

# PHYS-512, PS4

Muath Hamidi

October, 2022

## Problem 1

**a.**

The Lorentzian is

$$d = \frac{a}{1 + (t - t_0)^2/\omega^2} \quad (1)$$

Now, its derivatives in  $a$ ,  $t_0$ ,  $\omega$  are

$$\frac{\partial d}{\partial a} = \frac{1}{1 + (t - t_0)^2/\omega^2} \quad (2)$$

$$\frac{\partial d}{\partial t_0} = \frac{2a(t - t_0)}{\omega^2(1 + (t - t_0)^2/\omega^2)^2} \quad (3)$$

$$\frac{\partial d}{\partial \omega} = \frac{2a(t - t_0)^2}{\omega^3(1 + (t - t_0)^2/\omega^2)^2} \quad (4)$$

I will write the full code then give the results for each part. So, the code is:  
Use the following code:

```
1 #=====
2 # Course: PHYS 512
3 # Problem: PS4 P1
4 #=====
5 # By: Muath Hamidi
6 # Email: muath.hamidi@mail.mcgill.ca
7 # Department of Physics, McGill University
8 # September 2022
9
10 #=====
11 # Libraries
12 #=====
13 import numpy as np # For math
14 import matplotlib.pyplot as plt # For graphs
15
16 #=====
17 # Loading Data
18 #=====
19 stuff = np.load('sidebands.npz')
20 t = stuff['time']
21 d = stuff['signal']
22
23 #=====
24 #=====
```

```

25 # Part A
26 #=====
27 #=====
28 # Here you can find the codes related to part (a).
29 print("=====")
30 print("Part (a)")
31 print("=====")
32
33 #=====
34 # Lorentzian
35 #=====
36 def lorentz(t, a, t0, w):
37     y = a / (1 + ((t-t0)/w)**2)
38     return y
39
40 #=====
41 # Newton Method
42 #=====
43 def calc_lorentz(p, t):
44     # Parameters
45     a = p[0]
46     t0 = p[1]
47     w = p[2]
48
49     # Lorentzian
50     y = lorentz(t, a, t0, w)
51
52     grad = np.zeros([t.size, p.size])
53
54     # Differentiate w.r.t. all the parameters
55     grad[:,0] = (1 + (t - t0)**2 / w**2)**(-1)
56     grad[:,1] = (2 * a * (t - t0)) / (w**2 * (1 + (t - t0)**2 / w**2)**(2))
57     grad[:,2] = (2 * a * (t - t0)**2) / (w**3 * (1 + (t - t0)**2 / w**2)**(2))
58
59     return y, grad
60
61
62 p0 = np.array([1.4,0.0002,0.00002]) #starting guess, close but not exact
63 p = p0.copy()
64
65 for j in range(15):
66     pred, grad = calc_lorentz(p, t)
67     r = d - pred
68     err = (r**2).sum()
69     r = np.matrix(r).transpose()
70     grad = np.matrix(grad)
71
72     lhs = grad.transpose()*grad
73     rhs = grad.transpose()*r
74     dp = np.linalg.inv(lhs)*(rhs)
75     for jj in range(p.size):
76         p[jj] = p[jj] + dp[jj]
77     print("The parameters (a, t0, w):", p)
78
79
80 print("Best-fit parameters (a, t0, w):", p)

