# Laser Spectroscopy Lab Book

February 17, 2018

## 1 Laser Spectroscopy Lab Notebook

### 1.1 Group Chat:

Ben, tomorrow, I suggest you first try to replicate what work we did tonight. Try to see if you can pull 6 peaks as well. If you can do that, then we need to download the data onto a USB. I recommend pulling one of the nice TekTronix oscilloscopes from the electronics lab outside the room, and try to get one with a USB. I think the one on the far right end has what we are looking for. * Fire up the laser. Wait at least 20 minutes for the bitch to stabilize its temperature. Maybe do this first thing when you get in so it can be warming up while you do the Error Analysis assignment with Jun. * While you are waiting (or once you are done with the assignment), grab the oscilloscope and turn on both it and the function generator. * Input the settings mentioned below * Do science

I think we need to get a solid dataset before we start the interferometry. One of the critical steps is to do what is mentioned in the instructions in all caps. I think we need to use some cursors to determine what the timescale is on the oscilloscope so we can convert from time to frequency after we do the Michelson-Morely interferometer thingy. But we need this data first! You might want to ask Jun for clarification on what the instructions mean, as I am a little confused.

Let me know if you have any issues. I will have my phone on me most the day. Good luck!

Turn on laser Set current to 60.0 mA Set piezo voltage to 59.5ish V (voltage oscillates)

Turn on function generator Plug in Oscilloscope to FG Test triangle function on scope: We noticed there is a 20db Voltage gain between our inputs and the oscilloscope reading, so we adjusted the inputs on the oscilloscope from 10x to 1x: CH2 -> 1x

**FG Settings:**

Ramp wave

500 mHz

4.0 Vpp

**Laser Settings:**

~~111.8 V~~ Set to 100.0 V, but autoadjusts to 100.8 V

~~60 mA~~ 60.0 mA

Note: Higher frequencies can oscillate the piezo enough to generate sound!

Signal splitter from FG output: 1 to Oscilloscope, 1 to Laser Diode

We tried following the method mentioned in the Vortex Users guide: start at piezo voltage 59 V, set input voltage to a max of 4.5 Vpp. However, we were unable to see any fluorescence at these settings.

Adjusted baseline piezo voltage and set oscillation voltage to 0. We then scanned the piezo voltage spectrum until we found two "Intensity peaks": One at 109.3 V and the other at 113.8 V. I wonder if these are the $^{87}Rb \quad F = 1 \rightarrow F' = 1, 2, 3$ and the $^{85}Rb \quad F = 3 \rightarrow F' = 2, 3, 4$ transitions so close together (and mentioned in the lab manual).

88.8 Turning on 90.5 Peak 91.7 trough 93.2 Peak 94.1 Turning off 96.9 Turning on 98.1 Peak 99.0 Turned off 101.3 Turning on 103.6 Turning off

### 1.1.1 31 Jan:

Vortex Laser head would not fire up. After $\approx$ 45 minutes, the controller still read that the temperature was unstable. Replaced with different head and realigned beams.

**Current Settings**

Laser: * Piezo Voltage: 90.2 V * Current: 60.0 mA

Function Generator: * Ramp * Frequency: 20 Hz * Amplitude: 5.5 Vpp * Offset: 0.0 VDC

Oscilloscope: * Channel 1: Beam 1 * Channel 2: Beam 2 (blocked for calibration * EXT Trig: Function Generator * V Scale: 50.0 mV * T Scale: 100.0 $\mu s$ * Both Channels: * Coupling: DC * BW Limit: ON * Probe: 1x * Invert: ON * Trigger Menu: * Mode: Edge * Source: Ext * Slope: RightDownRight * Sweep: Auto * Coupling: DC

Laser voltage 91.9 current 60.0 mA

Triangle Wave 20 Hz 1-3 VPP

Turn on laser key, allow $\approx$ 20 minutes to warm up before turning on beam Turn on Waveform Generator Turn on Photodetectors

Adjust settings to those mentioned in previous block. Invert all plots Go for AC setting

Use laser voltage to shift plot L/R Once on 87Rb Peak (smaller one on right) adjust waveform generator frequency and Vpp amplitude voltage until signals become clear.

