MATS

Release 2.1

Erin M. Adkins

Dec 03, 2021

CONTENTS

Erin M. Adkins

National Institute of Standards and Technology

Last Revision Dec 03, 2021

This software is available on GitHub and can be cited with the following DOI https://doi.org/10.18434/M32200

The purpose of the MATS project is to develop a NIST-based multi-spectrum fitting and analysis tool for spectroscopic data that allows the flexibility to test and adapt to experimental and data-driven needs. This software allows for the use of several commonly used spectroscopic line profiles (Voigt, Nelkin-Ghatak, speed-dependent Voigt, speed-dependent Nelkin-Ghatak, and Hartmann-Tran) and allows for pressure, temperature, and sample composition constraints to be imposed during fits. In addition to fitting experimental spectra, MATS can generate simulated spectra, which allows for its use as an error analysis tool.

CHAPTER

ONE

CONTENTS

1.1 Getting Started

1.1.1 Set-Up

The MATS package was developed and tested using the Anaconda python environment. Example scripts have typically been written and run using jupyter notebooks (run in jupyter lab), as this allows for code to be run in segments and for iteration on certain code segments. Any python package distribution should work with the code as long as the dependent packages are installed.

1.1.2 Main Packages

- MATS
- HITRAN Application Programming Interface (HAPI) The v1.1.1.0 was tested and is available in the MATS repository.

1.1.3 Dependent Packages

MATS is not written as a basic package, meaning that there are several dependent packages that need to be installed.

If a desired package is not installed then the following command will install it. Many python package distributions have integrated package managers and required packages can also be installed through that mechanism.

```
pip install package_name
```

There is commonly a delay in the most recent package releases available in python package distribution package managers compared to that available through pip install. The following command line script will update to the newest release if a package is already installed. This should only be necessary if there is an error when running MATS with the currently installed version.

```
python -m pip install --upgrade package_name --user
```

Below are a list of the packages used in MATS.

- numpy - python's fundamental scientific computing package
- pandas - python data structure package
- os, sys - system variables
- lmfit - non-linear least-squares minimization and curve-fitting for python

- matplotlib - python plotting
- seaborn - pretty plotting
- scipy.fftpack - provides fft functionality
- jupyter lab - web-based user interface for Project Jupyter

1.1.4 Install

Taken from the INSTALL.md file

Basic *MATS* installation instructions. Note that development is alpha. In the future, full pypi/conda installs will be made available.

Create a conda environment with MATS dependencies

Download the [environment](environment.yml) file. Note that without setting the environment name below, by default an evironment *MATS-env* will be created.

```
conda env create -n {optional-name-of-environment} -f environment.yml
```

Alternatively, we provide a conda metapackage *mats-dependencies*, which includes all dependencies for *MATS`* (excluding python). This can be installed using

```
conda install -n {optional-name-of-environment} -c wpk-nist mats-dependencies
```

From source

After cloning the repo, you can do the following.

```
pip install .
```

To install an editable version, use option *-e*. To exclude dependencies, use option *–no-deps*

With pip from github

This requires git also be installed. Downside is that the whole repo (including all examples) are clones.

```
pip install git+https://github.com/wpk-nist-gov/MATS.git@feature/master-reformat
```

With pip from github using wheel Note, this is experimental. Do the following

```
pip install https://raw.githubusercontent.com/wpk-nist-gov/MATS/feature/master-reformat/
↪wheel/MATS-3-py3-none-any.whl
```

(Note that the actual version will not be 3)

Alternatively, download the wheel and run

```
pip install path-to-wheel.whl
```

1.2 MATS Version Summary

1.2.1 MATS Version 2.1

published on 12/03/2021

Focus of version update is cleanup to make package git installable

- renamed python files to lowercase, as this is the preferred python style.
- added __init__.py to include the most important functionality at the top level
- reformatted constants. Added codata.py file with includes CODATA dictionary (with values and metadata) and CONSTANTS dictionary (with only values)
- Added linelistdata.py file to autoload csv files from MATS/LineList.
- Got rid of all *from package import* * commands. Use explicit imports only
- Reworked example notebooks to reflect updates.
- Added INSTALL.md file to explain the install options.

1.2.2 MATS Version 2

published on 6/10/2021

New Features

- Beta correction to the Dicke Narrowing accounting for the hardness of coliisions based on broadener and perturber.
- Added CIA functionality. Currently, can manually enter CIA for each file
- Added capability pass non-fitting columns through the fit definition. This will help for future addition of bandwide functions or for ease of transforming fit outputs to reported results.
- Add ability to weight the spectra
- Added beyond HITRAN molecule capabilities, including update of isotope list.
- Added preliminary instrument line shape functionality.

Bug Fixes

- Changed the structure of MATS, so that each class is in its own python file
- Changed all constant values to be consistent with CODATA values and hardwired the values to definition in the utility script
- Added warning for floating parameters with an initial guess equal to 0.
- Changed the etalon_freq variables to etalon_period, since that is how it is coded.
- Added checks for molecule consistency between dataset and parameter line list.
- Ability to simulate at infinite SNR
- Indexing error for the spectrum number in baseline parameters
- Changed initial baseline guess to 0

1.2.3 MATS Version 1

published on 12/27/19

1.3 MATS Summary

In multi-spectrum fitting, there are a collection of spectra modeled by the same line-by-line spectroscopic parameters, but each spectrum might vary in pressure, temperature, and sample composition.

The MATS program is based on *Spectrum* objects, which are defined not only by their wavenumber and absorption data, but also information on the spectrum pressure, temperature, baseline characteristics, and sample composition. In addition to utilizing real spectra, MATS has a *simulate_spectrum()* function, which returns a spectrum object calculated from input simulation parameters. This is useful for performing error analysis in the same framework as primary data analysis by simply switching from experimental to simulated *Spectrum* objects.

These objects are combined to form a *Dataset* object, which is the collection of spectra that are being analyzed together in the multi-spectrum analysis.

The analysis of spectra in MATS requires an inital spectroscopic linelist. Details about the format of this linelist and how to generate it using the HITRAN Application Programming Interface are outlined on the *Generating Parameter Line lists* page. A few example line lists have been provided in the Line list folder , which can be accessed using the *LoadLineListData()* helper function. It should be noted that these linelists are provided for use with the examples and to provide an example of line list formatting. These linelists should not be used as reference data.

There are two files that contain parameters that are fit in this model, one for spectrum dependent parameters (polynomial baseline parameters, etalons, sample composition, and x-shift term) and the other for line-by-line spectroscopic parameters that are common across all spectra. These files are saved as .csv files with a column for each parameter and with rows corresponding to either the spectrum number or spectral line number. In addition to the columns containing the values for the fit parameters, there are two additional columns for each fittable parameter called param_vary and param_err. The param_vary column is a boolean (True/False) flag that is toggled to indicate whether a given parameter will be varied in the fit. The param_err column will be set to zero initially and replaced with the standard error for the parameter determined by the fit results. Calls of the *Generate_FitParam_File* class not only make these input files, but also set the line shape and define whether a parameter should be varied in the fit and if a parameter should be constrained across all spectra or allowed to vary by spectrum.

Finally, the *Fit_DataSet* class fits the spectra. Additionally, it allows the user to impose constraints on the parameters (min and max values), impose convergence criteria, update background and parameter line lists, and plot fit results.

Below is the sparse documentation for each of the classes and main functions in the MATS project with links to the full documentation provided.

1.3.1 Spectrum Class and Objects

Spectrum(filename[, molefraction, ...])	Spectrum class provides all information describing experimental or simulated spectrum.
simulate_spectrum(parameter_linelist[, ...])	Generates a synthetic spectrum, where the output is a spectrum object that can be used in MATS classes.
Spectrum.calculate_QF()	Calculates the quality of fit factor (QF) for a spectrum - QF = (maximum alpha - minimum alpha) / std(residuals).

continues on next page

Table 1 – continued from previous page

Spectrum.fft_spectrum()	Takes the FFT of the residuals of the spectrum, generates a plot of frequency (cm-1) versus amplitude (ppm/cm), and prints a dataframe with the 20 highest amplitude frequencies with the FFT frequency (period), amplitude, FFT phase, and frequency (cm-1).
Spectrum.plot_freq_tau()	Generates a plot of tau (us) as a function of frequency (MHz).
Spectrum.plot_model_residuals()	Generates a plot of the alpha and model (ppm/cm) as a function of wavenumber (cm-1) and on lower plot shows the residuals (ppm/cm) as a function of wavenumber (cm-1).
Spectrum.plot_wave_alpha()	Generates a plot of alpha (ppm/cm) as a function of wavenumber (cm-1).
Spectrum.save_spectrum_info([save_file])	Saves spectrum information to a pandas dataframe with option to also save as as a csv file.
Spectrum.segment_wave_alpha()	Defines the wavenumber, alpha, and indices of spectrum that correspond to a given spectrum segment.

1.3.2 Line-by-line Model

The line-by-line model is based on the HTP code provided in the HITRAN Application Programming Interface (HAPI). For the most part the conventions and definitions used by HITRAN are used in the MATS program. However, for some of the advanced line profile parameters the naming convention and temperature dependence is different. In the sections below, the temperature and pressure dependence of the various parameters is outlined for clarity. Additionally, MATS uses the CODATA values for calculations.

The Hartmann-Tran profile limiting cases that correspond to several commonly used line profiles. These limiting cases are achieved by setting line shape parameters equal to 0. The list below indicates the parameters that are not fixed equal to zero in each of the HTP limiting case line shapes. For more information about the HTP see the following references: Recommended isolated-line profile for representing high-resolution spectroscopic transitions (IUPAC Technical Report) and An isolated line-shape model to go beyond the Voigt profile in spectroscopic databases and radiative transfer codes

Voigt Profile (VP): $\Gamma_D, \Gamma_0, \Delta_0$

Nelkin-Ghatak Profile (NGP): $\Gamma_D, \Gamma_0, \Delta_0, \nu_{VC}$

speed-dependent Voigt Profile (SDVP): $\Gamma_D, \Gamma_0, \Delta_0, \Gamma_2, \Delta_2$

speed-dependent Nelkin-Ghatak Profile (SDNGP): $\Gamma_D, \Gamma_0, \Delta_0, \nu_{VC}, \Gamma_2, \Delta_2$

Hartmann-Tran (HTP): $\Gamma_D, \Gamma_0, \Delta_0, \nu_{VC}, \Gamma_2, \Delta_2, \eta$

Line Intensity

The line intensity for each line at the experimental temperature is calculated using the EnvironmentDependency_Intensity function in HAPI. This function takes as arguments the line intensity at 296 K ($S(T_{ref})$), the experimental temperature (T), the reference temperature 296 K (T_{ref}), the partition function at the experimental temperature ($Q(T)$), the partition function at the reference temperature ($Q(T_{ref})$), the lower state energy (E"), and the line center ((ν)), and constant ($c2 = hc/k$). The partition functions are calculated using TIPS-2017. Constants are defined by CODATA values

$$S(T) = S(T_{ref})\frac{Q(T_{ref})}{Q(T)}\frac{e^{-c2E"/T}}{e^{-c2E"/T_{ref}}}\frac{1-e^{-c2\nu/T}}{1-e^{-c2\nu/T_{ref}}}$$

Doppler Broadening

In MATS, the doppler broadening (Γ_D)is not a floatable parameter and is calculated based on the experimental temperature (T), line center (ν), and molecular mass (m). Constants are defined by CODATA values. The doppler width is calculated as:

$$\Gamma_D = \sqrt{\frac{2kT \cdot ln(2)}{cMassMol \cdot mc^2}} \cdot \nu$$
$$k = 1.380648813x10^{-16} ergK^{-1}$$
$$cMassMol = 1.66053873x10^{-24} mol$$

Collisional Half-Width

The collisional half-width (Γ_0) is a function of both the experimental pressure (P) and temperature (T) referenced to P_{ref} (1 atm) and T_{ref} (296 K). The contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble collisional broadening. The temperature dependence is modeled as a power law, where n is the temperature exponent for the collisional width. The collisional half-width for each line at experimental temperature and pressure can be represented as:

$$\Gamma_0(P,T) = \sum abun_k(\Gamma_0^k * \frac{P}{P_{ref}} * (\frac{T_{ref}}{T})^{n_{\Gamma_0^k}})$$

In the MATS nomenclature, the collisional half-width is called gamma0_diluent and the temperature dependence of the collisional half-width is called n_gamma0_diluent.

Pressure Shift

Just like the collisional half-width, the pressure shift (Δ_0) is a function of both the experimental pressure (P) and temperature (T) referenced to P_{ref} (1 atm) and T_{ref} (296 K). The contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble pressure shift. Unlike the collisional half-width, the pressure shift has a linear temperature dependence, where n represents the temperature dependence of the pressure shift. The pressure shift for each line at experimental pressure and temperature can be represented as:

$$\Delta_0(P,T) = \sum abun_k(\Delta_0^k + n_{\Delta_0^k} \cdot (T - T_{ref}))\frac{P}{P_{ref}}$$

In the MATS nomenclature, the pressure shift is called delta0_diluent and the temperature dependence of the pressure shift is called n_delta0_diluent.

Speed-Dependent Broadening

The speed-dependent mechanism accounts for the speed-dependence of relaxation rates and is parameterized in the speed-dependent Voigt (SDVP), speed-dependent Nelkin-Ghatak (SDNGP), and Hartmann-Tran (HTP) profiles. The speed-dependent broadening in MATS is tabulated as the ratio $aw = \frac{\Gamma_2}{\Gamma_0}$, but the actual fitted parameter is Γ_2. The temperature dependence of the speed-dependent broadening is a power law dependence on Γ_2. Currently in HITRAN, it is assumed that the $n_{\Gamma_0} = n_{\Gamma_2}$, such that a_w is assumed to be temperature independent. The introduction of n_{Γ_2} as a parameter in MATS allows for the option of this assumption to imposed, but the flexibility to explore non-equivalent temperature dependences between the speed-dependent and collisional broadening terms. The contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble speed-dependent broadening.

$$\Gamma_2(P,T) = \sum abun_k(a_w^k * \Gamma_0^k * \frac{P}{P_{ref}} * (\frac{T_{ref}}{T})^{n_{\Gamma_2^k}})$$

In the MATS nomenclature, the ratio of the speed-dependent broadening to the collisional broadening (a_w) is called SD_gamma_diluent and the temperature dependence of the speed-dependent broadening is called n_gamma2_diluent. The difference in the naming structure (SD_gamma vs gamma2) is chosen to emphasize the difference between the speed-dependent width being parameterized as a ratio versus as an absolute value.

Speed-Dependent Shifting

The speed-dependent mechanism accounts for the speed-dependence of relaxation rates and is parameterized in the speed-dependent Voigt (SDVP), speed-dependent Nelkin-Ghatak (SDNGP), and Hartmann-Tran (HTP) profiles. The speed-dependent shift in MATS is tabulated as the ratio $as = \frac{\Delta_2}{\Delta_0}$, but the actual fitted parameter is Δ_2. The temperature dependence of the speed-dependent shift is modeled with a linear dependence. Currently, the temperature dependence of the speed-dependent shift is not parameterized in HITRAN. The contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble speed-dependent shift.

$$\Delta_2(P,T) = \sum abun_k(a_s \cdot \Delta_0^k + n_{\Delta_2^k} \cdot (T - T_{ref}))\frac{P}{P_{ref}}$$

In MATS nomenclature, the ratio of the speed-dependent shift to the pressure shift (a_s) is called SD_shift_diluent and the temperature dependence of the speed-dependent shift is called n_delta2_diluent. The difference in the naming structure (SD_delta vs delta2) is chosen to emphasize the difference between the speed-dependent shift being parameterized as a ratio versus as an absolute value.

Dicke Narrowing

The Dicke narrowing mechanism models collisional induced velocity changes and is parameterized in the Nelkin-Ghatak (NGP), speed-dependent Nelkin-Ghatak (SDNGP), and Hartmann-Tran (HTP) profiles by the term ν_{VC}. The temperature dependence is modeled as a power law, where n represents the temperature dependence of the Dicke narrowing term. If the Dicke narrowing is assumed to behave like the diffusion coefficient, then the temperature dependence theoretically should be 1. The contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble Dicke narrowing.

$$\nu_{VC}(P,T) = \sum_{k=i} abun_k(\nu_{VC}^k * \frac{P}{P_{ref}} * (\frac{T_{ref}}{T})^{n_{\nu VC^k}})$$

In MATS nomenclature, the Dicke narrowing is referred to as nuVC_diluent and the temperature exponent is n_nuVC_diluent. This differs from the naming convention in HAPI, which changes based on the origin of the Dicke narrowing term (Galatry profile versus HTP). For simplicity, MATS has adopted a self-consistent naming convention.

Correlation Parameter

The correlation parameter (η) models the correlation between velocity and rotation state changes due to collisions and is only parameterized in the Hartmann-Tran profile (HTP). Currently, MATS has no temperature or pressure dependence associated with the correlation parameter. However, contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble correlation parameter.

$$\eta(k) = \sum abun_k \cdot \eta$$

In MATS nomenclature, the correlation parameter is referred to as eta_diluent.

Line Mixing

The first order line mixing (Y) can be calculated from the imaginary portion of any of the HTP derivative line profiles. Currently, there is no temperature dependence imposed on the line mixing, so there a different value is used for each nominal temperature, where the nominal temperature is specified in the *Spectrum* definition. However, contributions from each diluent (k) can be scaled by the diluent composition fraction ($abun$) and summed to model the ensemble line mixing.

$$Y(P) = \sum abun_k Y \frac{P}{P_{ref}}$$

The line mixing is implemented as:

$$\alpha = I * (ReHTP(\Gamma_D, \Gamma_0, \Delta_0, \Gamma_2, \Delta_2, \nu_{VC}, \eta, \nu) + Y * ImHTP(\Gamma_D, \Gamma_0, \Delta_0, \Gamma_2, \Delta_2, \nu_{VC}, \eta, \nu))$$

In MATS nomenclature, the line mixing parameter is referred to as y_diluent_nominaltemperature.

Line-by-line Models

HTP_from_DF_select(linelist, waves[, ...])	Calculates the absorbance (ppm/cm) based on input line list, wavenumbers, and spectrum environmental parameters.
HTP_wBeta_from_DF_select(linelist, waves[, ...])	Calculates the absorbance (ppm/cm) based on input line list, wavenumbers, and spectrum environmental parameters with capability of incorporating the beta correction to the Dicke Narrowing proposed in Analytical-function correction to the Hartmann–Tran profile for more reliable representation of the Dicke-narrowed molecular spectra.

1.3.3 Dataset Class

Dataset(spectra, dataset_name, param_linelist)	Combines spectrum objects into a Dataset object to enable multi-spectrum fitting.
Dataset.generate_baseline_paramlist()	Generates a csv file called dataset_name + _baseline_paramlist, which will be used to generate another csv file that is used for fitting spectrum dependent parameters with columns for spectrum number, segment number, x_shift, concentration for each molecule in the dataset, baseline terms (a = 0th order term, b = 1st order, etc), and etalon terms (set an amplitude, period, and phase for the number of etalons listed for each spectrum in the Dataset).
Dataset.generate_summary_file([save_file])	Generates a summary file combining spectral information from all spectra in the Dataset.
Dataset.get_spectra_extremes()	
Dataset.get_spectrum_extremes()	Gets the minimum and maximum wavenumber for the entire Dataset.
Dataset.average_QF()	Calculates the Average QF from all spectra.

continues on next page

Table 3 – continued from previous page

Dataset.plot_model_residuals()	Generates a plot showing both the model and experimental data as a function of wavenumber in the main plot with a subplot showing the residuals as function of wavenumber.

1.3.4 Generate FitParam File Class

Generate_FitParam_File(dataset, . . . [, . . .])	Class generates the parameter files used in fitting and sets up fit settings.
Generate_FitParam_File.generate_fit_baseline_linelist([. . .])	Generates the baseline line list used in fitting and updates the fitting booleans to desired settings.
Generate_FitParam_File.generate_fit_param_linelist_from_linelist([. . .])	Generates the parameter line list used in fitting and updates the fitting booleans to desired settings.

1.3.5 Fit DataSet Class

Fit_DataSet(dataset, base_linelist_file, . . .)	Provides the fitting functionality for a Dataset.
Fit_DataSet.constrained_baseline(params[, . . .])	Imposes baseline constraints when using multiple segments per spectrum, ie all baseline parameters can be the same across the entire spectrum except for the etalon phase, which is allowed to vary per segment.
Fit_DataSet.fit_data(params[, wing_cutoff, . . .])	Uses the lmfit minimizer to do the fitting through the simulation model function.
Fit_DataSet.generate_params()	Generates the lmfit parameter object that will be used in fitting.
Fit_DataSet.residual_analysis(result[, . . .])	Updates the model and residual arrays in each spectrum object with the results of the fit and gives the option of generating the combined absorption and residual plot for each spectrum.
Fit_DataSet.simulation_model(params[, . . .])	This is the model used for fitting that includes baseline, resonant absorption, and CIA models.
Fit_DataSet.update_params(result[, . . .])	Updates the baseline and line parameter files based on fit results with the option to write over the file (default) or save as a new file and updates baseline values in the spectrum objects.

1.3.6 Support Modules

etalon(x, amp, period, phase)	Etalon definition
molecularMass(M, I[, isotope_list])	molecular mass look-up based on the HAPI definition adapted to allow used to specify ISO list INPUT PARAMETERS: M: HITRAN molecule number I: HITRAN isotopologue number OUTPUT PARAMETERS: MolMass: molecular mass — DESCRIPTION: Return molecular mass of HITRAN isotolopogue.

continues on next page

Table 6 – continued from previous page

isotope_list_molecules_isotopes([isotope_list])	The HITRAN style isotope list in the format (M,I), this function creates a dictionary from this with M as the keys and lists of I as values.
add_to_HITRANstyle_isotope_list([...])	Allows for used to add to an existing isotope line list in the HITRAN format

LoadLineListData([paths])	Helper class to read in supplied LineList DataFrames

1.4 Generating Parameter Line lists

1.4.1 Parameter Line list Overview

The MATS program uses a spectroscopic line list that varies from the traditional HITRAN and HAPI formats to accomodate th

- molec_id: HITRAN molecule id number
- local_iso_id: HITRAN local isotope id number
- nu: line center (cm^{-1})
- sw : spectral line intensity ($\frac{cm^{-1}}{molecule \cdot cm^{-2}}$) at Tref=296K
- elower: The lower-state energy of the transition (cm^{-1})
- gamma0_diluent: collisional half-width ($\frac{cm^{-1}}{atm}$) at Tref=296K and reference pressure pref=1atm for a given diluent
- n_gamma0_diluent: coefficient of the temperature dependence of the collisional half-width
- delta0_diluent: pressure shift ($\frac{cm^{-1}}{atm}$) at Tref=296K and pref=1atm of the line position with respect to line center
- n_delta0_diluent: the coefficient of the temperature dependence of the pressure shift
- SD_gamma_diluent: the ratio of the speed dependent width to the collisional half-width at reference temperature and pressure
- n_gamma2_diluent: the coefficient of the temperature dependence of the speed-dependent width NOTE: This is the temperature dependence of the speed-dependent width not the temperature dependence of the ratio of the speed-dependent width to the collisional half-width
- SD_delta_diluent: the ratio of the speed-dependent shift to the collisional shift at reference temperature and pressure
- n_delta2_diluent: the coefficient of the temperature dependence of the speed-dependent shift NOTE: This is the temperature dependence of the speed-dependent shift not the temperature dependence of the ratio of the speed-dependent shift to the pressure shift
- nuVC_diluent: Dicke narrowing term ($\frac{cm^{-1}}{atm}$) at reference temperature and pressure
- n__nuVC_diluent: coefficient of the temperature dependence of the Dicke narrowing term
- eta_diluent: correlation parameter
- y_diluent_nominaltemp: line mixing term ($\frac{cm^{-1}}{atm}$) NOTE: As currently written , this doesn't have a temperature dependence, so there is a different column for each nominal temperature.

1.4.2 Generating Parameter Line list from HITRAN

The parameter line list .csv file can be generated manually, but this notebook generates a parameter list using the HITRAN Application Programming Interface (HAPI). The overview below walks through this notebook.

Imports

Import the numpy, pandas, os, and sys packages. Set pandas dataframe to show all of the rows.

```
import numpy as np
import pandas as pd
pd.set_option("display.max_rows", 101)
import os, sys
```

Optional imports include matplotlib and seaborn for plotting.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
sns.set_style("ticks")
sns.set_context("poster")
```

Molecule and Isotope Information

HAPI provides a dictionary called ISO_ID, where the global isotope id acts as the key, with a sub-dictionary containing the molecule id, local isotope id, isotope name, relative abundance, mass, and molecule name. The following command will generate the dictionary as the output. This provides the necessary global isotope id information for interfacing with HAPI to access HITRAN infomation.

```
print_iso_id()
```

```
The dictionary "ISO_ID" contains information on "global" IDs of isotopologues in HITRAN

   id            M    I                    iso_name       abundance       mass       ␣
↪mol_name
    1     :      1    1                     H2(16O)    0.9973170000  18.010565          ␣
↪  H2O
    2     :      1    2                     H2(18O)    0.0019998300  20.014811          ␣
↪  H2O
    3     :      1    3                     H2(17O)    0.0003720000  19.014780          ␣
↪  H2O
    4     :      1    4                      HD(16O)    0.0003106900  19.016740          ␣
↪  H2O
    5     :      1    5                      HD(18O)    0.0000006230  21.020985          ␣
↪  H2O
    6     :      1    6                      HD(17O)    0.0000001160  20.020956          ␣
↪  H2O
  129     :      1    7                      D2(16O)    0.0000000242  20.022915          ␣
↪  H2O
    7     :      2    1                 (12C)(16O)2    0.9842040000  43.989830          ␣
↪  CO2
    8     :      2    2                 (13C)(16O)2    0.0110570000  44.993185          ␣
↪  CO2
```

(continues on next page)

(continued from previous page)

```
    9     :     2    3          (16O)(12C)(18O)    0.0039471000  45.994076
↪  CO2
   10     :     2    4          (16O)(12C)(17O)    0.0007340000  44.994045
↪  CO2
   11     :     2    5          (16O)(13C)(18O)    0.0000443400  46.997431
↪  CO2
   12     :     2    6          (16O)(13C)(17O)    0.0000082500  45.997400
↪  CO2
   13     :     2    7              (12C)(18O)2    0.0000039573  47.998322
↪  CO2
   14     :     2    8          (17O)(12C)(18O)    0.0000014700  46.998291
↪  CO2
  121     :     2    9              (12C)(17O)2    0.0000001368  45.998262
↪  CO2
   15     :     2   10              (13C)(18O)2    0.0000000450  49.001675
↪  CO2
  120     :     2   11          (18O)(13C)(17O)    0.0000000165  48.001650
↪  CO2
  122     :     2   12              (13C)(17O)2    0.0000000015  47.001618
↪  CO2
   16     :     3    1                  (16O)3    0.9929010000  47.984745
↪   O3
   17     :     3    2          (16O)(16O)(18O)    0.0039819400  49.988991
↪   O3
   18     :     3    3          (16O)(18O)(16O)    0.0019909700  49.988991
↪   O3
   19     :     3    4          (16O)(16O)(17O)    0.0007400000  48.988960
↪   O3
   20     :     3    5          (16O)(17O)(16O)    0.0003700000  48.988960
↪   O3
   21     :     4    1              (14N)2(16O)    0.9903330000  44.001062
↪  N2O
   22     :     4    2          (14N)(15N)(16O)    0.0036409000  44.998096
↪  N2O
   23     :     4    3          (15N)(14N)(16O)    0.0036409000  44.998096
↪  N2O
   24     :     4    4              (14N)2(18O)    0.0019858200  46.005308
↪  N2O
   25     :     4    5              (14N)2(17O)    0.0003690000  45.005278
↪  N2O
   26     :     5    1               (12C)(16O)    0.9865400000  27.994915
↪   CO
   27     :     5    2               (13C)(16O)    0.0110800000  28.998270
↪   CO
   28     :     5    3               (12C)(18O)    0.0019782000  29.999161
↪   CO
   29     :     5    4               (12C)(17O)    0.0003680000  28.999130
↪   CO
   30     :     5    5               (13C)(18O)    0.0000222200  31.002516
↪   CO
   31     :     5    6               (13C)(17O)    0.0000041300  30.002485
↪   CO
```

(continues on next page)

(continued from previous page)

```
   32     :      6    1                (12C)H4    0.9882700000  16.031300
↪  CH4
   33     :      6    2                (13C)H4    0.0111000000  17.034655
↪  CH4
   34     :      6    3               (12C)H3D    0.0006157500  17.037475
↪  CH4
   35     :      6    4               (13C)H3D    0.0000049203  18.040830
↪  CH4
   36     :      7    1                  (16O)2    0.9952620000  31.989830
↪   O2
   37     :      7    2              (16O)(18O)    0.0039914100  33.994076
↪   O2
   38     :      7    3              (16O)(17O)    0.0007420000  32.994045
↪   O2
   39     :      8    1              (14N)(16O)    0.9939740000  29.997989
↪   NO
   40     :      8    2              (15N)(16O)    0.0036543000  30.995023
↪   NO
   41     :      8    3              (14N)(18O)    0.0019931200  32.002234
↪   NO
   42     :      9    1            (32S)(16O)2    0.9456800000  63.961901
↪  SO2
   43     :      9    2            (34S)(16O)2    0.0419500000  65.957695
↪  SO2
   44     :     10    1            (14N)(16O)2    0.9916160000  45.992904
↪  NO2
   45     :     11    1                (14N)H3    0.9958715000  17.026549
↪  NH3
   46     :     11    2                (15N)H3    0.0036613000  18.023583
↪  NH3
   47     :     12    1           H(14N)(16O)3    0.9891100000  62.995644
↪ HNO3
  117     :     12    2           H(15N)(16O)3    0.0036360000  63.992680
↪ HNO3
   48     :     13    1                 (16O)H    0.9974730000  17.002740
↪   OH
   49     :     13    2                 (18O)H    0.0020001400  19.006986
↪   OH
   50     :     13    3                 (16O)D    0.0001553700  18.008915
↪   OH
   51     :     14    1                  H(19F)    0.9998442500  20.006229
↪   HF
  110     :     14    2                  D(19F)    0.0001150000  21.012505
↪   HF
   52     :     15    1                 H(35Cl)    0.7575870000  35.976678
↪  HCl
   53     :     15    2                 H(37Cl)    0.2422570000  37.973729
↪  HCl
  107     :     15    3                 D(35Cl)    0.0001180050  36.982954
↪  HCl
  108     :     15    4                 D(37Cl)    0.0000377350  38.980004
↪  HCl
```

(continues on next page)

(continued from previous page)

```
   54     :     16    1                  H(79Br)    0.5067800000  79.926160
↪  HBr
   55     :     16    2                  H(81Br)    0.4930600000  81.924115
↪  HBr
  111     :     16    3                   D(79Br)    0.0000582935  80.932439
↪  HBr
  112     :     16    4                   D(81Br)    0.0000567065  82.930392
↪  HBr
   56     :     17    1                  H(127I)    0.9998442500 127.912297
↪   HI
  113     :     17    2                   D(127I)    0.0001150000 128.918575
↪   HI
   57     :     18    1              (35Cl)(16O)    0.7559100000  50.963768
↪  ClO
   58     :     18    2              (37Cl)(16O)    0.2417200000  52.960819
↪  ClO
   59     :     19    1          (16O)(12C)(32S)    0.9373900000  59.966986
↪  OCS
   60     :     19    2          (16O)(12C)(34S)    0.0415800000  61.962780
↪  OCS
   61     :     19    3          (16O)(13C)(32S)    0.0105300000  60.970341
↪  OCS
   62     :     19    4          (16O)(12C)(33S)    0.0105300000  60.966371
↪  OCS
   63     :     19    5          (18O)(12C)(32S)    0.0018800000  61.971231
↪  OCS
   64     :     20    1             H2(12C)(16O)    0.9862400000  30.010565
↪ H2CO
   65     :     20    2             H2(13C)(16O)    0.0110800000  31.013920
↪ H2CO
   66     :     20    3             H2(12C)(18O)    0.0019776000  32.014811
↪ H2CO
   67     :     21    1              H(16O)(35Cl)    0.7557900000  51.971593
↪ HOCl
   68     :     21    2              H(16O)(37Cl)    0.2416800000  53.968644
↪ HOCl
   69     :     22    1                   (14N)2    0.9926874000  28.006147
↪   N2
  118     :     22    2                (14N)(15N)    0.0072535000  29.997989
↪   N2
   70     :     23    1              H(12C)(14N)    0.9851100000  27.010899
↪  HCN
   71     :     23    2              H(13C)(14N)    0.0110700000  28.014254
↪  HCN
   72     :     23    3              H(12C)(15N)    0.0036217000  28.007933
↪  HCN
   73     :     24    1          (12C)H3(35Cl)    0.7489400000  49.992328
↪CH3Cl
   74     :     24    2          (12C)H3(37Cl)    0.2394900000  51.989379
↪CH3Cl
   75     :     25    1                H2(16O)2    0.9949520000  34.005480
↪ H2O2
```

(continues on next page)

(continued from previous page)

```
   76     :     26    1                (12C)2H2    0.9776000000  26.015650  ␣
↪ C2H2
   77     :     26    2              (12C)(13C)H2  0.0219700000  27.019005  ␣
↪ C2H2
  105     :     26    3                (12C)2HD    0.0003045500  27.021825  ␣
↪ C2H2
   78     :     27    1                (12C)2H6    0.9769900000  30.046950  ␣
↪ C2H6
  106     :     27    2             (12C)H3(13C)H3 0.0219526110  31.050305  ␣
↪ C2H6
   79     :     28    1                 (31P)H3    0.9995328300  33.997238  ␣
↪  PH3
   80     :     29    1           (12C)(16O)(19F)2 0.9865400000  65.991722  ␣
↪ COF2
  119     :     29    2           (13C)(16O)(19F)2 0.0110834000  66.995083  ␣
↪ COF2
  126     :     30    1              (32S)(19F)6   0.9501800000 145.962492  ␣
↪  SF6
   81     :     31    1                 H2(32S)    0.9498800000  33.987721  ␣
↪  H2S
   82     :     31    2                 H2(34S)    0.0421400000  35.983515  ␣
↪  H2S
   83     :     31    3                 H2(33S)    0.0074980000  34.987105  ␣
↪  H2S
   84     :     32    1          H(12C)(16O)(16O)H 0.9838980000  46.005480  ␣
↪HCOOH
   85     :     33    1                 H(16O)2    0.9951070000  32.997655  ␣
↪  HO2
   86     :     34    1                   (16O)    0.9976280000  15.994915  ␣
↪   O
   87     :     36    1                (14N)(16O)+ 0.9939740000  29.997989  ␣
↪  NOp
   88     :     37    1                H(16O)(79Br) 0.5056000000  95.921076  ␣
↪ HOBr
   89     :     37    2                H(16O)(81Br) 0.4919000000  97.919027  ␣
↪ HOBr
   90     :     38    1                (12C)2H4    0.9773000000  28.031300  ␣
↪ C2H4
   91     :     38    2             (12C)H2(13C)H2 0.0219600000  29.034655  ␣
↪ C2H4
   92     :     39    1              (12C)H3(16O)H 0.9859300000  32.026215  ␣
↪CH3OH
   93     :     40    1              (12C)H3(79Br) 0.5013000000  93.941811  ␣
↪CH3Br
   94     :     40    2              (12C)H3(81Br) 0.4876600000  95.939764  ␣
↪CH3Br
   95     :     41    1           (12C)H3(12C)(14N) 0.9748200000  41.026549  ␣
↪CH3CN
   96     :     42    1              (12C)(19F)4    0.9893000000  87.993616  ␣
↪  CF4
  116     :     43    1                (12C)4H2    0.9559980000  50.015650  ␣
↪ C4H2
```

(continues on next page)

(continued from previous page)

```
  109      :     44    1                 H(12C)3(14N)    0.9646069000  51.010899
↪ HC3N
  103      :     45    1                          H2    0.9996880000   2.015650
↪   H2
  115      :     45    2                          HD    0.0003114320   3.021825
↪   H2
   97      :     46    1                 (12C)(32S)    0.9396240000  43.971036
↪   CS
   98      :     46    2                 (12C)(34S)    0.0416817000  45.966787
↪   CS
   99      :     46    3                 (13C)(32S)    0.0105565000  44.974368
↪   CS
  100      :     46    4                 (12C)(33S)    0.0074166800  44.970399
↪   CS
  114      :     47    1                 (32S)(16O)3    0.9423964000  79.956820
↪  SO3
  123      :     48    1                 (12C)2(14N)2    0.9707524330  52.006148
↪ C2N2
  124      :     49    1           (12C)(16O)(35Cl)2    0.5663917610  97.932620
↪COCl2
  125      :     49    2      (12C)(16O)(35Cl)(37Cl)    0.3622352780  99.929670
↪COCl2
```

Generate HITRAN and Initial Guess Line lists from HAPI Call

The next section of the example contains a function and function call where the output is a MATS compatible line list. The HITRANlinelist_to_csv takes a list of global isotope numbers and minimum and maximum wavenumbers as variables with a tablename, filename, temperature, and option to calculate the speed-dependent broadening as optional parameters. The spectroscopic data for the isotopes in the global isotope list over the specified wavenumber range will be retrieved from HITRANOnline. HITRAN breaks-up the HTP line parameters into temperature regimes, so the temperature specied selects for the most relevant parameters. The tablename parameter sets the ame of the table generated by the HAPI call, where the filename parameter sets the base for the resulting .csv files. The HITRANlinelist_to_csv function generates two outputs, the first is a HITRAN line list with all data available in HITRAN for the isotopes and spectral range (based on the parsed parameters in the HITRAN_parameter_list). The second file generates the highest order line shape list using the HITRAN values and formats for MATS. This line list also will fill in temperature dependences and missing broadener information and if calculate_aw is True calculate the speed dependence based on theory.

