SpectralRadex

Release 0.1.1

Apr 12, 2023

User Guide

SpectralRadex is a python module made up of two parts: a RADEX wrapper and a spectral model.

A number of libraries exist for the first purpose. However, SpectralRadex uses F2PY to compile a version of RADEX written in modern Fortran. As a result, running a RADEX model creates no subprocesses, no text files, and can be easily parallelized. Check our tutorials section for an example of running a grid of RADEX models quickly and entirely within Python.

For the second purpose, we use the RADEX calculated line opacities and excitation temperatures to calculate the brightness temperature as a function of frequency. This allows observed spectra to be modelled in python in a non-LTE fashion. See our spectral modelling tutorial for more.

CHAPTER 1

Installation

1.1 Pypi

We recommend the simple approach of using pypi:

```
pip install spectralradex
```

Note, you may receive an error of the kind

```
could not find a version that satisfies the requirement spectralradex
```

which is caused by a bug in how the requirements for spectralradex is configured. Pip will be in the process of installing a library (pandas/numpy) when this occurs and it can be solved by installing that library through pip before installing SpectralRadex.

1.2 Manual Install

However, if you wish to install manually, clone the repo and from the main directory run the following

```
python3 setup.py install
```

optionally, specify a path for the installation using

```
python3 setup.py install --prefix=/path/to/my/install
```

making sure that the install path is part of your PYTHONPATH environmental variable. You'll need to add the .egg directory eg. `` `path/to/my/install/lib/python3.7/site-packages/spectralradex-0.0.2-py3.7-linux-x86_64.egg` ``

CHAPTER 2

Formalism

2.1 RADEX

RADEX is a non-LTE radiative transfer solver that calculates the intensities of molecular lines assuming an homogeneous medium with a simple geometry. For a full description of the code, please see their release paper. The RADEX code has been modified to meet modern Fortran specifications but is otherwise unchanged in SpectralRadex.

2.2 Spectral Modelling

In order to calculate the emission from a molecular transition as a function of frequency, we need the excitation temperature and the optical depth as a function of velocity. This allows us to calculate the brightness temperature as a function of velocity:

$$T_B = [J_\nu(T_{ex}) - J_\nu(T_{BG})](1 - \exp(-\tau_v))$$

Where T_{ex} is the excitation temperature and T_{BG} is the background temperature, likely 2.73 K. In LTE, the optical depth at line centre can be calculated from the column density and Boltzmann distribution whilst the excitation temperature is assumed to be the LTE temperature. We can then calculate τ_v assuming a gaussian line profile:

$$\tau_v = \tau_0 e^{\left(-4ln(2)\frac{(v-v_0)^2}{\Delta v^2}\right)}$$

However, using RADEX, we can do better than to assume LTE. For a given set of physical parameters RADEX will provide the optical depth at line centre for every transition and the excitation temperature that gives the correct brightness temperature at line centre.

Thus, rather than using our gas kinetic temperature and an LTE derived τ_0, we can take the values for each line from an appropriate RADEX output. In the high density limit, this tends to the LTE solution but at lower densities it can deviate significantly.

In SpectralRadex, we do this for each transition in a collisional datafile between a minimum and maximum frequency set by the user. T_B is calculated as a function of frequency for each line and then combined to give the overall spectrum of the molecule.

Finally, we need to consider what to do with overlapping lines. We follow Hsieh et al 2015 and use an opacity weighted radiation temperature:

$$T_B = \left(\frac{\Sigma_i J\nu(T^i_{ex})\tau^i_v}{\Sigma_i \tau^i_v} - J_\nu(T_{BG}) \right) (1 - \exp(-\tau_v))$$

We can multiply T_B by the filling factor to get the main beam temperature.

CHAPTER 3

Referencing

SpectralRadex has been released under a MIT license and therefore you are free to use, modify or copy it in any way you wish. However, if you use SpectralRadex for any research purposes there are several references you should include.

