MATS

Release 1.0

Erin M. Adkins

Jun 07, 2021

CONTENTS

Erin M. Adkins

National Institute of Standards and Technology

Last Revision Jun 07, 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 code was orginally developed using the Enthought Canopy python package distribution. However, further development will use Anaconda. 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 version used in development (v1.1.0.9.6) 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
- qgrid - provides interactive sorting, filtering, and editing of pandas dataframes in jupyter notebooks. Make sure to follow the install instructions for both installation and jupyter installation or this won't work. This is an optional package that allows for the *MATS.Edit_Fit_Param_Files* to function.

- 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

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

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 *MATS.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. The *MATS.Edit_Fit_Param_Files* class allows for edits to the parameter files in the Jupyter Notebook interface rather than editing in the .csv file.

Finally, the *MATS.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.

1.2.1 Spectrum Class and Objects

MATS.Spectrum(filename[, molefraction, ...])	Spectrum class provides all information describing experimental or simulated spectrum.
MATS.simulate_spectrum(parameter_linelist, ...)	Generates a synthetic spectrum, where the output is a spectrum object that can be used in MATS classes.
MATS.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

MATS.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).
MATS.Spectrum.plot_freq_tau()	Generates a plot of tau (us) as a function of frequency (MHz).
MATS.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).
MATS.Spectrum.plot_wave_alpha()	Generates a plot of alpha (ppm/cm) as a function of wavenumber (cm-1).
MATS.Spectrum.save_spectrum_info([save_file])	Saves spectrum information to a pandas dataframe with option to also save as as a csv file.
MATS.Spectrum.segment_wave_alpha()	Defines the wavenumber, alpha, and indices of spectrum that correspond to a given spectrum segment.

1.2.2 Line-by-line Model and Etalon Models

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.

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 ($c2 = hc/k$ = 1.4388028496642257). The partition functions are calculated using TIPS-2017.

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

Todo: The MATS line intensity calculation currently uses the HAPI EnvironmentDependency_Intensity function, which has specific values for physical constants. The second ratiation constant, c2, is defined as hc/k and hardwired into the code. This value includes the Boltzmann constant, which was recently redefined by the CIPM, for the sake of maintining consistency with the SI the next version of MATS will update to use the CODATA values and move from the use of the HAPI EnvironmentDependency_Intensity function.

CODATA: Boltzmann Constant

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). The doppler width is calculated as:

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

Todo: MATS uses the above values for the Boltzmann constant and the term cMassMol, which is equal to the inverse of Avogadro's constant. This is consistent with the HAPI calculation of the doppler width. However, the Boltzmann and Avogadro's constants were recently redefined by the CIPM, for the sake of maintining consistency with the SI the next version of MATS will update to use the CODATA values. This will involve re-defining the doppler width to include Avogadro's number and not the cMassMol value.

CODATA: Boltzmann Constant

CODATA: Avogadro constant

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 nearest-neighbor 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 *MATS.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.

MATS Functions

MATS.HTP_from_DF_select(linelist, waves[, ...])	Calculates the absorbance (ppm/cm) based on input line list, wavenumbers, and spectrum environmental parameters.
MATS.etalon(x, amp, freq, phase)	Etalon definition

1.2.3 Dataset Class

MATS.Dataset(spectra, dataset_name[, ...])	Combines spectrum objects into a Dataset object to enable multi-spectrum fitting.
MATS.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).
MATS.Dataset.generate_summary_file([save_file])	Generates a summary file combining spectral information from all spectra in the Dataset.
MATS.Dataset.get_spectra_extremes()	Gets the minimum and maximum wavenumber for the entire Dataset.
MATS.Dataset.get_spectrum_extremes()	Gets the minimum and maximum wavenumber for each spectrum in the Dataset.
MATS.Dataset.average_QF()	Calculates the Average QF from all spectra.
MATS.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.