```
def HITRANlinelist_to_csv(isotopes, minimum_wavenumber, maximum_wavenumber, tablename =
↪'tmp', temperature = 296, calculate_aw = False):

                """Generates two .csv files generated information available from HTIRAN.
↪  The first line list matches the information available from HITRAN (_HITRAN.csv) and␣
↪the second supplements the HITRAN information with theoretical values and translates␣
↪into MATS input format (_initguess.csv)

 Outline

 1. Gets a line list from HITRAN and saves all available parameters to filename_HITRAN.
↪csv
 2. Goes through the data provided from HITRAN and collects the highest order line shape␣
↪information.
```

(continues on next page)

(continued from previous page)

```
 3.  Where there is missing information for the complete HTP linelist set to 0 or make␣
↪the following substitutions
     - for missing diluent information fill values with air
     - set missing shift temperature dependences equal to 0 (linear temperature␣
↪dependence)
     - calculate the SD_gamma based on theory (if calculate aw = True)
     - set the gamma_2 temperature exponent equal to the gamma0 temperature exponent
     - set the delta_2 temperature exponent equal to the delta0 temperature exponent
     - set the dicke narrowing temperature exponent to 1
 4. Save the supplemented and MATS formatted HITRAN information as filename_initguess.csv


 Parameters
 ----------
 isotopes : list
     list of the HITRAN global isotope numbers to include in the HAPI call
 minimum_wavenumber : float
     minimum line center (cm-1) to include in the HAPI call.
 maximum_wavenumber : float
     maximum line center (cm-1) to include in the HAPI call.
 tablename : str, optional
     desired name for table generated from HAPI call. The default is 'tmp'.
 temperature : float, optional
     Nominal temperature of interest.  HITRAN breaks-up the HTP line parameters into␣
↪temperature regimes.  This allows for selection of the most approriate parameter␣
↪information. The default is 296.
 calculate_aw : float, optional
     Boolean flag to present option to calculate speed-dependent shift based on␣
↪theoretical approximation based on temperature exponent, mass of the absorber, and␣
↪mass of the perturber

 Returns
 -------
 linelist_select : dataframe
     pandas dataframe corresponding to the HITRAN information supplemented by␣
↪theoretical values/assumptions.
 tablename_HITRAN.csv : .csv file
     file corresponding to available HITRAN information
 tablename_initguess.csv : .csv file
     file corresponding to available HITRAN information supplemented by theory and␣
↪assumptions in MATS format

 """
```

Use Example for HITRANlinelist_to_csv

To select the relevant information from HITRAN you will need to provide:

- table name (str)
- an array containing the global isotope numbers of the molecules/isotopes of interest
- the minimum and maximum wavenumbers
- the minimum line intensity of interest

The example below would generate a HITRAN table named 'CO' that contains all CO isotopes (global isotopes 26 - 31) and the most abundant CO_2 isotope (global isotope 7) in the spectral region between 6200 and 6500 cm^{-1} that have line intensities greater than 1e-30 $\frac{cm^{-1}}{molecule \cdot cm^{-2}}$. Additionally, it calculates the theoretical aw value and assumes that measurements are at 296 (temperature = 296 is default).

```
tablename = 'CO'
global_isotopes = [26, 27, 28, 29, 30,31,7]
wave_min = 6200
wave_max = 6500
intensity_cutoff = 1e-30

linelist_select = (HITRANlinelist_to_csv(global_isotopes, wave_min, wave_max, tablename␣
↪= tablename, calculate_aw = True))
```

1.5 Examples

1.5.1 MATS Example Scripts

The MATS package includes examples highlighting major features.

Fitting Experimental Spectra

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

This example fits isolated Oxygen A-Band transitions in experimental spectra

Import Modules and Set-Up

This example starts with importing modules and setting up file locations

```
import numpy as np
import pandas as pd
import os, sys
import matplotlib.pyplot as plt
from matplotlib import gridspec
%matplotlib inline
import MATS
```

Optional import of seaborn package for figure generation

```
import seaborn as sns
sns.set_style("whitegrid")
sns.set_style("ticks")
sns.set_context("poster")
```

If not working within the MATS file structure, change the path to the working directory that contains experimental spectra or to the folder that you want to work in.

```
os.chdir(path)
```

Load Spectra from files

There are two options for generating *Spectrum* objects. The first is from a file by instantiating an instance of the class, which is the focus of this example. The second option is by using the *simulate_spectrum()* function described in the fitting synthetic spectra example.

Before generating *Spectrum* objects from your experimental data, it is helpful to set some variables for terms that will be used in all of the *Spectrum* objects and/or throughout the fitting. In this example the minimum intensity threshold for simulation (IntensityThreshold), the minimum line intensity of lines fit in the analysis (Fit_Intensity), the order of the polynomial used in the baseline fits, and the names of columns used for the absorption, frequency, pressure, and temperature data are defined at the top of the example.

```
wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-24 #intensities must be above this value for the line to be fit
order_baseline_fit = 1
tau_column = 'Corrected Tau (us)' # Mean tau/us
freq_column = 'Total Frequency (Detuning)' # Total Frequency /MHz
pressure_column = 'Cavity Pressure /Torr'
temperature_column = 'Cavity Temperature Side 2 /C'
```

After that, 4 instances of the *Spectrum* class are instantiated from 4 experimental spectra. In the class instantiation, the mole fraction of the oxygen sample used is defined, the etalon amplitude and period are defined, the sample is confirmed to be at natural abundance, the diluent is set to air, and the columns defined for pressure, temperature, frequency, and absorbance data are set. All of these spectra were collected at nominally 296K, so the nominal temperature is set to be 296. This variable is only used for a flag as to whether to allow temperature exponents to be floatable parameters and for generation of line mixing parameters for each nominal temperature as the temperature dependence of line mixing is not currently implemented.

```
spec_1 = MATS.Spectrum('190510_2per_43_forfit',
                       molefraction = { 7 :0.01949}, natural_abundance = True, diluent =
↪'air',
                       etalons = {1:[0.001172, 1.19574]},
                       input_freq = True, frequency_column = freq_column,
                       input_tau = True, tau_column = tau_column, tau_stats_column = None,
                       pressure_column = pressure_column, temperature_column = temperature_
↪column,
                       nominal_temperature = 296, x_shift = 0.00)
spec_2 = MATS.Spectrum('190510_2per_55_forfit',
                       molefraction = { 7 : 0.01949}, natural_abundance = True, diluent =
↪'air',
                       etalons = {1:[0.001172, 1.19574]},
```

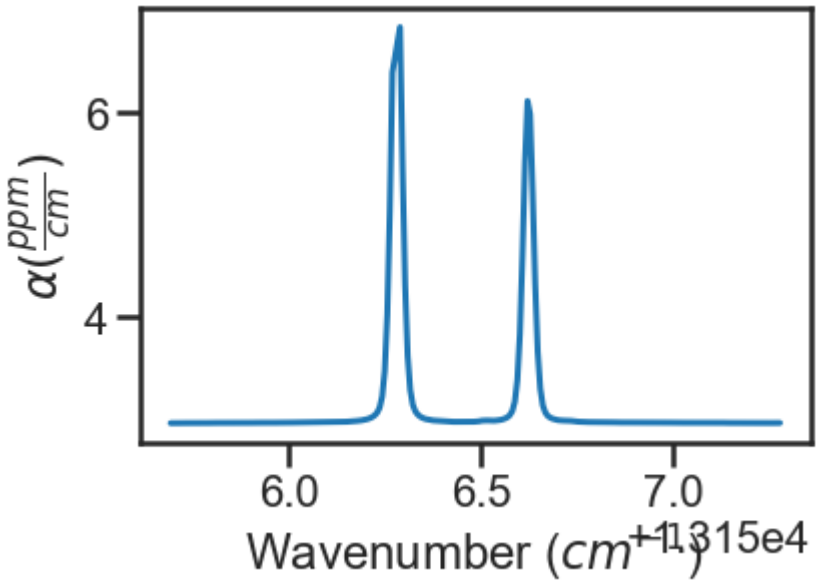
(continues on next page)

(continued from previous page)

```
                    input_freq = True, frequency_column = freq_column,
                    input_tau = True, tau_column = tau_column, tau_stats_column = None,
                    pressure_column = pressure_column, temperature_column = temperature_
↪column,
                    nominal_temperature = 296, x_shift = 0.00)
spec_3 = MATS.Spectrum('190513_2per_82_forfit',
                    molefraction = { 7 :0.01949}, natural_abundance = True, diluent =
↪'air',
                    etalons = {1:[0.001172, 1.19574]},
                    input_freq = True, frequency_column = freq_column,
                    input_tau = True, tau_column = tau_column, tau_stats_column = None,
                    pressure_column = pressure_column, temperature_column = temperature_
↪column,
                    nominal_temperature = 296, x_shift = 0.00)
spec_4 = MATS.Spectrum('190514_2per_126_forfit',
                    molefraction = { 7 :0.01949}, natural_abundance = True, diluent =
↪'air',
                    etalons = {1:[0.001172, 1.19574]},
                    input_freq = True, frequency_column = freq_column,
                    input_tau = True, tau_column = tau_column, tau_stats_column = None,
                    pressure_column = pressure_column, temperature_column = temperature_
↪column,
                    nominal_temperature = 296, x_shift = 0.00)
```

The *Spectrum.plot_wave_alpha()* function can be called to plot any of the spectra.

```
spec_1.plot_wave_alpha()
```



Generate a Dataset

If the parameter line list hasn't been read in from a .csv file, then do that now making sure to switch to the appropriate directories as needed. This file can be generated following the *Generating Parameter Line lists*. Alternatively, the code block below shows how to use the py:func:*LoadLineListData* function to read in the line list provided for the Oxygen A-Band.

```
from MATS.linelistdata import linelistdata
PARAM_LINELIST = linelistdata['O2_ABand_Drouin_2017_linelist']
```

The next step is to combine all desired `Spectrum` objects into a *`Dataset`* object, where we give the dataset a name and specify the initial parameter line list to use for the fits.

```
SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4], 'Line Intensity', baseline_
↪order = order_baseline_fit)
```

The *`Dataset`* class contains a function to generate a baseline line list, analogous to the one for the parameter line list (done outside of this example), based on the order of the baselines, etalons, molecules, x-shift parameters, and segments as defined by both the spectrum objects.

```
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()
```

Generate Fit Parameter Files

The next section of code uses the *`Generate_FitParam_File`* class to define what line shape to use for the initial fits, whether to use line mixing, the minimum line intensity to fit a line, mimimum intensity to included in the simulation, and for each line parameter whether that parameter is going to be constrained across all spectra or whether there will be a parameter for each spectrum (multi-spectrum vs single-spectrum fits) on a parameter by parameter basis. In the example below, the SDVP line profile without line mixing will be used to fit lines with line intensities greater than 1e-24 and the line centers and line intensities will be allowed to float for each line, while all other lines are constrained across all spectra in the dataset. The additional_columns parameter allows for inclusion of additional columns in the line shape parameter line list to be included in the output file.

```
FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,␣
↪lineprofile = 'SDVP', linemixing = False,
                                fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
                                nu_constrain = False, sw_constrain = False, gamma0_
↪constrain = True, delta0_constrain = True,
                                aw_constrain = True, as_constrain = True,
                                nuVC_constrain = True, eta_constrain =True, linemixing_
↪constrain = True,
                                additional_columns = ['trans_id', 'local_lower_quanta', 'm
↪'])
```

The next step is to generate fit parameter and baseline line lists that include columns that specify whether that parameter should be varied during fitting, in addition to adding uncertainty columns for the fit error for each parameter. For the following example the line centers, line intensities, collisional half-widths, and speed-dependent broadening terms will be floated for all main oxygen isotopes for lines where the line intensity is greater than 1e-24. Additionally, the baseline terms will float, as will the etalon amplitude and phase.

```
FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {7:{1:True, 2:False, 3:
↪False}}, vary_sw = {7:{1:True, 2:False, 3:False}},
                                                    vary_gamma0 = {7:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma0 = {7:{1:True}},
                                                    vary_delta0 = {7:{1: False, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_delta0 = {7:{1:True}},
                                                    vary_aw = {7:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma2 = {7:{1:False}},
                                                    vary_as = {}, vary_n_delta2 = {7:{1:
↪False}},
                                                    vary_nuVC = {7:{1:False}}, vary_n_nuVC␣
↪= {7:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {7:{1:
↪False}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = True, vary_molefraction = {7:
↪False, 1:False}, vary_xshift = False,
                                    vary_etalon_amp= True, vary_etalon_period= False,␣
↪vary_etalon_phase= True)
```

These functions will generate .csv files corresponding to these selections, which are read in by the `Fit_DataSet` class instantiation. This means that edits can be made manually to the .csv files or reading in the .csv, editing, and resaving before the next code segment is run.

Fit Dataset

Instantiating the *Fit_DataSet* class reads in the information from the baseline and parameter linelists generated in the previous step. It also allows for limits to be placed on the parameters, so that they don't result in divergent solutions. The example below includes several limits including limiting the line center to be within 0.1 cm-1 of the initial guess and the Line intensity to be within a factor of 2 of the intial guess. Placing limits on the parameters can be restrictive on the solution and cause the fit to not converge or return NaN for the standard error if it doesn't allow for a local minima to be found. The *Fit_DataSet* class also allows for the option to weight_spectra, currently this is set to False. Later in this example we will explore using weights in fitting.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity, weight_spectra = False,
             baseline_limit = False, baseline_limit_factor = 10,
             molefraction_limit = False, molefraction_limit_factor = 1.1,
             etalon_limit = False, etalon_limit_factor = 2, #phase is constrained to +/-␣
↪2pi,
             x_shift_limit = False, x_shift_limit_magnitude = 0.5,
             nu_limit = True, nu_limit_magnitude = 0.1,
             sw_limit = True, sw_limit_factor = 2,
             gamma0_limit = False, gamma0_limit_factor = 3, n_gamma0_limit= False, n_
↪gamma0_limit_factor = 50,
             delta0_limit = False, delta0_limit_factor = 2, n_delta0_limit = False, n_
↪delta0_limit_factor = 50,
             SD_gamma_limit = False, SD_gamma_limit_factor = 2, n_gamma2_limit = False,␣
↪n_gamma2_limit_factor = 50,
             SD_delta_limit = False, SD_delta_limit_factor = 50, n_delta2_limit = False,␣
↪n_delta2_limit_factor = 50,
             nuVC_limit = False, nuVC_limit_factor = 2, n_nuVC_limit = False, n_nuVC_
↪limit_factor = 50,
```

(continues on next page)

(continued from previous page)

```
            eta_limit = False, eta_limit_factor = 50, linemixing_limit = False,␣
↪linemixing_limit_factor = 50)
```

The next step is to generate the lmfit params dictionary object through the *Fit_DataSet.generate_params()* function. This translates baseline and parameter line list .csv files into a lmfit parameter dictionary that is used in the fits. After the parameters object is generated you can use the keys to set values and impose constraints on individual parameters, if desired. While this is not coded in the MATS toolkit, it is incredibly powerful as it lets you define min, max, vary, and expression values for any parameter. In the example below, two additional constraints are imposed on specific fit parameters. The first constrains all speed-dependent width parameters to be between the values of 0.01 and 0.25 and the second forces the amplitude of the etalon to be constant across all spectra.

```
params = fit_data.generate_params()



for param in params:
            if 'SD_gamma' in param:
                    params[param].set(min = 0.01, max = 0.25)
            if 'etalon_1_amp' in param:
                    if param != 'etalon_1_amp_1_1':
                            params[param].set(expr='etalon_1_amp_1_1')
```

The params file is then used to fit the spectra in the dataset using the *Fit_DataSet.fit_data()* function, where the result is a lmfit result object. The lmft prettyprint function prints the parameter fit results. Included below is an abbreviated prettyprint output that not only shows the fit result values and standard errors, but also highlights that constraints were imposed on the SD_gamma (speed dependent broadening) parameters and an expression was imposed on the etalon_amplitudes. It also shows that the there is a line intensity reported for every line and spectrum (sw_spectrum **number_** line) as the line intensities were not constrained to global fits. The reported sw shows that the fitted line intensity value is scaled by the minimum fit value. This scalar term is saved in a column called sw_scale_factor for reference. Scaling the line intensity aids in the fitting as line intensities are so much smaller than other fitted parameters.

```
result = fit_data.fit_data(params, wing_cutoff = 25)
print (result.params.pretty_print())
```

```
Name                     Value      Min      Max   Stderr     Vary     Expr Brute_Step
Pressure_1_1           0.07911     -inf      inf        0    False     None     None
Pressure_2_1           0.06556     -inf      inf        0    False     None     None
Pressure_3_1           0.04602     -inf      inf        0    False     None     None
Pressure_4_1           0.02488     -inf      inf        0    False     None     None
SD_delta_air_line_1          0     -inf      inf        0    False     None     None
SD_delta_air_line_10         0     -inf      inf        0    False     None     None
SD_delta_air_line_13         0     -inf      inf        0    False     None     None
SD_delta_air_line_25         0     -inf      inf        0    False     None     None
SD_delta_air_line_26         0     -inf      inf        0    False     None     None
SD_gamma_air_line_1        0.1     0.01     0.25        0    False     None     None
SD_gamma_air_line_10    0.1137     0.01     0.25 0.0008273     True     None     None
SD_gamma_air_line_13    0.1313     0.01     0.25 0.001115     True     None     None
SD_gamma_air_line_25      0.09     0.01     0.25        0    False     None     None
SD_gamma_air_line_26       0.1     0.01     0.25        0    False     None     None
. . .
etalon_1_amp_1_1      0.001762     -inf      inf 4.007e-05     True     None     None
etalon_1_amp_2_1      0.001762     -inf      inf 4.007e-05    False etalon_1_amp_1_1␣
↪None
```

(continues on next page)

(continued from previous page)

```
etalon_1_amp_3_1      0.001762     -inf      inf 4.007e-05    False etalon_1_amp_1_1  ␣
↪None
etalon_1_amp_4_1      0.001762     -inf      inf 4.007e-05    False etalon_1_amp_1_1  ␣
↪None
etalon_1_freq_1_1        1.196     -inf      inf         0    False     None     None
etalon_1_freq_2_1        1.196     -inf      inf         0    False     None     None
etalon_1_freq_3_1        1.196     -inf      inf         0    False     None     None
etalon_1_freq_4_1        1.196     -inf      inf         0    False     None     None
etalon_1_phase_1_1     -0.3479     -inf      inf   0.04585     True     None     None
etalon_1_phase_2_1    -0.09384     -inf      inf   0.04288     True     None     None
etalon_1_phase_3_1       -1.04     -inf      inf   0.04446     True     None     None
etalon_1_phase_4_1      -1.266     -inf      inf   0.04394     True     None     None
gamma0_air_line_1         0.04     -inf      inf         0    False     None     None
gamma0_air_line_10     0.04501     -inf      inf 4.919e-05     True     None     None
gamma0_air_line_13     0.04339     -inf      inf 7.531e-05     True     None     None
gamma0_air_line_25        0.04     -inf      inf         0    False     None     None
gamma0_air_line_26        0.04     -inf      inf         0    False     None     None
. . .
sw_1_line_1              4.369    2.184    8.738         0    False     None     None
sw_1_line_10             4.735      2.4    9.598 0.0008558     True     None     None
sw_1_line_13             3.087    1.562    6.246 0.0007302     True     None     None
sw_1_line_25             2.083    1.042    4.166         0    False     None     None
sw_1_line_26             3.399    1.699    6.798         0    False     None     None
sw_2_line_1              4.369    2.184    8.738         0    False     None     None
sw_2_line_10             4.752      2.4    9.598 0.0006929     True     None     None
sw_2_line_13             3.091    1.562    6.246 0.0006913     True     None     None
sw_2_line_25             2.083    1.042    4.166         0    False     None     None
sw_2_line_26             3.399    1.699    6.798         0    False     None     None
sw_3_line_1              4.369    2.184    8.738         0    False     None     None
sw_3_line_10             4.744      2.4    9.598 0.0007446     True     None     None
sw_3_line_13             3.095    1.562    6.246 0.0007499     True     None     None
sw_3_line_25             2.083    1.042    4.166         0    False     None     None
sw_3_line_26             3.399    1.699    6.798         0    False     None     None
sw_4_line_1              4.369    2.184    8.738         0    False     None     None
sw_4_line_10               4.8      2.4    9.598  0.001158     True     None     None
sw_4_line_13             3.118    1.562    6.246   0.00117     True     None     None
sw_4_line_25             2.083    1.042    4.166         0    False     None     None
sw_4_line_26             3.399    1.699    6.798         0    False     None     None
```

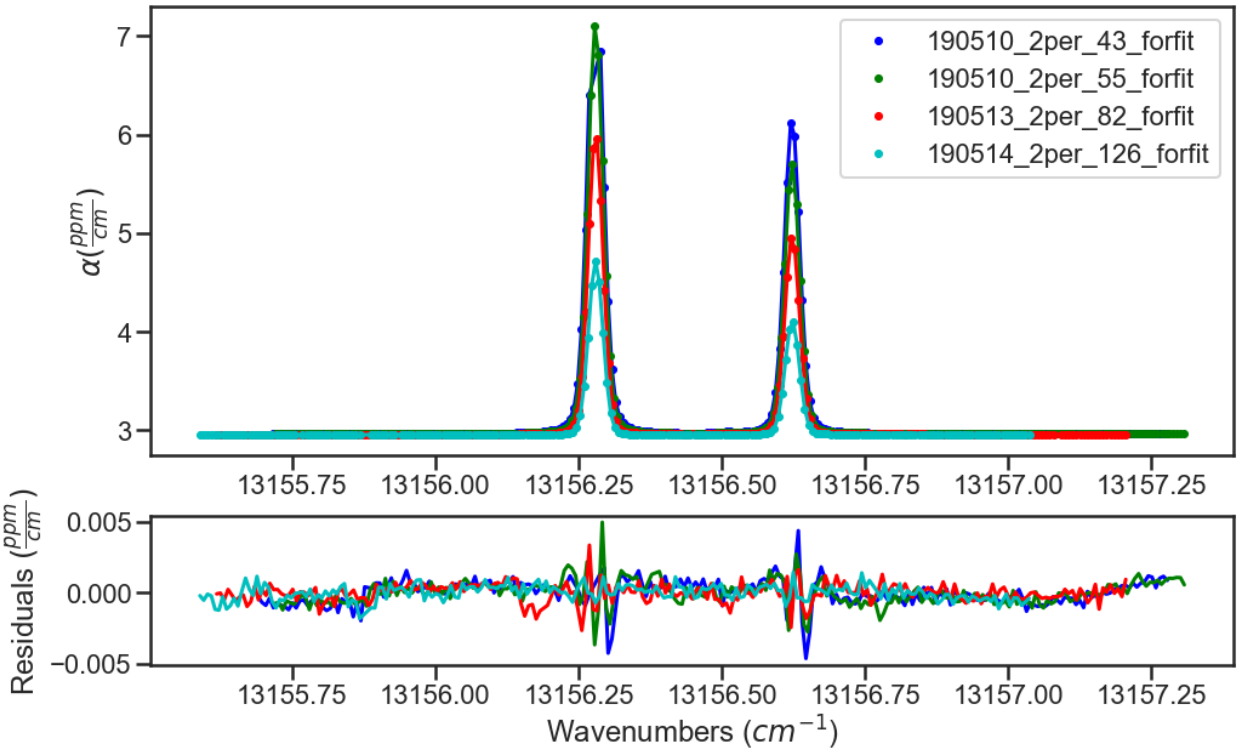
The last segment of code provides residual plots and updates residuals through the *Fit_DataSet.residual_analysis()* and

Dataset.plot_model_residuals() functions, updates the parameter and baseline line lists through

Fit_DataSet.update_params(), and generates a summary file with the fit results using

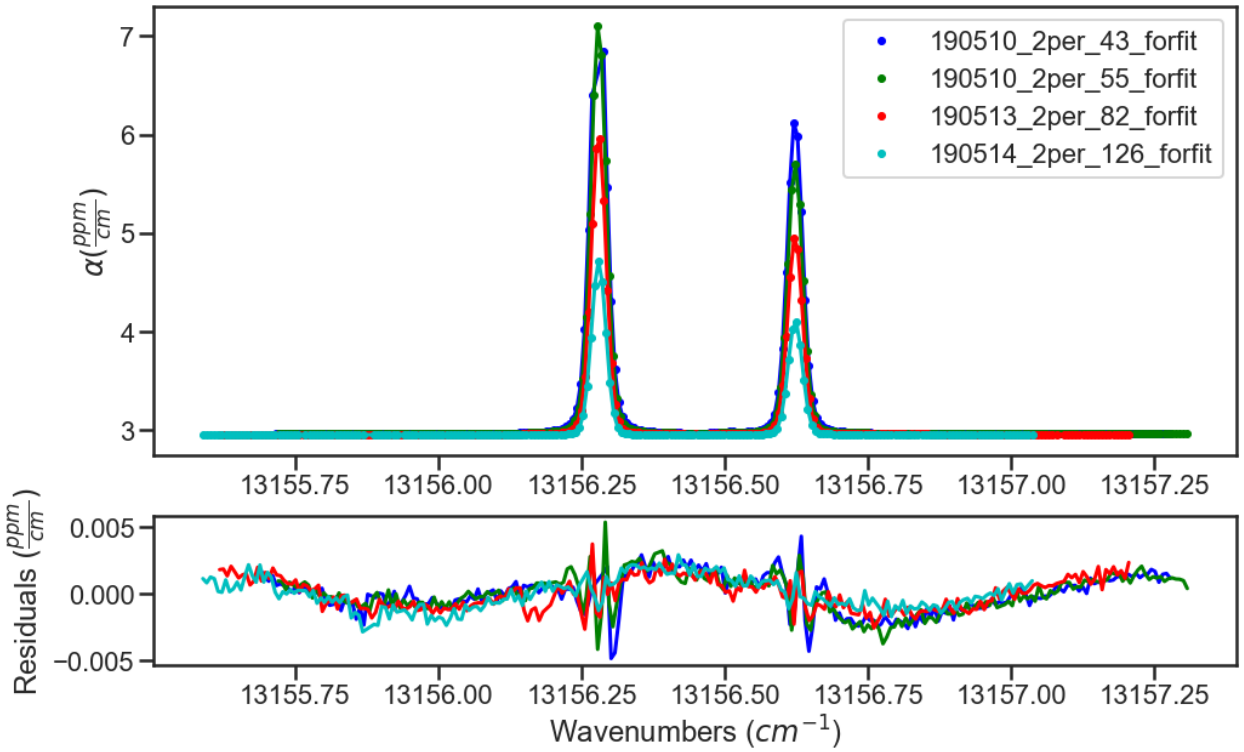
Dataset.generate_summary_file().

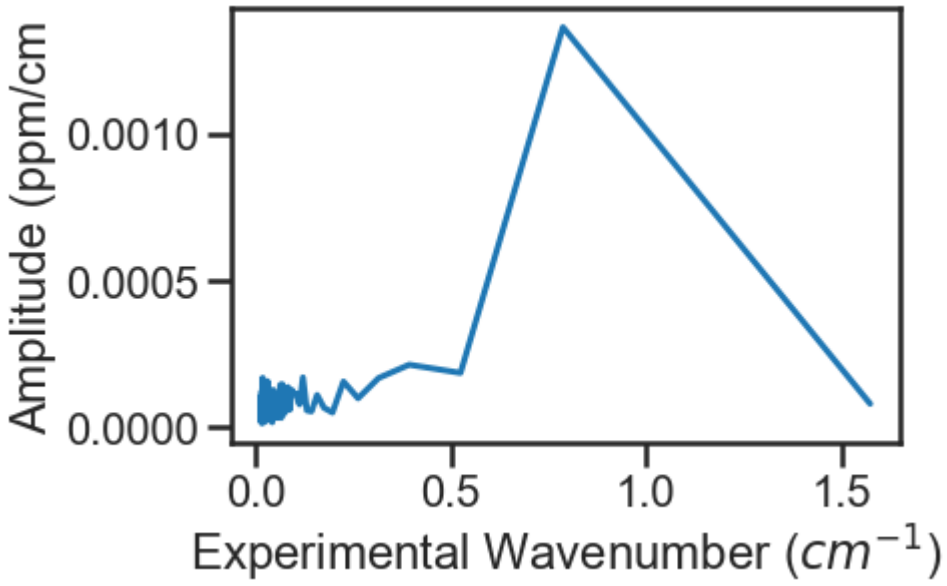
```
fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



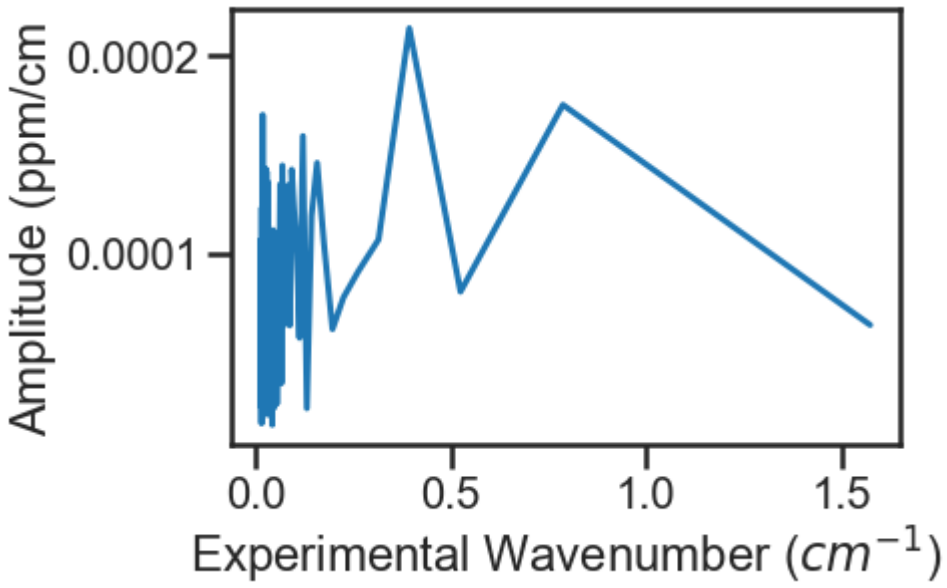
Call to the *Spectrum.fft_spectrum()* function takes an FFT of the residuals. If we hadn't included the etalon, the result of the *Spectrum.fft_spectrum()* function would show a peak with the most abundant period being 1.271443 cm-1 and an amplitude of 0.001364, which were used as the initial guess for the etalon in the spectrum class definitions. The more etalon periods present in the spectral region being fit the more precise the etalon amplitude and frequency determined by the FFT will be.

If we hadn't included the etalon, the fit residuals and FFT would like the plots below:





Using these values as the etalon period and amplitude give the fit residuals shown in the use example above and when incorporated the FFT no longer shows a substantial peak at 1.271443 cm-1.



Explore the Ability to Weight Spectra

MATS v2 introduced the ability to weight spectra in two different ways, using the stats column defined for each spectrum or weighting the entire spectrum. In the first example below, we use use the same code that was introduced previously, but set the weight_spectra variable in the *Fit_DataSet* definitions to True.

This weights the contribution to the solution at each point in each spectrum by 1/tau_stats_column. If the tau_stats_column is not defined in the *Spectrum* then this will default to equal weights for all data points.

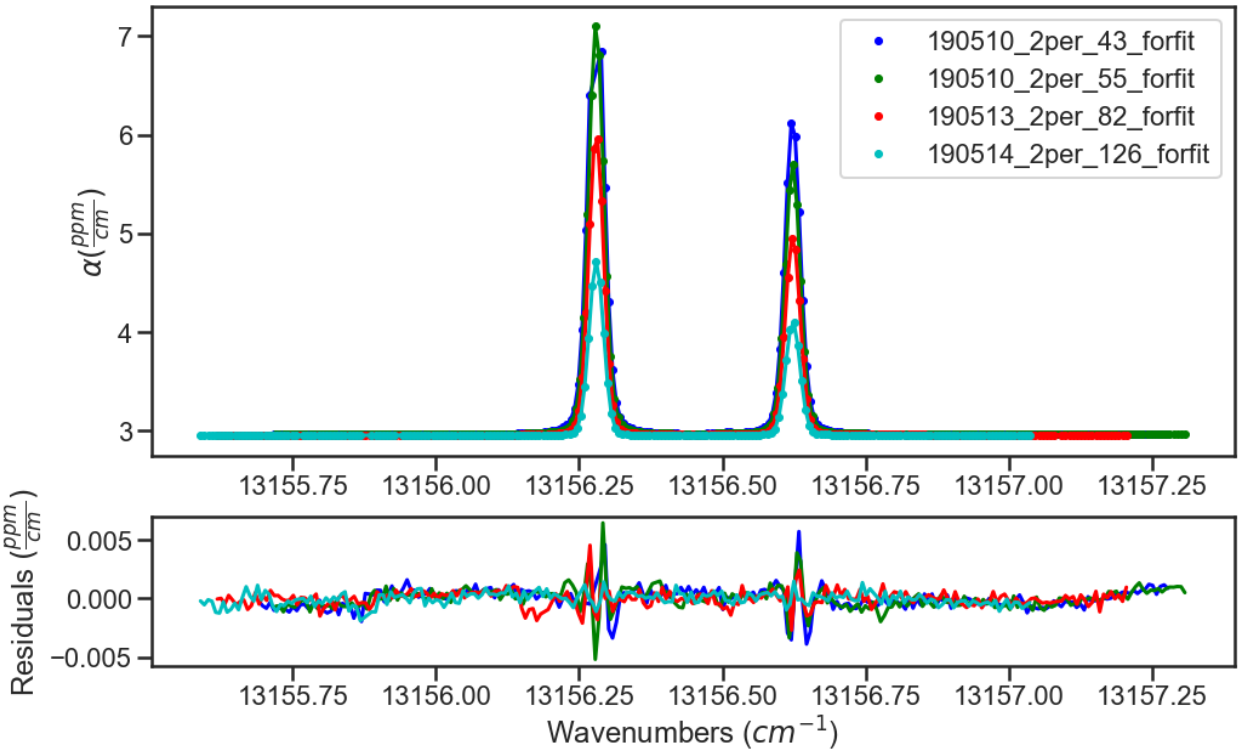
```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity, weight_spectra = True,
            baseline_limit = False, baseline_limit_factor = 10,
            molefraction_limit = False, molefraction_limit_factor = 1.1,
            etalon_limit = False, etalon_limit_factor = 2, #phase is constrained to +/-␣
↪2pi,
            x_shift_limit = False, x_shift_limit_magnitude = 0.5,
            nu_limit = True, nu_limit_magnitude = 0.1,
            sw_limit = True, sw_limit_factor = 2,
            gamma0_limit = False, gamma0_limit_factor = 3, n_gamma0_limit= False, n_
↪gamma0_limit_factor = 50,
            delta0_limit = False, delta0_limit_factor = 2, n_delta0_limit = False, n_
↪delta0_limit_factor = 50,
            SD_gamma_limit = False, SD_gamma_limit_factor = 2, n_gamma2_limit = False,␣
↪n_gamma2_limit_factor = 50,
            SD_delta_limit = False, SD_delta_limit_factor = 50, n_delta2_limit = False,␣
↪n_delta2_limit_factor = 50,
            nuVC_limit = False, nuVC_limit_factor = 2, n_nuVC_limit = False, n_nuVC_
↪limit_factor = 50,
            eta_limit = False, eta_limit_factor = 50, linemixing_limit = False,␣
↪linemixing_limit_factor = 50)
params = fit_data.generate_params()
```

(continues on next page)

(continued from previous page)

```
for param in params:
 if 'SD_gamma' in param:
     if params[param].vary == True:
         params[param].set(min = 0.01, max = 0.25)
 if 'etalon_1_amp' in param:
     if param != 'etalon_1_amp_1_1':
         params[param].set(expr='etalon_1_amp_1_1')



result = fit_data.fit_data(params, wing_cutoff = 25)
print (result.params.pretty_print())

fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```

For this example the statistics are relatively consistent across all points in the dataset, as shown in the plot below, so the fit residuals are very similar with some slight differences in the quality of fits and the parameter fit results.



