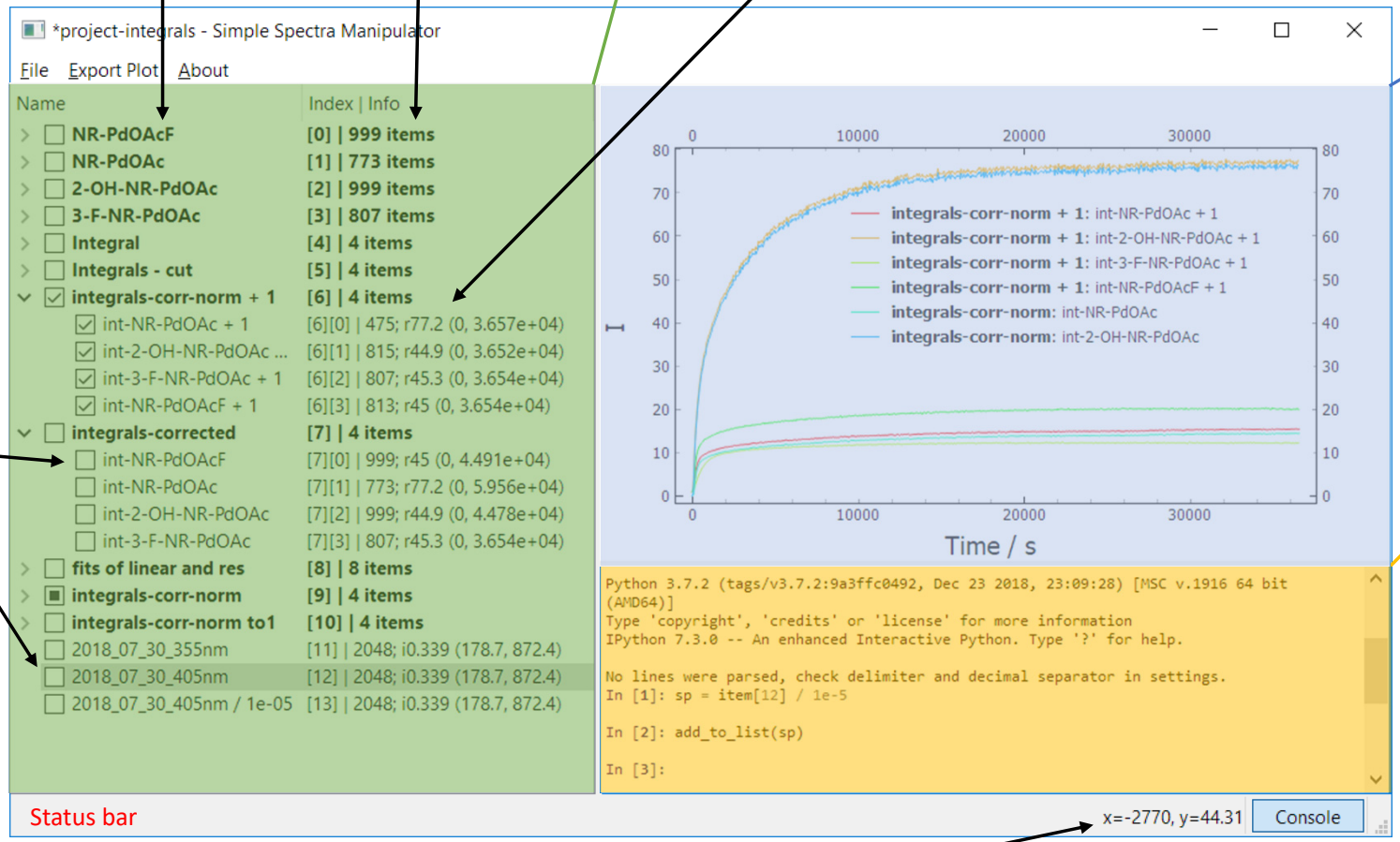
Overview



- spacing (x difference among points):
  - r – probably regular spacing among all datapoints
  - i – probably irregular spacing among all datapoints
  - value calculated as average for all datapoints
- x range (x_{min}, x_{max}) written with 4 significant figures

- running on PyQtGraph (modified)

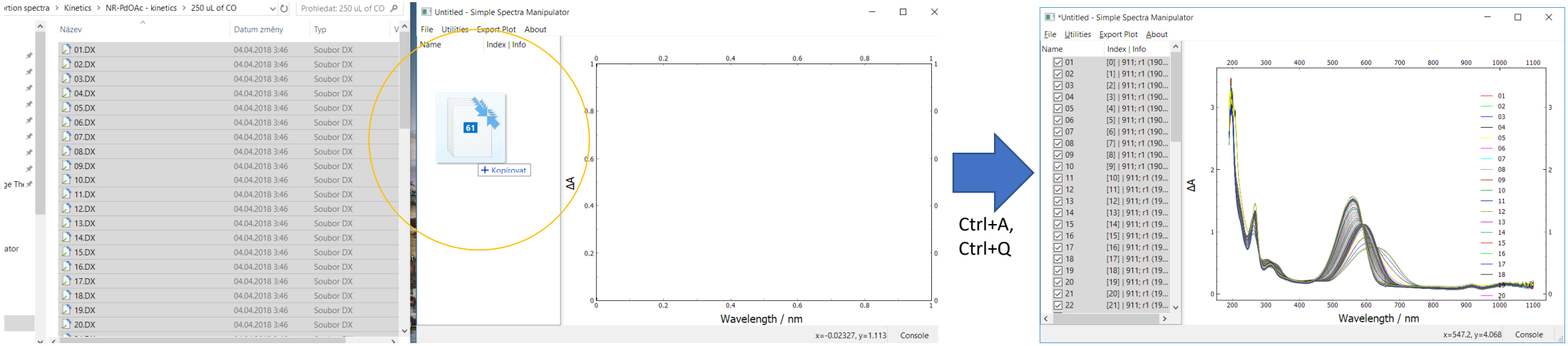
http://www.pyqtgraph.org/

- IPython console running
- runs on qtconsole

https://ipython.org/ipython-doc/3/interactive/qtconsole.html

Importing files and Plot Widget interaction

- To import some Spectra, just **drag** the files from file exporer and **drop** them in the Tree Widget window (you can also from **File->Import Files**), then to select all of the imported spectra, press Ctrl + A and then Ctrl + Q which is the shortcut for checking the selected spectra. And so on.... The program is intuitive I hope...



- Some DX files contains also errors, in this way, the two items will be imported for one DX file, you can get rid of the errors by deleting these items only (use select every n-th item function – *vide infra*)
- To put spectra in group, just select the spectra you want to put in the group and right click on the selection and choose **Add Items To Group**
- Saving to project is made by the same way as in other programs, Ctrl + S for Save or Ctrl + Shift + S for Save As, the extension for project files is **.smpj,** you can load the project from the **File->Open project** or just drag-drop the **smpj** file to Tree Widget window like for spectra

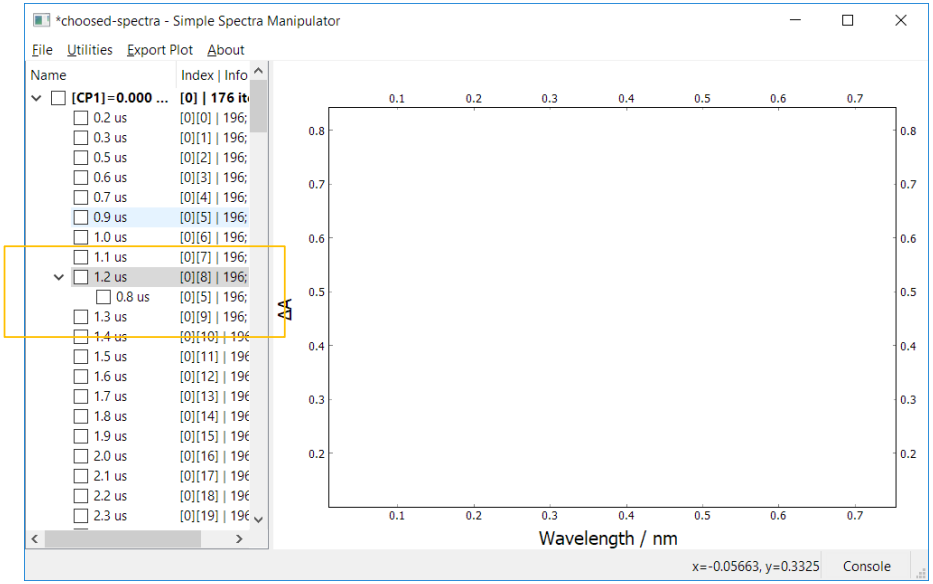
Plot Widget interaction

- Strongly prefer to use default mouse mode (3 button mode, you will get use to it), that is, the **scaling** is done by right mouse button dragging, left mouse button dragging **pans** the scene, **zooming** by mouse wheel as usual
- If you loose the image, just right click in the Plot Widget and click **View All**
- For more information, see the pyqtgraph documentation http://pyqtgraph.org/documentation/mouse_interaction.html

Mooving and copying the items/groups

- You can freely move the individual and multiple in the Tree Widget, as well as groups of spectra. For moving, just use left mouse button dragging. The **copying** withing the list works the same, but you have to **hold Ctrl** when releasing the items in the desired location. The interaction is fast for small number of items, but for large lists, >1000 items, it became slower. This is a bug in PyQt/Qt package and I cannot do anything about that...

- The list is designed that it can only have 2 levels (spectra in group cannot have children). But, during some fast manipulation, the similar situation that is shown below can happen:

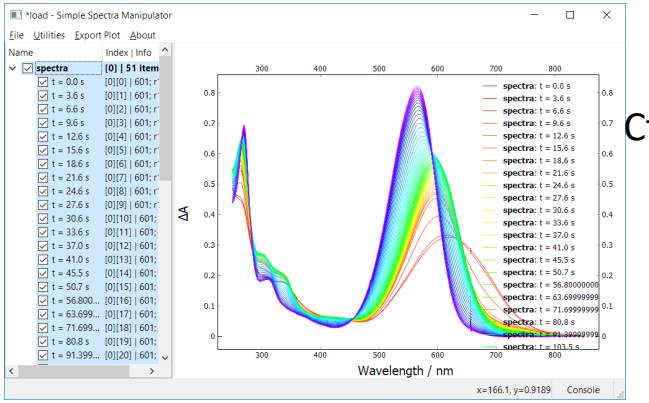


- If that happen, you have to **manually move it** to the grandparent or root node, otherwise, the program will crash sooner or later during some operations, its just a bug that can happen a you have to count with that

- You can also open another SSM program and move the items/groups between different program instances, in this way, the data are always copied regardless of holidng Ctrl or not

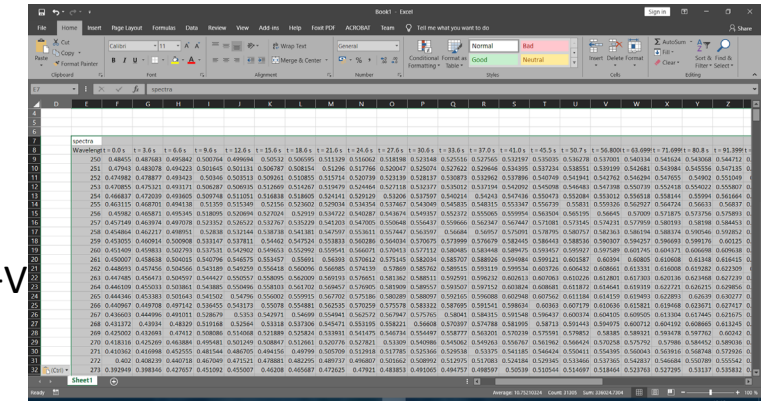
Copying to Excel/Origin

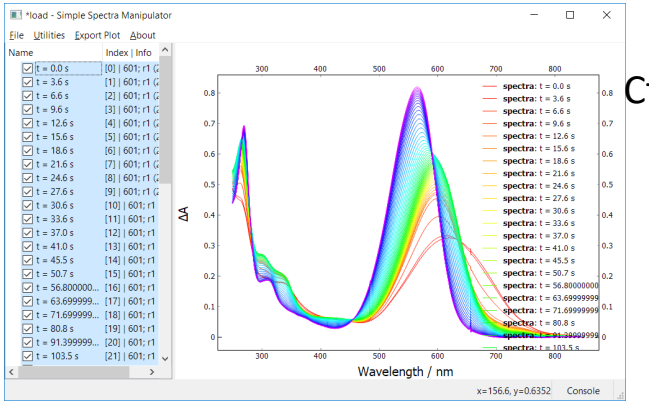
- To copy the spectra, just select the spectra you would like to copy and press Ctrl+C. The results are shown below. If the spectra are **in group**, they will be **concatenated** (this is usually what we want). In this case, all spectra in the group must have the same number of points and same x range, otherwise, error will be raised. If the spectra are **placed in the root** (top level items), they will be **stuck to one another** (only first spectrum seen in Excel). Please, look at Setting Export tab, to change appropriate settings. If you want to retain this compatibility, the delimiter in Clipboard has to be **tabulator \t**. Only with this settings, we can easily transfer data to Excel/Origin by Windows Clipboard. You can also import data from clipboard to the program and also from Excel. Check Settings Import Tab: Clipboard for more info.