1.2.4 Generate FitParam File Class

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

1.2.5 Edit Fit Param Files Class

MATS.Edit_Fit_Param_Files(... [, ...])	Allows for the baseline and parameter files to be editted in jupyter notebook.
MATS.Edit_Fit_Param_Files.edit_generated_baselist()	Allows editting of baseline linelist in notebook
MATS.Edit_Fit_Param_Files.edit_generated_paramlist()	Allows editting of parameter linelist in notebook
MATS.Edit_Fit_Param_Files.save_editted_baselist(...)	Saves edits of baseline linelist in notebook
MATS.Edit_Fit_Param_Files.save_editted_paramlist(...)	Saves edits of parameter linelist in notebook

1.2.6 Fit Dataset Class

MATS.Fit_DataSet(dataset, ...[, ...])	Provides the fitting functionality for a Dataset.
MATS.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.
MATS.Fit_DataSet.fit_data(params[, ...])	Uses the lmfit minimizer to do the fitting through the simulation model function.
MATS.Fit_DataSet.generate_params()	Generates the lmfit parameter object that will be used in fitting.
MATS.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.
MATS.Fit_DataSet.simulation_model(params[, ...])	This is the model used for fitting that includes baseline and resonant absorption models.
MATS.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.

1.3 Generating Parameter Line lists

1.3.1 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.

1.3.2 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")
```

Add location of hapi.py file to system path

```
sys.path.append(r'C:\Users\Documents\Python Scripts\HAPI')#Add hapi.py folder location␣
↪to system path
from hapi import *
```

Set working file location

```
spec_path = r'C:\Users\Documents\Python Scripts\HAPI' # Location of the Summary Data File
os.chdir(spec_path)
```

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
    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
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

```
   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
  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
```

Set-Up Variables for Line list

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 $\frac{cm^{-1}}{molecule \cdot cm^{-2}}$.

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

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.

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

            """Generates two .csv files containing 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.
            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
                    - 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 MATS formatted HITRAN information as filename_
↪initguess.csv
```

(continues on next page)

(continued from previous page)

```
            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
↪'.
            filename : str, optional
                    acts as a base filename for the .csv files generated. The default␣
↪is tablename.
            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.

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

            """
```

The line mixing term MATS needs has a subscript with the nominal temperatures included in the dataset. These columns can be added by hand to the .csv by copying, pasting, and renaming the generated line mixing column. Alternatively, the following code can be added to do this in the script for each nominal temperature.

```
DF['y_air_296'] = DF['y_air'].copy()
```

1.4 MATS Example using Experimental and Synthetic Data

Provided in the MATS v1.0 release are two examples using MATS in the Oxygen A-Band. The first uses experimental spectra and the second uses synthetic spectra. This overview steps through the common elements of both examples and highlights the differences between using experimental data and simulated spectra generated through the *MATS.simulate_spectrum()* function.

1.4.1 Import Modules and Set-Up

Both examples start with importing modules and setting up file locations

```
import numpy as np
import pandas as pd
import qgrid # only need if using the edit MATS.Edit_Fit_Param_Files class
import os, sys
import matplotlib.pyplot as plt
from matplotlib import gridspec
```

Append the system path to include the location of both the hapi and MATS modules

```
sys.path.append(r'C:\Users\Documents\MATS\MATS')# set location of HAPI.py module

from hapi import *
from MATS import *
```

Change the path to the working directory that contains experimental spectra or to the folder that you want to work in.

```
path = r'C:\Users\Documents\MATS\MATS\Examples\A-Band - Experimental Spectra'
os.chdir(path)
```

1.4.2 Load Spectra from files

There are two options for generating *MATS.Spectrum* objects. The first is from a file by instantiating an instance of the class. The second option is by using the *MATS.simulate_spectrum()* function described in the following section.