The other option for weighting, that can be used with or without the point by point weighting, is to weight each spectrum by setting the spectrum weight value. The default for each spectrum is 1. This value can be set through the weight parameter in the spectrum definition or by using the *Spectrum.set_weight()* function, which is shown below.

For all weighting options, the residuals are adjusted by the weight factors. For this example, where the spec_1 is given a weight of 0 this effectively removed spec_1 from the solution consideration depicted by the flat residuals for spectrum

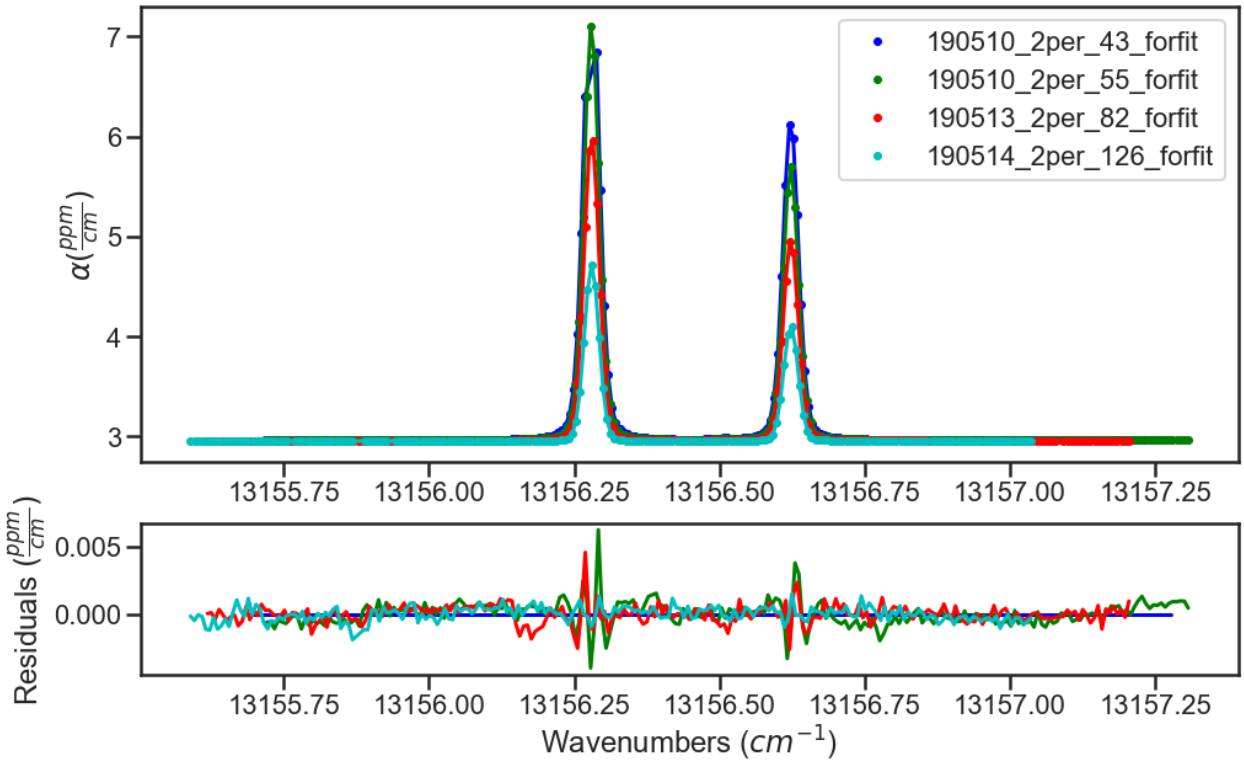
190510_2per_43_forfit.

```
spec_1.set_weight(0)
FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,␣
↪lineprofile = 'SDVP', linemixing = False,
                                fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
                                nu_constrain = False, sw_constrain = False, gamma0_
↪constrain = True, delta0_constrain = True,
                                 aw_constrain = True, as_constrain = True,
                                 nuVC_constrain = True, eta_constrain =True, linemixing_
↪constrain = True)
                                #additional_columns = ['trans_id', 'local_lower_quanta', 'm'])

FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {7:{1:True, 2:False, 3:
↪False}}, vary_sw = {7:{1:True, 2:False, 3:False}},
                                                    vary_gamma0 = {7:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma0 = {7:{1:True}},
                                                    vary_delta0 = {7:{1: False, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_delta0 = {7:{1:True}},
                                                    vary_aw = {7:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma2 = {7:{1:False}},
                                                    vary_as = {}, vary_n_delta2 = {7:{1:
↪False}},
                                                    vary_nuVC = {7:{1:False}}, vary_n_nuVC␣
↪= {7:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {7:{1:
↪False}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = True, vary_molefraction = {7:
↪False, 1:False}, vary_xshift = False,
                                    vary_etalon_amp= True, vary_etalon_period= False,␣
↪vary_etalon_phase= True)



fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity, weight_spectra = True,
             baseline_limit = False, baseline_limit_factor = 10,
             molefraction_limit = False, molefraction_limit_factor = 1.1,
             etalon_limit = False, etalon_limit_factor = 2, #phase is constrained to +/-␣
↪2pi,
             x_shift_limit = False, x_shift_limit_magnitude = 0.5,
             nu_limit = True, nu_limit_magnitude = 0.1,
             sw_limit = True, sw_limit_factor = 2,
             gamma0_limit = False, gamma0_limit_factor = 3, n_gamma0_limit= False, n_
↪gamma0_limit_factor = 50,
             delta0_limit = False, delta0_limit_factor = 2, n_delta0_limit = False, n_
↪delta0_limit_factor = 50,
             SD_gamma_limit = False, SD_gamma_limit_factor = 2, n_gamma2_limit = False,␣
↪n_gamma2_limit_factor = 50,
             SD_delta_limit = False, SD_delta_limit_factor = 50, n_delta2_limit = False,␣
↪n_delta2_limit_factor = 50,
```

(continues on next page)

(continued from previous page)

```
            nuVC_limit = False, nuVC_limit_factor = 2, n_nuVC_limit = False, n_nuVC_
↪limit_factor = 50,
            eta_limit = False, eta_limit_factor = 50, linemixing_limit = False,␣
↪linemixing_limit_factor = 50)
params = fit_data.generate_params()



for param in params:
 if 'SD_gamma' in param:
     if params[param].vary == True:
         params[param].set(min = 0.01, max = 0.25)
 if 'etalon_1_amp' in param:
     if param != 'etalon_1_amp_1_1':
         params[param].set(expr='etalon_1_amp_1_1')



result = fit_data.fit_data(params, wing_cutoff = 25)
print (result.params.pretty_print())

fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



Fitting Synthetic Spectra

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

This example simulates and fits Oxygen A-Band spectra

Import Modules and Set-Up

This example starts with importing modules and setting up file locations

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from matplotlib import gridspec
import MATS
```

Optional import of seaborn package for figure generation

```
import seaborn as sns
sns.set_style("whitegrid")
sns.set_style("ticks")
sns.set_context("poster")
```

If you aren't using the MATS file strucutre, change the path to the working directory that contains experimental spectra or to the folder that you want to work in.

```
os.chdir(path)
```

Simulate Spectra

If you are simulating spectra, opposed to reading them in from a file as discussed above, then you can use the *simulate_spectrum()* function.

When simulating spectra, the first step is to read in the reference line list. This file can be generated following the *Generating Parameter Line lists* and read in using pd.read_csv function. The following code reads in the reference line list using the py:func:*LoadLineListData* function to read in the line list provided for the Oxygen A-Band.

```
from MATS.linelistdata import linelistdata
PARAM_LINELIST = linelistdata['O2_ABand_Drouin_2017_linelist']
```

Just as you would do if reading in the experimental spectrum, this example defines some common simulation and fit variables. In addition to variables that would be used in fitting experimental spectra, the minimum and maximum wavenumbers for the simulation and the simulation wavenumber spacing are defined. Alternatively, a wavenumbers term can be used and an array can be used as the input defining the x-axis. This feature allows the use of a non-uniform x-axis and will take precident of the use of wave_min, wave_max, and wave_step parameterization.

The baseline is defined by a polynomial where the array index is the parameter coefficient order, such that the [1, 0] would correspond to a linear baseline with a slope of 0 and an offset of 1.

```
wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
```

(continues on next page)

(continued from previous page)

```
Fit_Intensity = 1e-24 #intensities must be above this value for the line to be fit
order_baseline_fit = 1
sample_molefraction = {7 :0.002022}
wave_min = 13155 #cm-1
wave_max = 13157.5 #cm-1
wave_space = 0.005 #cm-1
wavenumbers = np.arange(13155, 13158, 0.02)
baseline_terms = [0] #polynomial baseline coefficients where the index is equal to the␣
↪coefficient order
```

The *simulate_spectrum()* function also allows for error to be added in the following ways:

- to the absorption axis through signal-to-noise ratio (SNR). The SNR is implemented by adding gaussian noise to the spectra such that the (maximum alpha - minimum alpha) / noise is equal to the set SNR.
- to the wavenumber axis through the wave_err parameter. The wavenumber error is implemented by adding a gaussian noise error of the specified magnitude to the wavenumber axis.
- to the mole fraction through the molefraction_err parameter. The molefraction error is implemented as a percent error bias on each (could enter a negative percent error to get negative offset). This mimics the maximum impact that a constant error in sample mole fraction would have.
- to the temperature/pressure through the temperature_err and pressure_err dictionaries. In experiments there are generally two type of errors with pressure and temperature measurements. The first is a constant bias in the reading. The second type of error is an actual change in the pressure/temperature during the collection of the spectrum. To account for both error types the pressure_err and temperature_err are dictionaries, where the keys correspond to 'bias/per_bias' (bias for temperature and per_bias for pressure), function (allows 'linear' or 'sine'), and params. If the function is 'linear' then the param keys are 'm' and 'b' corresponding to the slope and interecept. If the function is 'sine' then the param keys are 'amp', 'freq', and 'phase' corresponding to the amplitude, period, and phase of the sine function. For both temperature and pressure, the pressure/temperature recorded in the simulated spectra output include the average pressure or temperature over the segment (analogous to the frequency of the pressure/temperature measurement in an experiment) and does not include the bias in pressure/temperature as this is would be an unknown in an experiment.

```
SNR = 4000
wave_error = 1e-4
temperature_err =  {'bias': 0.01, 'function': None, 'params': {}}
pressure_err = {'per_bias': 0.01, 'function': None, 'params': {}}
molefraction_err = {7:0.01}
```

These parameters and the additional settings for filenames and number of segments can be used to call the *simulate_spectrum()* function setting the output equal to a variable as would be done for generating an instance of the *Spectrum* class from a .csv file. This makes it simple to transition code from analysis of experimental spectra to error analysis through simulations.

```
spec_1 =  MATS.simulate_spectrum(PARAM_LINELIST, wavenumbers = wavenumbers, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪temperature_err = temperature_err, pressure = 25,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '25_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
```

(continues on next page)

(continued from previous page)

```
spec_2 =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_
↪error = wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪temperature_err = temperature_err, pressure = 50,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '50_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
spec_3 =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_
↪error = wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪temperature_err = temperature_err, pressure = 100,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '100_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
spec_4 =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_
↪error = wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪temperature_err = temperature_err, pressure = 200,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '200_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
```

Generate a Dataset

The procedure for analysis for both simulating and loading spectrum are the same as described in *Fitting Experimental Spectra*. The next step is to combine all desired *Spectrum* objects

into a *Dataset* object, where we give the dataset a name and specify the parameter line list to use for analysis/

```
SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4], 'Line Intensity', PARAM_
↪LINELIST)
```

The *Dataset* class contains a function to generate a baseline line list, analogous to the one for the parameter line list done outside of this example, based on the order of the baseline fit, etalons, molecules, x-shift parameters, and segments as defined by both the spectrum and dataset objects.

```
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()
```

Generate Fit Parameter Files

The next section of code uses the *Generate_FitParam_File* class to define what line shape to use for the initial fits, whether to use line mixing, the minimum line intensity to fit a line, mimimum intensity to included in the simulation, and for each line parameter whether that parameter is going to be constrained across all spectra or whether there will be a parameter for each spectrum (multi-spectrum vs single-spectrum fits) on a parameter by parameter basis. In the example below, the SDVP line profile without line mixing will be used to fit lines with line intensities greater than 1e-24 with no parameters floated.

```
FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,␣
↪lineprofile = 'SDVP', linemixing = False,
                                fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
                                nu_constrain = True, sw_constrain = True, gamma0_
↪constrain = True, delta0_constrain = True,
                                 aw_constrain = True, as_constrain = True,
                                 nuVC_constrain = True, eta_constrain =True, linemixing_
↪constrain = True)
```

The next step is to generate fit parameter and baseline line lists that include columns that specify whether that parameter should be varied during fitting, in addition to adding error columns for the fit error for each parameter. For the following example the line centers, line intensities, collisional half-widths, and speed-dependent broadening terms will be floated for all main oxygen isotopes for lines where the line intensity is greater than 1e-24. None of the baseline parameters are floated

```
FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {7:{1:False, 2:False, 3:
↪False}}, vary_sw = {7:{1:False, 2:False, 3:False}},
                                                    vary_gamma0 = {7:{1: False, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma0 = {7:{1:True}},
                                                    vary_delta0 = {7:{1: False, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_delta0 = {7:{1:True}},
                                                    vary_aw = {7:{1: False, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma2 = {7:{1:False}},
                                                    vary_as = {}, vary_n_delta2 = {7:{1:
↪False}},
                                                    vary_nuVC = {7:{1:False}}, vary_n_nuVC␣
↪= {7:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {7:{1:
↪False}}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = False, vary_molefraction = {7:
↪False, 1:False}, vary_xshift = False,
                                    vary_etalon_amp= False, vary_etalon_period= False,␣
↪vary_etalon_phase= False,
                                       vary_pressure = False, vary_temperature = False)
```

These functions will generate .csv files corresponding to these selections, which are read in by the *Fit_DataSet* class instantiation. When simulating spectra, if you don't adjust the baseline and parameter line lists, then your simulated variables are equal to your initial guesses. This can lead to a local minimum where the fit will not move from the initial guesses. The last segment of the code shows an example where random uncertainties from the simulated values are applied and then the Parameter line list file is resaved.

```
Parameter_LineList = pd.read_csv('Parameter_LineList.csv', index_col = 0)
```

(continues on next page)

(continued from previous page)

```
for index in Parameter_LineList[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min)].index.unique():
 Parameter_LineList.loc[Parameter_LineList.index == index, 'nu'] = Parameter_
↪LineList[Parameter_LineList.index == index]['nu'].values[0] + np.random.normal(loc = 0,
↪ scale =0.005) #adjust by random number scale 0.005 cm-1
 Parameter_LineList.loc[Parameter_LineList.index == index, 'sw'] = Parameter_
↪LineList[Parameter_LineList.index == index]['sw'].values[0]*(1 + np.random.normal(loc␣
↪= 0, scale =0.01)) # adjust by random amount at 1% scale
 Parameter_LineList.loc[Parameter_LineList.index == index, 'gamma0_air'] = Parameter_
↪LineList[Parameter_LineList.index == index]['gamma0_air'].values[0]*(1 + np.random.
↪normal(loc = 0, scale =0.01)) # adjust by random amount at 1% scale
 Parameter_LineList.loc[Parameter_LineList.index == index, 'delta0_air'] = Parameter_
↪LineList[Parameter_LineList.index == index]['delta0_air'].values[0]*(1 + np.random.
↪normal(loc = 0, scale =0.02)) # adjust by random amount at 2% scale
 Parameter_LineList.loc[Parameter_LineList.index == index, 'SD_gamma_air'] = Parameter_
↪LineList[Parameter_LineList.index == index]['SD_gamma_air'].values[0]*(1 + np.random.
↪normal(loc = 0, scale =0.1)) # adjust by random amount at 10% scale

Parameter_LineList.to_csv('Parameter_LineList.csv')
```

Fit Dataset

Instantiating the *Fit_DataSet* class reads in the information from the baseline and parameter linelists generated in the previous step. The Fitting Experimental Spectra documentation gives a more basic example of how to perform a fit. The example below iterates on a fit, where in the first iteration no parameters are floated, in the second iteration the line centers and line intensities are floated, and in the final iteration the collisional broadening, pressure shift, and speed-dependent broadening are floated. Sometimes it is not possible to float all parameters at the same time, specifically with poor initial guesses for line center. In that case an iterative approach like this is advantageous.

The iteration could also be applied to the lmfit parameter object by setting the the vary term, opposed to through the Parameter_LineList. In that approach the final Parameter_LineList would have the standard uncertainities and fit values saved, but wouldn't necessarily have the parameter_vary column appropriately set.

```
iteration = 0
while iteration <= 2:
 print ('ITERATION ' + str(iteration))
 if iteration == 1:
     Parameter_LineList = pd.read_csv('Parameter_LineList.csv', index_col = 0)
     Parameter_LineList.loc[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min), 'nu_vary'] = True
     Parameter_LineList.loc[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min), 'sw_vary'] = True
     Parameter_LineList.to_csv('Parameter_LineList.csv')
 if iteration ==2:
     Parameter_LineList = pd.read_csv('Parameter_LineList.csv', index_col = 0)
     Parameter_LineList.loc[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min), 'gamma0_air_vary'] = True
     Parameter_LineList.loc[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min), 'delta0_air_vary'] = True
```

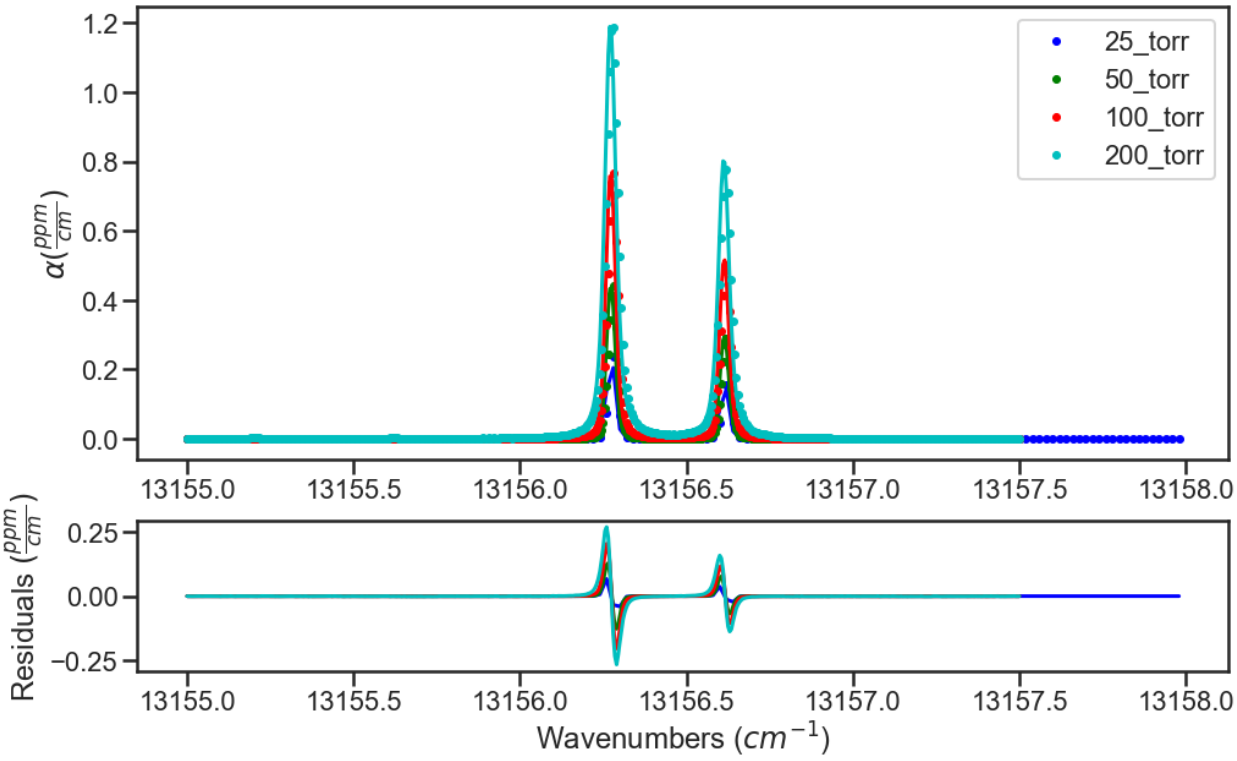
(continues on next page)

(continued from previous page)

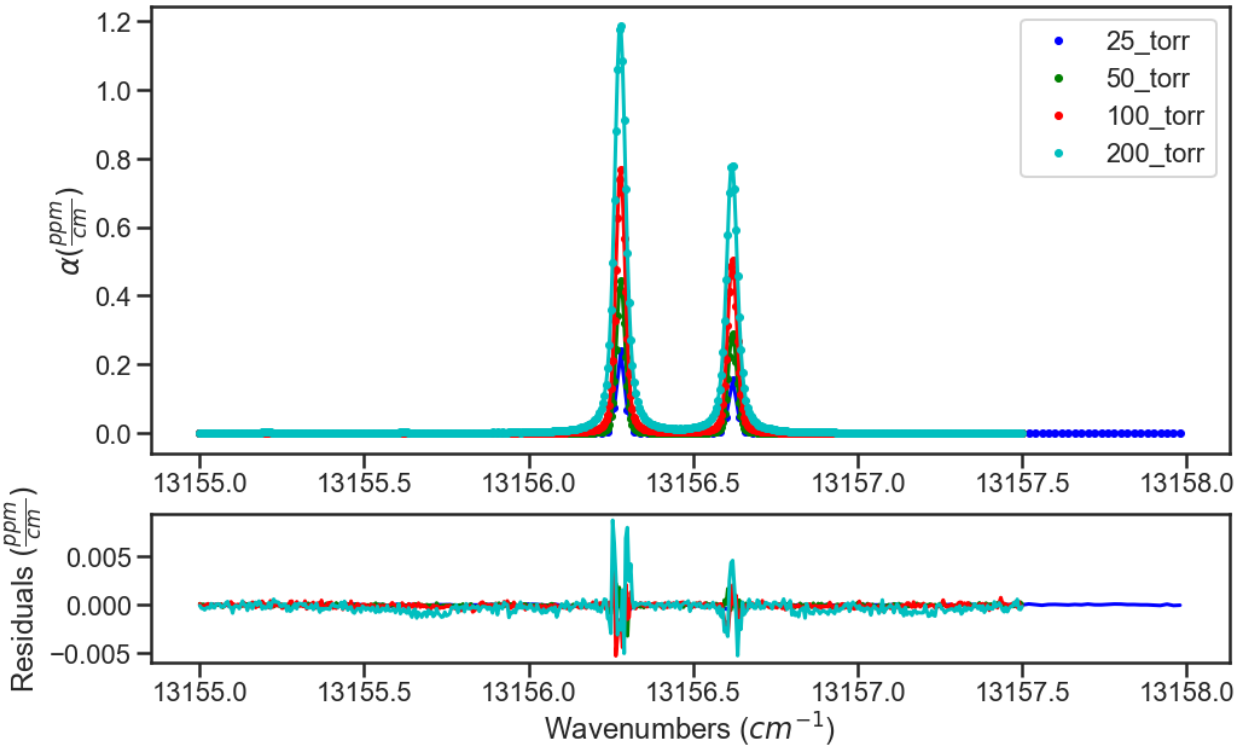
```
    Parameter_LineList.loc[(Parameter_LineList['sw']>1) & (Parameter_LineList['nu']
↪<wave_max) & (Parameter_LineList['nu']>wave_min), 'SD_gamma_air_vary'] = True
    Parameter_LineList.to_csv('Parameter_LineList.csv')


fit_data = Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity)
params = fit_data.generate_params()
result = fit_data.fit_data(params, wing_cutoff = 25, wing_wavenumbers = 1)
fit_data.residual_analysis(result, indv_resid_plot=False)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
iteration+=1
```

The plots below show the results from the various iterations, where the fit residuals decrease with each iteration.

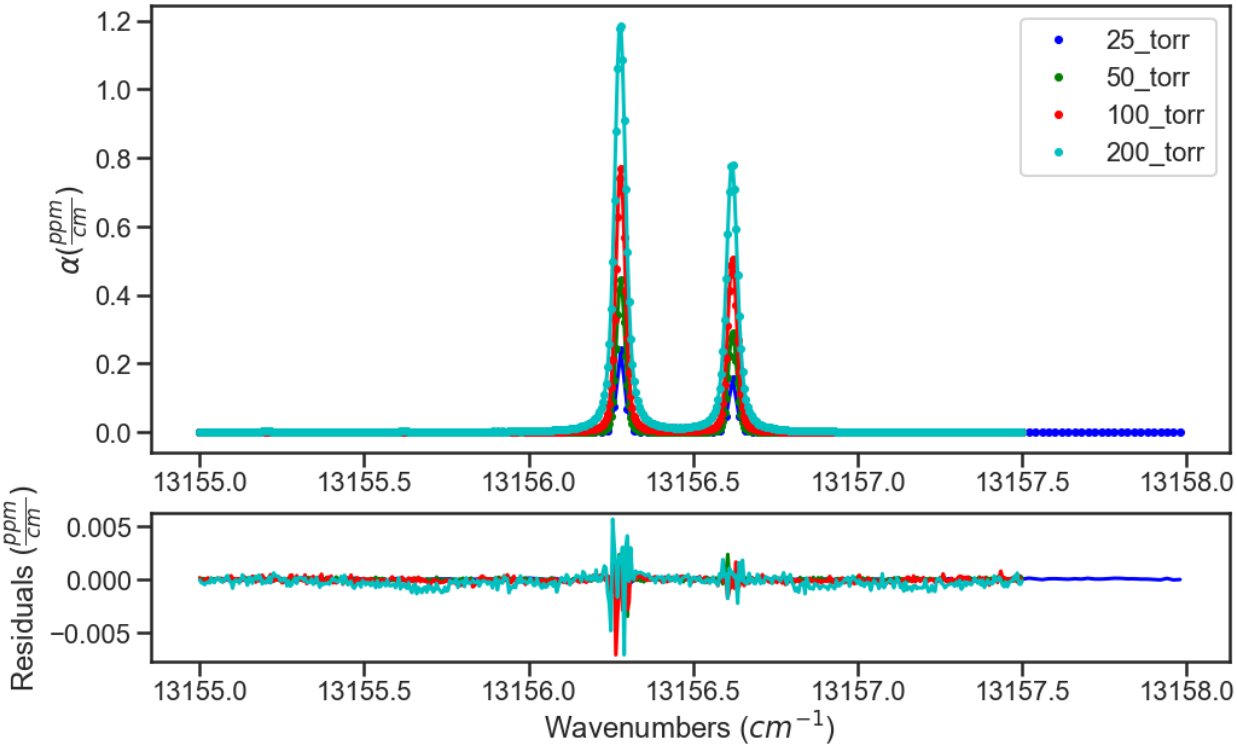
Iteration 0



Iteration 1


25_torr
50_torr
100_torr
200_torr
$\alpha(\frac{ppm}{cm})$
Residuals $(\frac{ppm}{cm})$
Wavenumbers (cm^{-1})

Iteration 2


25_torr
50_torr
100_torr
200_torr
$\alpha(\frac{ppm}{cm})$
Residuals $(\frac{ppm}{cm})$
Wavenumbers (cm^{-1})

Analysis of ASCENDS Line Spectra

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

The 30012<–00001 CO2 band is frequently used for remote sensing applications, where the R16e line in this band centered at 6359.967 cm-1 is the target line for the ASCENDS - Active Sensing of CO2 Emissions over Nights, Days, and Seasons mission. This makes this line an important target for spectroscopic reference data and thus a convenient target for single line spectroscopic studies or examples.

Data originally reported in Long, D., et al., Frequency-agile, rapid scanning cavity ring-down spectroscopy (FARS-CRDS) measurements of the (30012)←(00001) near-infrared carbon dioxide band. Journal of Quantitative Spectroscopy and Radiative Transfer, 2015. 161: p. 35-40. This example is explicitly referenced in the Adkins and Hodges, Assessment of the precision, bias and numerical correlation of fitted parameters obtained by multi-spectrum fits of the Hartmann-Tran line profile to simulated spectra, JQSRT (under review).

This example. shows the fitting of real experimental data. This data has a non-zero baseline and several etalons that need to be considered in order to effectively fit the spectra. In addition to fitting the experimental spectra, this example also uses simulations to explore the anticipated impact of improving the SNR of the data and increasing the pressure range of the dataset.

Define Spectra

Module import follows from the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples. The experimental spectra and the linelists are read-in in the same manner as described in the *Fitting Experimental Spectra* example.

In this example three etalons are included in the simulation and linear baseine is considered for each spectrum in the dataset. These fits also make use of the segment column, which allows the spectrum-specific parameters (baseline, etalons) to be treated on smaller segments of the spectrum, opposed to across the whole spectrum. The segment column is set using the segment_column parameter in the *Spectrum* definition and is simply the name of a column in the spectrum .csv that contains integers that are common for continuous segments of the spectrum. This is especially useful for treatment of faster etalons, which may not maintain a consistent phase over entire spectrum. Additional constraints can be imposed to help with fit convergence and physical modeling.

```
from MATS.linelistdata import linelistdata
import MATS.hapi as hapi

wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-23 #intensities must be above this value for the line to be fit
order_baseline_fit = 1
tau_column = 'Alpha' # Mean tau/us
freq_column = 'Wavenumber' # Total Frequency /MHz
pressure_column = 'Pressure'
temperature_column = 'Temperature'
tau_stats_column = None


PARAM_LINELIST = linelistdata['JQSRT2021_SDNGP_2015']
PARAM_LINELIST.loc[PARAM_LINELIST['n_delta0_air'].isna(), 'n_delta0_air'] = 0
etalons = {1:[0.004321,1.168], 2:[0.001377, 59.38], 3:[0.0004578, 29.75]}


spec_1 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_56Torr_03_formatted_R16e',  molefraction =
↪{2:0.0004254}, natural_abundance = True, diluent = 'air',
```

(continues on next page)

(continued from previous page)

```
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)


spec_2 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_83Torr_02_formatted_R16e',  molefraction =
↪{2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_3 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_101Torr_01_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_4 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_109Torr_03_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                      etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
```

(continues on next page)

(continued from previous page)

```
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_5 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_152Torr_01_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_6 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_186Torr_03_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_7 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_269Torr_01_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)

spec_8 = MATS.Spectrum('spectrum_CO2_Air_1%Ar_271Torr_03_formatted_R16e',  molefraction␣
↪= {2:0.0004254}, natural_abundance = True, diluent = 'air',
                                                                    etalons = etalons,
                                                                    baseline_order = order_
↪baseline_fit, segment_column = 'Segment',
                                                                    input_freq = False,␣
↪frequency_column = freq_column,
```

(continues on next page)

(continued from previous page)

```
                                                                    input_tau = False, tau_
↪column = tau_column, tau_stats_column = tau_stats_column,
                                                                    pressure_column =␣
↪pressure_column, temperature_column = temperature_column,
                                                                    nominal_temperature =␣
↪296, x_shift = 0.00)
```

Construct Dataset and Generate Fit Parameters

The *Dataset* object is defined as is outlined in the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples.

The baseline and parameter fit parameter files are generated using the *Generate_FitParam_File* class. In this example, the line intensity is allowed to float for each spectrum, but all other line shape parameters are constrained to their pressure dependence. Long et al. fixed the beyond Voigt SDNGP line shape parameters to those reported by Lin, H., et al., Cavity ring-down spectrometer for high-fidelity molecular absorption measurements. Journal of Quantitative Spectroscopy and Radiative Transfer, 2015. 161: p. 11-20 , while allowing the VP parameters and line mixing to float. In this example, we are allowing all SDNGP parameters to float using the Lin et al. values as the initial guess, but only considering spectra in the isolated transition pressure regime, where line mixing can be ignored.

The slope and intercept for the baseline of each spectra are floated, along with the amplitude, period, and phase of each etalon. Because this example uses the segment function, you can see that each spectrum has 4 segments in the print out of the baseline linelist, so the number of rows in the baseline fit parameter list is four times larger than the number of spectra. Additional constraints will be added to the etalon parameters in the next section.

```
SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4, spec_5, spec_6, spec_7, spec_8],
↪'MATS_Example',PARAM_LINELIST)

#Generate Baseline Parameter list based on number of etalons in spectra definitions and␣
↪baseline order
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()



    FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,␣
↪lineprofile = 'SDNGP', linemixing = False,
                                                                                    ␣
↪fit_intensity = Fit_Intensity, threshold_intensity = IntensityThreshold, sim_window =␣
↪wave_range,
                                                                                    ␣
↪nu_constrain = True, sw_constrain = False, gamma0_constrain = True, delta0_constrain =␣
↪True,
                                                                                    ␣
↪aw_constrain = True, as_constrain = True,
                                                                                    ␣
↪nuVC_constrain = True, eta_constrain =True, linemixing_constrain = True,
                                                                                    ␣
↪    additional_columns = ['local_lower_quanta'])

    FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {2:{1:True}}, vary_sw␣
↪= {2:{1:True}},
                                                                                    ␣
↪             vary_gamma0 = {2:{1: True}},
```

(continues on next page)

(continued from previous page)

```
                vary_delta0 = {2:{1: True}},
                vary_aw = {2:{1: True}},
                vary_as = {2:{1: True}},
                vary_nuVC = {2:{1:True}},
                vary_eta = {}, vary_linemixing = {2:{1:False}})
    FITPARAMS.generate_fit_baseline_linelist(vary_baseline = True, vary_molefraction =
↪{2:False}, vary_pressure = False, vary_temperature = False, vary_xshift = False,
↪vary_etalon_amp= True, vary_etalon_period= True, vary_etalon_phase= True)
```

Spectrum Number	Segment Number	Pressure	Pressure_err	Pressure_vary	Temperature	Temperature_err	Temperature_vary	baseline_a	baseline_a_err	baseline_a_vary	...
1.0	0.0	0.073747	0	False	296.275731	0	False	0.0	0	True	...
1.0	1.0	0.073747	0	False	296.275731	0	False	0.0	0	True	...
1.0	2.0	0.073747	0	False	296.275731	0	False	0.0	0	True	...
1.0	3.0	0.073747	0	False	296.275731	0	False	0.0	0	True	...
2.0	0.0	0.109482	0	False	296.318849	0	False	0.0	0	True	...
2.0	1.0	0.109482	0	False	296.318849	0	False	0.0	0	True	...
2.0	2.0	0.109482	0	False	296.318849	0	False	0.0	0	True	...
2.0	3.0	0.109482	0	False	296.318849	0	False	0.0	0	True	...
3.0	0.0	0.132529	0	False	296.337066	0	False	0.0	0	True	...
3.0	1.0	0.132529	0	False	296.337066	0	False	0.0	0	True	...
3.0	2.0	0.132529	0	False	296.337066	0	False	0.0	0	True	...
3.0	3.0	0.132529	0	False	296.337066	0	False	0.0	0	True	...
4.0	0.0	0.142618	0	False	296.233753	0	False	0.0	0	True	...
4.0	1.0	0.142618	0	False	296.233753	0	False	0.0	0	True	...
4.0	2.0	0.142618	0	False	296.233753	0	False	0.0	0	True	...
4.0	3.0	0.142618	0	False	296.233753	0	False	0.0	0	True	...
5.0	0.0	0.199886	0	False	296.291868	0	False	0.0	0	True	...
5.0	1.0	0.199886	0	False	296.291868	0	False	0.0	0	True	...
5.0	2.0	0.199886	0	False	296.291868	0	False	0.0	0	True	...
5.0	3.0	0.199886	0	False	296.291868	0	False	0.0	0	True	...
6.0	0.0	0.243783	0	False	296.364158	0	False	0.0	0	True	...
6.0	1.0	0.243783	0	False	296.364158	0	False	0.0	0	True	...
6.0	2.0	0.243783	0	False	296.364158	0	False	0.0	0	True	...
6.0	3.0	0.243783	0	False	296.364158	0	False	0.0	0	True	...
7.0	0.0	0.353395	0	False	296.404053	0	False	0.0	0	True	...
7.0	1.0	0.353395	0	False	296.404053	0	False	0.0	0	True	...
7.0	2.0	0.353395	0	False	296.404053	0	False	0.0	0	True	...
7.0	3.0	0.353395	0	False	296.404053	0	False	0.0	0	True	...
8.0	0.0	0.355892	0	False	296.319387	0	False	0.0	0	True	...
8.0	1.0	0.355892	0	False	296.319387	0	False	0.0	0	True	...
8.0	2.0	0.355892	0	False	296.319387	0	False	0.0	0	True	...
8.0	3.0	0.355892	0	False	296.319387	0	False	0.0	0	True	...