Firstly, RADEX itself was copied essentially in its entirety and forms the basis of all model ouputs. Therefore, one should reference Van der Tak & Black 2007, the original work. Further, one should take care to acknowledge the work of those producing their collisional data. If you used the collisional files built into SpectralRadex, these were all taken from the Lamda Database where you can find the appropriate references for your molecules.

Finally, and particularly if you used the spectral modelling capabilities of SpectralRadex, we'd ask that you reference the paper in which we first described the module Holdship et al. 2021 (in prep). and point your readers in the direction of spectralradex.readthedocs.io

CHAPTER 4

Trouble Shooting

4.1 Malloc() Error

SpectralRadex can return results for up to 500 transitions. This number is hard coded because Fortran cannot use variable sized arrays as part of the python interface and so we had to choose a number which trades off a reasonably high maximum with the fact a massive array would take up a lot of memory without being needed in 99% of cases. However, if you run a species such as CH3OH over a very large frequency range, you can have more transitions than this. This will result in an error

```
malloc(): corrupted top size
Abort (core dumped)
```

which can be resolved by setting fmin and fmax such that there are fewer than 500 transitions in the range of interest. If you require more than 500 transitions, please contact us via github or email.

4.2 pip cannot find version

Could not find a version that satisfies the requirement spectralradex

If you get an error like this when trying to install spectralradex, you may need to install it from source. This can be done by running

```
git clone https://github.com/uclchem/SpectralRadex.git
pip install ./SpectralRadex
```

This happens because we use Github Actions to pre-build the library for various python versions and OS combinations. Not every combination is possible and so if your combination doesn't exist, you need to build it from source.

4.3 Mac Issues

Import Error… library not loaded

Recent updates to Mac OS have resulted in many Mac user's python distributions expecting the standard Fortran libraries to be in one place when they're actually in another. The resulting error message looks like

```
Exception has occurred: ImportErrordlopen(/usr/local/lib/python3.9/site-packages/
↪radexwrap.cpython-39-darwin.so, 2): Library not loaded: /usr/local/opt/gcc/lib/gcc/
↪10/libgfortran.5.dylib   Referenced from: /usr/local/lib/python3.9/site-packages/
↪radexwrap.cpython-39-darwin.so   Reason: image not found
```

In this case, SpectralRadex wants the libgfortran.5.dylib library and can't find it. You can solve this with locate

```
locate libgfortran.5.dylib
```

which will tell you the actual location of the required library and then you can create a symbolic link to the expected location.

```
ln /actual/path/to/libgfortran.5.dylib /the/path/python/expected/libgfortran.5.dylib
```

CHAPTER 5

Spectral Modelling Functions

`spectralradex.`**`noise_from_spectrum`**(*intensities*)

Estimate the rms noise level from a spectrum by assuming it is a gaussian noise distribution plus positive signal. If this is true, the median should be the peak of the noise distribution and values smaller are just noise. Thus, the mean square difference between the median and smaller values is the square of the noise rms.

Parameters **`intensities`** (*`float, iterable`*) – An array of the intensity values representing a spectrum

Returns The rms noise value

Return type float

`spectralradex.`**`convert_intensity_to_kelvin`**(*frequencies*, *intensities*, *minor_beam*, *major_beam*)

Convert a spectrum from jy/beam to kelvin. All spectra produced by spectralradex use kelvin so this function is intended to convert observed spectra for fitting. Treatment taken from https://science.nrao.edu/facilities/vla/proposing/TBconv

Parameters

- **`intensities`** (*`float, iterable`*) – An array of the frequency values representing a spectrum, in GHz
- **`intensities`** – An array of the intensity values at each of the frequencies in the frequency array in Jy/beam.
- **`minor_beam`** (*`float`*) – beamsize along minor axis in arcseconds
- **`major_beam`** (*`float`*) – beamsize along major axis in arcseconds

`spectralradex.`**`maxwellian_distribution`**(*v0*, *delta_v*, *tau_0*, *velocities*)

