Characterizing the time dependence of source intensity using the JCMT

Rebecca Chen
ASIAA Summer Student Program 2018
Supervisors: Sofia Wallstrom, Jonty Marshall,
Peter Scicluna, Sundar Srinivasan

August 31st, 2018

Contents

1 Project Introduction

This project works with data from the James Clerk Maxwell Telescope (JCMT) in the NESS group, or Nearby Evolved Stars survey, which has long term goals to investigate physical mechanisms such as AGB mass loss and dust formation.

JCMT is a dish telescope designed for sub-mm observation and therefore has the characteristics of antennae. For antennae, the radiation pattern has larger values for a certain range of pointing direction than the rest of the sphere; this region is called the main beam or main lobe, whereas the rest are called the side and back lobes. The main beam efficiency, or the fraction of power concentrated in the main beam, depends on elevation and weather, among other factors, and will therefore vary with time. This time dependence makes calibration somewhat difficult.

This project was originally intended to fit a function and characterize the time dependence of main beam efficiency of the HARP and RxA3 heterodyne receivers on the JCMT as a part of NESS and eventually integrated into STARLINK for all JCMT users. We intended to be able to input a date and time, and output a value for main beam efficiency, as well as an uncertainty to propagate. It developed into an exploration of the limits of using Bayesian inference and MCMC, as well as forward modeling and various complications of fitting a model to data. This document is intended to describe in detail what was tried and what didn't work, as well as to document the organization of my scripts and functions. All of my cleaned up code is accessible on my github at `https://github.com/rebeccachen0/asiaa`.

A description of the folders and scripts on my github can be found in FILES.txt. Throughout this document, I've bolded the file names as they appear in the chronology of my work. In general, the iPython notebooks will have diagnostic plots, and the Python scripts are more cleaned up and general. For any clarification, feel free to email me at rebeccachen@uchicago.edu.

2 Bayesian Inference and MCMC

2.1 Bayesian Inference

Bayesian Inference uses Bayes' Theorem as a method of incorporating already known information a.k.a "priors" into computing probabilities. It's often seen as a counterpart to frequentist statistics. While a frequentist method asks "given a model, how likely am I to observe my data?", Bayesian inference lets us ask "given our data, how likely is this model?", which is the way science is actually conducted. It also lets us supply information about what we expect or place limits what what we know is physically impossible. Bayes' Theorem is a mathematical formulation of this idea:

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)} \tag{1}$$

The notation $P(A \mid B)$ is simply read as "the probability of A given B." It

may be easier to understand in the concrete terms we need to use:

$$P(model \mid data) = P(\theta \mid y) = \frac{P(data \mid model) \times P(model)}{P(data)} \tag{2}$$

Here is a breakdown of the terms:

1. Likelihood – $P(data \mid model)$ – The probability of observing our data given a model
2. Prior – $P(model)$ – The probability the model is correct (before observing data)
3. Evidence – $P(data)$ – The probability of observing our data under all possible models, also given by $\int P(\theta)P(y \mid \theta)d\theta$
4. Posterior – $P(model \mid data)$ – The probability of a model given our data

2.2 MCMC

MCMC stands for Markov Chain Monte Carlo. The Markov Chain part has to do with a particular sequential method of generating random samples, where new samples depend exclusively on the previous one, and none of the other preceding ones. The Monte Carlo part has to do with drawing samples from a distribution in order to estimate properties about it. Essentially, MCMC is a set of algorithms for sampling from a probability distribution without having to know all of its mathematical properties, for example its exact height.

This project utilized emcee, a Python implementation of Goodman and Weare's affine invariant MCMC Ensemble sampler (Foreman-Mackey et al. 2012). An algorithm that is affine invariant performs equally well under all linear transformations and will therefore be insensitive to covariances among parameters. The basic idea for the algorithm is that a set of "walkers" step through the parameter space, at each step either accepting and rejecting a proposed sample based on its comparison with the current step and their heights in the posterior distribution. Ultimately, this allows us to work with a set of samples for each parameter and extract statistical information from them. Because the denominator on the right hand side of Bayes' Theorem (the evidence) is in practicality hard to compute, we can use MCMC to sample from the numerator (likelihood*prior), which is proportional to the posterior.