32 rows × 46 columns

Fit Spectra

The fitting of the dataset is looped to iterate on the fit results. The result and residual plots are shown after each iteration (with the first and last iteration shown below). In the first few iterations not all of the etalons are well modeled leading to systematic residuals, iterating on the best fit results helps better model the etalons minimizing the residuals.

Using the segments in a spectrum, allows us to model the spectrum specific parameters by segment opposed to across the whole spectrum. However, this flexibility can lead to divergent solutions if additional constraints aren't included. This example will constrain these parameters using the *Fit_DataSet.constrained_baseline()* function and by setting constraints directly in the lmfit parameter dictionary. The *Fit_DataSet.constrained_baseline()* function indicates the baseline parameters that should be constrained to be equal across the entire spectrum (parameters that won't take advantage of the segment structure). For this example the baseline and etalon periods are constrained to be

equal equal across the entire spectrum.

This example also sets a few additional constraints on the etalons. First, the period of each etalon is set to be equal across all spectra in the dataset. Second, the phase of the first etalon is constrained to be equal across each spectrum.

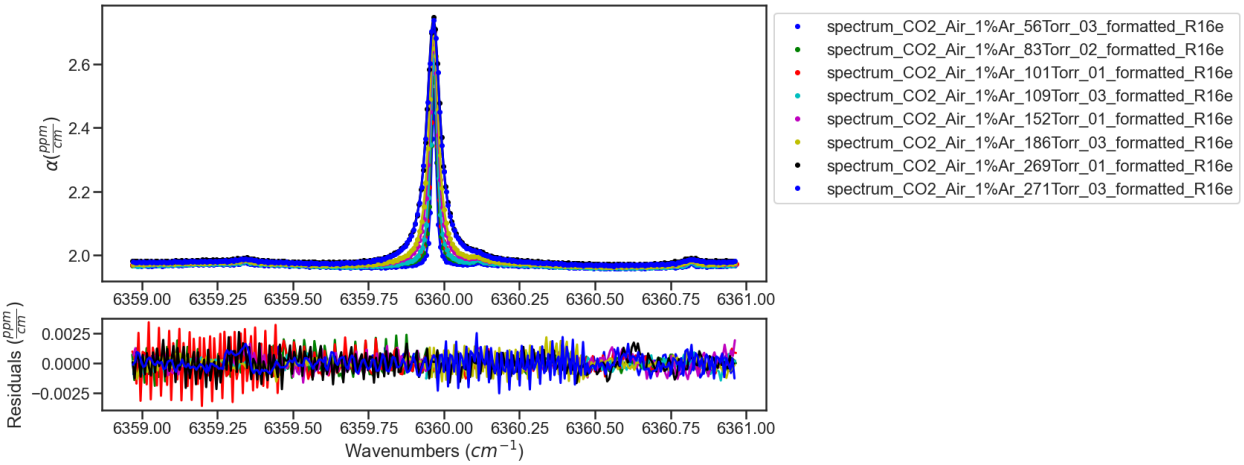
```
counter = 0
iterations = 5
while counter < iterations:
        fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList',␣
↪minimum_parameter_fit_intensity = Fit_Intensity)
        params = fit_data.generate_params()




        fit_data.constrained_baseline(params, baseline_segment_constrained = True,␣
↪xshift_segment_constrained = True, molefraction_segment_constrained = True,
                                                                        etalon_
↪amp_segment_constrained = False, etalon_period_segment_constrained = True, etalon_
↪phase_segment_constrained = False,
                                                                        pressure_
↪segment_constrained = True, temperature_segment_constrained = True)


        minimum_segment_spec_1 = spec_1.segments.min()


        for param in params:

                if 'etalon_1_period_' in param:
                        #Sets the period of etalon_1 to be equal for all spectra
                        if param != 'etalon_1_period_1_' + str(int(minimum_segment_spec_
↪1)):
                                params[param].set(expr='etalon_1_period_1_' +␣
↪str(int(minimum_segment_spec_1)))
                if 'etalon_1_phase_' in param:
                        #Constrains the phase of etalon_1 across each spectrum
                        start_ =  (param.find('_', 9))
                        end_ =  (param.find('_', param.find('_', 9) + 1))
                        spectrum_number = str(param[start_ + 1:end_])

                        if param != 'etalon_1_phase_'+spectrum_number + '_' +␣
↪str(int(minimum_segment_spec_1)):
                                params[param].set(expr= 'etalon_1_phase_'+spectrum_
↪number + '_' + str(int(minimum_segment_spec_1)))

                if 'etalon_2_period_' in param:
                        #Sets the period of etalon_1 to be equal for all spectra
                        if param != 'etalon_2_period_1_' + str(int(minimum_segment_spec_
↪1)):
                                params[param].set(expr='etalon_2_period_1_' +␣
↪str(int(minimum_segment_spec_1)))

                if 'etalon_3_period_' in param:
```

(continues on next page)

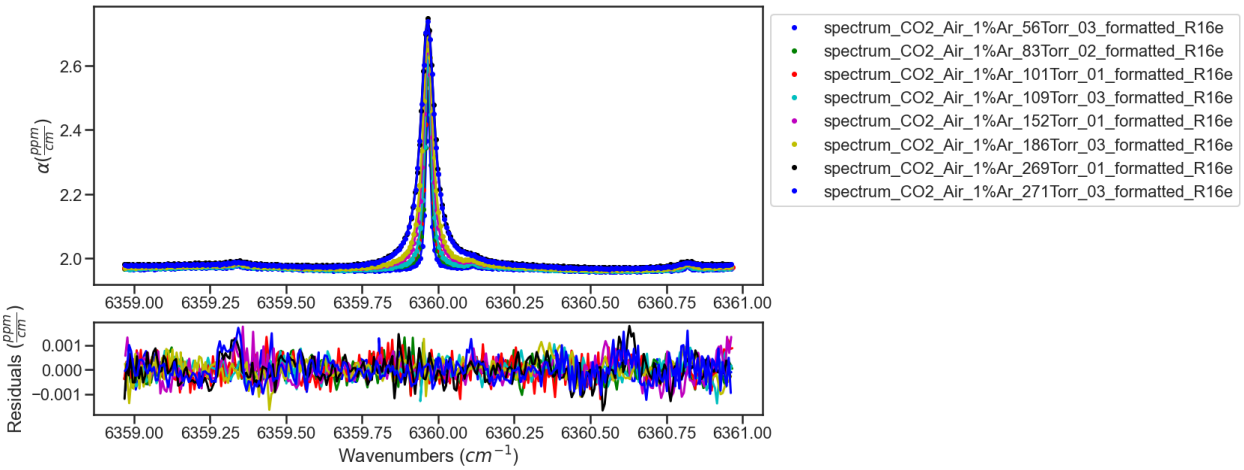
(continued from previous page)

```
                        #Sets the period of etalon_1 to be equal for all spectra
                        if param != 'etalon_3_period_1_' + str(int(minimum_segment_spec_
↪1)):
                                params[param].set(expr='etalon_3_period_1_' +␣
↪str(int(minimum_segment_spec_1)))


        result = fit_data.fit_data(params, wing_cutoff = 25)


        fit_data.residual_analysis(result, indv_resid_plot=False)
        fit_data.update_params(result)
        SPECTRA.generate_summary_file(save_file = True)
        SPECTRA.plot_model_residuals()
        counter+=1

        if counter == iterations:
                print (result.params.pretty_print())
```

ITERATION 1



ITERATION 2



Compare to Literature

The Lin et al. work reported spectra with QFs between 11000 and 29000 for pressures between 1.33kPa and 26.7kPa. The current example work was conducted over a slightly higher pressure range of 7.5 to 36 kPa and had QFs of about 1500.

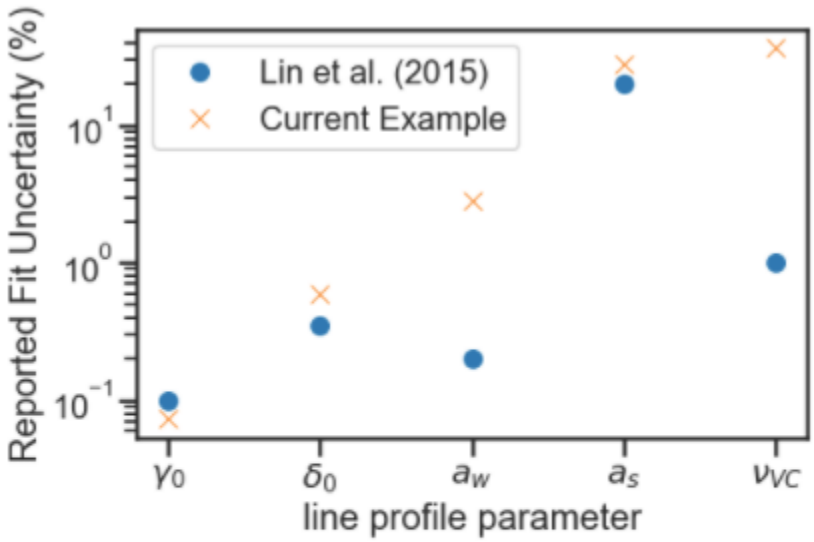
The Lin et al. work reports uncertainty on the collisional broadening of 0.1%, the pressure shift of 0.35%, the speed-dependent broadening of 0.2%, the speed-dependent shift of 20%, and the Dicke narrowing of 1%. The code below compares the results from the Lin et al. and current fit example including a plot highlighting the reported relative uncertainty in line shape parameters reported by the fits. The current example shows much higher fit uncertainties for speed-dependent width and Dicke narrowing, which indicates that at this QF and pressure range the spectra used in this analysis have a difficult time distinguishing between these narrowing mechanisms. This motivates a Monte Carlo analysis that explores how improving the SNR and pressure range would decrease the uncertainty in the reported line shape parameters.

```
Compare_Results = pd.DataFrame()
Compare_Results['Parameters'] = ['gamma0_air', 'delta0_air', 'SD_gamma_air', 'SD_delta_
↪air', 'nuVC_air']

#Get Spectra line number from the parameter results list.  Alternatively, could read in␣
↪the Linelist file and parse the same way as the initial Parameter linelist
for param in result.params:
        if ('SD_gamma' in param) and (result.params[param].vary == True):
                line_number = param.replace('SD_gamma_air_line_', '')


for parameter in Compare_Results['Parameters'].unique():
        # Get Lin et al Parameter values and uncertainties
        Compare_Results.loc[Compare_Results['Parameters'] == parameter, 'Lin et al. Value
↪'] =  PARAM_LINELIST[(PARAM_LINELIST['local_lower_quanta']=='R16e') & (PARAM_LINELIST[
↪'global_upper_quanta']==30012)][parameter].values[0]
        lin_reported_uncertainty = {'gamma0_air':0.1, 'delta0_air':0.35, 'SD_gamma_air':
↪0.2, 'SD_delta_air':20, 'nuVC_air':1}
        Compare_Results.loc[Compare_Results['Parameters'] == parameter, 'Lin et al.␣
↪Uncertainty (%)'] =  lin_reported_uncertainty[parameter]

        # Get Parameter results from fitting
        Compare_Results.loc[Compare_Results['Parameters'] == parameter, 'Fit Value'] = ␣
↪result.params[parameter + '_line_' + line_number].value
        Compare_Results.loc[Compare_Results['Parameters'] == parameter, 'Fit Uncertainty␣
↪(%)'] =  100*np.abs(result.params[parameter + '_line_' + line_number].stderr / result.
↪params[parameter + '_line_' + line_number].value)


Compare_Results['Fit Percent Difference (%)'] = 100*(Compare_Results['Fit Value'] -␣
↪Compare_Results['Lin et al. Value']) / Compare_Results['Fit Value']


plt.figure(figsize=(8,5))


plt.semilogy(Compare_Results['Parameters'], Compare_Results['Lin et al. Uncertainty (%)
↪'].values, 'o',label='Lin et al. (2015)')
```

(continues on next page)

(continued from previous page)

```
plt.semilogy(Compare_Results['Parameters'], Compare_Results['Fit Uncertainty (%)'].
↪values,  'x',label='Current Example')
ax = plt.gca()
ax.set_xticklabels(['$\gamma_{0}$', '$\delta_{0}$', '$a_{w}$', '$a_{s}$', '$\\nu_{VC}$'␣
↪])
plt.xlabel('line profile parameter')
plt.ylabel('Reported Fit Uncertainty (%)')

#plt.bar(Compare_Results['Parameters'], Compare_Results['Lin et al. Percent Difference (%)
↪'])
#plt.errorbar(Compare_Results['Parameters'], Compare_Results['Lin et al. Percent␣
↪Difference (%)'], yerr=Compare_Results['Lin et al. Uncertainty (%)'], fmt="o", color="r")

plt.show()

Compare_Results
```



	Parameters	Lin et al. Value	Lin et al. Uncertainty (%)	Fit Value	Fit Uncertainty (%)	Fit Percent Difference (%)
0	gamma0_air	0.074492	0.10	0.074399	0.072829	-0.124604
1	delta0_air	-0.005408	0.35	-0.005407	0.598812	-0.008841
2	SD_gamma_air	0.088400	0.20	0.094939	2.788409	6.887772
3	SD_delta_air	0.055000	20.00	0.057652	27.792124	4.600130
4	nuVC_air	0.003099	1.00	0.001806	36.417243	-71.636851

Monte Carlo Analysis

The Monte Carlo uncertainty analysis involves running numerous iterations fitting synthetic spectra at experimental conditions. To simplify the analysis the simulation line list only includes the R16e line and will use the results of the experimental fits as the simulation values.

```
# Generate linelist only using the R16e line and using the results from the experimental␣
↪fits
SIMULATION_LINELIST = PARAM_LINELIST[(PARAM_LINELIST['local_lower_quanta']=='R16e') &␣
↪(PARAM_LINELIST['global_upper_quanta']==30012)].copy()
SIMULATION_LINELIST.reset_index(inplace = True)


SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index ==0, 'gamma0_air'] = result.params[
↪'gamma0_air_line_' + line_number].value
SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index == 0, 'delta0_air'] = result.params[
↪'delta0_air_line_' + line_number].value
SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index == 0, 'nuVC_air'] = result.params[
↪'nuVC_air_line_' + line_number].value
SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index == 0, 'SD_gamma_air'] = result.params[
↪'SD_gamma_air_line_' + line_number].value
SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index == 0, 'SD_delta_air'] = result.params[
↪'SD_delta_air_line_' + line_number].value
SIMULATION_LINELIST.loc[SIMULATION_LINELIST.index == 0, 'y_air_296'] = 0
SIMULATION_LINELIST['nuVC_air']
```

Additionally, a dataframe summarizing the pressures and SNR (approximated by the QF from fits to the experimental spectra) are necessary for defining the simulation conditions. In this analysis we are comparing the synthetic results at experimetnal conditions, to those with three addtional low pressure spectra and with SNR of 5000 (about three times larger than observed in the experimental spectra).

```
# Pressure and QF of Fit Results for Simulation at Experiment Conditions
Exp_Simulation_conditions = pd.DataFrame()
Exp_Simulation_conditions['Spectrum Number'] = SPECTRA.get_list_spectrum_numbers()
for spec in SPECTRA.spectra:
        Exp_Simulation_conditions.loc[Exp_Simulation_conditions['Spectrum Number']==spec.
↪spectrum_number,'SNR']=spec.calculate_QF()
        Exp_Simulation_conditions.loc[Exp_Simulation_conditions['Spectrum Number']==spec.
↪spectrum_number,'Pressures']= np.around(spec.get_pressure_torr())

New_Simulation_conditions = Exp_Simulation_conditions.copy()
New_Simulation_conditions['SNR'] = 5000
new_pressures = pd.DataFrame(data = {'Pressures' : [5, 10, 25], 'SNR': [5000, 5000,␣
↪5000]})
New_Simulation_conditions = New_Simulation_conditions.append(new_pressures)
```

In the Monte Carlo simulation the inital guesses will be perturbed from the simulation value by the uncertainty reported in the experimental fit.

```
gamma0_err = Compare_Results[Compare_Results['Parameters'] == 'gamma0_air']['Fit␣
↪Uncertainty (%)'].values[0]/100 #per
delta0_err = Compare_Results[Compare_Results['Parameters'] == 'delta0_air']['Fit␣
↪Uncertainty (%)'].values[0]/100 #per
```

(continues on next page)

(continued from previous page)

```
aw_err = Compare_Results[Compare_Results['Parameters'] == 'SD_gamma_air']['Fit␣
↪Uncertainty (%)'].values[0]/100#per
as_err = Compare_Results[Compare_Results['Parameters'] == 'SD_delta_air']['Fit␣
↪Uncertainty (%)'].values[0]/100 #per
nuVC_err = Compare_Results[Compare_Results['Parameters'] == 'nuVC_air']['Fit Uncertainty␣
↪(%)'].values[0]/100 #per
```

We define a function to generate spectrum based on the input simulation conditions (SNR and pressure) and with a given input linelist.

```
def gen_spec(simulation_conditions, i, simulation_linelist):
        sample_molefraction =  {2:0.0004254}
        wave_min = 6358.972 #cm-1
        wave_max = 6360.963 #cm-1
        wave_space = 0.006772 #cm-1
        wave_error = 4.67e-7

        etalons = {1:[0.004321,1.168], 2:[0.001377, 59.38], 3:[0.0004578, 29.75]}

        spec = MATS.simulate_spectrum(simulation_linelist,
                                                        baseline_terms = [0,0],
                                                        wave_min = wave_min, wave_max =␣
↪wave_max,  wave_space = wave_space, wave_error = wave_error,
                                                        SNR = simulation_conditions['SNR
↪'].values[i], etalons = etalons,
                                                        temperature = 23.25,
                                                        pressure = simulation_conditions[
↪'Pressures'].values[i],
                                                        wing_cutoff = 25,  wing_method =
↪'wing_cutoff',
                                                        filename = str(int(simulation_
↪conditions['Pressures'].values[i])) + 'torr',
                                                        molefraction = sample_
↪molefraction,
                                                        natural_abundance = True,␣
↪nominal_temperature = 296, IntensityThreshold = 1e-30, num_segments = 1)
        return spec
```

The **mc_** function defined below simply takes the simulation conditions and number of iterations as inputs. Spectra are simulated at the conditions outlined in the simulation condition dataframe for just the R16e line using the linelist determined from the fits to the experimental spectra. Again, the inital guesses for the fitting are perturbed randomly at the magnitude of the reported fit error of the fits to the experimental data. The fits are conducted in the same manner as those to the experimental data without the use of the segment function. Results from each iteration are saved to a MC_result .csv file.

```
def mc_(iterations, simulation_conditions, MC_result_file):
        '''
        Runs the MC simulation for given number of iterations using the pressures and␣
↪SNR defined in simulation condition dataframe for the R16e line and saves results to␣
↪file
        '''
        MC_Results = pd.DataFrame()
```

(continues on next page)

(continued from previous page)

```
        MC_Results['Iterations'] = np.arange(0, iterations)
        MC_Results['gamma0_air'] = SIMULATION_LINELIST['gamma0_air'].values[0]
        MC_Results['delta0_air'] = SIMULATION_LINELIST['delta0_air'].values[0]
        MC_Results['SD_gamma_air'] = SIMULATION_LINELIST['SD_gamma_air'].values[0]
        MC_Results['SD_delta_air'] = SIMULATION_LINELIST['SD_delta_air'].values[0]
        MC_Results['nuVC_air'] = SIMULATION_LINELIST['nuVC_air'].values[0]
        MC_Results['y_air_296'] = SIMULATION_LINELIST['y_air_296'].values[0]

        for iteration in np.arange(0, iterations):

                #Read in Possible linelists
                PARAM_LINELIST = SIMULATION_LINELIST.copy()

                if len(simulation_conditions) ==8:
                        spec_1 = gen_spec(simulation_conditions, 0, SIMULATION_LINELIST)
                        spec_2 = gen_spec(simulation_conditions, 1, SIMULATION_LINELIST)
                        spec_3 = gen_spec(simulation_conditions, 2, SIMULATION_LINELIST)
                        spec_4 = gen_spec(simulation_conditions, 3, SIMULATION_LINELIST)
                        spec_5 = gen_spec(simulation_conditions, 4, SIMULATION_LINELIST)
                        spec_6 = gen_spec(simulation_conditions, 5, SIMULATION_LINELIST)
                        spec_7 = gen_spec(simulation_conditions, 6, SIMULATION_LINELIST)
                        spec_8 = gen_spec(simulation_conditions, 7, SIMULATION_LINELIST)
                        SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4, spec_5,␣
↪spec_6, spec_7, spec_8], 'Monte Carlo Analysis',PARAM_LINELIST)
                elif len(simulation_conditions) == 11:
                        spec_1 = gen_spec(simulation_conditions, 0, SIMULATION_LINELIST)
                        spec_2 = gen_spec(simulation_conditions, 1, SIMULATION_LINELIST)
                        spec_3 = gen_spec(simulation_conditions, 2, SIMULATION_LINELIST)
                        spec_4 = gen_spec(simulation_conditions, 3, SIMULATION_LINELIST)
                        spec_5 = gen_spec(simulation_conditions, 4, SIMULATION_LINELIST)
                        spec_6 = gen_spec(simulation_conditions, 5, SIMULATION_LINELIST)
                        spec_7 = gen_spec(simulation_conditions, 6, SIMULATION_LINELIST)
                        spec_8 = gen_spec(simulation_conditions, 7, SIMULATION_LINELIST)
                        spec_9 = gen_spec(simulation_conditions, 8, SIMULATION_LINELIST)
                        spec_10 = gen_spec(simulation_conditions, 9, SIMULATION_LINELIST)
                        spec_11 = gen_spec(simulation_conditions, 10, SIMULATION_
↪LINELIST)
                        SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4, spec_5,␣
↪spec_6, spec_7, spec_8, spec_9, spec_10, spec_11], 'Monte Carlo Analysis',PARAM_
↪LINELIST)









                #Generate Baseline Parameter list based on number of etalons in spectra␣
↪definitions and baseline order
                BASE_LINELIST = SPECTRA.generate_baseline_paramlist()



                gamma0_err = Compare_Results[Compare_Results['Parameters'] == 'gamma0_air
↪']['Fit Uncertainty (%)'].values[0]/100 #per
```

(continues on next page)

(continued from previous page)

```
                delta0_err = Compare_Results[Compare_Results['Parameters'] == 'delta0_air
↪']['Fit Uncertainty (%)'].values[0]/100 #per
                aw_err = Compare_Results[Compare_Results['Parameters'] == 'SD_gamma_air
↪']['Fit Uncertainty (%)'].values[0]/100#per
                as_err = Compare_Results[Compare_Results['Parameters'] == 'SD_delta_air
↪']['Fit Uncertainty (%)'].values[0]/100 #per
                nuVC_err = Compare_Results[Compare_Results['Parameters'] == 'nuVC_air'][
↪'Fit Uncertainty (%)'].values[0]/100 #per


                PARAM_LINELIST.loc[PARAM_LINELIST['molec_id'] == 2, 'gamma0_air'] =␣
↪PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['gamma0_air'].values*(1 + np.random.
↪normal(loc = 0, scale =1, size = len(PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['sw
↪']))*gamma0_err)
                PARAM_LINELIST.loc[PARAM_LINELIST['molec_id'] == 2, 'delta0_air'] =␣
↪PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['delta0_air'].values*(1 + np.random.
↪normal(loc = 0, scale =1, size = len(PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['sw
↪']))*delta0_err)
                PARAM_LINELIST.loc[PARAM_LINELIST['molec_id'] == 2, 'SD_gamma_air'] =␣
↪PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['SD_gamma_air'].values*(1 + np.random.
↪normal(loc = 0, scale =1, size = len(PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['sw
↪']))*aw_err)
                PARAM_LINELIST.loc[PARAM_LINELIST['molec_id'] == 2, 'SD_delta_air'] =␣
↪PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['SD_delta_air'].values*(1 + np.random.
↪normal(loc = 0, scale =1, size = len(PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['sw
↪']))*as_err)
                PARAM_LINELIST.loc[PARAM_LINELIST['molec_id'] == 2, 'nuVC_air'] = PARAM_
↪LINELIST[PARAM_LINELIST['molec_id']==2]['nuVC_air'].values*(1 + np.random.normal(loc =␣
↪0, scale =1, size = len(PARAM_LINELIST[PARAM_LINELIST['molec_id']==2]['sw']))*nuVC_err)



                FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_
↪LINELIST, lineprofile = 'SDNGP', linemixing = False,
                                                                                    ␣
↪          fit_intensity = Fit_Intensity, threshold_intensity = IntensityThreshold, sim_
↪window = wave_range,
                                                                                    ␣
↪          nu_constrain = True, sw_constrain = False, gamma0_constrain = True, delta0_
↪constrain = True,
                                                                                    ␣
↪           aw_constrain = True, as_constrain = True,
                                                                                    ␣
↪           nuVC_constrain = True, eta_constrain =True, linemixing_constrain = True,
                                                                                    ␣
↪                additional_columns = ['local_lower_quanta'])



                FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {2:{1:True}
↪}, vary_sw = {2:{1:True}},
                                                                                    ␣
↪                          vary_gamma0 = {2:{1: True}},
```

(continues on next page)

(continued from previous page)

```
                                                                                ␣
↪                       vary_delta0 = {2:{1: True}},
                                                                                ␣
↪                       vary_aw = {2:{1: True}},
                                                                                ␣
↪                       vary_as = {2:{1: True}},
                                                                                ␣
↪                       vary_nuVC = {2:{1:True}},
                                                                                ␣
↪                       vary_eta = {}, vary_linemixing = {2:{1:False}})
            FITPARAMS.generate_fit_baseline_linelist(vary_baseline = True, vary_
↪molefraction = {2:False}, vary_pressure = False, vary_temperature = False, vary_xshift␣
↪= False,
                                                                                ␣
↪       vary_etalon_amp= True, vary_etalon_period= True, vary_etalon_phase= True)


            fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_
↪LineList', minimum_parameter_fit_intensity = Fit_Intensity)
            params = fit_data.generate_params()


            result = fit_data.fit_data(params, wing_cutoff = 25)

            fit_data.residual_analysis(result, indv_resid_plot=False)
            fit_data.update_params(result)
            #SPECTRA.generate_summary_file(save_file = True)
            #SPECTRA.plot_model_residuals()


            #line Shape Parameters
            MC_Results.loc[MC_Results['Iterations']== iteration, 'gamma0_air_fit']␣
↪=result.params['gamma0_air_line_0'].value
            MC_Results.loc[MC_Results['Iterations']== iteration, 'gamma0_air_fit_err
↪'] =result.params['gamma0_air_line_0'].stderr
            MC_Results.loc[MC_Results['Iterations']== iteration, 'delta0_air_fit'] =␣
↪result.params['delta0_air_line_0'].value
            MC_Results.loc[MC_Results['Iterations']== iteration, 'delta0_air_fit_err
↪'] = result.params['delta0_air_line_0'].stderr
            MC_Results.loc[MC_Results['Iterations']== iteration, 'nuVC_air_fit'] =␣
↪result.params['nuVC_air_line_0'].value
            MC_Results.loc[MC_Results['Iterations']== iteration, 'nuVC_air_fit_err']␣
↪= result.params['nuVC_air_line_0'].stderr
            MC_Results.loc[MC_Results['Iterations']== iteration, 'SD_gamma_air_fit']␣
↪=result.params['SD_gamma_air_line_0'].value
            MC_Results.loc[MC_Results['Iterations']== iteration, 'SD_gamma_air_fit_
↪err'] =result.params['SD_gamma_air_line_0'].stderr
            MC_Results.loc[MC_Results['Iterations']== iteration, 'SD_delta_air_fit']␣
↪=result.params['SD_delta_air_line_0'].value
            MC_Results.loc[MC_Results['Iterations']== iteration, 'SD_delta_air_fit_
↪err'] =result.params['SD_delta_air_line_0'].stderr
```

(continues on next page)

(continued from previous page)

```
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_1'] = spec_1.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_2'] = spec_2.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_3'] = spec_3.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_4'] = spec_4.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_5'] = spec_5.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_6'] = spec_6.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_7'] = spec_7.
↪calculate_QF()
                MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_8'] = spec_8.
↪calculate_QF()
                if len(simulation_conditions) == 11:
                        MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_9'] =␣
↪spec_9.calculate_QF()
                        MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_10'] =␣
↪spec_10.calculate_QF()
                        MC_Results.loc[MC_Results['Iterations']== iteration, 'QF_11'] =␣
↪spec_11.calculate_QF()


                MC_Results.to_csv(MC_result_file + '.csv', index = False)
```

Below are the calls to the **mc_** function at the experimental and new simulation conditions, which will generate and fit 50 iterations of synthetic data for each case.

```
mc_(50, Exp_Simulation_conditions, 'Experimental_Conditions_MC')
mc_(50, New_Simulation_conditions, 'New_Conditions_MC')
```

The parameter_eval function summarizes the MC results comparing the fit average and standard deviation across the iterations to the reported fit uncertainty on each iteration. This also calculates the fractional error compared to the simulation value. Plots of summary and dataframe are generated for visualization.

```
def parameter_eval(df, report_df):
        '''
        Summarizes MC results for a given parameter
        '''
        params = ['gamma0_air', 'delta0_air', 'SD_gamma_air', 'SD_delta_air', 'nuVC_air']
        fig = plt.figure(constrained_layout=True, figsize = [20,3])
        spec = gridspec.GridSpec(ncols=5, nrows=1, figure=fig)
        ax1 = fig.add_subplot(spec[0, 0])
        ax2 = fig.add_subplot(spec[0,1])
        ax3 = fig.add_subplot(spec[0,2])
        ax4 = fig.add_subplot(spec[0,3])
        ax5 = fig.add_subplot(spec[0,4])
```

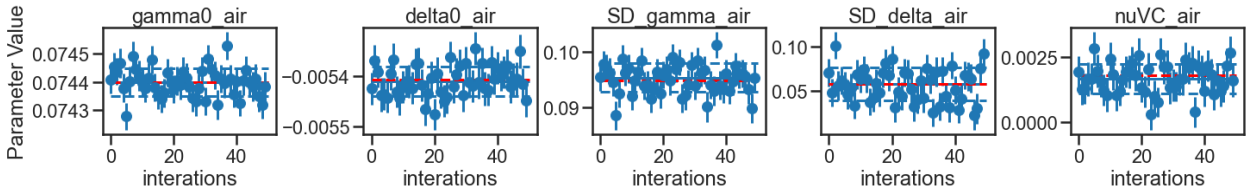
(continues on next page)

(continued from previous page)

```
        axes = [ax1, ax2, ax3, ax4, ax5]
        count =0
        for simulated_column in params:
                param_avg = df[simulated_column + '_fit'].median()
                param_std = df[simulated_column + '_fit'].std()
                param_err_avg = df[simulated_column + '_fit_err'].median()

                if df[simulated_column].values[0] == 0:
                        param_frac_err = 0
                        param_fit_err = 0
                else:
                        param_frac_err = 100*(param_avg - df[simulated_column].
↪values[0]) / df[simulated_column].values[0]
                        param_fit_err = np.abs(100*param_std / param_avg)
                if param_err_avg == 0:
                        param_err_ratio = 0
                else:
                        param_err_ratio = param_std/param_err_avg

                report_df.loc[report_df['Reported Values']==simulated_column, 'Simulated␣
↪Value'] = df[simulated_column].values[0]
                report_df.loc[report_df['Reported Values']==simulated_column, 'Parameter␣
↪Fit Average'] = param_avg
                report_df.loc[report_df['Reported Values']==simulated_column, 'Parameter␣
↪Fit StDev'] = param_std
                report_df.loc[report_df['Reported Values']==simulated_column, 'Parameter␣
↪Error Average'] = param_err_avg
                report_df.loc[report_df['Reported Values']==simulated_column, 'Parameter␣
↪Fractional Error'] = param_frac_err
                report_df.loc[report_df['Reported Values']==simulated_column, 'Parameter␣
↪Fit Error'] = param_fit_err
                report_df.loc[report_df['Reported Values']==simulated_column, 'Error␣
↪Ratio'] = param_err_ratio

                ax = axes[count]
                ax.errorbar(df['Iterations'], df[simulated_column + '_fit'], fmt = 'o',␣
↪yerr = df[simulated_column + '_fit_err'])
                ax.hlines(param_avg, 0, 50)
                ax.hlines(param_avg + param_std, 0, 50, linestyles = 'dashed')
                ax.hlines(param_avg - param_std, 0, 50, linestyles = 'dashed')
                ax.hlines(df[simulated_column].values[0], 0, 50, linestyles = 'dashed',␣
↪colors = 'r')
                ax.set_title(simulated_column)
                ax.set_xlabel('interations')

                ax.ticklabel_format(useOffset=False)
                count+=1
        ax1.set_ylabel('Parameter Value')

        plt.show()

        return report_df
```

The Experimental parameter result table shows good agreement in the fit uncertainty reported in the experimental data and averaged over the iterations of the MC analysis conducted at experimental conditions. This also provides a measure of the ratio between the standard deviations in the fit values across the iterations compared to the average fit uncertainty. Ratios greater than one indicate that the fit uncertainty reported by one iteration might under represent the uncertainty, most likely because of correlation between parameters that aren't included in the calculation of the fit uncertainty. This ratio, when greater than one, could act as a correction factor to increase the experiment fit uncertainty.