Returns the optical depth as a function of velocity, assuming gaussian line profiles and given an optical depth a line centre

Parameters

- **`v0`** (*`float`*) – Peak velocity of the emission

- **`delta_v`** – FWHM of the peaks, taken from linewidth parameter of RADEX when called via `model_spectrum()`
- **`tau_0`** (*`float`*) – The optical depth at line centre. Taken from RADEX when called via `model_spectrum()`
- **`velocities`** (*`float, iterable`*) – An iterable containing the velocity values at which to calculate tau

Returns An array with the tau value at each velocity in velocities

Return type ndarray,float

`spectralradex.`**`model_spectrum`**(*obs_freqs*, *v0*, *radex_params*, *tau_profile=<function maxwellian_distribution>*)

Calculates the brightness temperature as a function of frequency for given input frequencies, V_{LSR} velocity and RADEX parameters.

Parameters

- **`obs_freqs`** (*`iterable, float`*) – An array of frequency values in GHz at which the brightness temperature should be calculated.
- **`v0`** (*`float`*) – The V_{LSR} velocity of the emitting object to be modelled in km/s
- **`radex_params`** (*`dict`*) – A dictionary containing the inputs for the RADEX model. See `radex.get_default_parameters()` for a list of possible parameters. Note this includes the linewidth in km/s that will be used to set the shape of the emission lines.
- **`tau_profile`** (*`function, optional`*) – A function with the same arguments as `maxwellian_distribution()` that returns the optical depth as a function of velocity. If not set, spectralradex will assume gaussian line profiles centred on *v0* and a FWHM taken from the RADEX parameters.

CHAPTER 6

Radex Wrapper Functions

`spectralradex.radex.`**`run`**(*parameters*, *output_file=None*)

Run a single RADEX model using a dictionary to set parameters.

Parameters

- **`parameters`** (*`dict`*) – A dictionary containing the RADEX inputs that the user wishes to set, all other parameters will use the default values. See *`get_default_parameters()`* for a list of possible parameters and *`run_params()`* for descriptions.
- **`output_file`** (*`str`*) – If not `None`, the RADEX results are stored to this file in csv format/

`spectralradex.radex.`**`run_params`**(*molfile*, *tkin*, *cdmol*, *nh=0.0*, *nh2=0.0*, *op_ratio=3.0*, *ne=0.0*, *nhe=0.0*, *nhx=0.0*, *linewidth=1.0*, *fmin=0.0*, *fmax=500.0*, *tbg=2.73*, *geometry=1*, *output_file=None*)

Run a single RADEX model from individual parameters

Parameters

- **`molfile`** (*`float`*) – Either a full path or a relative path beginning with "." to a datafile in the Lamda database format. Alternatively, the filename of a datafile from *list_data_files()*.
- **`tkin`** – Temperature of the Gas in Kelvin
- **`cdmol`** – Column density of the emitting species in cm $^{-2}$
- **`nh`** (*`float, optional`*) – Number density of H atoms
- **`nh2`** (*`float, optional`*) – Total number density of H2 molecules, set this to o-H2 + p-H2 if using ortho and para H2 as collisional partners.
- **`op_ratio`** (*`float, optional`*) – Ortho to para ratio for H2. Defaults to statistical limit of 3 and used to set o-H2 and p-H2 densities from nh2.
- **`ne`** (*`float, optional`*) – Number density of electron.
- **`nhe`** (*`float, optional`*) – Number density of He atoms.
- **`nhx`** – Number density of H+ ions.

- **`linewidth`** (*`float, optional`*) – FWHM of the line in km s $^{-1}$.
- **`fmin`** (*`float, optional`*) – Minimum frequency below which a line is not included in the results.
- **`fmax`** (*`float, optional`*) – Maximum frequency above which a line is not included in the results.
- **`tbg`** (*`float, optional`*) – Background temperature, defaults to CMB temperature 2.73 K.
- **`geometry`** (*`int, optional`*) – Choice of geometry of emitting object. 1 for sphere, 2 for LVG, 3 for slab.

