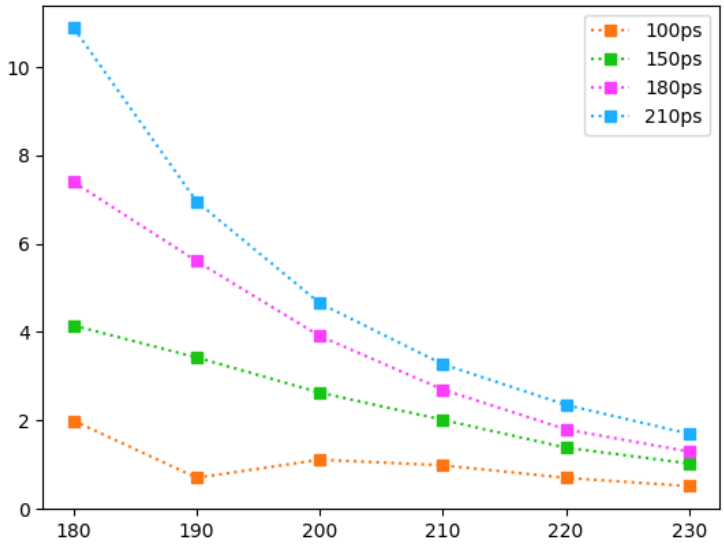
Figure 3.12: intensities, rows = 20-80, 50-50, 80-20, columns = diff, std dev



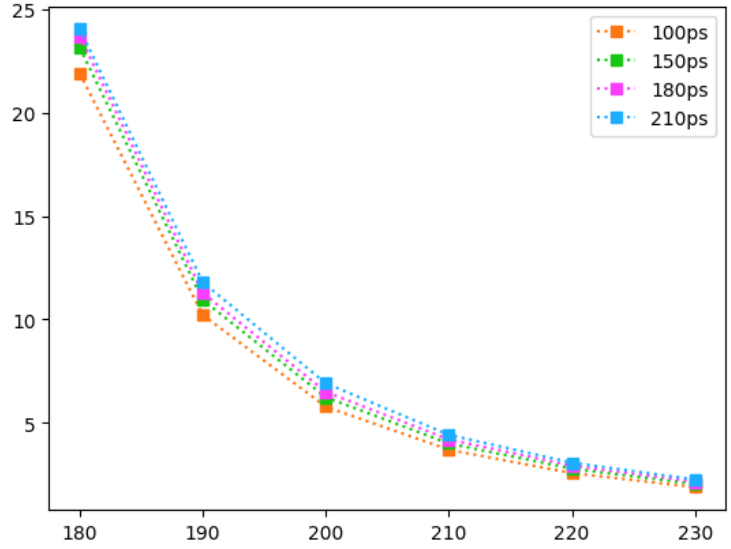
(a)



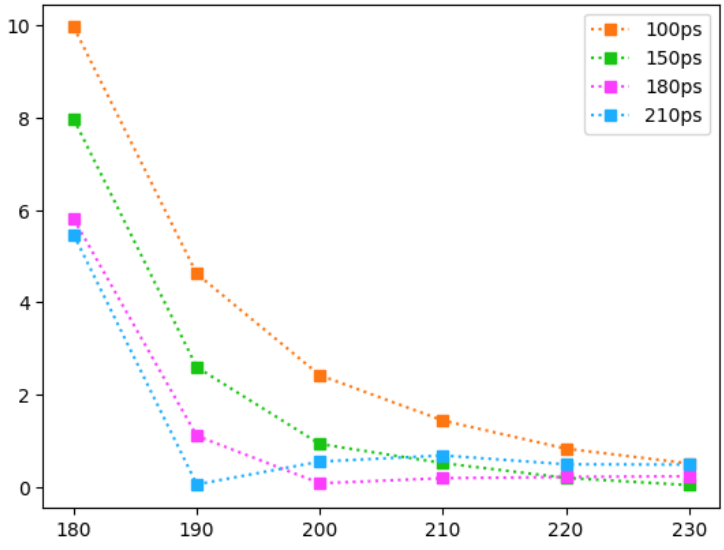
(b)



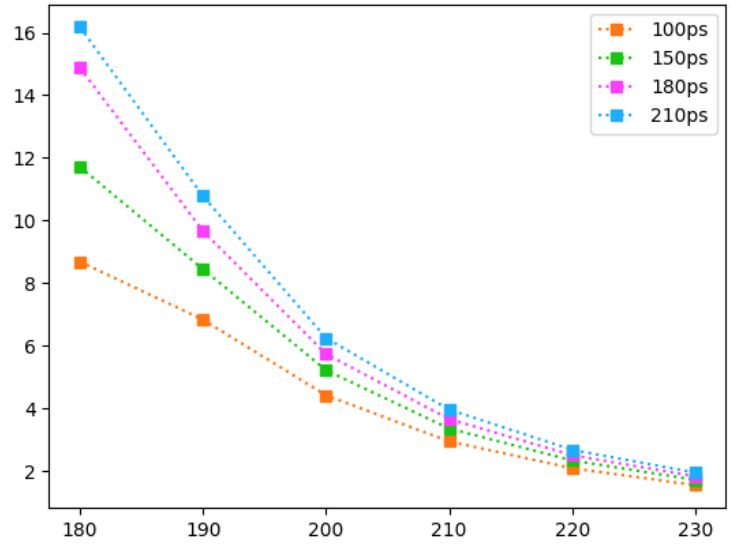
(c)



