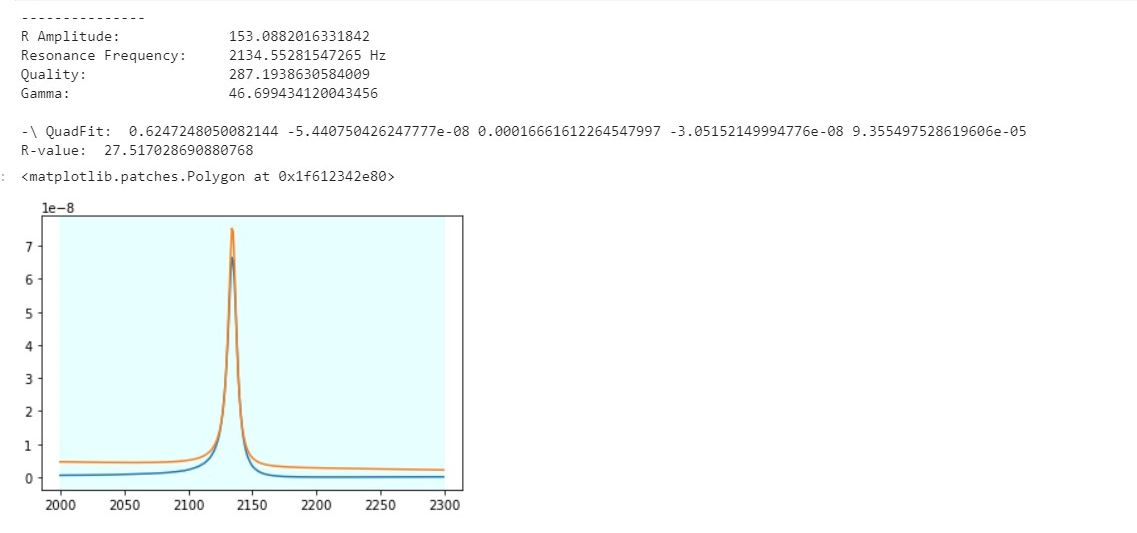
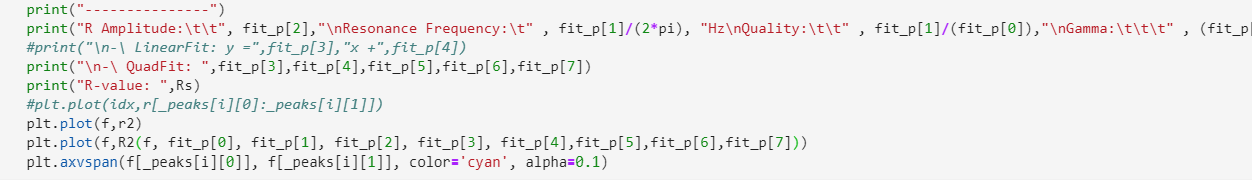
With x = A sin theta + E cos theta, y = A cos theta – E sin theta, and x0, y0 linear , so:

* Note produces a quadratic background
* The terms allow absorptive/elastic-like behavior in , so there may be asymmetric peaks in .

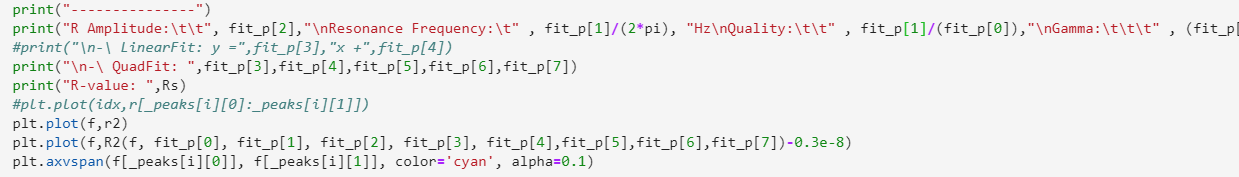
Fitting the above using gradient descent on parameters\*(since Scipy.optimize.curveFit fails with 8 parameters):