We started by trying to implement this simple example given here in the emcee documentation, which can be used to fit a linear model to data with slope m, y-intercept b, and some fractional underestimated uncertainty f (**exact_emcee_example.ipynb, exact_example_data.ipynb**). Note: The source code link for the example is 404 and can be found here instead, but the code seems to be out of date. Many of the diagnostic plots (e.g. the walker paths) do not work as intended if directly copy-pasted. Edited, working versions can be found in my code instead.

Quick Notes on Terminology

- Uncertainty: This is due to of the measurement of data. When you make a measurement you're drawing from a probability distribution with some spread.
- Scatter: This is due to extra physics/details/things that are complicated and you can't put explicitly into your model.
- Systematic error/Bias: In our case, something like the misalignment of telescope or a consistent mismeasurement.

2.3 Uncertainty

The original intent was for the model to account for uncertainty, scatter, and systematic error/bias (largely due to misalignment of the telescope). We first tried just removing the f term to get the example to work; this required converting the dates into days elapsed and removing daytime observations. The data provided does not outright provide uncertainties, so we started by assuming a 10% uncertainty. While everything ran properly, it was difficult to tell whether it was actually working, as we had yet to account for scatter and bias as well (**mcmc.ipynb**). This notebook also includes a version that tries to infer the uncertainty as well.

2.4 Accounting for Intrinsic Scatter

The method we used is detailed in Chapter 7 and 8 of a paper by Hogg, Bovy, and Lang found here. This required coding up a new likelihood function (eq 35):

$$\ln \mathscr{L} = K - \sum_{i=1}^{N} \frac{1}{2} \ln(\Sigma_i^2 + V) - \sum_{i=1}^{n} \frac{\Delta_i^2}{2[\Sigma_i^2 + V]} \tag{3}$$

This accounts for intrinsic scatter and two dimensional uncertainties. We model the distribution beginning with a set of points lying on a narrow linear relation, with a Gaussian offset drawn from a 2D Gaussian with a covariance tensor. Luckily the matrix algebra simplifies out, as we only have a y error, as running code using NumPy matrix multiplication is much slower. We can then introduce a intrinsic Gaussian variance V, or the scatter, which is orthogonal to the line. The likelihood then becomes the convolution of the 2D uncertainty Gaussian with V. Instead of *f*, we now have *lnyerr* and lnV (**week2.ipynb**).

2.5 Bias and Non-Gaussian Likelihood

We first tried to implement the bias by adding another scatter-like term, but this caused problems as the scatter is symmetric while the bias is asymmetric and only lowers values (**bias.ipynb**, **test_nobias.ipynb** – same version but without bias). At this point, instead of using the least squares result as the starting point for the walkers, we manually inputted an array of informed guesses.

We looked to Section 5 in the same Hogg paper on Non-Gaussian uncertainties, comparing eqs 22 and 27 (**non_gaussian_likeli.ipynb**). Eq 22 is as

follows:

$$p(y_i|x_i, \sigma_{yi}, m, b) = \sum_{j=1}^{k} \frac{a_{ij}}{\sqrt{2\pi\sigma_{yij}^2}} \exp\left(-\frac{[y_i + \Delta y_{ij} - mx_i - b]^2}{2\sigma_{yij}^2}\right) \quad (4)$$

Note that the likelihood is the product over all i datapoints of the probability function $p(y_i \mid \theta)$. We removed the bias term and instead tried to implement an asymmetrical y error term, at first just subtracting a Δy term. I also worked out a log approximation for equation 22 in an attempt to derive the proper likelihood. We started with the case of two Gaussians for simplicity (uncertainty and bias), deciding to temporarily ignore the scatter, and tried various combinations of hard-coding and varying parameters (**no_scatter_eq22_2gauss.ipynb**). The parameters of interest here are m and b as usual, and then *delty1*, *sigsq1*, and *a1* for the center, sigma squared, and weight of the first Gaussian respectively, and similarly for the second. Using two Gaussians caused the walkers to branch into two paths, either using all Gaussian 1 or all Gaussian 2, as the distribution was too bimodal, so we tried a version with three Gaussians to smooth out the likelihood (**no_scatter_eq22_3gauss.ipynb**).

We also tried adding an inverse correlation between the widths of the Gaussians by adding an additional factor parameter and defining limits based on the relationships between their means and variances, but it didn't work and pushed every parameter to be zero. This looked like it might be working, but the uncertainties on some values (given by the 16th and 84th percentile of the sampling results) were enormous, on the order of 10^7. Each run of MCMC also seems to give wildly varying results.

2.5.1 Matrix Version

In order to generalize the method, I coded up Sundar's derivation of a matrix version of equation 22 (see Sundar's writeup). We had also hoped to potentially convolve this with the intrinsic scatter V to reinclude it in the model, but did not end up doing so. This version at first pushed all the parameters to be zero again (**matrix_eq22.ipynb**). This was due to the H matrix, which contained an exponent to a large number. We fixed this by forcing the chisq term to be no greater than a limit based on machine accuracy. While the code ran, there were more problems: the resulting b was essentially unconstrained, and it was selecting entirely one Gaussian while forcing the other to be zero.

2.5.2 Constrained Optimization

We then implemented a Lagrange multiplier L in the likelihood, hoping that it would force the a coefficients to be non-zero. Sundar also derived the equation for the optimal lambda, L^*, which we used to set a prior. In order to do this you can simply input your best guesses into the likelihood, uncomment the L^* print statement, and run it once. While this outputted non-zero coefficients, the other parameter values seemed to be still only arbitrarily or randomly reasonable, and the walker paths seemed to freeze in place all across the parameter space.

2.6 Last Attempts

As a last try, I generated two sets of fake data, one with the simple model detailed below as used for forward modeling (Gaussians centered on zero), and one with a mixture of Gaussians, with one offset, as the likelihood is coded up to be (**mcmc_fakedata.ipynb**). Both were generated with more evenly spaced data points as well as four times the number of data points, but neither successfully reproduced the data or even outputted consistent results. It remains somewhat unclear why this method did not work fully– it could be a problem with the implementation or it may be features of the data. We decided to focus on the forward modeling method.

2.7 Notes on Using MCMC

2.7.1 Priors

We started by using flat or uninformative priors– in other words, ones that simply restrict parameters to be within a certain range. We moved on to using more informative priors, Gaussians centered at some value that we expect to be reasonable.

2.7.2 Troubleshooting

Have you tried . . . ?

- Changing the actual starting positions (based on first guesses instead of chi-sq optimization result, which is useless after adding the bias)
- Increasing/decreasing the size of the starting position range for the walkers (proportional to the initial guess range and specific to the parameter)
- Adding more threads to the sampler, using the threads argument in the sampler creation
- Double checking that your prior is implemented correctly by plotting it or sampling it
- Checking the acceptance fraction and autocorrelation time (may throw a "chain is too short to estimate" error)
- Increasing the number of walkers
- Increasing the number of steps (longer runtime) and/or step size (using the a argument in the sampler creation
- Note: we didn't realize for a while, but the walker paths plot shouldn't look spiky! If they are, you may be using the outdated code from the emcee example.
- Checking out this comprehensive paper, by Hogg and Foreman-Mackey

Other sources of error could be:

- Many of the parameters should be thought about in logspace, so it's easy to input incorrect parameter guesses or priors
- Keeping track of where σ vs. σ^2 are used

3 Forward Modeling

3.1 Introduction

The basic idea behind this approach is the opposite direction of the previous method. The process is to first create a model that generates datasets based on input parameters. We then create a grid that represents the parameter space, or essentially generate all the possible combinations of the parameters. Then for each set of parameters we generate 1000 sample datasets, in order to obtain a distribution of results that will allow us to estimate our uncertainties – this is a Monte Carlo method, the same MC from as the MCMC method. We then compare these datasets to the real data in order to determine the best fit, complete with uncertainties. (**forward_modeling.ipynb**, **forward_modeling.py** – clean version).