Ctrl+C

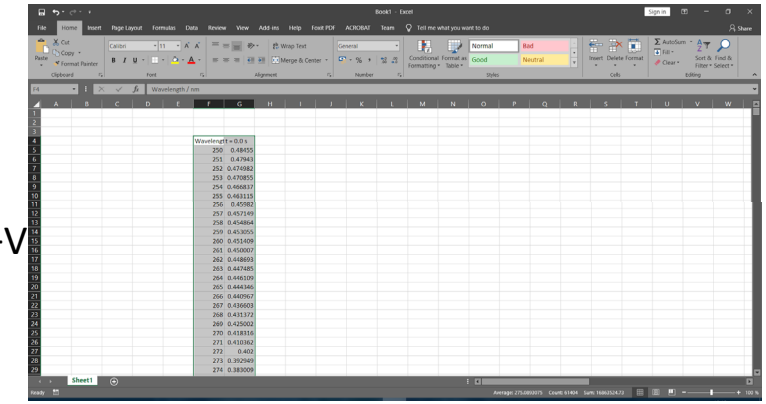
Ctrl+V





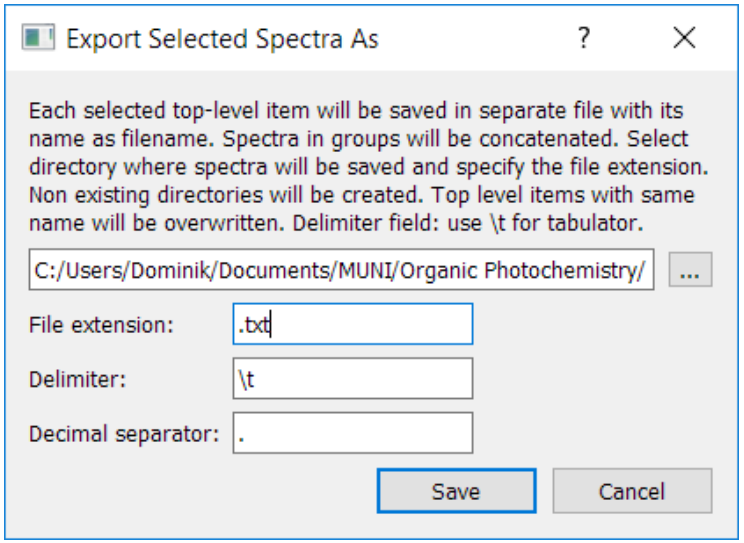
Ctrl+C

Ctrl+V



Exporting spectra to files

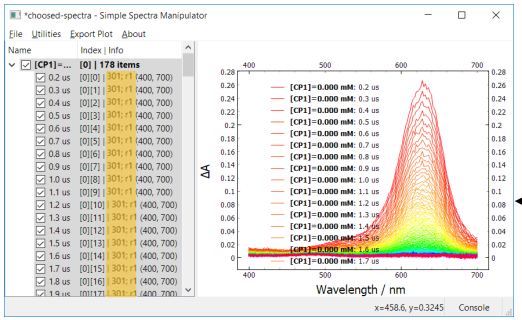
- The exporting of spectra to files works the same way as with copying. Spectra in group will be concatenated. Spectra in root (top level items) will be saved as individual files. Just select the root items you want to export as files and go to **File->Export Selected Spectra As** and this dialog below should appear:

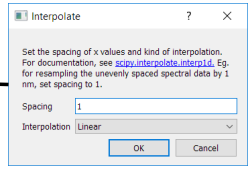


files are saved with **utf-8 encoding**

- Just select the **directory** where you want your spectra to be saved and set the file extension (you can omit the dot, so eg. just write ‘txt’ or ‘csv’), delimiter which is the characted that separates the columns and decimal separator, then click Save
- **The name of the file is determined by the name of the top level item** (if the top level item has no name, ‘Untitled’ with index is used as name)
- **Files** with the same name in the saved directory will be **overwritten** without notification, beware ot that
- For header options, look at settings Export tab

Operations Overview





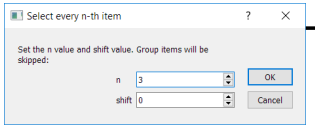
the number of points doubled from 151 to 301 and spacing is changed from ‘r2’ to ‘r1’, the shape of spectra is unchanged, because we upsampled by linear interpolation (increased the number of points)

Interpolate

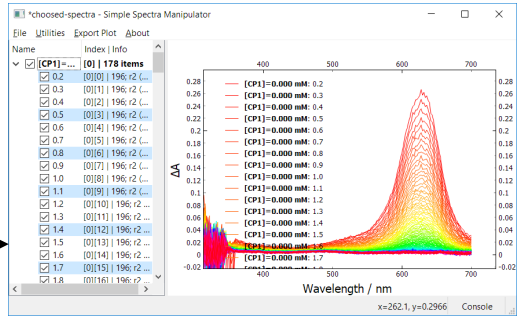
- used for example to change irregular spacing to regular

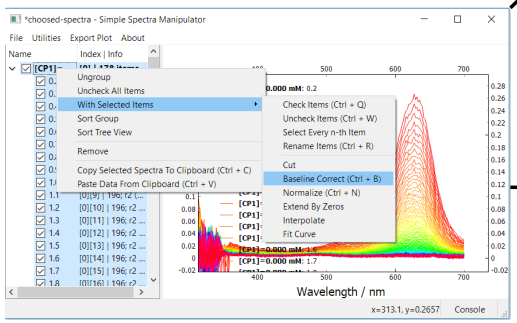
here - change the spacing from 2 nm to 1 nm

Select Every n-th Item

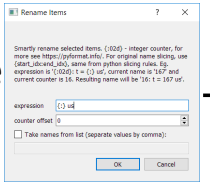


n=3, shift=0

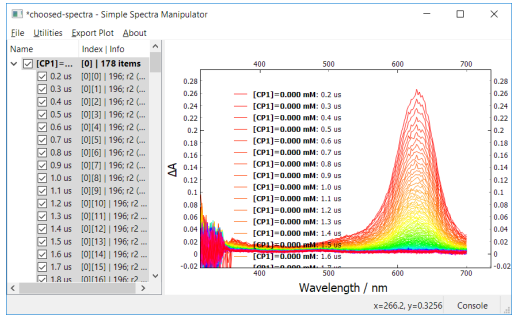


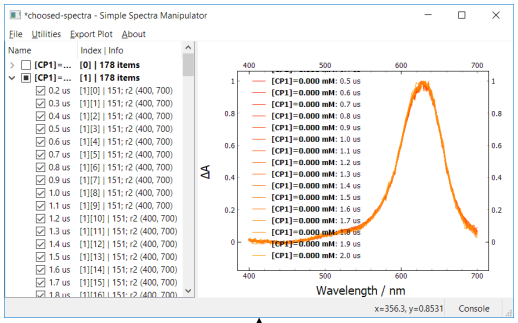


Rename Items



added ‘ us’ after each spectrum name

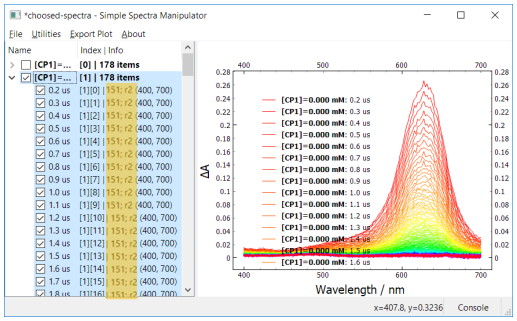




Normalize

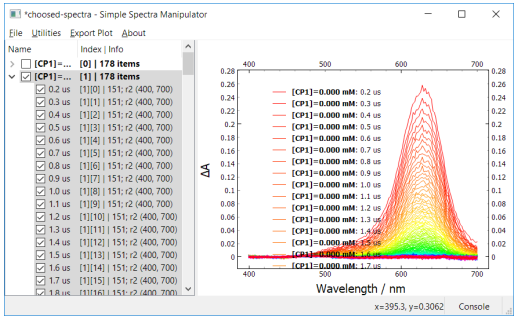
plotted only up to 2 us

Cut to 400 - 700 nm



Baseline correct

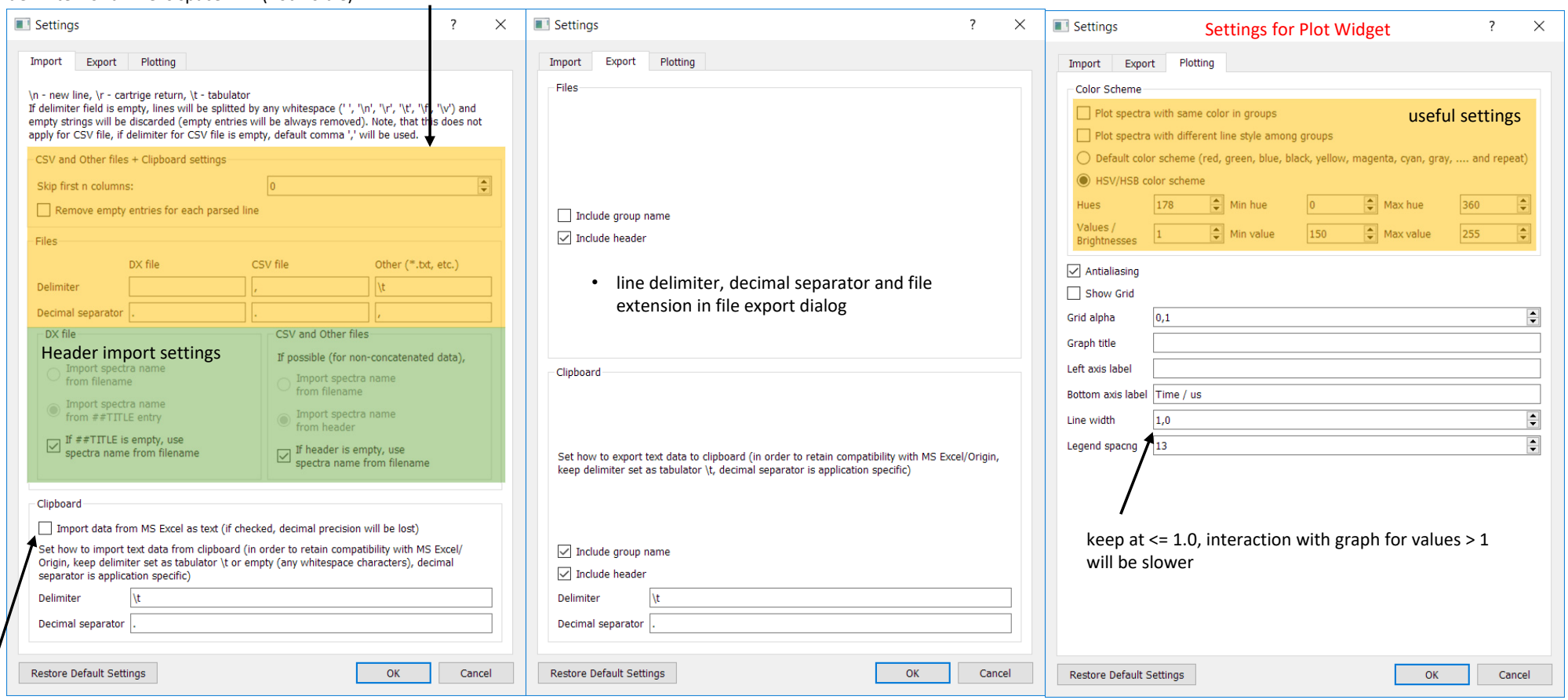
this should not be done for transient spectra this way, here just for demonstration purposes (subtracted average from 410-470 nm)



