PHYS-512, PS4

Muath Hamidi

October, 2022

Problem 1

a.

The Lorentzian is

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

Now, its derivatives in a, t_0, ω are

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

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

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

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

```python
#==================================================
# Course: PHYS 512
# Problem: PS4 P1
#==================================================
# By: Muath Hamidi
# Email: muath.hamidi@mail.mcgill.ca
# Department of Physics, McGill University
# September 2022

#==================================================
# Libraries
#==================================================
import numpy as np # For math
import matplotlib.pyplot as plt # For graphs

#==================================================
# Loading Data
#==================================================
stuff = np.load('sidebands.npz')
t = stuff['time']
d = stuff['signal']

#==================================================
#==================================================
# Part A
#====================================================
#====================================================
# Here you can find the codes related to part (a).
print("====================================================")
print("Part (a)")
print("====================================================")

#====================================================
# Lorantzian
#====================================================
def lorentz(t, a, t0, w):
    y = a / (1 + ((t-t0)/w)**2)
    return y

#====================================================
# Newton Method
#====================================================
def calc_lorentz(p, t):
    # Parameters
    a = p[0]
    t0 = p[1]
    w = p[2]

    # Lorentzian
    y = lorentz(t, a, t0, w)

    grad = np.zeros([t.size, p.size])

    # Differentiate w.r.t. all the parameters
    grad[:,0] = (1 + (t - t0)**2 / w**2)**(-1)
    grad[:,1] = (2 * a * (t - t0)) / (w**2 * (1 + (t - t0)**2 / w**2)**(2))
    grad[:,2] = (2 * a * (t - t0)**2) / (w**3 * (1 + (t - t0)**2 / w**2)**(2))

    return y, grad


p0 = np.array([1.4,0.0002,0.00002]) #starting guess, close but not exact
p = p0.copy()

for j in range(15):
    pred, grad = calc_lorentz(p, t)
    r = d - pred
    err = (r**2).sum()
    r = np.matrix(r).transpose()
    grad = np.matrix(grad)

    lhs = grad.transpose()*grad
    rhs = grad.transpose()*r
    dp = np.linalg.inv(lhs)*(rhs)
    for jj in range(p.size):
        p[jj] = p[jj] + dp[jj]
    print("The parameters (a, t0, w):", p)


print("Best-fit parameters (a, t0, w):", p)
```

```python


# Data
plt.ion()
plt.clf()
plt.scatter(t, d, c="blue", s=0.2, label="Data")

# Calculated
plt.plot(t, pred, c="red", label="Function")
plt.title("d vs t (Newton)")
plt.ylabel("$d$")
plt.xlabel("$t$")
plt.legend()
plt.savefig('4.1.1.pdf', format='pdf', dpi=1200)
plt.show()
plt.close()


#==================================================
#==================================================
# Part B
#==================================================
#==================================================
# Here you can find the codes related to part (b).
print("==================================================")
print("Part (b)")
print("==================================================")

#==================================================
# Noise & Errors
#==================================================
Noise = np.mean((d - pred)**2)
Errors = np.sqrt(Noise * np.diag(np.linalg.inv(grad.T@grad)))
print("Noise:", Noise)
print("Errors in (a, t0, w):", Errors)


#==================================================
#==================================================
# Part C
#==================================================
#==================================================
# Here you can find the codes related to part (c).
print("==================================================")
print("Part (c)")
print("==================================================")

#==================================================
# Numerical Differentiator
#==================================================
def NDiff(f, x, dx=10**-8): # f:function, x:variable, d:delta
    NDiff = (8 * (f(x + dx) - f(x - dx)) - f(x + 2*dx) + f(x - 2*dx)) / (12*dx)
    return NDiff

#==================================================