(d)



(e)



(f)

3.4 Number of counts

Another variable worth examining is the total number of counts in a given spectrum. When generating spectra using PALSSIM, this is given as the area of the spectrum without the background and a separate background value. Both were kept constant in all previous runs, with the area set to 5.65×10^6 and the background value set to 8.5. Expectations are that a higher number of counts would make spectrum analysis easier. To test this hypothesis, spectra with generated for $\tau_1 = 150$ps, $\tau_2 = 180$-230ps and a 210ps FWHM single gaussian resolution function, tripling the background area to 1.7035×10^7.

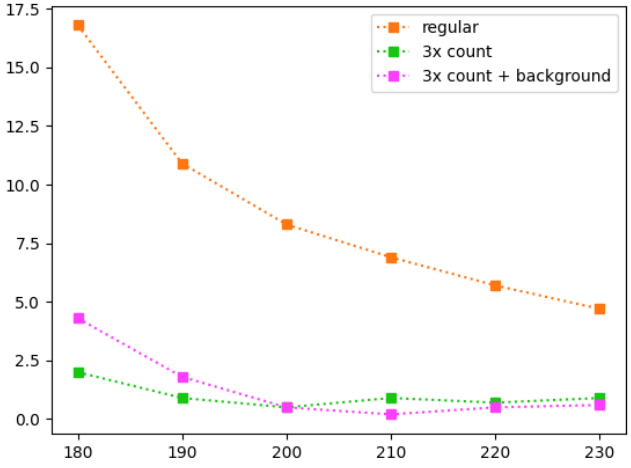
If we just increase the total area of the spectra and maintain the same background value as before, however, we inadvertently end up increasing the signal to noise ratio. While this would be desirable, in a practical setting an increase in total number of counts most likely corresponds to an increase in background noise. As such, spectra were also generated with both the tripled area and a proportional 3x increase in the background value (set to 25.5).

The generated spectra were then analyzed with PALSFIT, and the results are summarized in Figures **??**-**??**, with a "regular" run for comparison. The regular run is simply the one represented in Figure **??**, which used the original, unaltered area and background value.

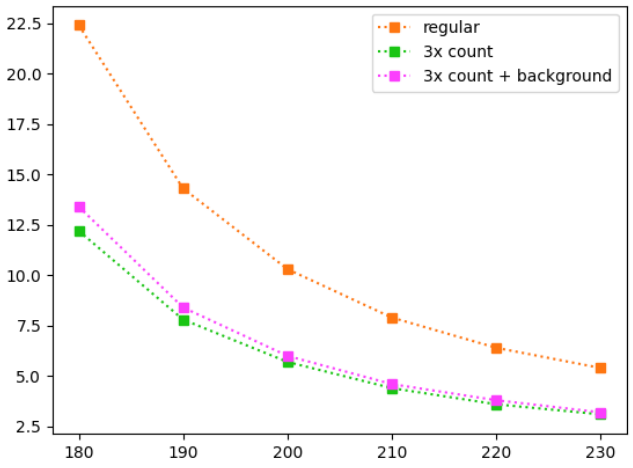
Looking first at the standard deviations (subfigures (b), (d) and (f), on the right side of the page), we see that, as expected, the data from the regular run is in most cases significantly less precise than the other two runs. Additionally, the data with the proportionally scaled background value seems to be slightly less precise than its unscaled counterpart, though in many cases just barely so, if at all. Exceptions tend to occur, however, for the shortest lifetime separation, when τ_2 is set to 180ps.

Looking at the left side of the figures ((a), (c) and (e)), we see that for the most part the first trend seem to hold true, with the regular run being significantly less accurate than the other two. The exception to this seems to be when the relative τ_1-τ_2 intensity is 80%-20%, in which we often have a reversal of expectations, at times with some significant crossover of the relevant trendlines. While, at a glance, the normal and scaled background triple area runs are still for the most part relatively close, with the proportional background looking less accurate overall, this isn't as universally true as it is when looking at the standard deviation. Specifically in Figures **??**, **??** and **??**, when $\tau_1 = 20\%$ and $\tau_2 = 80\%$, PALSFIT seems to have performed worse with the non-scaled background, at least for significant portions of the graphs.