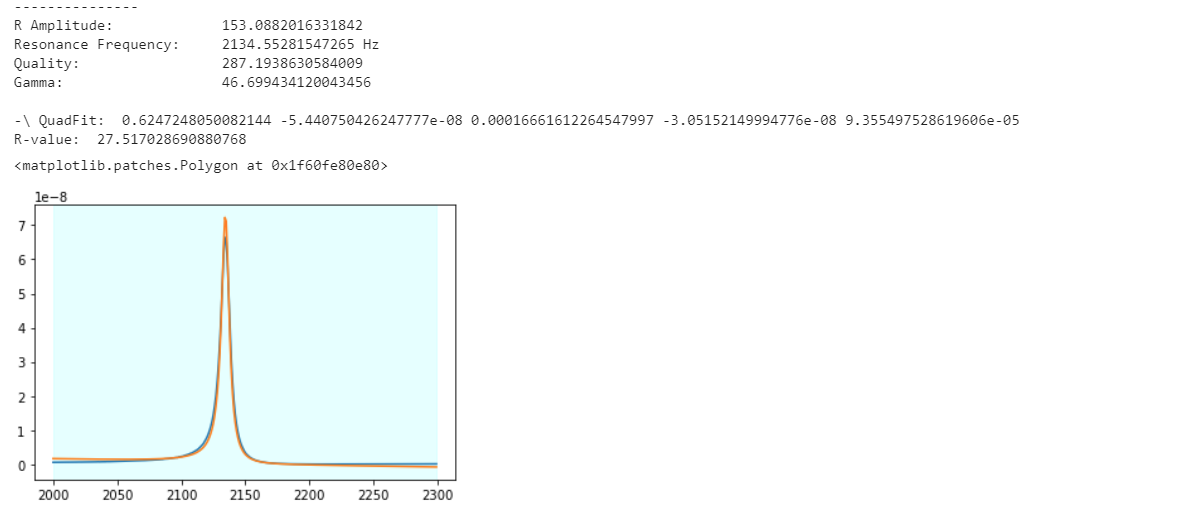
EX: Run14\_40



* There is clearly some unaccounted background, and the fit(yellow) seems shifted upward.

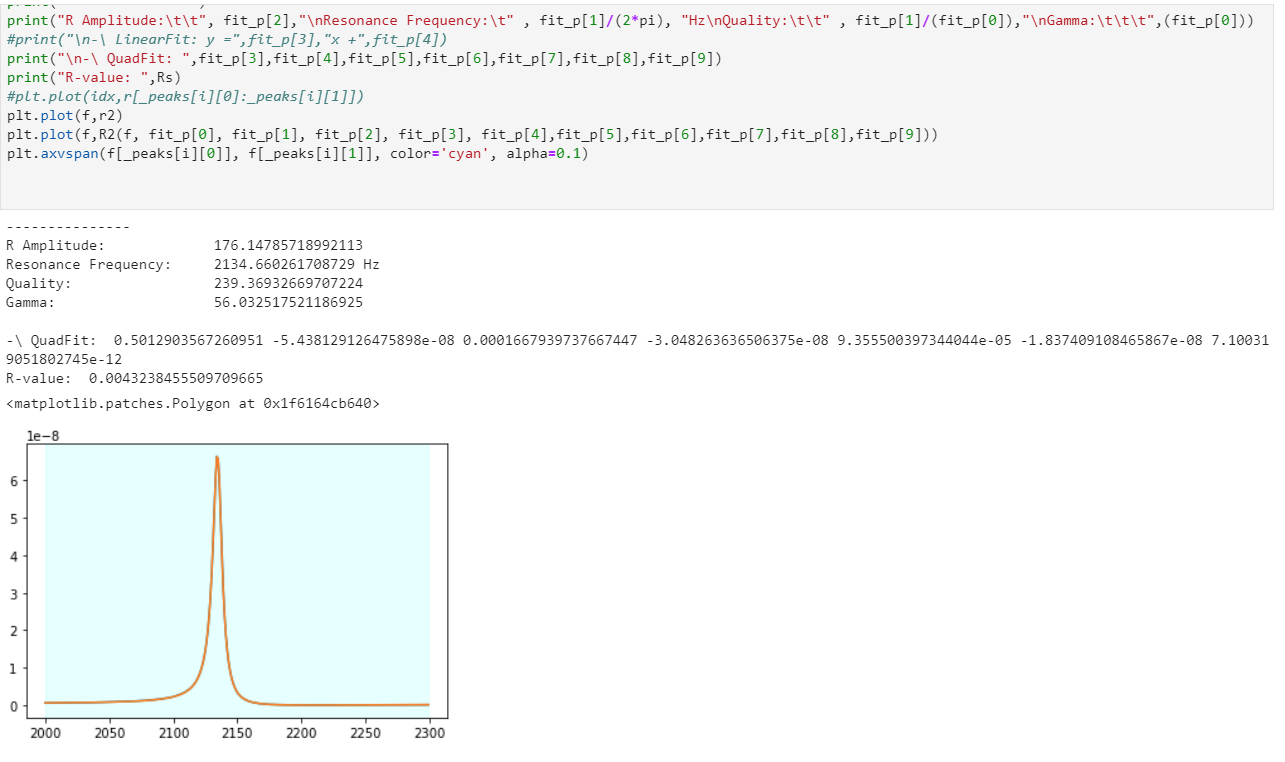
Trying with a downward shift of 0.3\*10^-8