- Extend by zeros only extends the spectra by zeros for selected x range, while keeping the spacing the same as original spectrum

Settings

- most important part, sets how to import files (sets the **line delimiter** and **decimal separator**) for different kind of files, for files, that has for example empty entries (only delimiter eg. two commas) at the beginning of the file, sets the number of columns that will be skipped, or there is an option that will remove all empty entries from each line, please, consider these options based on the file you are trying to import
- the CSV file is parsed using the csv module for python
- delimiter for dx file is space - " " (not visible)
- columns that will contain at least one NAN (not a number) value will be excluded

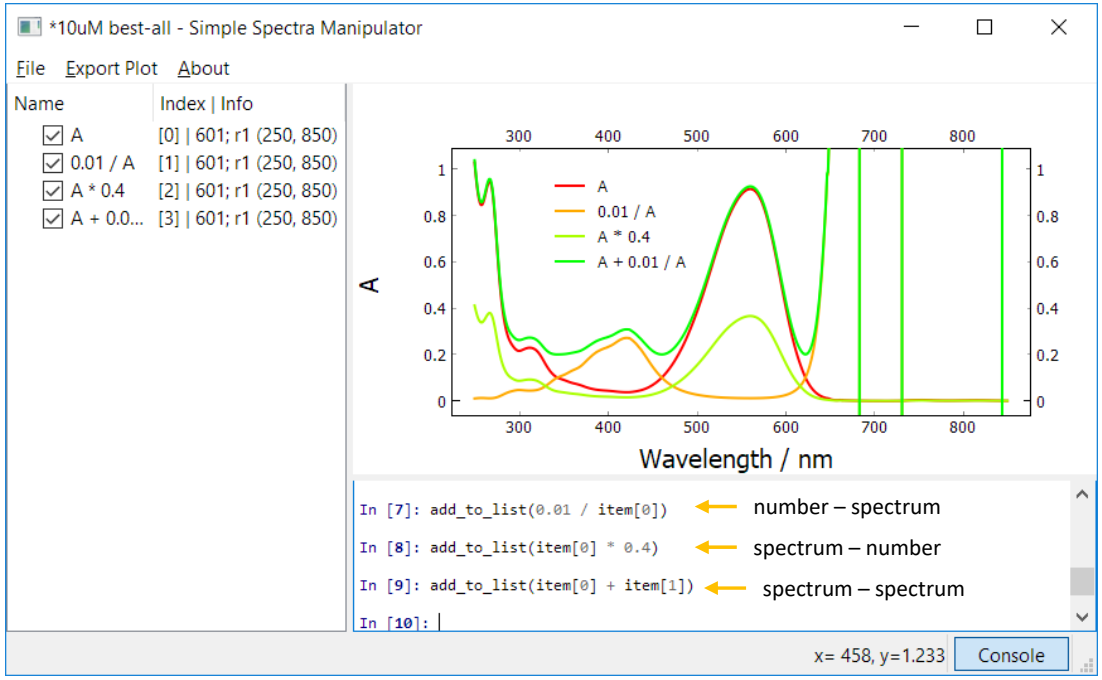


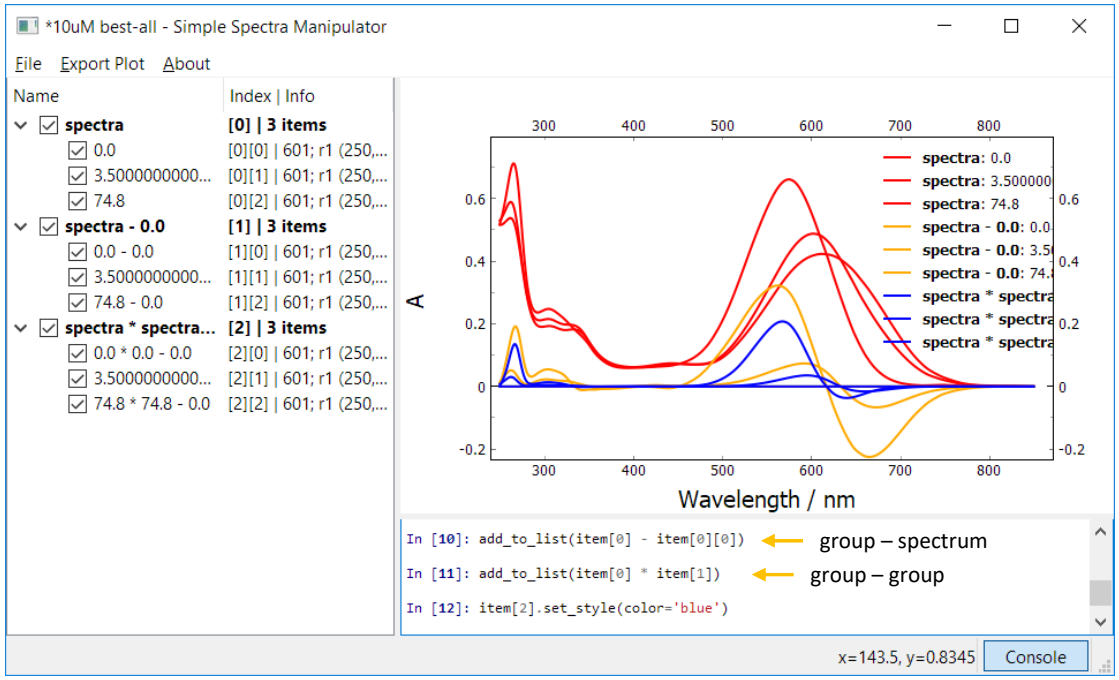
When copying data from Excel, Excel saves data into clipboard in many formats, text and XML raw data format among others. Data in text are saved as they are seen in Excel (when rounding is on, values may be saved without full decimal precision). To import data copied from Excel with full decimal precision, raw Excel XML data has to be parsed. Therefore, for full decimal precision, keep this checkbox unchecked.

Console and simple computations

- The main reason for having a console in the program is to provide interface to fast arithmetic operations with spectra items and all groups of items and show program messages and reports. Regular IPython console is running in the window (qtconsole widget – lot of thanks to programmers for this package), so also for a low level interaction with the running program (for experienced users, see documentation).
- When performing arithmetic operation between Spectrum – Spectrum, the **number of points** and same x range (**start and end value**) of both spectra have to be the same, otherwise, error is raised
- The most important is variable `item`, which is internally a root item in Tree Widget and provides an access to all members. Just write `item` and then in square bracket rewrite the index, that is in the info column for each Item/Group. When performing any arithmetic operation (addition, subtraction, multiplication, division, negation, power) among number/spectrum/group, the new Spectrum or SpectrumList object is always returned as a result of computation. So, nothing is happening with the original data. The y values are used in calculations. To add the result to the Tree Widget, you have to call function `add_to_list(result)` which takes result of computation as an argument. Examples are given below:

- There is a bug in the console, **don't ever write `type()`** to the console, because program will crash... It is bug in qtconsole or related packages...

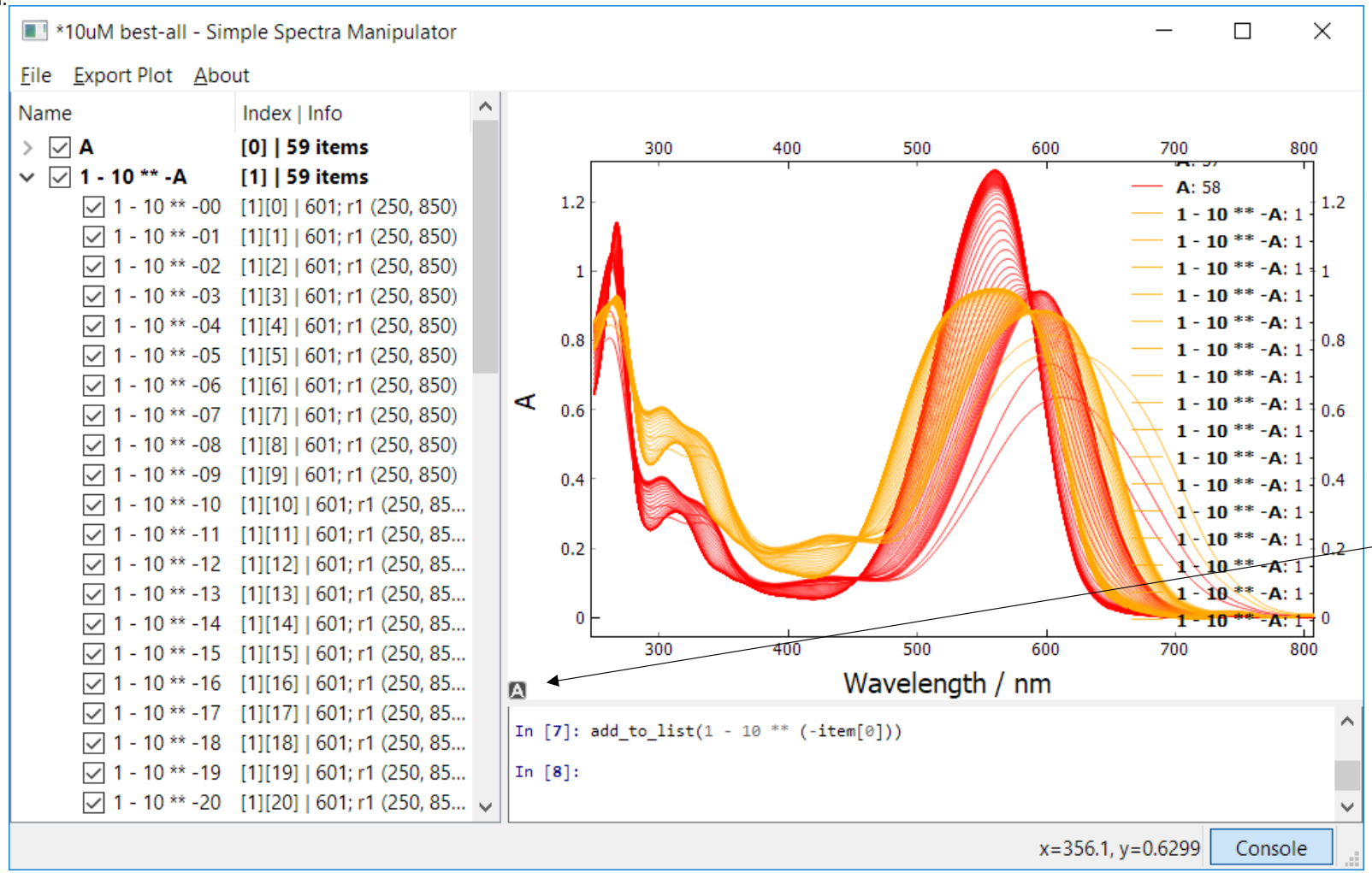




- The main power of the Console comes with the use of defined functions in class Spectrum / SpectrumList (see documentation) like integration, differentiation, savitsky-golay filter, fourier transform and others for the groups of spectra. It is a time and life saver, because doing this work in Excel/Origin would take forever. Who would want to integrate thousands of emission spectra in Excel by trapezoid method? I don't know anyone... Here, just with one line of code.