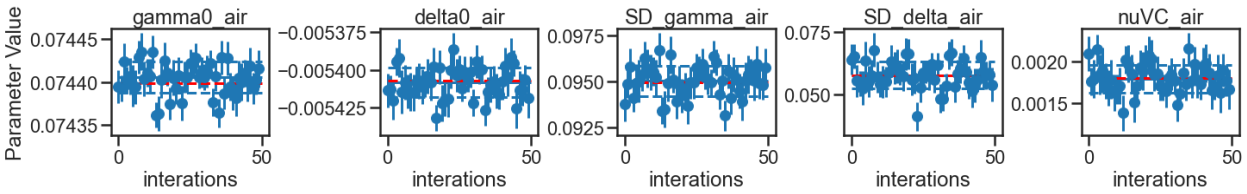
```
exp_MC = pd.read_csv('Experimental_Conditions_MC.csv')
new_MC = pd.read_csv('New_Conditions_MC.csv')

EXP_SUMMARY = pd.DataFrame()
EXP_SUMMARY['Reported Values'] = ['gamma0_air', 'delta0_air', 'SD_gamma_air', 'SD_delta_
↪air', 'nuVC_air']
EXP_SUMMARY['Simulated Value'] = 0
EXP_SUMMARY['Parameter Fit Average'] = 0
EXP_SUMMARY['Parameter Fit StDev'] = 0
EXP_SUMMARY['Parameter Fractional Error'] = 0
EXP_SUMMARY['Parameter Fit Error'] = 0
EXP_SUMMARY['Error Ratio'] = 0

NEW_SUMMARY = EXP_SUMMARY.copy()

EXP_SUMMARY = parameter_eval(exp_MC, EXP_SUMMARY)
EXP_SUMMARY
```



	Reported Values	Simulated Value	Parameter Fit Average	Parameter Fit StDev	Parameter Fractional Error	Parameter Fit Error	Error Ratio	Parameter Error Average
0	gamma0_air	0.074399	0.074400	0.000050	0.001393	0.067188	0.985763	0.000051
1	delta0_air	-0.005407	-0.005411	0.000029	0.073471	0.542051	0.913536	0.000032
2	SD_gamma_air	0.094939	0.095413	0.002568	0.498602	2.691814	1.023063	0.002510
3	SD_delta_air	0.057652	0.057898	0.018668	0.426623	32.242316	1.161376	0.016074
4	nuVC_air	0.001806	0.001647	0.000564	-8.783515	34.216014	0.926340	0.000608

By comparing the average fit uncertainties over MC simulations in the new conditions (higher SNR and 3 extra pressures) to the experimental conditions indicate that the proposed improvements to the experiments can be expected to reduce the uncertainties in the spectroscopic parameters by factors of 2.8 to 3.7. The largest improvement would be to the Dicke narrowing parameter, which is aided by not only the higher SNR, but also by the inclusion of additional low-pressure spectra that fall in a domain where the Dicke narrowing mechanism dominates over the speed-dependent narrowing.

```
NEW_SUMMARY= parameter_eval(new_MC, NEW_SUMMARY)
NEW_SUMMARY['Improvement Over Experiment'] = EXP_SUMMARY['Parameter Fit Error'].values/
↪NEW_SUMMARY['Parameter Fit Error'].values
NEW_SUMMARY
```



	Reported Values	Simulated Value	Parameter Fit Average	Parameter Fit StDev	Parameter Fractional Error	Parameter Fit Error	Error Ratio	Parameter Error Average	Improvement Over Experiment
0	gamma0_air	0.074399	0.074405	0.000018	0.008798	0.024280	0.959518	0.000019	2.767236
1	delta0_air	-0.005407	-0.005408	0.000010	0.016886	0.183127	0.887421	0.000011	2.959978
2	SD_gamma_air	0.094939	0.095043	0.000821	0.109324	0.864239	0.891607	0.000921	3.114665
3	SD_delta_air	0.057652	0.057801	0.005355	0.258189	9.263983	0.894288	0.005988	3.480395
4	nuVC_air	0.001806	0.001789	0.000168	-0.916053	9.363885	0.785498	0.000213	3.654040

Applying Analytical Correction to hardness of collisions for HTP

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

Dicke narrowing accounts for effective narrowing of the doppler width because of velocity changing collisions. The magnitude of this effect is dependent on the hardness of the collisions. Different line shapes make different assumptions on the hardness of the collision with the Nelkhin-Ghatak making the assumption that collisions are hard, meaning that the molecular velocity before a collision is forgotten after the collision, and the Galatry profile assuming collisions are soft, meaning that many collisions are necessary to change the velocity. In reality, the hardness of the collision is related to the perturbers and absorbers being studied and acts on a continuum. In a paper by Konefał et al. (citation below), authors map the hard Dicke Narrowing term onto the billiard ball collision model, based on a parameter beta, which is related to the pressure (parameterized as the Dicke narrowing at a pressure relative to the doppler width) and the ratio of the masses between the perturber and absorber. As Beta can be calculated, this correction provides a more robust modeling of the Dicke Narrowing without the addition of additional floated parameters.

M. Konefał, M. Słowiński, M. Zaborowski, R. Ciuryło, D. Lisak, P. Wcisło, Analytical-function correction to the Hartmann–Tran profile for more reliable representation of the Dicke-narrowed molecular spectra, Journal of Quantitative Spectroscopy and Radiative Transfer, Volume 242, 2020,106784,ISSN 0022-4073.

The Beta correction has been included in the MATS software and the following example provides an example of how to use it.

Simulate Spectrum Objects

Module import follows from the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples with additional details on how to simulate spectra found in *Fitting Synthetic Spectra* and the source documentation.

After the generic parameters are introduced the line list for simulations is read in using the `LoadLineListData()` function. In this example we are making some adjustments to the line list before simulating. In the MATS fitting, simulations are only conducted to some user-defined range outside of the spectral frequency range (wave_range = 1.5 cm^{-1} in the current example). This same constraint is not imposed when simulating spectra. This can lead to some far-wing issues between simulation and fitting if care is not taken to match the wave_range to the lines in the parameter linelist and the simulation wing_cutoff magnitude, which should also match the fit wing_cutoff magnitude. For this example, we are truncating the simulation line list to [wave_min - wave_range, wave_max + wave_range]. Additionally, values for the Dicke Narrowing air and self parameters are set and the line mixing terms are set to 0.

```
from MATS.linelistdata import linelistdata
from MATS import simulate_spectrum
```

(continues on next page)

(continued from previous page)

```
#Generic Fit Parameters
wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-25 #intensities must be above this value for the line to be fit
order_baseline_fit = 1
wave_min = 6326 #cm-1
wave_max = 6328 #cm-1
wave_space = 0.005 #cm-1
baseline_terms = [0] #polynomial baseline coefficients where the index is equal to the␣
↪coefficient order
etalon = {}

#Read in  linelists
PARAM_LINELIST = linelistdata['CO2_30012']
##Adjust the linelist before simualting spectra
PARAM_LINELIST = pd.read_csv('CO2_30012.csv')
PARAM_LINELIST = PARAM_LINELIST[(PARAM_LINELIST['nu']>= wave_min - wave_range) & (PARAM_
↪LINELIST['nu']<= wave_min + wave_range)]
PARAM_LINELIST.loc[(PARAM_LINELIST['sw'] > Fit_Intensity)& (PARAM_LINELIST['local_iso_id
↪'] == 1) & (PARAM_LINELIST['molec_id'] == 2), 'nuVC_air'] = 0.018596
PARAM_LINELIST.loc[(PARAM_LINELIST['sw'] > Fit_Intensity) & (PARAM_LINELIST['local_iso_id
↪'] == 1) & (PARAM_LINELIST['molec_id'] == 2), 'nuVC_self'] = 0.01993
PARAM_LINELIST.loc[(PARAM_LINELIST['sw'] > Fit_Intensity) & (PARAM_LINELIST['local_iso_id
↪'] == 1) & (PARAM_LINELIST['molec_id'] == 2), 'y_self_296'] = 0
PARAM_LINELIST.loc[(PARAM_LINELIST['sw'] > Fit_Intensity) & (PARAM_LINELIST['local_iso_id
↪'] == 1) & (PARAM_LINELIST['molec_id'] == 2), 'y_air_296'] = 0
```

For the purpose of showing the mechanism, we will be simulating data for CO_2, air, and 10% CO_2 in air samples between 1 and 500 torr. The structure of the script follows that outlined in the *Fitting Synthetic Spectra* example.

The *simulate_spectrum()* function is used to generate several spectrum class objects. The beta_formalism variable is set to True, which simulates the spectra accounting for the change in Dicke Narrowing with the hardness of the collision. The synthetic spectra span 3 samples: CO_2, air, and 10% CO_2 in air. The air and CO_2 samples can use the diluent = 'air' and 'self' variables as the Diluent variable can be auto-generated. However, the 10% CO_2 in air sample requires the explicit definition of the Diluent variable, which is a dictionary of dictionaries where each diluent is a dictionary with composition and mass (m) keys. This allows the calculation of the mass of perturber, which is necessary for the β implementation. The use of the self term will generate a warning to consider switching to an explicitly name the broadener. This warning is to have the user consider if this is causing ambiguity or unanticipated behavior. Specifically, this could be an issue in multi-species fits or if the explict broadener and self term are both used. To remove the use of the self broadening, have the line shape parameter file with the explicit broadener (ie. gamma0_CO2 instead of gamma0_self) and use the Diluent sample definition as modeled for the 10% CO_2 in air sample (ie pure CO_2 would be Diluent = {'CO2':{'composition':1, 'm':43.98983} }).

```
#Synthetic Air Simulation
print ('Synthetic Air Samples')
spec_1 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 1, diluent = 'air',filename = 'Air_1_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :400*1e-6},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
```

(continues on next page)

(continued from previous page)

```
spec_2 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 10, diluent = 'air',filename = 'Air_10_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :400*1e-6},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_3 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 100, diluent = 'air',filename = 'Air_100_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :400*1e-6},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_4 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 250, diluent = 'air',filename = 'Air_250_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :400*1e-6},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_5 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 500, diluent = 'air',filename = 'Air_500_torr',
                   wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :400*1e-6},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
#Mixture of CO2 and Air
## Note that we had to use the Diluent method opposedc to the diluent method. ␣
↪Additionally, the use of the self broadening term will generate some warnings
print ('Mixture Samples')
spec_6 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 1, filename = 'Mix_1_torr',
                    Diluent = {'self':{'composition':0.1, 'm':43.98983} , 'air' :{
↪'composition':0.9, 'm':28.95734}},
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :0.1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_7=  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 10, filename = 'Mix_10_torr',
                    Diluent = {'self':{'composition':0.1, 'm':43.98983} , 'air' :{
↪'composition':0.9, 'm':28.95734}},
```

(continues on next page)

(continued from previous page)

```
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :0.1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_8 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 100, filename = 'Mix_100_torr',
                    Diluent = {'self':{'composition':0.1, 'm':43.98983} , 'air' :{
↪'composition':0.9, 'm':28.95734}},
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :0.1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_9 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error =␣
↪wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 250,filename = 'Mix_250_torr',
                    Diluent = {'self':{'composition':0.1, 'm':43.98983} , 'air' :{
↪'composition':0.9, 'm':28.95734}},
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :0.1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_10 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 500, filename = 'Mix_500_torr',
                         Diluent = {'self':{'composition':0.1, 'm':43.98983} , 'air' :{
↪'composition':0.9, 'm':28.95734}},
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :0.1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)

#Pure CO2.  Notice that the use of the self broadening term will cause some warnings.
print ('Pure CO2 Samples')
spec_11 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 1, diluent = 'self', filename = 'CO2_1_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_12 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 10, diluent = 'self',filename = 'CO2_10_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
```

(continues on next page)

(continued from previous page)

```
spec_13 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 100, diluent = 'self',filename = 'CO2_100_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers',  molefraction =
↪{2 :1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_14 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 250,  diluent = 'self', filename = 'CO2_250_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
spec_15 =  simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_space, wave_error␣
↪= wave_error,
                    SNR = SNR, baseline_terms = baseline_terms, temperature = 25,␣
↪pressure = 500,  diluent = 'self', filename = 'CO2_500_torr',
                    wing_cutoff = 25, wing_method = 'wing_wavenumbers', molefraction =
↪{2 :1},
                    etalons = etalon, natural_abundance = True, nominal_temperature =␣
↪296, IntensityThreshold = 1e-30, num_segments = 1, beta_formalism = True)
```

Generate Dataset and Fit Parameter Files

Following the procedure outlined in *Fitting Synthetic Spectra* and *Fitting Experimental Spectra*. This example is using the SDNGP without line mixing, is constraining all parameters to multi-spectrum fits, and is floating the line center, line intensity, collisional broadening, pressure shift, speed-dependent broadening, and Dicke narrowing terms.

```
SPECTRA = MATS.Dataset([spec_1, spec_2, spec_3, spec_4, spec_5, spec_6, spec_7, spec_8,␣
↪spec_9, spec_10, spec_11, spec_12, spec_13, spec_14, spec_15], 'CO2 Study',PARAM_
↪LINELIST )

#Generate Baseline Parameter list based on number of etalons in spectra definitions and␣
↪baseline order
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()
FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,␣
↪lineprofile = 'SDNGP', linemixing = False,
                                fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
                                nu_constrain = True, sw_constrain = True, gamma0_
↪constrain = True, delta0_constrain = True,
                                 aw_constrain = True, as_constrain = True,
                                 nuVC_constrain = True, eta_constrain =True, linemixing_
↪constrain = True)
                                #additional_columns = ['trans_id', 'local_lower_quanta', 'm'])

FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {2:{1:True, 2:False, 3:
↪False}}, vary_sw = {2:{1:True, 2:False, 3:False}},
```

(continues on next page)

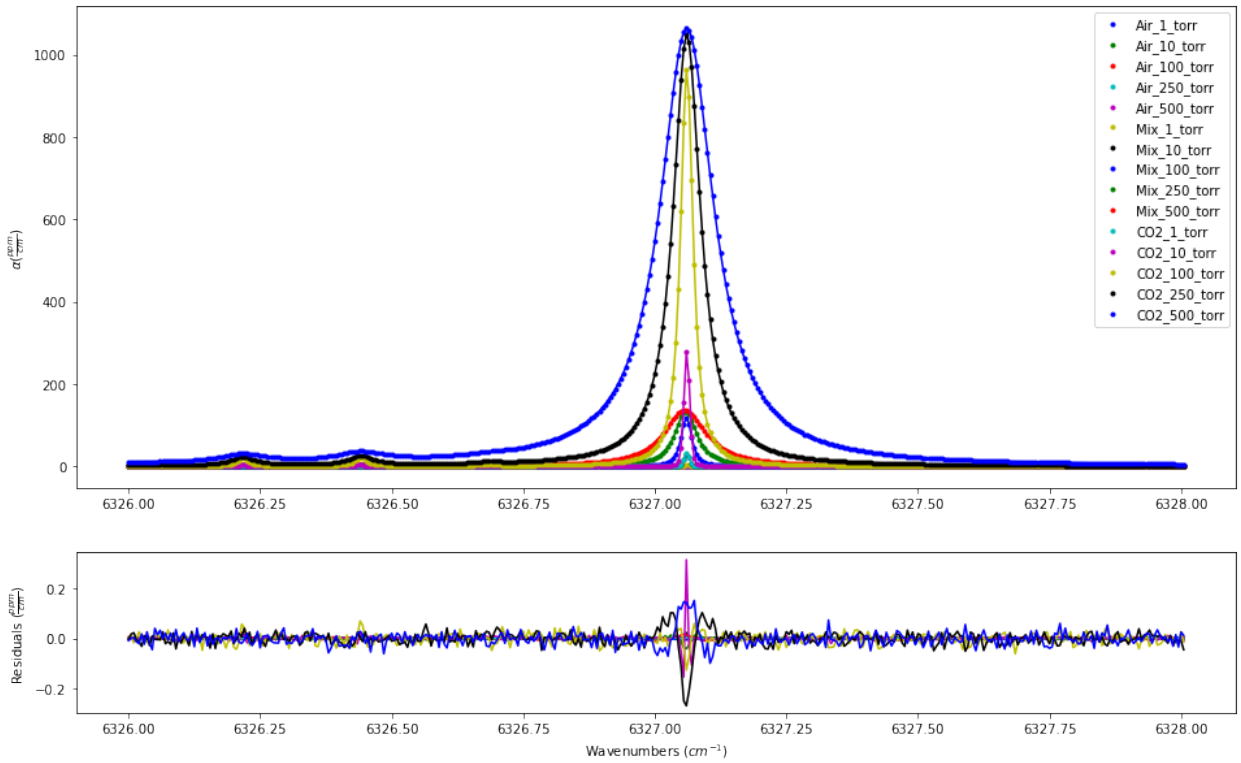
(continued from previous page)

```
                                                    vary_gamma0 = {2:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma0 = {2:{1:False}},
                                                    vary_delta0 = {2:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_delta0 = {2:{1:False}},
                                                    vary_aw = {2:{1: True, 2:False, 3:␣
↪False}, 1:{1:False}}, vary_n_gamma2 = {2:{1:False}},
                                                    vary_as = {}, vary_n_delta2 = {2:{1:
↪False}},
                                                    vary_nuVC = {2:{1:True}}, vary_n_nuVC =
↪{2:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {2:{1:
↪False}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = False, vary_molefraction = {7:
↪False, 1:Fa   lse}, vary_xshift = False,
                                    vary_etalon_amp= False, vary_etalon_period= False,␣
↪vary_etalon_phase= False)
```

Fit Data

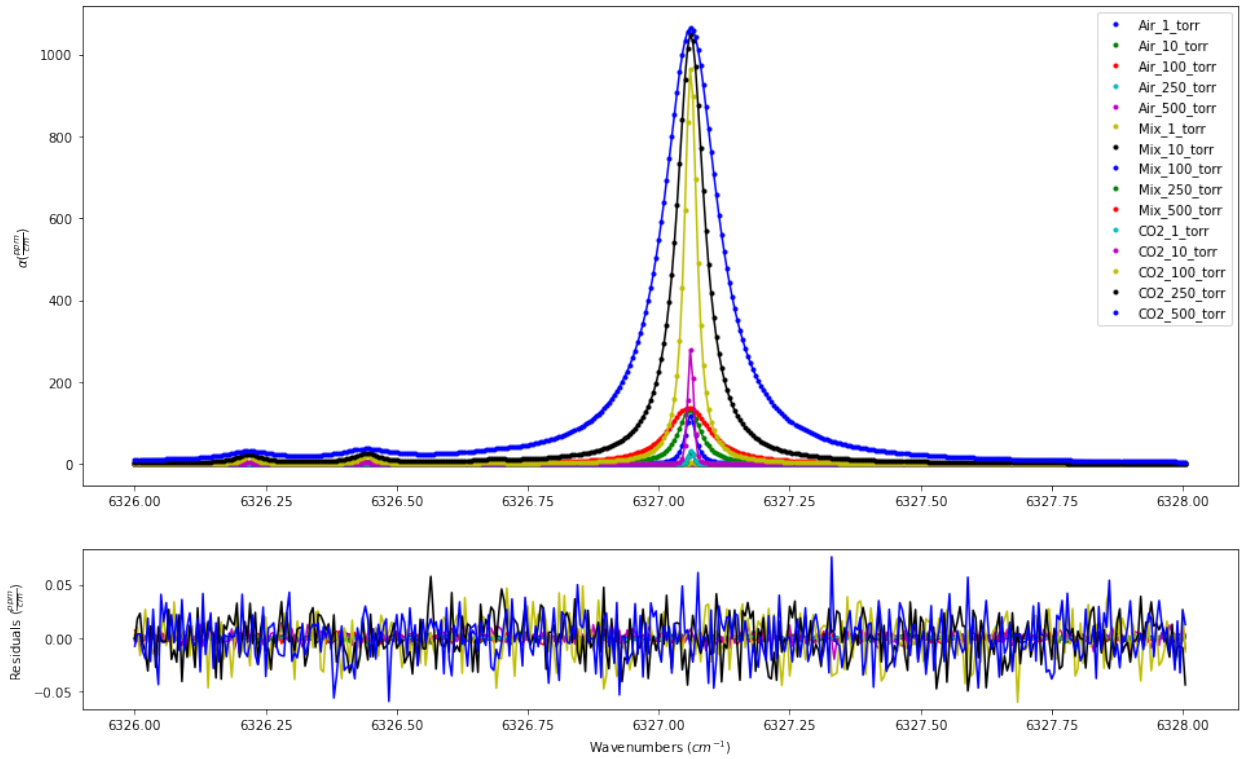
In this first iteration, we are fitting the data without including the beta_formalism to correct for the hardness of the collisions by setting the beta_formalism to False in the *Fit_DataSet* instance, which is the default. The results show systematic residuals at the line core based on the large range of pressures.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = 1e-24,
                    beta_formalism = False)
params = fit_data.generate_params()
result = fit_data.fit_data(params, wing_cutoff = 25,wing_method = 'wing_wavenumbers')
#print (result.params.pretty_print())

fit_data.residual_analysis(result, indv_resid_plot=False)
fit_data.update_params(result, param_linelist_update_file = 'Results without Beta')
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



In the second iteration, the beta_formalism term is set to True. Unsuprisingly, using the beta correction gives better results as this is how the spectra were simulated.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = 1e-24,
                    beta_formalism = True)
params = fit_data.generate_params()
result = fit_data.fit_data(params, wing_cutoff = 25,wing_method = 'wing_wavenumbers')
#print (result.params.pretty_print())

fit_data.residual_analysis(result, indv_resid_plot=False)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



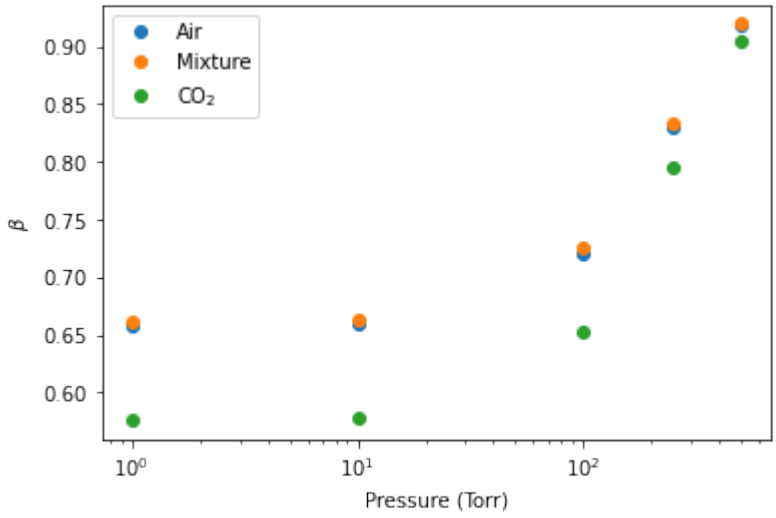
The *Fit_DataSet.generate_beta_output_file()* definition can be used to tabulate a value of the math:*beta* parameters that are used for all lines and each spectrum. By default this is saved in a file called Beta Summary File.csv. This plot shows how this math:*beta* parameter changes with sample and pressure.

```
fit_data.generate_beta_output_file()
beta_summary_file = pd.read_csv('Beta Summary File.csv', index_col = 0)
beta_summary_file[beta_summary_file['nuVC_air_vary']==True]

pressures = [1,10, 100, 250, 500]
air_beta = [beta_summary_file[beta_summary_file.index == 62]['Beta_1'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_2'].values[0], beta_summary_
↪file[beta_summary_file.index == 62]['Beta_3'].values[0],
        beta_summary_file[beta_summary_file.index == 62]['Beta_4'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_5'].values[0]]
mix_beta = [beta_summary_file[beta_summary_file.index == 62]['Beta_6'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_7'].values[0], beta_summary_
↪file[beta_summary_file.index == 62]['Beta_8'].values[0],
        beta_summary_file[beta_summary_file.index == 62]['Beta_9'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_10'].values[0]]
CO2_beta = [beta_summary_file[beta_summary_file.index == 62]['Beta_11'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_12'].values, beta_summary_file[beta_
↪summary_file.index == 62]['Beta_13'].values[0],
        beta_summary_file[beta_summary_file.index == 62]['Beta_14'].values[0], beta_
↪summary_file[beta_summary_file.index == 62]['Beta_15'].values[0]]

plt.semilogx(pressures, air_beta, 'o', label = 'Air')
plt.semilogx(pressures, mix_beta, 'o', label = 'Mixture')
plt.semilogx(pressures, CO2_beta, 'o', label = 'CO$_{2}$')
plt.xlabel('Pressure (Torr)')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('$\\beta$')
plt.legend()
plt.show()
```



Incorporating CIA into fits

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

Collision induced absorption (CIA) is a broadband effect stemming from intermolecular interactions leading to a collisionally-induced dipole. Traditionally, CIA is treated as the remaining absorption after modeling baseline and resonant absorption. To accomodate this approach, MATS allows for the input of a static CIA for each spectrum to account for in the fitting solution. The example below takes synthetic spectra that include CIA, evalutates using the initial line list, generates a CIA for each spectrum, and then fits the line shape parameters.

Generate Spectrum Objects

Module import follows from the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples with additional details on how to simulate spectra found in *Fitting Synthetic Spectra* and the source documentation.

The provided spectra were simulated in MATS offline using the Drouin 2017 Multispectrum analysis of the Oxygen A-Band line list and also includes the CIA reported in that work. The code below generates *Spectrum* objects from these spectra.

```
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-26#intensities must be above this value for the line to be fit
wave_range = 1.5 #range outside of experimental x-range to simulate
segment_column = None
etalon = {}
sample_concentration = {7:  0.209}



freq_column = 'Wavenumber + Noise (cm-1)'
```

(continues on next page)

(continued from previous page)

```
tau_column = 'Alpha + LM + CIA'
pressure_column = 'Pressure (Torr)'
temp_column = 'Temperature (C)'
order_baseline_fit = 2



spec_1 = MATS.Spectrum(  '100 Torr',
                         molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                         etalons = etalon,
                         input_freq = False, frequency_column = freq_column,
                         input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                         pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                         nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_2 = MATS.Spectrum(  '200 Torr',
                         molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                         etalons = etalon,
                         input_freq = False, frequency_column = freq_column,
                         input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                         pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                         nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_3 = MATS.Spectrum(  '400 Torr',
                         molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                         etalons = etalon,
                         input_freq = False, frequency_column = freq_column,
                         input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                         pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                         nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_4 = MATS.Spectrum(  '700 Torr',
                         molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                         etalons = etalon,
                         input_freq = False, frequency_column = freq_column,
                         input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                         pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                         nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_5 = MATS.Spectrum(  '1000 Torr',
```

(continues on next page)

(continued from previous page)

```
                        molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                        etalons = etalon,
                        input_freq = False, frequency_column = freq_column,
                        input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                        pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                        nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_6 = MATS.Spectrum(  '2000 Torr',
                        molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                        etalons = etalon,
                        input_freq = False, frequency_column = freq_column,
                        input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                        pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                        nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
spec_7 = MATS.Spectrum(  '3000 Torr',
                        molefraction = sample_concentration, natural_abundance = True,␣
↪diluent = 'air',
                        etalons = etalon,
                        input_freq = False, frequency_column = freq_column,
                        input_tau = False, tau_column = tau_column, tau_stats_column =␣
↪None,
                        pressure_column = pressure_column, temperature_column = temp_
↪column,#segment_column = segment_column,
                        nominal_temperature = 296, x_shift = 0, baseline_order = order_
↪baseline_fit)
```

Generate Dataset

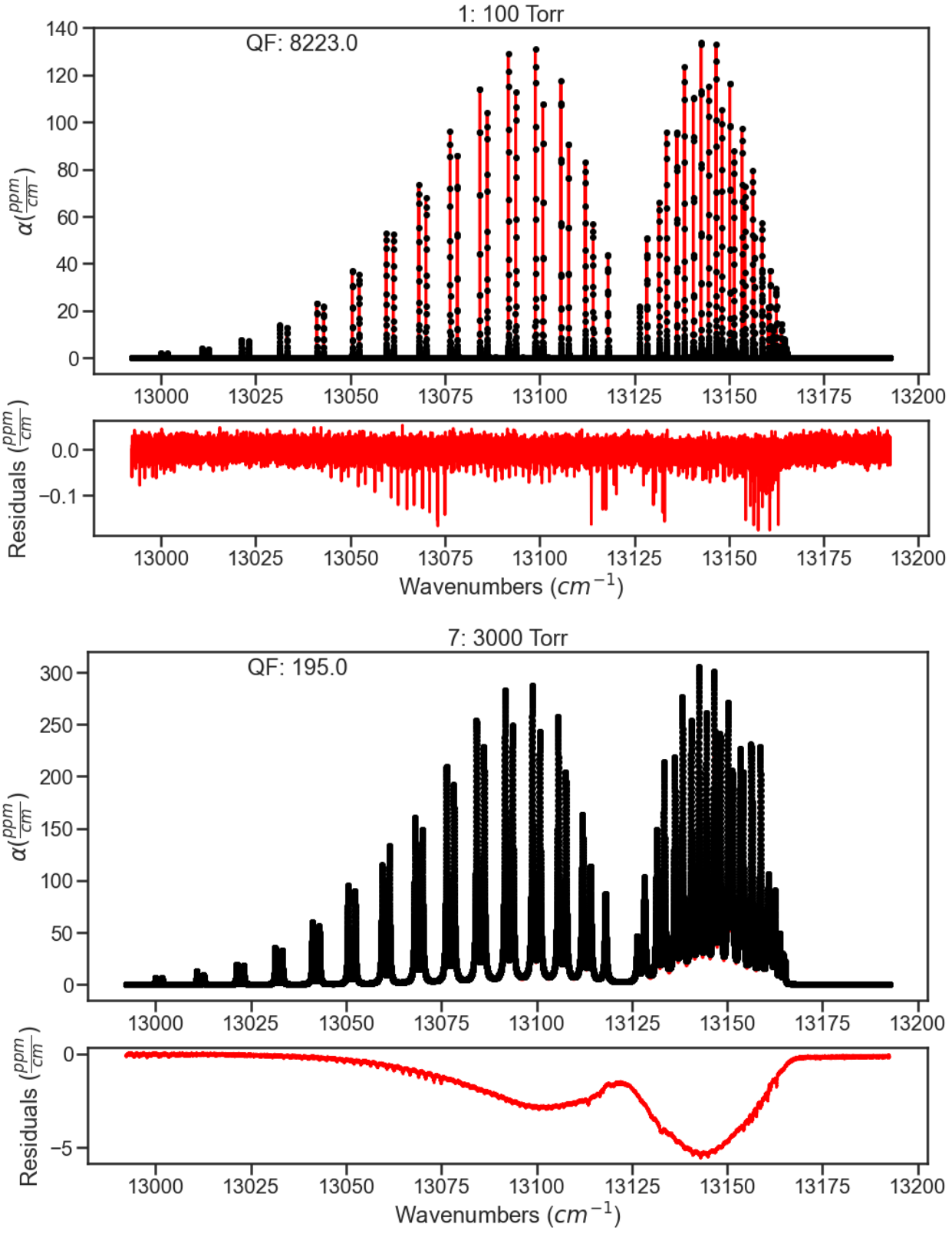
A line list generated from the Drouin 2017 Oxygen A-Band data combined with HITRAN 2016 values is used as the initial guess. A similar line list was used in the generation of the spectra, with truncation errors being the most like cause of discrepencies. Spectra are combined to form a *Dataset* object.

```
from MATS.linelistdata import linelistdata
PARAM_LINELIST = linelistdata['O2_ABand_Drouin_2017_linelist']

SPECTRA = Dataset([spec_1, spec_2, spec_3, spec_4, spec_5, spec_6, spec_7], 'CIA Spectra␣
↪Study', PARAM_LINELIST)

#Generate Baseline Parameter list based on number of etalons in spectra definitions and␣
↪baseline order
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()
```

Evaluation

Inital fit doesn't float any parameters. The residual plots shown are only for the lowest and highest pressures. In the high pressure residuals, you can see that the unaccounted for CIA is dominating the residuals and wouldn't allow for optimization of line shape parameters without treatment.

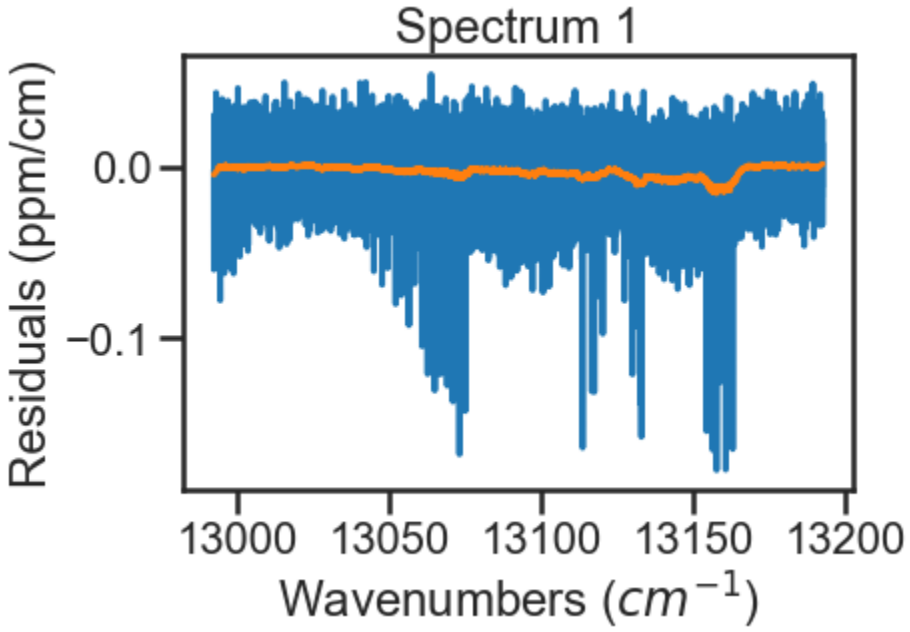
```
FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,
                                lineprofile = 'SDVP', linemixing = True,
                                   fit_intensity = Fit_Intensity, threshold_intensity =␣
↪Fit_Intensity,
                                   nu_constrain = True, sw_constrain = True, gamma0_
↪constrain = True, delta0_constrain = True,
                                    aw_constrain = True, as_constrain = True,
                                    nuVC_constrain = True, eta_constrain =True,␣
↪linemixing_constrain = True)


FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {7:{1:False, 2:False, 3:
↪False}}, vary_sw = {7:{1:False, 2:False, 3:False}},
                                                    vary_gamma0 = {7:{1: False, 2:False, 3:␣
↪False}}, vary_n_gamma0 = {7:{1:False}},
                                                    vary_delta0 = {7:{1: False, 2:False, 3:␣
↪False}}, vary_n_delta0 = {7:{1:False}},
                                                    vary_aw = {7:{1: False, 2:False, 3:␣
↪False}}, vary_n_gamma2 = {7:{1:False}},
                                                    vary_as = {7:{1:False}}, vary_n_delta2␣
↪= {7:{1:False}},
                                                    vary_nuVC = {7:{1:False}}, vary_n_nuVC␣
↪= {7:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {7:{1:
↪False}}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = False, vary_molefraction = {7:
↪False, 1:False}, vary_xshift = False,
                                    vary_etalon_amp= False, vary_etalon_period= False,␣
↪vary_etalon_phase= False)


fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList',
                    minimum_parameter_fit_intensity = Fit_Intensity)
params = fit_data.generate_params()


result = fit_data.fit_data(params, wing_wavenumbers = 50, wing_method = 'wing_wavenumbers
↪', xtol = 1e-7, maxfev = 2000, ftol = 1e-7)

fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
```

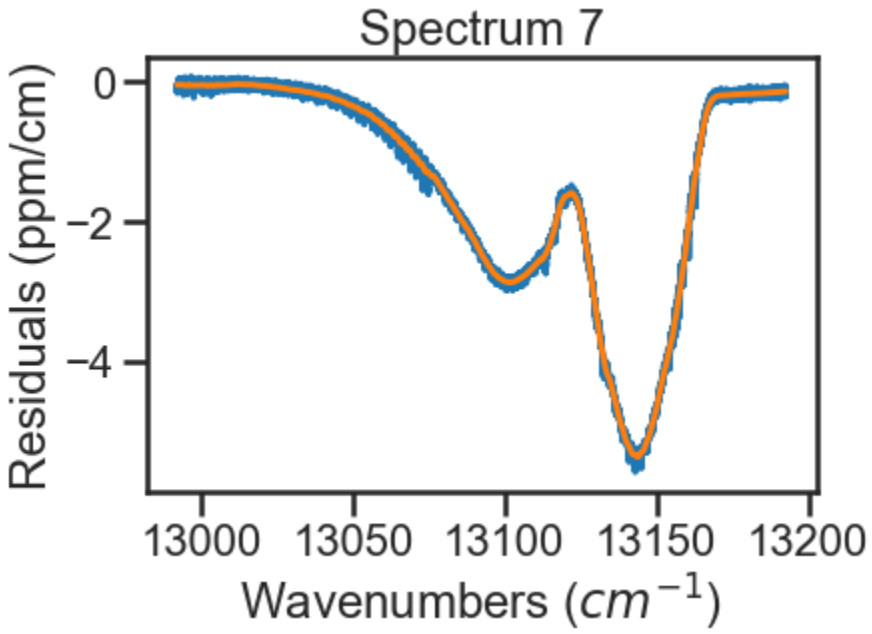


A simple function is defined to treat the CIA as the smooth residuals. To do this we will use a filter to smooth over any resonant absorbance and then define the CIA as the negative of what is leftover. This is done for all spectra making up the dataset. Note that this definiinition includes the *Spectrum.set_cia()* function to assign this array to a given

Spectrum instance. Again the plots below show this process for the lowest and highest pressure spectra, where the blue lines are the residuals and the orange lines are the smoothed residuals that will define the magnitude of the CIA.