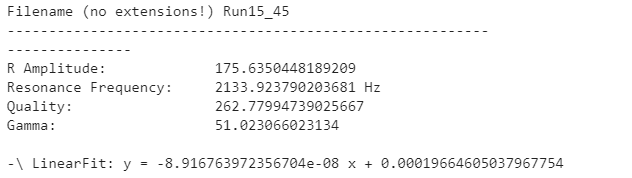
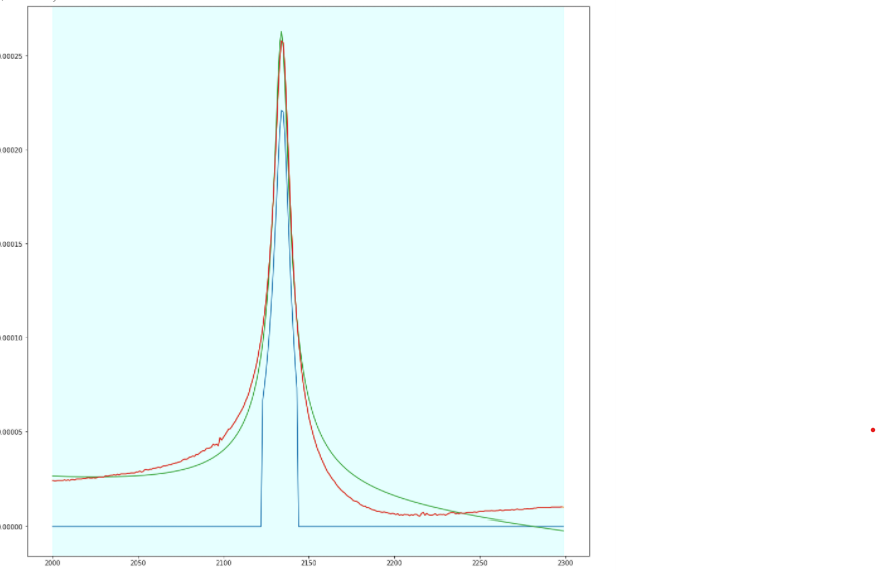




* Much nicer – note the fit parameters do not change, since we have not re-fitted the curve.