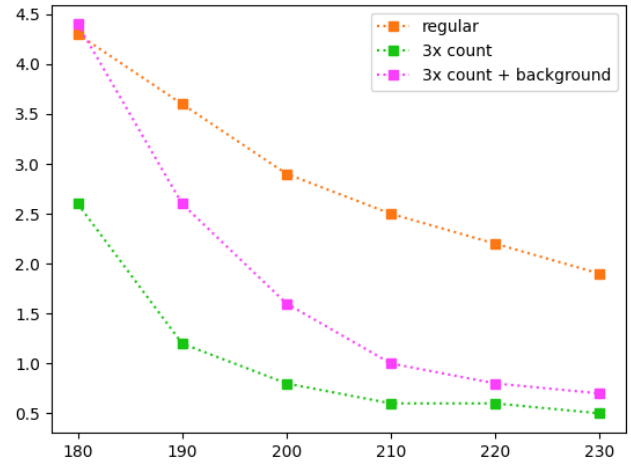
Figure 3.13: τ_1, rows = 20-80, 50-50, 80-20, columns = diff, std dev



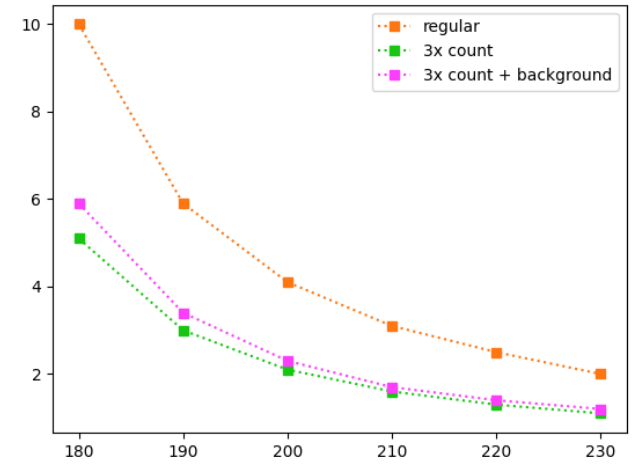
(a)



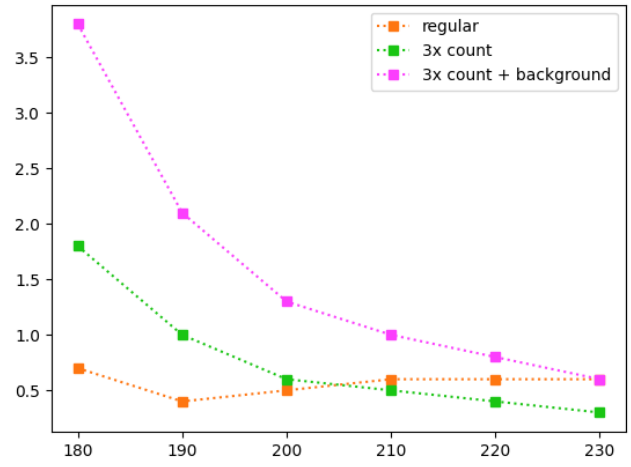
(b)



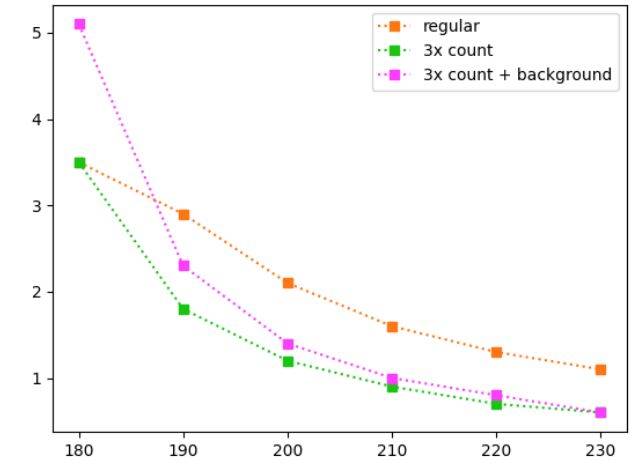
(c)



(d)

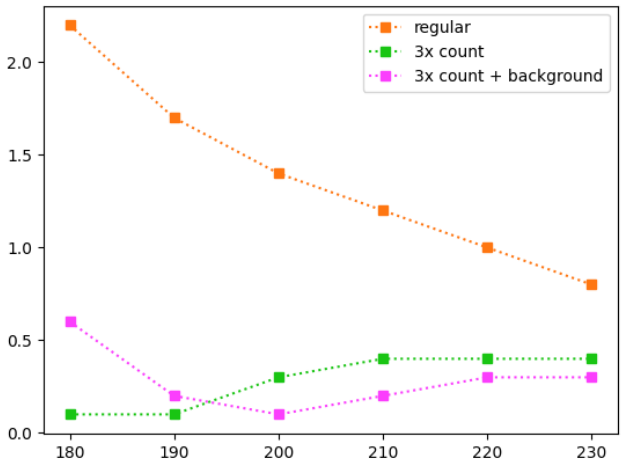


(e)