Before generating *MATS.Spectrum* objects from your experimental data, it is helpful to set some variables for terms that will be used in all of the *MATS.Spectrum* objects 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 *MATS.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.

```
spec_1 = 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,
```

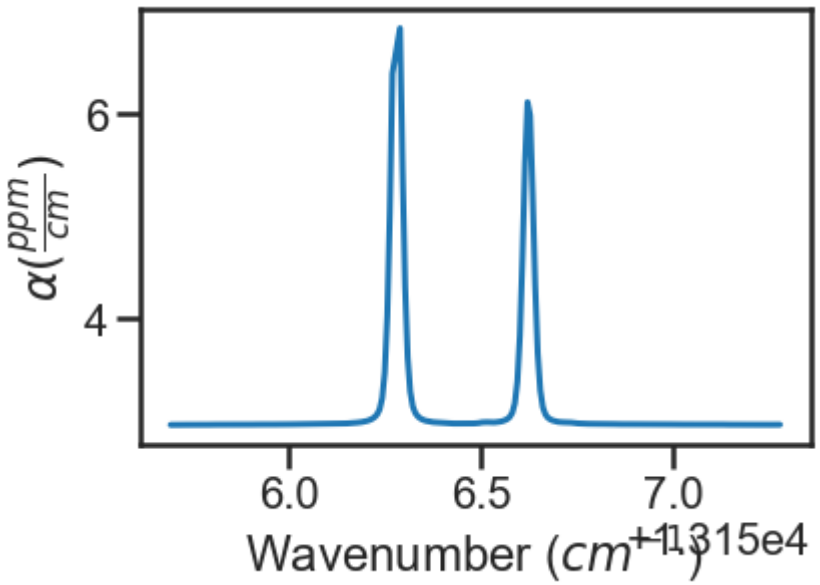
(continues on next page)

(continued from previous page)

```
                    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 = Spectrum('190510_2per_55_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_3 = 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 = 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 *MATS.Spectrum.plot_wave_alpha()* function can be called to plot any of the spectra.

```
spec_1.plot_wave_alpha()
```



1.4.3 Simulate Spectra

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

When simulating spectra, the first step is to read in the reference line list. Generation of this line list is described in the Generating Parameter Line lists page. The following code reads in the reference line list as a pandas dataframe, then resets the working directory.

```
hapi = r'C:\Users\Documents\MATS\MATS\Linelists'
os.chdir(hapi)
PARAM_LINELIST = pd.read_csv('O2_ABand_Drouin_2017_linelist.csv')

path = r'C:\Users\Documents\MATS\MATS\Examples\A-Band - Synthetic Spectra'
os.chdir(path)
```

Just as you would do if reading in the experimental spectrum, this example defines some common simulation and fit variables. In addition to variables defined above, the minimum and maximum wavenumbers for the simulation and the simulation wavenumber spacing are defined. 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
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
baseline_terms = [0] #polynomial baseline coefficients where the index is equal to the␣
↪coefficient order
```

The *MATS.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 = 10000
wave_error = 5e-5
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 *MATS.simulate_spectrum()* function setting the output equal to a variable as was done when the *MATS.Spectrum* was used. This makes it simple to transition code from analysis of experimental spectra to error analysis through simulations.

```
spec_1 =  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 = 20,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '20_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
spec_2 =  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 = 40,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '40_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
spec_3 =  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 = 60,
                    pressure_err = pressure_err,
```

(continues on next page)

(continued from previous page)

```
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '60_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
spec_4 =  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 = 80,
                    pressure_err = pressure_err,
                    wing_cutoff = 50,  wing_method = 'wing_cutoff', filename = '80_torr
↪', molefraction = sample_molefraction, molefraction_err = molefraction_err,
                    natural_abundance = True, nominal_temperature = 296,␣
↪IntensityThreshold = 1e-30, num_segments = 1)
```

1.4.4 Generate a Dataset

The procedure for analysis for both spectrum class generation methods illustrated above is the same from this point on. The next step is to combine all desired *MATS.Spectrum* objects into a *MATS.Dataset* object, where we give the dataset a name. The order of polynomial to use for the baseline fit is also defined.

```
SPECTRA = Dataset([spec_1, spec_2, spec_3, spec_4], 'Line Intensity', baseline_order =␣
↪order_baseline_fit)
```

The *MATS.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()
```

If the parameter line list hasn't been read in from the .csv file, then do that now making sure to switch to the appropriate directories as needed.