# Newton Method
```

```python
#==================================================
def Grad(p, t, f):
    a = P[0]
    t0 = P[1]
    w = P[2]

    y = lorentz(t, a, t0, w)

    # Derivative
    Fa = lambda A: f(t, A, t0, w)
    Ft0 = lambda T0: f(t, a, T0, w)
    Fw = lambda W: f(t, a, t0, W)

    # Grad
    Grad_a = NDiff(Fa, a)
    Grad_t0 = NDiff(Ft0, t0)
    Grad_w = NDiff(Fw, w)

    return y, np.array([Grad_a, Grad_t0, Grad_w]).transpose()


P = p0.copy()

for j in range(15):
    pred, grad = Grad(P, t, lorentz)
    r = d - pred
    err = (r**2).sum()
    r = np.matrix(r).transpose()
    grad = np.matrix(grad)

    lhs = grad.transpose()*grad
    rhs = grad.transpose()*r
    dP = np.linalg.inv(lhs)*(rhs)
    for jj in range(P.size):
        P[jj] = P[jj] + dP[jj]
    print("The parameters (a, t0, w):", P)


print("Best-fit parameters (a, t0, w):", P)


#==================================================
#==================================================
# Part D
#==================================================
#==================================================
# Here you can find the codes related to part (d).
print("==================================================")
print("Part (d)")
print("==================================================")

#==================================================
# Lorantzian
#==================================================
def lorentz3(t, a, b, c, t0, dt, w):
    y1 = a / (1 + ((t - t0) / w)**2)
    y2 = b / (1 + ((t - t0 + dt) / w)**2)
    y3 = c / (1 + ((t - t0 - dt) / w)**2)
    return y1 + y2 + y3

def Grad3(p, t, f):
    a, b, c, t0, dt, w = P3

    y = lorentz3(t, a, b, c, t0, dt, w)

    # Derivative
    Fa = lambda A: f(t, A, b, c, t0, dt, w)
    Fb = lambda B: f(t, a, B, c, t0, dt, w)
    Fc = lambda C: f(t, a, b, C, t0, dt, w)
    Ft0 = lambda T0: f(t, a, b, c, T0, dt, w)
    Fdt = lambda dT: f(t, a, b, c, t0, dT, w)
    Fw = lambda W: f(t, a, b, c, t0, dt, W)

    # Grad
    Grad_a = NDiff(Fa, a)
    Grad_b = NDiff(Fb, b)
    Grad_c = NDiff(Fc, c)
    Grad_t0 = NDiff(Ft0, t0)
    Grad_dt = NDiff(Fdt, dt)
    Grad_w = NDiff(Fw, w)

    return y, np.array([Grad_a, Grad_b, Grad_c, Grad_t0, Grad_dt, Grad_w]).
   transpose()


p03 = np.array([1.4, 0.1, 0.06, 0.0002, 0.00005, 0.00002])
P3 = p03.copy()

for j in range(15):
    pred, grad = Grad3(P3, t, lorentz3)
    r = d - pred
    err = (r**2).sum()
    r = np.matrix(r).transpose()
    grad = np.matrix(grad)

    lhs = grad.transpose()*grad
    rhs = grad.transpose()*r
    dP3 = np.linalg.inv(lhs)*(rhs)
    for jj in range(P3.size):
        P3[jj] = P3[jj] + dP3[jj]
    print("The parameters (a, b, c, t0, dt, w):", P3)


print("Best-fit parameters (a, b, c, t0, dt, w):", P3)


# Data
plt.ion()
plt.clf()
plt.scatter(t, d, c="blue", s=0.2, label="Data")

# Calculated
```

```python
plt.plot(t, pred, c="red", label="Function")
plt.title("d vs t (Newton, Numerical)")
plt.ylabel("$d$")
plt.xlabel("$t$")
plt.legend()
plt.savefig('4.1.2.pdf', format='pdf', dpi=1200)
plt.show()
plt.close()

#==================================================
# Noise & Errors
#==================================================
Noise = np.mean((d - pred)**2)
Errors = np.sqrt(Noise * np.diag(np.linalg.inv(grad.T@grad)))
print("Noise:", Noise)
print("Errors in (a, b, c, t0, dt, w):", Errors)


#==================================================
#==================================================
# Part E
#==================================================
#==================================================
# Here you can find the codes related to part (e).
print("==================================================")
print("Part (e)")
print("==================================================")

#==================================================
# Residuals & Residuals Errors
#==================================================
Res = pred - d # Residuals
ResErrs = Noise # Residuals errors

plt.plot(t, Res)
plt.title("Residuals vs t")
plt.ylabel("Residuals")
plt.xlabel("$t$")
plt.savefig('4.1.3.pdf', format='pdf', dpi=1200)
plt.show()
plt.close()


#==================================================
#==================================================
# Part F
#==================================================
#==================================================
# Here you can find the codes related to part (f).
print("==================================================")
print("Part (f)")