Console and simple computations

- More complicated transformations, eg. $1 - 10^{-A}$ (fraction of absorbed light), can be performed as easily as simple ones. ** is operator for power in Python. Here, the degree of abstraction is pretty significant. We are performing operation on a group of spectra. The operation $1 - 10^{-A}$ is performed for every y value of each spectrum in the group, in total 601 * 59 calculations (actually many more, because this operation is splitted internally into 3 separate operations – negation, then power and lastly subtraction). In Excel, we would have to make a matrix of 601 x 59 absorbances and then perform operation on each cell.



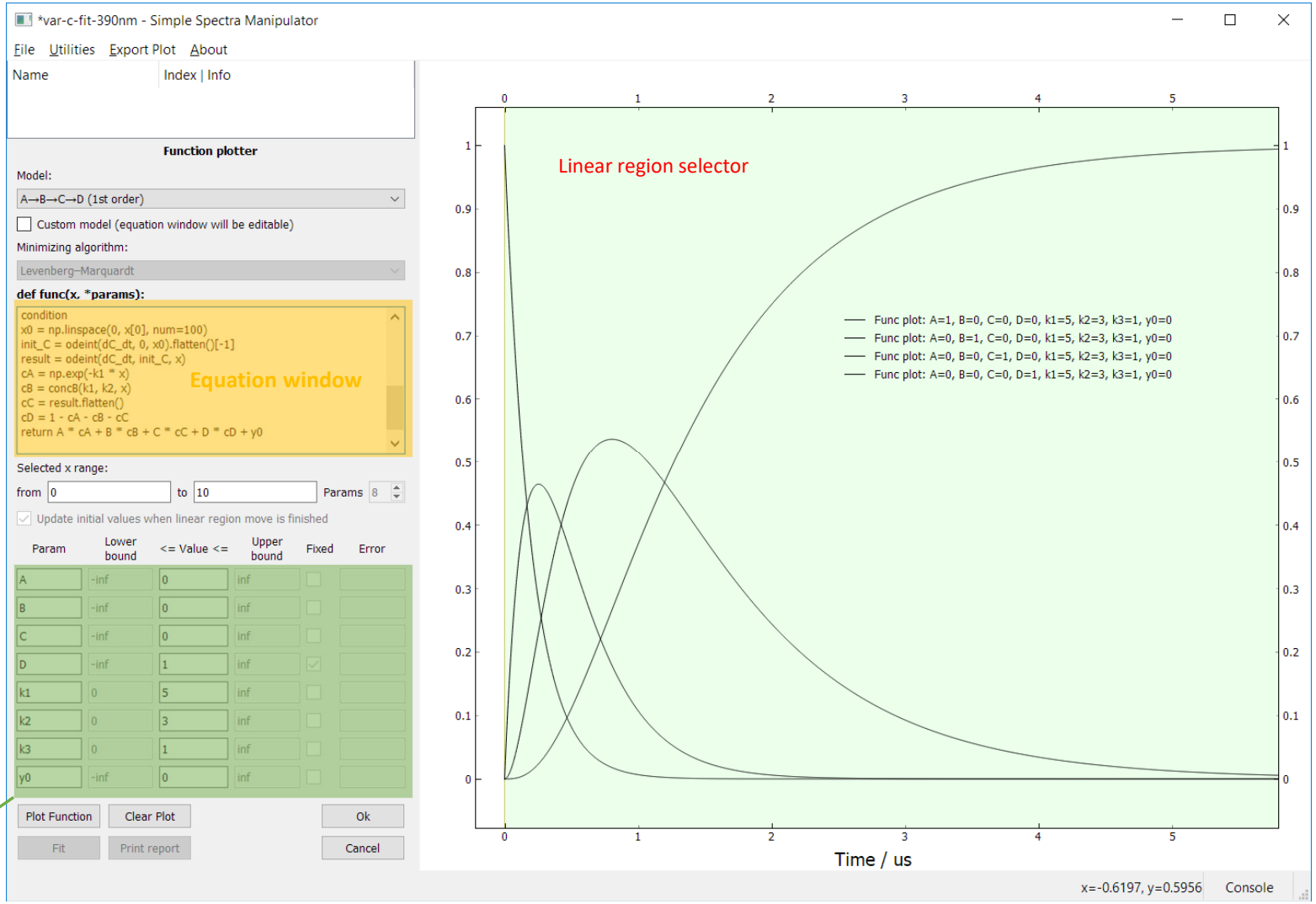
this is for **autoscale**, by clicking this, the autoscale will be ON till you manually change the scale or move with the plot

Function plotter

- By going to Utilities in Menu bar a clicking on the Function plotter, the widget on the left side should appear. This widget is designed for fitting, but can be used also for plotting custom or predefined models. Just select a model, adjust the x range by manipulating the linear region selector in graph or in the widget will be saved as items by changing the values and pressing Tab or Enter to update the plot. Then fill the parameter values and click Plot Function. You can change parameters many times and see how it looks. Then click OK and all plotted functions will be saved in the Tree Widget. The example below is plot for each component in ABCD 1st oder model.

- The analytical solution for ABCD model is too complicated, therefore, the result is calculated by numerical integration of differential equations (only for component C with function odeint from scipy module)

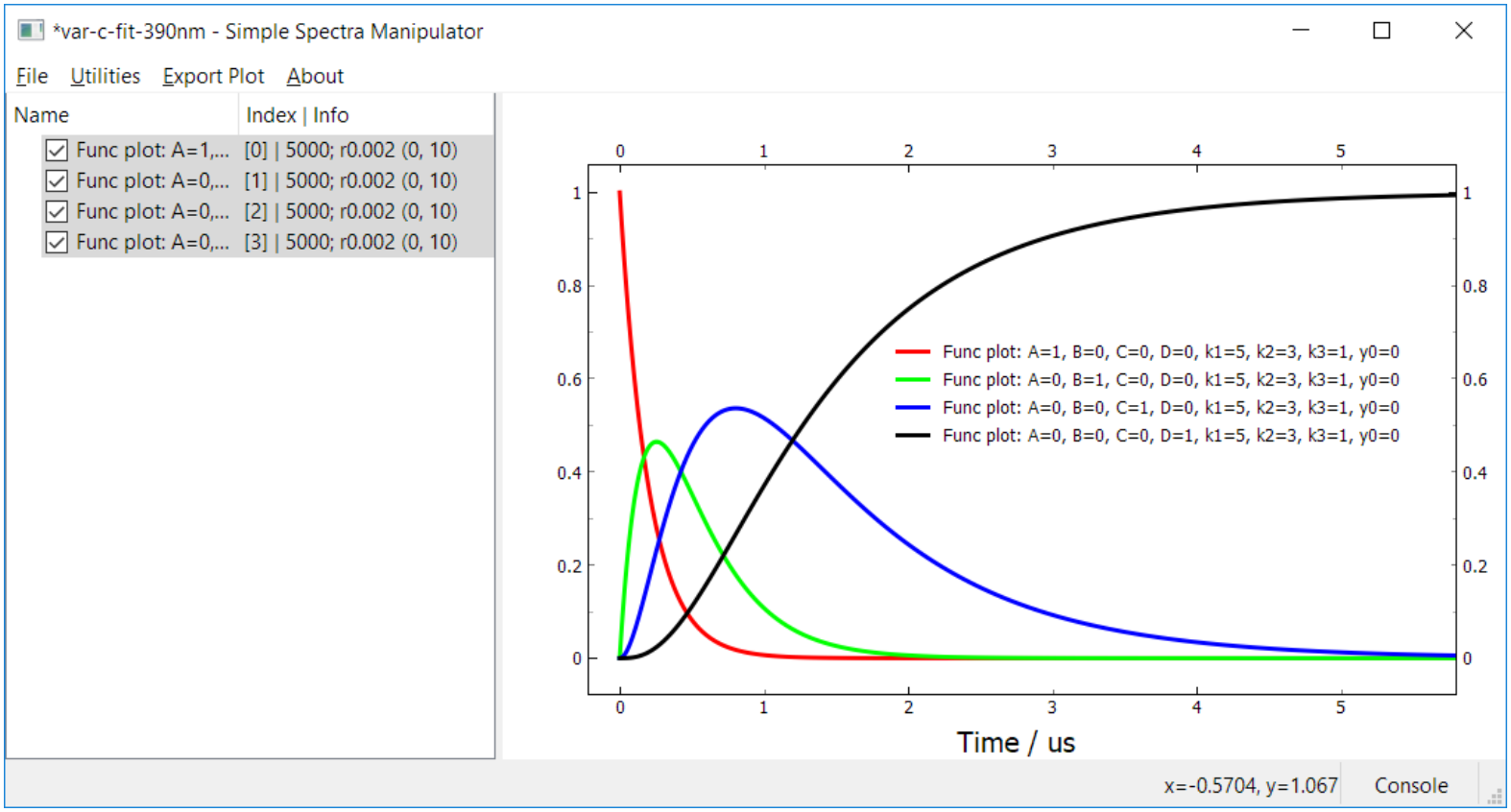
- **Change parameters A, B, C, D to modifiy mixing of the components**