To generate solid looking plot, go to aquire and set averages

Qs: Use Trigger Edge, but see about setting diffenent direction choices

In [129]:
```python
%matplotlib inline
import os
import glob
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.special import erf,erfc
from scipy.optimize import curve_fit
from scipy import signal

plt.style.use('ggplot')
```

In [130]:
```python
# Physical Constants (SI)
c = 2.998E8 # Speed of light (m/s)
kB = 1.381E-23 # Boltzmann constant (J/K)
m85 = 1.410E-25 # 85Rb mass (kg)
m87 = 1.443E-25 # 87Rb mass (kg)
lambda0 = 780.241E-9 #D2 line (m)
v0 = c/lambda0
v0/1E12
```

Out[130]: 384.24025397281093

```
In [131]: # 2/lambda0*np.sqrt(2*kB*293/m85*np.log(2))/10**6
          2/lambda0*np.sqrt(2*kB*293/m87*np.log(2))/10**6
```

```
Out[131]: 505.38976225508907
```

```
In [132]: # data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/NewFile0.csv'))
          # #data1 = pd.read_csv('/home/bjorn/Documents/CU/PHYS_4430/Test_Profile_Data.csv', ind

          # data1 = data[:]
          # del data1['Unnamed: 3']
          # data1 = data1.drop(0)
          # # xa = np.array(float(data1['X']))
          # # y1 = np.array(data['CH1'])
          # # y2 = np.array(data['CH2'])

          # data1=data1.astype('float')
          # data1['CH1'] = data1['CH1']*-1
          # data1['CH2'] = data1['CH2']*-1
          # data1 = data1.rename(columns={'X':'t (s)', 'CH1':'Pumped Probe (V)', 'CH2':'Doppler


          # fig = plt.figure()
          # ax = plt.subplot(111)
          # plt.figure(figsize=(12,15))
          # data1.plot(kind = 'scatter',x='t (s)', y='Pumped Probe (V)',style = 'b.', ax = ax, l
          # data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler Shifted Probe (V)', style = 'b.
          # ax.set_xlim(data1['t (s)'].min(), data1['t (s)'].max())
          # ax.set(ylabel='Voltage (V)')
          # ax.legend(loc='best')
          # fig.set_figheight(15)
          # fig.set_figwidth(15)
          # ax.set_title('Data from lame oscilloscope')

          # plt.show()
```

### 1.1.2 2 Feb:

Collected new data on Tektronix MDO3014:
**Laser Settings** * 92.8 V * 60.3 mA
**Function Generator** * Ramp * 7 Hz * 800 mVpp

```
In [133]: # data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/tek0014ALLclear.csv'))
          # #data1 = pd.read_csv('/home/bjorn/Documents/CU/PHYS_4430/Test_Profile_Data.csv', ind

          # data1 = data[:]
          # # del data1['Unnamed: 3']
          # # data1 = data1.drop(0)
          # # xa = np.array(float(data1['X']))
          # # y1 = np.array(data['CH1'])
```

```
# # y2 = np.array(data['CH2'])

# data1=data1.astype('float')
# # data1['CH1'] = data1['CH1']*-1
# # data1['CH2'] = data1['CH2']*-1
# data1 = data1.rename(columns={'TIME':'t (s)', 'CH1':'Pumped Probe (V)', 'CH2':'Doppl


# fig = plt.figure()
# ax = plt.subplot(111)
# plt.figure(figsize=(12,15))
# data1.plot(kind = 'scatter',x='t (s)', y='Pumped Probe (V)',color = 'yellow', ax = a
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler Shifted Probe (V)', color='blue
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler-Free Saturated Absorption Spect
# ax.set_xlim(data1['t (s)'].min(), data1['t (s)'].max())
# ax.set(ylabel='Voltage (V)')
# ax.legend(loc='best')
# fig.set_figheight(15)
# fig.set_figwidth(15)
# ax.set_title('Data from lame oscilloscope')

# plt.show()
```

Questions:

How much of an overlap betweeen the first probe beam and the pump beam is required? How close can the pump beam get to the second probe beam?

```
In [134]: # data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/tek0014ALLclear.csv'))
          # data
```