print("==================================================")

#==================================================
# Realizations Generation
#==================================================
Covariance = np.linalg.inv(lhs)

RealN = 200 # Realizations number
pred_Gen = np.zeros((RealN, t.size))
for i in range(RealN):
    P_Gen = np.random.multivariate_normal(P3, Covariance) # Generated P
    a, b, c, t0, dt, w = P_Gen
    pred_Gen[i,:] = lorentz3(t, a, b, c, t0, dt, w)
    plt.plot(t, pred_Gen[i,:])


plt.scatter(t, d, c="blue", s=0.2, label="Data")
plt.title("d vs t (Newton, Many fits)")
plt.ylabel("$d$")
plt.xlabel("$t$")
plt.legend()
plt.savefig('4.1.4.pdf', format='pdf', dpi=1200)
plt.show()
plt.close()

#==================================================
# Xi^2
#==================================================
def Xi2(d, pred, Errors):
    Xi2 = np.sum((pred - d)**2 / Errors**2)
    return Xi2

#==================================================
# Xi^2 Generation
#==================================================
typical_diff = np.mean([Xi2(d, pred, Noise) - Xi2(d, pred_Gen[i,:], Noise) for i in
    range(RealN)])
print("Typical difference in X^2: {}".format(typical_diff))

plt.axhline(Xi2(d, pred, Noise), c="r") # Best-Fit X^2
for i in range(RealN):
    plt.scatter(i+1, Xi2(d, pred_Gen[i,:], Noise))

plt.title("$\chi^2$")
plt.ylabel("$\chi^2$")
plt.xlabel("index")
plt.savefig('4.1.5.pdf', format='pdf', dpi=1200)
plt.show()
plt.close()

#==================================================
#==================================================
# Part G
#==================================================
#==================================================
# Here you can find the codes related to part (g).
print("==================================================")
print("Part (g)")
print("==================================================")

#==================================================
# MCMC
#==================================================
def get_step(trial_step):
    return np.random.multivariate_normal(len(trial_step)*[0], trial_step)


iterations = 15000

def MCMC(t, d, p03, Cov, errs, iterations):

    a, b, c, t0, dt, w = p03 # Initial parameters

    chain = np.zeros((iterations, p03.size))
    chain[0,:] = a, b, c, t0, dt, w

    pred = lorentz3(t, a, b, c, t0, dt, w)
    chisq = np.zeros(iterations)
    chisq[0] = Xi2(d, pred, errs) # Initial Xi^2

    # Chain Generation
    for i in range(1, iterations):
        ps = chain[i-1,:]

        # Update
        A, B, C, T0, dT, W = ps + get_step(Cov)
        prediction = lorentz3(t, A, B, C, T0, dT, W)

        # Acceptable Change
        Acc = 0.5*(chisq[0] - Xi2(d, prediction, errs))

        if  np.log(np.random.rand(1)) < Acc:
            A, B, C, T0, dT, W = ps + get_step(Cov)
        else:
            A, B, C, T0, dT, W = ps

        # Prediction After Update
        prediction = lorentz3(t, A, B, C, T0, dT, W)

        # Filling The Chains
        chain[i,:] = A, B, C, T0, dT, W
        chisq[i] = Xi2(d, prediction, errs)

    return chain, chisq


chain, chisq = MCMC(t, d, p03, Covariance, Noise, iterations)

p_names = ["a", "b", "c", "t0", "dt" , "w"]
for i in range(p03.size):
    plt.plot(np.arange(iterations), chain[:,i])
    plt.title("MCMC (${}$)".format(p_names[i]))
    plt.ylabel("${}$".format(p_names[i]))
    plt.xlabel("Iteration")
    plt.savefig('4.1.{}.pdf'.format(6+i), format='pdf', dpi=1200)
    plt.show()
    plt.close()
```

```

#==================================================
# Error
#==================================================
Size = 7500
Error = np.std(chain[Size:,:], axis=0)
print("Standard Deviation in (a, b, c, t0, dt, w):", Error)


#==================================================
#==================================================
# Part H
#==================================================
#==================================================
# Here you can find the codes related to part (h).
print("==================================================")
print("Part (h)")
print("==================================================")

#==================================================
# Width of the Cavity Resonance
#==================================================
# Real w
w_real = 9 * chain[-1,:][5] / chain[-1,:][4]

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:

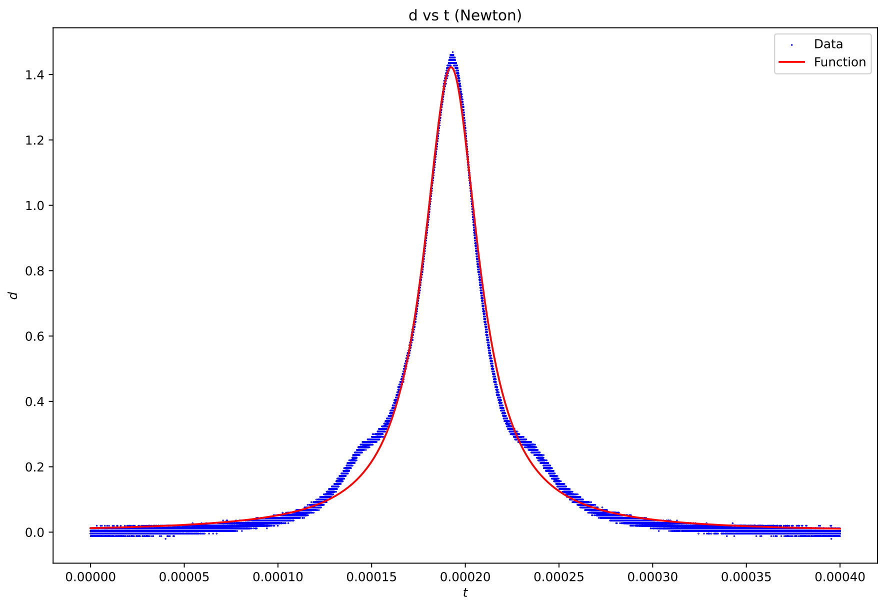

Figure 1: d vs t (Newton).