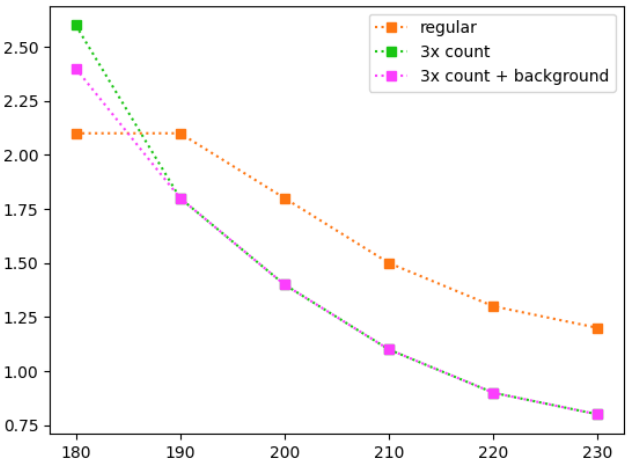


(f)

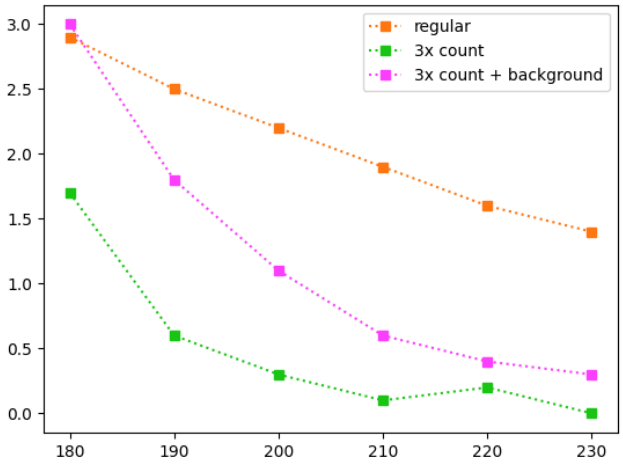
Figure 3.14: τ_2, rows = 20-80, 50-50, 80-20, columns = diff, std dev



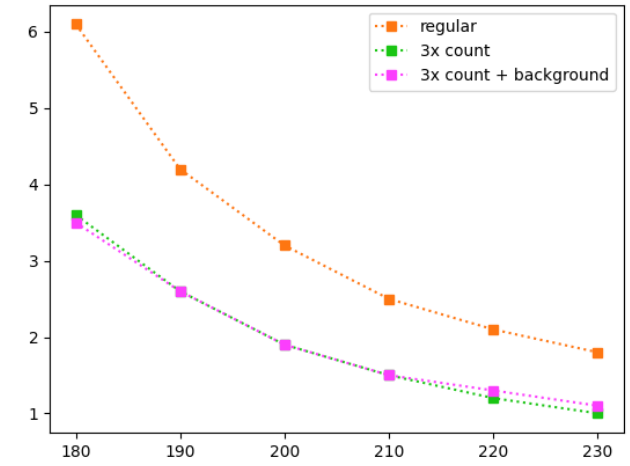
(a)



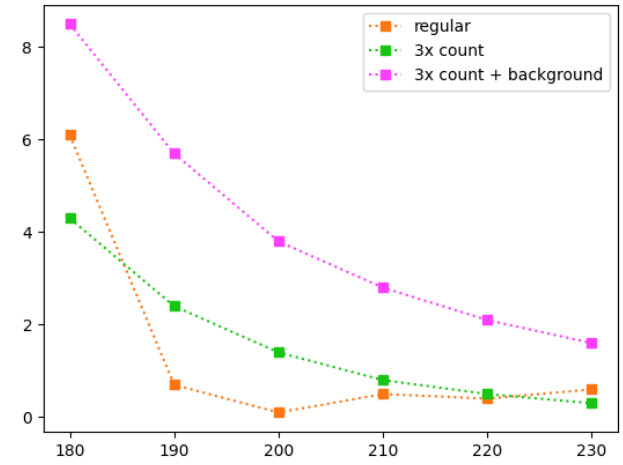
(b)