`spectralradex.radex.`**`run_grid`**(*parameters*, *target_value='FLUX (K*km/s)'*, *pool=None*)

Runs a grid of RADEX models using all combinations of any iterable items in the parameters dictionary whilst keeping other parameters constant. Returns a dataframe of results and can be parallelized with the `pool` parameter.

Parameters

- **`parameters`** – A dictionary of parameters as provided by *`get_default_parameters()`* or *`get_example_grid_parameters()`*. Parameters should take a single value when they are constant over the grid and contain and interable if they are to be varied.
- **`molfile`** (*`str`*) – Either a full path or a relative path beginning with "." to a datafile in the Lamda database format. Alternatively, the filename of a datafile from *list_data_files()*.
- **`target_value`** (*`str,optional`*) – RADEX output column to be returned. Select one of 'T_R (K)', 'FLUX (K*km/s)', 'FLUX (erg/cm2/s)'
- **`pool`** (*`Pool, optional`*) – a Pool object with `map()`, `close()` , and `join()` methods such as multiprocessing.Pool or schwimmbad.MPIPool. If supplied, the grid will be calculated in parallel.

`spectralradex.radex.`**`get_default_parameters`**()

Get the default RADEX parameters as a dictionary, this largely serves as an example for the input required for *`run()`*.

molfile should be a collsional datafile in the LAMDA database format. If using a local file, a full path or a relative path beginning with "." is required. Otherwise, one of the files listed by *`list_data_files()`* can be supplied without a path.

method is 1 (uniform sphere), 2 (LVG), or 3 (slab)

`spectralradex.radex.`**`get_example_grid_parameters`**()

Returns a dictionary of parameters for RADEX with iterables which can be used with *`run_grid()`*.

`spectralradex.radex.`**`get_transition_table`**(*molfile*)

Reads a collisional data file and returns a pandas DataFrame for the molecule with one row per transition containing the Einstein coefficients, upper level energy and frequency.

Parameters **`molfile`** (*`str`*) – Either the full path to a collisional datafile or the filename of one supplied with SpectralRadex

`spectralradex.radex.`**`get_collisional_partners`**(*molfile*)

Reads a collisional data file and returns a dictionary containing the number of collisional partners and their names. The partner names match the input keys for *`run()`*

Parameters **`molfile`** (*`str`*) – Either the full path to a collisional datafile or the filename of one supplied with SpectralRadex

`spectralradex.radex.`**`thermal_h2_op_ratio`**`(`*tkin*`)`

If your data file has collisions with p-h2 and o-h2, you may want to use the thermal ratio to split your total H2 density. You can check that value for any given temperature with this function. Returns the ortho:para ratio as a float

Parameters **`tkin`** (*`float`*) – Gas kinetic temperature

`spectralradex.radex.`**`list_data_files`**`()`

SpectralRadex is packaged with a selection of LAMDA collisional datafiles. This function prints the list of available files. You can provide the full path to another file in the parameter dictionary to use one not packaged with SpectralRadex.

Note: This tutorial was generated from an IPython notebook that can be downloaded here.

CHAPTER 7

Radex

```
from spectralradex import radex
from multiprocessing import Pool
import numpy as np
import time
```

The simplest use case for SpectralRadex is to be a simple python wrapper for RADEX. This allows large grids of RADEX models or complex parameter inference procedures to be run in an environment suited to those tasks.