```
hapi = r'C:\Users\Documents\MATS\MATS\Linelists'
os.chdir(hapi)
PARAM_LINELIST = pd.read_csv('O2_ABand_Drouin_2017_linelist.csv')
os.chdir(path)
```

1.4.5 Generate Fit Parameter Files

The next section of code uses the *MATS.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.

```
FITPARAMS = Generate_FitParam_File(SPECTRA, PARAM_LINELIST, BASE_LINELIST, lineprofile =
↪'SDVP', linemixing = False,
                                   fit_intensity = Fit_Intensity, threshold_intensity =␣
↪IntensityThreshold, sim_window = wave_range,
```

(continues on next page)

(continued from previous page)

```
                                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)
```

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. 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_freq= False, vary_
↪etalon_phase= True)
```

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

1.4.6 Fit Dataset

Instantiating the *MATS.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.

```
fit_data = Fit_DataSet(SPECTRA,'Baseline_LineList', 'Parameter_LineList', minimum_
↪parameter_fit_intensity = Fit_Intensity,
            baseline_limit = False, baseline_limit_factor = 10,
            molefraction_limit = True, molefraction_limit_factor = 1.1,
            etalon_limit = False, etalon_limit_factor = 2, #phase is constrained to +/-␣
↪2pi,
            x_shift_limit = True, x_shift_limit_magnitude = 0.5,
```

(continues on next page)

(continued from previous page)

```
            nu_limit = True, nu_limit_magnitude = 0.1,
            sw_limit = True, sw_limit_factor = 2,
            gamma0_limit = True, gamma0_limit_factor = 3, n_gamma0_limit= False, n_
↪gamma0_limit_factor = 50,
            delta0_limit = True, delta0_limit_factor = 2, n_delta0_limit = False, n_
↪delta0_limit_factor = 50,
            SD_gamma_limit = True, SD_gamma_limit_factor = 2, n_gamma2_limit = False, n_
↪gamma2_limit_factor = 50,
            SD_delta_limit = True, 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 = True, eta_limit_factor = 50, linemixing_limit = False,␣
↪linemixing_limit_factor = 50)
```

The next step is to generate the lmfit params dictionary object through the *MATS.Fit_DataSet.generate_params()* function. This translates baseline and parameter line list .csv files into the 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 *MATS.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 (suffix number) and spectrum (after **sw_**) 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 aids in the fitting as line intensities are so much smaller than other values.

```
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
```

(continues on next page)