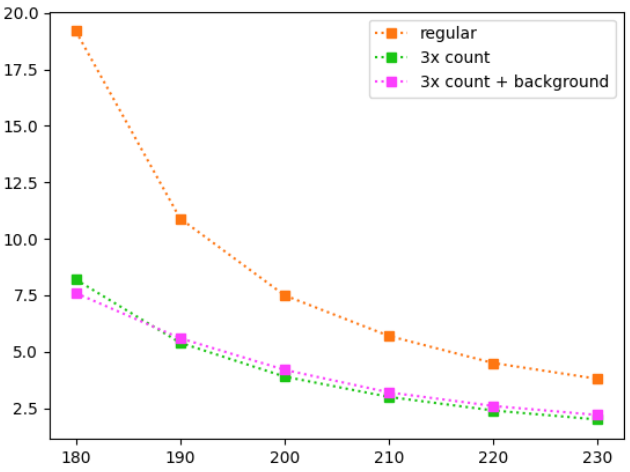
(c)



(d)

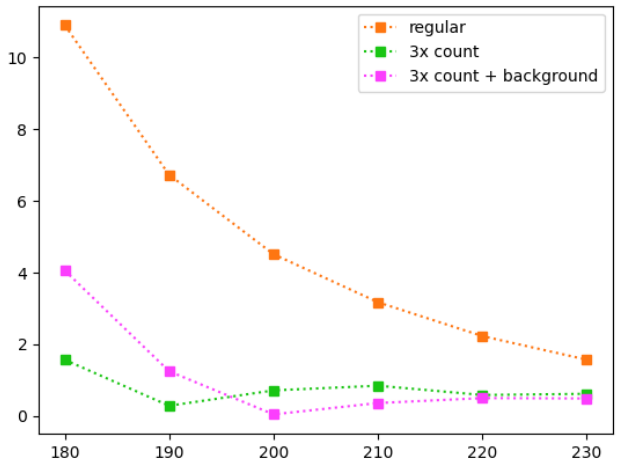


(e)

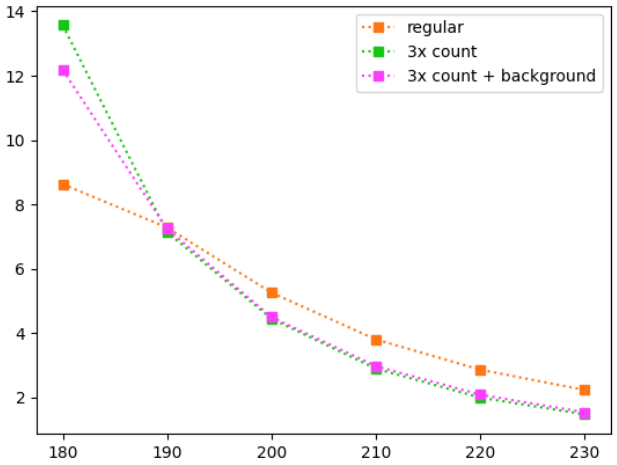


(f)

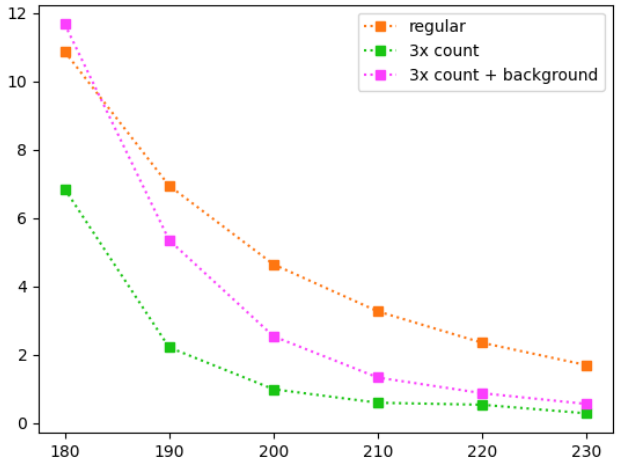
Figure 3.15: intensities, rows = 20-80, 50-50, 80-20, columns = diff, std dev



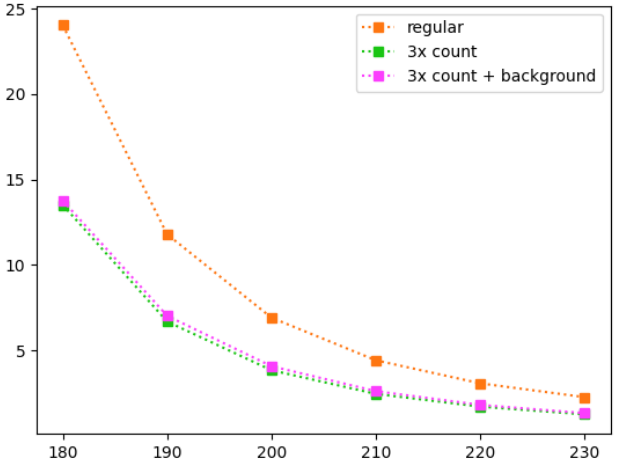
(a)



