LIFSim

LIFSim

Modeling and analyzing LIF spectra of diatomic molecules

User and developer's documentation

Version 4.0

Text version: Nov. 8. 2024

Developers: Abbas El Moussawi, Torsten Endres, Jan Menser

Institute for Energy and Materials Processes – Reactive Fluids
University of Duisburg-Essen

47048 Duisburg

Germany

LIFSim

Contents

LIFSim

1 Introduction

LIFSim is a tool for simulating laser-induced fluorescence spectra of diatomic species. The version 4.0 achieves a transparent implementation of the models used through a MATLAB implementation, which is open-source. This manual target two audience groups, the first group are researchers with limited programming skills (sections 1–3). The second group are researchers with programming skills for which we provide the documentation of the functions and scripts.

1.1 Prerequisites

Download and install MATLAB 2024a. During installation you will be prompted to install further toolboxes. LIFSim uses the following:

1. Optimization Toolbox, necessary for fitting
2. Parallel Computing Toolbox, necessary for simultaneous fitting
3. Communications Toolbox, necessary for sensitivity analysis. This toolbox can be replaced by implementing your own script for white-noise generation
4. Image Processing Toolbox, for handling images prior to further processing. Not essential if the images are already processed

1.2 Downloading

Under https://github.com/LIFSim/LIFSim, click on Code and acquire the code with your preferred method.

Check regularly the link, for newer versions of this documentation or code updates.

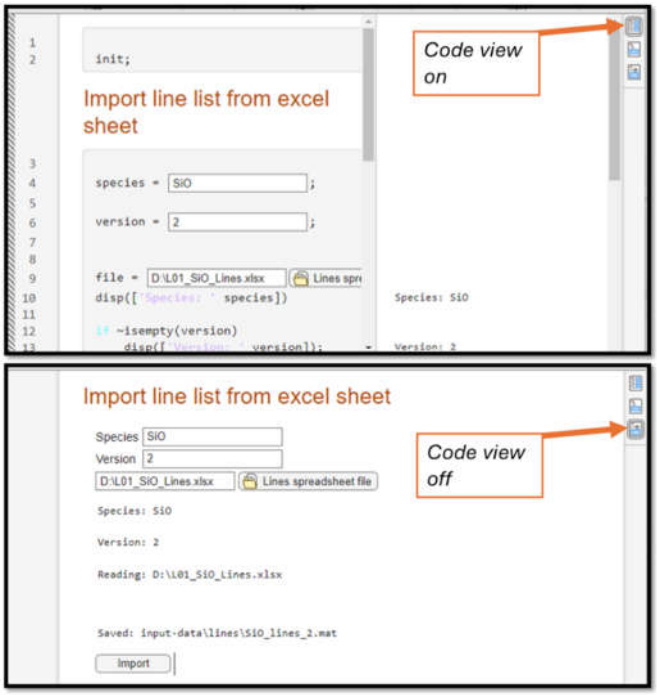
1.3 Notes on MATLAB's live scripts

MATLAB livescripts[1] are used to provide a graphical interface and reduce the interaction with the code to enable easy use. This approach encourages the scientists to interact with the code casually.

Code view

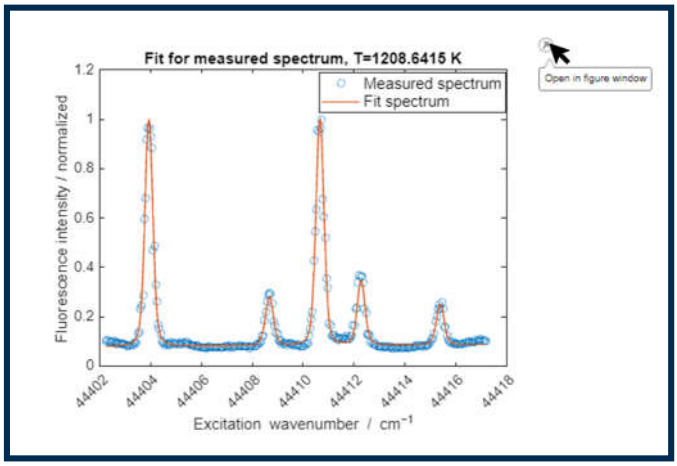
Under the folder 'livescripts', you will find the live scripts with '.mlx', with those scripts you can switch from code view (top in the next figure) to graphical view (bottom).

[1] For further information on live scripts go to https://de.math-works.com/help/matlab/matlab_prog/what-is-a-live-script-or-function.html



Pop-out live-script figures

It is possible to get a conventional MATLAB figure in an independent window. As the mouse hovers over the figure, an icon appears in the top right corner to open in figure window.



1.4 Developers' quick help

For the functions we implemented, the help dialog of MATLAB can be triggered. We embedded a description, list of inputs/outputs, and if present related functions.

To call the help dialog, as you write the name of a LIFSim function you may quickly review the description of this function. To do so, right click on the function name in your script and click help. Or while the cursor is in the function's name, click F1.

1.5 Sample data

For demonstration, the user can find sample data under the paths:

1. The NO data reported in manuscript:
   LIFSim\input-data\exampleImagedSpectra\MATLAB_out\FOR0064
2. The SiO data reported in manuscript:
   LIFSim\input-data\exampleImagedSpectra\MATLAB_out\FOR0072
3. The OH data reported in manuscript:
   LIFSim\input-data\exampleImagedSpectra\MATLAB_out\SPPQ5

Under these directory files ending with _scan.settings.mat could be used with the livescript fitExcSpectrumImage.mlx, see section 3.6.2.
The files ending with _fitResults.mat contain the fit results for a given measurement.
For SiO, the _fitResults.mat can be used to infer SiO mole fraction, qualitatively, refer to section to section 3.7 for further details.

2 Input data

This chapter describes the format of the input data for working with LIFSim, and how to store new data for working with software. The imported data will be stored in the folder structure for multiple use with live script or functions later.

2.1 Importing line lists

This section describes how to import spectroscopic data for a molecule. The data can be for a new molecule or can be a newer version for an existing molecule. This import process is not needed every time the function `selectLines`[2] is used and rather serves for updating your lists if you made changes. If new line lists are needed for further species or updated versions, use the live script 'importLines.mlx' under 'input-data\', see Figure 1.

[2] Note: In this document, a code snippet, variables from a code or function names are formatted as: `code snippet`.

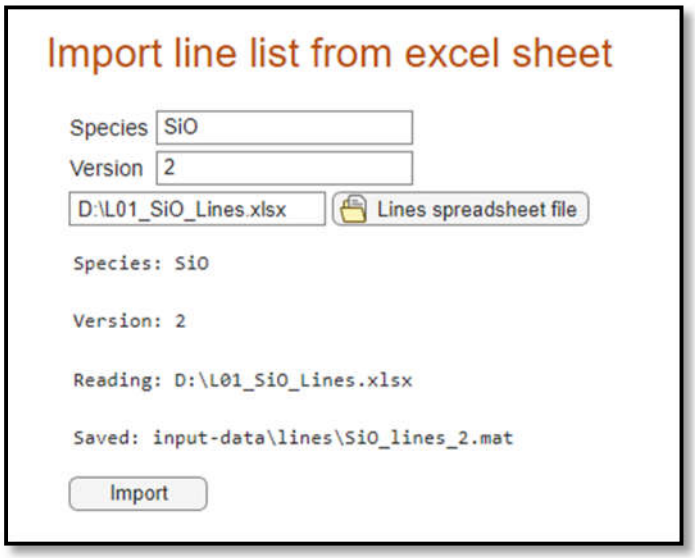


Figure 1: Example for importing an SiO line list.

1. Specify the molecular formula of your species 2.
2. If versioning is needed, specify the version[3]. Otherwise keep empty. If the list for the given species is already available and no version is entered, the file will be overwritten.
3. Specify the path to your spreadsheet; the following file formats are supported: 'XLSX', 'XLS', 'XLSM', 'XLSB', 'XLTX', 'XLTM', or 'ODS'. The spreadsheet should have the following information in the first 10 columns:
   A. Spectral band (i.e., electronic transition) / optional, keep empty if not possible
   B. j'': Lower rotational quantum number (-)
   C. j': Upper rotational quantum number (-)
   D. E_{trans}: Transition energy (cm^{-1})
   E. E_{ground}: Ground state energy (cm^{-1})
   F. A_{kj}: Einstein coefficient for spontaneous emission (s^{-1})
   G. B_{ik}: Einstein coefficient for absorption ($\text{m}^3/(\text{Js}^2)$)
   H. Predissociation rate (s^{-1}) / optional, set to zero if not known
   I. v'': Lower vibrational quantum number (-)
   J. v': Upper vibrational quantum number (-)

   Note that the first row will be omitted while importing, assuming that it is serving as the table header.
4. After clicking 'Import', the list will be saved as a '.mat' file for use with the function `selectLines`.

Note: Predissociation and ionization (P_k, and W_k) modify the function `lineStruct`, where indicated in the script. This will then be considered in the calculated LIF intensity.

[3] Numbers (0–9) or characters (A–Z or a–z)

2.2 Importing partition function lists

Like the line list, the partition function list is imported once at the initialization of this script package and no need to repeat the following process each time you use the code.

1. Specify the species of the partition function list to be imported.
2. If versioning is needed, specify the version[3]. Otherwise leave empty. If the list for the given species is already available and no version is entered, the file will be overwritten.
3. Specify the path to your spreadsheet, the following file formats are supported: 'XLSX', 'XLS', 'XLSM', 'XLSB', 'XLTX', 'XLTM', or 'ODS'. The script assumes here no headers and the columns are as follows:
   A. Temperature (K)
   B. *Z*, partition function

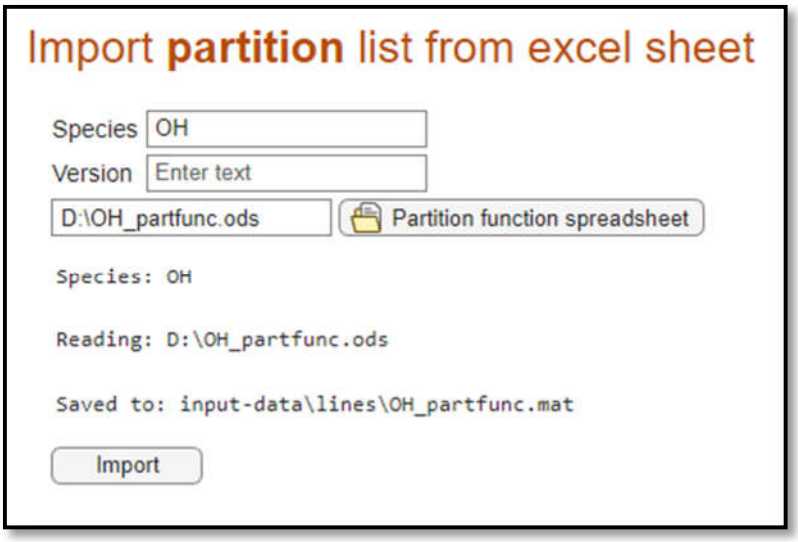


Figure 2: Example for importing the partition function list for OH.

2.3 Detection filter

In LIFSim, the transmission curve for a detection filter can be imported and used in simulation and fitting.

Save the filter transmission data in a comma-separated value file (.csv) with the wavelength or wave-numbers in the first column and the transmission value in the second column. Note that the maximum transmission is 1 and the minimum is 0.

The file can be stored under 'input-data\filters\'. This can be later read in with the functions `loadSpectrum` and `fluorTransm`, see section 4.3, or within the live scripts supporting this.

2.4 Gas compositions

For calculating quenching rates and collision-induced broadening the gas composition of a probed volume is required. Note that NO collisional model is supported, and for adding further support review function `collisions`. To save a new gas composition:

1. Create a comma-separated value file (simply an empty file with .csv extension) under the folder 'input-data\gas-compositions'

2. In the first line of the file write, the headers: `molecule,fraction`
3. In the following lines, insert the gas composition, see the supported values in review function `collisions`. E.g., csv file:

```
molecule, fraction
CH4, 0.545
O2, 0.455
Ar, 1.090
NO, 0.010
```

4. Note that you do not have to normalize your numbers, as in the above example. The function `loadGasComposition` will calculate the fractions assuming the list is complete.

3 Livescripts

For a demonstration of LIFSim 4.0 main functions, MATLAB live scripts can be found under the directory path LIFSim\livescripts. These scripts serve two purposes and target two user groups. The first use case of the live scripts is to provide a user a graphical interface to interact with the software without the need to program. In this case, the user can hide the code view (refer to section 1.3) and directly interact with the input fields only.

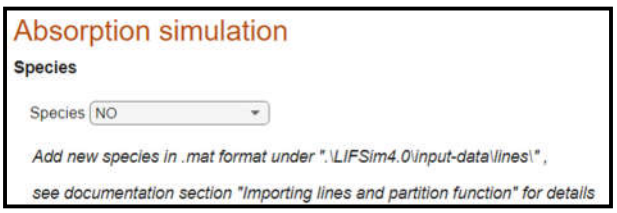
The second option addresses researchers who want to develop around LIFSim. Here, the developer can use these scripts as an example on how to use the functions.