```
    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 Lorantzian

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

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]:

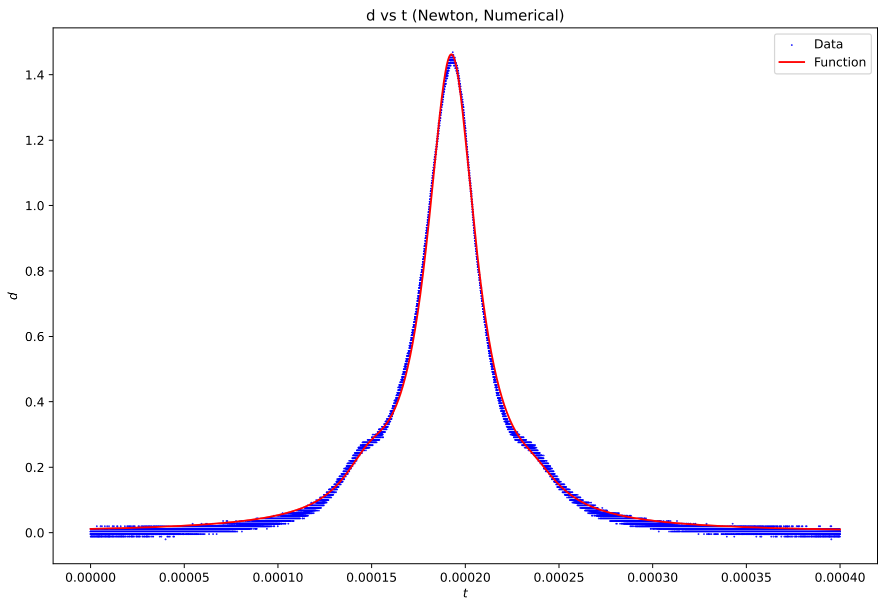

Figure 2: d vs t (Newton, Numerical).