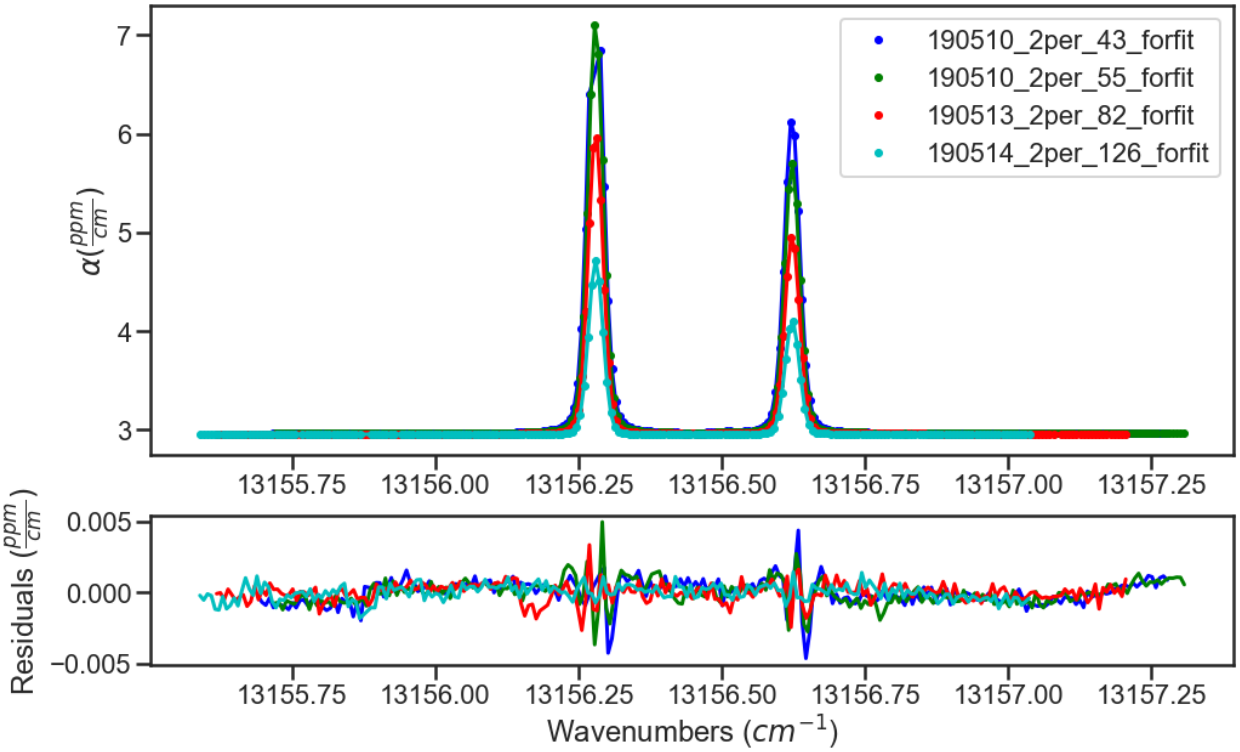
(continued from previous page)

```
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
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
```

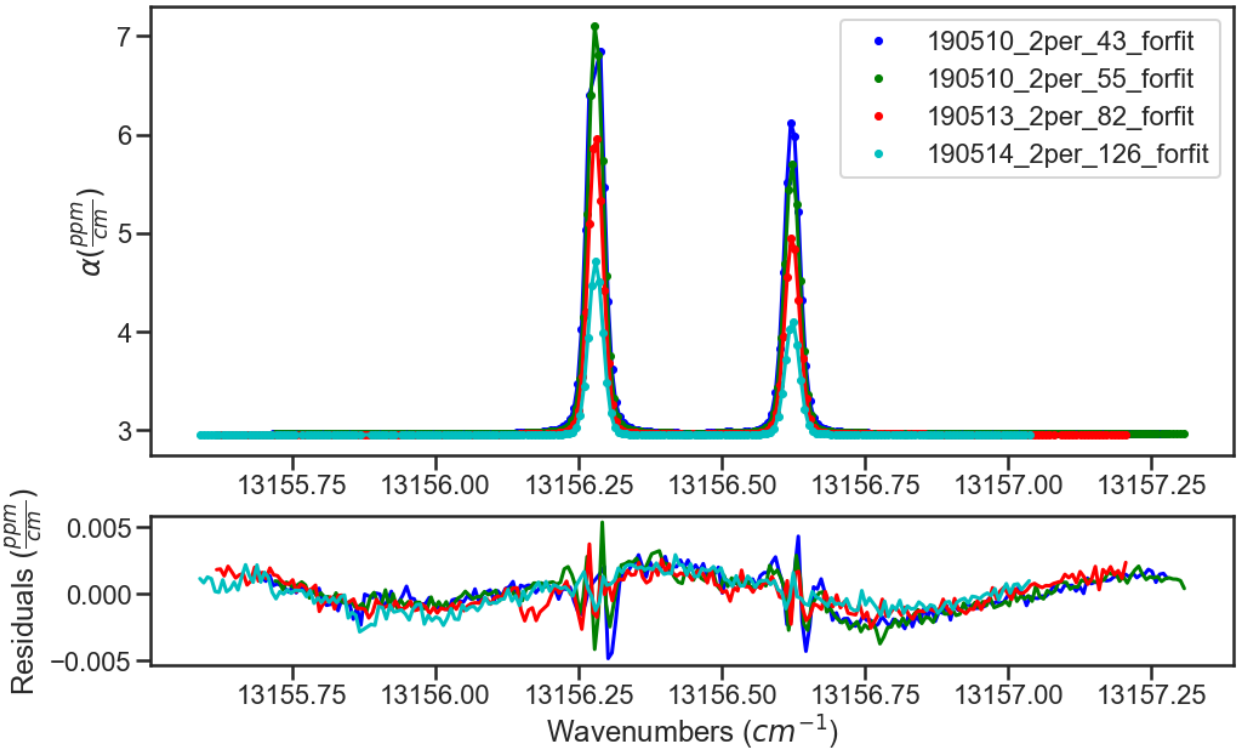
The last segment of code provides residual plots and updates residuals through the `MATS.fit_data.`