Parameters

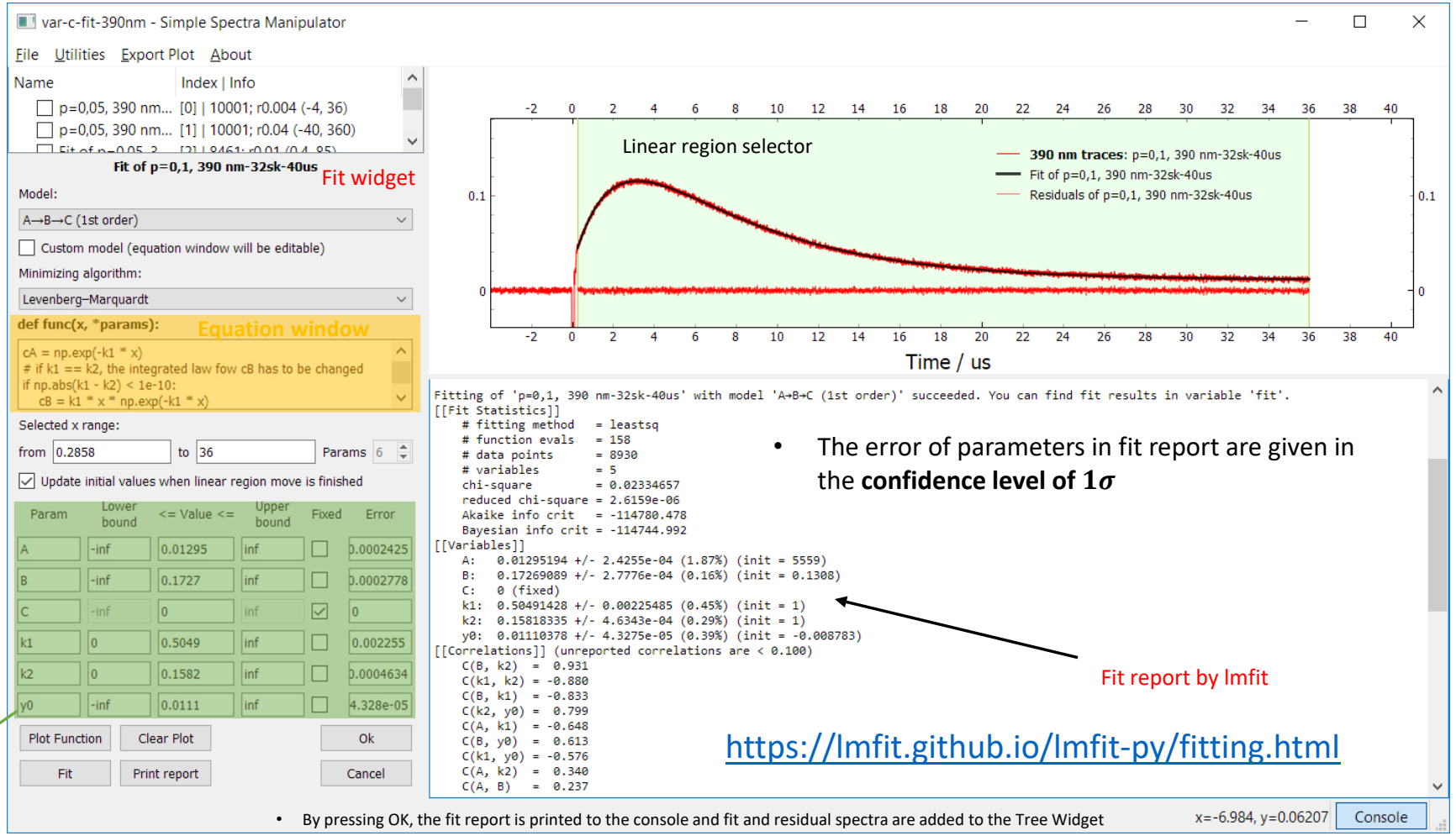
Function plotter and Export of the plot

- After clicking OK, these plots will appear in the Tree Widget. The number of plotted points is set up to **5000**, you can change this number in **config.json** file under the „**`FP_num_of_points`**” variable (or to change the default value, change `FP_num_of_points` variable in settings.py). Then, some beautiful color scheme is selected in settings along with the line width and the result is the plot below. The red curve corresponds to concentration profile of component A, the green one to component B, blue to component C and finally black curve to component D. The plot then can be copied to clipboard as image in **Export Plot-> Copy As Image** or as vector format SVG with **Export Plot-> Copy As SVG.** To be able to paste the vector SVG format to PowerPoint, one trick has to be made, because direct paste to PowerPoint sometime leads to cropped image. So, you have to first paste it in some free vector editing program (eg. Inkscape) and then re-copy from it and then paste to PowerPoint. Now, It will work.



Fit curve

- By right clicking on the spectrum (works only for 1 spectrum) and choosing from the menu: **With Selected Items -> Curve Fit**, the Fit widget will show. There you can choose from many predefined kinetic models. Each model encompasses all possible intermediates, so for example A->B->C model has parameters A, B, C which are the maximum absorbances (magnitudes) of that species. To make any component not visible (not in the system), just fix the value of this component to zero. Below is an example of fitting with such a model, with A and B components visible, but C not. Also, you can choose from many minimizing algorithms, most known and used is **Lavenberg-Marquardt** (LM), but also others, like Trust region reflective method or algorithms for scalar functions minimization (does not provide error), eg. **Nelder-Mead** Simplex method, which is slower, but more robust and tends to find global minimum in parameter space. LM algorithm is prone to stay in local minima, so, for fitting complicated models, it can be better to first find global minimum by Nelder-Mead and the switch to LM to calculate errors. Of course, sometimes, we want to stuck in local minima, because the parameters obtained from global better fit would not make any sense. The parameter boundaries can be limited by setting up lower and upper bounds. Also, the fitting result depends on initial parameters (noticeable for more complicated models).



- The error of parameters in fit report are given in the **confidence level of 1σ**

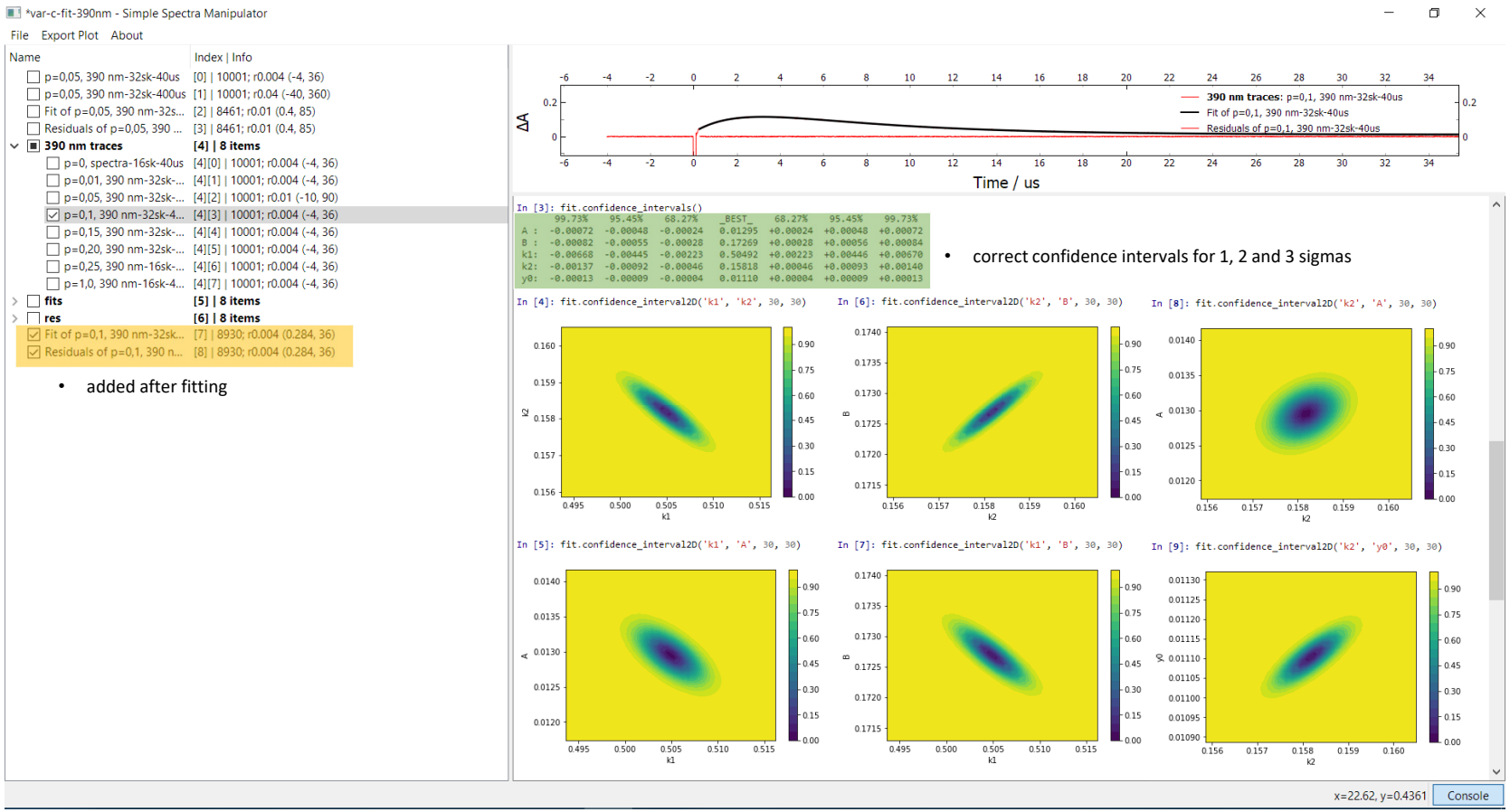
- for quick fitting of own function, check custom model checkbox and edit the python code (np – numpy) and parameter names
- for future and multiple usage of own model, define new model in **`userfitmodels.py`** (make new class that inherits from `_Model`: just uncomment the example and change it, modify *func*, *init_params* and *initialize_values* (optional) methods and change the static *name* variable), parameter order in *func* (after x) must be in the order of added paramaters in *init_params* function

- By pressing OK, the fit report is printed to the console and fit and residual spectra are added to the Tree Widget

Fit results and confidence intervals

see https://lmfit.github.io/lmfit-py/confidence.html

- The error of parameters in fit report are given in the confidence level of 1σ. That is 68.27% probability that the value lies in the interval $(\bar{x}-\sigma, \bar{x}+\sigma)$, where $\bar{x}$ is the average value. Errors are calculated from the covariance matrix. For modern and correct estimation of confidence intervals, we can use the functions in lmfit package (https://lmfit.github.io/lmfit-py/confidence.html). Read for more information, but in sumary, in some models (eg. sum of exponentials), estimation of parameter errors from covariance matrix could be wrong. Lmfit package provides function to calculate confidence intervals for these parameters correctly. Also, 2D confidence plot for two parameters is possible. The calculation takes some time, because the parameters are varied and others reoptimized by fitting. This can be really computationally demanding for complicated models.
- variable `fit` is pushed to the console after fitting and is of type FitResult (API documentation). This variable has some members, the most important is `values_errors`, which stores values and errors of fitted parameters as ndarray. Data can be saved to clipboard by typing to the console this command: `copy_to_clipboard(fit.values_errors)` and then passed to Excel for example. First column contain params values, the second corresponding errors.
- also, one can call `confidence_intervals()`, which calculates confidence intervals in a correct way, in the example below, which was the A->B->C model fit, confidence intervals for 3 sigmas (default) are given for all parameters
  - parameters are symmetrical of both sides, therefore, the analysis would not be necessary, errors from fit report are correct, but for some other models, this could not be the case
  - correlation between parameters can be also plotted by calling `confidence_interval2D()` function, see API documentation for parameter descriptions
  - we can see elipses, which is correct, but for other models, which does not behave normally, other shapes can be observes, also, if too complicated model is chosen, more minima can be observed



- added after fitting
- correct confidence intervals for 1, 2 and 3 sigmas

Example: Photodecomposition QY calculation

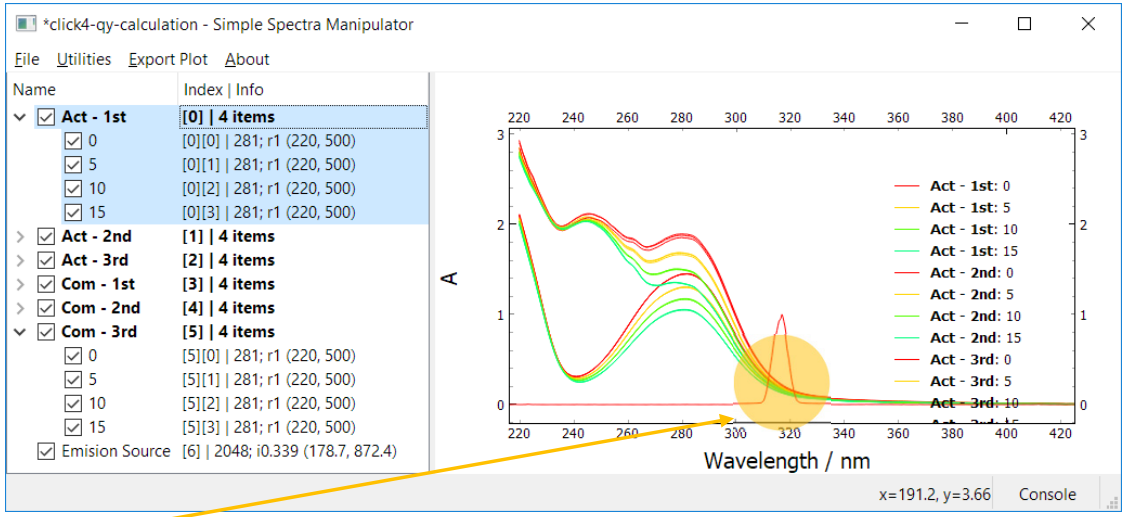
- In this example, I show how to use this program to calculate the quantum yield of photodecomposition of measured compound based on the known photodecomposition quantum yield of an actinometer