```

```

81
82
83 # Data
84 plt.ion()
85 plt.clf()
86 plt.scatter(t, d, c="blue", s=0.2, label="Data")
87
88 # Calculated
89 plt.plot(t, pred, c="red", label="Function")
90 plt.title("d vs t (Newton)")
91 plt.ylabel("$d$")
92 plt.xlabel("$t$")
93 plt.legend()
94 plt.savefig('4.1.1.pdf', format='pdf', dpi=1200)
95 plt.show()
96 plt.close()
97
98
99 #=====
100 #=====
101 # Part B
102 #=====
103 #=====
104 # Here you can find the codes related to part (b).
105 print("=====")
106 print("Part (b)")
107 print("=====")
108
109 #=====
110 # Noise & Errors
111 #=====
112 Noise = np.mean((d - pred)**2)
113 Errors = np.sqrt(Noise * np.diag(np.linalg.inv(grad.T@grad)))
114 print("Noise:", Noise)
115 print("Errors in (a, t0, w):", Errors)
116
117
118 #=====
119 #=====
120 # Part C
121 #=====
122 #=====
123 # Here you can find the codes related to part (c).
124 print("=====")
125 print("Part (c)")
126 print("=====")
127
128 #=====
129 # Numerical Differentiator
130 #=====
131 def NDiff(f, x, dx=10**-8): # f:function, x:variable, d:delta
132     NDiff = (8 * (f(x + dx) - f(x - dx)) - f(x + 2*dx) + f(x - 2*dx)) / (12*dx)
133     return NDiff
134
135 #=====
136 # Newton Method

```

```

137 #=====
138 def Grad(p, t, f):
139     a = P[0]
140     t0 = P[1]
141     w = P[2]
142
143     y = lorentz(t, a, t0, w)
144
145     # Derivative
146     Fa = lambda A: f(t, A, t0, w)
147     Ft0 = lambda T0: f(t, a, T0, w)
148     Fw = lambda W: f(t, a, t0, W)
149
150     # Grad
151     Grad_a = NDiff(Fa, a)
152     Grad_t0 = NDiff(Ft0, t0)
153     Grad_w = NDiff(Fw, w)
154
155     return y, np.array([Grad_a, Grad_t0, Grad_w]).transpose()
156
157
158 P = p0.copy()
159
160 for j in range(15):
161     pred, grad = Grad(P, t, lorentz)
162     r = d - pred
163     err = (r**2).sum()
164     r = np.matrix(r).transpose()
165     grad = np.matrix(grad)
166
167     lhs = grad.transpose()*grad
168     rhs = grad.transpose()*r
169     dP = np.linalg.inv(lhs)*(rhs)
170     for jj in range(P.size):
171         P[jj] = P[jj] + dP[jj]
172     print("The parameters (a, t0, w):", P)
173
174
175 print("Best-fit parameters (a, t0, w):", P)
176
177
178 #=====
179 #=====
180 # Part D
181 #=====
182 #=====
183 # Here you can find the codes related to part (d).
184 print("=====")
185 print("Part (d)")
186 print("=====")
187
188 #=====
189 # Lorentzian
190 #=====
191 def lorentz3(t, a, b, c, t0, dt, w):
192     y1 = a / (1 + ((t - t0) / w)**2)

```

```

193     y2 = b / (1 + ((t - t0 + dt) / w)**2)
194     y3 = c / (1 + ((t - t0 - dt) / w)**2)
195     return y1 + y2 + y3
196
197 def Grad3(p, t, f):
198     a, b, c, t0, dt, w = P3
199
200     y = lorentz3(t, a, b, c, t0, dt, w)
201
202     # Derivative
203     Fa = lambda A: f(t, A, b, c, t0, dt, w)
204     Fb = lambda B: f(t, a, B, c, t0, dt, w)
205     Fc = lambda C: f(t, a, b, C, t0, dt, w)
206     Ft0 = lambda T0: f(t, a, b, c, T0, dt, w)
207     Fdt = lambda dT: f(t, a, b, c, t0, dT, w)
208     Fw = lambda W: f(t, a, b, c, t0, dt, W)
209
210     # Grad
211     Grad_a = NDiff(Fa, a)
212     Grad_b = NDiff(Fb, b)
213     Grad_c = NDiff(Fc, c)
214     Grad_t0 = NDiff(Ft0, t0)
215     Grad_dt = NDiff(Fdt, dt)
216     Grad_w = NDiff(Fw, w)
217
218     return y, np.array([Grad_a, Grad_b, Grad_c, Grad_t0, Grad_dt, Grad_w]).
219     transpose()
220
221 p03 = np.array([1.4, 0.1, 0.06, 0.0002, 0.00005, 0.00002])
222 P3 = p03.copy()
223
224 for j in range(15):
225     pred, grad = Grad3(P3, t, lorentz3)
226     r = d - pred
227     err = (r**2).sum()
228     r = np.matrix(r).transpose()
229     grad = np.matrix(grad)
230
231     lhs = grad.transpose()*grad
232     rhs = grad.transpose()*r
233     dP3 = np.linalg.inv(lhs)*(rhs)
234     for jj in range(P3.size):
235         P3[jj] = P3[jj] + dP3[jj]
236     print("The parameters (a, b, c, t0, dt, w):", P3)
237
238
239 print("Best-fit parameters (a, b, c, t0, dt, w):", P3)
240
241
242 # Data
243 plt.ion()
244 plt.clf()
245 plt.scatter(t, d, c="blue", s=0.2, label="Data")
246
247 # Calculated