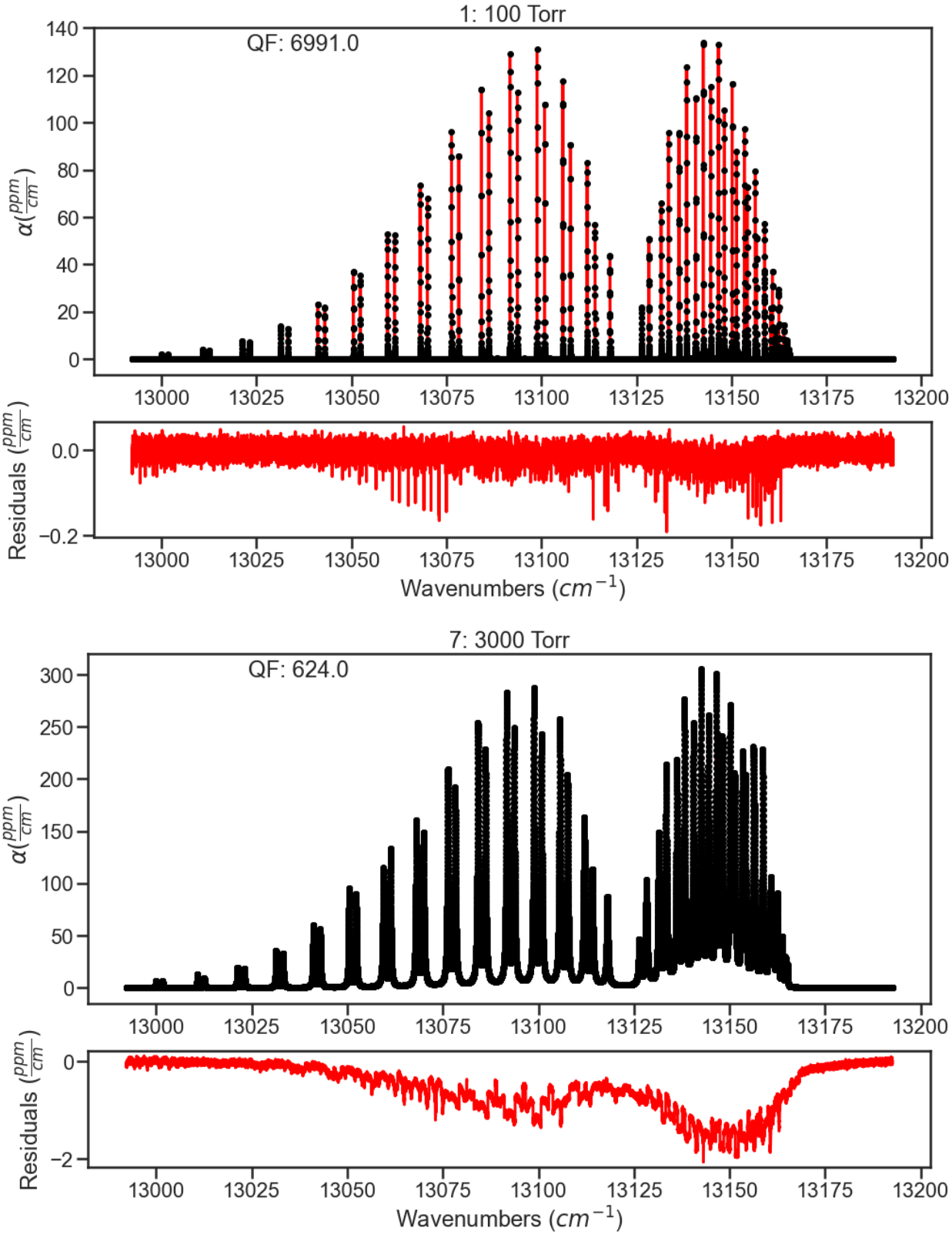
```
from scipy.signal import savgol_filter

def ad_hoc_CIA(spec):

 waves = spec.wavenumber
 resds = spec.residuals

 CIA = savgol_filter(resds, 501, 1) # Savitsky Golay Filte that uses a 501 pt sliding␣
↪window with a first order polynomial to smooth the data.  These settings were␣
↪arbitrary to smooth out the data.
 plt.plot(waves, resds)
 plt.plot(waves, CIA)
 plt.xlabel('Wavenumbers ($cm^{-1}$)')
 plt.ylabel('Residuals (ppm/cm)')
 plt.title( 'Spectrum '+ str(spec.spectrum_number))
 plt.show()
 spec.set_cia(-1*CIA)
ad_hoc_CIA(spec_1)
ad_hoc_CIA(spec_2)
ad_hoc_CIA(spec_3)
ad_hoc_CIA(spec_4)
ad_hoc_CIA(spec_5)
ad_hoc_CIA(spec_6)
ad_hoc_CIA(spec_7)
```





The spectra are evaluated again and we see that the residuals have improved. To further improve fits, I would optimize line parameters in small wavenumber sections using the baseline parameter to account for remaining CIA. These line shape parameters could then be used in evaluation of the full band (like above) and then the CIA column could be ammended based on those results. Iterating on these steps should allow for optimization of both the line shape parameters and the CIA.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList',
                    minimum_parameter_fit_intensity = Fit_Intensity)
params = fit_data.generate_params()
result = fit_data.fit_data(params, wing_wavenumbers = 25, wing_method = 'wing_cutoff',␣
↪xtol = 1e-7, maxfev = 2000, ftol = 1e-7)
fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
```


1: 100 Torr
QF: 6991.0
$\alpha(\frac{ppm}{cm})$
Residuals $(\frac{ppm}{cm})$
Wavenumbers (cm^{-1})
7: 3000 Torr
QF: 624.0
$\alpha(\frac{ppm}{cm})$
Residuals $(\frac{ppm}{cm})$
Wavenumbers (cm^{-1})

Including instrument line shapes in simulation and fits

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

MATS has added the capability to simulate and fit spectra using an instrument lineshape. MATS automatically loads the slit functions defined in HAPI, but could use any slit function with the form slit_function(x, resolution) or slit_function(x, [resolutions]). MATS uses a slight variation of the HAPI convolveSpectrumSame, where the **arange_** function was redefined to address an integer/float bug in the underlying np.linspace call. This implementation currently assumes an equal wavenumber spacing based on the application of the slit function calculation.

The example below simulates the same spectrum and applies the different HAPI instrument line shapes. The simulated molefraction is perturbted and then floated during a fit.

Simulate Spectra with ILS

Module import follows from the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples with additional details on how to simulate spectra found in *Fitting Synthetic Spectra* and the source documentation.

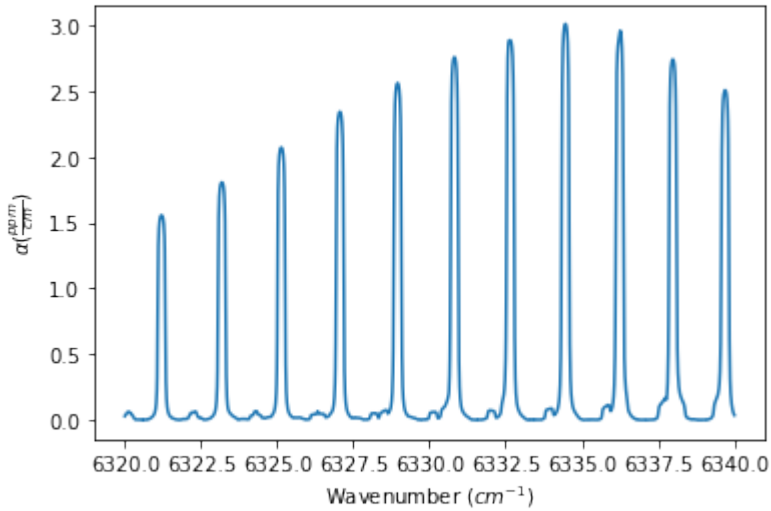
```
from MATS.linelistdata import linelistdata
import MATS.hapi as hapi

# Shared Simulation and Fit Parameters
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-24#intensities must be above this value for the line to be fit
wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
segment_column = None
order_baseline_fit = 1
etalon = {}
SNR = 5000
resolution = 0.25
molefraction = {2:0.01}
wave_min = 6320
wave_max = 6340
wave_step = 0.01
```

When fitting spectra, the simulated spectral range is the minimum and maximum wavenumbers in your dataset +/- an extra spectral window defined in this example by the wave_range parameter. This is used to truncate the input line list when generating the Parameter linelist used in fitting to increase the fit speed. However, this trunctation doesn't occur when simulating spectra, which can lead to far-wing effects as the default is to calculate each transition to +/- 25 half-widths from the line center. For this example, we are truncating input line list used for the simulation to match the fit simulation window. Alternatively, the wave_range parameter could be increased substantially to include these far-wing impacts.

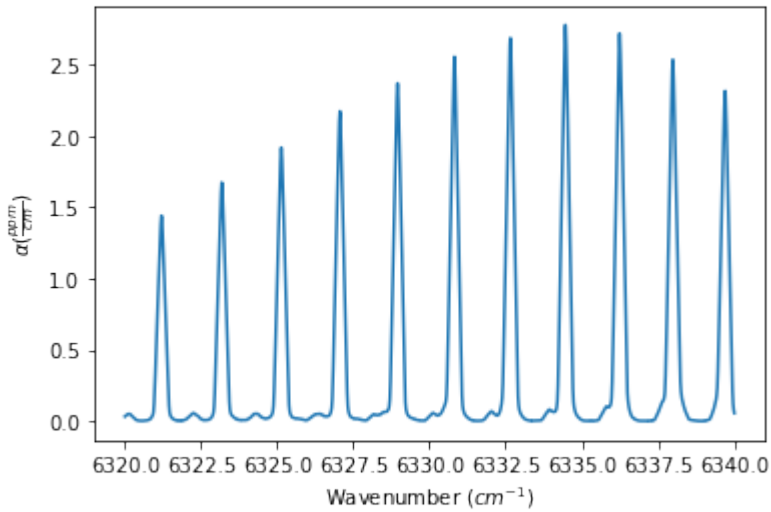
```
PARAM_LINELIST = linelistdata['CO2_30012']
PARAM_LINELIST = PARAM_LINELIST[(PARAM_LINELIST['nu'] <= wave_max + wave_range) & (PARAM_
↪LINELIST['nu']>= wave_min - wave_step)]
```

The *Spectrum* object definitions below show how to use the ILS_function, ILS_resolution, and ILS_wing (defines the length of the ILS slit function to use in the convolution, such that the length is +/- ILS_wing / spectrum stepsize) to define the ILS function. The outputs below show the generated spectrum for each ILS_function

```
spec_rectangular =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                            SNR = SNR, baseline_terms = [0], temperature = 22.85,
                            pressure = 150,
                             filename = 'Rectangular',
                           molefraction =  {2:0.01} , ILS_function = SLIT_RECTANGULAR,␣
↪ILS_resolution = resolution, ILS_wing = 10)
```




```
spec_triangular=  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                            SNR = SNR, baseline_terms = [0], temperature = 22.85,
                            pressure = 150,
                             filename = 'Triangular',
                           molefraction =  {2:0.01}, ILS_function = SLIT_TRIANGULAR,␣
↪ILS_resolution = resolution, ILS_wing = 10)
```

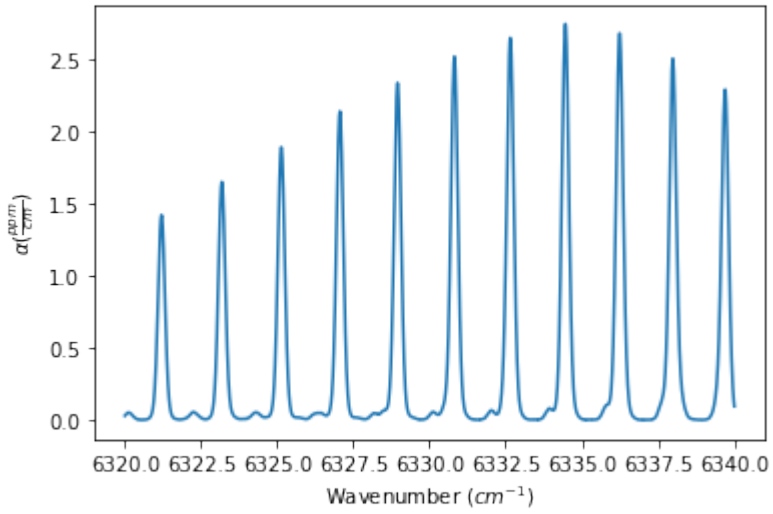



```
spec_gaussian =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                            SNR = SNR, baseline_terms = [0], temperature = 22.85,
                            pressure = 150,
```

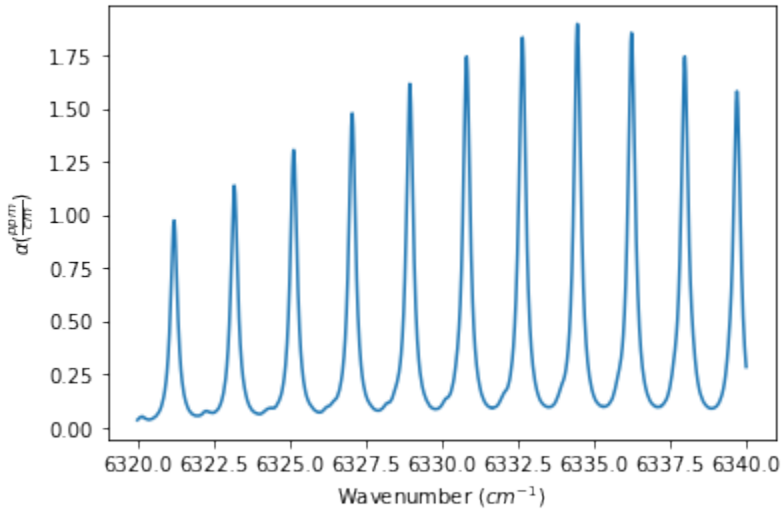
(continues on next page)

(continued from previous page)

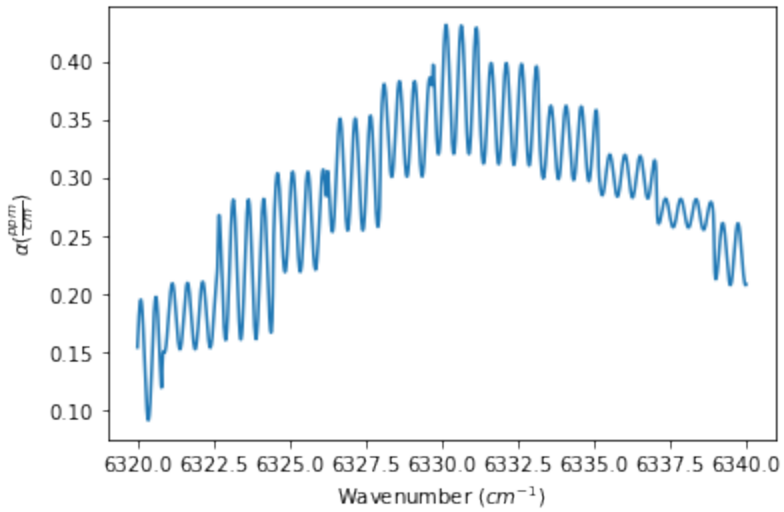
```
                                filename = 'Gaussian',
                               molefraction =  {2:0.01}, ILS_function = SLIT_GAUSSIAN, ILS_
↪resolution = resolution, ILS_wing = 10)
```



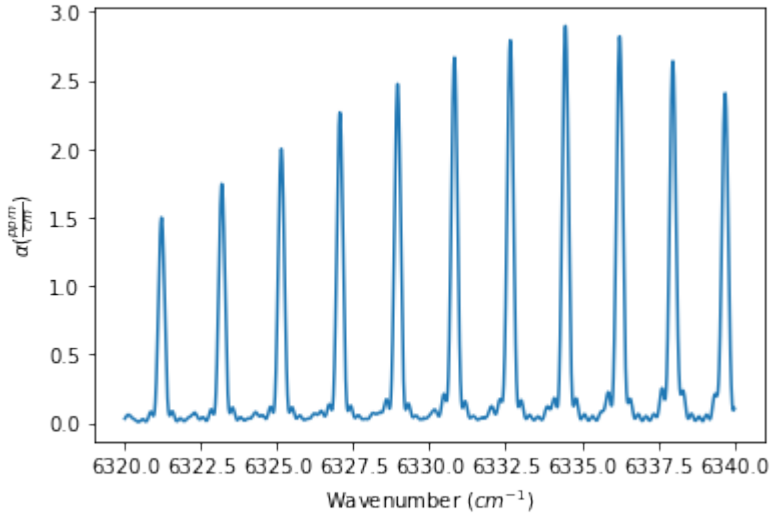

```
spec_dispersion =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                                SNR = SNR, baseline_terms = [0], temperature = 22.85,
                                pressure = 150,
                                 filename = 'Dispersion',
                               molefraction =  {2:0.01}, ILS_function = SLIT_DISPERSION,␣
↪ILS_resolution = resolution, ILS_wing = 10)
```



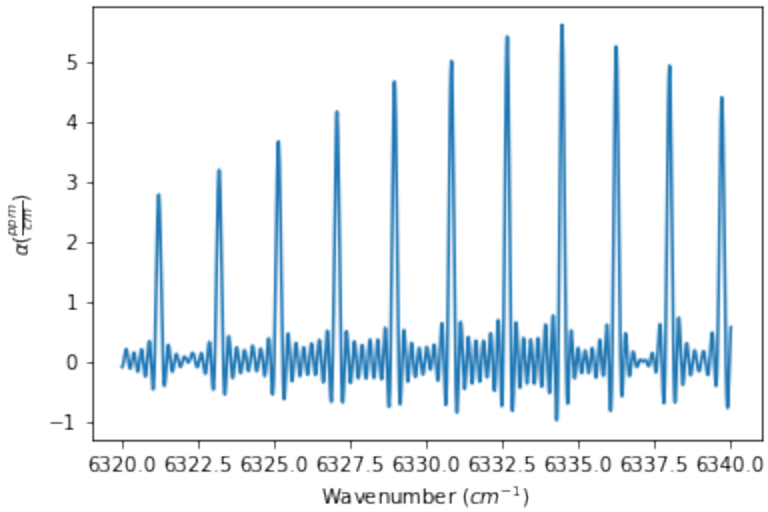

```
spec_cosinus =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                                SNR = SNR, baseline_terms = [0], temperature = 22.85,
                                pressure = 150,
                                 filename = 'Cosinus',
                               molefraction =  {2:0.01}, ILS_function = SLIT_COSINUS, ILS_
↪resolution = resolution, ILS_wing = 10)
```




```
spec_diffraction =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                            SNR = SNR, baseline_terms = [0], temperature = 22.85,
                            pressure = 150,
                             filename = 'Diffraction',
                           molefraction =  {2:0.01}, ILS_function = SLIT_DIFFRACTION,␣
↪ILS_resolution = resolution, ILS_wing = 10)
```




```
spec_michelson =  MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                            SNR = SNR, baseline_terms = [0], temperature = 22.85,
                            pressure = 150,
                             filename = 'Michelson',
                           molefraction =  {2:0.01}, ILS_function = SLIT_MICHELSON, ILS_
↪resolution = resolution, ILS_wing = 10)
```



Generate Dataset and Fit Parameters

The dataset and fit parameters are generated with instances of the *Dataset* and

Generate_FitParam_File classes. Additionally, the initial CO2 molefraction is perturbed within 2% of the simulated mole fraction.

```
SPECTRA = MATS.Dataset([ spec_rectangular, spec_triangular, spec_gaussian, spec_
↪dispersion, spec_cosinus,
                         spec_diffraction, spec_michelson],  'ILS Study', PARAM_
↪LINELIST)
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()

BASE_LINELIST['molefraction_CO2'] = BASE_LINELIST['molefraction_CO2'].values*(1 + np.
↪random.normal(loc = 0, scale =1, size = len(BASE_LINELIST['molefraction_CO2']))*(2/
↪100)) #Adjust the mole fraction of each sample by a mole fraction of random value␣
↪normally distributed 2%




FITPARAMS = MATS.Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST,
                                        lineprofile = 'SDNGP', linemixing = True,
                                           fit_intensity = Fit_Intensity, threshold_
↪intensity = IntensityThreshold,
                                          nu_constrain = True, sw_constrain = True,␣
↪gamma0_constrain = True, delta0_constrain = True,
                                           aw_constrain = True, as_constrain = True,
                                           nuVC_constrain = True, eta_constrain =True,␣
↪linemixing_constrain = True)


FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {2:{1:False, 2:False, 3:
↪False}}, vary_sw = {2:{1:False, 2:False, 3:False}},
                                                    vary_gamma0 = {2:{1: False, 2:False, 3:␣
↪False}}, vary_n_gamma0 = {2:{1:False}},
```

(continues on next page)

(continued from previous page)

```
                                                    vary_delta0 = {2:{1:False, 2:False, 3:␣
↪False}}, vary_n_delta0 = {2:{1:False}},
                                                    vary_aw = {2:{1: False, 2:False, 3:␣
↪False}}, vary_n_gamma2 = {2:{1:False}},
                                                    vary_as = {2:{1:False}}, vary_n_delta2␣
↪= {2:{1:False}},
                                                    vary_nuVC = {2:{1:False}}, vary_n_nuVC␣
↪= {2:{1:False}},
                                                    vary_eta = {}, vary_linemixing = {2:{1:
↪False}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = False, vary_molefraction = {2:
↪True}, vary_ILS_res = False)
```

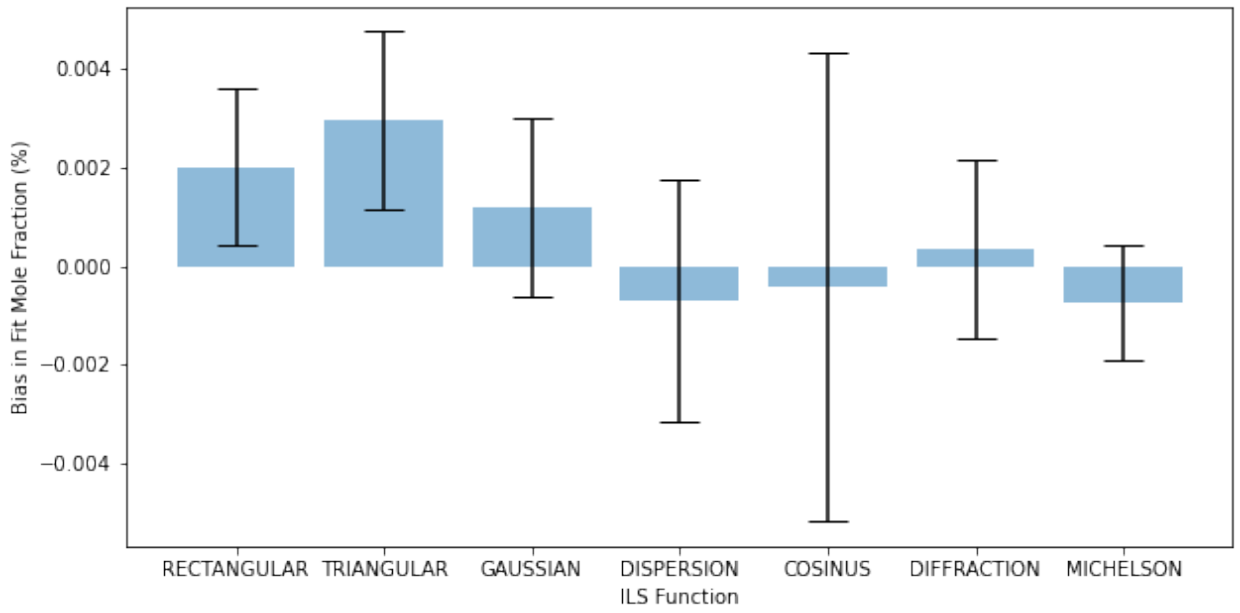
In the *Generate_FitParam_File.generate_fit_baseline_linelist()* function, in addition to the normal baseline parameters the ILS resolution parameters are also fittable parameters. Additionally, this is coded in such a way that each ILS_function has a unique resolution parameter. This allows for different spectra to have different ILS functions and the baseline linelist table adequately accounts for this. Subset of a *Generate_FitParam_File.generate_fit_baseline_linelist()* output highlights this. While floating the resolution parameters is possible, it should be done with caution as all other parameters are dependent on this value, so correlation and poor fits are likely.

```
Spectrum Number    SLIT_RECTANGULAR_res_0    SLIT_RECTANGULAR_res_0_err    SLIT_
↪RECTANGULAR_res_0_vary    SLIT_MICHELSON_res_0
1                      0.25                                          0                ␣
↪                                    False                                          ␣
↪      0
. . .
7                      0                                             0                ␣
↪                                    False                                          ␣
↪      0.25
```

Fit Data

In this example only the sample concentration is allowed to float, which is a spectrum dependent parameter. At this SNR all of the results yield fit mole fractions very similar to the simulated value. The plot below shows the relative Bias in the fit mole fraction with error bars equal to the relative fit uncertainty for each ILS. Note that this is only one iteration and not generalizable.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity)
params = fit_data.generate_params()
for param in params:
 if ('_res_' in param) and (params[param].vary == True):
     params[param].set(min = 0.01)

result = fit_data.fit_data(params)
print (result.params.pretty_print())
fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
```



Fitting and Simulating isotopes and molecules not in HITRAN

Provided in the MATS v2 release are several examples highlighting MATS capabilities, which can be found in the MATS examples folder.

HITRAN's focus is on atmospherically relevent molecules. This was the original focus of MATS, however recent feature development has allowed for adaption of to include non-HITRAN molecules.

The example shows how to supplement the HITRAN isotope list and then use that in simulations and fits. This example uses Mercury as an examaple case This example uses Mercury as an example case.

Append to the HITRAN isotope list to include desired isotopes

Module import follows from the *Fitting Experimental Spectra* and *Fitting Synthetic Spectra* examples.

The *add_to_HITRANstyle_isotope_list()* function allows you to ammend a given isotope list to include additional molecules and/or isotopes using the HITRAN-style molcule_id, local_isotope id, and global isotope id parameterization. This function has some checks to verify expected behavior - if the molecule id and isotope id are in the initial isotope line list, it will return 'Already entry with that molec_id and local_iso_id. This result will write over that entry. - if the molecule id is in the initial isotope list it will return 'This is being added as a new isotope of molecule name. Proceed if that was the intention. - if the global isotope id is already in the isotope list it will return, 'There is another entry with this global isotope id. Consider changing the value for consistency'

Mercury line shape parameters are not included in the HITRAN database. The code below uses *add_to_HITRANstyle_isotope_list()* function to add 10 Mercury isotopes to the HITRAN isotope list. This is saved in a new isotope list called HITRAN_Hg_isolist that can be used in simulating and fitting spectra. The HITRAN isotope list provides the HITRAN molecule and global ids that are already in use.

```
from MATS.hapi import ISO
from MATS import add_to_HITRANstyle_isotope_list

HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = ISO, molec_id =␣
↪100, local_iso_id = 1,
```

(continues on next page)

(continued from previous page)

```
                                                    global_isotope_id = 200, iso_name = '196
↪',
                                                    abundance = 0.0015, mass = 195.96581,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 2,
                                                    global_isotope_id = 201, iso_name = '198
↪',
                                                    abundance = 0.1004, mass =197.96674,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 3,
                                                    global_isotope_id = 202, iso_name =
↪'199A',
                                                    abundance = 0.1694, mass =198.96825,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 4,
                                                    global_isotope_id = 203, iso_name =
↪'199B',
                                                    abundance = 0.1694, mass =198.96825,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 5,
                                                    global_isotope_id = 204, iso_name = '200
↪',
                                                    abundance = 0.2314, mass =199.96825,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 6,
                                                    global_isotope_id = 205, iso_name =
↪'201a',
                                                    abundance = 0.1317, mass =200.97028,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 7,
                                                    global_isotope_id = 206, iso_name =
↪'201b',
                                                    abundance = 0.1317, mass =200.97028,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 8,
                                                    global_isotope_id = 207, iso_name =
↪'201c',
                                                    abundance = 0.1317, mass =200.97028,␣
↪mol_name = 'Hg')
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 9,
                                                    global_isotope_id = 208, iso_name = '202
↪',
                                                    abundance = 0.2974, mass =201.97062,␣
↪mol_name = 'Hg')
```

(continues on next page)

(continued from previous page)

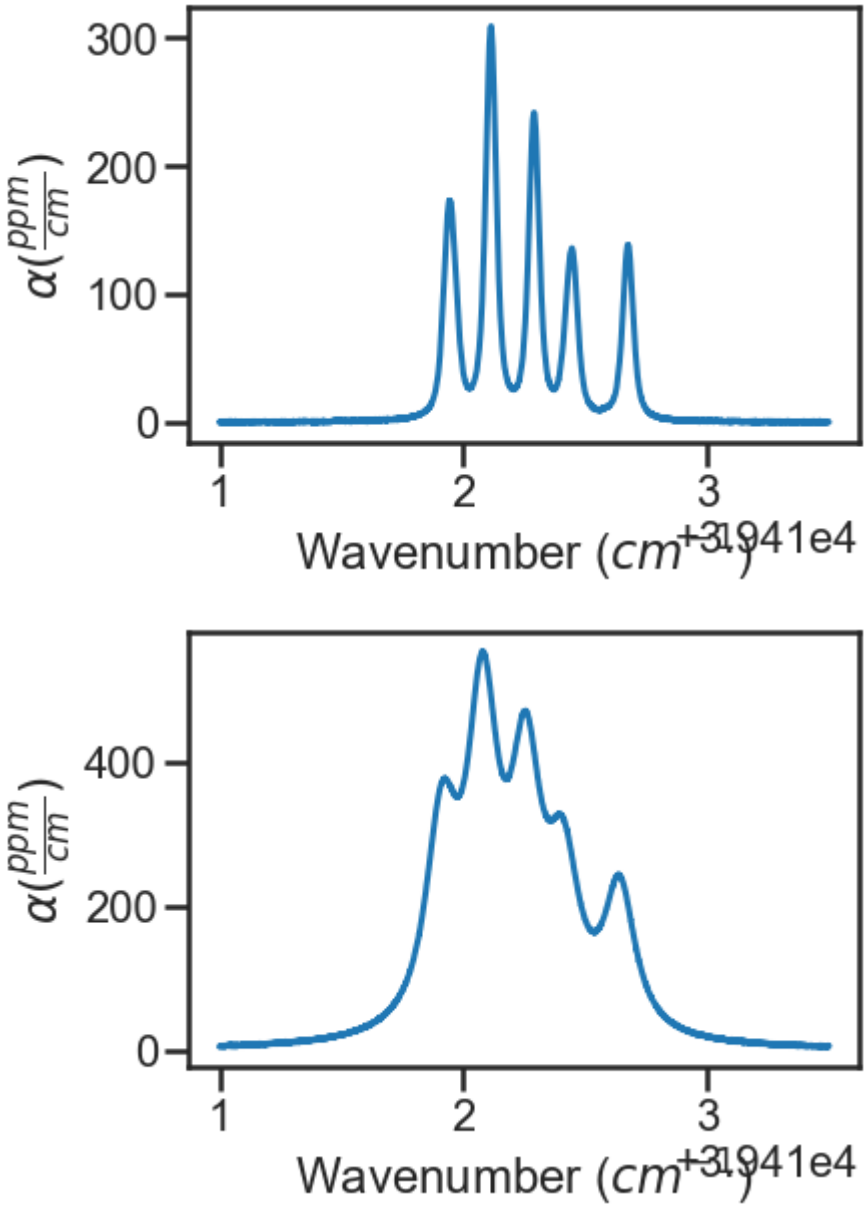
```
HITRAN_Hg_isolist = add_to_HITRANstyle_isotope_list(input_isotope_list = HITRAN_Hg_
↪isolist, molec_id = 100, local_iso_id = 10,
                                                    global_isotope_id = 209, iso_name = '204
↪',
                                                    abundance = 0.0682, mass =203.97347,␣
↪mol_name = 'Hg')
```

Generate Spectra

The simulate spectrum function is used to generate spectra (similar to in the *Fitting Synthetic Spectra* example), but the isotope_list variable is set to use the isotope list that was just generated opposed to using the default HITRAN isotope list defined in HAPI.