3.1 Simulation of absorption spectra 'simulateAbsorptionSpectra.mlx'

Absorption spectra (for NO, SiO, OH, or O_2) can be simulated based on the absorbance, and this live script facilitates the use of the function in section 4.5.3. The user can choose the species, laser parameters, and set the environment parameters as in section 4.7.1. Also, the collision data can be provided through selecting a gas composition from your file system. The gas composition should be a comma-separated values file (.csv) as established in section 2.4.

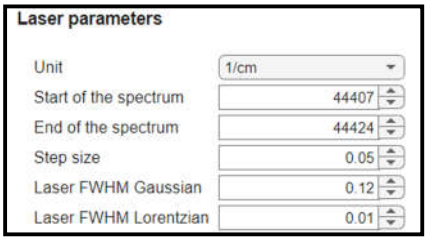
Species

The species can be selected in the drop-down menu:



Laser parameters

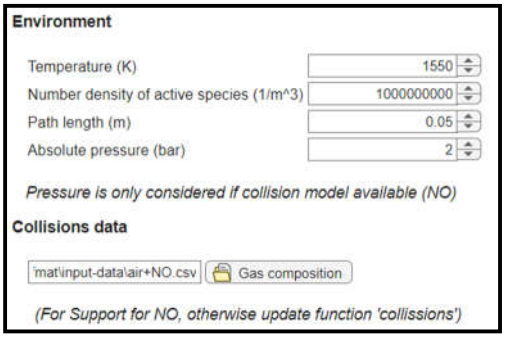
The laser parameters consist of the tuning region and the scan resolution (step size). Furthermore, the line shape can be defined through setting the FWHM of the Gaussian and Lorentzian contributions in the Voigt function. The units supported here are nm and cm^{-1}.



Environment and molecular collisions

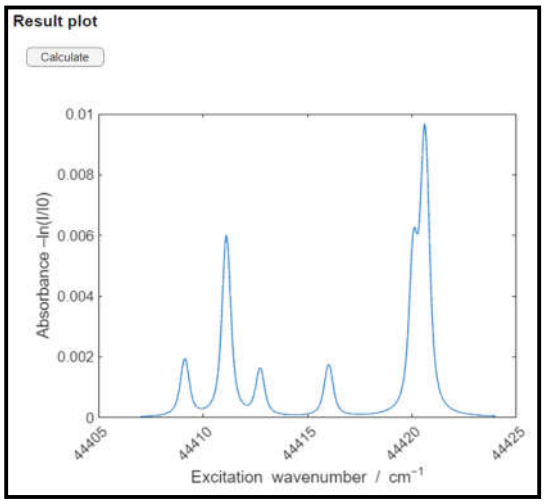
The following parameters can be set in this section: Temperature (in Kelvin), number density of active species (in m^{-3}), path length (in m), and pressure (in bar).

If the collision model is defined (see section 4.6) for the chosen species, the gas composition will be considered. Then, the collisional broadening and shift are calculated. Otherwise, for controlling the Lorentzian FWHM of the final overlap shape, the user can use the "Laser FWHM Lorentzian" parameter to do so.



Results

Clicking the "calculate" button will start the simulation and plot the absorbance.



3.2 Simulation of emission spectra 'simulateEmSpectra.mlx'

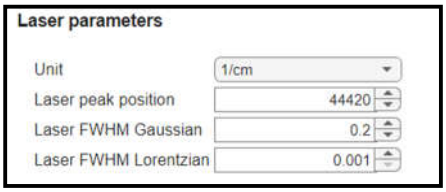
In this live script, laser-induced emission spectra (for NO, SiO, OH, or O_2) can be simulated for the available species at given laser parameters, detection parameters, and collision data (if supported).

Species

The species can be selected in the drop-down menu:

Laser parameters

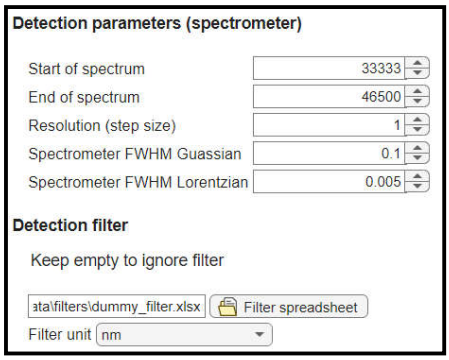
The laser parameters can be defined in the next section with the option to choose between nm and cm^{-1}. These parameters include the laser position and the laser line-shape function that can be defined through setting the full width at half maximum (FWHM) of the Gaussian and Lorentzian contributions of the Voigt function.



Detection

The next section sets the detection parameters for the spectrometer and optionally for a collection filter. The collection range of the spectrometer can be set along with its resolution (step size), and instrument line shape can be defined with the Gaussian/Lorentzian FWHMs.

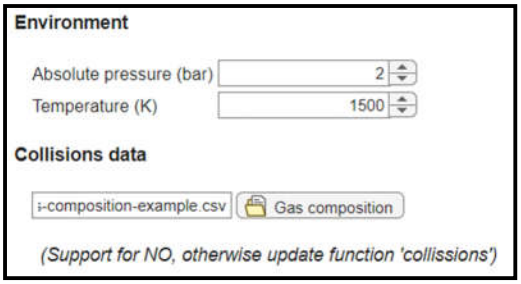
If a filter function is needed, this can be added as a spreadsheet with two columns as define in section 2.3. The unit of the spectral dimension must be defined (nm or cm^{-1}). The file path field can be left empty to disregard the filter.



Environment and collisions

Pressure (in bar) and temperature (in Kelvin) can be defined in this section. If the collision model is defined (see section 4.6) for the chosen species, the gas composition will be considered. Then, the collisional quenching, line broadening and line shift are calculated. Otherwise, these values will be

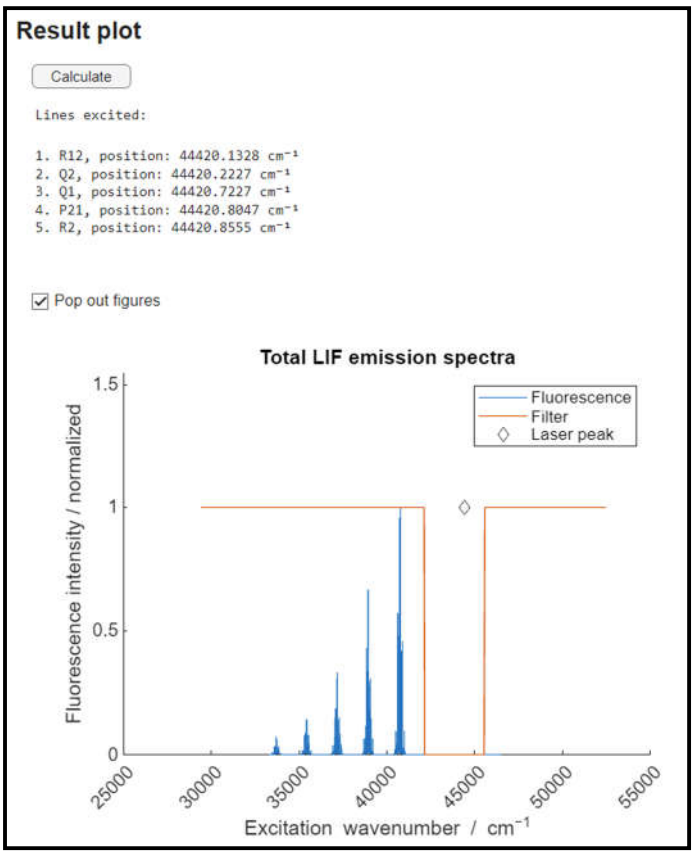
considered negligible. For manipulating the Lorentzian FWHM of the overlap line shape, the laser FWHM parameters should be used.



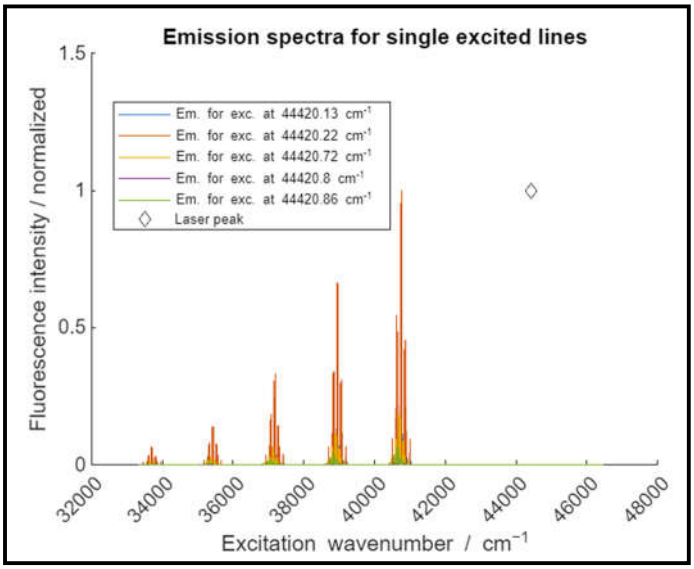
Results

Clicking the “calculate” button will start the simulation and will show, which transitions are covered by the line shape of the laser. The total emission spectra will be displayed for the range transmitted by the chosen filter.

To display the figure outside the live script window, click pop-out figures.



In addition, the emission spectra corresponding to each excited stated for the listed lines are displayed



3.3 Simulation of excitation spectra 'simulateExcSpectra.mlx'

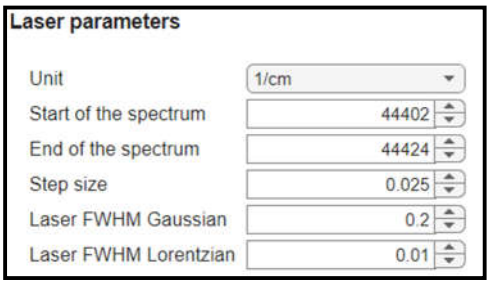
In this live script, fluorescence excitation spectra (for NO, SiO, OH, or O_2) can be simulated for the available species at given laser parameters, detection parameters, and collision data (if supported).

Species

The species can be selected in the drop-down menu:

Laser parameters

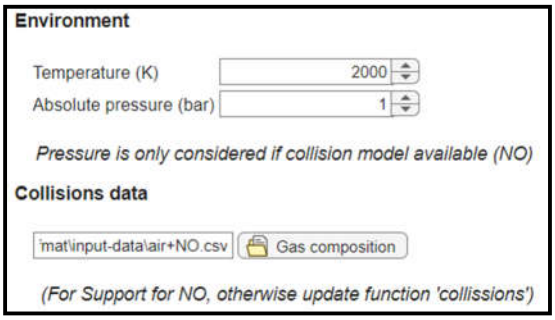
For defining the laser parameters the user can set the tuning region and the scan resolution (step size). Furthermore, the line shape can be defined through setting the FWHM of the Gaussian and Lorentzian contributions of the Voigt function. The units supported here are either nm and cm^{-1}.



Environment and collisions

Pressure (in bar) and temperature (in Kelvin) can be defined in this section. If the collision model is defined for the chosen species (section 4.6), the gas composition will be considered. Then, collisional

quenching, line broadening and line shift are calculated. Otherwise, these values will be considered negligible and for manipulating the Lorentzian FWHM of the overlap line shape, the laser FWHM parameters should be used.

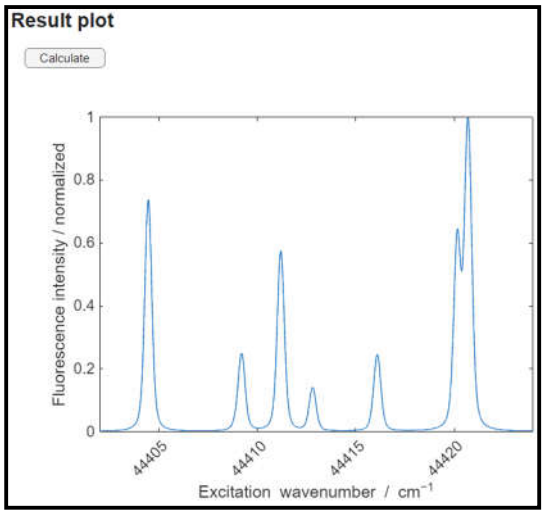


Detection

If a filter function is needed, this can be added as a spreadsheet with two columns as defined in section 2.3. The unit of the spectral dimension must be given (nm or cm^{-1}). The file path field can be left empty to disregard the filter.

Results

Finally, clicking the calculate button will start the simulation and plot the fluorescence intensities as function of excitation wavenumber.

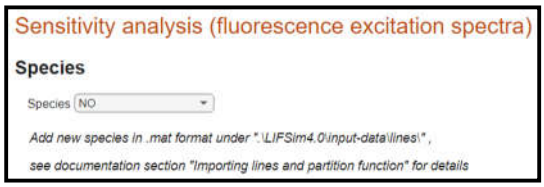


3.4 Sensitivity analysis for excitation spectra 'sensitivityAnalysis.mlx'

In this live script, the temperature sensitivity of the fluorescence excitation spectra (for NO, SiO, OH, or O_2) can be analyzed at given laser parameters, and detection parameters; according to a user-controlled synthetic measurement.