If one wishes to run radex, we simply need a dictionary of the parameters RADEX expects. An example can be obtained using the `get_default_parameters()` function like so

```
params = radex.get_default_parameters()
print("{")
for key,value in params.items():
    print(f"\t{key} : {value}")
print("}")
```

```
{
    molfile : co.dat
    tkin : 30.0
    tbg : 2.73
    cdmol : 10000000000000.0
    h2 : 100000.0
    h : 0.0
    e- : 0.0
    p-h2 : 0.0
    o-h2 : 0.0
    h+ : 0.0
    linewidth : 1.0
    fmin : 0.0
    fmax : 1000.0
    geometry : 1
}
```

Note that each possible collisional partner has a separated entry for its density. You can check the collisional partners in your datafile with get_collisional_partners()

```
radex.get_collisional_partners("co.dat")
```

```
{'Number': 2, 'Partners': ['p-h2', 'o-h2']}
```

You only need to provide densities for the partners you wish to include in the calculation but you must include at least one of the partners. Two species cases are:

- RADEX will sum the o-h2 and p-h2 density values to produce a h2 value.
- A small number of datafiles have p-H2 collsions only and you may wish to place your total h2 density in that entry to approximate the o-h2 collisions

Once your parameter dictionary is set up, we pass that to the `run()` function.

```
output = radex.run(params)
output.head()
```

7.1 Parameter Grids

It is more likely that one will want to run the code over many combinations of input parameters. This can be achieved via the `run_grid()` function. This function also takes a parameter dictionary of the same format as `run()`. However, variables which are too be varied over the grid should be supplied as iterables.

Furthermore, to keep things simple, the desired RADEXtakes iterables for the three variables (density, temperature and column density) as well as fixed values for the other RADEX parameters. It then produces the RADEX output for all combinations of the three iterables.

We'll use an example grid which can be acquired using the `get_example_grid_parameters()` function.

```
parameters=radex.get_example_grid_parameters()
parameters
```

```
{'molfile': 'co.dat',
 'tkin': array([ 10. ,  82.5, 155. , 227.5, 300. ]),
 'tbg': 2.73,
 'cdmol': array([1.e+14, 1.e+15, 1.e+16, 1.e+17, 1.e+18]),
 'h2': array([   10000.        ,    56234.13251903,   316227.76601684,
         1778279.41003892, 10000000.        ]),
 'h': 0.0,
 'e-': 0.0,
 'p-h2': 0.0,
 'o-h2': 0.0,
 'h+': 0.0,
 'linewidth': 1.0,
 'fmin': 0.0,
 'fmax': 800.0,
 'geometry': 1}
```

```
tic = time.perf_counter()

grid_df = radex.run_grid(parameters,target_value="T_R (K)")
toc = time.perf_counter()
print(f"run_grid took {toc-tic:0.4f} seconds without a pool")
```

```
run_grid took 2.8573 seconds without a pool
```

```
grid_df.iloc[:,0:6].head()
```

7.1.1 Parallelization

In order to be as flexible as possible, SpectralRadex has no built in multiprocessing. However, the `run_grid()` function does take the optional parameter `pool` which should be an object with `map()`, `join()`, and `close()` methods that allow functions to be evaluated in parallel. For example, the python standard multiprocessing.pool obect or Schwimmbad's MPIPool.

If such an object is supplied, the grid will be evaluated in parallel. Note the time in the example below compared to the grid above.

```
tic = time.perf_counter()
pool=Pool(8)
grid_df = radex.run_grid(parameters,target_value="T_R (K)",pool=pool)
toc = time.perf_counter()
print(f"run_grid took {toc-tic:0.4f} seconds with a pool of 8 workers")
grid_df.iloc[:,0:6].head()
```

```
run_grid took 0.7338 seconds with a pool of 8 workers
```

Note: This tutorial was generated from an IPython notebook that can be downloaded here.

CHAPTER 8

Spectral Modelling

One of SpectralRadex's key features is the ability to generate model spectra from RADEX models. In this example, we show how to generate a spectrum.

```
import spectralradex
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

First, we need a radex model. This is just a dictionary with the RADEX inputs as keys. For this example, we'll start by grabbing the default parameters from the radex wrapper. Then we'll increase the CO column density and the linewidth as well as setting fmax to 300 GHz.