The initial parameter line list should have molecule and local isotope ids that correspond to the new isotope list. The mole fraction definition should also use the appropriate mole fraction id. The same would be true for the abundance_ratio_MI term if the sample was not at natural abundance.

```
PARAM_LINELIST = pd.read_csv('Hg_Linelist.csv')

#Generic Fit Parameters
wave_range = 1.5 #range outside of experimental x-range to simulate
IntensityThreshold = 1e-30 #intensities must be above this value to be simulated
Fit_Intensity = 1e-17 #intensities must be above this value for the line to be fit
order_baseline_fit = 0
tau_column = 'Absoprtion Coefficient (ppm/cm)' # Mean tau/us
freq_column = 'Frequency (cm-1)' # Total Frequency /MHz
pressure_column = 'Pressure (Torr)'
temperature_column = 'Temperature'

wave_min = 39411
wave_max = 39413.5
wave_step = 0.001
SNR = 1000


spec_1 = MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                     temperature = 25, pressure = 100, molefraction = { 100 :1e-9},
                     isotope_list = HITRAN_Hg_isolist, natural_abundance = True, SNR =␣
↪1000, filename = '100 Torr',)

spec_2 = MATS.simulate_spectrum(PARAM_LINELIST, wave_min, wave_max, wave_step,
                     temperature = 25, pressure = 500, molefraction = { 100 :1e-9},
                     isotope_list = HITRAN_Hg_isolist, natural_abundance = True, SNR =␣
↪1000,filename = '500 Torr',)

spec_1.plot_wave_alpha()
spec_2.plot_wave_alpha()
```



Construct Dataset and Generate Fit Parameters

The construction of the dataset and the generate fit parameter classes act as normal. For the generation of the fit parameter line list, the example below shows how the vary_variable input dictionaries function with multiple isotopes. Specifically, in the example below the VP is used as the fitting line shape, where the spectra were simulated with the SDVP. All parameters are constrained across the spectra and the only parameters being floated are the collisional broadening terms for all isotopes.

```
SPECTRA = MATS.Dataset([spec_1, spec_2], 'Mercury Fitting',PARAM_LINELIST )

#Generate Baseline Parameter list based on number of etalons in spectra definitions and␣
↪baseline order
BASE_LINELIST = SPECTRA.generate_baseline_paramlist()
```

(continues on next page)

(continued from previous page)

```
FITPARAMS = Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST, lineprofile =
↪'VP', linemixing = False,
                                   fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
                                   nu_constrain = True, sw_constrain = True, gamma0_
↪constrain = True, delta0_constrain = True,
                                    aw_constrain = True, as_constrain = True,
                                    nuVC_constrain = True, eta_constrain =True, linemixing_
↪constrain = True,
                                   additional_columns = ['trans_id'])

FITPARAMS.generate_fit_param_linelist_from_linelist(vary_nu = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_sw ={100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_gamma0 = {100:{1:True, 2:True, 3:
↪True, 4: True, 5:True, 6:True, 7:True, 8:True, 9:True, 10: True}},
                                                    vary_delta0 = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_aw = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_as = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_nuVC = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_eta = {100:{1:False, 2:False, 3:
↪False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}},
                                                    vary_linemixing = {100:{1:False, 2:
↪False, 3:False, 4: False, 5:False, 6:False, 7:False, 8:False, 9:False, 10: False}})

FITPARAMS.generate_fit_baseline_linelist(vary_baseline = False, vary_molefraction = {100:
↪ False}, vary_xshift = False,
                                         vary_etalon_amp= False, vary_etalon_period= False,␣
↪vary_etalon_phase= False)
```

Fit Spectra

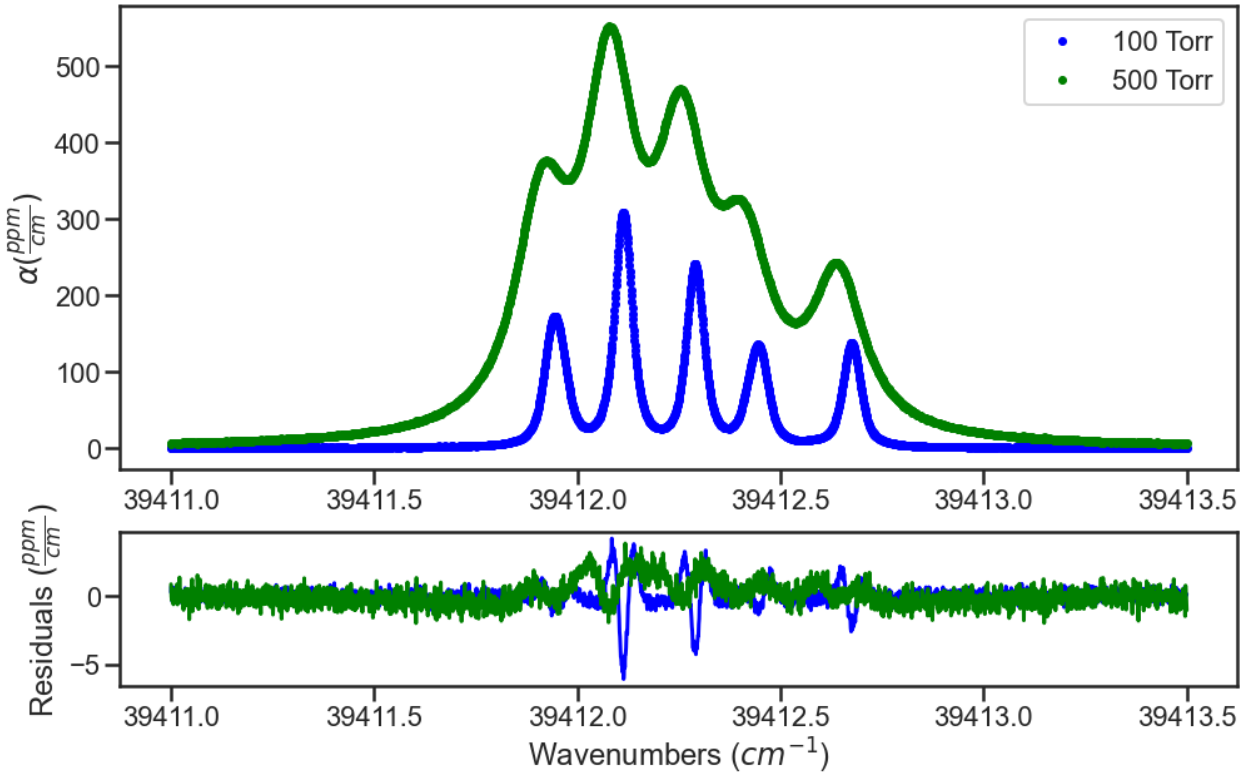
Fitting follows the same strucutre described in *Fitting Experimental Spectra*. The fit results shown below are the result of fitting spectra simulated with the SDVP with the VP and only allowing the collsional broadening terms to float. The residuals at both pressures show the expected w-shaped residuals.

```
fit_data = MATS.Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity, weight_spectra = False)
params = fit_data.generate_params()
result = fit_data.fit_data(params, wing_cutoff = 25)
fit_data.residual_analysis(result, indv_resid_plot=False)
```

(continues on next page)

(continued from previous page)

```
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



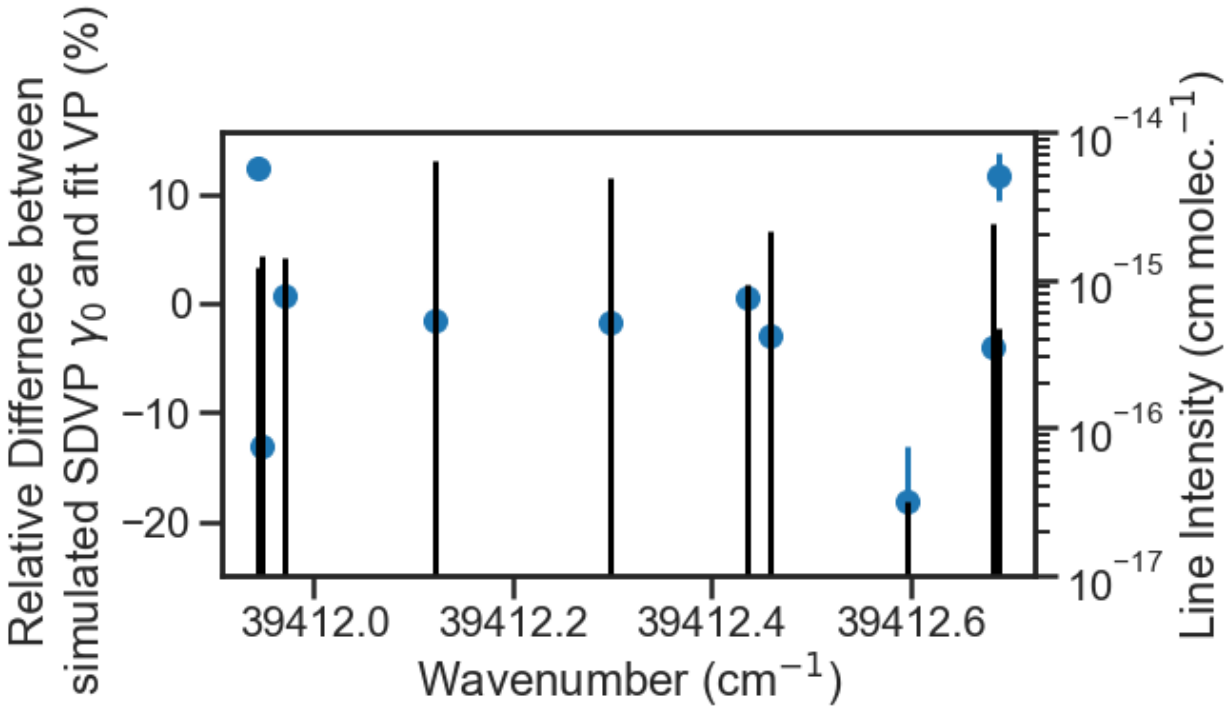
We can compare the VP fit collisional broadening fit results to the SDVP collisional broadening values used in the simulation. On the secondary y-axis is a stem plot showing the line intensity for the lines. This highlights that the lines that deviated the most from the simulated collisional broadening value were weaker and/or closely spaced to other isotope transitions.

```
Parameter_Linelist= pd.read_csv('Parameter_Linelist.csv', index_col = 0)
simulated_gamma0_air = PARAM_LINELIST['gamma0_air'].values[0]

fig, ax1 = plt.subplots(constrained_layout=True, figsize= [9, 4.5])
ax2 = ax1.twinx()
ax2.stem(Parameter_Linelist['nu'].values,Parameter_Linelist['sw'].values*Parameter_
↪Linelist['sw_scale_factor'].values, "k", markerfmt = 'None', basefmt = 'None')
ax1.errorbar(Parameter_Linelist['nu'].values, 100*(Parameter_Linelist['gamma0_air'].
↪values - simulated_gamma0_air)/ simulated_gamma0_air,
          yerr = 100*(Parameter_Linelist['gamma0_air_err'].values)/ Parameter_Linelist[
↪'gamma0_air'].values,fmt = "o")


ax2.set_ylim(1e-17, 1e-14)
ax1.ticklabel_format(axis = 'x', useOffset = False)
ax2.set_yscale('log')
ax1.set_xlabel('Wavenumber (cm$^{-1}$)')
ax2.set_ylabel('Line Intensity (cm molec.$^{-1}$)')
```

(continues on next page)

(continued from previous page)