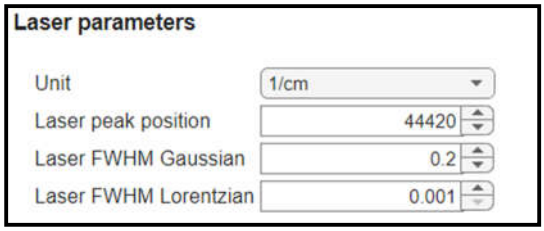
Species

The species can be selected in the drop-down menu:



Laser parameters

The laser parameters consist of the tuning region and the scan resolution (step size). Furthermore, the line shape can be defined through setting the FWHM of the Gaussian and Lorentzian contributions of the Voigt function. The units supported here are nm and cm^{-1}.



Synthetic measurement parameters

The sensitivity analysis supports the user in finding the most suitable wavelength range for determining temperature from a multi-line LIF scan. The script simulates spectra for the entire scan region defined under "laser parameters" and investigates the sensitivity in various "candidate spectral regions". These candidate regions are performed as defined in the following steps:

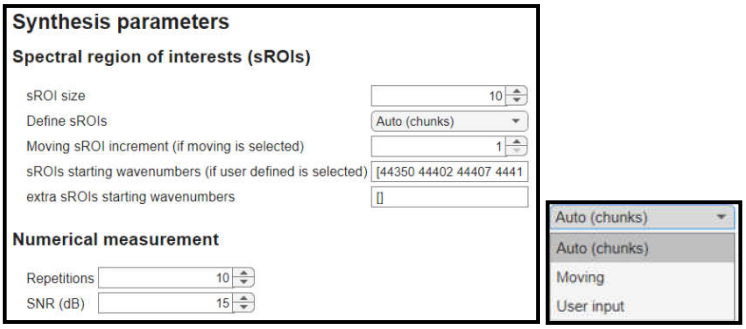
1. The size of the candidate spectral region of interest (sROI) defines the width of the spectral region potentially considered for the measurement. The wider the region, the more transitions will be covered and hence higher sensitivities can be reached.
2. The entire region is separated in "chunks" that are evaluated and compared. The chunking of the sROIs is controlled by three criteria:
   i. Auto: This produces sROI with their start spaced as the sROI size. E.g., if the overall spectral region is set to 44400 to 44450 cm^{-1} and the sROI size (step 1) is 10 cm^{-1}, five sub-regions (chunks) will be analyzed 44400–44410, 44410–44420, 44420–44430, 44430–44440, and 44440–44450 cm^{-1}.
   ii. Moving: This will consider the increment value in the following field. The next sROI will start after the beginning of the previous plus the increment value. E.g., if the overall spectral region is set to 44400 to 44450 cm^{-1}, the sROI size is 10 cm^{-1}, and the moving increment is 5, the script will initialize the nine regions 44400–44410, 44405–44415,

44410–44420, 44415–44425, 44420–44430, 44425–44435, 44430–44440, 44435–44445, and 44440–44450 cm^{-1}.

iii. User input: This will consider the start of the sROI defined by the user in the following field. The start of each chunk must then be given in this field.

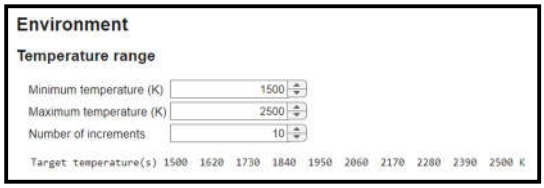
Regardless of which criterion is selected, if the field 'extra sROI' is defined, the script will add these to the initialized sROI. Only the start of the sROIs is needed in this field.

3. For each spectrum, artificial noise (in dB) is added before a temperature analysis is performed. The number of repetitions to be calculated can be determined to analyze the sensitivity to noise. E.g., if 10 repetitions are set, the script generates 10 noise patterns and add them to the spectra, resulting in 10 spectra simulating 10 measured spectra with noise patterns.



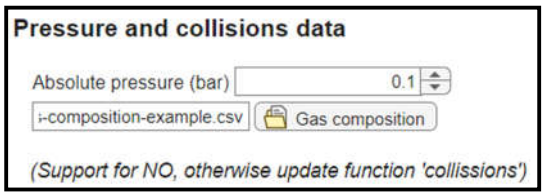
Temperature range

The temperature range for which the temperature sensitivity is to be evaluated can now be set by defining minimum and maximum temperatures (in Kelvin). The number of temperature increments can be set in the following field. E.g., a value of 10 increments for 1500 through 2500 K will define 10 target temperatures for the simulation, 1500, 1620, 1730, 1840, 1950, 2060, 2170, 2280, 2390, and 2500 K.



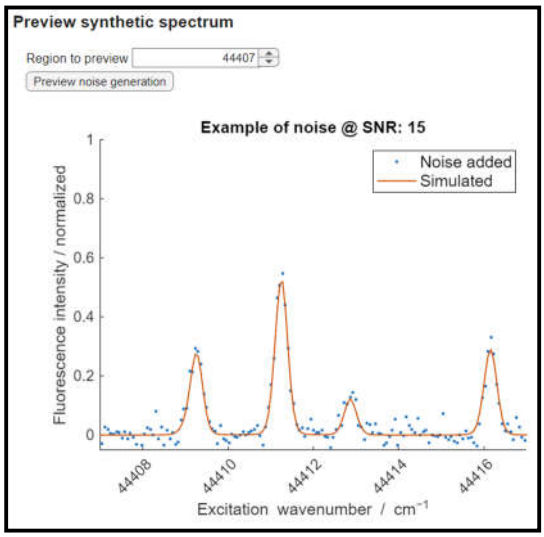
Pressure and collisions

The pressure (in bar) can be defined in this section. If the collision model is defined for the chosen species (section 4.6), the gas composition will be considered. Then, collisional quenching, line broadening and line shift are calculated. Otherwise, these values will be considered negligible and for manipulating the Lorentzian FWHM of the line-shape overlap, the laser FWHM parameters should be used.



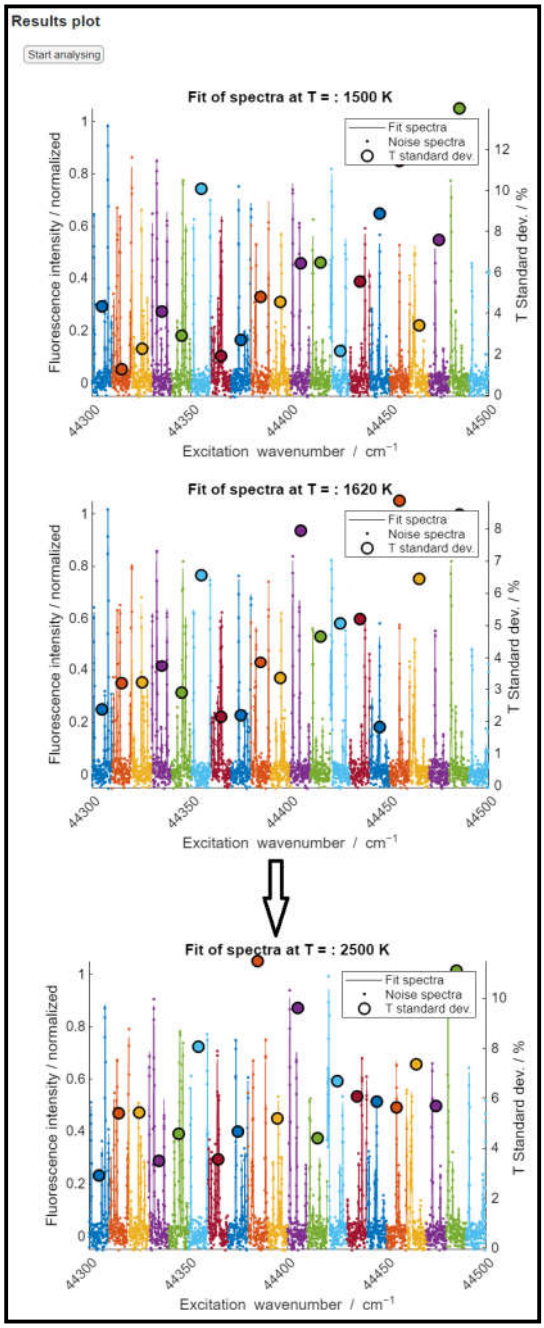
Preview

The synthetic spectrum based on the user input can be previewed before committing it to the analysis. A region to preview can be set as the default display. The figure can be interacted with, as usual with MATLAB figures.



Results

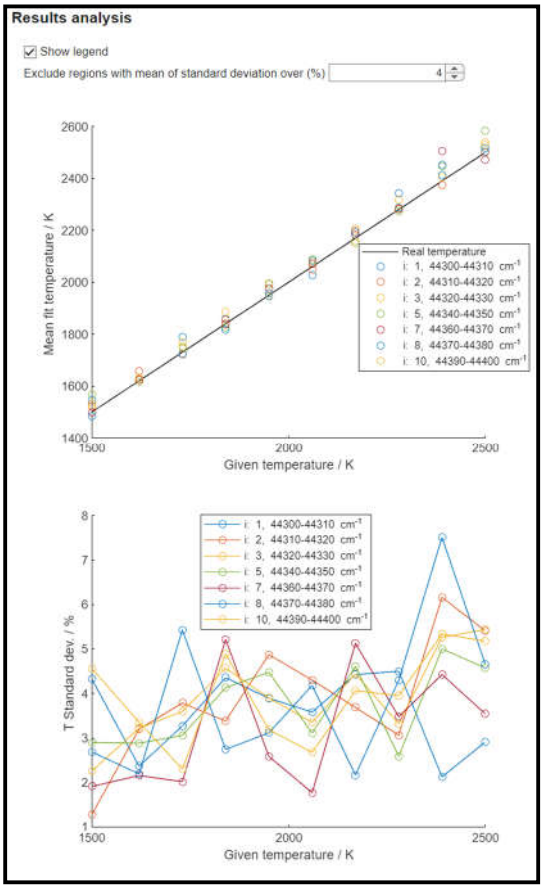
Clicking the “calculate” button will start the analysis and will plot for each temperature an example fit for each noisy sROI with the corresponding temperature standard deviation. The standard deviation is calculated based on the temperature fit results of the repetition.



The analysis is later displayed assessing each sROI for temperature sensitivity. This is estimated by calculating the mean fits temperature from the repetitions of each sROI. The result is plotted against the given or “real” temperature.

The second plot is displaying the standard deviation of the temperature for each sROI against the given temperature.

If the results for all sROIs are displayed, the plot will be crowded. Therefore, a display threshold based on the mean standard deviation for each sROI can be defined. Here then, the results for sROI with standard deviations under the threshold will be displayed.

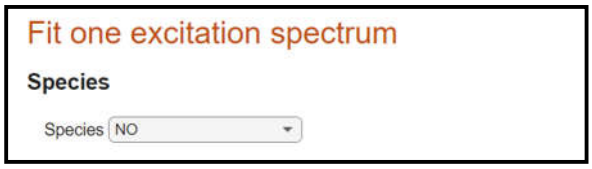


3.5 Fitting interface for a single spectrum 'fitExcSpectrumSingle.mlx'

This function fits a measured multi-line LIF spectrum with temperature, intensity and offset as free parameters from a given input spectrum.

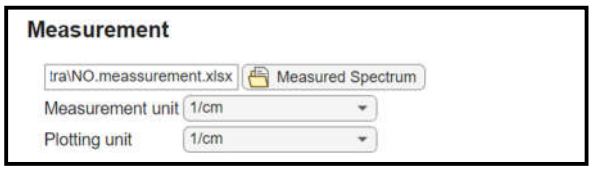
Species

The target species can be selected in the drop-down menu.



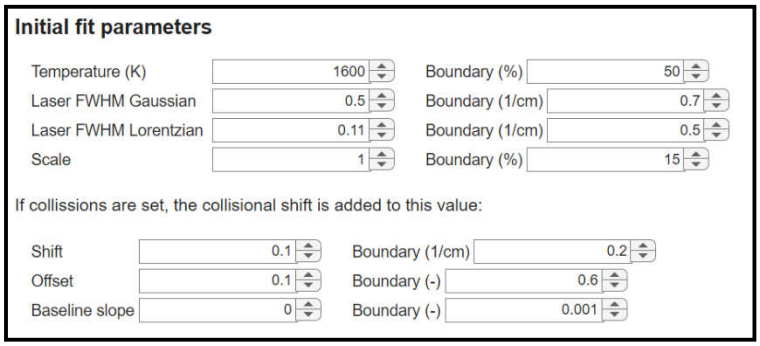
Measurement

The measured spectrum can be loaded from a spreadsheet. The LIF intensity should corrected for variations in the laser intensity. The file needs to provide wavenumber (or wavelength) and the related intensity. The unit (cm^{-1} or nm) can be selected for the measured and the plotted data.