```
radex_params=spectralradex.radex.get_default_parameters()

radex_params["cdmol"]=1e16
radex_params["p-h2"]=1e4
radex_params["o-h2"]=1e4

radex_params["linewidth"]=10
radex_params["fmax"]=300

print(radex_params)
```

```
{'molfile': 'co.dat', 'tkin': 30.0, 'tbg': 2.73, 'cdmol': 1e+16, 'h2': 100000.0, 'h':␣
↪0.0, 'e-': 0.0, 'p-h2': 10000.0, 'o-h2': 10000.0, 'h+': 0.0, 'linewidth': 10, 'fmin
↪': 0.0, 'fmax': 300, 'geometry': 1}
```

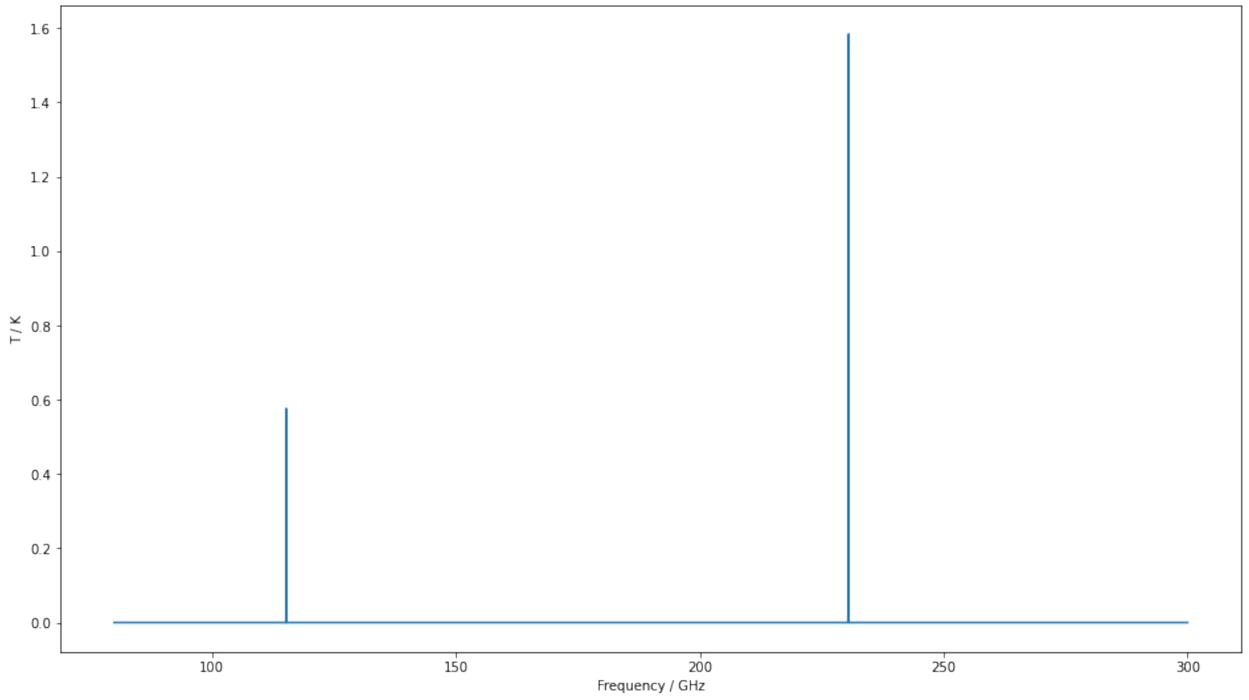
We also need a list of frequencies over which we'd like the spectrum. Here, we'll generate a spectrum with a 5 MHz resolution between 80 and 300 GHz. Getting the model intensities is a simple function call which will return a pandas dataframe of Frequency and Intensity.

The intention of SpectralRadex is to model observations. Thus, the first two inputs to the spectral modelling function are intended to match some observations: the frequency bins you observed and the assume V_{LSR} of the object.