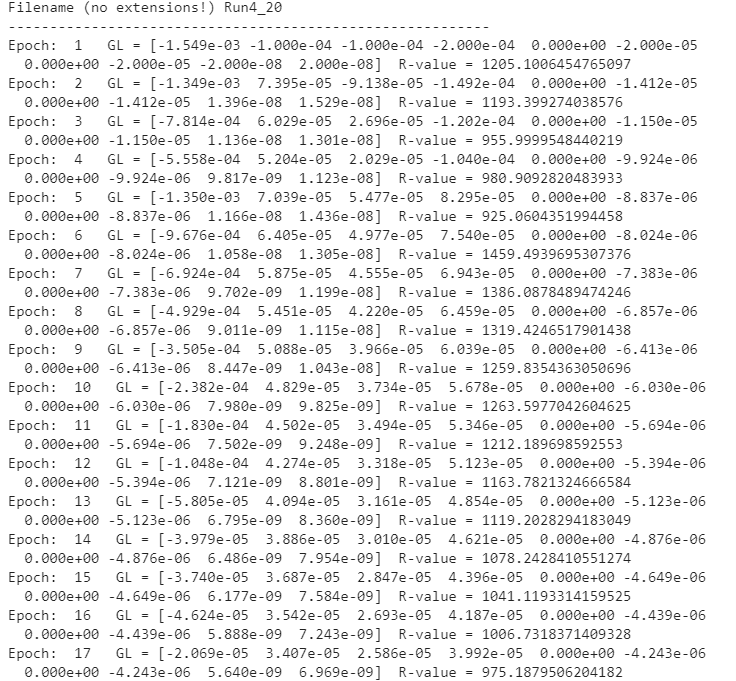
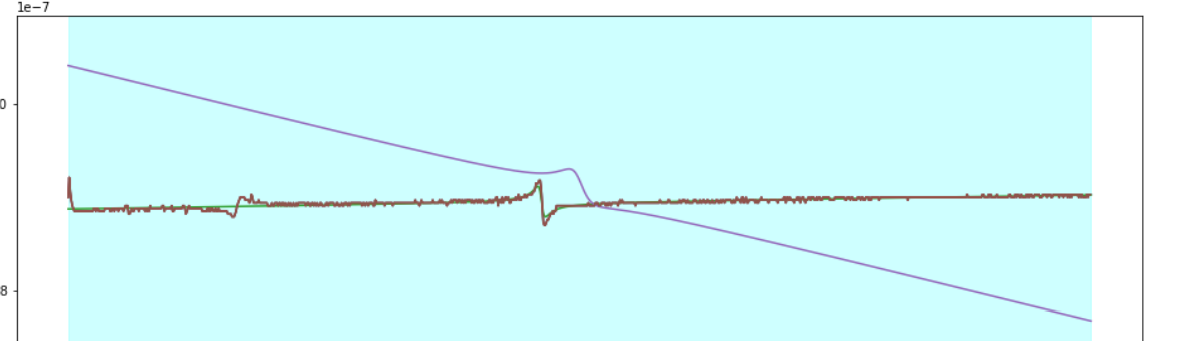
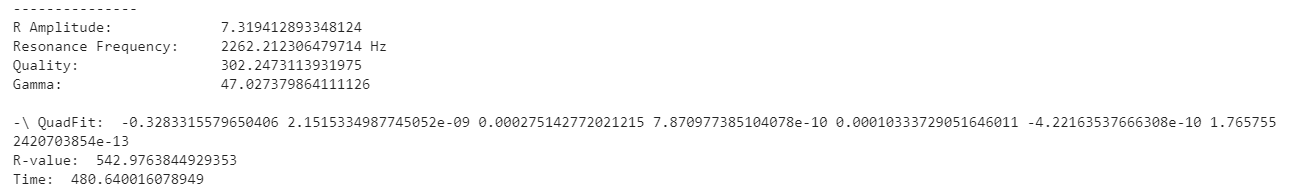
Adding an additional linear background, to account for the downward shift and mild deviation at the ends of the graph:



* Note there are two plots (superimposed)!
* This also agrees with the old Radial fit (without including x0, y0) I did previously – which is shown below:
* 
* Run15\_45 and Run 14\_40 were taken on the same setup, but with different phase angles.
* Note that this old fitting method incorrectly calculates background – there is a large deviation at the ends of the plot, but is much faster as it does not use gradient descent!