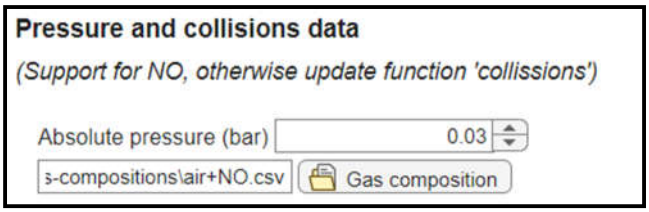
Initial fitting parameters

The fit is initialized with specific parameters. Also, boundary conditions can be given to restrict the data analysis to the expected possible values.



Pressure and collisions

The pressure (in bar) can be defined in this section. If the collision model is defined for the chosen species (section 4.6), the gas composition will be considered. Then, collisional quenching, line broadening and line shift are calculated. Otherwise, these values will be considered negligible and for manipulating the Lorentzian FWHM of the line shape overlap, the laser FWHM parameters should be used.



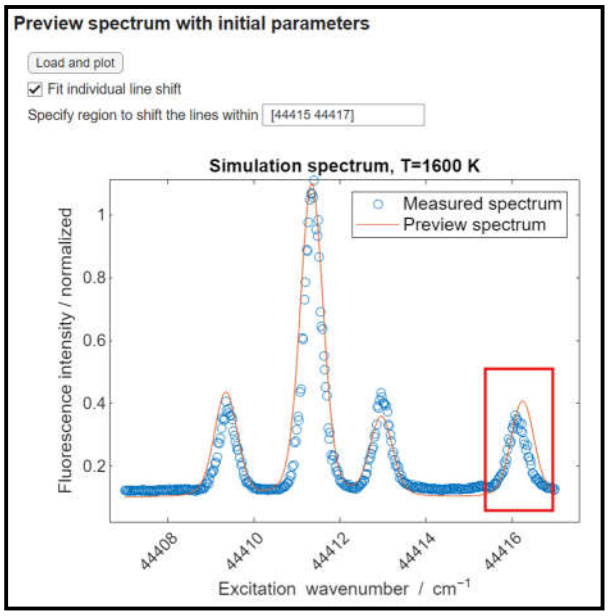
Detection

If a filter function is needed, this can be added as a spreadsheet with two columns as defined in section 2.3. The unit of the spectral dimension can be selected (nm or cm^{-1}). The file path field can be left empty to disregard the filter.

Preview before fitting

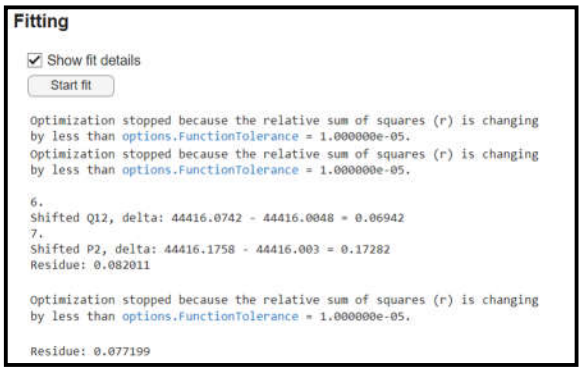
The user can preview a calculated spectrum based on the given initial parameters. This spectrum is displayed together with the measured data. In some cases, a part of the measured spectra has a wavelength offset (red box). To correct, the user can opt to fit the line positions. If the spectrum consists of high number of lines (e.g., 10 or more), it is recommended to define regions, where the lines

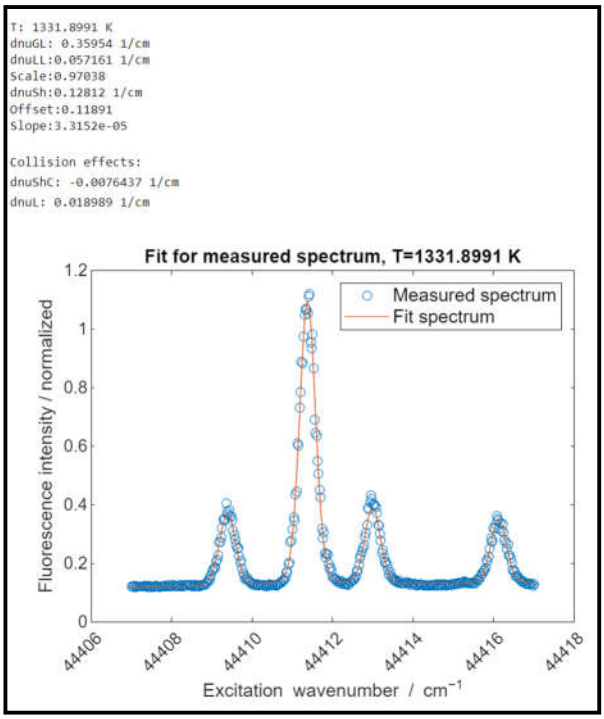
are suspected to have a wavelength bias, otherwise write in the field mask empty square brackets "`[]`" to regard correction for all line positions.



Fit results

The code will run a first fit if the line position correction is opted in and shows details on why the fit converged. It will also show, which lines are found in the given region and how much they are shifted, along with residual. Then, a final fit will be run showing the solution fit parameters.





3.6 Fitting interface for multi-line LIF images

This function helps to analyze multi-line LIF images, i.e., stacks of images with varying excitation wavelength are transformed to a temperature image.

3.6.1 Prepare multi-line LIF images 'loadTIFImages.mlx'

LIFSim provides the tools to import a multi-line LIF images through this script. The recording technologies or image correction may differ across research facilities, depending on the equipment or laser pulse energy correction, in some cases also laser absorption correction. This script expects the images to be corrected for laser pulse energy variations and local variation due to absorption, in addition to other necessary image processing (e.g., rotation, pixel binning, etc.).

This script will generate the necessary input for a measurement and create a MATLAB file, named '<timestamp>_scan.settings.mat'. This file can be later imported and processed be the script `fitExcSpectrumImage`.

The species can be selected in the first step.

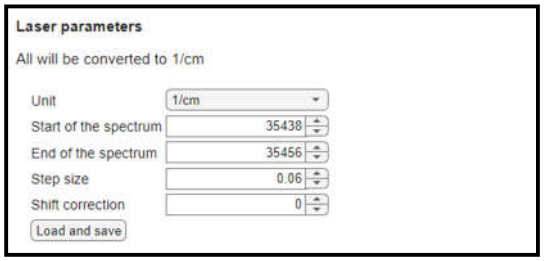
The user can point to the measurements folder, then the script will populate the following drop-down menu with the subfolder in the measurements folder; each subfolder here is regarded as a measurement containing the TIF images for a scanned spectrum.

The script recognizes just TIF images, where this extension can be as '.tif' or '.tiff'. This can be set in the input. This script is not tested with other formats.

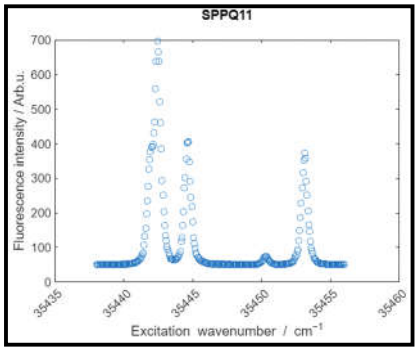
The export folder may be defined now, the script will create in a folder with the same name as the measurement name and store the 'scan.settings.mat' file under the defined export folder.

Next, the scanning parameters of the laser are defined. The script will read inputs in 'cm^{-1}' or 'nm'. This will be considered just for the input data; the script will eventually convert 'nm' to 'cm^{-1}'.

Finally, the user can load the data and save the output.



After processing, the script shows an example spectrum extracted from the dataset, based on binned 10×10 super pixels from the center of the frames.

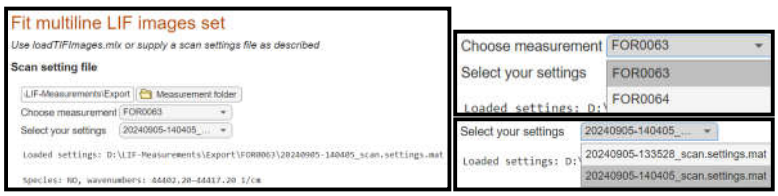


The containing folder will be displayed and can be opened in the final part. The settings file can be now used with following script 'fitExcSpectrumImage.mlx'

3.6.2 Fitting interface for LIF multiline imaging 'fitExcSpectrumImage.mlx'

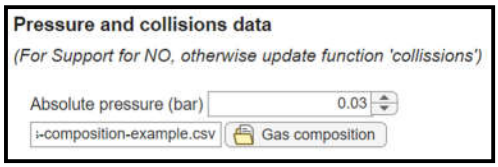
Loading settings

After importing the images and specifying the metadata for a measurement with 'loadTIFImages.mlx', the script can be used to fit the image set. First, specify the directory of the measurement data, choose the measurement, and select the settings file.



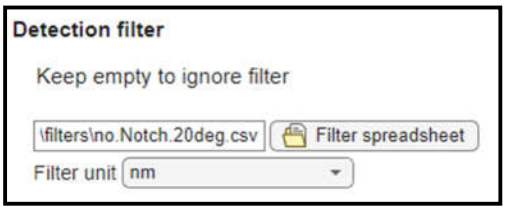
Pressure and collisions

The pressure and collision data are specified, if an existing model is available.



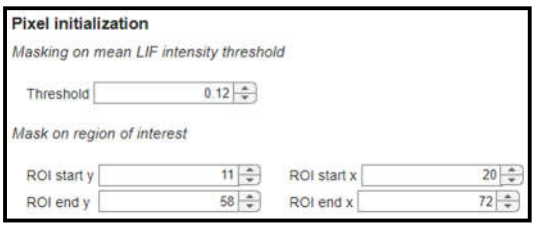
Detection

If a filter function is needed, this can be added as a spreadsheet with two columns as defined in section 2.3. The unit of the spectral dimension can be selected (nm or cm^{-1}). The file path field can be left empty to disregard the filter.



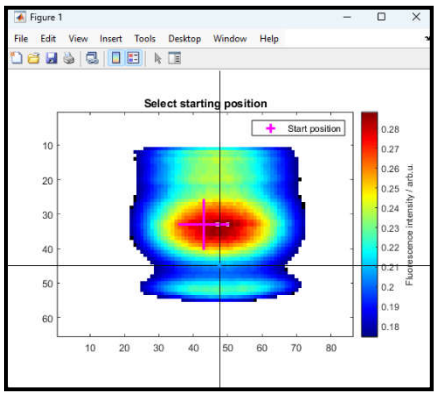
Pixel initialization

The temperature analysis is performed for a subset of the pixels (region of interest, ROI) in the image. The ROI can either be defined by setting a threshold value (areas below this threshold value will be disregarded) or by geometrically defining a ROI.

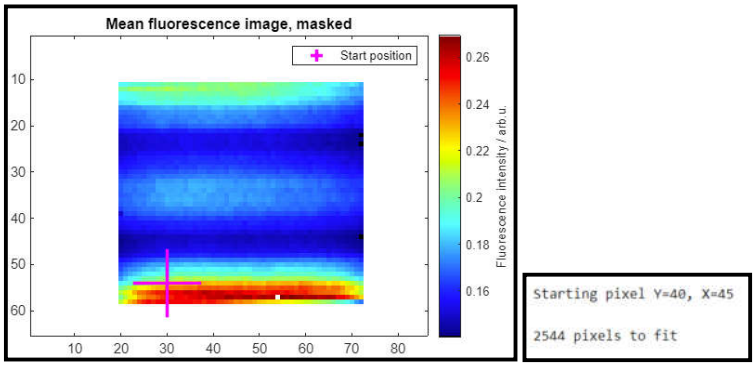


A starting pixel position should be defined for a test fit, ideally at a location with good signal quality. This position can be defined to be the center pixel, user input, or pop out on-graph selection.

On-plot selection:

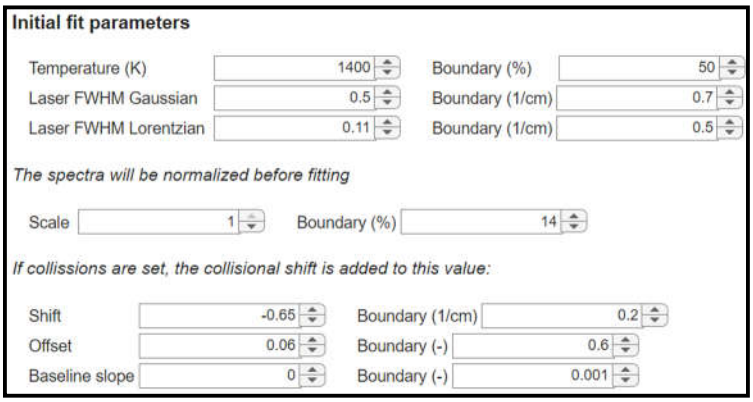


After the ROI and starting pixel selection, the averaged signal intensity image of the selected pixels will be displayed with the coordinates of the starting pixel and the count of total pixels.