```

```

248 plt.plot(t, pred, c="red", label="Function")
249 plt.title("d vs t (Newton, Numerical)")
250 plt.ylabel("$d$")
251 plt.xlabel("$t$")
252 plt.legend()
253 plt.savefig('4.1.2.pdf', format='pdf', dpi=1200)
254 plt.show()
255 plt.close()
256
257 #=====
258 # Noise & Errors
259 #=====
260 Noise = np.mean((d - pred)**2)
261 Errors = np.sqrt(Noise * np.diag(np.linalg.inv(grad.T@grad)))
262 print("Noise:", Noise)
263 print("Errors in (a, b, c, t0, dt, w):", Errors)
264
265
266 #=====
267 #=====
268 # Part E
269 #=====
270 #=====
271 # Here you can find the codes related to part (e).
272 print("=====")
273 print("Part (e)")
274 print("=====")
275
276 #=====
277 # Residuals & Residuals Errors
278 #=====
279 Res = pred - d # Residuals
280 ResErrs = Noise # Residuals errors
281
282 plt.plot(t, Res)
283 plt.title("Residuals vs t")
284 plt.ylabel("Residuals")
285 plt.xlabel("$t$")
286 plt.savefig('4.1.3.pdf', format='pdf', dpi=1200)
287 plt.show()
288 plt.close()
289
290
291 #=====
292 #=====
293 # Part F
294 #=====
295 #=====
296 # Here you can find the codes related to part (f).
297 print("=====")
298 print("Part (f)")
299 print("=====")
300
301 #=====
302 # Realizations Generation
303 #=====

```

```

304 Covariance = np.linalg.inv(lhs)
305
306 RealN = 200 # Realizations number
307 pred_Gen = np.zeros((RealN, t.size))
308 for i in range(RealN):
309     P_Gen = np.random.multivariate_normal(P3, Covariance) # Generated P
310     a, b, c, t0, dt, w = P_Gen
311     pred_Gen[i,:] = lorentz3(t, a, b, c, t0, dt, w)
312     plt.plot(t, pred_Gen[i,:])
313
314
315 plt.scatter(t, d, c="blue", s=0.2, label="Data")
316 plt.title("d vs t (Newton, Many fits)")
317 plt.ylabel("$d$")
318 plt.xlabel("$t$")
319 plt.legend()
320 plt.savefig('4.1.4.pdf', format='pdf', dpi=1200)
321 plt.show()
322 plt.close()
323
324 #=====
325 # Xi^2
326 #=====
327 def Xi2(d, pred, Errors):
328     Xi2 = np.sum((pred - d)**2 / Errors**2)
329     return Xi2
330
331 #=====
332 # Xi^2 Generation
333 #=====
334 typical_diff = np.mean([Xi2(d, pred, Noise) - Xi2(d, pred_Gen[i,:], Noise) for i in
335     range(RealN)])
336 print("Typical difference in X^2: {}".format(typical_diff))
337
338 plt.axhline(Xi2(d, pred, Noise), c="r") # Best-Fit X^2
339 for i in range(RealN):
340     plt.scatter(i+1, Xi2(d, pred_Gen[i,:], Noise))
341
342 plt.title("$\chi^2$")
343 plt.ylabel("$\chi^2$")
344 plt.xlabel("index")
345 plt.savefig('4.1.5.pdf', format='pdf', dpi=1200)
346 plt.show()
347 plt.close()
348
349 #=====
350 # Part G
351 #=====
352 #=====
353 # Here you can find the codes related to part (g).
354 print("=====")
355 print("Part (g)")
356 print("=====")
357
358 #=====