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 χ^2s fig[5]:

Typical difference in χ^2: -146745989.3838729

Which is expected, since the best-fit χ^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]

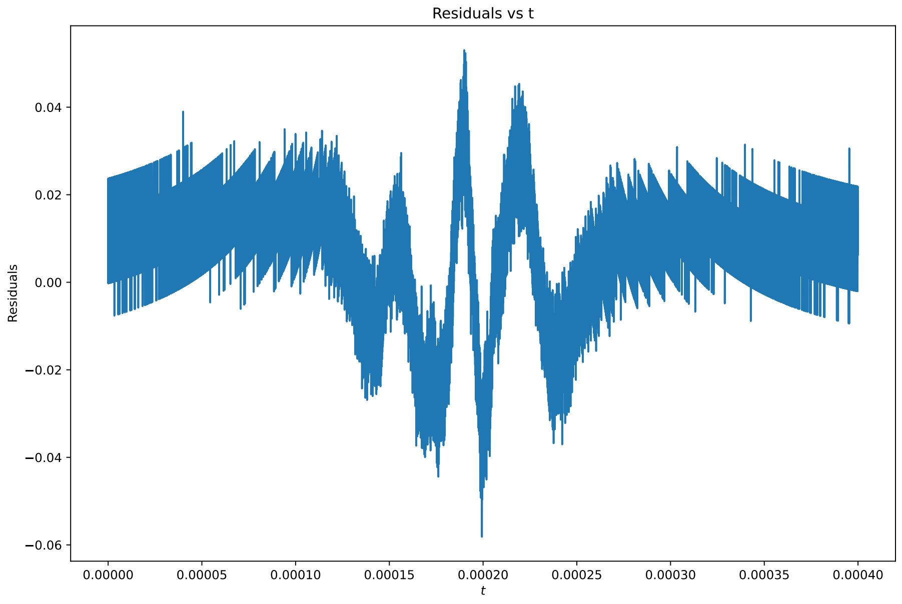

Figure 3: Residuals.

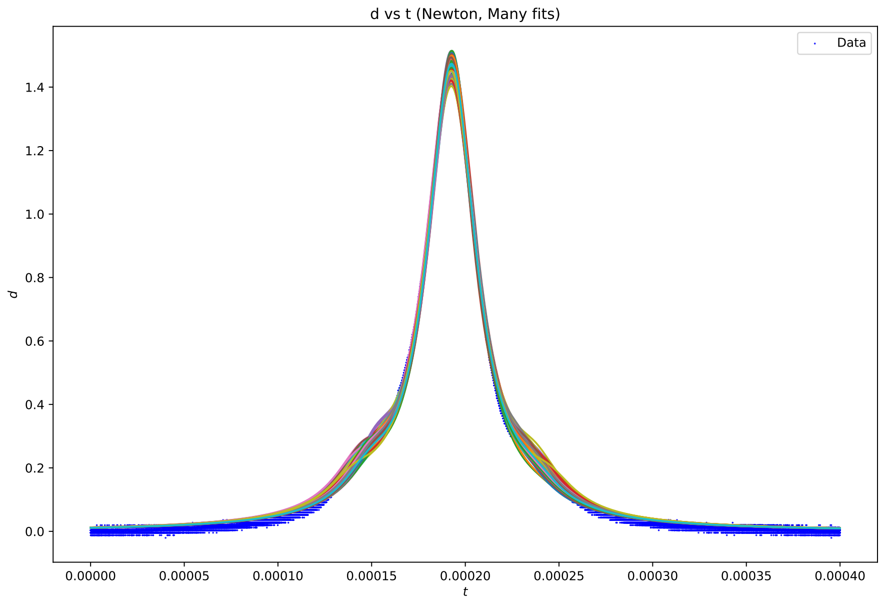

Figure 4: d vs t (Newton, Many Fits).

h.