`residual_analysis()` and *MATS.Dataset.plot_model_residuals()* functions, updates the parameter and baseline line lists through `MATS.fit_data.update_params()`, and generates a summary file with the fit results using *MATS.Dataset.plot_model_residuals()*.

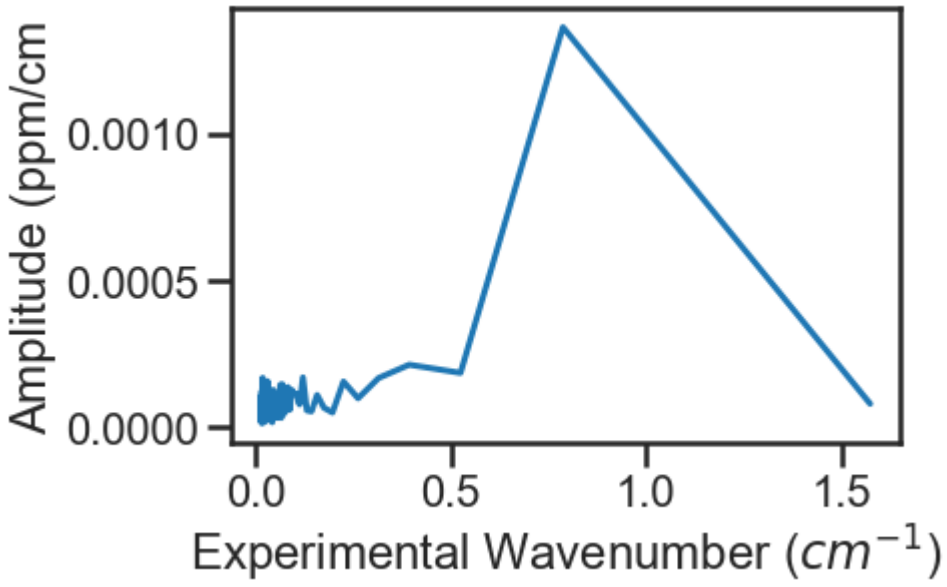
```
fit_data.residual_analysis(result, indv_resid_plot=True)
fit_data.update_params(result)
SPECTRA.generate_summary_file(save_file = True)
SPECTRA.plot_model_residuals()
```