- First, the light source was established, it was Hg(Xe) arc lamp with suitable filter (here ~320 ± 20 nm) and emission spectrum was measured using OceanOptics

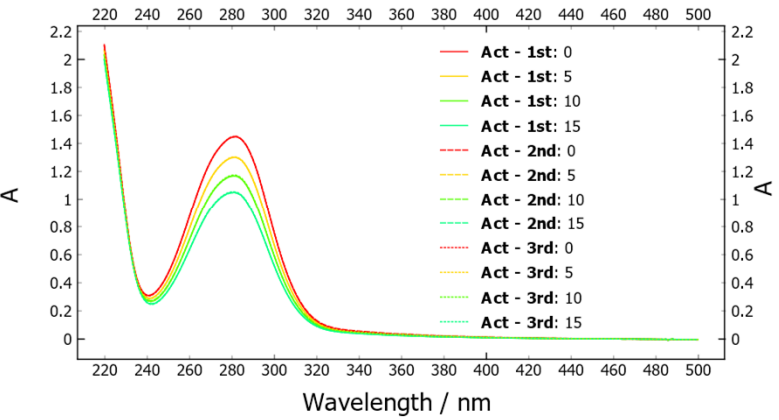
- Then, the actinometer solution in cuvette was irradiated for 5, 10 and 15 minutes and UV-VIS spectra were taken, this was then repeated 2 times which gave in total 3 sets of measured spectra, the red curves are spectra before irradiation and yellow, green and cyan are after 5, 10 and 15 **minutes** of irradiation, respectively
- The same procedure was repeated for compound whose photodecomposition quantum yield was not known

- The volumes of the cuvette were kept the same for all measurements, the initial absorbance should be similar and high (~2), here, the actinometer absorbance was a bit lower (the absorbance at irradiation wavelength is actually very small, but we could not follow the reaction by absorption spectroscopy if we wanted the absorbance at irradiation wavelength to be high)
- The showed spectra of actinometer and compound are already processed by baseline correction and cut to (220 to 500 nm) range, the emission source spectrum was baseline corrected and normalized

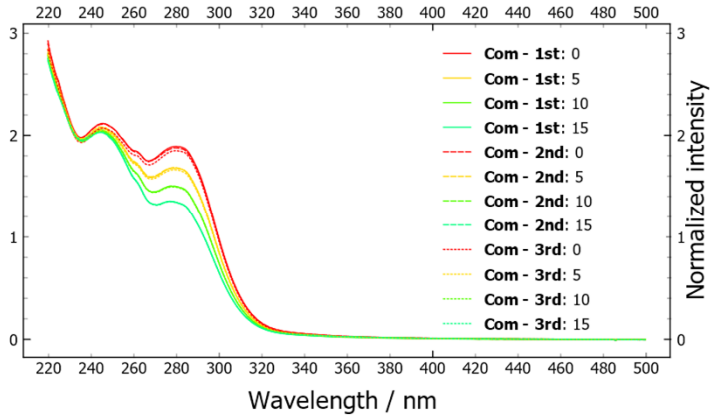
- The goal is to calculate the photodecomposition qantum yield of measured compound with the use of proper statistics and with the use of irradiation source spectrum, the correction to **unequal absorbances** of actinometer and compound at irradiation wavelength has to be used - in this region



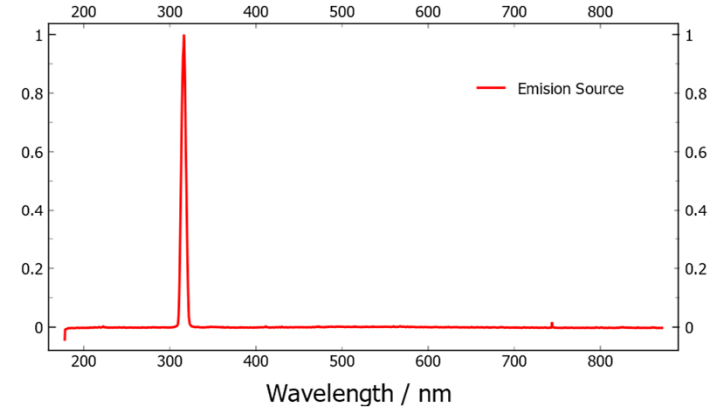
Actinometer (3 sets)



Compound (3 sets)



Irradiation source



Example: Photodecomposition QY calculation - theory

- The decrease of absorbance (concentration of compound) during photoirradiation at constant intensity is proportional to the photodecomposition quantum yield, light intensity and the fraction of the light $X_{abs}(t)$, that was absorbed by the solution at that time. The differential equation is similar to the **photobleaching** model (**irradiated and plotted wavelength are the same**), found in Documentation under the fit models.

$$\frac{\mathrm{d}c(t)}{\mathrm{d}t} = -\frac{1}{V} I_0 \Phi \times X_{abs}(t)$$

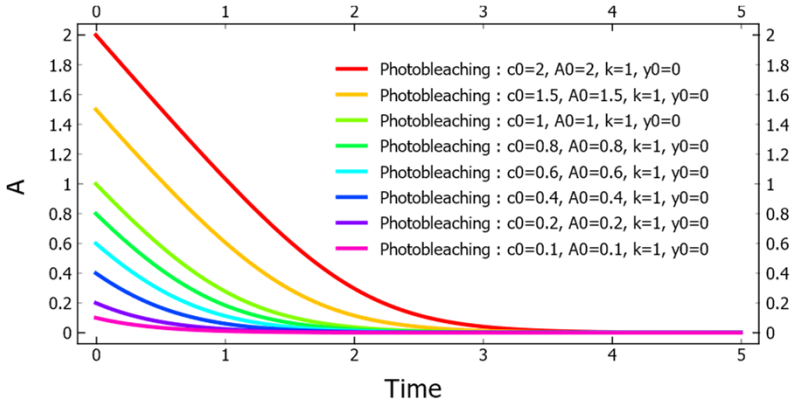
$$X_{abs}(t) = 1 - 10^{-A(t,\lambda_{irr})}$$

$$X_{abs}(t) \propto \int_{\lambda_0}^{\lambda_1} I_{source}(\lambda) \left(1 - 10^{-A(t,\lambda)}\right) \mathrm{d}\lambda$$

➢ irradiation at single wavelength

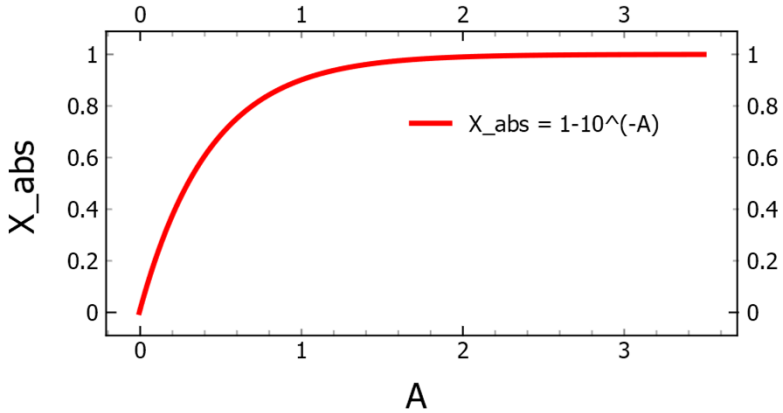
➢ irradiation with broad emission intensity**

- If we would solve the differential equation for the simple case (irradiation at single wavelength, photobleaching model), it is obvious that at higher initial absorbances (A > 2), the fraction of absorbed light is close to 1 and the value is 'constant' with changing of absorbance, as a result, the rate of decrease is almost constant, because all light is absorbed, the result is linear decrease
- However, with lower absorbances, the fraction of absorbed light decreases and the rate of decay slows down, which causes changing of the **linear decay** into **first order decay** (in the limit that initial absorbance goes to zero)



- So we can approximate the differential equation by changing the infinitesimal changes to discrete changes
- By combining the equations, we get rid of the constant light intensity and volume of the solution, that was kept the same for both actinometer and compound
- The quantum yield for actionometer is known, and the only unknown in the equation is now the photodecomposition quantum yield of compound (concentration can be calculated from molar abs. coefficients or from initial concentration, which are known)

Dependence of the fraction of absorbed light on the absorbance of the solution



actinometer

$$\frac{\Delta c_{act}}{\Delta t_{act}} \approx -\frac{1}{V} I_0 \Phi_{act} \times X_{abs,act}(t_{act})$$

compound

$$\frac{\Delta c_{com}}{\Delta t_{com}} \approx -\frac{1}{V} I_0 \Phi_{com} \times X_{abs,com}(t_{com})$$

$$\frac{\Delta c_{act} \Delta t_{com}}{\Delta c_{com} \Delta t_{act}} = \frac{\Phi_{act}}{\Phi_{com}} \times \frac{X_{abs,act}(t_{act})}{X_{abs,com}(t_{com})} \Rightarrow \Phi_{com} = \Phi_{act} \times \frac{\Delta c_{com} \Delta t_{act}}{\Delta c_{act} \Delta t_{com}} \times \frac{X_{abs,act}(t_{act})}{X_{abs,com}(t_{com})}$$

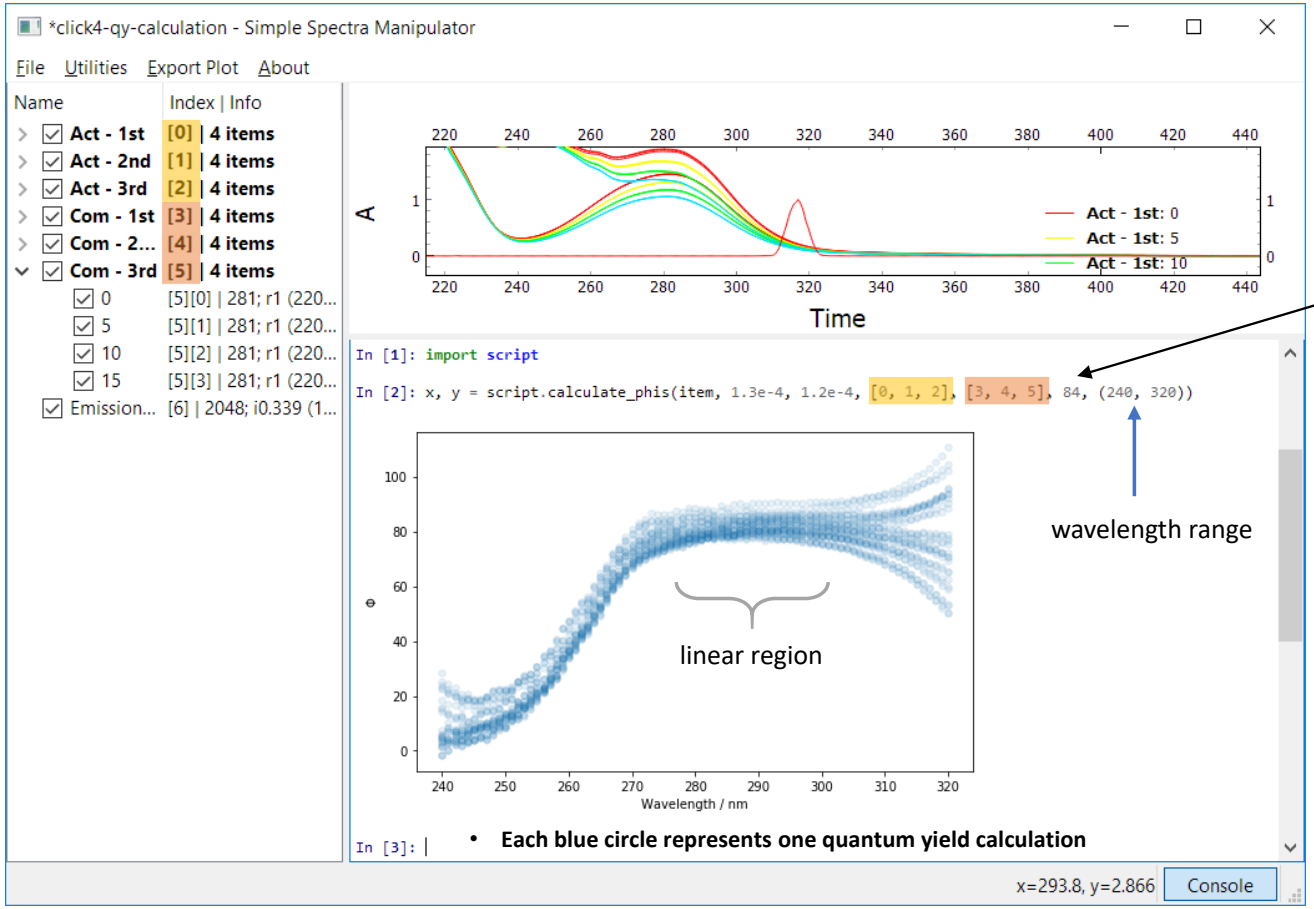
***The part of absorbed light must be a number in the interval of (0, 1), so it has to be normalized, however, because we actually need to calculate ratio of these fractions for actinometer and measured compound, normalization factor would cancel out, so we don't have to compute this factor...*