```
frequencies=np.arange(80,300,0.005)
v0=0.0
spectrum=spectralradex.model_spectrum(frequencies,v0,radex_params)
```

```
/home/jon/.local/lib/python3.8/site-packages/spectralradex/__init__.py:178:␣
↪RuntimeWarning: invalid value encountered in true_divide
  rad_weights=np.sum(rad_weights,axis=0)/taus
```

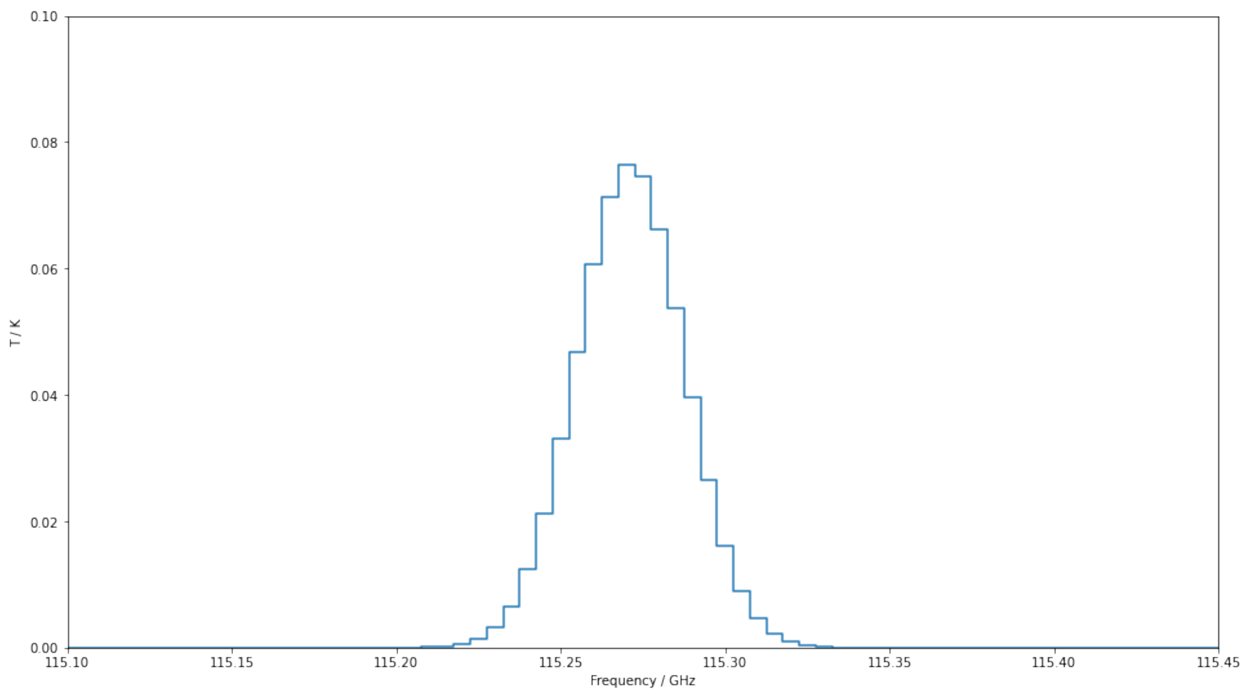
```
fig,ax=plt.subplots(figsize=(16,9))
ax.plot(spectrum["Frequency"],spectrum["Intensity"],drawstyle="steps-mid")
settings=ax.set(xlabel="Frequency / GHz",ylabel="T / K")
```



The above example shows two extremely narrow peaks but if we increase the linewidth a little and zoom in, we can see the Gaussian shape we assume for all line in SpectralRadex.