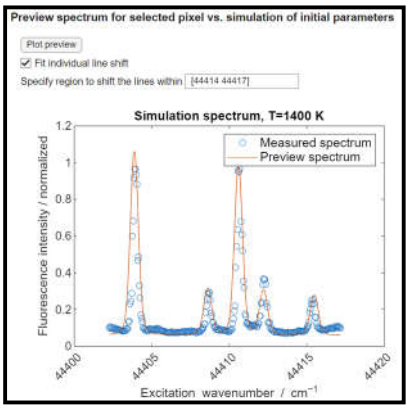
Initial fitting parameters

The initial fit parameters can be defined by setting boundary conditions. Contrary to the single spectrum fit, the spectra from each pixel position are individually normalized to unity before fitting, hence the scale here is one.



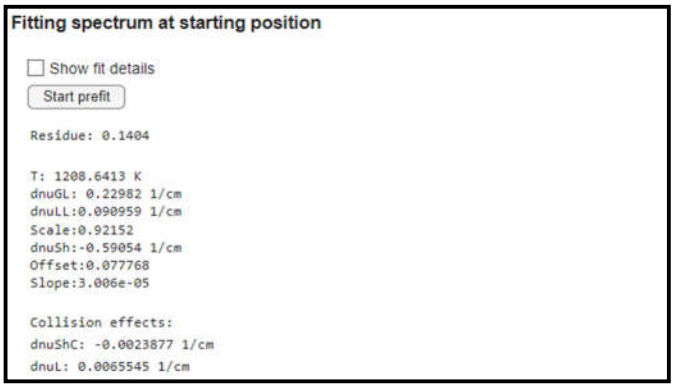
Preview before fitting

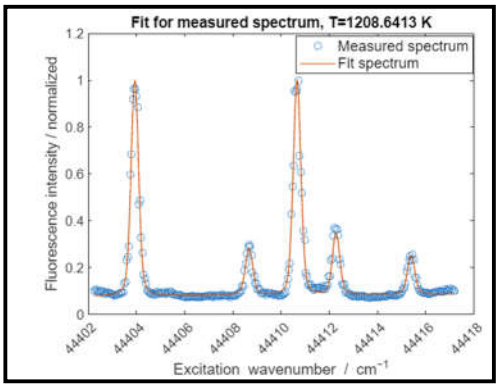
The user can preview a calculated spectrum based on the given initial parameters. This spectrum is displayed together with the measured data. In some cases, a part of the measured spectra has a wavelength offset (red box). To correct, the user can opt to fit the line positions. If the spectrum consists of high number of lines (e.g., 10 or more), it is recommended to define regions, where the lines are suspected to have a wavelength bias, otherwise write in the field mask empty square brackets "`[]`" to regard correction for all line positions.



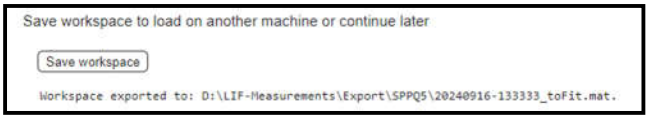
Test fit

Now, the spectrum at the selected spatial starting position can be extracted and fitted and the results are displayed.





After the test fit, the workspace can be saved optionally for exporting to another machine.



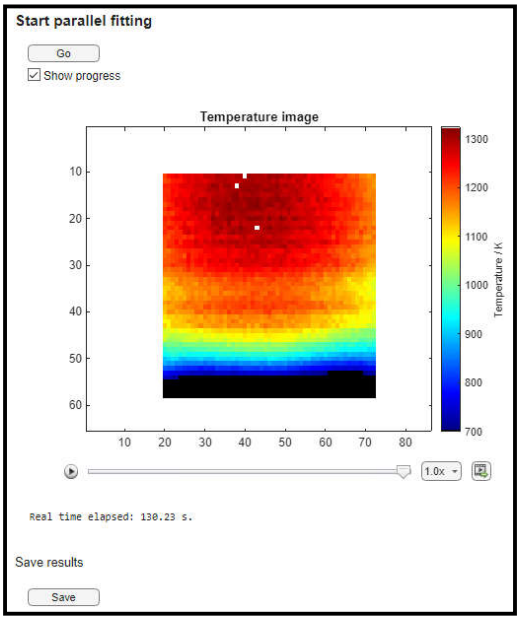
Parallel fit

Before starting the parallel, you may configure the number of cores can be defined through MATLAB's parallel preferences[4], however more cores would increase communication overhead.

If the test fit was satisfactory, the user can click "go" to start the parallel fit for the pixel. When the option "Show progress" is selected, the temperature fit results will be displayed for the processed spectra at the corresponding positions.

When all pixels are processed, the data can be saved within the same directory selected in the first step with file name formatted as '<time stamp>__fitResults.mat'.

[4] More on setting cores in MATLAB: https://de.mathworks.com/help/parallel-computing/discover-clusters-and-use-cluster-profiles.html



3.7 Interface for inferring mole fraction semiQuantMoleFractImage.mlx

The evaluation of mole fractions from the LIF signal intensity requires first the result of a line scan and temperature measurement. If a region is scanned including suitable lines for thermometry and mole fraction evaluation.

Case 1: Importing workspace

The saved workspace from the live script in section 3.6.2 can be imported here, assuming the multi-line LIF imaging thermometry performed covers also an isolated line with a low temperature sensitivity, e.g. Ref. [1].

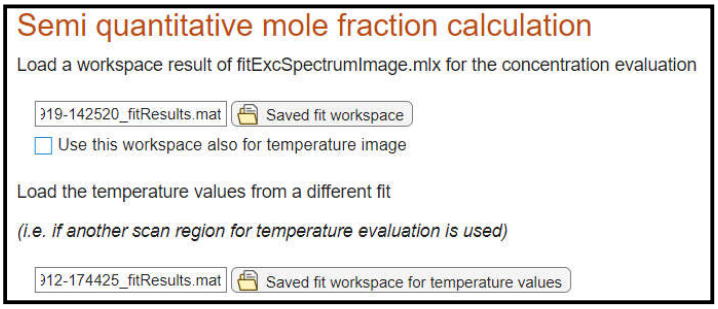
Case 2: Importing two workspaces

Opt-out the check box for using the same workspace in case two measurements are performed, for (i) scanning an isolated line for mole fraction evaluation and (ii) multi-line LIF thermometry.

Use the live script in section 3.6.2, twice:

1. For fitting the first region for thermometry.
2. For fitting the second region where usually a single overlap is scanned, hence here the temperature values are not relevant. However, the line-shape fitted parameters are needed from this workspace.

Then import as specified:

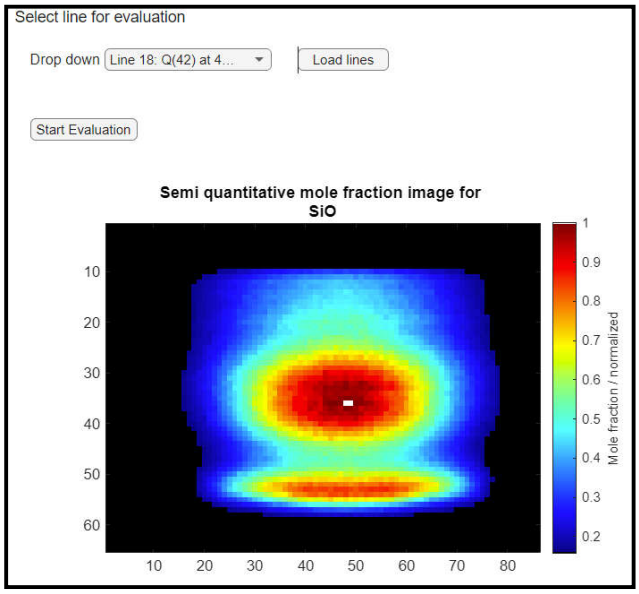


Selecting line

Select the line used for mole fraction evaluation.

Click “load lines” to repopulate the drop-down menu.

Start the evaluation, here then the field image for the semi-quantitative mole fraction will be displayed.



4 Description of functions

4.1 Function ‘selectLines’, loading lines information

This function loads the line database and the partition function for a given species (NO, SiO, OH, O_2). If further species are needed, the related line database should be added as described in sections 2.1 and 2.2. Then this script should be updated. E.g., add the following under the switch statement after adding the .mat files.

Syntax

```
[linelist, Z, n, emRange, MM, emList, listCell] = selectLines(spe-
cies, wnrange, o)
```

Inputs

`species`: A char verctor, e.g., `species = 'NO'`.

Optional inputs

`wnrange`: Wavenumber range in cm^{-1}, leave empty to load all lines available.

`outside`: Get more lines in an extended region around `wnrange`. This is useful in case the laser has a wavelength bais and you need to correct for this.

`listVersion`: The version of the list to load, what have already been used when importing the list with 'importLines.mlx'. See section 2.1.

`partVersion`: The version of the partition list to load, what have already been used when importing the list with 'importPartitionFunc.mlx'. See section 2.2.

Outputs

`linelist`: A cell vector containing the list of lines, each line data is stored in a line struct, see section 4.1.1.

`Z`: Partition function in a matrix (*T, Z*).

`n`: Number of lines found in the given range.

`emRange`: The begin and end of emission range for the excited range. See section 3.2.

`MM`: Species molar mass.

`emList`: Cell list containing at each index a list of emissions for a given line in `linelist`. The lists `emList` and `linelist` are of the same order, so `emList{i}` is the list emission lines for the line in `linelist{i}`, (`i` is any given index).

`listCell`: This returns the lines in a cell matrix as load from the selected MATLAB workspace, see section 2.1. The indices are appended to the last column, with a consistent order as in `linelist`.

Related

`lineStruct`, `fluorTransm`, `importLines.mlx`, `importPartitionFunc.mlx`

4.1.1 Function 'lineStruct', data structure for containing line parameters

This function is used mainly by the function `selectLines` to load each line into a structure. It converts a row entry from the imported line list into a structure, from a cell vector. This function is used to define the structure of the line, and therefore some quantities are set to an ineffective value, e.g., quenching to zero or sum of Einstein A coefficients for a line to 1, for a later processing in `selectLines` and `fluorTransm`.

Syntax

```
line = lineStruct(lineRaw)
```

Input

`lineRaw`: Cell vector, which contains parameters for one line, in the following order:

Spectral band
J'': Lower rotational quantum number
J': Upper rotational quantum number
E_{trans}: Line position / transition energy (cm^{-1})
E_{ground}: Ground state energy (cm^{-1})
A_{kj}: Einstein A emission coefficient (s^{-1})
B_{ik}: Einstein B absorption coefficient ($m^3/(Js^2)$)
Predissociation rate (s^{-1})
v'': Lower vibrational quantum number
v': Upper vibrational quantum number

Output

`line`: The line info as struct:

- `line.A`: Einstein A emission coefficient (s^{-1})
- `line.B`: Einstein B absorption coefficient ($m^3/(Js^2)$)
- `line.EGr`: Ground state energy (cm^{-1})
- `line.nu0`: Line position / transition energy (cm^{-1})
- `line.jLo`: Lower rotational quantum number
- `line.jUp`: Upper rotational quantum number
- `line.vLo`: Lower vibrational quantum number
- `line.vUp`: Upper vibrational quantum number

Placeholder variables:

- `line.Q`: Quenching rate (s^{-1}), initialized to 0, handled in `quenchRate`. See section 4.5.
- `line.W`: Photoionization (s^{-1}), initialized to 0, if not changed, the effect is not considered.
- `line.P`: Predissociation rate (s^{-1}), initialized to 0, if not changed, the effect is not considered.
- `line.emSum`: Einstein A coefficients sum for an excited state, initialized to 1, handled later
- `line.emSumTrapped`: Same as `line.emSum`, initialized to 1, handled later
- `line.transm`: The transmitted emission fraction, calculated in

4.2 Function 'loadGasComposition', loads gas compositions

This function loads a given gas composition, which can be used with functions `collisions`, `collisionalBroadening`, and `quenchRate` see section 4.6. The output will be a table with columns 'molecule' and 'fraction'. The elements with 0 will not be imported. The fractions will be normalized to 1.

Syntax

```
gas = loadGasComposition(path)
```

Input

`path`: System path to your file, e.g., "D:/path/to/file.csv", if this parameter is not passed, a prompt window will open to browse to the file location and select it.

Output

`gas`: A table with two columns, molecule and fraction corresponding to your .csv file.

Related functions:

`collisions`, `quenchRate`, `collisionalBroadening`

4.3 Function 'fluorTransm', interpolates filter transmission for emissions

This function interpolates a filter transmission at the emission wavenumbers for given data points of the filter transmission curve. This function is used with the function `excitationSpec`, see section 4.5.6, for calculating excitation spectra. The filter function and the emission lines results in an overlap function, here we assume that the filter transmission for a narrow region spanning around the emission line shape is constant. For each transition and related excited state, the Einstein A coefficients for the allowed transitions are attenuated with the given filter transmission curve `filter`. The sum of the attenuated coefficients is then calculated and updated in the line list.

Syntax

```
linelist = fluorTransm(linelist, emList, filter)
```

Input

`linelist`: The list of lines as returned by function `selectLines` in section 4.1.

`emList`: The emission lists of the given transitions, as returned by function `selectLines`

`filter`: The transmission curve of the used filter, as a table with `wnum` and `intens` as headers. See function `loadSpectrum` in 4.3.1. The `intens` column is the transmission of the filter.