```

```

359 # MCMC
360 #=====
361 def get_step(trial_step):
362     return np.random.multivariate_normal(len(trial_step)*[0], trial_step)
363
364
365 iterations = 15000
366
367 def MCMC(t, d, p03, Cov, errs, iterations):
368
369     a, b, c, t0, dt, w = p03 # Initial parameters
370
371     chain = np.zeros((iterations, p03.size))
372     chain[0,:] = a, b, c, t0, dt, w
373
374     pred = lorentz3(t, a, b, c, t0, dt, w)
375     chisq = np.zeros(iterations)
376     chisq[0] = Xi2(d, pred, errs) # Initial  $\chi^2$ 
377
378     # Chain Generation
379     for i in range(1, iterations):
380         ps = chain[i-1,:]
381
382         # Update
383         A, B, C, T0, dT, W = ps + get_step(Cov)
384         prediction = lorentz3(t, A, B, C, T0, dT, W)
385
386         # Acceptable Change
387         Acc = 0.5*(chisq[0] - Xi2(d, prediction, errs))
388
389         if np.log(np.random.rand(1)) < Acc:
390             A, B, C, T0, dT, W = ps + get_step(Cov)
391         else:
392             A, B, C, T0, dT, W = ps
393
394         # Prediction After Update
395         prediction = lorentz3(t, A, B, C, T0, dT, W)
396
397         # Filling The Chains
398         chain[i,:] = A, B, C, T0, dT, W
399         chisq[i] = Xi2(d, prediction, errs)
400
401     return chain, chisq
402
403
404 chain, chisq = MCMC(t, d, p03, Covariance, Noise, iterations)
405
406 p_names = ["a", "b", "c", "t0", "dt" , "w"]
407 for i in range(p03.size):
408     plt.plot(np.arange(iterations), chain[:,i])
409     plt.title("MCMC ({}{}$)".format(p_names[i]))
410     plt.ylabel("{}{}$".format(p_names[i]))
411     plt.xlabel("Iteration")
412     plt.savefig('4.1.{}.pdf'.format(6+i), format='pdf', dpi=1200)
413     plt.show()
414     plt.close()

```



```

415
416 #=====
417 # Error
418 #=====
419 Size = 7500
420 Error = np.std(chain[Size:,:], axis=0)
421 print("Standard Deviation in (a, b, c, t0, dt, w):", Error)
422
423
424 #=====
425 #=====
426 # Part H
427 #=====
428 #=====
429 # Here you can find the codes related to part (h).
430 print("=====")
431 print("Part (h)")
432 print("=====")
433
434 #=====
435 # Width of the Cavity Resonance
436 #=====
437 # Real w
438 w_real = 9 * chain[-1,:][5] / chain[-1,:][4]
439
440 print("The actual width of the cavity resonance: {} GHz".format(w_real))

```

The parameters:

The parameters (a, t0, w): [1.25114351e+00 1.92158337e-04 2.11178874e-05]  
The parameters (a, t0, w): [1.40733307e+00 1.92295568e-04 1.77182907e-05]  
The parameters (a, t0, w): [1.42302366e+00 1.92367162e-04 1.79192751e-05]  
The parameters (a, t0, w): [1.42284083e+00 1.92358564e-04 1.79229277e-05]  
The parameters (a, t0, w): [1.42281207e+00 1.92358675e-04 1.79236557e-05]  
The parameters (a, t0, w): [1.42281082e+00 1.92358650e-04 1.79236873e-05]  
The parameters (a, t0, w): [1.42281069e+00 1.92358649e-04 1.79236906e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
Best-fit parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]

Plot d vs t (Newton) fig[1]:

**b.**

The noise and error:

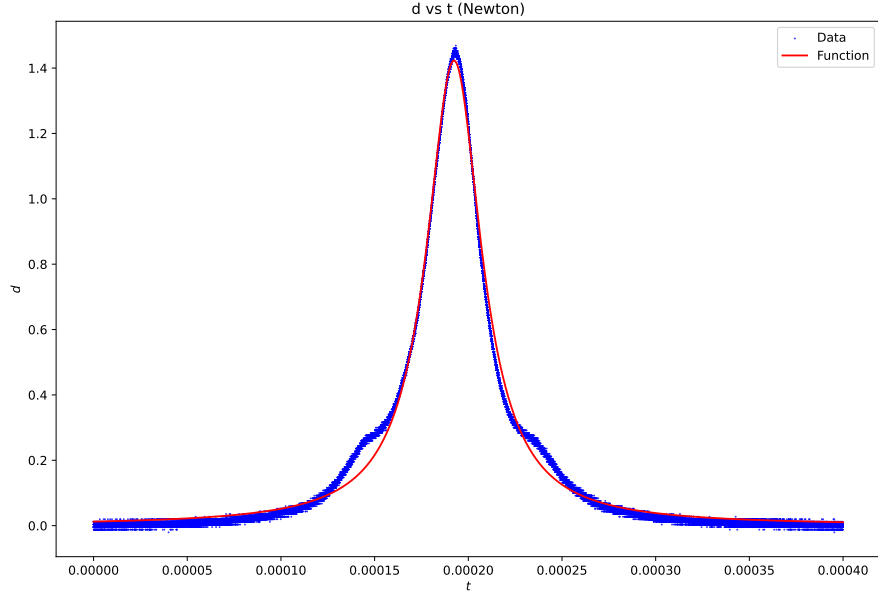


Figure 1: d vs t (Newton).

Noise: 0.0006367266230717332

Errors in (a, t0, w): [4.25479046e-04 5.35834556e-09 7.58809724e-09]

**c.**

The parameters:

The parameters (a, t0, w): [1.25114351e+00 1.92158337e-04 2.11178874e-05]  
The parameters (a, t0, w): [1.40733307e+00 1.92295568e-04 1.77182907e-05]  
The parameters (a, t0, w): [1.42302366e+00 1.92367162e-04 1.79192751e-05]  
The parameters (a, t0, w): [1.42284083e+00 1.92358564e-04 1.79229277e-05]  
The parameters (a, t0, w): [1.42281207e+00 1.92358675e-04 1.79236557e-05]  
The parameters (a, t0, w): [1.42281082e+00 1.92358650e-04 1.79236873e-05]  
The parameters (a, t0, w): [1.42281069e+00 1.92358649e-04 1.79236906e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
The parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]  
Best-fit parameters (a, t0, w): [1.42281068e+00 1.92358649e-04 1.79236908e-05]

My answers statistically are not significantly different from my answers in (a).

**d.**

Using the Lorentzian

$$d = \frac{a}{1 + (t - t_0)^2/\omega^2} + \frac{b}{1 + (t - t_0)^2/\omega^2} + \frac{c}{1 + (t - t_0)^2/\omega^2} \quad (5)$$

The parameters:

The parameters (a, b, c, t0, dt, w): [1.25461846e+00 3.36646157e-02 3.53983470e-02 1.91977899e-04 4.25700696e-05 2.00105634e-05]

The parameters (a, b, c, t0, dt, w): [1.42640089e+00 1.23044956e-01 7.91154570e-02 1.92720149e-04 4.59123984e-05 1.47389189e-05]

The parameters (a, b, c, t0, dt, w): [1.43966998e+00 1.05139454e-01 6.63871778e-02 1.92613733e-04 4.48108928e-05 1.60560110e-05]

The parameters (a, b, c, t0, dt, w): [1.44311775e+00 1.04065418e-01 6.50566095e-02 1.92578529e-04 4.45265037e-05 1.60576432e-05]

The parameters (a, b, c, t0, dt, w): [1.44298320e+00 1.03895432e-01 6.46805599e-02 1.92578887e-04 4.45800193e-05 1.60662053e-05]

The parameters (a, b, c, t0, dt, w): [1.44299530e+00 1.03915695e-01 6.47482796e-02 1.92578447e-04 4.45637145e-05 1.60647878e-05]