```
radex_params["linewidth"]=100
spectrum=spectralradex.model_spectrum(frequencies,v0,radex_params)
fig,ax=plt.subplots(figsize=(16,9))
ax.plot(spectrum["Frequency"],spectrum["Intensity"],drawstyle="steps-mid")
settings=ax.set(xlim=(115.1,115.45),ylim=(0,0.1),xlabel="Frequency / GHz",ylabel="T /␣
↪K")
```

```
/home/jon/.local/lib/python3.8/site-packages/spectralradex/__init__.py:178:␣
↪RuntimeWarning: invalid value encountered in true_divide
  rad_weights=np.sum(rad_weights,axis=0)/taus
```

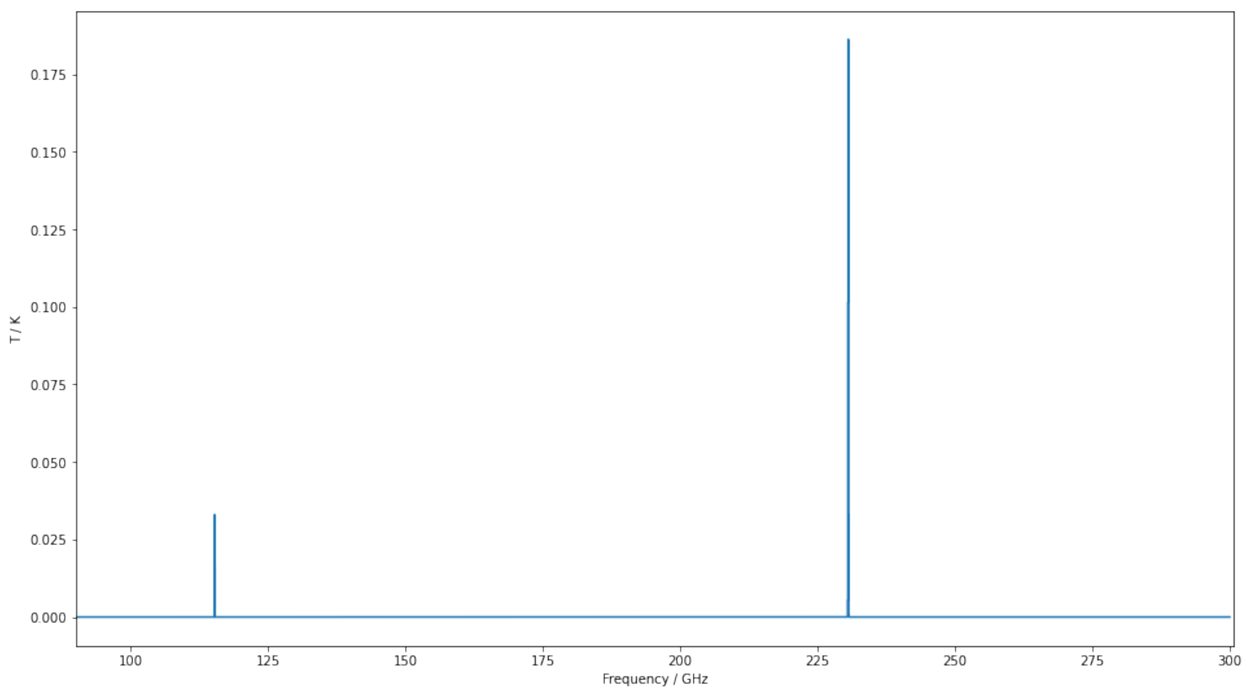


Finally, please note that if you sample with too large a frequency bin, you'll miss lines. We are still considering what the default behaviour should be in this case. For now, SpectralRadex will warn you if the velocity bins are larger than the linewidth.

Here we repeat the above calculation with a 50 MHz frequency spacing.

```
frequencies=np.arange(30,300,0.05)
v0=0.0
spectrum=spectralradex.model_spectrum(frequencies,v0,radex_params)
fig,ax=plt.subplots(figsize=(16,9))
ax.plot(spectrum["Frequency"],spectrum["Intensity"],drawstyle="steps-mid")
settings=ax.set(xlim=(90,300.6),xlabel="Frequency / GHz",ylabel="T / K")
```

```
/home/jon/.local/lib/python3.8/site-packages/spectralradex/__init__.py:178:␣
↪RuntimeWarning: invalid value encountered in true_divide
  rad_weights=np.sum(rad_weights,axis=0)/taus
```


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
T / K
100
125
150
175
200
225
250
275
300
Frequency / GHz

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index