**Prodedure** * Set up table as shown in picture, align beams. * Turn off lights * Once ready to take data, block pump beam and manipulate variable attenuator until voltage readings from both channels equalize * Remove block from pump beam, adjust laser voltage/current until fluorescence observed. * Turn on function generator and set baseline settings. * Toggle (some combo of 2: laser voltage/oscilloscope trigger/fn gen freq/fn gen vpp) until desired peak is on scope. * Toggle (other combo of 2) until desired resolution is achieved. * Make sure ramp (trigger from fn generator) is on screen * Collect Data, output to USB * Observe voltage of ramp where peaks of spectra occur * Insert Beam diverter into Michelson interferometer to obtain voltage to frequency conversion. * convert ramp voltage to frequency to determine frequency of observed spectral lines. * compare to known frequencies for hyperfine transitions.

```
In [135]: print((113.8+109.8)/2)
          print(113.9-111.8)
          print(111.8-109.8)
```

```
111.8
2.1000000000000085
2.0
```

**Change Log** 31 Jan: Had to swap out Laser head. After ≈ 45 minutes, head was still not temperature stable. Adjusted laser settings accordingly.

**Helpful Links** Rigol Function Generator http://pmaweb.caltech.edu/~phy003/DG1022_UserGuide_EN.pdf
Rigol Oscilloscope https://cdn-shop.adafruit.com/datasheets/Users+Guide+DS1000E.pdf

### 1.1.3 2 February

- I noticed that when I have the voltage ramping up, the datasheet for the laser says the frequency should be increasing. I will record data this way, since I want increasing frequency going to the right.

**Procedure** * On Oscope, set trigger to waveform generator. Once trigger is set, it shouldnt have to be reset * Adjust laser voltage and current until desired peaks are on scope * Manipulate function generator Vpp concurrently with laser voltage until you achieve desired window of data. Use the trigger signal to tell where this window is. * Adjusting frequency seems to spread out the signal without increasing resolution. Adjust until satisfied. Use aquire->averages to clean up signal * Save Data

95.9 V 60.0 mA

```
In [136]: # data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/New_Data/87Rb_01_clean.csv'))
          # #data1 = pd.read_csv('/home/bjorn/Documents/CU/PHYS_4430/Test_Profile_Data.csv', ind

          # data1 = data[:]
          # # del data1['Unnamed: 3']
          # # data1 = data1.drop(0)
          # # xa = np.array(float(data1['X']))
          # # y1 = np.array(data['CH1'])
          # # y2 = np.array(data['CH2'])

          # data1=data1.astype('float')
          # # data1['CH1'] = data1['CH1']*-1
          # # data1['CH2'] = data1['CH2']*-1
          # data1 = data1.rename(columns={'TIME':'t (s)', 'CH1':'Pumped Probe (V)', 'CH2':'Doppl


          # fig = plt.figure()
          # ax = plt.subplot(111)
          # plt.figure(figsize=(12,15))
          # data1.plot(kind = 'scatter',x='t (s)', y='Pumped Probe (V)',color = 'yellow', ax = a
          # data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler Shifted Probe (V)', color='blue
          # data1.plot(kind = 'scatter', x='t (s)', y = 'Trigger Signal', color='green',  ax = a
          # data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler-Free Saturated Absorption Spect
          # ax.set_xlim(data1['t (s)'].min(), data1['t (s)'].max())
          # ax.set(ylabel='Voltage (V)')
          # ax.legend(loc='best')
          # fig.set_figheight(15)
          # fig.set_figwidth(15)
```

```
# ax.set_title('Doppler-Free Saturated Absorption Spectroscopy with ${}^{87}Rb$')

# plt.show()
```

In [137]:
```
# data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/New_Data/85Rb_01_clean.csv'))
# #data1 = pd.read_csv('/home/bjorn/Documents/CU/PHYS_4430/Test_Profile_Data.csv', ind

# data1 = data[:]
# # del data1['Unnamed: 3']
# # data1 = data1.drop(0)
# # xa = np.array(float(data1['X']))
# # y1 = np.array(data['CH1'])
# # y2 = np.array(data['CH2'])

# data1=data1.astype('float')
# # data1['CH1'] = data1['CH1']*-1
# # data1['CH2'] = data1['CH2']*-1
# data1 = data1.rename(columns={'TIME':'t (s)', 'CH1':'Pumped Probe (V)', 'CH2':'Doppl


# fig = plt.figure()
# ax = plt.subplot(111)
# plt.figure(figsize=(12,15))
# data1.plot(kind = 'scatter',x='t (s)', y='Pumped Probe (V)',color = 'yellow', ax = a
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler Shifted Probe (V)', color='blue
# data1.plot(kind = 'scatter', x='t (s)', y = 'Trigger Signal', color='green',  ax = a
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler-Free Saturated Absorption Spect
# ax.set_xlim(data1['t (s)'].min(), data1['t (s)'].max())
# ax.set(ylabel='Voltage (V)')
# ax.legend(loc='best')
# fig.set_figheight(15)
# fig.set_figwidth(15)
# ax.set_title('Doppler-Free Saturated Absorption Spectroscopy with ${}^{85}Rb$')

# plt.show()
```