Output

`linelist`: The updated input `linelist`, with the field `emSumTransm` from the `lineStruct` updated.

Related

`lineStruct`, `loadSpectrum`, `selectLines`, `excitationSpec`, `emissionSpec`

4.3.1 Function 'loadSpectrum', loads spectrum data (intensity vs wavenumber)

This function loads a filter transmission curve from a spreadsheet file and outputs a table with wavenumbers and transmission, `wndata.wnum` and `wndata.intens`. For example, a filter transmission can be loaded with this function and passed to the function `fluorTransm`.

Syntax

```
[wndata, nmData] = loadSpectrum(filepath, unit)
```

Input

`path`: System path to your file

Optional input:

`unit`: Set the unit of the file cm^{-1} or nm.

Output

`wndata`: The wavenumbers (if input is in nm, it will be converted to cm^{-1}), and transmission. As `wndata.wnum` and `wndata.intens`

`nmData`: The wavelength (if input is in cm^{-1}, it will be converted to nm), and transmission. As `nmData.wnum` and `nmData.intens`

Related

`fluorTransm`

4.4 Auxiliary functions

4.4.1 Function 'molMass', calculates the molar mass for a molecule

Calculates an approximate molar mass of a given formula in simple form. This function supports no brackets support. Input, e.g.,: 'NO', 'SiO', 'OH','O2'.

Syntax

```
MM = molMass(formula)
```

Input

`formula`: Simple molecular formula; case sensitive and no brackets supported.

Output

`MM`: The calculated molar mass in g/mol.

Related

`selectLines`, `excitationSpec`, `emissionSpec`, `absorptionSpec`, `dnuGFun`, `quenchRate`

4.4.2 Function 'calcFb', calculates the Boltzmann factor

This function calculates the Boltzmann factor for the population of the energy level *i* for a given ground state energy, temperature, partition function, and lower rotation quantum number.

$$f_{\mathrm{B}}(T) = \frac{2J'' + 1}{Z} \exp\left(-\frac{\epsilon_i}{kT}\right), \tag{1}$$

Syntax

```
fB = calcFb(jLo, T, Z, EGr)
```

Input

`jLo`: Lower rotation quantum number

`T`: Temperature in Kelvin

`Z`: Partition function for the temperature `T`.

`EGr`: Ground state energy in cm^{-1}.

Output

`fB`: The calculated Boltzmann factor

Related

`excitationSpec`, `emissionSpec`, `absorptionSpec`, `fluorTransm`

4.5 Spectra calculation

4.5.1 Function 'overlap', calculates the overlap function of two Voigt functions

This function computes the spectral overlap between a laser profile and an absorption profile for a specific transition. The absorption profiles are modeled as combinations of Lorentzian and Gaussian functions, and this function quantifies the overlap with the laser profile. The function here uses the McLean model of calculating the Voigt line-shapes for the transition. These Voigt line-shapes are then convoluted to obtain the overlap.

Syntax

```
[gOver,g1,g2] = overlap(nu0,res,range,aL,dnuG1,dnuL1,dnuG2,dnuL2)
```

Input

`nu0`: The central frequency or reference frequency.

`res`: The resolution of the analysis.

`range`: The spectral range. It can be an array of frequencies.

`aL`: The Lorentzian amplitude.

`dnuG1`: The Gaussian component of the transition profile (linewidth).

`dnuL1`: The Lorentzian component of the transition profile (linewidth).

`dnuG2`: The Gaussian component of the laser profile (linewidth).

`dnuL2`: The Lorentzian component of the laser profile (linewidth), this can be kept minimal.

Output

`gOver`: The calculated spectral overlap between signal 1 and signal 2.

`g1`: The calculated lineshape 1.

`g2`: The calculated lineshape 2.

Example:

```
gOver = overlap(nu0, res, range, aL, dnuG1, dnuL1, dnuG2, dnuL2);
pos = abs(laserPos-nu0-shift)/resolution;
gamma = interp1(0:length(gOver)-1, gOver, pos, 'linear', 'extrap');
```

Related

`voigtlineMcLean`, `dnuGFun`, `collisionalBroadening`, `excitationSpec`, `emissionSpec`, `absorptionSpec`

4.5.2 Function 'voigtlineMcLean', calculates a Voigt profile

Calculates a Voigt profile for modeling a single spectral transition based on the C implementation provided by McLean et al. [2]. The line shapes are calculated in non-dimensional form, for usage in modeling absorption, emission, and excitation spectra, where subsequently each line-shape is weighted accordingly.

Syntax

```
V = voigtlineMcLean(nu,nu0,dnuG,dnuL,aL)
```

Input

`nu`: Frequency range.

`nu0:` The central frequency or reference frequency.

`dnuG`: The Gaussian component of the transition profile (linewidth).

`dnuL`: The Lorentzian component of the transition profile (linewidth).

`aL`: The Lorentzian amplitude

Output

`V`: The Voigt profile as a vector

Related

`overlap`, `dnuGFun`, `collisionalBroadening`, `excitationSpec`, `emissionSpec`, `absorptionSpec`

4.5.3 Function 'dnuGFun', calculates Doppler Gaussian broadening of a transition

This is a shorthand function to calculate the Doppler Gaussian broadening of a transition based on temperature, mole mass, and center wavenumber.

Syntax

```
dnuG = dnuGFun(T, MM, nu0)
```

Input

`T`: The temperature in Kelvin

`MM`: The molar mass of the diatomic species in g/mol.

`nu0`: The center wavenumber.

Output

`dnuG`: The Doppler broadening linewidth for the given inputs.

Related

`excitationSpec`, `emissionSpec`, `absorptionSpec`

4.5.4 Function 'absorptionSpec', absorption spectrum simulation

This function calculates the absorption spectrum using the absorbance, eq. (2), for a given wavenumber region of the incident light (laser), molecule, temperature, and instrument characteristics. An overlap function is calculated, based on two Voigt functions of the laser and the transition, see function `overlap`, with a given laser linewidth parameter (Gaussian and Lorentzian linewidth contributions).

$$-\ln\left(\frac{I}{I_0}\right) = h\,\tilde{\nu}\,NL\,f_{\mathrm{B}}(T)\,B_{ik}\,\Gamma_{\tilde{\nu}}(x,p,T). \tag{2}$$

Syntax

```
spec = absorptionSpec(wnum, linelist, MM, T, dnuGL, dnuLL,
params)
```

Input

`wnum`: The wavenumber range to be simulated, in cm^{-1}, e.g., `wnum = 44407:0.1:44417;`.

`linelist`: The list of lines, each line data is stored in a line struct, see `selectLines` and `linesStruct` in 4.1.

`MM`: Molar mass, g/mol

`T`: Temperature, Kelvin

`dnuGL`: The Gaussian linewidth of the laser line-shape, cm^{-1}.

`dnuLL`: The Lorentzian linewidth of the laser line-shape, cm^{-1}.

Optional input

`params`: This is a set of optional parameters used as a name-value cell or struct

- `Z`: The partition function acquired from `selectLines`. This may be omitted if not available, however, this is not recommended.
- `dnuL`: The Lorentzian linewidth of the transition, resulting from collisional broadening. This parameter is set by using `collisions` and `collisionalBroadening`. If the latter is not available and no value for `dnuL` is set, then a minimal default value is assumed, `0.0001` cm^{-1}, and thereby the overall Lorentzian feature of the overlap function is controlled mainly through the laser parameter `dnuLL`.
- `dnuSh`: The collisional shifting parameter, to be set using `collisions` and `collisionalBroadening`. In case no collision data is available, this value is zero and maybe used as a free parameter to adjust the shift.
- `resFactor`: A divider factor to the step value of `wnum`. This is used for defining the resolution for the overlap function generation. See `overlap` in section 4.5.1. If not given, the default value is 5.
- `normalize`: A Boolean flag to normalize the spectra. The default value is false.
- `limit`: This is a limiting factor used as multiplier to the sum of the linewidth (`dnuG`, `dnuL`, `dnuGL`, `dnuLL`). The result multiplication is considered as a range limit around a given center frequency. The higher the limit is, the further lines from the center frequency would be considered for interpolation. This is set to 12 by default, if the user did not pass the parameter.
- `N`: The number density of the absorbing molecules, m^{-3}.
- `L`: The length of the absorption path, m.

Output

`spec`: The calculated absorption spectrum with the same length as `wnum`.

Example:

See the code of live script `simulateAbsorptionSpectra.mlx`, described in section 3.3.

Related

`selectLines`, `collisions`, `quenchRate`, `collisionalBroadening`, `overlap`, `voigtlineMcLean`, `simulateAbsorptionSpectra.mlx`

4.5.5 Function 'emissionSpec', laser-induced emission spectrum simulation

This function calculates the spectrally resolved fluorescence emissions based on a laser excitation at a given wavenumber position.

$$I_{k\to j}^{ikj}(\tilde{\nu}^{\mathrm{em}},\tilde{\nu}^{\mathrm{ex}}) = N\, I_{\tilde{\nu}}^{0}\, B_{ik}\, f_{\mathrm{B}}(T)\, \Gamma_{\tilde{\nu}}(x,p,T)\, \frac{A_{kj}}{\sum_{j'} A_{kj'} + Q_k(x,p,T) + P_k + W_k} \frac{\Omega}{4\pi}\, F(\tilde{\nu}^{em})\, \varepsilon\, \eta \tag{3}$$

$$I_{\mathrm{em\_spec}}^{ikj}(\tilde{\nu}^{\mathrm{em}},\tilde{\nu}^{\mathrm{ex}}) = \sum_{j} I_{k\to j}^{ikj}(\tilde{\nu}^{\mathrm{em}},\tilde{\nu}^{\mathrm{ex}})$$

Here an overlap function will be computed based on the laser and transition line-shape functions. The laser function is computed from the FWHMs of the Gaussian and Lorentzian contributions and then convoluted with the transitions.

Syntax

```
function [EmSpectra, excitedLines, sumFluor] =
emissionSpec(laserPos, emWnum, linelist, MM, T, emList, em, inst,
ex)
```

Input

`laserPos`: The laser position in wavenumbers.

`emWnum`: The emission wavenumber range to be simulated, in cm^{-1}, e.g. `emWnum = 33000:1:47000`.

`linelist`: The list of lines, each line data is stored in a line struct, see `selectLines` and `linesStruct` in 4.1.

`MM`: Molar mass, g/mol

`T`: Temperature, Kelvin

`emList`: Cell list containing at each index a list of emissions for a given line in `linelist`. The lists `emList` and `linelist` are of the same order, so `emList{idx}` is the list emission lines for the line in `linelist{idx}`, (`idx` is any given index). See function `selectLines` in section 4.1.

Optional input

`em`:

- `dnuL`: The Lorentzian linewidth of the emission transition, resulting from collisional broadening. This parameter is set by using `collisions` and `collisionalBroadening`. If the latter is not available and no value for `dnuL` is set, then a minimal default value is assumed, `0.0001` cm^{-1}, and thereby the overall Lorentzian feature of the overlap function is controlled mainly through the laser parameter `dnuLL`.

- `dnuSh`: The collisional shifting parameter, to be set using `collisions` and `collisionalBroadening`. In case no collision data is available, this value is zero and maybe used as a free parameter to adjust the shift.
- `normalize`: A Boolean flag to normalize the spectra. The default value is true. This normalization considers the maximum of the output matrix `EmSpectra`.
- `Z`: The partition function acquired from `selectLines`. This may be omitted if not available, however it is not recommended.
- `limit`: This is a limiting factor used as multiplier to the sum of the linewidth (`em.dnuL`, `emdnuG`, `inst.dnuGsm`, `inst.dnuLsm`). The result multiplication is considered as a range limit around a given center frequency. The higher the limit is, the further lines from the center frequency would be considered for interpolation. This is set to 100 by default, if the user did not pass the parameter.

`inst`:

- `dnuGsm`: The Gaussian linewidth of the spectrometer line-shape, cm^{-1}.
- `dnuLsm`: The Lorentzian linewidth of the spectrometer line-shape, cm^{-1}.

`ex`:

- `resEx`: The resolution for calculating the laser lineshape.
- `dnuGL`: The Gaussian linewidth of the laser line-shape, cm^{-1}, default = 0.1.
- `dnuLL`: The Lorentzian linewidth of the laser line-shape, cm^{-1}, default = 0.01.
- `rangeWidth`: The width of wavenumber range for generating the excitation overlap.
- `dnuLex`: Same as `dnuL`, unless otherwise specified.
- `dnuShex`: Same as `dnuSh`, unless otherwise specified.
- `exLimit`: The threshold limit of the excitation overlap function when to consider lines to excite. A higher limit will consider a narrower region around the peak, hence less exited lines. This is important to omit regions of the overlap when the intensity is negligible, which makes the computation faster.

Output

`EmSpectra`: The emission spectra for every excited transition within the line-shape function of the laser.

`excitedLines`: The indices of the lines found in the region of excitation. The indices can be used in the cell list `linelist`.

`sumFluor`: The sum of the emission spectra `EmSpectra`.

Example:

See the code of live script 'simulateEmSpectra.mlx', described in section 3.2.

Related

`selectLines`, `collisions`, `quenchRate`, `collisionalBroadening`, `overlap`, `voigtlineMcLean`, `simulateEmSpectra.mlx`

4.5.6 Function 'excitationSpec', fluorescence excitation spectra simulation

Calculates the LIF excitation spectrum for a specified wavenumber range (in cm^{-1}). This function computes two Voigt functions for the laser and line, based on the Voigt McLean model (see function 'voigtlineMcLean'). It then calculates the overlap function, which is weighted with the given physical parameters.

$$I_{\mathrm{LIF}}(\tilde{\nu}^{\mathrm{em}},\tilde{\nu}^{\mathrm{ex}}) = \sum_i I^{ikj}_{\mathrm{em_{spec.}}}(\tilde{\nu}^{\mathrm{em}},\tilde{\nu}^{\mathrm{ex}})$$
$$= \sum_i \left[N\, I^0_{\tilde{\nu}}\, B_{ik}\, f_{\mathrm{B}}(T)\, \Gamma_{\tilde{\nu}}(x,p,T) \frac{\sum_j A_{kj} F(\tilde{\nu}^{\mathrm{em}})}{\sum_{j'} A_{kj'} + Q_k(x,p,T) + P_k + W_k} \frac{\Omega}{4\pi} \varepsilon\eta \right]. \tag{4}$$

Absorption Quantum yield and spectral detection efficiency

$$S(\tilde{\nu}) = \frac{I_{\mathrm{LIF}}(\tilde{\nu})}{I^0_{\tilde{\nu}}} = \sum_i \left[N\, I^0_{\tilde{\nu}}\, B_{ik}\, f_{\mathrm{B}}(T) \Gamma_{\tilde{\nu}}(x,p,T) \frac{\sum_j A_{kj} F(\tilde{\nu}^{em})}{\sum_{j'} A_{kj'} + Q_k(x,p,T) + P_k + W_k} \right] \tag{5}$$

To consider the quenching rate, `Q`, use the functions `collisions` and `quenchRate` to calculate it. Furthermore, the Predissociation P_k and photoionization W_k are considered only if these values are introduced in the `linelist` through updating the values `W` and `P` in each `lineStruct`. For thermometry, these values are rather not relevant, since the experiment is not causing these phenomena.

Syntax

```
spec = excitationSpec(wnum, linelist, MM, T, dnuGL, dnuLL, params)
```

Inputs

`wnum`: The wavenumber range to be simulated, in cm^{-1}, e.g., `wnum = 44407:0.1:44417;`.

`linelist`: The list of lines, each line data is stored in a line struct, see `selectLines` and `linesStruct` in 4.1.

`MM`: Molar mass g/mol

`T`: Temperature in Kelvin

`dnuGL`: The Gaussian linewidth of the laser line-shape, cm^{-1}.

`dnuLL`: The Lorentzian linewidth of the laser line-shape, cm^{-1}.

`params`: This is a set of optional parameters used as a name-value cell or struct

- `Z`: The partition function acquired from `selectLines`. This may be omitted if not available, however it is not recommended.
- `dnuL`: The Lorentzian linewidth of the transition, resulting from collisional broadening. This parameter is set by using `collisions` and `collisionalBroadening`. If the latter is

not available and no value for `dnuL` is set, then a minimal default value is assumed, `0.0001` cm^{-1}, and thereby the overall Lorentzian feature of the overlap function is controlled mainly through the laser parameter `dnuLL`.

- `dnuSh`: The collisional shifting parameter, to be set using `collisions` and `collisionalBroadening`. In case no collision data available, this value is zero and maybe used as a free parameter to adjust the shift.
- `resFactor`: A divider factor to the step value of `wnum`. This is used for defining the resolution for the overlap function generation. See `overlap` in section 4.5.1. If not given, the default value is 5.
- `normalize`: A Boolean flag to normalize the spectra. The default value is true.
- `limit`: This is a limiting factor used as multiplier to the sum of the linewidth (`dnuG`, `dnuL`, `dnuGL`, `dnuLL`). The result multiplication is considered as a range limit around a given center frequency. The higher the limit is, the further lines from the center frequency would be considered for interpolation. This is set to 12 by default, if the user did not pass the parameter.

Output

`spec`: The calculated excitation spectrum with the same length as `wnum`.

Example:

See the code of live script 'simulateExcSpectra.mlx', described in section 3.3.

Related

`selectLines`, `collisions`, `quenchRate`, `collisionalBroadening`, `overlap`, `voigtlineMcLean`, `simulateExcSpectra.mlx`

4.6 Collision data

4.6.1 Function 'collisions', loads a collision model

In this function the models for collisional broadening, shift, and quenching are implemented as structure. This version supports the NO species based on the Harpoon model, by Paul et al. [3].

Syntax

```
C = collisions(species)
```

Broadening:

The supported partners for collisional broadening and shift are: N_2, O_2, H_2O, Ar, CO_2, CO, CH_4.

Quenching:

The supported collision partners are: N_2, O_2, CO_2, CO, H_2O, CH_4, C_2H_6, C_3H_8, C_2H_4, C_2H_2, NO, NO_2, N_2O, NH_3, NH, H_2, O, H, OH, CH, He, Ne, Ar, Kr, Xe.

Input

`Species`: The species to which the model should be returned, e.g., 'NO'.

Output

`C`: The collisional model as a structure:

`C.(partner).br(T)`, e.g., `C.N2.br = @(T) 0.585*(295/T).^0.75;`

`C.(partner).sh(T)`, e.g., `C.N2.sh = @(T) -0.18*(295/T).^0.56;`

`C.(partner).qu(T)`,e.g., `C.N2.qu = @(T) Harpoon2(0,0.88,4.9,48,32,T);`

e.g.,:

The broadening, shift and collisions, related to a corresponding partner, at a temperature T can be acquired as:

```
T = 1500;
C.O2.br(T)
C.O2.sh(T)
C.O2.qu(T)
C.Ar.br(T)
C.Ar.sh(T)
C.Ar.qu(T)
```

Related

`collisionalBroadening`, `quenchRate`, `excSpecFitCost`, `fitExcitationSpec`, `excitationSpec`, `emissionSpec`

4.6.2 Function 'collisionalBroadening', calculates the collisional broadening

The function calculates the collisional broadening, `dnuL`, and shifting parameters, `dnuSh`, for a given gas mixture at specified pressure and temperature conditions. It calculates the total collisional line broadening and shifting resulting from the contributing collision partners. These partners and their corresponding fraction are provided to the function as input, along with temperature and pressure. If a partner is added in the gas composition, where this molecule has no model provided, this will be notified within a third output.

Syntax

```
[dnuL, dnuSh, missing] = collisionalBroadening(gas, colls, P, T)
```

Input

`gas`: A table containing two columns (`molecule`, `fraction`) listing the composition. See section 2.4.

`colls`: The collision model based on the function `collisions`.

`P`: Pressure in bar.

`T`: Temperature in Kelvin.

Output

`dnuL`: The total collisional broadening from the calculated model, see section 4.6.1.

`dnuSh`: The total collisional shift from the calculated model, see section 4.6.1.

`missing` A cell list of missing collisional model for a given molecule/atom in gas was not present. Otherwise, this will be empty.

Related

`loadGasComposition`, `collisions`, `quenchRate`, `excSpecFitCost`, `fitExcitationSpec`, `excitationSpec`, `emissionSpec`

4.6.3 Function 'quenchRate', calculates the quenching rate

This function computes the total quenching rate for a given molecule, *A*, with respect to a gas composition of collision partners, *p*. The quenching rate, eq. (6), is calculated based on the pressure in bar, Temperature in Kelvin, the gas composition of different collision partners as fractions, the molar masses of the collision partners to calculate the reduced mass, μ_p eq (8), in the relative velocity, v_p eq (7). The Avogadro constant, bar to Pa (10^5), and $\dot{A}^2$ to m^2 (10^{-20}), are used to have *Q* in s^{-1}.

$$Q_A(x_p, p, T) = \frac{p}{kT}\sum_p x_p v_p \sigma_p(T) \tag{6}$$

$$v_p = \sqrt{\frac{8kT}{\pi\mu_p}} \tag{7}$$

$$\mu_p = \frac{m_A m_p}{m_A + m_p} \tag{8}$$

$$\frac{1}{\mu_p} = \frac{1}{m_A} + \frac{1}{m_p} = N_A\left(\frac{1}{M_A} + \frac{1}{M_p}\right) \tag{9}$$

Syntax

```
[quen, linelist] = quenchRate(gas, colls, T, MM, linelist)
```

Input

`gas`: A table containing two columns (`molecule`, `fraction`) listing the composition. See section 2.4.

`colls`: The collision model based on the function `collisions`.

`T`: Temperature in Kelvin.

`P`: Pressure in bar.

`MM`: The molar mass of the diatomic species in g/mol.

`linelist`: A loaded `linelist` can be optionally passed to the function. This will update the field `Q` in each `lineStruct` in the list.

Output

`quen`: The quenching rate in s^{-1}.

`linelist`: The updated input `linelist`, with the field `Q` from the `lineStruct` updated.

Related

`loadGasComposition`, `collisions`, `collisionalBroadening`, `excSpecFitCost`, `fitExcitationSpec`, `excitationSpec`, `emissionSpec`

4.7 Fitting excitation spectra

4.7.1 Fit parameters

The fit parameters are used in a matrix vector of type double and contains the following seven quantities:

1. Temperature in K
2. FWHM of Gaussian contribution to the laser line-shape.
3. FWHM of Lorentzian contribution to the laser line-shape.
4. The scale of the simulated spectrum, to match the measured data.
5. The shift correction for the spectrum, positive or negative values. If `colls` are available, the shift due to collisions is calculated and the fit value of this position will act as an extra play room for the fit. To reduce its effect, narrow the boundaries.
6. The offset of the baseline correction, used in function `calBaseline`.
7. The slope of the baseline correction, used in function `calBaseline`.

4.7.2 Function 'calBaseline', calculates linear line for baseline correction

This function calculates the linear baseline with the equation $y = ax + b$ for a given length.

Syntax

```
baseline = calBaseline (a,b,L)
```

Input

`a`: The slope of the line.

`b`: The *y*-intercept or the offset.

`L`: The length of the baseline vector needed.

Output

`baseline`: The baseline vector.

Related

`excSpecFitCost`, `fitExcitationSpec`, `fitExcSpectrumImage.mlx`, `fitExcSpectrumSingle.mlx`

4.7.3 Function 'excSpecFitCost', the cost function between two spectra

The difference between a measured and simulated spectrum is calculated with a baseline correction and a scaling factor. This function is used as an error function for fitting with `lsqnonlin`. A baseline is calculated from the user parameters and then subtracted from the measured spectrum. The excitation parameters are used to simulate the spectrum, which is scaled by the given factor. As follows,

$$cost = \widehat{S}(\tilde{\nu}) - \text{baseline} - \text{scale} \times S(\tilde{\nu}). \tag{10}$$

Syntax

```
cost = excSpecFitCost(wnum, measSpec, linelist, MM, T, dnuGL, dnuLL,
a, specParams, fitParams, colls)
```

Input

`measSpec`: The measurement data points, which should have the same length as `wnum`.

Forwarded input

The inputs (`wnum`, `linelist`, `MM`, `T`, `dnuGL`, `dnuLL`, `resFactor`, `Z`, `limit`) are forwarded to the function `excitationSpec`, see section 4.5.6 for details about these. Also, if `colls` is not available, then `dnuSh` and `dnuL` of `specParams` are forwarded to `excitationSpec`, if passed to the current function.

Optional input

`a`: The slope of the baseline (default = 1).

`offset`: The base line offset (default = 0).

`scale`: The scaling factor for the simulated spectrum to match the measured data (default = 1).

`collParam`:

- `colls` and `gas`: The collision model and the gas composition, see section 4.6. If `colls` is empty the quenching and collisional broadening are not calculated, then the given or default

dnuSh and dnuL value would not be overridden. If colls and gas are available and dnuSh is given by user, then this given shift will be added to the collisional shift.
- P: Pressure in bar. This value is only considered if colls are available.

Output

cost: The difference between the measured and simulated spectra with respect to the given parameters.

Related

fitExcitationSpec, lsqnonlin, fitImageDataset, calBaseline, fitExcSpectrumImage.mlx, fitExcSpectrumSingle.mlx, lineShiftCost

4.7.4 Function 'fitExcitationSpec', optimization function for measured excitation spectrum

This is the fit optimization function for a measured excitation spectrum. This function uses the optimization function lsqnonlin with a user-given options, see getFitOptions. Here, the expected fit parameters are seven with lower and upper boundaries vectors.

Syntax

```
[solutionVals, residue] = fitExcitationSpec(wnum, measSpec, linelist, MM, bestGuess, options, fit, specParams, collParam)
```

Input:

wnum: Wavenumber region in cm^{-1}.

measSpec: The measured spectrum data point, with the same length as wnum.

linelist: The linelist obtained from function selectLines.