```
ax1.set_ylabel('Relative Differnece between \n simulated SDVP $\\gamma_{0}$ and fit VP (
↪%)')
```



1.5.2 Publications that used MATS

- Adkins EM, Long DA, Fleisher AJ, Hodges JT. Near-infrared cavity ring-down spectroscopy measurements of nitrous oxide in the (4200)←(0000) and (5000)←(0000) bands. Journal of Quantitative Spectroscopy and Radiative Transfer. 2021;262.
- Adkins EM, Long DA, Hodges JT. Air-broadening in near-infrared carbon dioxide line shapes: Quantifying contributions from O2, N2, and Ar. Journal of Quantitative Spectroscopy and Radiative Transfer. 2021;270.
- Fleurbaey H, Reed ZD, Adkins EM, Long DA, Hodges JT. High accuracy spectroscopic parameters of the 1.27 μm band of O2 measured with comb-referenced, cavity ring-down spectroscopy. Journal of Quantitative Spectroscopy and Radiative Transfer. 2021;270.
- Hashemi R, Gordon IE, Adkins EM, Hodges JT, Long DA, Birk M, et al. Improvement of the spectroscopic parameters of the air- and self-broadened N2O and CO lines for the HITRAN2020 database applications. Journal of Quantitative Spectroscopy and Radiative Transfer. 2021:107735.

1.6 MATS module

class MATS.dataset.**Dataset**(*spectra*, *dataset_name*, *param_linelist*, *CIA_model=None*)
Bases: object

Combines spectrum objects into a Dataset object to enable multi-spectrum fitting.

Parameters

- **spectra** (*list*) – list of spectrum objects to be included in the Dataset. Example [spectrum_1, spectrum_2, ...]
- **dataset_name** (*str*) – Used to provide a name for the Dataset to use when saving files
- **param_linelist** (*pandas dataframe*) – Reads in the parameter linelist used in fitting. This enables for consistency checks between the input spectra and the parameter line list.
- **baseline_order** (*int*) – sets the baseline order for all spectra in the dataset. This will automatically be set to the maximum baseline order across all spectrum included in the Dataset.
- **CIA_model** (*str*) – Future development will allow CIA model specification. Default is None and

average_QF()
Calculates the Average QF from all spectra.

Returns **average_QF** – average QF of all spectra in dataset

Return type float

check_iso_list()
Checks to make sure that all molecules are in the isotope_list and also checks to make sure all spectra use the same isotope list

correct_component_list()
Corrects so that all spectra and the parameter line list share the same molecules, but the mole fraction is fixed to zero where molecules are not present (called in the initialization of the class).

correct_etalon_list()
Corrects so that all spectrum share the same number of etalons, but the amplitude and period are fixed to zero where appropriate(called in the initialization of the class).

generate_CIA_paramlist()
Future development will generates a csv file called dataset_name + _CIA_paramlist, which will be used to generate another csv file that is used for fitting the broadband CIA that is common across all spectra, where the columns will be dependent on the CIA model used.

Returns **CIA_paramlist** – currently a place holder for future feature.

Return type pandas dataframe

generate_baseline_paramlist()
Generates a csv file called dataset_name + _baseline_paramlist, which will be used to generate another csv file that is used for fitting spectrum dependent parameters with columns for spectrum number, segment number, x_shift, concentration for each molecule in the dataset, baseline terms (a = 0th order term, b = 1st order, etc), and etalon terms (set an amplitude, period, and phase for the number of etalons listed for each spectrum in the Dataset).

Returns **baseline_paramlist** – dataframe containing information describing baseline parameters. Also saves to .csv. Either file can be edited before making the baseline parameter list used for fitting. If editting the .csv file will need to regenerate dataframe from .csv.

Return type dataframe

generate_summary_file(*save_file=False*)
Generates a summary file combining spectral information from all spectra in the Dataset.

Parameters **save_file** (*bool, optional*) – If True, then a .csv is generated in addition to the dataframe. The default is False.

Returns **summary_file** – Summary dataframe comprised of spectral information inculding model and residuals for all spectra in Dataset.

Return type dataframe

get_ILS_function_dict()
Provides a dictionary of all ILS functions used in the dataset and the number of resolution parameters

Returns **dataset_ILS_list** – list of strings matching the name of the ILS functions used in the dataset

Return type list

get_baseline_order()

get_broadener_list()
Provides a list of all broadeners in the dataset.

Returns **dataset_broadener_list** – list of all broadeners in the dataset.

Return type list

get_dataset_name()

get_etalons()
Get list of number of etalons for spectra.

Returns **dataset_etalon_list** – etalon keys across spectra

Return type list

get_list_spectrum_numbers()
Generates a list of all spectrum_numbers.

Returns **spec_num_list** – list of all spectrum numbers used in the dataset.

Return type list

get_molecules()
Get list of molecules in spectra.

Returns **dataset_molecule_list** – list of molecules in spectra

Return type list

get_number_nominal_temperatures()
Get the number of nominal temperatures in the .

Returns

- **num_nominal_temperatures** (*int*) – number of nominal temperatures in the Dataset
- **nominal_temperatures** (*list*) – list of all nominal temperatures listed in the input spectra

get_number_spectra()

get_spectra()

get_spectra_extremes()

get_spectrum_extremes()
Gets the minimum and maximum wavenumber for the entire Dataset.

Returns

- **wave_min** (*float*) – The minimum wavenumber in all spectra in the Dataset.
- **wave_max** (*float*) – The maximum wavenumber in all spectra in the Dataset.

`get_spectrum_filename`(*spectrum_num*)
Gets spectrum filename for spectrum in Dataset. :param spectrum_num: Integer assigned to a given spectrum. :type spectrum_num: int

Returns filename – if spectrum_num in Dataset then the filename for that the spectrum_number is returned

Return type str

`get_spectrum_pressure`(*spectrum_num*)
Gets spectrum pressure for spectrum in Dataset.

Parameters `spectrum_num` (*int*) – Integer assigned to a given spectrum.

Returns pressure – if spectrum_num in Dataset then the pressure (torr) for that the spectrum_number is returned.

Return type float

`get_spectrum_temperature`(*spectrum_num*)
Gets spectrum temperature for spectrum in Dataset.

Parameters `spectrum_num` (*int*) – Integer assigned to a given spectrum.

Returns temperature – if spectrum_num in Dataset then the temperature (K) for that the spectrum_number is returned.

Return type float

`max_baseline_order`()
sets the baseline order to be equal to the maximum in any of the included spectra.

`plot_model_residuals`()
Generates a plot showing both the model and experimental data as a function of wavenumber in the main plot with a subplot showing the residuals as function of wavenumber.

`renumber_spectra`()
renumbers the spectra to be sequential starting at 1 (called in the initialization of the class).

`set_baseline_order`(*new_baseline_order*)

`set_dataset_name`(*new_dataset_name*)

`set_spectra`(*new_spectra*)

class MATS.fit_dataset.**Fit_DataSet**(*dataset*, *base_linelist_file*, *param_linelist_file*, *CIA_linelist_file=None*, *minimum_parameter_fit_intensity=1e-27*, *weight_spectra=False*, *baseline_limit=False*, *baseline_limit_factor=10*, *pressure_limit=False*, *pressure_limit_factor=10*, *temperature_limit=False*, *temperature_limit_factor=10*, *molefraction_limit=False*, *molefraction_limit_factor=10*, *etalon_limit=False*, *etalon_limit_factor=50*, *x_shift_limit=False*, *x_shift_limit_magnitude=0.1*, *nu_limit=False*, *nu_limit_magnitude=0.1*, *sw_limit=False*, *sw_limit_factor=10*, *gamma0_limit=False*, *gamma0_limit_factor=10*, *n_gamma0_limit=True*, *n_gamma0_limit_factor=10*, *delta0_limit=False*, *delta0_limit_factor=10*, *n_delta0_limit=True*, *n_delta0_limit_factor=10*, *SD_gamma_limit=False*, *SD_gamma_limit_factor=10*, *n_gamma2_limit=True*, *n_gamma2_limit_factor=10*, *SD_delta_limit=True*, *SD_delta_limit_factor=10*, *n_delta2_limit=True*, *n_delta2_limit_factor=10*, *nuVC_limit=False*, *nuVC_limit_factor=10*, *n_nuVC_limit=True*, *n_nuVC_limit_factor=10*, *eta_limit=True*, *eta_limit_factor=10*, *linemixing_limit=False*, *linemixing_limit_factor=10*, *beta_formalism=False*)

Bases: `object`

Provides the fitting functionality for a Dataset.

Parameters

- **dataset** (*object*) – Dataset Object.
- **base_linelist_file** (*str*) – filename for file containing baseline parameters.
- **param_linelist_file** (*str*) – filename for file containing parmeter parameters.
- **CIA_linelist_file** (*str, optional*) – Future Feature: filename for file constraining CIA parameters
- **minimum_parameter_fit_intensity** (*float, optional*) – minimum intensity for parameters to be generated for fitting. NOTE: Even if a value is floated in the param_linelist if it is below this threshold then it won't be a floated.. The default is 1e-27.
- **weight_spectra** (*boolean*) – If True, then the pt by pt percent uncertainty for each spectrum and the spectrum weighting will be used in the calculation of the residuals. Default is False.
- **baseline_limit** (*bool, optional*) – If True, then impose min/max limits on baseline parameter solutions. The default is False.
- **baseline_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor that the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **pressure_limit** (*bool, optional*) – If True, then impose min/max limits on pressure solutions. The default is False.
- **pressure_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.

- **temperature_limit** (*bool, optional*) – If True, then impose min/max limits on temperature solutions. The default is False.
- **temperature_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **molefraction_limit** (*bool, optional*) – If True, then impose min/max limits on mole fraction solutions. The default is False.
- **molefraction_limit_factor** (*float, optional*) – DESCRIPTION. The default is 10.
- **etalon_limit** (*bool, optional*) – If True, then impose min/max limits on etalon solutions. The default is False.
- **etalon_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 50. #phase is constrained to +/- 2pi
- **x_shift_limit** (*bool, optional*) – If True, then impose min/max limits on x-shift solutions. The default is False.
- **x_shift_limit_magnitude** (*float, optional*) – The magnitude variables set the +/- range of the variable in cm-1. The default is 0.1.
- **nu_limit** (*bool, optional*) – If True, then impose min/max limits on line center solutions. The default is False.
- **nu_limit_magnitude** (*float, optional*) – The magnitude variables set the +/- range of the variable in cm-1. The default is 0.1.
- **sw_limit** (*bool, optional*) – If True, then impose min/max limits on line intensity solutions. The default is False.
- **sw_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **gamma0_limit** (*bool, optional*) – If True, then impose min/max limits on collisional half-width solutions. The default is False.
- **gamma0_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **n_gamma0_limit** (*bool, optional*) – DESCIf True, then impose min/max limits on temperature exponent for half width solutions. The default is True.
- **n_gamma0_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.

- **delta0_limit** (*bool, optional*) – If True, then impose min/max limits on collisional shift solutions. The default is False.
- **delta0_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **n_delta0_limit** (*bool, optional*) – If True, then impose min/max limits on temperature exponent of the collisional shift solutions. The default is True.
- **n_delta0_limit_factor** (*float, optional*) – DESCRIPTION. The default is 10.
- **SD_gamma_limit** (*bool, optional*) – If True, then impose min/max limits on the aw solutions. The default is False.
- **SD_gamma_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **n_gamma2_limit** (*bool, optional*) – If True, then impose min/max limits on temperature exponent of the speed-dependent width solutions. The default is True.
- **n_gamma2_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **SD_delta_limit** (*bool, optional*) – If True, then impose min/max limits on as solutions. The default is True.
- **SD_delta_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **n_delta2_limit** (*bool, optional*) – If True, then impose min/max limits on temperature exponent of the speed-dependent shift solutions. The default is True.
- **n_delta2_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **nuVC_limit** (*bool, optional*) – If True, then impose min/max limits on dicke narrowing solutions. The default is False.
- **nuVC_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **n_nuVC_limit** (*bool, optional*) – If True, then impose min/max limits on temperature exponent of dicke narrowing solutions. The default is True.

- **n_nuVC_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is 10.
- **eta_limit** (*bool, optional*) – If True, then impose min/max limits on correlation parameter solutions.. The default is True.
- **eta_limit_factor** (*float, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.The default is 10.
- **linemixing_limit** (*bool, optional*) – The factor variable describes the multiplicative factor the value can vary by min = 1/factor * init_guess, max = factor* init_guess. NOTE: If the init_guess for a parameter is equal to zero, then the limits aren't imposed because 1) then it would constrain the fit to 0 and 2) LMfit won't let you set min = max.. The default is False.
- **linemixing_limit_factor** (*float, optional*) – If True, then impose min/max limits on line-mixing solutions.. The default is 10.
- **beta_formalism** (*boolean, optional*) – If True, then the beta correction on the Dicke narrowing is used in the simulation model.

constrained_baseline(*params*, *baseline_segment_constrained=True*, *xshift_segment_constrained=True*, *molefraction_segment_constrained=True*, *etalon_amp_segment_constrained=True*, *etalon_period_segment_constrained=True*, *etalon_phase_segment_constrained=True*, *pressure_segment_constrained=True*, *temperature_segment_constrained=True*)

Imposes baseline constraints when using multiple segments per spectrum, ie all baseline parameters can be the same across the entire spectrum except for the etalon phase, which is allowed to vary per segment.

Parameters

- **params** (*lmfit parameter object*) – the params object is a dictionary comprised of all parameters translated from dataframes into a dictionary format compatible with lmfit.
- **baseline_segment_constrained** (*bool, optional*) – True means the baseline terms are constrained across each spectrum. The default is True.
- **xshift_segment_constrained** (*bool, optional*) – True means the x_shift terms are constrained across each spectrum. The default is True.
- **molefraction_segment_constrained** (*bool, optional*) – True means the mole fraction for that molecule is constrained across each spectrum. The default is True.
- **etalon_amp_segment_constrained** (*bool, optional*) – True means the etalon amplitude is constrained across each spectrum. The default is True.
- **etalon_period_segment_constrained** (*bool, optional*) – True means the etalon period is constrained across each spectrum. The default is True.
- **etalon_phase_segment_constrained** (*bool, optional*) – True means the etalon phase is constrained across each spectrum. The default is True.
- **pressure_segment_constrained** (*bool, optional*) – True means the pressure is constrained across each spectrum. The default is True.
- **temperature_segment_constrained** (*bool, optional*) – True means the temperature is constrained across each spectrum. The default is True.

Returns **params** – the params object is a dictionary comprised of all parameters translated from dataframes into a dictionary format compatible with lmfit.

Return type lmfit parameter object

fit_data(*params*, *wing_cutoff=25*, *wing_wavenumbers=25*, *wing_method='wing_cutoff'*, *xtol=1e-07*, *maxfev=2000*, *ftol=1e-07*)

Uses the lmfit minimizer to do the fitting through the simulation model function.

Parameters

- **params** (*lmfit parameter object*) – the params object is a dictionary comprised of all parameters translated from dataframes into a dictionary format compatible with lmfit.
- **wing_cutoff** (*float, optional*) – number of voigt half-widths to simulate on either side of each line. The default is 50.
- **wing_wavenumbers** (*float, optional*) – number of wavenumbers to simulate on either side of each line. The default is 50.
- **wing_method** (*str, optional*) – Provides choice between the wing_cutoff and wing_wavenumbers line cut-off options. The default is 'wing_cutoff'.
- **xtol** (*float, optional*) – Absolute error in xopt between iterations that is acceptable for convergence. The default is 1e-7.
- **maxfev** (*float, optional*) – DESCRIPTION. The default is 2000.
- **ftol** (*The maximum number of calls to the function., optional*) – Absolute error in func(xopt) between iterations that is acceptable for convergence.. The default is 1e-7.

Returns **result** – contains all fit results as LMFit results object.

Return type LMFit result Object

generate_beta_output_file(*beta_summary_filename=None*)

Generates a file that summarizes the beta values used in the fitting in the case that beta was used to correct the Dicke narrowing term (beta_formalism = True).

Parameters **beta_summary_filename** (*str, optional*) – Filename to save the beta information. The default is Beta Summary File.

Returns

Return type The generated file has the beta values for each line and spectra that were used in the fitting. Access to this information is critical as the relationship between beta and nuVC is what generates a spoecific nuVC.

generate_params()

Generates the lmfit parameter object that will be used in fitting.

Returns **params** – the parameter object is a dictionary comprised of all parameters translated from dataframes into a dictionary format compatible with lmfit

Return type lmfit parameter object

residual_analysis(*result*, *indv_resid_plot=False*)

Updates the model and residual arrays in each spectrum object with the results of the fit and gives the option of generating the combined absorption and residual plot for each spectrum.

Parameters

- **result** (*LMFit result Object*) – contains all fit results as LMFit results object.

- **indv_resid_plot** (*bool, optional*) – True if you want to show residual plots for each spectra.. The default is False.

simulation_model(*params*, *wing_cutoff=25*, *wing_wavenumbers=25*, *wing_method='wing_cutoff'*)
This is the model used for fitting that includes baseline, resonant absorption, and CIA models.

Parameters

- **params** (*lmfit parameter object*) – the params object is a dictionary comprised of all parameters translated from dataframes into a dictionary format compatible with lmfit.
- **wing_cutoff** (*float, optional*) – number of voigt half-widths to simulate on either side of each line. The default is 25.
- **wing_wavenumbers** (*float, optional*) – number of wavenumbers to simulate on either side of each line. The default is 25
- **wing_method** (*TYPE, optional*) – Provides choice between the wing_cutoff and wing_wavenumbers line cut-off options. The default is 'wing_cutoff'.

Returns **total_residuals** – residuals for all spectra in Dataset.

Return type array

update_params(*result*, *base_linelist_update_file=None*, *param_linelist_update_file=None*)
Updates the baseline and line parameter files based on fit results with the option to write over the file (default) or save as a new file and updates baseline values in the spectrum objects.

Parameters

- **result** (*LMFit result Object*) – contains all fit results as LMFit results object.
- **base_linelist_update_file** (*str, optional*) – Name of file to save the updated baseline parameters. Default is to override the input. The default is None.
- **param_linelist_update_file** (*str, optional*) – Name of file to save the updated line parameters. Default is to override the input. The default is None.

MATS.fit_dataset.**HTP_from_DF_select**(*linelist*, *waves*, *wing_cutoff=25*, *wing_wavenumbers=25*, *wing_method='wing_cutoff'*, *p=1*, *T=296*, *molefraction={}*, *isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D', 0.00015537, 18.008915, 'OH'], (14, 1): [51, 'H(19F)', 0.99984425, 20.006229, 'HF'], (14, 2): [110, 'D(19F)', 0.000115, 21.0125049978, 'HF'], (15, 1): [52, 'H(35Cl)', 0.757587, 35.976678, 'HCl'], (15, 2): [53, 'H(37Cl)', 0.242257, 37.973729, 'HCl'], (15, 3): [107, 'D(35Cl)', 0.000118005, 36.9829544578, 'HCl'], (15, 4): [108, 'D(37Cl)', 3.7735e-05, 38.9800043678, 'HCl'], (16, 1): [54, 'H(79Br)', 0.50678, 79.92616, 'HBr'], (16, 2): [55, 'H(81Br)', 0.49306, 81.924115, 'HBr'], (16, 3): [111, 'D(79Br)', 5.82935e-05, 80.9324388778, 'HBr'], (16, 4): [112, 'D(81Br)', 5.67065e-05, 82.9303923778, 'HBr'], (17, 1): [56, 'H(127I)', 0.99984425, 127.912297, 'HI'], (17, 2): [113,*

eters.

Outline

1. Uses provided wavenumber axis
2. Calculates the molecular density from pressure and temperature
3. Sets up Diluent dictionary if not given as input
4. Calculates line intensity and doppler width at temperature for all lines
5. Loops through each line in the line list and loops through each diluent, generating a line parameter at experimental conditions that is the appropriate ratio of each diluent species corrected for pressure and temperature. For each line, simulate the line for the given simulation cutoffs and add to cross section
6. Return wavenumber and cross section arrays

Parameters

- **linelist** (*dataframe*) –

  Pandas dataframe with the following column headers, where species corresponds to each diluent in the spect

  nu = wavenumber of the spectral line transition (cm-1) in vacuum

  sw = The spectral line intensity (cm1/(moleculecm2)) at Tref=296K

  elower = The lower-state energy of the transition (cm-1)

  molec_id = HITRAN molecular ID number

  local_iso_id = HITRAN isotopologue ID number

  gamma_0_species = half width at half maximum (HWHM) (cm1/atm) at Tref=296K and reference pressure pref=1atm for a given diluent (air, self, CO2, etc.)

  n_gamma0_species = The coefficient of the temperature dependence of the half width

  delta_0_species = The pressure shift (cm1/atm) at Tref=296K and pref=1atm of the line position with respect to the vacuum transition wavenumber ij

  n_delta0_species = the coefficient of the temperature dependence of the pressure shift

  SD_gamma_species = the ratio of the speed dependent width to the half-width at reference temperature and pressure

  n_gamma2_species = the coefficient of the temperature dependence of the speed dependent width NOTE: This is the temperature dependence of the speed dependent width not the ratio of the speed dependence to the half-width

  SD_delta_species = the ratio of the speed dependent shift to the collisional shift at reference temperature and pressure

  n_delta2_species = the coefficient of the temperature dependence of the speed dependent shift NOTE: This is the temperature dependence of the speed dependent shift not the ratio of the speed dependence to the shift

  nuVC_species = dicke narrowing term at reference temperature

  n_nuVC_species = coefficient of the temperature dependence of the dicke narrowing term

  eta_species = the correlation parameter for the VC and SD effects

  y_species_nominaltemperature = linemixing term (as currently written this doesn't have a temperature dependence, so read in a different column for each nominal temperature)

- **waves** (*array*) – 1-D array comprised of wavenumbers (cm-1) to use as x-axis for simulation.
- **wing_cutoff** (*float, optional*) – number of half-widths for each line to be calculated for. The default is 50.
- **wing_wavenumbers** (*float, optional*) – number of wavenumber for each line to be calculated. The default is 50.
- **wing_method** (*str, optional*) – defines which wing cut-off option to use. Options are wing_cutoff or wing_wavenumbers The default is 'wing_cutoff'.
- **p** (*float, optional*) – pressure for simulation in atmospheres. The default is 1.
- **T** (*float, optional*) – temperature for simulation in kelvin. The default is 296.
- **molefraction** (*dict, optional*) – takes the form {molecule_id : mole fraction, molecule_id: mole fraction, ...}. The default is {}.
- **isotope_list** (*dict, optional*) – provides opportunity to specify the isotope look-up table. Default is ISO, which is from HAPI. If not using ISO, then must use this format and suggested you use function to add to ISO
- **natural_abundance** (*bool, optional*) – True indicates the sample is at natural abundance. The default is True.
- **abundance_ratio_MI** (*dictionary, optional*) – If sample is not at natural abundance, then the natural abundance defines the enrichment factor compared to natural abundance(ie a sample where the main isotope is the only isotope would have a 1/natural abundance as the enrichment factor). Defined as {M:{I:enrichment factor, I: enrichment factor, I: enrichment factor}, ... }. The default is {}.
- **Diluent** (*dict, optional*) – contains the species as the key with the value being the abundance of that diluent in the sample, ie {'He':0.5, 'self':0.5}. The default is {}.
- **diluent** (*str, optional*) – If Diluent = {}, then this value will be used to set the only diluent to be equal to this diluent. The default is 'air'.
- **IntensityThreshold** (*float, optional*) – minimum line intensity that will be simulated. The default is 1e-30.

Returns

- **wavenumbers** (*array*) – wavenumber axis that should match the input waves
- **xsect** (*array*) – simulated cross section as a function of wavenumbers (ppm/cm)

MATS.fit_dataset.**HTP_wBeta_from_DF_select**(*linelist*, *waves*, *wing_cutoff=25*, *wing_wavenumbers=25*, *wing_method='wing_cutoff'*, *p=1*, *T=296*, *molefraction={}*, *isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D', 0.00015537, 18.008915, 'OH'], (14, 1): [51, 'H(19F)', 0.99984425, 20.006229, 'HF'], (14, 2): [110, 'D(19F)', 0.000115, 21.0125049978, 'HF'], (15, 1): [52, 'H(35Cl)', 0.757587, 35.976678, 'HCl'], (15, 2): [53, 'H(37Cl)',*

eters with capability of incorporating the beta correction to the Dicke Narrowing proposed in Analytical-function correction to the Hartmann–Tran profile for more reliable representation of the Dicke-narrowed molecular spectra.

Outline

1. Uses provided wavenumber axis
2. Calculates the molecular density from pressure and temperature
3. Sets up Diluent dictionary if not given as input
4. Calculates line intensity and doppler width at temperature for all lines
5. Loops through each line in the line list and loops through each diluent, generating a line parameter at experimental conditions that is the appropriate ratio of each diluent species corrected for pressure and temperature. For each line, simulate the line for the given simulation cutoffs and add to cross section
6. Return wavenumber and cross section arrays

Parameters

- **linelist** (*dataframe*) –

  Pandas dataframe with the following column headers, where species corresponds to each diluent in the spect

  nu = wavenumber of the spectral line transition (cm-1) in vacuum

  sw = The spectral line intensity (cm1/(moleculecm2)) at Tref=296K

  elower = The lower-state energy of the transition (cm-1)

  molec_id = HITRAN molecular ID number

  local_iso_id = HITRAN isotopologue ID number

  gamma_0_species = half width at half maximum (HWHM) (cm1/atm) at Tref=296K and reference pressure pref=1atm for a given diluent (air, self, CO2, etc.)

  n_gamma0_species = The coefficient of the temperature dependence of the half width

  delta_0_species = The pressure shift (cm1/atm) at Tref=296K and pref=1atm of the line position with respect to the vacuum transition wavenumber ij

  n_delta0_species = the coefficient of the temperature dependence of the pressure shift

  SD_gamma_species = the ratio of the speed dependent width to the half-width at reference temperature and pressure

  n_gamma2_species = the coefficient of the temperature dependence of the speed dependent width NOTE: This is the temperature dependence of the speed dependent width not the ratio of the speed dependence to the half-width

  SD_delta_species = the ratio of the speed dependent shift to the collisional shift at reference temperature and pressure

  n_delta2_species = the coefficient of the temperature dependence of the speed dependent shift NOTE: This is the temperature dependence of the speed dependent shift not the ratio of the speed dependence to the shift

  nuVC_species = dicke narrowing term at reference temperature

  n_nuVC_species = coefficient of the temperature dependence of the dicke narrowing term

  eta_species = the correlation parameter for the VC and SD effects

y_species_nominaltemperature = linemixing term (as currently written this doesn't have a temperature dependence, so read in a different column for each nominal temperature)

- **waves** (*array*) – 1-D array comprised of wavenumbers (cm-1) to use as x-axis for simulation.
- **wing_cutoff** (*float, optional*) – number of half-widths for each line to be calculated for. The default is 50.
- **wing_wavenumbers** (*float, optional*) – number of wavenumber for each line to be calculated. The default is 50.
- **wing_method** (*str, optional*) – defines which wing cut-off option to use. Options are wing_cutoff or wing_wavenumbers The default is 'wing_cutoff'.
- **p** (*float, optional*) – pressure for simulation in atmospheres. The default is 1.
- **T** (*float, optional*) – temperature for simulation in kelvin. The default is 296.
- **molefraction** (*dict, optional*) – takes the form {molecule_id : mole fraction, molecule_id: mole fraction, ...}. The default is {}.
- **isotope_list** (*dict, optional*) – provides opportunity to specify the isotope look-up table. Default is ISO, which is from HAPI. If not using ISO, then must use this format and suggested you use function to add to ISO
- **natural_abundance** (*bool, optional*) – True indicates the sample is at natural abundance. The default is True.
- **abundance_ratio_MI** (*dictionary, optional*) – If sample is not at natural abundance, then the natural abundance defines the enrichment factor compared to natural abundance(ie a sample where the main isotope is the only isotope would have a 1/natural abundance as the enrichment factor). Defined as {M:{I:enrichment factor, I: enrichment factor, I: enrichment factor}, ... }. The default is {}.
- **Diluent** (*dict, optional*) – contains the species as the key with the value being the abundance of that diluent in the sample, ie {'He':0.5, 'self':0.5}. The default is {}.
- **diluent** (*str, optional*) – If Diluent = {}, then this value will be used to set the only diluent to be equal to this diluent. The default is 'air'.
- **IntensityThreshold** (*float, optional*) – minimum line intensity that will be simulated. The default is 1e-30.

Returns

- **wavenumbers** (*array*) – wavenumber axis that should match the input waves
- **xsect** (*array*) – simulated cross section as a function of wavenumbers (ppm/cm)

class MATS.generate_fitparam_file.**Generate_FitParam_File**(*dataset*, *param_linelist*, *base_linelist*, *CIA_linelist=None*, *lineprofile='VP'*, *linemixing=False*, *threshold_intensity=1e-30*, *fit_intensity=1e-26*, *fit_window=1.5*, *sim_window=5*, *param_linelist_savename='Parameter_LineList'*, *base_linelist_savename='Baseline_LineList'*, *CIA_linelist_savename='CIA_LineList'*, *nu_constrain=True*, *sw_constrain=True*, *gamma0_constrain=True*, *delta0_constrain=True*, *aw_constrain=True*, *as_constrain=True*, *nuVC_constrain=True*, *eta_constrain=True*, *linemixing_constrain=True*, *additional_columns=[]*)

Bases: object

Class generates the parameter files used in fitting and sets up fit settings.

Parameters

- **dataset** (*object*) – Dataset object to be used in the fits
- **param_linelist** (*dataframe*) – parameter linelist dataframe name, where the dataframe has the appropriate column headers
- **base_linelist** (*dataframe*) – baseline parameter dataframe name generated from the dataset.generate_baseline_paramlist() function.
- **CIA_linelist** (*dataframe, optional*) – Future Function: CIA linelist dataframe name generated from the dataset.generate_CIA_paramlist() function.
- **lineprofile** (*str*) – lineprofile to use for the simulation. This sets values in the parameter linelist to 0 and forces them to not vary unless manually changed. Default values is VP, voigt profile. Options are VP, SDVP, NGP, SDNGP, HTP.
- **linemixing** (*bool*) – If False, then all linemixing parameters are set to 0 and not able to float. Default is False.
- **threshold_intensity** (*float*) – This is the minimum line intensity that will be simulated. Default value is 1e-30.
- **fit_intensity** (*float*) – This is the minimum line intensity for which parameters will be set to float. This can be overwritten manually. Default value is 1e-26.
- **fit_window** (*float*) – currently not used
- **sim_window** (*float*) – This is the region outside of the wavelength region of the dataset that will be simulated for analysis. Default value is 5 cm-1.
- **base_linelist_savename** (*str*) – filename that the baseline linelist will be saved as. Default is Baseline_LineList
- **param_linelist_savename** (*str*) – filename that the parameter linelist will be saved as. Default is Parameter_LineList.
- **CIA_linelist_save_name** (*str*) – Future Feature: filename that the CIA linelist will be saved as. Default is CIA_LineList.

- **nu_constrain** (*bool*) – True indicates that the line centers will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **sw_constrain** (*bool*) – True indicates that the line intensities will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **gamma0_constrain** (*bool*) – True indicates that the collisional width will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **delta0_constrain** (*bool*) – True indicates that the shift will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **aw_constrain** (*bool*) – True indicates that the speed dependent width will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **as_constrain** (*bool*) – True indicates that the speed dependent shift will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **nuVC_constrain** (*bool*) – True indicates that the dicke narrowing term will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **eta_constrain** (*bool*) – True indicates that the correlation parameter will be a global variable across all spectra. False generates a new value for each spectrum in the dataset.
- **linemixing_constrain** (*bool*) – True indicates that the first-order linemixing term will be a global variable across all spectra. False generates a new value for each spectrum in the datas

generate_fit_baseline_linelist(*vary_baseline=True*, *vary_pressure=False*, *vary_temperature=False*, *vary_molefraction={1: False, 7: True}*, *vary_xshift=False*, *vary_etalon_amp=False*, *vary_etalon_period=False*, *vary_etalon_phase=False*, *vary_ILS_res=False*)

Generates the baseline line list used in fitting and updates the fitting booleans to desired settings.

Parameters

- **vary_baseline** (*bool, optional*) – If True, then sets all baseline parameters for all spectra to float. The default is True.
- **vary_pressure** (*bool, optional*) – If True, then the pressures for all spectra are floated. This should be used with caution as the impact these parameters have on other floated parameters might lead to an unstable solution. The default is False.
- **vary_temperature** (*bool, optional*) – If True, then the temperatures for all spectra are floated. This should be used with caution as the impact these parameters have on other floated parameters might lead to an unstable solution.The default is False.
- **vary_molefraction** (*dict, optional*) – keys to dictionary correspond to molecule id where the value is boolean flag, which dictates whether to float the mole fraction. The default is {}. Example: {7: True, 1: False}
- **vary_xshift** (*bool, optional*) – If True, then sets x-shift parameters for all spectra to float. The default is False.
- **vary_etalon_amp** (*bool, optional*) – If True, then sets etalon amplitude parameters for all spectra to float. The default is False.
- **vary_etalon_period** (*bool, optional*) – If True, then sets etalon period parameters for all spectra to float. . The default is False.
- **vary_etalon_phase** (*bool, optional*) – If True, then sets etalon phase parameters for all spectra to float.. The default is False.

- **vary_ILS_res** (*bool, optional*) – If True, then sets ILS resolution parameters for all spectra to float.. The default is False.

Returns **base_linelist_df** – returns dataframe based on baseline line list with addition of a vary and err column for every floatable parameter. The vary columns are defined by the inputs. The err columns will be populated from fit results. The dataframe is also saved as a .csv file..

Return type dataframe

generate_fit_param_linelist_from_linelist(*vary_nu={1: {1: False}, 7: {1: True, 2: False, 3: False}}*, *vary_sw={7: {1: True, 2: False, 3: False}}*, *vary_gamma0={1: {1: False}, 7: {1: True, 2: False, 3: False}}*, *vary_n_gamma0={}*, *vary_delta0={1: {1: False}, 7: {1: True, 2: False, 3: False}}*, *vary_n_delta0={}*, *vary_aw={1: {1: False}, 7: {1: True, 2: False, 3: False}}*, *vary_n_gamma2={}*, *vary_as={}*, *vary_n_delta2={}*, *vary_nuVC={}*, *vary_n_nuVC={}*, *vary_eta={}*, *vary_linemixing={}*)

Generates the parameter line list used in fitting and updates the fitting booleans to desired settings.

Parameters

- **vary_nu** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope line centers should be floated. Each key in the dictionary corresponds to a molecule, where the values are a dictionary corresponding to local_iso_ids as keys and boolean values acting as boolean flags. True means float the nu for that molecule and isotope. Example vary_nu = {7:{1:True, 2: False, 3: False}, 1:{1:False, 2: False, 3: False}}, would indicate that line centers of all main oxygen isotopes would be floated (if the line intensity is greater than the fit intensity), where the line centers for all other O2 isotopes and all water isotopes would not be varied. If these value are left blank, then all variable would have to be manually switched to float. The default is {}.
- **vary_sw** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope line intensities should be floated. Follows nu_vary example. The default is {}.
- **vary_gamma0** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope collisional half-width should be floated. Follows nu_vary example. . The default is {}.
- **vary_n_gamma0** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope coefficient of the temperature dependence of the half width should be floated. Follows nu_vary example. . The default is {}.
- **vary_delta0** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope pressure shift should be floated. Follows nu_vary example. . The default is {}.
- **vary_n_delta0** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope coefficient of the temperature dependence of the pressure shift should be floated. Follows nu_vary example. . The default is {}.
- **vary_aw** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope ratio of the speed dependent width to the half-width should be floated. Follows nu_vary example. . The default is {}.
- **vary_n_gamma2** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope coefficient of the temperature dependence of the speed dependent width should be floated. Follows nu_vary example. . The default is {}.

- **vary_as** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope the ratio of the speed dependent shift to the shift should be floated. Follows nu_vary example. . The default is {}.
- **vary_n_delta2** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope coefficient of the temperature dependence of the speed dependent shift should be floated. Follows nu_vary example. . The default is {}.
- **vary_nuVC** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope dicke narrowing should be floated. Follows nu_vary example. . The default is {}.
- **vary_n_nuVC** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope coefficient of the temperature dependence of the dicke narrowing should be floated. Follows nu_vary example. . The default is {}.
- **vary_eta** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope correlation parameter for the VC and SD effects should be floated. Follows nu_vary example. . The default is {}.
- **vary_linemixing** (*bool, optional*) – Dictionary of dictionaries setting whether the molecule and isotope first-order line-mixing should be floated. Follows nu_vary example. . The default is {}.

Returns **param_linelist_df** – returns dataframe based on parameter line list with addition of a vary and err column for every floatable parameter. The vary columns are defined by the inputs and the fit_intensity value. The err columns will be populated from fit results. The dataframe is also saved as a .csv file. line intensity will be normalized by the fit_intensity (set to the sw_scale_factor). The term 'sw' is now equal to the normalized value, such that in the simulation 'sw'*sw_scale_factor is used for the line intensity. Because line intensities are so small it is difficult to fit the value without normalization.

Return type dataframe

get_CIA_linelist()

get_base_linelist()

get_dataset()

get_param_linelist()

Routines to load example data locally or from web

class MATS.linelistdata.**LoadLineListData**(*paths=None*)

Bases: object

Helper class to read in supplied LineList DataFrames

names

names of LineList files available

Type list

__getitem__ : get file DataFrame by position/name

self[x] returns a copy of the linelist frame corresponding to *self.names[x]* if *x* is an integer, otherwise returns Frame corresponding to *name* = *x*

property names

property paths

class MATS.spectrum.**Spectrum**(*filename*, *molefraction={}*, *natural_abundance=True*, *isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D', 0.00015537, 18.008915, 'OH'], (14, 1): [51, 'H(19F)', 0.99984425, 20.006229, 'HF'], (14, 2): [110, 'D(19F)', 0.000115, 21.0125049978, 'HF'], (15, 1): [52, 'H(35Cl)', 0.757587, 35.976678, 'HCl'], (15, 2): [53, 'H(37Cl)', 0.242257, 37.973729, 'HCl'], (15, 3): [107, 'D(35Cl)', 0.000118005, 36.9829544578, 'HCl'], (15, 4): [108, 'D(37Cl)', 3.7735e-05, 38.9800043678, 'HCl'], (16, 1): [54, 'H(79Br)', 0.50678, 79.92616, 'HBr'], (16, 2): [55, 'H(81Br)', 0.49306, 81.924115, 'HBr'], (16, 3): [111, 'D(79Br)', 5.82935e-05, 80.9324388778, 'HBr'], (16, 4): [112, 'D(81Br)', 5.67065e-05, 82.9303923778, 'HBr'], (17, 1): [56, 'H(127I)', 0.99984425, 127.912297, 'HI'], (17, 2): [113, 'D(127I)', 0.000115, 128.918574778, 'HI'], (18, 1): [57, '(35Cl)(16O)', 0.75591, 50.963768, 'ClO'], (18, 2): [58, '(37Cl)(16O)', 0.24172, 52.960819, 'ClO'], (19, 1): [59, '(16O)(12C)(32S)', 0.93739, 59.966986, 'OCS'], (19, 2): [60, '(16O)(12C)(34S)', 0.04158, 61.96278, 'OCS'], (19, 3): [61, '(16O)(13C)(32S)', 0.01053, 60.970341, 'OCS'], (19, 4): [62, '(16O)(12C)(33S)', 0.01053, 60.966371, 'OCS'], (19, 5): [63, '(18O)(12C)(32S)', 0.00188, 61.971231, 'OCS'], (20, 1): [64, 'H2(12C)(16O)', 0.98624, 30.010565, 'H2CO'], (20, 2): [65, 'H2(13C)(16O)', 0.01108, 31.01392, 'H2CO'], (20, 3): [66, 'H2(12C)(18O)', 0.0019776, 32.014811,*

Spectrum class provides all information describing experimental or simulated spectrum.

Parameters

- **filename** (*str*) – file containing spectrum with .csv extension. File extension is not included in the name
- **molefraction** (*dict*) – mole fraction of each molecule in spectra in the format {molec_id: mole fraction (out of 1), molec_id: molefraction, … }
- **natural_abundance** (*bool, optional*) – flag for if the molecular species in the spectrum are at natural abundance
- **abundance_ratio_MI** (*dict, optional*) – if not at natural abundance sets the enhancement factor for each molecule and isotope in the following format {molec_id:{iso_id: enhancement, iso_id: enhancement}, … }. The enhancement is the ratio of the new abundance to the natural abundance.
- **isotope_list** (*dict, optional*) – provides opportunity to specify the isotope look-up table. Default is ISO, which is from HAPI. If not using ISO, then must use this format and suggested you use function to add to ISO
- **diluent** (*str, optional*) – sets the diluent for the sample. Default = 'air'
- **Diluent** (*dict, optional*) – sets the diluent for the sample if there are a combination of several diluents. Format {'he': 0.5, 'air': 0.5). NOTE: the line parameter file must have parameters that correspond to the diluent (ie gamma0_he, and gamma0_air). Additionally, the contribution from all diluents must sum to 1.
- **spectrum_number** (*int, optional*) – sets a number for the spectrum that will correspond to fit parameters. This is set in the Dataset class, so should not need to be defined manually
- **input_freq** (*bool, optional*) – True indicates that the frequency_column is in MHz. False indicates that the frequency column is in wavenumbers.
- **input_tau** (*bool, optional*) – True indicates that the tau_column is in us. False indicates that the tau column is in 1/c*tau (ppm/cm).
- **pressure_column** (*str, optional*) – Name of the pressure column in input file. Column should be in torr. Default is Cavity Pressure /Torr.
- **temperature_column** (*str, optional*) – Name of the temperature column in input file. Column should be in celsius. Default is Cavity Temperature Side 2 /C.
- **frequency_column** (*str, optional*) – Name of the frequency column in input file. Column should be in MHz or Wavenumbers (with appropriate flag for input_freq). NOTE: This is not a detuning axis. Default is Total Frequency /MHz.
- **tau_column** (*str, optional*) – Name of the tau column in input file. Column should be in us or in 1/ctau (ppm/cm) (with appropriate flag for input_tau). Default is Mean tau/us.
- **tau_stats_column** (*str, optional*) – Name of column containing the pt by pt error as a percent of the y-axis. Default is 'tau rel. std. dev./%
- **segment_column** (*str, optional*) – Name of column with segment numbers in input file. This column allows spectrum background parameters to be treated in segments across the spectrum. If None then the entire spectrum will share the same segment.
- **etalons** (*dict, optional*) – Allows you to define etalons by an amplitude and period (1/ frequency in cm-1). Default is no etalons. Input is dictionary with keys being a number and the value being an array with the first index being the amplitude and the second being the period.

- **nominal_temperature** (*int, optional*) – nominal temperature indicates the approximate temperature of the spectrum. When combining spectra into a dataset this acts as a flag to whether temperature dependence terms should be parameters that can be fit or whether they should act as constants. Default = 296
- **x_shift** (*float, optional*) – value in wavenumbers of the x shift for the spectrum axis. This is a fittable parameter. Be careful in using this parameter as floating multiple parameters with similar effects cause fits to not converge (ie. unconstrained line centers + x_shift or fits with line center, shifts, and x_shifts terms without enough lines per spectrum to isolate the different effects). Floating this term works best if you have a good initial guess. Default = 0
- **baseline_order** (*int, optional*) – sets the baseline order for this spectrum. Allows you to change the baseline order across the broader dataset.()
- **weight** (*float, optional,*) – set the weighting to used for the spectrum if using weighted fits. This can be used to weight a whole spectrum in addition to the pt by pt weighting using the stats column.
- **ILS_function** (*string, optional*) – Default is None and means that no instrument line shape is used in the fitting. Function can be: SLIT_MICHELSON, SLIT_DIFFRACTION, SLIT_COSINUS, SLIT_DISPERSION, SLIT_GAUSSIAN, SLIT_TRIANGULAR, SLIT_RECTANGULAR corresponding to the ILS functions defined in HAPI or a user defined function.
- **ILS_resolution** (*float/array, optional*) – Resolution is a float or array of ILS resolutions in wavenumbers. The SlitFunctions defined in HAPI have 1 resolution, but this opens the option for the user defined function to be comprised of several functions with varying resolutions. Default is 0.1 cm-1.
- **ILS_wing** (*float/array, optional*) – AF_wing is the a float or array consisting of the range the ILS is calculted over in cm-1. This is a single value fort he HAPI slit functions, but could be an array of multiple values for user-defined functions. Default is 10 cm-1

calculate_QF()

Calculates the quality of fit factor (QF) for a spectrum - QF = (maximum alpha - minimum alpha) / std(residuals).

Returns QF.

Return type float

diluent_sum_check()

Checks that if multiple broadeners are used that the contributions sum to one.

Returns Warning if the diluents don't sum to one

Return type str

fft_spectrum()

Takes the FFT of the residuals of the spectrum, generates a plot of frequency (cm-1) versus amplitude (ppm/cm), and prints a dataframe with the 20 highest amplitude frequencies with the FFT frequency (period), amplitude, FFT phase, and frequency (cm-1).

get_Diluent()

get_abundance_ratio_MI()

get_alpha()

get_background()

get_cia()

get_diluent()

get_etalons()

get_filename()

get_frequency()

get_model()

get_molefraction()

get_natural_abundance()

get_nominal_temperature()

get_pressure()

get_pressure_torr()

get_residuals()

get_spectrum_number()

get_tau()

get_tau_stats()

get_temperature()

get_temperature_C()

get_wavenumber()

plot_freq_tau()
Generates a plot of tau (us) as a function of frequency (MHz).

plot_model_residuals()
Generates a plot of the alpha and model (ppm/cm) as a function of wavenumber (cm-1) and on lower plot shows the residuals (ppm/cm) as a function of wavenumber (cm-1).

plot_wave_alpha()
Generates a plot of alpha (ppm/cm) as a function of wavenumber (cm-1).

save_spectrum_info(*save_file=False*)
Saves spectrum information to a pandas dataframe with option to also save as as a csv file.

Parameters **save_file** (*bool, optional*) – If False, then only a dataframe is created. If True, then a csv file will be generated with the name filename + '_saved.csv'. The default is False.

Returns **new_file** – returns a pandas dataframe with columns related to the spectrum information

Return type dataframe

segment_wave_alpha()
Defines the wavenumber, alpha, and indices of spectrum that correspond to a given spectrum segment.

Returns

- **wavenumber_segments** (*dict*) – dictionary where the key corresponds to a segment number and the values correspond to the wavenumbers for that segment
- **alpha_segments** (*dict*) – dictionary where the key corresponds to a segment number and the values correspond to the alpha values for that segment.
- **indices_segments** (*dict*) – dictionary where the key corresponds to a segment number and the values correspond to the array indices for that segment.

set_Diluent(*new_Diluent*)

set_abundance_ration_MI(*new_abundance_ratio_MI*)

set_background(*new_background*)

set_cia(*new_cia*)

set_diluent(*new_diluent*)

set_etalons(*new_etalons*)

set_frequency_column(*new_frequency_column*)

set_model(*new_model*)

set_molefraction(*new_molefraction*)

set_natural_abundance(*new_natural_abundance*)

set_nominal_temperature(*new_nominal_temperature*)

set_pressure_column(*new_pressure_column*)

set_residuals(*new_residuals*)

set_spectrum_number(*new_spectrum_number*)

set_tau_column(*new_tau_column*)

set_tau_stats_column(*new_tau_stats_column*)

set_temperature_column(*new_temperature_column*)

set_weight(*new_weight*)

MATS.spectrum.**simulate_spectrum**(*parameter_linelist*, *wave_min=None*, *wave_max=None*, *wave_space=None*, *wavenumbers=[]*, *wave_error=0*, *SNR=None*, *baseline_terms=[0]*, *temperature=25*, *temperature_err={'bias': 0, 'function': None, 'params': {}}*, *pressure=760*, *pressure_err={'function': None, 'params': {}, 'per_bias': 0}*, *wing_cutoff=25*, *wing_wavenumbers=25*, *wing_method='wing_cutoff'*, *filename='temp'*, *molefraction={}*, *molefraction_err={}*, *isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D', 0.00015537, 18.008915, 'OH'], (14, 1): [51, 'H(19F)', 0.99984425, 20.006229, 'HF'], (14, 2): [110, 'D(19F)', 0.000115, 21.0125049978, 'HF'], (15, 1): [52, 'H(35Cl)', 0.757587, 35.976678, 'HCl'], (15, 2): [53, 'H(37Cl)', 0.242257, 37.973729, 'HCl'], (15, 3): [107, 'D(35Cl)', 0.000118005, 36.9829544578, 'HCl'], (15, 4): [108, 'D(37Cl)', 3.7735e-05, 38.9800043678, 'HCl'], (16, 1): [54, 'H(79Br)', 0.50678, 79.92616, 'HBr'], (16, 2): [55, 'H(81Br)', 0.49306, 81.924115, 'HBr'], (16, 3): [111, 'D(79Br)', 5.82935e-05, 80.9324388778, 'HBr'], (16, 4): [112, 'D(81Br)', 5.67065e-05, 82.9303923778, 'HBr'], (17, 1): [56, 'H(127I)', 0.99984425, 127.912297, 'HI'], (17, 2): [113, 'D(127I)',*

Parameters

- **parameter_linelist** (*dataframe*) – linelist following the convention of the linelists used for the HTP_from_DF_select. Note that there will need to be a linemixing column for each nominal temperature, which you will have to do manually (ie y_air_296, y_self_296).
- **wavenumbers** (*array of floats, optional*) – array of wavenumbers for the simulation (cm-1). If provided, then this axis will be used. If wavenumbers = None, then the wave_min, wave_max, and wave_space will be used to calculate wavenumber grid.
- **wave_min** (*float, optional*) – minimum wavenumber for the simulation (cm-1)
- **wave_max** (*float, optional*) – maximum wavenumber for the simulation (cm-1).
- **wave_space** (*float, optional*) – wavenumber spacing for the simulation (cm-1).
- **wave_error** (*float, optional*) – absolute error on the wavenumber axis (cm-1) to include in simulations. The default is 0.
- **SNR** (*float, optional*) – Signal to noise ratio to impose on the simulated spectrum. The default is None. If SNR is none there is no noise on the simulation.
- **baseline_terms** (*list, optional*) – polynomial baseline coefficients where the index is equal to the coefficient order, ie. [0, 1, 2] would correspond to baseline = 0 + 1*(wavenumber - minimum wavenumber) + 2*(wavenumber - minimum wavenumber)^2. The default is [0].
- **temperature** (*float, optional*) – temperature for simulation in celsius. The default is 25.
- **temperature_err** (*dict, optional*) – possible keys include 'bias', 'function', and 'params'. The bias indicates the absolute bias in Celsius of the temperature reading, which will be added to the input temperature. Function can be 'linear' with params 'm' and 'b' or 'sine' with parameters 'amp', 'phase', and 'phase'. These define a function that is added to both the bias and set temperature as a function of the wavenumber. Note: if 'function' key is not equal to None, then there also needs to be a params key to define the function.. The default is {'bias': 0, 'function': None, 'params': {}}.
- **pressure** (*float, optional*) – pressure for simulation in torr. The default is 760.
- **pressure_err** (*dict, optional*) – possible keys include bias, function, and params. The bias indicates the percent bias in of the pressure reading, which will be added to the input pressure. Function can be 'linear' with params 'm' and 'b' or 'sine' with parameters 'amp', 'phase', and 'phase'. These define a function that is added to both the bias and set pressure as a function of the wavenumber. Note: if 'function' key is not equal to None, then there also needs to be a params key to define the function.. The default is {'per_bias': 0, 'function': None, 'params': {}}.
- **wing_cutoff** (*float, optional*) – number of voigt half-widths to simulate on either side of each line. The default is 25.
- **wing_wavenumbers** (*float, optional*) – number of wavenumbers to simulate on either side of each line. The default is 25.
- **wing_method** (*str, optional*) – Provides choice between the wing_cutoff and wing_wavenumbers line cut-off options. The default is 'wing_cutoff'.
- **filename** (*str, optional*) – allows you to pick the output filename for the simulated spectra. The default is 'temp'.
- **molefraction** (*dict, optional*) – mole fraction of each molecule to be simulated in the spectrum in the format {molec_id: mole fraction (out of 1), molec_id: molefraction, ... }. The default is {}.

- **molefraction_err** (*dict, optional*) – percent error in the mole fraction of each molecule to be simulated in the spectrum in the format {molec_id: percent error in mole fraction, molec_id: percent error in mole fraction, … }. The default is {}.
- **natural_abundance** (*bool, optional*) – flag for if the spectrum contains data at natural abundance. The default is True.
- **abundance_ratio_MI** (*dict, optional*) – if not at natural abundance sets the enhancement factor for each molecule and isotope in the following format {molec_id:{iso_id: enhancement, iso_id: enhancement}, … }. The default is {}. The enhancement is the ratio of the new abundance to the natural abundance.
- **isotope_list** (*dict, optional*) – provides opportunity to specify the isotope look-up table. Default is ISO, which is from HAPI. If not using ISO, then must use this format and suggested you use function to add to ISO
- **diluent** (*str, optional*) – sets the diluent for the sample if only using one broadener. The default is 'air'.
- **Diluent** (*dict, optional*) – sets the diluent for the sample if there are a combination of several. Format {'he': 0.5, 'air': 0.5). NOTE: the line parameter file must have parameters that correspond to the diluent (ie gamma0_he, and gamma0_air). Additionally, the contribution from all diluents must sum to 1.. The default is {}.
- **nominal_temperature** (*int, optional*) – nominal temperature indicates the approximate temperature of the spectrum. When combining spectra into a dataset this acts as a flag to whether temperature dependence terms should be parameters that can be fit or whether they should act as constants.. The default is 296.
- **etalons** (*dict, optional*) – Allows you to define etalons by an amplitude and period (1/frequency in cm-1). Default is no etalons. Input is dictionary with keys being a number and the value being an array with the first index being the amplitude and the second being the period.. The default is {}.
- **x_shift** (*float, optional*) – value in wavenumbers of the x shift for the spectrum axis.. The default is 0.
- **IntensityThreshold** (*float, optional*) – minimum line intensity to use in the simulation. The default is 1e-30.
- **num_segments** (*int, optional*) – Number of segments in the file, which is implemented labeling the segment column into equal sequential se . The default is 10.
- **beta_formalism** (*boolean, optional*) – Indicates whether the beta correction for Dicke Narrowing should be used. The default is False.
- **ILS_function** (*string, optional*) – Default is None and means that no instrument line shape is used in the fitting. Function can be: SLIT_MICHELSON, SLIT_DIFFRACTION, SLIT_COSINUS, SLIT_DISPERSION, SLIT_GAUSSIAN, SLIT_TRIANGULAR, SLIT_RECTANGULAR corresponding to the ILS functions defined in HAPI or a user defined function.
- **ILS_resolution** (*float/array, optional*) – Resolution is a float or array of ILS resolutions in wavenumbers. The SlitFunctions defined in HAPI have 1 resolution, but this opens the option for the user defined function to be comprised of several functions with varying resolutions. Default is 0.1 cm-1.
- **ILS_wing** (*float, optional*) – AF_wing is the a float consisting of the range the ILS is calculted over in cm-1. Default is 10 cm-1

Returns

- **spectrum_file** (*.csv*) – File that contains the simulated wavenumber axis, noisy wavenumber axis, absorbance data, noisy absorbance data, percent noise, pressure (torr), and temperature (C). The filename will correspond to the filename parameter, which has a default value of temp. The pressure and temperature columns will include whatever functional change there is to the pressure or temperature, but not the bias offset. This is coded to match how this error would manifest in experiments.
- **spectrum_object** (*object*) – Outputs a Spectrum class object. This makes it so the you can easily switch between reading in an experimental spectrum and simulated a synthetic spectrum by simply switching out whether the spectrum object is defined through the class definition or through the simulate_spectrum function.

MATS.utilities.**add_to_HITRANstyle_isotope_list**(*input_isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D',*

Parameters

- **input_isotope_list** (*dictionary, optional*) – Isotope list dictionary in HAPI format. The default is ISO.
- **molec_id** (*int, optional*) – molec_id for new isotope entry. The default is 100.
- **local_iso_id** (*int, optional*) – local isotope id number for new isotope entry. The default is 1.
- **global_isotope_id** (*int, optional*) – global isotope id for new istope entry. The default is 200.
- **iso_name** (*str, optional*) – isotoope name for new isotope entry. The default is ‘’.
- **abundance** (*float, optional*) – relative abundance of new isotope entry. The default is 1.
- **mass** (*float, optional*) – Mass of isotope in g. The default is 1.
- **mol_name** (*str, optional*) – name of the molecule for new istope entry. The default is ‘’.

Returns **output_isotope_list** – Isotope list with additionall entry

Return type dictionary

MATS.utilities.**arange_**(*lower*, *upper*, *step*)
originally from HAPI 1.1.0.9.6, but corrects npnt to be int

MATS.utilities.**convolveSpectrumSame**(*Omega*, *CrossSection*, *Resolution=0.1*, *AF_wing=10.0*, *SlitFunction=<function SLIT_RECTANGULAR>*, *Wavenumber=None*)
Convolves cross section with a slit function with given parameters. Originally from HAPI 1.1.0.9.6 with correction to **arange_** to prevent float/int error

MATS.utilities.**etalon**(*x*, *amp*, *period*, *phase*)
Etalon definition

Parameters

- **x** (*array*) – array of floats used to define the x-axis
- **amp** (*float*) – amplitude of the etalon.
- **period** (*float*) – period of the etalon.
- **phase** (*float*) – phase of the etalon.

Returns **etalon** – etalon as a function of input x-axis, amplitude, period, and phase.

Return type array

MATS.utilities.**hasNumbers**(*inputString*)
Determines whether there are numbers in a string

Parameters **inputString** (*str*) – string for analysis

Returns Returns True if the there are numbers in a string

Return type bool

MATS.utilities.**isotope_list_molecules_isotopes**(*isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D',*

and lists of I as values.

Parameters **isotope_list** (*dictionary, optional*) – HITRAN style isotope list. The default is ISO.

Returns **molecule_isotope_dictionary** – Dictionary with the molecules in an isotope list with the available I values.

Return type dictionary

MATS.utilities.**max_iter**(*pars*, *iter*, *resid*, **args*, ***kws*)

MATS.utilities.**molecularMass**(*M*, *I*, *isotope_list={(1, 1): [1, 'H2(16O)', 0.997317, 18.010565, 'H2O'], (1, 2): [2, 'H2(18O)', 0.00199983, 20.014811, 'H2O'], (1, 3): [3, 'H2(17O)', 0.000372, 19.01478, 'H2O'], (1, 4): [4, 'HD(16O)', 0.00031069, 19.01674, 'H2O'], (1, 5): [5, 'HD(18O)', 6.23e-07, 21.020985, 'H2O'], (1, 6): [6, 'HD(17O)', 1.16e-07, 20.020956, 'H2O'], (1, 7): [129, 'D2(16O)', 2.4197e-08, 20.022915, 'H2O'], (2, 1): [7, '(12C)(16O)2', 0.984204, 43.98983, 'CO2'], (2, 2): [8, '(13C)(16O)2', 0.011057, 44.993185, 'CO2'], (2, 3): [9, '(16O)(12C)(18O)', 0.0039471, 45.994076, 'CO2'], (2, 4): [10, '(16O)(12C)(17O)', 0.000734, 44.994045, 'CO2'], (2, 5): [11, '(16O)(13C)(18O)', 4.434e-05, 46.997431, 'CO2'], (2, 6): [12, '(16O)(13C)(17O)', 8.25e-06, 45.9974, 'CO2'], (2, 7): [13, '(12C)(18O)2', 3.9573e-06, 47.998322, 'CO2'], (2, 8): [14, '(17O)(12C)(18O)', 1.47e-06, 46.998291, 'CO2'], (2, 9): [121, '(12C)(17O)2', 1.368e-07, 45.998262, 'CO2'], (2, 10): [15, '(13C)(18O)2', 4.4967e-08, 49.001675, 'CO2'], (2, 11): [120, '(18O)(13C)(17O)', 1.654e-08, 48.00165, 'CO2'], (2, 12): [122, '(13C)(17O)2', 1.5375e-09, 47.001618, 'CO2'], (3, 1): [16, '(16O)3', 0.992901, 47.984745, 'O3'], (3, 2): [17, '(16O)(16O)(18O)', 0.00398194, 49.988991, 'O3'], (3, 3): [18, '(16O)(18O)(16O)', 0.00199097, 49.988991, 'O3'], (3, 4): [19, '(16O)(16O)(17O)', 0.00074, 48.98896, 'O3'], (3, 5): [20, '(16O)(17O)(16O)', 0.00037, 48.98896, 'O3'], (4, 1): [21, '(14N)2(16O)', 0.990333, 44.001062, 'N2O'], (4, 2): [22, '(14N)(15N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 3): [23, '(15N)(14N)(16O)', 0.0036409, 44.998096, 'N2O'], (4, 4): [24, '(14N)2(18O)', 0.00198582, 46.005308, 'N2O'], (4, 5): [25, '(14N)2(17O)', 0.000369, 45.005278, 'N2O'], (5, 1): [26, '(12C)(16O)', 0.98654, 27.994915, 'CO'], (5, 2): [27, '(13C)(16O)', 0.01108, 28.99827, 'CO'], (5, 3): [28, '(12C)(18O)', 0.0019782, 29.999161, 'CO'], (5, 4): [29, '(12C)(17O)', 0.000368, 28.99913, 'CO'], (5, 5): [30, '(13C)(18O)', 2.222e-05, 31.002516, 'CO'], (5, 6): [31, '(13C)(17O)', 4.13e-06, 30.002485, 'CO'], (6, 1): [32, '(12C)H4', 0.98827, 16.0313, 'CH4'], (6, 2): [33, '(13C)H4', 0.0111, 17.034655, 'CH4'], (6, 3): [34, '(12C)H3D', 0.00061575, 17.037475, 'CH4'], (6, 4): [35, '(13C)H3D', 4.9203e-06, 18.04083, 'CH4'], (7, 1): [36, '(16O)2', 0.995262, 31.98983, 'O2'], (7, 2): [37, '(16O)(18O)', 0.00399141, 33.994076, 'O2'], (7, 3): [38, '(16O)(17O)', 0.000742, 32.994045, 'O2'], (8, 1): [39, '(14N)(16O)', 0.993974, 29.997989, 'NO'], (8, 2): [40, '(15N)(16O)', 0.0036543, 30.995023, 'NO'], (8, 3): [41, '(14N)(18O)', 0.00199312, 32.002234, 'NO'], (9, 1): [42, '(32S)(16O)2', 0.94568, 63.961901, 'SO2'], (9, 2): [43, '(34S)(16O)2', 0.04195, 65.957695, 'SO2'], (10, 1): [44, '(14N)(16O)2', 0.991616, 45.992904, 'NO2'], (11, 1): [45, '(14N)H3', 0.9958715, 17.026549, 'NH3'], (11, 2): [46, '(15N)H3', 0.0036613, 18.023583, 'NH3'], (12, 1): [47, 'H(14N)(16O)3', 0.98911, 62.995644, 'HNO3'], (12, 2): [117, 'H(15N)(16O)3', 0.003636, 63.99268, 'HNO3'], (13, 1): [48, '(16O)H', 0.997473, 17.00274, 'OH'], (13, 2): [49, '(18O)H', 0.00200014, 19.006986, 'OH'], (13, 3): [50, '(16O)D', 0.00015537, 18.008915, 'OH'], (14, 1): [51, 'H(19F)', 0.99984425, 20.006229, 'HF'], (14, 2): [110, 'D(19F)', 0.000115, 21.0125049978, 'HF'], (15, 1): [52, 'H(35Cl)', 0.757587, 35.976678, 'HCl'], (15, 2): [53, 'H(37Cl)', 0.242257, 37.973729, 'HCl'], (15, 3): [107, 'D(35Cl)', 0.000118005, 36.9829544578, 'HCl'], (15, 4): [108, 'D(37Cl)', 3.7735e-05, 38.9800043678, 'HCl'], (16, 1): [54, 'H(79Br)', 0.50678, 79.92616, 'HBr'], (16, 2): [55, 'H(81Br)', 0.49306, 81.924115, 'HBr'], (16, 3): [111, 'D(79Br)', 5.82935e-05, 80.9324388778, 'HBr'], (16, 4): [112, 'D(81Br)', 5.67065e-05, 82.9303923778, 'HBr'], (17, 1): [56, 'H(127I)', 0.99984425, 127.912297, 'HI'], (17, 2): [113, 'D(127I)', 0.000115, 128.918574778, 'HI'], (18, 1): [57, '(35Cl)(16O)', 0.75591, 50.963768, 'ClO'], (18, 2): [58, '(37Cl)(16O)', 0.24172, 52.960819, 'ClO'], (19, 1): [59, '(16O)(12C)(32S)', 0.93739, 59.966986, 'OCS'], (19, 2): [60, '(16O)(12C)(34S)', 0.04158, 61.96278, 'OCS'], (19, 3): [61, '(16O)(13C)(32S)', 0.01053, 60.970341, 'OCS'], (19, 4): [62, '(16O)(12C)(33S)', 0.01053, 60.966371, 'OCS'], (19, 5): [63, '(18O)(12C)(32S)', 0.00188, 61.971231, 'OCS'], (20, 1): [64, 'H2(12C)(16O)', 0.98624, 30.010565, 'H2CO'], (20, 2): [65, 'H2(13C)(16O)', 0.01108, 31.01392, 'H2CO'], (20, 3): [66, 'H2(12C)(18O)', 0.0019776, 32.014811, 'H2CO'], (21, 1): [67, 'H(16O)(35Cl)', 0.75579, 51.971593, 'HOCl'], (21, 2):*

ETERS:

M: HITRAN molecule number I: HITRAN isotopologue number

OUTPUT PARAMETERS: MolMass: molecular mass

— DESCRIPTION:

Return molecular mass of HITRAN isotolopogue.

CHAPTER

TWO

LEGAL

This software is subject to the NIST Software License (revised as of August 2018).

CHAPTER

THREE

CONTACT

Erin Adkins

- Email
- NIST Staff Page
- GitHub profile

CHAPTER

FOUR

LINKS

NIST GitHub Organization

NIST Optical Measurements Group

NIST Home Page

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

m

INDEX

H

I

L

M

N

P

R

S

U