In [138]:
```
c = 2.998E8
dNu = .5E9
print(c/(2*dNu))
```

```
0.2998
```

### 1.1.4 6 Feb

**Laser** * 60.0 mA * 92.4 V

**Fn Generator** * 180 mVpp * 32 Hz

Goal: Today, we want to get the Michelson interferometer working.

We noticed (Thanks Jun) that the interferometer is sending a beam back into the setup. We need to block the beam into the interferometer when collecting data, or else we will get interference fringes imposed on the data.

**Mirror Distances in Interferometer** * M1: * M2:

We placed an attenuator between the interferometer array and the spectroscopy array to minimize optical feedback into the laser cavity.

Once we properly adjusted the mirrors so that we obtained the proper interferometry signal, we plugged the signal from the interferometer setup into the main oscilloscope collecting data on the Rubidium spectroscopy experiment. We can now directly compare the fringe widths from the interferometry with the transition lines from the spectroscopy.

We can directly compare the

The spectra we are interested in measuring are much closer spaced than the peaks of the interference pattern. We are going to try to increase $\Delta L$ so we can get more fringes per unit time.

**Procedure** * Before turning on function generator, measure DC coupled signals from photodetectors. Adjust attenuator on probe 2 so that its output voltage is of the same intensity. This should help ensure subtraction of signals is most accurate * We want at least 3 peaks from the interferometer signal so we can ensure we have a reasonable sampling. This should also help with error analysis