The parameters (a, b, c, t0, dt, w): [1.44299166e+00 1.03909499e-01 6.47283099e-02 1.92578543e-04 4.45681048e-05 1.60651951e-05]

The parameters (a, b, c, t0, dt, w): [1.44299260e+00 1.03911133e-01 6.47336797e-02 1.92578516e-04 4.45669072e-05 1.60650860e-05]

The parameters (a, b, c, t0, dt, w): [1.44299234e+00 1.03910687e-01 6.47322161e-02 1.92578523e-04 4.45672331e-05 1.60651157e-05]

The parameters (a, b, c, t0, dt, w): [1.44299241e+00 1.03910808e-01 6.47326144e-02 1.92578521e-04 4.45671444e-05 1.60651077e-05]

The parameters (a, b, c, t0, dt, w): [1.44299239e+00 1.03910775e-01 6.47325060e-02 1.92578522e-04 4.45671686e-05 1.60651099e-05]

The parameters (a, b, c, t0, dt, w): [1.44299240e+00 1.03910784e-01 6.47325355e-02 1.92578522e-04 4.45671620e-05 1.60651093e-05]

The parameters (a, b, c, t0, dt, w): [1.44299239e+00 1.03910782e-01 6.47325275e-02 1.92578522e-04 4.45671638e-05 1.60651094e-05]

The parameters (a, b, c, t0, dt, w): [1.44299240e+00 1.03910783e-01 6.47325297e-02 1.92578522e-04 4.45671633e-05 1.60651094e-05]

The parameters (a, b, c, t0, dt, w): [1.44299240e+00 1.03910782e-01 6.47325291e-02 1.92578522e-04 4.45671634e-05 1.60651094e-05]

Best-fit parameters (a, b, c, t0, dt, w): [1.44299240e+00 1.03910782e-01 6.47325291e-02 1.92578522e-04 4.45671634e-05 1.60651094e-05]

The noise and error:

Noise: 0.00021247274184334357

Errors in (a, b, c, t0, dt, w): [2.66428695e-04 2.54116769e-04 2.48823333e-04 3.15440252e-09

3.80268481e-08 5.64926769e-09]

Plot d vs t (Newton, Numerical) fig[2]:

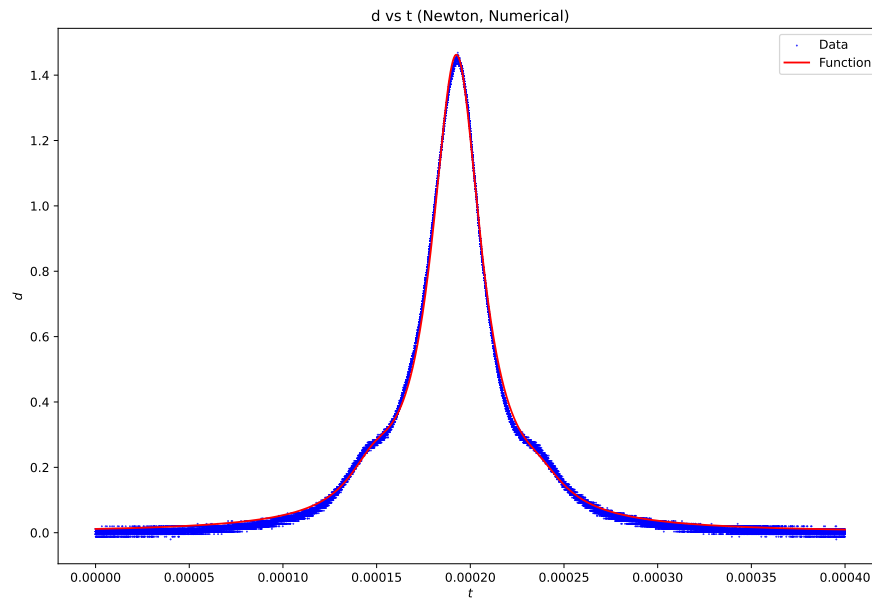


Figure 2: d vs t (Newton, Numerical).