The actual width of the cavity resonance: 4.46638285825994 GHz

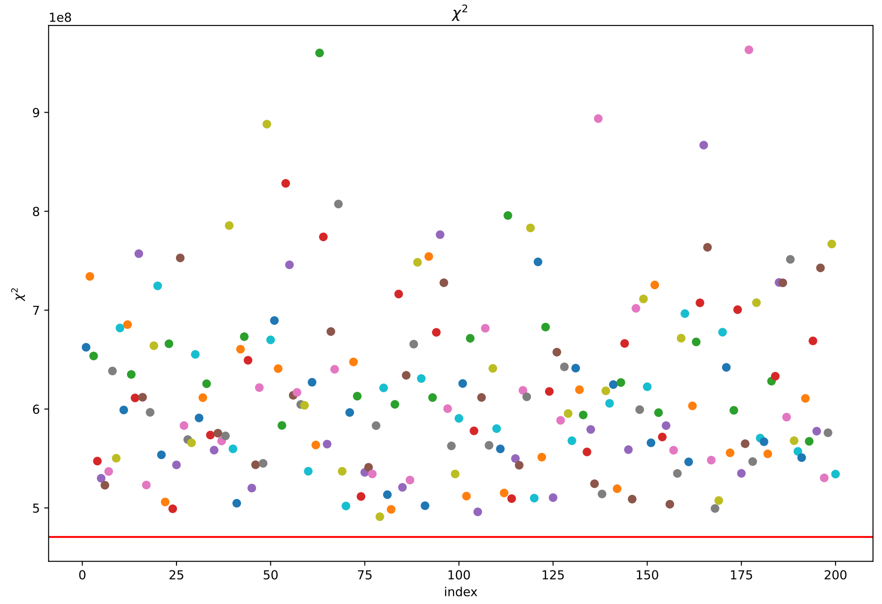

Figure 5: χ^2.

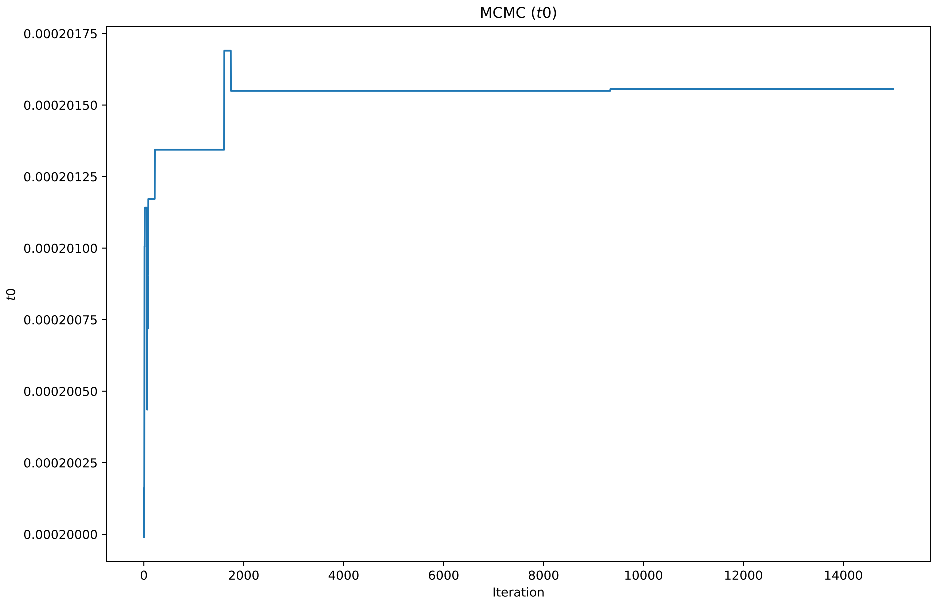

Figure 6: MCMC(t_0).