MM: Molar mass of the species, obtained from function molMass or selectLines.

bestGuess: A matrix vector with seven places, which are the initial fit parameters to start the optimization. See section 4.7.1.

options: The options objects needed by the optimization function lsqnonlin. Some presets can be obtained from the function getFitOptions.

Optional input

fit:

- lb: Lower boundary values for the fit not to exceed, with respect to bestGuess. The lower boundaries for temperature, laser Guassian/Lorentzian line widths, and scale are set to slightly over zero if they are set to zero or less.
- ub: Upper boundary values for the fit not to exceed, with respect to bestGuess.

specParams:

- `resFactor`, `Z`, `limit`: These parameters are forwarded to the function `excitationSpec`. See section 4.5.6.

`collParam`:

- `colls`, `gas` and `P` are the parameters to calculate the collisional broadening and quenching rate. These are forwarded to the function `excSpecFitCost`. See section 4.7.2.

Output

`solutionVals`: The solution result for the optimization problem. This is vector with same variable positions as `bestGuess`.

`residue`: The normalized residual for the fit function returned by `lsqnonlin`, $\sum \text{cost}^2$.

Related

`excSpecFitCost`, `lsqnonlin`, `fitImageDataset`, `collisions`, `quenchRate`, `collisionalBroadening`, `getFitOptions`, `fitExcSpectrumImage.mlx`, `fitExcSpectrumSingle.mlx`

4.7.5 Function 'getFitOptions', retrieves options preset for fit optimization

This function stores preset for fitting parameters for the function `lsqnonlin` that is used for fitting the spectra in LIFSim. The user may add here more presets as needed.

Syntax

```
options = getFitOptions(opts)
```

Input

`opts`: The name of the options preset to be returned.

Output

`options`: The options object for fitting functions.

Related

`fitExcitationSpec`, `lsqnonlin`, `fitImageDataset`, `fitExcSpectrumImage.mlx`, `fitExcSpectrumSingle.mlx`

4.7.6 Function 'lineShiftCost', cost function for using in line shift optimization

This function uses the cost function `excSpecFitCost` for calculating the difference between a measured and a simulated spectrum with given parameters. However, it uses two extra arguments to adjust a single line position. These arguments are the index of the line in the array `linelist` and the new position in wavenumbers. This is necessary for the optimization function `linesShiftCorrection`.

Syntax

```
cost = lineShiftCost(wnum,measSpec,linelist,lineIndex,pos,params,MM,
varargin)
```

Input

`wnum`: Wavenumber region in cm^{-1}.

`measSpec`: The measured spectrum data point, with the same length as `wnum`.

`linelist`: The `linelist` obtained from function `selectLines`.

`bestGuess`: A matrix vector with seven places, which are the initial fit parameters to start the optimization. See section 4.7.1.

`lineIndex`: A matrix vector with the index of each line to be shifted in the `linelist` cell vector. The positions can be adjusted by specifying the new position through `pos`. The length of `pos` should be identical to `lineIndex`.

`pos`: The adjusted line position wavenumber in cm^{-1}.

`params`: A matrix vector with seven places, which are the initial fit parameters to start the optimization. See section 4.7.1.

`MM`: Molar mass of the species, obtained from function `molMass` or `selectLines`.

Optional input

`varargin`: Further variable arguments to forward to function `excSpecFitCost`. Use named argument cells as:

```
v = {'resFactor',12};
```

```
cost = lineShiftCost(wnum,measSpec,linelist,lineIndex,pos,params,MM,
v{:});
```

or

```
cost = lineShiftCost(wnum,measSpec,linelist,lineIndex,pos,params,MM,
'resFactor',12);
```

Output

`cost`: The difference between the measured and simulated spectra, with respect to the given parameters.

Related

`linesShiftCorrection`, `excSpecFitCost`, `lineStruct`, `selectLines`

4.7.7 Function 'linesShiftCorrection', line shift optimization

This function optimizes the individual line positions. The lines in the array `linelist` will be looped over and each corresponding position will be adjusted based on a fit with the line position as free parameter. If no indices are specified, all lines in the array `linelist` will be shifted if needed.

Furthermore, a boundary can be set as well to limit the shifting of lines. The variable of `nu0` will be adjusted for each line in the array `linelist` and this array will be returned.

Syntax

```
linelist = linesShiftCorrection(linelist, wnum, measSpec, fitParams,
MM, varargin, opts)
```

Input

`linelist`: The `linelist` obtained from function `selectLines`.

`wnum`: Wavenumber region in cm^{-1}.

`measSpec`: The measurement data points, which should have the same length as `wnum`.

`fitParams`: A matrix vector with seven places, which are the initial fit parameters to start the optimization. See section 4.7.1.

`MM`: Molar mass in g/mol.

Optional input

`fitRegions`: A matrix of size `(n,2)` defining the regions to consider for position fitting. Each row of this matrix defines a region to search for lines in it. E.g. A matrix, `fitRegions = [44415.4 44416.8; 44410.3 44412.3]`, will allow the function to adjust the positions of lines within 44415.4–44416.8 cm^{-1} and 44410.3–44412.3 cm^{-1}. If not specified, all lines will be considered.

`boundary`: The boundary for shifting the lines. This will be considered as upper and lower boundary around the line position `nu0` (default = 1 cm^{-1}).

`fitOpts`: The options objects needed by the optimization function `lsqnonlin`. Some presets can be obtained from the function `getFitOptions`.

`verbose`: A logical flag, if true, the progress and a final report for the fit details will be displayed. (default = false).

`varargin`: Further variable arguments to forward to function `excSpecFitCost`. Use named argument cell.

Output

`linelist`: The given array `linelist` with the adjusted variable `nu0` for each line.

Related

`lineShiftCost`, `excSpecFitCost`, `getFitOptions`, `lineStruct`, `selectLines`

4.7.8 Function 'fitExcImageDataset', multi-line LIF images thermometry

Spatially resolved excitation spectra, named as excitation image dataset, can be handled by this equation. This function relies on an initial position to start extracting the spectra from the image set and fitting. For optimal fit results, this initial position should have been already fitted and the fit results must be provided, if not the function will fit this initial spectrum and propagate further through the image. The pixel positions to handle after the processed positions are determined based on the previously fitted positions, here the function `getNextPixels` is utilized, see section 4.7.9.

Furthermore, a mask can be passed to this function to omit regions where no fit is required. This could be regions with low LIF intensity and regions outside the laser sheet. The number of simultaneous spectra to be fitted is determined by the number of available parallel workers. To use this function, configure the MATLAB parallel profile, as needed, by clicking the icon in the bottom left corner and then "Parallel Preferences".

Syntax

```
[fittedImage, fitResidue, timeElapsed, tempFig] = fitExcImageDataset(wnum, images3d, MM, linelist, startPos, startFit, mask, options, plotImage, varargin)
```

Input

`wnum`: Wavenumber region in cm^{-1}.

`images3d`: The LIF image set stored in a 3D matrix with dimensions as follows:

- x: Width of the image.
- y: Height of the image.
- z: The image index. The length along this coordinate should be equal to wnum. For an image, `idx`, the image is then `images3d(:,:,idx)` that corresponds to the excitation wavenumber `wnum(idx)`.

`MM`: Molar mass in g/mol.

`linelist`: The `linelist` obtained from function `selectLines`.

`startPos`: The (*y, x*) coordinates of the starting position in the image set. The spectrum at this position must be first fitted and the resulting parameters must be provided in `startFit`.

`startFit`: A matrix vector with seven places, which are the resulting fit parameters for `startPos`. See section 4.7.1.

`mask`: A logical mask with height and width matching those of `image3d`. The positions set to true in this matrix would be processed.

`options`: The options objects needed by the optimization function `lsqnonlin`. Some presets can be obtained from the function `getFitOptions`.

Optional input

`plotImage`: A logical variable to show the resulting temperature images as the fit propagates through the positions (default = true).

`bounds`: This defines the fit boundaries for the parameters in section 4.7.1. The boundaries here are not in percentage. The upper and lower boundaries are then set around the result fit parameters of previously nearest fit. For the initial fit they are calculated from the `startFit` vector.

`varargin`: Other arguments to this function are not processed and would be redirected to `fitExcitationSpec`.

Output

`fittedImage`: The resulting fit parameters are stored here. The height and width of this 3D matrix corresponds to `image3d`, and the z direction has the seven resulting fit parameters for each position.

`fitResidue`: A 2D matrix matching the height and width of `image3d`, storing the resulting residual of the fits.

`timeElapsed`: The duration of the fit in seconds.

`tempFig`: The figure object in case `plotImage` was set to true.

Related

`fitExcitationSpec`, `parfeval`, `getFitOptions`, `getNextPixels`, `fitExcSpectrumImage.mlx`, `sensitivityAnalysis.mlx`

4.7.9 Function 'getNextPixels', searches for nearest pixels to a given pixel

This function searches for the next target pixels to fit. It considers two binary matrices representing processed and masked pixels to identify neighboring pixels within the specified region of interest. It begins by isolating unprocessed pixels within the mask and those adjacent to processed ones using convolution with a Laplacian operator of a Gaussian function to detect the edge. If adjacent pixels are found, it calculates their nearest neighbors based on Euclidean distance. Otherwise, it locates the nearest unprocessed pixel within the mask. The output provides coordinates of the nearest pixel, for extracting the fit parameters.

Syntax

```
nearestPixels = getNextPixels(processed, mask, fetchLen)
```

Input

`processed`: A logic matrix indicating the processed pixels as true.

`mask`: A logic matrix indicating pixels to ignore as false.

`fetchLen`: The number of pixels to return.

Output

`nearestPixels`: The nearest pixels found.

Related

`fitExcImageDataset`, `getNearestPixel`, `distEuc`

4.7.10 Function 'excitationSpecSolution', calculates the excitation spectrum from a fit result

The function calculates the excitation spectrum based on a fit solution. Use this to facilitate calculating a spectrum from existing fit parameters. Unlike `excitationSpec`, it includes accounting for scale and baseline (those are in the argument `solutionParams`).

Syntax

```
spec = excitationSpecSolution(wnum, linelist, MM, solutionParams,
varargin)
```

Input

`wnum`: Wavenumber region in cm^{-1}.

`linelist`: The `linelist` obtained from function `selectLines`.

`MM`: Molar mass in g/mol.

`solutionParams`: A matrix vector with seven places, which are the resulting fit parameters. See section 4.7.1. Output of `fitExcitationSpec`.

Optional input

`varargin`: Other arguments to this function are not processed and would be redirected to `excitationSpec`.

Output

`spec`: The calculated excitation spectrum with the same length as `wnum`.

Related

`fitExcitationSpec`, `excitationSpec`, `fitExcSpectrumImage.mlx`, `sensitivityAnalysis.mlx`, `collisionalBroadening`, `quenchRate`, `semiQuanMoleFraction`

4.7.11 Function 'semiQuanMoleFraction', calculates mole fractions qualitatively

The semi quantitative mole fraction of a probed species can be inferred with this function, for an isolated transition, i to k. The number density N_k can be substituted with, $px/(kT)$, according to the ideal gas law. After omitting all constant terms in equation (5), the qualitative mole fraction x can be written as:

$$x \propto \frac{\sum_{\tilde{\nu}} S(\tilde{\nu}) \,/\, \sum_{\tilde{\nu}} \Gamma_{\tilde{\nu}}}{f_{\mathrm{B}}/T} \sum_{j} \frac{\sum_{j'} A_{kj'} + Q_k}{A_{kj} F(\tilde{\nu}^{em})} \tag{11}$$

Here, the overlap function is integrated along the excitation wavenumber. A qualitative mole fraction detection is valuable for a relative analysis, where concentrations at multiple conditions can be investigated and compared.

Syntax

```
x = semiQuanMoleFraction(wnum, solutionValues, line, Z, MM)
```

Input

`wnum`: Wavenumber region in cm^{-1}.

`solutionValues`: A matrix vector with seven places, which are the resulting fit parameters. See section 4.7.1. Output of `fitExcitationSpec`.

`line`: A struct, `lineStruct`, for the line to be used for this calculation.

`Z`: Partition function from `selectLines`.

`MM`: Molar mass in g/mol.

Output

`x`: The semi quantitative mole fraction.

Related

`fitExcitationSpec`, `lineStruct`, `selectLines`, `fluorTransm`, `excitationSpecSolution`, `semiQuantMoleFract.mlx`

References

[1] A. El Moussawi, T. Endres, S. Peukert, S. Zabeti, T. Dreier, M. Fikri, C. Schulz, Multi-line SiO fluorescence imaging in the flame synthesis of silica nanoparticles from $SiCl_4$, Combust. Flame 224, 260-272 (2021)
[2] A.B. McLean, C.E.J. Mitchell, D.M. Swanston, Implementation of an Efficient Analytical Approximation to the Voigt Function for Photoemission Lineshape Analysis, J. Electron. Spectrosc. Relat. Phenom. 69, 125-132 (1994)
[3] P.H. Paul, C.D. Carter, J.A. Gray, J.L.J. Durant, J.W. Thoman, M.R. Furlanetto, Correlations for the NO $A^2\Sigma^+$ electronic quenching cross-section, Office of Scientific and Technical Information (OSTI), 1995-01-01, 1995.