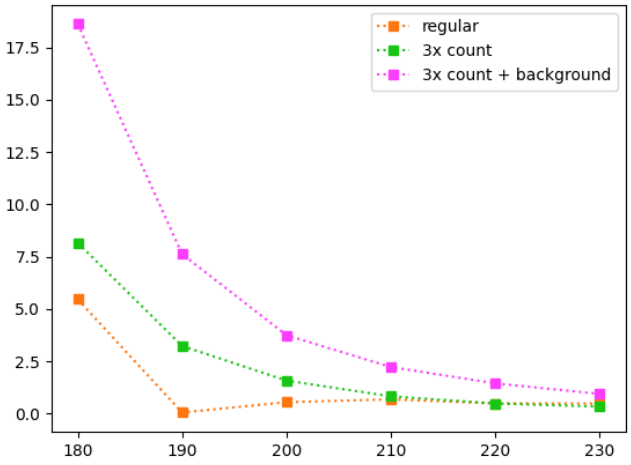
(b)



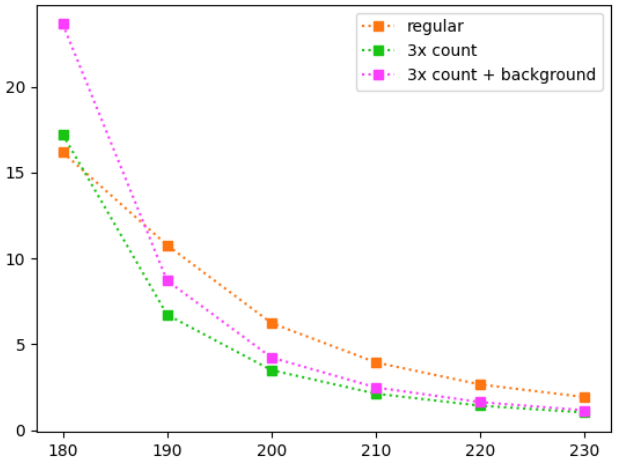
(c)



(d)



(e)



(f)

3.5 Realistic data

Analysis of real world data often differs from how we've conducted our analysis so far. One key factor is that usually the number of lifetimes present in a real life spectrum isn't known beforehand. As PALSFIT requires the user to input the number of fitted lifetimes, this might lead to situations in which the user predicts the wrong number of lifetimes, thus it might be useful to explore how the software behaves in such a situation.

To do so, three materials (?) were modeled, with three lifetime components. All three have two shorter components that differed between them, and a long lifetime component set to 2.6ns with a 0.15% intensity that was kept equal. The relative intensities of the two shorter lifetime components were adjusted to the following values:

relative		absolute	
τ_1	τ_2	τ_1	τ_2
99.5%	0.5%	99.3508%*	0.4993%*
90%	10%	89.865%	9.985%
80%	20%	89.88%	19.97%
50%	50%	49.925%	49.925%
20%	80%	19.97%	9.985%

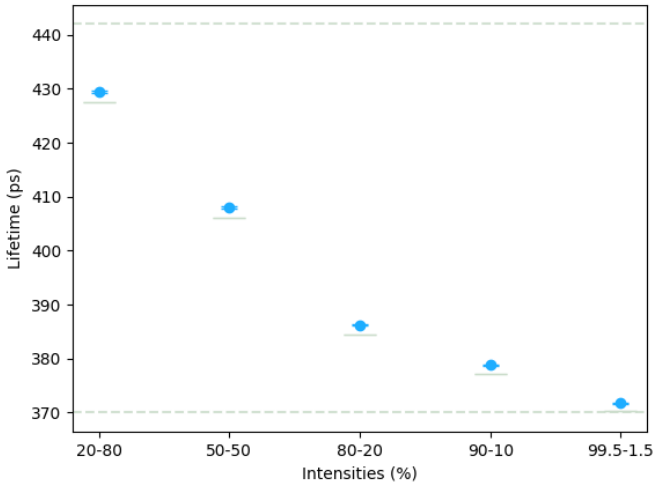
(a) intensities

τ_1(ps)	τ_2(ps)	τ_3(ns)
370	442	2.6
355	444	2.6
348	440	2.6

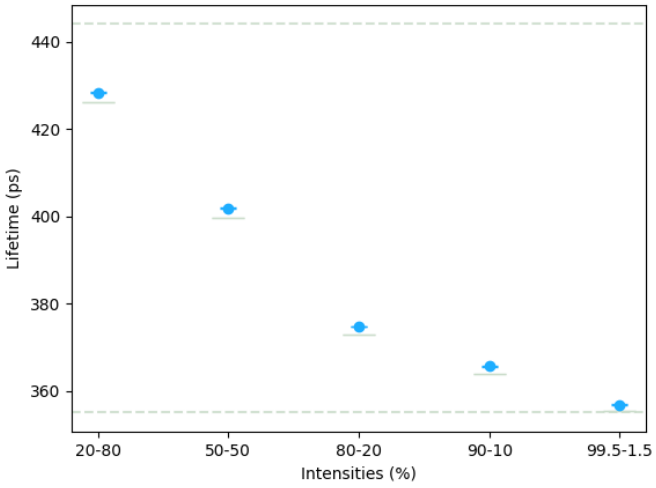
(b) lifetimes

When a single lifetime is fitted, the resulting lifetime seems to tend towards a weighted average of the two shorter lifetimes, as can be seen in the following figures:

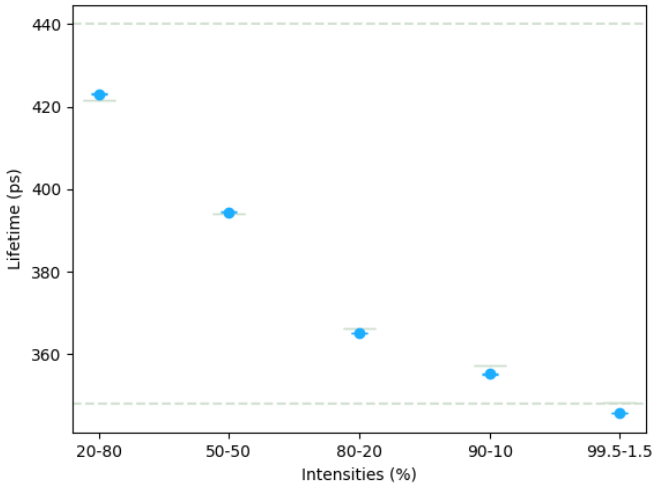
Needs to be finished.



(a) $\tau_1 = 370$, $\tau_2 = 442$

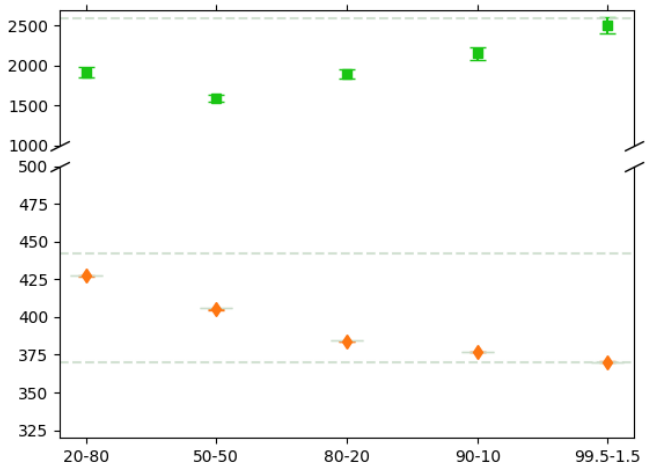


(b) $\tau_1 = 355$, $\tau_2 = 444$

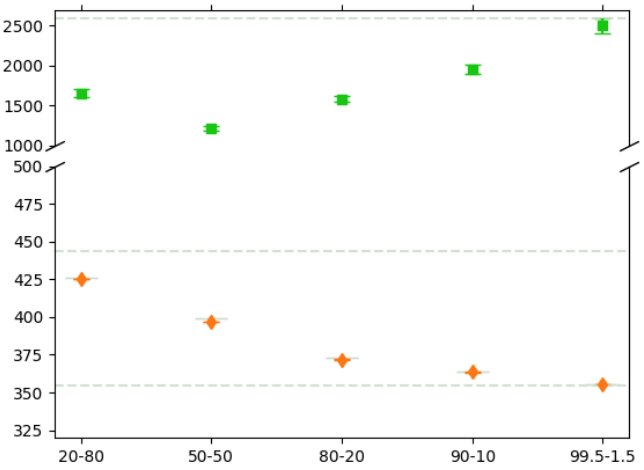


(c) $\tau_1 = 348$, $\tau_2 = 440$

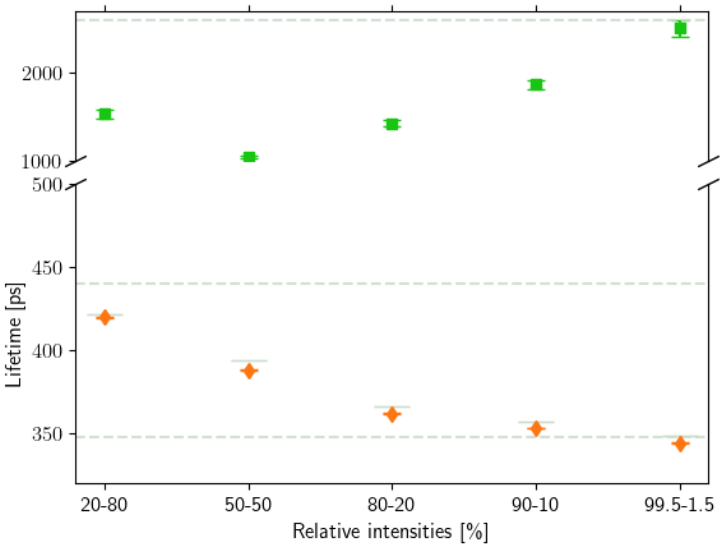
Figure 3.16: single lifetime fit



(a) $\tau_1 = 370$, $\tau_2 = 442$

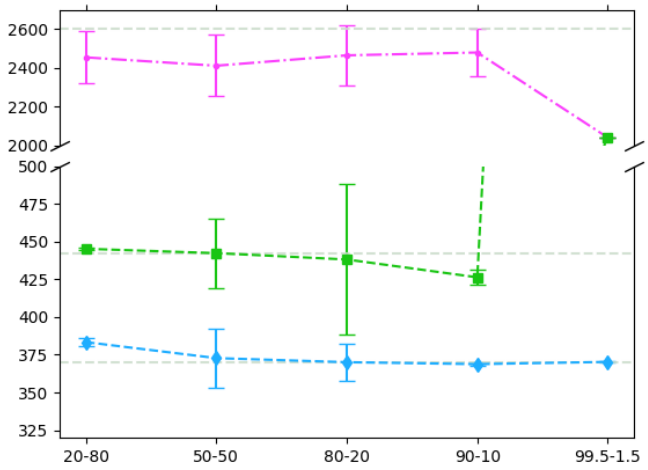


(b) $\tau_1 = 355$, $\tau_2 = 444$

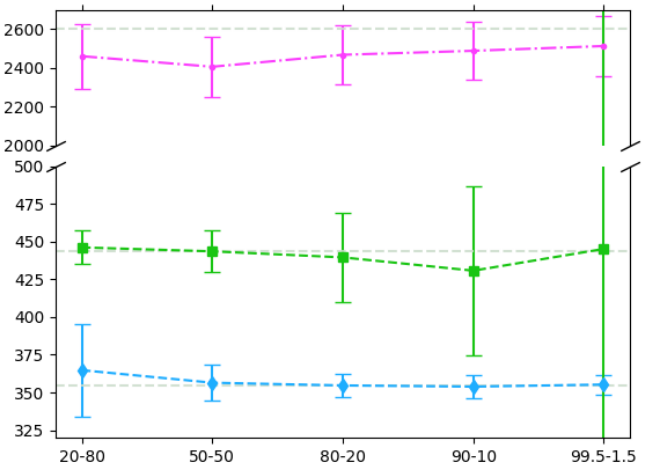


(c) $\tau_1 = 348$, $\tau_2 = 440$