**e.**

The residuals indicates that the model is not a complete description of the data since we have some kind of perturbation. The residuals shown in fig[3]:

**f.**

Generate some some realizations fig[4]:

The  $\chi^2$ 's fig[5]:

Typical difference in  $\chi^2$ : -146745989.3838729

Which is expected, since the best-fit  $\chi^2$  should have the lowest value.

**g.**

Our code finds the chains for all parameters. However, looking at  $t_0$  for example fig[6]:

Which is clearly converges.

Standard Deviation in (a, b, c, t0, dt, w): [4.05058327e-04 6.14927423e-05 8.73150872e-03 2.66352993e-09 8.34320502e-07 1.69070395e-07]

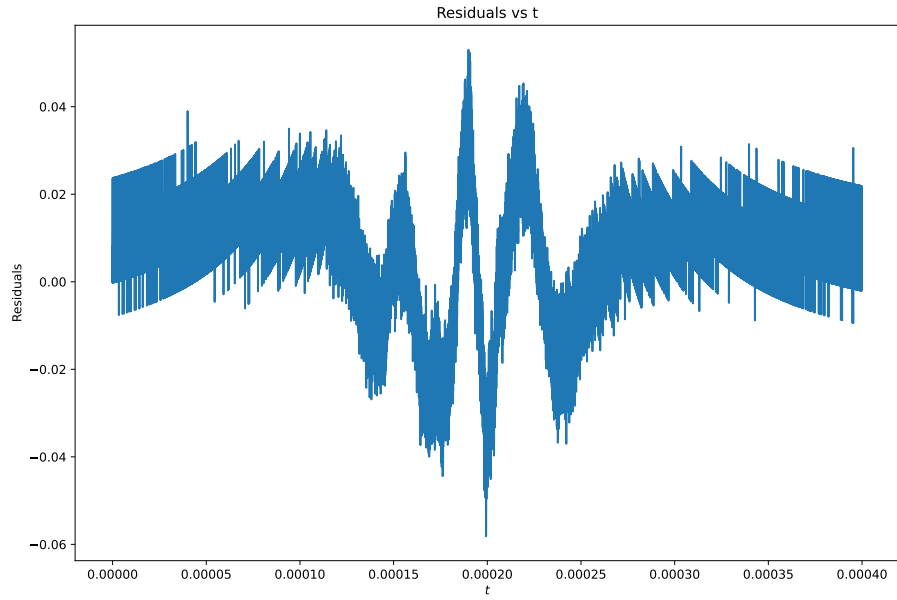


Figure 3: Residuals.

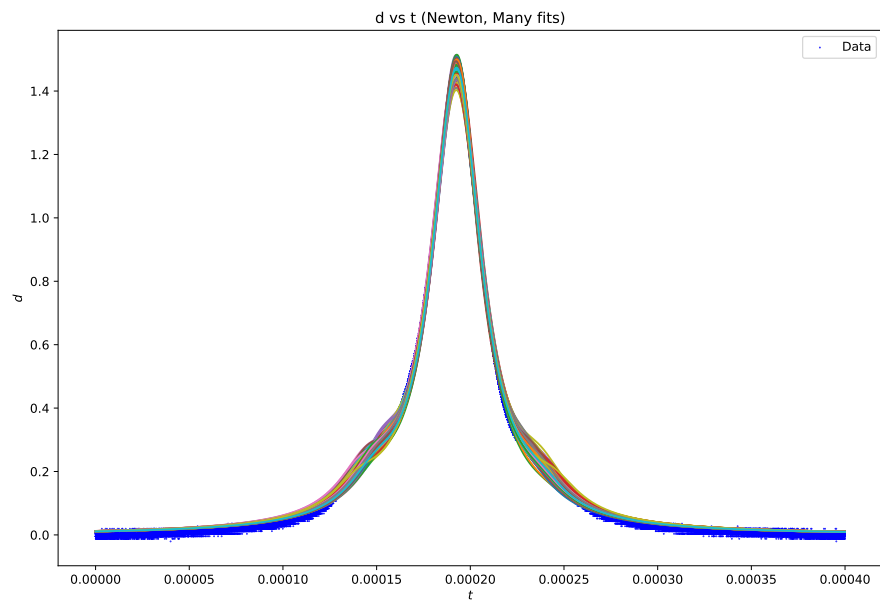


Figure 4: d vs t (Newton, Many Fits).