3.2 The Model

We started with a line defined by $mx + b$ with a y error and scatter drawn from normal distributions and a bias drawn from a half-normal distribution. We later removed the scatter term, as the y error estimate was large enough to account for it, changed the bias distribution to an exponential, and added a multiplicative factor on the bias (*rel* parameter). We also tried using Peter's calculation of the y error directly instead of drawing it from a distribution (functions with "yerrfix" label).

3.3 Best Fit

We looked into a few statistical tests as a measure for comparing goodness of fit– the KS test, Mann Whitney U test, and Wilcoxon signed rank test. The relationship between test statistics and p-values for the Wilcoxon test proved to be strange, and the Mann Whitney U is used for independent samples, so we went ahead with the standard KS test. After obtaining the test statistic distributions, we fit a Gaussian to each distribution and found the lowest mean, which corresponded to the best fit parameter set. Due to computation time, we had to start with coarser grids of 5 across, testing 4 parameters (m, b, *bias_sig*, *rel*). Each time we took the parameters corresponding to the minimumized statistic as the center of the new range and halved the size of the range. We did this five times to obtain a set of starting parameters for our finer grid.

3.3.1 KS Test

The KS test, which stands for Kolmogorov-Smirnov test, compares two Cumulative Distribution functions. For example, the CDF of a variable X evaluated

at some value x is the probability that the variable X will take x or smaller. The KS test essentially looks at the maximum distance between two CDF's as the test statistic, and we minimized this stat to find the best fit.

3.4 Gridding

As computing the fine grid required more computing power than I had, I split it along one parameter, m, and ran it as many times as the number of spacings. The number of parameter sets produced is given by $\#ofspacings^{\#ofparameters}$, and based on the time I had remaining the program we went with a spacing of 8. The resulting slices require approx 2.5 GB of storage space, so they are not on my github, but I have them on a hard drive if they're needed. I then stitched the slices back together and found the minimum test statistic.

In order to obtain the uncertainties, I calculated the MADM, or median absolute deviation median. This is done by taking some x% of the lowest test stats, finding the median of these values, then calculating the absolute deviation for each of the stats from the median, and finally taking the median of those deviations. Then, we list all sets of parameters with test statistics that lie within a range defined by an estimator for the uncertainty (MADM*k, where $k = 1.4826$) and obtain a distribution for the corresponding parameters. I fitted a Gaussian to each of these distributions and used the means as the best fit parameters, with the standard deviation as the symmetrical uncertainty (**fine_grid_analysis.py**). Somehow the CDF of the dataset resulting from these parameters was visually a very poor fit with the CDF of the actual data, so I included instead the result of this analysis on the 5th coarse grid instead. I also ran into quite a few issues with storage and memory here, as well as weirdness with pickle not running on OS, but working fine on Scientific Linux.

3.5 Results

The following is the result after running the 5th coarse grid through fine_grid_analysis.py

Parameter	Result	Uncertainty
m	3.3×10^{-20}	± 0.001
b	0.57	± 0.014
$bias\_sig$	0.35	± 0.014
rel	0.45	± 0.049

Table 1: Preliminary Results

4 Future Work

The finer grid method still needs more computational power, and the method for determining uncertainties, as well as whether the appropriate distributions are actually Gaussian, should be double-checked. Alternatively, we could automate the process of refining the coarse grids, as I did that manually. The RxA3 plot

also needs to be looked at, and have a function fitted to each section. Finally, we could also try maximizing entropy instead of the likelihood for the MCMC method.

References

[1] Foreman-Mackey, D., Hogg, D. W., Lang, J., & Goodman, J., 2012, emcee: The MCMC Hammer, arXiv:1202.3665 [astro-ph.IM]

[2] Hogg, D. W., Bovy, J., & Lang, D., 2010, Data analysis recipes: Fitting a model to data, arXiv:1008.4686 [astro-ph.IM]