$$X_{abs}(t) = \frac{1}{\int_{\lambda_0}^{\lambda_1} I_{source}(\lambda)\,\mathrm{d}\lambda} \int_{\lambda_0}^{\lambda_1} I_{source}(\lambda) \left(1 - 10^{-A(t,\lambda)}\right) \mathrm{d}\lambda$$

Example: Photodecomposition QY calculation - uncorrected

- There is number of ways how to perform the calculation, but we will focus on the easier one, only with the use of **closest combinations**. It means, that we will combine first difference of actinometer with first difference of the compound, the second difference of actinometer with the second difference of the compound and so on... If we take two sets (each has 4 measurement), we only make calculation 3 times. We will however combine all sets, so that is in total 9 combinations (3x3). So, for one wavelength, we get in total 9x3=27 results. We can perform this calculation for every wavelength and plot the 2D map. Then, we can pick the quantum yields in a range of wavelengths, where it looks linear and perform statistics and error estimation from the results.

- Now, we have to write python code that will perform the calculation, lets first try it without correction. The code was written to file cript.py and was placed into folder of SSM



$$c_{act} = 1.3 \times 10^{-4}\ \mathrm{M}$$
$$c_{com} = 1.2 \times 10^{-4}\ \mathrm{M}$$
$$\Phi_{act} = 84\ \%$$

- The changes in concentrations are computed from changes in absorbances according to Beer-Lamber law, A_0 is absorbance before irradiation

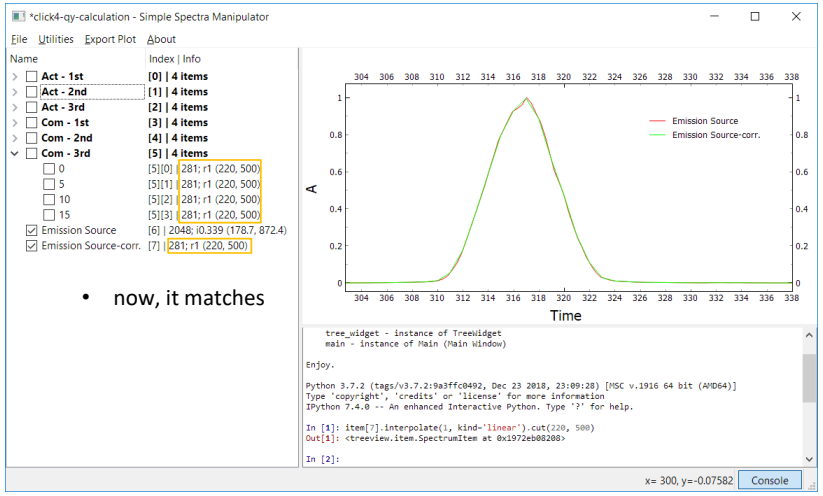
$$\Delta c = \frac{\Delta A}{\epsilon l} \qquad \epsilon l = \frac{A_0}{c_0}$$

$$\Delta c = \Delta A \times \frac{c_0}{A_0}$$

- Now, we should incorporate the correction to unequal absorbances

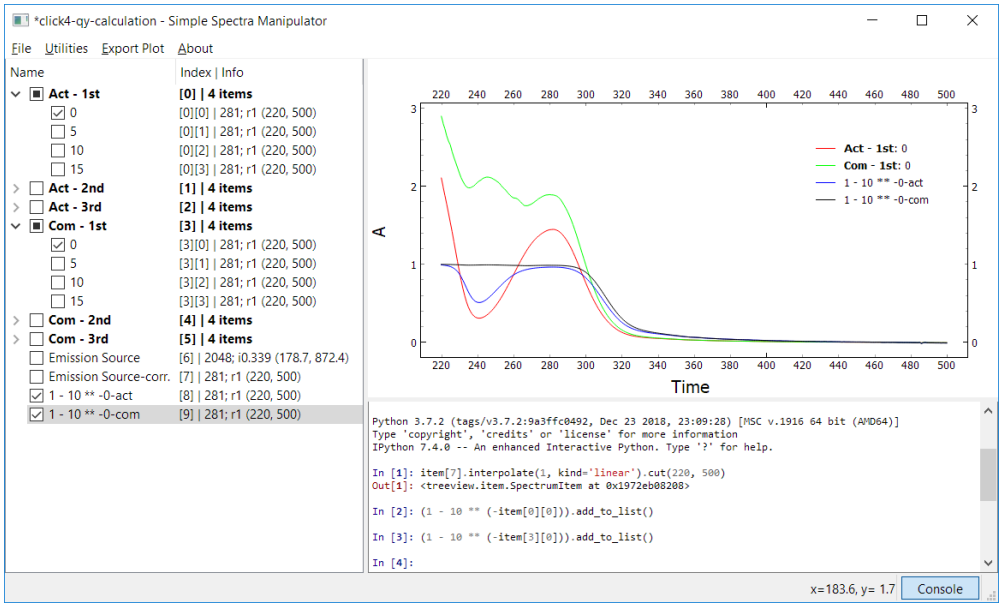
Example: Photodecomposition QY calculation - correction

- First, we have to further process the emission source spectrum, because the points are unevenly spaced and the x range does not match with the spectra of actinometer and compound, so we would not be able to calculate any integrals... So we can do it manually, or by just one line of code in the console. I made a copy of the emission spectrum, renamed it and performed the computation. The result i apparent.

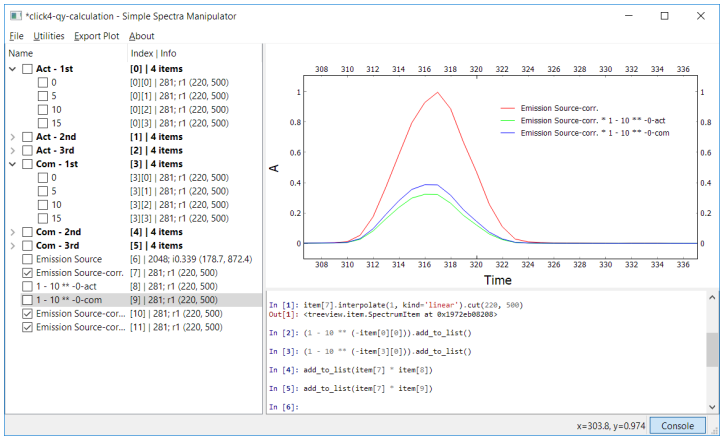


- now, it matches

- Now, we can try to calculate how much light is gonna be absorbed by for example first stock solutions of compond and actinometer, the black and blue spectra are the result after perfoming $1 - 10^{-A}$ transformations for compound and actinometer, respectively



- We can then try to calculate the actual light intensity, that would be abosrbed by the solution (blue and green spectra), and we can see, that less than half of the light was absorbed



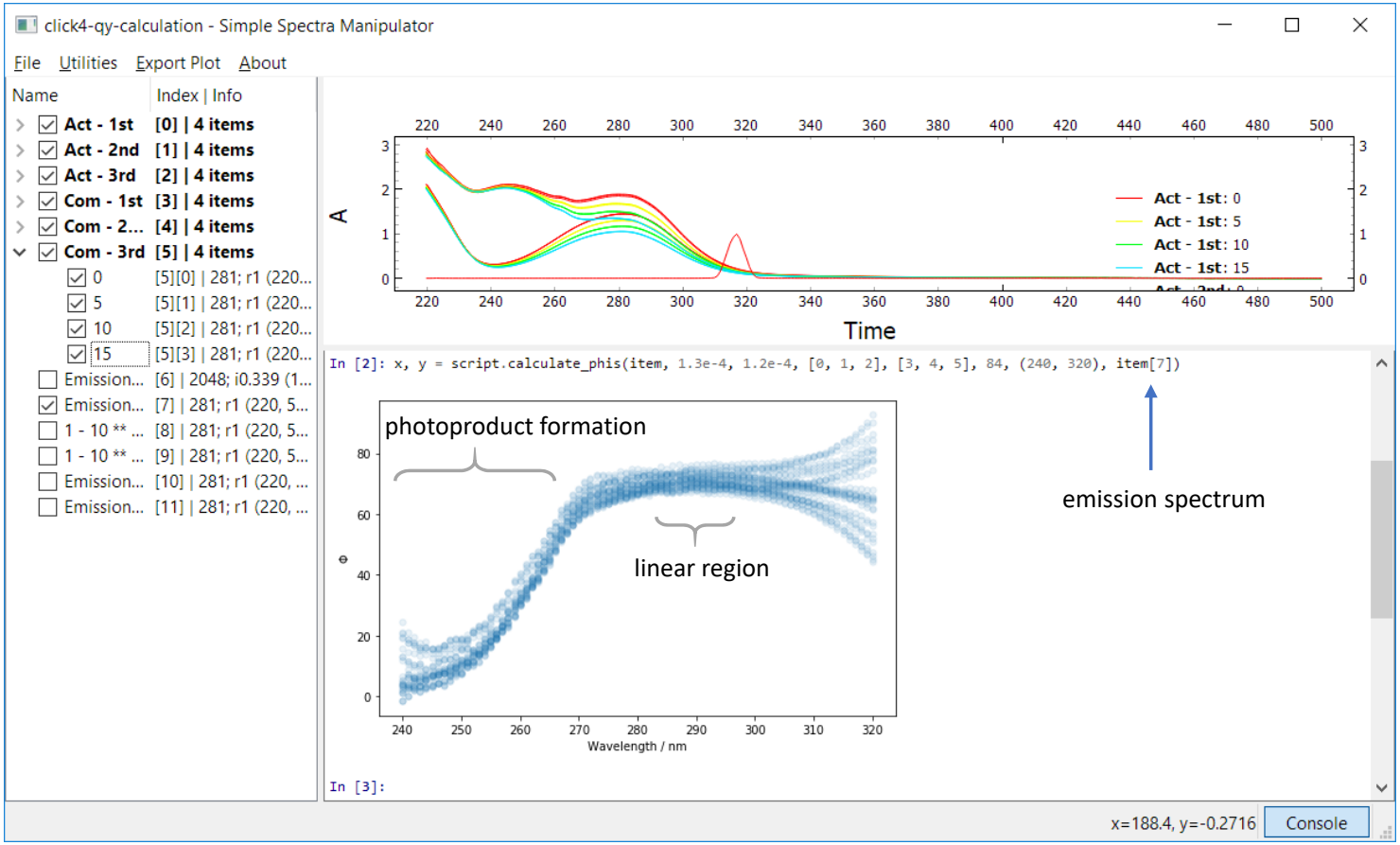
- We can calculate it precisesly by integrating the resulting spectrum and normalizing by normalization factor (integral of emission source)
- So, by calling `integral()` function without any arguments, the whole spectrum is integrated (here from 220 to 500 nm) by trapezoid integration method. The actinometer and compound actually absorbed 33 % and 39 % light from emission source, respectively

$$X_{abs}(t) = \frac{1}{\int_{\lambda_0}^{\lambda_1} I_{source}(\lambda)\,\mathrm{d}\lambda} \int_{\lambda_0}^{\lambda_1} I_{source}(\lambda)\left(1 - 10^{-A(t,\lambda)}\right)\,\mathrm{d}\lambda$$

```
In [4]: add_to_list(item[7] * item[8])

In [5]: add_to_list(item[7] * item[9])

In [6]: norm_factor = item[7].integral()

In [7]: X_abs_act, X_abx_com = item[10].integral()/norm_factor, item[11].integral()/norm_factor

In [8]: X_abs_act, X_abx_com
Out[8]: (0.3284490163431788, 0.3920187209579002)

In [9]:
```

Example: Photodecomposition QY calculation - corrected

- Now, we can automate this and add this to our script and perform the calculation again, but now with the correction. We can see, that the linear region part was decreased a little bit (from cca 80 to 70). This make sense, because the amount of absorbed light by **compound** was a bit **higher** than that of actinometer, so more of it was bleached. If the changes of concentrations are the same, quantum yield of compound photobleching must be lower when compared to that of uncorrected result.



- The linar region seems to be around 285 to 295 nm.
- Before this region ($\lambda < 270\ nm$), there is a formation of **photoproduct** from the measured compound, therefore, the quantum yield values are lower.
- After this region ($\lambda > 300\ nm$), both actinometer and spectrum are not absorbing anymore, therefore, very **large error** is obvious in the the result.

Example: Photodecomposition QY calculation – statistics on 285-295 nm region

- Now, we can pick the linear region, average the values in it and calculate the standard deviation. For that, we crop x and y variables that we got from the calculation. The x variable represent the wavelengths and y value the quantum yields at these wavelengths. Both object are NumPy 2D arrays.

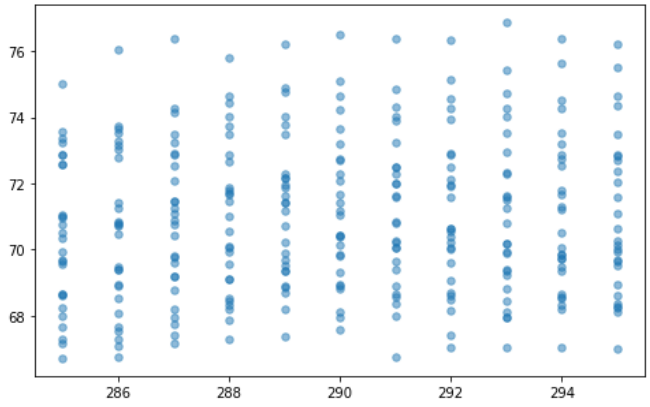
```
In [12]: x[0]
Out[12]:
array([240., 241., 242., 243., 244., 245., 246., 247., 248., 249., 250.,
       251., 252., 253., 254., 255., 256., 257., 258., 259., 260., 261.,
       262., 263., 264., 265., 266., 267., 268., 269., 270., 271., 272.,
       273., 274., 275., 276., 277., 278., 279., 280., 281., 282., 283.,
       284., 285., 286., 287., 288., 289., 290., 291., 292., 293., 294.,
       295., 296., 297., 298., 299., 300., 301., 302., 303., 304., 305.,
       306., 307., 308., 309., 310., 311., 312., 313., 314., 315., 316.,
       317., 318., 319., 320.])

In [13]: start_idx, end_idx = 285 - 240, 295 - 240

In [14]: start_idx, end_idx
Out[14]: (45, 55)

In [15]: x = x[:, start_idx:end_idx+1]
```

crop

```
In [16]: y = y[:, start_idx:end_idx+1]

In [17]: y.shape
Out[17]: (27, 11)
```

statistics from 27*11 = 297 values

```
In [18]: plt.scatter(x, y, s=30, alpha=0.5)
Out[18]: <matplotlib.collections.PathCollection at 0x16e1286a438>
```

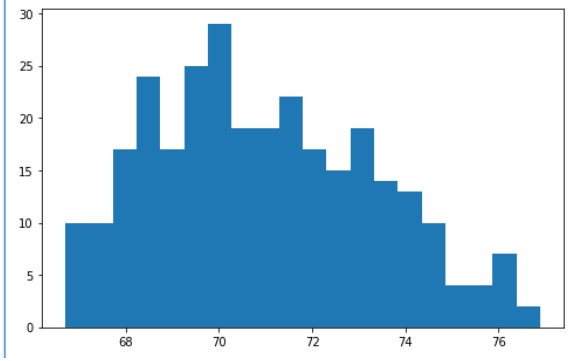



```
In [19]: avrg, stdev = np.average(y), np.std(y)

In [20]: avrg, stdev
Out[20]: (70.98725947331127, 2.3565530300937145)
```

- So, we can conlude, that the photodecomposition quantum yield is $\Phi_{com} = \mathbf{71 \pm 2}$

- Also, we can plot histogram from these 297 values to see, whether we can use standard deviation estimation, because it should be used only if the data are normally distributed. Well, it seems more or less OK, but more results would be better for proper estimation.

```
In [27]: plt.hist(y.flatten(), bins=20)
Out[27]:
(array([10., 10., 17., 24., 17., 25., 29., 19., 19., 22., 17., 15., 19.,
        14., 13., 10.,  4.,  4.,  7.,  2.]),
 array([66.70384466, 67.21353199, 67.72321932, 68.23290666, 68.74259399,
        69.25228132, 69.76196865, 70.27165599, 70.78134332, 71.29103065,
        71.80071799, 72.31040532, 72.82009265, 73.32977998, 73.83946732,
        74.34915465, 74.85884198, 75.36852931, 75.87821665, 76.38790398,
        76.89759131]),
 <a list of 20 Patch objects>)
```



- Stdev is calculated by function `np.std()` in the statistical level of one sigma

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \quad \text{where} \quad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html

- Some comment regarding the script design is needed. Here, the QY values were calculated by using closest combination, however, we could combine the two sets differently, eg. by combining each difference in one set with each difference in other set. We would get more points. Or we could combine all possible differences from one set with all possible differences with other set. If the set would contain 4 spectra, we can choose max 6 differences. If we combined two sets, we would get 6*6=36 results (instead of 3 as in this case).
- Also, the correction here is only approximation, it can be done better, but the code would be much more complicated
- **Possible code modifications are left as an excercise for the reader**

Example: Photodecomposition QY calculation

- **Source code from script.py**

```python
import numpy as np
import matplotlib.pyplot as plt
from spectrum import Spectrum


def calculate_phis(item, c_0_act, c_0_com, act_idxs, com_idxs, phi_act, wl_range=(240, 330), em_source=None):
    sp = item[act_idxs[0]][0]

    idx_0 = Spectrum.find_nearest_idx(sp.data[:, 0], wl_range[0])
    idx_1 = Spectrum.find_nearest_idx(sp.data[:, 0], wl_range[1]) + 1

    # hardcoded number of solutions, this is not general, for other use, please change this number or rewrite the code
    solutions = 27
    wls = idx_1 - idx_0  # number of wavelengths
    wls_mat = np.tile(sp.data[idx_0:idx_1, 0], (solutions, 1))  # define matrix of wavelengths (x values for plotting)
    matrix = np.zeros((solutions, wls))  # define matrix of results, dimensions: 27 x [number of wavelengths]

    # for each wavelength in defined range
    for i in range(wls):
        # get the current wavelength
        wl = sp.data[idx_0 + i, 0]

        # calculate the 27 values at this wavelength
        phis = calculate_phi(item, c_0_act, c_0_com, act_idxs, com_idxs, phi_act, wl=wl, em_source=em_source)

        # fill the i-th column of the matrix with the solution
        matrix[:, i] = np.asarray(phis)

    # plot the matrix, parameter s is size of the circle, alpha sets transparency
    plt.rcParams['figure.figsize'] = [8, 5]
    plt.scatter(wls_mat, matrix, s=30, alpha=0.1)
    plt.ylabel('$\Phi$')
    plt.xlabel('Wavelength / nm')
    plt.show()

    return wls_mat, matrix


def calculate_phi(item, c_0_act, c_0_com, act_idxs, com_idxs, phi_act, wl=285, em_source=None):
    # define list of results
    phis = []

    # for every actinometer set
    for act in act_idxs:
        # for every compound set
        for com in com_idxs:
            # perform calculation between current sets and add the results into list
            phis += calculate_set(item[act], item[com], c_0_act, c_0_com, phi_act, wl, em_source)

    return phis
```

QY calculation

correction

```python
def calculate_set(act_set, com_set, c_0_act, c_0_com, phi_act, wl=285, em_source=None):
    # assuming that the number spectra in actinometer and compound set is the same
    # thus, len(act_set) == len(com_set), ** both act_set and com_set are instances of SpectrumList!

    # number of items in set
    length = len(act_set)

    # find index in data array of actinometer and compound that corresponds to the choosed wavelength
    idx_act = Spectrum.find_nearest_idx(act_set[0].data[:, 0], wl)
    idx_com = Spectrum.find_nearest_idx(com_set[0].data[:, 0], wl)

    # set absorbances at time zero for both actinometer and compound at wavelength wl
    A0_act = act_set[0].data[idx_act, 1]
    A0_com = com_set[0].data[idx_com, 1]

    # calculate the ratio of c0 and A0 for actinometer and compound
    c0A0_act = c_0_act / A0_act
    c0A0_com = c_0_com / A0_com

    # define list of computed quantum yields
    phis = []

    for i in range(1, length):
        # calculate change of absorbance between two measurements for act and com
        dA_act = act_set[i - 1].data[idx_act, 1] - act_set[i].data[idx_act, 1]
        dA_com = com_set[i - 1].data[idx_com, 1] - com_set[i].data[idx_com, 1]

        # calculate corresponding change in concentrations
        dc_act = c0A0_act * dA_act
        dc_com = c0A0_com * dA_com

        # calculate the changes in times, use the value in name of the spectrum
        dt_act = float(act_set[i - 1].name) - float(act_set[i].name)
        dt_com = float(com_set[i - 1].name) - float(com_set[i].name)

        # calculate the quantum yield
        phi_com = phi_act * dc_com * dt_act / (dc_act * dt_com)

        # if user provided emission source, calculate the correction
        if em_source is not None:
            # calculate the absorbed light intensity (we use absorption spectrum
            # at the begining of the interval - i-1
            abs_int_act = em_source * (1 - 10 ** (-act_set[i - 1]))
            abs_int_com = em_source * (1 - 10 ** (-com_set[i - 1]))

            # integrate those spectra
            x_abs_act = abs_int_act.integral()
            x_abs_com = abs_int_com.integral()

            # apply the correction
            phi_com *= x_abs_act / x_abs_com

        # add the result to the list
        phis.append(phi_com)

    return phis
```