**h.**

The actual width of the cavity resonance: 4.46638285825994 GHz

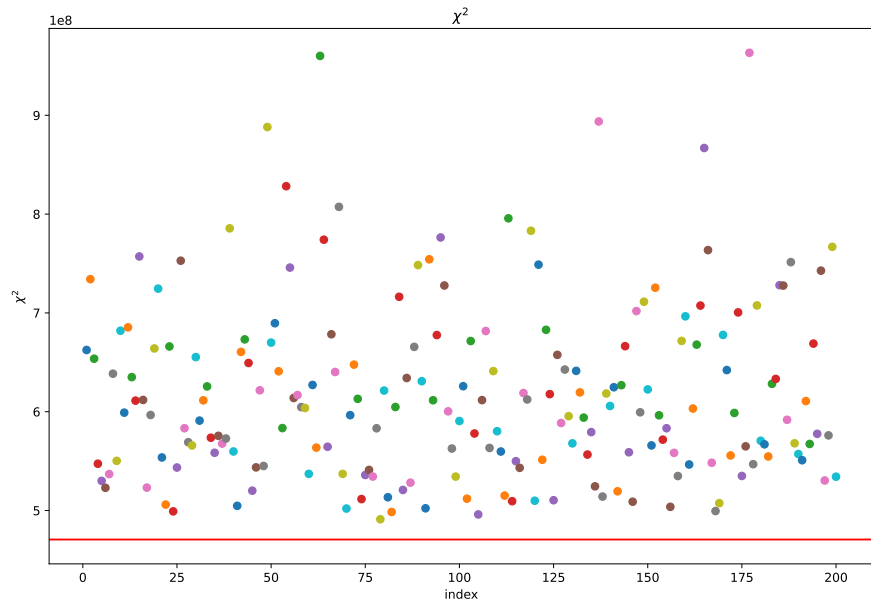


Figure 5:  $\chi^2$ .

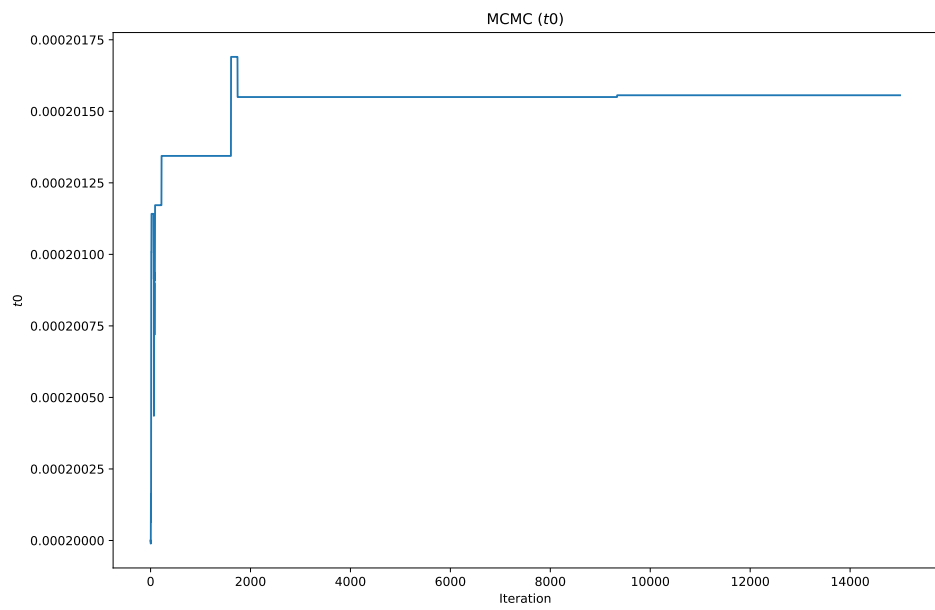


Figure 6: MCMC( $t_0$ ).