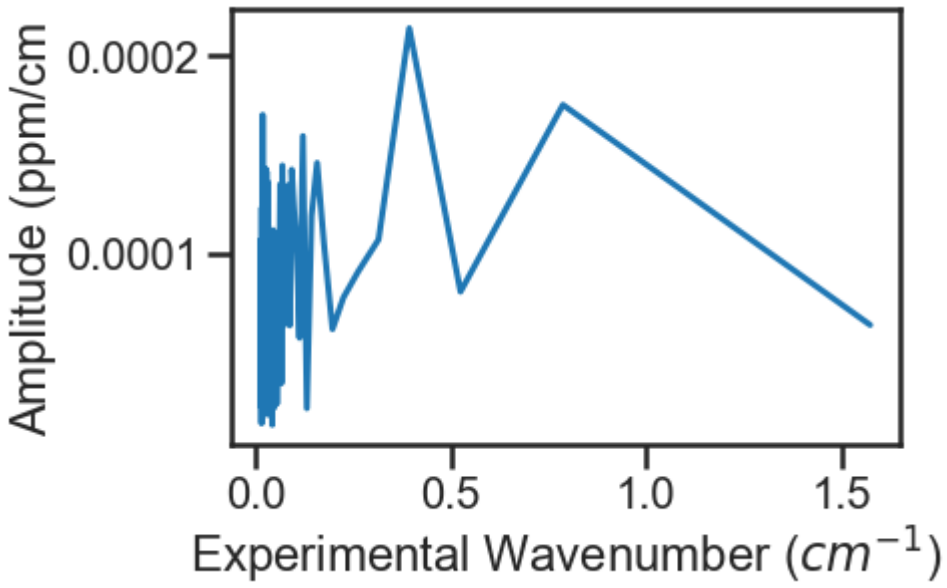
At the bottom of the experimental spectra example there is call to the *MATS.Spectrum.fft_spectrum()* function. If we hadn't included the etalon, the fit residuals would look like this:



Calling the *MATS.Spectrum.fft_spectrum()* function on these residuals gives the following result with the most abundant period being 1.271443 cm-1 and an amplitude of 0.001364. 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.



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



1.5 MATS module

class MATS.**Dataset**(*spectra*, *dataset_name*, *baseline_order=1*)

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
- **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.

average_QF()

Calculates the Average QF from all spectra.

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

Return type float

correct_component_list()

Corrects so that all spectrum 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_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_baseline_order`()

`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`()
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_extremes()
Gets the minimum and maximum wavenumber for each spectrum in the Dataset.

Returns **extreme_dictionary** – Dictionary where the key corresponds to the spectrum_num with the value equal to a list where list[0] = minimum wavenumber in spectrum and list[1] = maximum wavenumber in spectrum

Return type dict

get_spectrum_filename(*spectrum_num*)
Gets spectrum filename for spectrum in Dataset.

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

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.**Edit_Fit_Param_Files**(*base_linelist_file*, *param_linelist_file*, *new_base_linelist_file=None*, *new_param_linelist_file=None*)

Bases: object

Allows for the baseline and parameter files to be editted in jupyter notebook. Can also edit everything in .csv

Parameters

- **base_linelist_file** (*str*) – name of the .csv file containing the baseline parameters generated in format specified in the generate fit parameters class.

- **param_linelist_file** (*str*) – name of the .csv file containing the linelist parameters generated in format specified in the generate fit parameters class.
- **new_base_linelist_file** (*str*) – name of the to save the baseline param list as after editing. IF the value is None (default) then it will edit the input.
- **new_param_linelist_file** (*str*) – name of the to save the linelist param list as after editing. IF the value is None (default) then it will edit the input.

edit_generated_baselist()
Allows editting of baseline linelist in notebook

edit_generated_paramlist()
Allows editting of parameter linelist in notebook

save_editted_baselist(*baseline_widget*)
Saves edits of baseline linelist in notebook

save_editted_paramlist(*param_widget*)
Saves edits of parameter linelist in notebook

class MATS.**Fit_DataSet**(*dataset*, *base_linelist_file*, *param_linelist_file*, *minimum_parameter_fit_intensity=1e-27*, *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*)

