GaussPy Documentation

Release 1.0

Robert Lindner, Carlos Vera-Ciro

February 12, 2016

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

When interpreting data, it is useful to fit models made up of parametrized functions. Gaussian functions, described simply by three parameters and often physically well-motivated, provide a convenient basis set of functions for such a model. However, any set of Gaussian functions is not an orthogonal basis, and therefore any solution implementing them will not be unique. Furthermore, determining fits to spectra involving more than one Gaussian function is complex, and the crucial step of guessing the number of functions and their parameters is not straightforward. Typically, reasonable fits can be determined iteratively and by-eye. However, for large sets of data, analysis by-eye requires an unreasonable amount of processing time and is unfeasible.

This document describes the installation and use of GaussPy, a code for implementing an algorithm called Autonomous Gaussian Decomposition (AGD). AGD using computer vision and machine learning techniques to provide optimized initial guesses for the parameters of a multi-component Gaussian model automatically and efficiently. The speed and adaptability of AGD allow it to interpret large volumes of spectral data efficiently. Although it was initially designed for applications in radio astrophysics, AGD can be used to search for one-dimensional Gaussian (or any other single-peaked spectral profile)-shaped components in any data set.

To determine how many Gaussian functions to include in a model and what their parameters are, AGD uses a technique called derivative spectroscopy. The derivatives of a spectrum can efficiently identify shapes within that spectrum corresponding to the underlying model, including gradients, curvature and edges. The details of this method are described fully in Lindner et al. (2015), AJ, 149, 138.

CHAPTER TWO

INSTALLATION

2.1 Dependencies

- Python 2.7
- Numpy
- Scipy
- h5py
- GNU Scientific Library (GSL)

If you do not already have Python 2.7, you can install the Anaconda Scientific Python distribution, which comes pre-loaded with Numpy, Scipy, and h5py.

To obtain GSL:

```
sudo apt-get install libgsl0-dev
```

2.2 Download GaussPy

Download GaussPy from...

2.3 Installing GaussPy

To install make sure that all dependences are already installed and properly linked to python –python has to be able to load them–. Then cd to the local directory containing gausspy and type

$ python setup.py install

If you don’t have root access and/or wish a local installation of gausspy then use

$ python setup.py install –user

change the ‘requires’ statement in setup.py to include scipy and lmfit

CHAPTER

THREE

QUICKSTART TUTORIAL

Before applying AGD to your data using GaussPy, you must first train the AGD algorithm to determine the optimal value of the smoothing parameter α. This training requires you to apply AGD to a dataset with known underlying Gaussian decomposition. In other words, you need to have a training dataset for which you know (or have an estimate of) the true Gaussian model. This training dataset can be composed of real (i.e. previously analyzed) or synthetically-constructed data, for which you have prior inofmraiton about the underlying correct decomposition. This prior information is used to maximize the model accuracy by calibrating the α parameter used by AGD.

3.1 Creating a Synthetic Training Dataset

Training datasets can be constructed by adding Gaussian functions with parameters drawn from known distributions with known uncertainties. For example, we can create a mock dataset with NSPECTRA-realizations of the function

$$S(x_i) = \sum_{k=1}^{\mathtt{NCOMPS}} \mathrm{AMP}_k \exp\left[-\frac{4\ln 2(x_i - \mathtt{MEAN}_k)^2}{\mathtt{FWHM}_k^2}\right] + \mathtt{NOISE}, \qquad i = 1, \cdots, \mathtt{NCHANNELS} \tag{3.1}$$

where

1. NSPECTRA is then the number of synthetic spectra to be created
2. NCOMPS is the number of components in each synthetic spectrum
3. (AMP, MEAN, FWHM) are the parameters of each Gaussian component
4. NOISE is the level of noise introduced in each spectrum

In the next example we will show how to implement this in python. We have made the following assumptions

1. $\mathrm{NOISE} \sim N(0, \mathrm{RMS}) + f \times \mathrm{RMS}$ with RMS=0.05 and $f = 0$
2. NCOMPS = 4 **This number is fixed, i.e. not random**
3. NCHANNELS = 512 This number sets the resolution of each spectrum. **Does this number need to be the same for all spectra in AGD?**
4. $\mathrm{AMP} \sim \mu(5\mathrm{RMS}, 25\mathrm{RMS})$, this way we ensure that every spectral feature is above the noise level. Spectra with a more dominant contribution from the noise can also be generated and used as training sets for AGD
5. $\mathrm{FWHM} \sim \mu(10, 35)$ and $\mathrm{MEAN} \sim \mu(0.25, 0.75) \times \mathrm{NCHANNELS}$, note that for our choice of the number of channels, this selection of FWHM ensures that even the wider component can be fit within the spectrum.

```
# GaussPy Example 1
# Create spectra with Gaussian profiles

import numpy as np
import matplotlib.pyplot as plt
import pickle

# Specify the number of spectral channels (NCHANNELS)
NCHANNELS = 512
# Specify the number of spectra (NSPECTRA)
NSPECTRA = 1000

# Estimate of the root-mean-square uncertainty per channel (RMS)
RMS = 0.05

# Estimate the mean number of Gaussian functions to add per spectrum (NCOMPS)
NCOMPS = 4

# Specify the min-max range of possible properties of the Gaussian function paramters:
# Amplitude (AMP)
AMP_lims = [RMS * 5, RMS * 25]
# Full width at half maximum in channels (FWHM)
FWHM_lims = [10, 35] # channels
# Mean channel position (MEAN)
MEAN_lims = [0.25 * NCHANNELS, 0.75 * NCHANNELS]

# Indicate whetehre the data created here will be used as a training set
# (a.k.a. decide to store the "true" answers or not at the end)
TRAINING_SET = True

# Specify the pickle file to store the results in
FILENAME = 'agd_data_science.pickle'
```

With the above parameters specified, we can proceed with constructing a set of synthetic training data composed of Gaussian functions with known parameters (i.e., for which we know the "true" decompositon), sampled randomly from the parameter ranges specified above. The resulting data, including the channel values, spectral values and error estimates, are stored in the pickle file specified above. If we want this to be a training set (`TRAINING_SET = True`), the "true" decomposition answers for estimating the accuracy of a decomposition are also stored in the output file. For example, to construct a synthetic dataset:

```
# GaussPy Example 1
# Create spectra with Gaussian profiles -cont-

# Initialize
agd_data = {}
chan = np.arange(NCHANNELS)
errors = chan * 0. + RMS # Constant noise for all spectra

# Begin populating data
for i in range(NSPECTRA):
    spectrum_i = np.random.randn(NCHANNELS) * RMS

    # Sample random components:
    amps = np.random.rand(NCOMPS) * (AMP_lims[1] - AMP_lims[0]) + AMP_lims[0]
    fwhms = np.random.rand(NCOMPS) * (FWHM_lims[1] - FWHM_lims[0]) + FWHM_lims[0]
    means = np.random.rand(NCOMPS) * (MEAN_lims[1] - MEAN_lims[0]) + MEAN_lims[0]

    # Create spectrum
    for a, w, m in zip(amps, fwhms, means):
        spectrum_i += gaussian(a, w, m)(chan)

    # Enter results into AGD dataset
    agd_data['data_list'] = agd_data.get('data_list', []) + [spectrum_i]
    agd_data['x_values'] = agd_data.get('x_values', []) + [chan]
    agd_data['errors'] = agd_data.get('errors', []) + [errors]

    # If training data, keep answers
    if TRAINING_SET:
        agd_data['amplitudes'] = agd_data.get('amplitudes', []) + [amps]
        agd_data['fwhms'] = agd_data.get('fwhms', []) + [fwhms]
        agd_data['means'] = agd_data.get('means', []) + [means]

# Dump synthetic data into specified filename
pickle.dump(agd_data, open(FILENAME, 'w'))
```

3.2 Training the Algorithm

With a real or synthetic training dataset in hand, we will apply AGD to the training dataset and compare the results with the known underlying decompositon to determine the optimal value for the smoothing parameter α.

To begin, import GaussianDecomposer from GaussPy:

```
import gausspy.GaussianDecomposer as gp

g = gp.GaussianDecomposer()
```

Next, load the training dataset for analysis:

```
g.load_training_data('agd_data.pickle')
```

We will begin with a one-phase decomposition (two-phase decomposiiton will be explained in later sections):

```
# One phase training
g.set('phase', 'one')
```

Next, we set the signal to noise ratio (SNR) threshold below which AGD will not be allowed to include Gaussian functions in the model:

```
# threshold below which Gaussian components will not be fit
g.set('SNR_thresh', 5.)
```

Finally, we specify an initial guess for the $alpha$ value [**how close does this have to be?] and begint he training process:

```
# initial guess for the alpha value
g.train(alpha1_initial = 10.)
```

GausspPy will iterate over a range of $alpha$ values and compare the decomposition associated with each $alpha$ value to the correct decomposition specified within the training dataset to maximize the accuracy of the decomposition.

Once the training is completed, we can view the “trained” value of alpha by looking at the attribute of our GaussianDecomposer instance.

```
# get the parameters attribute of g, which is a dictionary of important
# variables
print(g.p['alpha1'])
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search