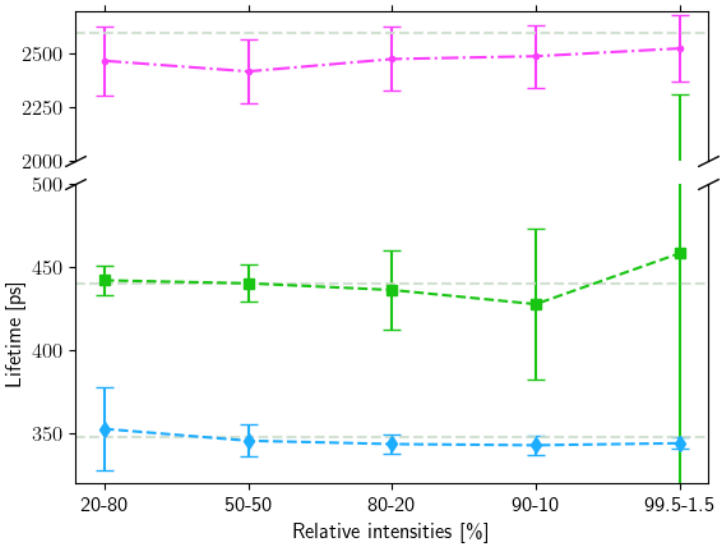
Figure 3.17: two lifetime fit



(a) $\tau_1 = 370$, $\tau_2 = 442$



(b) $\tau_1 = 355$, $\tau_2 = 444$



(c) $\tau_1 = 348$, $\tau_2 = 440$

Figure 3.18: three lifetime fit

3.6 Using the STM to vary τ_1:

Using STM, vacancy concentration rate was varied for a given value of defect lifetime τ_2. This was done for $\tau_2 = 370, 442$. Vacancy conc. rate was varied as such:

Table 3.2: Vacancy Conc.

Conc.
1E15
2E15
4E15
6E15
8E15
1E16
2E16
4E16
6E16
8E16
1E17
2E17
4E17
6E17
8E17
1E18

Needs to be typed up