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** (*TYPE*) – filename for file containing parmeter 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.
- **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.

constrained_baseline(*params*, *baseline_segment_constrained=True*, *xshift_segment_constrained=True*, *molefraction_segment_constrained=True*, *etalon_amp_segment_constrained=True*, *etalon_freq_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_freq_segment_constrained** (*bool, optional*) – True means the etalon frequency 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=50*, *wing_wavenumbers=50*, *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_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=50*, *wing_wavenumbers=50*, *wing_method='wing_cutoff'*)
This is the model used for fitting that includes baseline and resonant absorption 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 50.
- **wing_wavenumbers** (*float, optional*) – number of wavenumbers to simulate on either side of each line. The default is 50.
- **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.

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.

class MATS.**Generate_FitParam_File**(*dataset*, *param_linelist*, *base_linelist*, *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'*, *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*)

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.
- **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.
- **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={}*, *vary_xshift=False*, *vary_etalon_amp=False*, *vary_etalon_freq=False*, *vary_etalon_phase=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_freq** (*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.

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={}*, *vary_sw={}*, *vary_gamma0={}*, *vary_n_gamma0={}*, *vary_delta0={}*, *vary_n_delta0={}*, *vary_aw={}*, *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_base_linelist()

get_dataset()

get_param_linelist()

MATS.**HTP_from_DF_select**(*linelist*, *waves*, *wing_cutoff=50*, *wing_wavenumbers=50*, *wing_method='wing_cutoff'*, *p=1*, *T=296*, *molefraction={}*, *natural_abundance=True*, *abundance_ratio_MI={}*, *Diluent={}*, *diluent='air'*, *IntensityThreshold=1e-30*)

Calculates the absorbance (ppm/cm) based on input line list, wavenumbers, and spectrum environmental parameters.

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 {}.
- **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.**Spectrum**(*filename*, *molefraction={}*, *natural_abundance=True*, *diluent='air'*, *Diluent={}*, *abundance_ratio_MI={}*, *spectrum_number=1*, *input_freq=True*, *input_tau=True*, *pressure_column='Cavity Pressure /Torr'*, *temperature_column='Cavity Temperature Side 2 /C'*, *frequency_column='Total Frequency /MHz'*, *tau_column='Mean tau/us'*, *tau_stats_column=None*, *segment_column=None*, *etalons={}*, *nominal_temperature=296*, *x_shift=0*, *baseline_order=1*)

Bases: object

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}, ... }
- **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 the tau stats column in input file. 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.()

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_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_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*)

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

Parameters

- **x** (*array*) – array of floats used to define the x-axis
- **amp** (*float*) – amplitude of the etalon.
- **freq** (*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.**hasNumbers**(*inputString*)
Determines whether there are numbers in a string

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

Returns DESCRIPTION.

Return type bool

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

Used to stop fitting if the number of iterations is greater that 2500

MATS.**simulate_spectrum**(*parameter_linelist*, *wave_min*, *wave_max*, *wave_space*, *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={}*, *natural_abundance=True*, *abundance_ratio_MI={}*, *diluent='air'*, *Diluent={}*, *nominal_temperature=296*, *etalons={}*, *x_shift=0*, *IntensityThreshold=1e-30*, *num_segments=10*)

Generates a synthetic spectrum, where the output is a spectrum object that can be used in MATS classes.

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).
- **wave_min** (*float*) – minimum wavenumber for the simulation (cm-1)
- **wave_max** (*float*) – maximum wavenumber for the simulation (cm-1).
- **wave_space** (*float*) – 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', 'freq', 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', 'freq', 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 {}.
- **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** (*TYPE, optional*) – Number of segments in the file, which is implemented labeling the segment column into equal sequential se . The default is 10.

Returns

- **spectrum_file** (*.csv*) – File that contains the simulated wavenumber axis, noisy wavenumber axis, absorbance data, noisy absorbance data, 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.

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
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

H

M

P

R

S

U