The main drawback of using gradient descent is time! For this example, since the R^2 curve is mainly dependent on A^2 and B^2 - there is minimal background, the fit is quite fast (a few seconds).

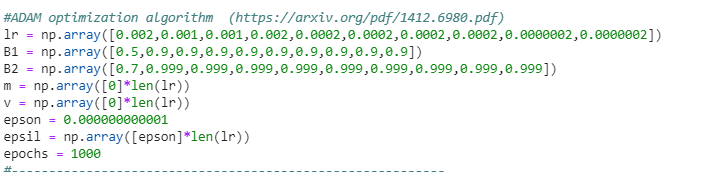
For: Run4\_20

* The initial R-value after the first epoch is ~1205, while with Run14\_40 this was already near 0.004.
* 
* As can be seen, the model takes a long time to converge – possibly due to unoptimized learning rates, which at present are set to minimal values (so as to work for any possible run).
* After 901 seconds (~15min) :
* 
* Which seems like a horrible fit, but note that there are two fits in this plot (green and purple)
* 
* The green fit is okay, but would be much better with more time to converge.
* The final green values: (note the R-squared-value is still quite large!)
* 
* Note that the green fit only took 480 seconds, while the purple fit (probably due to incorrect peak detection) took another 421 seconds for 1000 epochs each.
* Also note that the old radial fit method cannot fit this curve, since this is an asymmetric peak in R^2 (as mentioned previously), which would not occur if not for a large background.

So while this method is slow, it seems to eventually fit all runs.

Gradient descent implementation:



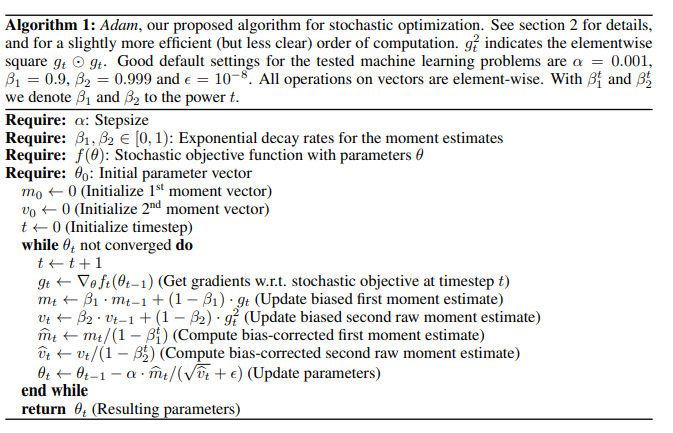


Note that I am using two fits to calculate Rsquared from and initial parameter guess (p02). The bounding function simply keeps this p03 within the bounds, and the epoch loops do the optimization.

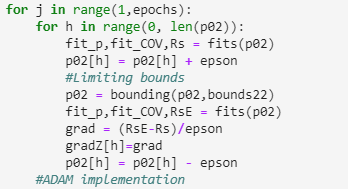
Inside the loop, I calculate the gradient of Rsquared (w.r.t the parameters), using “epson" as an exceedingly small value (epsilon). Then every epoch, I normally would change p02 by – gradient \* learning rate.

There is a slightly more complicated change here (not just negative gradient \* learning rate) – as described in this paper (the ADAM optimization algorithm) : (<https://arxiv.org/pdf/1412.6980.pdf>)

I have also noted it below for your reference (quote from above paper).



All this fitting contributes to delay – 4 fits every derivative (since I use a pair of fits in the fit function), and 10 derivatives to calculate – so 40 fits per epoch!

So in 1000 epochs, ~40000 fits are being done! Note that I can optimize this somewhat, by moving the initial value fit outside the derivative loop, but have not done this yet. 

Finally, the R-square-value function:

