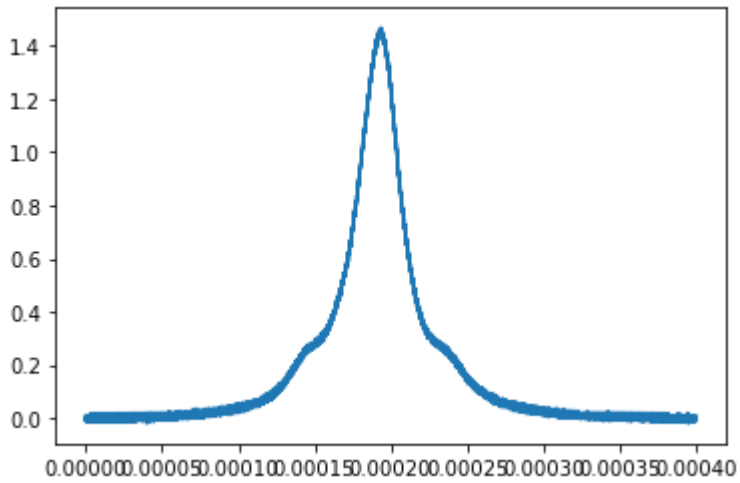
PHYS 512 PS4 Results

(Sorry, I didn't know we had to submit PDFs if we were using Jupyter notebooks until my PS3 was graded, so that's why this is over a week late)

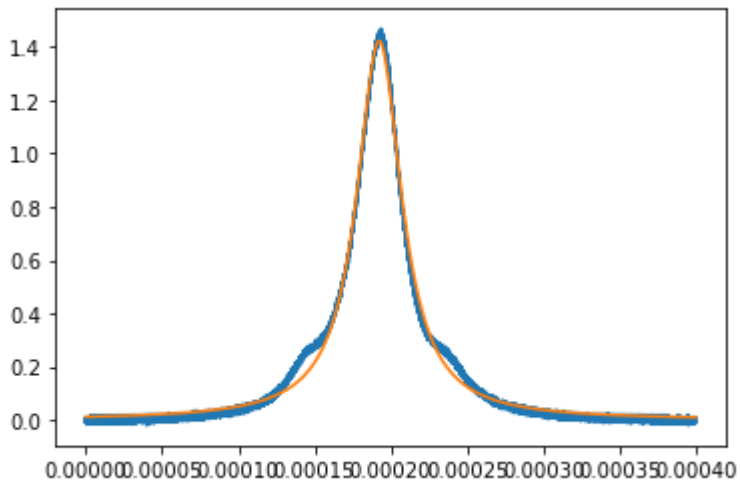
a) Model the data as a Lorentzian

Raw data looks like this:



We can model this as a Lorentzian: $y = \frac{a}{1+(t-t_0)^2/w^2}$ using analytic partial derivatives and the inital parameters as: `p0 = np.array([1.4,0.0002,0.0001])` in order a,t0,w

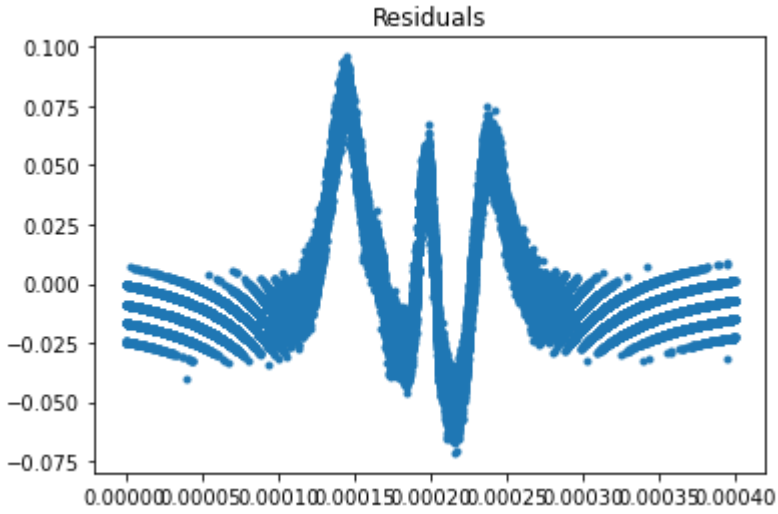
This gives us the following fit:



Best fit parameters are: a = 1.422810680632242, t0 = 0.0001923586493756531, and w = 1.792369079401261e-05

b) Model the noise in the parameters

Our residuals from the above fit look like this:



Estimated noise is $\sum r^2$, and we use the RMS of this value as our estimate for noise. It's hard to tell if the residuals are centered exactly at zero and they have some apparent structure, so we won't use standard deviation.

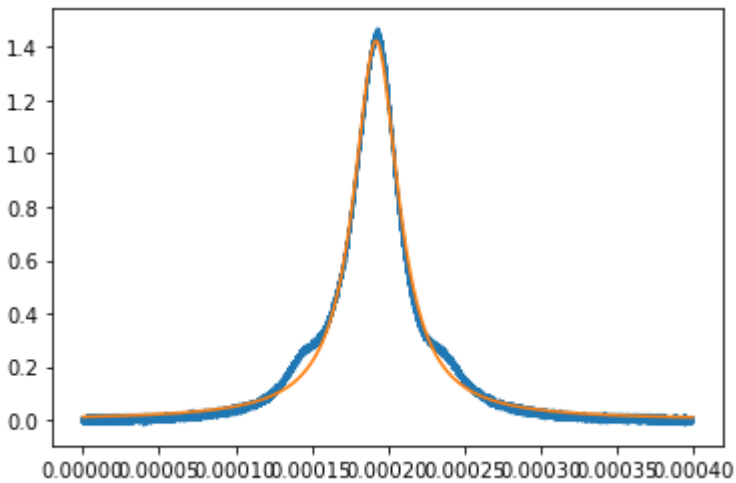
The parameter errors are calculated from $\sqrt{diag(cov)}$, where cov is the covariance matrix.

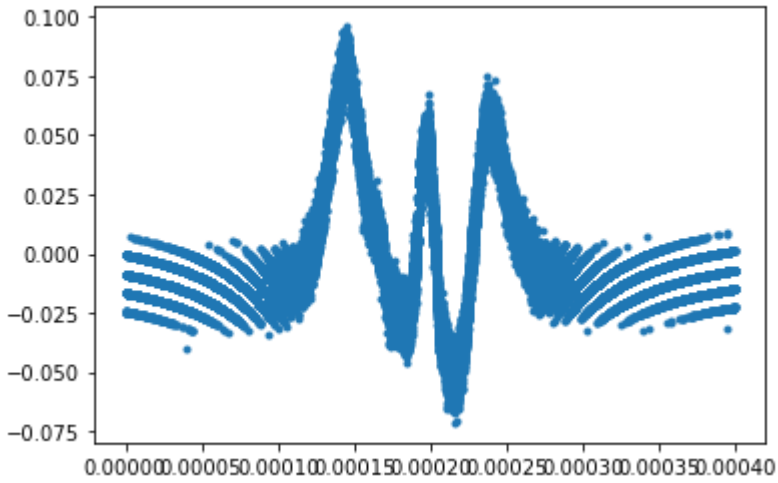
This gives us:

Chi^2 : 63.672662307173326
Noise in data is estimated to be 0.025233442552924347
Error in parameters: a = 0.016861712189248406, t_0 = 2.1235095259315283e-07, w = 3.007158943456581e-07

c) Use numerical derivatives

Fitting the same Lorentzian to the data with numerical derivatives (from `scipy.misc.derivative`), I get the following:





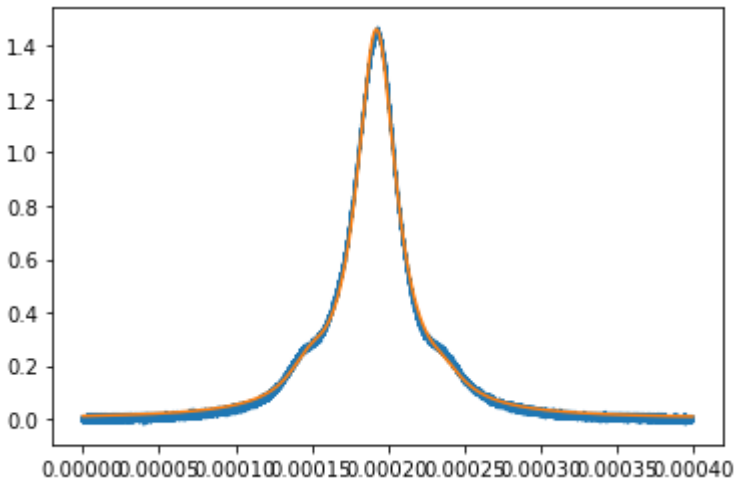
Estimated noise is: 0.0252334425529793
Chi^2 = 63.67266230745065
Error in parameters: a = 0.016861711612780564, t_0 = 2.12351036014805e-07, w = 3.00715989438103e-07

We can see that the differnce between the parameters fit by Newton's method with analytical derivaties and fit with numerical derivatives are all at least 6 orders of magnitude smaller than the uncertainties on the parameters fit by analytical derivatives, so there is statistically no difference between the two methods.

d) Model as the sum of 3 Lorentzians

Initial parameters are now `[1.4, 0.00019, 1.792e-5, 0.2, 0.2, 2e-5]`, in order a,t0,w,b,c,dt. The guesses from a,t0,w are from the best-fit above.

The fit now looks like:



Best fit parameters are:
a = 1.4429923932750095
t0 = 0.00019257852173625134
w = 1.606510942360116e-05
b = 0.10391077999606547

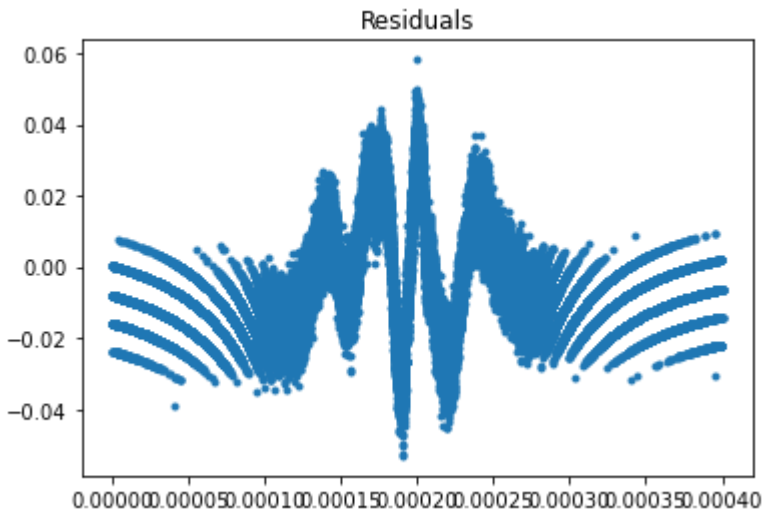
c = 0.06473253001386294
dt = 4.456716295426202e-05

From error estimations, we get:
Chi^2 = 21.247274184334692
Estimated noise is: 0.014576444760069135

Parameter uncertainties are:
a = 0.018278031813332398,
t0 = 2.1640415984078534e-07
w = 3.8756145689578853e-07
b = 0.0174333849558119
c = 0.017070234824102536
dt = 2.6087882033056806e-06

e) Look at residuals and determine if error bars are to be believed

The residuals do not look like white noise and clearly have some structure, which means that there must be correlation between the parameters. This means that the error bars calculated above are not a complete description of the data, and we need to take correlation into account.



f) Generate some realizations for the parameter errors using the full covariance matrix $A^T N^{-1} A$ from d)

Up until now we've been assuming the noise is uncorrelated, so $N = \mathbb{I}$. Now we need to figure out what the correlation between parameters is, in order to get a better estimation of our parameter errors.

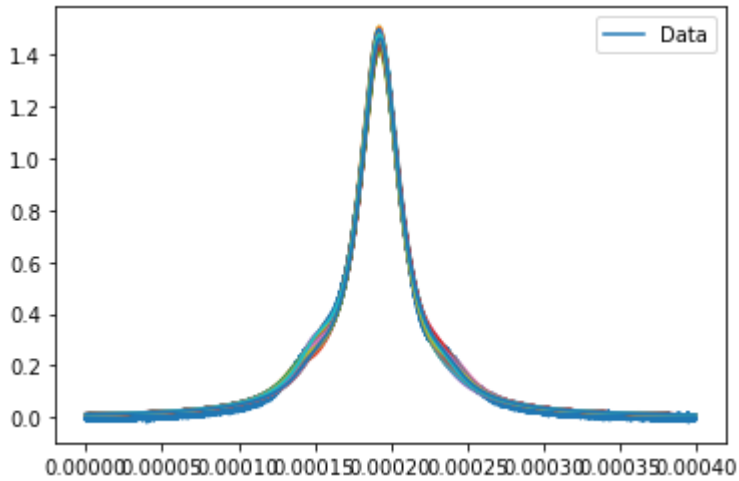
To do this, we'll plot the 3-peak Lorentzian for parameters offset from our best fit by a value picked from a Gaussian distribution of the full distribution of the parameter covariance matrix, $A^T N^{-1} A$. Note that the covariance matrix is just the inverse of `lhs` in our code above.

Over many realizations, $< dd^T >$, where d is the correlated data, should converge to the noise matrix, N, which is unknown to us. This means that if we only work with correlated data, we can get a better estimate of χ^2, since this will include N in our covariance matrix.

The correlated data, d, is generated by evaluating the model (3-peak Lorentzian) at offset parameters. This offset is obtained by sampling the collective set of parameters from the full distribution given by the parameter covariance matrix (this is done by `np.random.multivariate_normal(mean,cov)`, where `mean` is the best-fit parameters, and `cov` is the covariance matrix we got from our best-fit, assuming $N = \mathbb{I}$).

Here, I compute `n_real` = 100 realizations, compute χ^2, and then average this over `n_real`.

I get $\chi^2_{random} > \chi^2_{best-fit}$. This doesn't make sense, because now we are taking into account the correlation between parameters (which is like calculating this with a noise matrix instead of the identity matrix), so we should, on average, get a better result for χ^2 but even averaging over 100 realizations (I also tried 1000 and got the same result), we don't get any improvement over the best-fit χ^2.



Best-fit chi^2: 21.247274184334692
Average chi^2 of 100 realizations 27.970001867120505
Difference in chi^2 = -6.722727682785813

g) Repeat d) using an MCMC

The `mcmc` funtion is adapted from Jon's mcmc_class.py (from 2021 github). I use a helper function `lorentz_chisq`, that calculates χ^2 by $\sum r^2$.

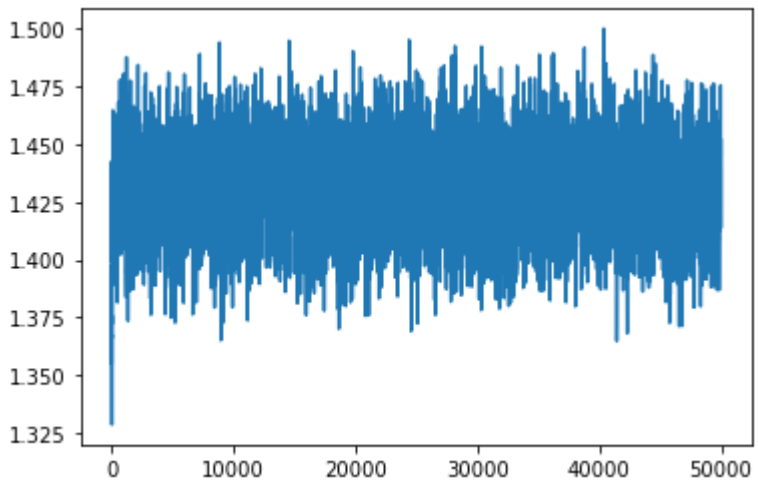
The function takes a step size of `np.random.multivariate_normal(np.zeros(len(pars)),cov)` (from above, but parameters are all set to 0 so we get the amount of parameter shift) to ensure that we are taking statistically relevant steps along the directions of parameter space that match the correlation between our parameters.

This should be multiplied by a scaling function to get shifts for each parameter. I played around with this a bit and found that a scale factor of 1 converged quickly and accurately.

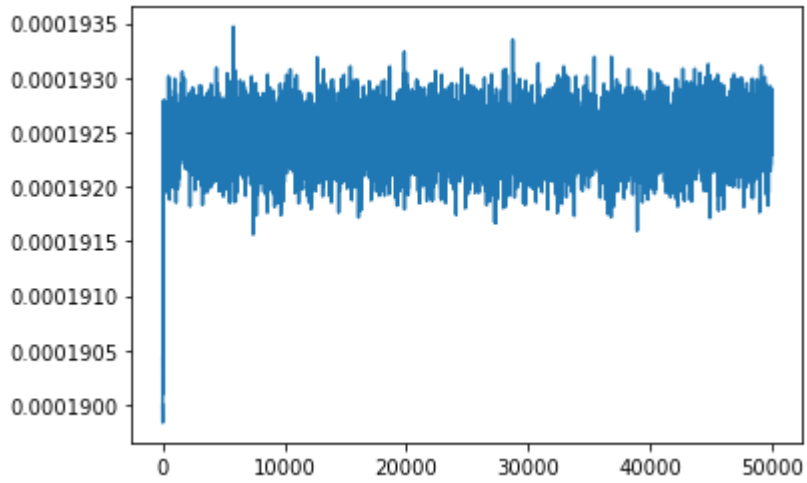
My MCMC accepts 27.5% of events, so based on the rule of thumb that it should accept 25% of events, it seems that I am stopping at a reasonable number of steps.

Below is each parameter chain plotted - these all look like white noise after burn-in:

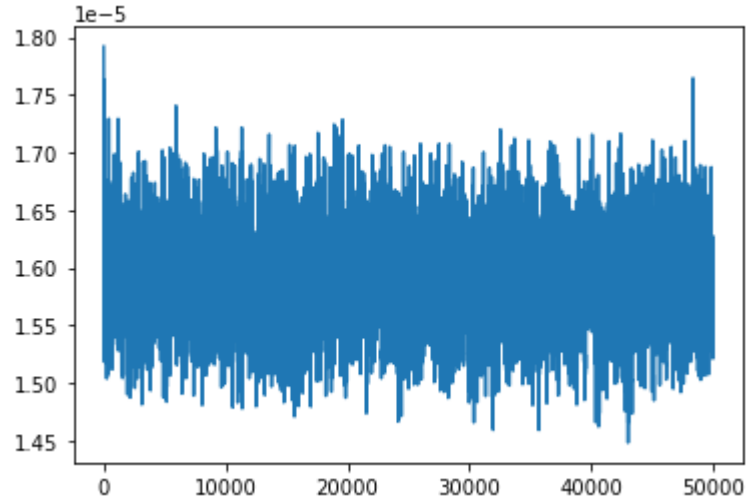
a:



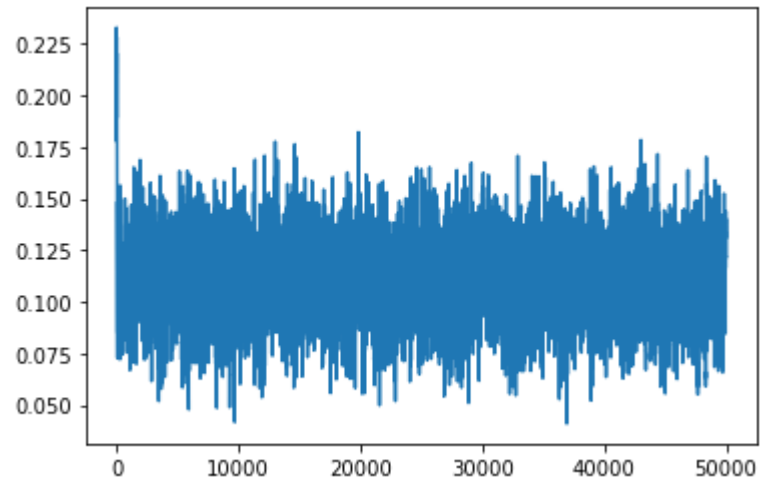
t0:



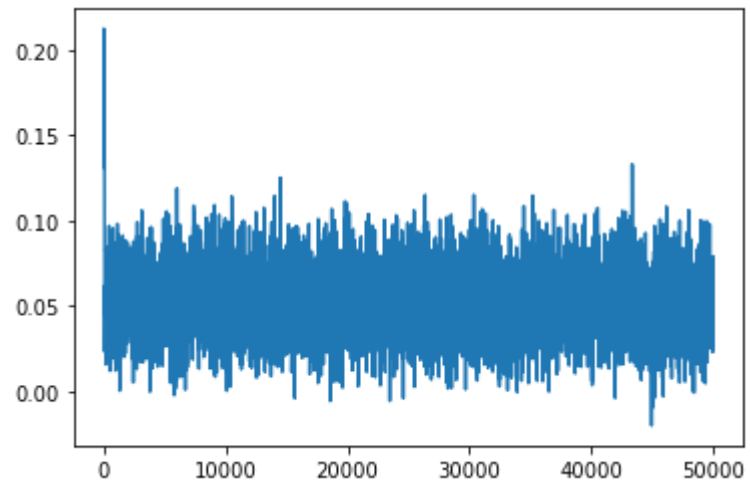
w:


1e−5
1.80
1.75
1.70
1.65
1.60
1.55
1.50
1.45
0
10000
20000
30000
40000
50000

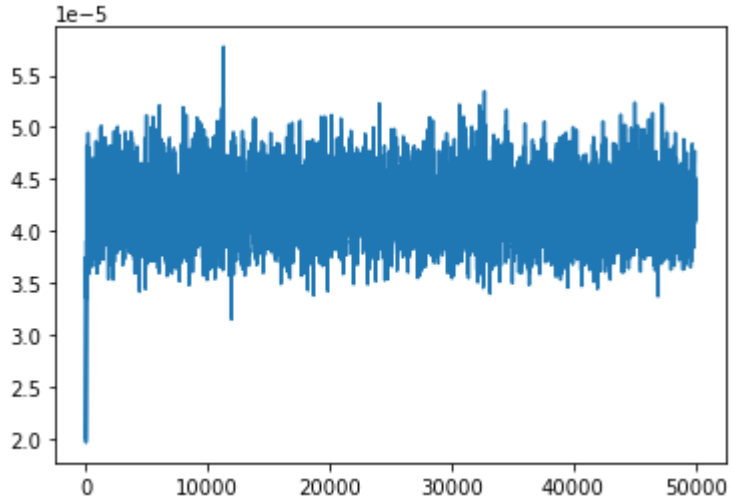
b:


0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
10000
20000
30000
40000
50000

c:


0.20
0.15
0.10
0.05
0.00
0
10000
20000
30000
40000
50000

dt:

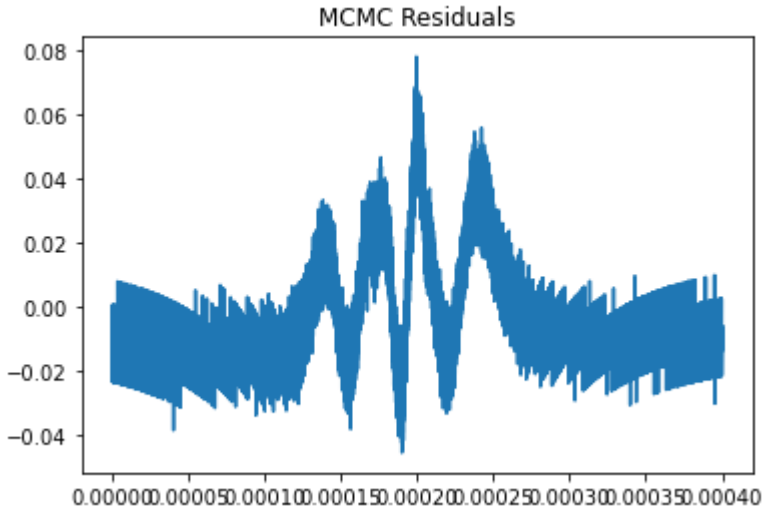


Parameters from MCMC are:
a = 1.4314502801486813
t0 = 0.000192449868049886
w = 1.5940346557459447e-05
b = 0.11008700277812004
c = 0.05225978990574879
dx = 4.258264794472378e-05

Parameter Errors from MCMC are:
Δa = 0.018181944297012365
Δt0 = 2.2128484250698643e-07
Δw = 4.0252123054867363e-07
Δb = 0.01843719132842037
Δc = 0.01802887910932404
Δdx = 2.6972209672020797e-06

Difference in parameter errors compared to Newton's method::
a = 9.608751632003351e-05
t0 = -4.880682666201088e-09
w = -1.49597736528851e-08
b = -0.0010038063726084678
c = -0.0009586442852215042
dx = -8.843276389639911e-08

Noise in MCMC fit is: 0.016082682352244814



h) Since $dx \rightarrow$ 9 GHz, what is the width of the cavity resonance, w, in GHz?

Using the output from the MCMC, we can apply the following formula to get the width of cavity resonance:

$w_{res} = w_{MCMC} \times \frac{dx_{actual}}{dx_{MCMC}}$, where both dx values should be expressed in units of time (seconds):

9 GHz = 1.1×10^{-10} s

To calculate the error, we can use the standard deviation:
$\Delta w_{res} = w_{res} \sqrt{(\frac{\Delta w}{w})^2 + (\frac{\Delta dx}{dx})^2}$

The error seems a little large (I'd expect it to be at least an OOM smaller than the width, since the uncertainties are pretty small), but I can't figure out what's going on.

Width of cavity resonance = 24.04237763094123 +/- 352.5854356727813 GHz