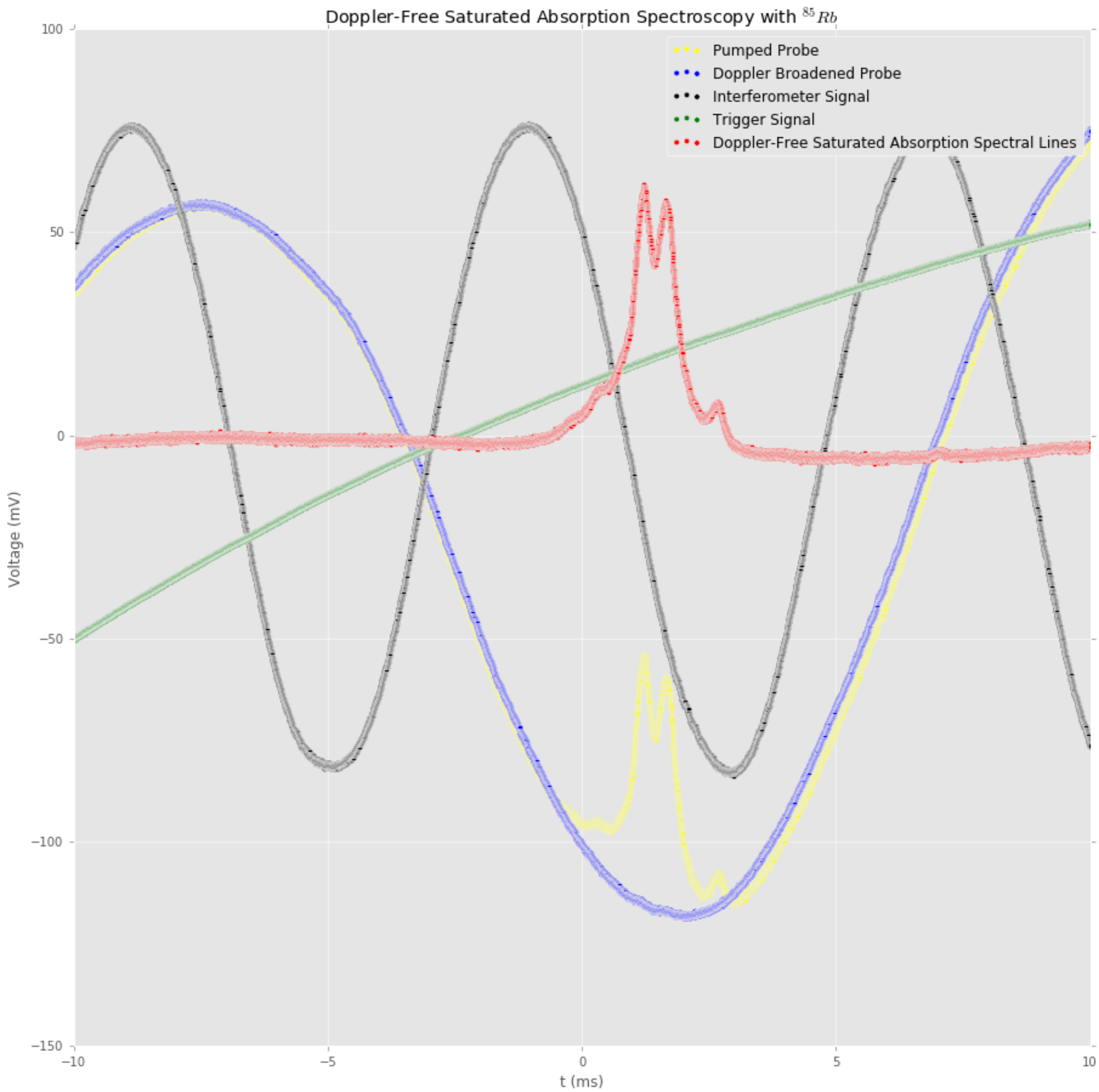
In [139]:
```
# data = pd.DataFrame(pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_03.csv',usecols=
#data1 = pd.read_csv('/home/bjorn/Documents/CU/PHYS_4430/Test_Profile_Data.csv', index

# data1 = data[:]
# del data1['Unnamed: 3']
# data1 = data1.drop(0)
# xa = np.array(float(data1['X']))
# y1 = np.array(data['CH1'])
# y2 = np.array(data['CH2'])

# data1=data1.astype('float')
# data1['CH1'] = data1['CH1']*-1
# data1['CH2'] = data1['CH2']*-1
# data1 = data1.rename(columns={'TIME':'t (s)', 'CH1':'Pumped Probe (V)', 'CH2':'Doppl


# fig = plt.figure()
# ax = plt.subplot(111)
# plt.figure(figsize=(12,15))
# data1.plot(kind = 'scatter',x='t (s)', y='Pumped Probe (V)',color = 'yellow', ax = a
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler Broadened Probe (V)', color='bl
# data1.plot(kind = 'scatter', x='t (s)', y = 'Interferometer Signal', color='black',
# data1.plot(kind = 'scatter', x='t (s)', y = 'Trigger Signal', color='green',  ax = a
# data1.plot(kind = 'scatter', x='t (s)', y = 'Doppler-Free Saturated Absorption Spect
# ax.set_xlim(data1['t (s)'].min(), data1['t (s)'].max())
# ax.set(ylabel='Voltage (V)')
# ax.legend(loc='best')
# fig.set_figheight(15)
# fig.set_figwidth(15)
```

```
# ax.set_title('Doppler-Free Saturated Absorption Spectroscopy with ${}^{85}Rb$')

# plt.show()
```

In [140]:
```
# Rb85_1 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_01.csv',usecols=[0,1,2,3,4,
# Rb85_2 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_02.csv',usecols=[0,1,2,3,4,
# Rb85_3 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_03.csv',usecols=[0,1,2,3,4,
# Rb85_4 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_04.csv',usecols=[0,1,2,3,4,

Rb85_1 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_01.csv',usecols=[0,1,2,3,4,7]
Rb85_2 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_02.csv',usecols=[0,1,2,3,4,7]
Rb85_3 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_03.csv',usecols=[0,1,2,3,4,7]
Rb85_4 = pd.read_csv('Laser_Spec_Data/6-Feb/85Rb_06FEB18_04.csv',usecols=[0,1,2,3,4,7]
Rb87_1 = pd.read_csv('Laser_Spec_Data/6-Feb/87Rb_06FEB18_01.csv',usecols=[0,1,2,3,4,7]
Rb87_2 = pd.read_csv('Laser_Spec_Data/6-Feb/87Rb_06FEB18_02.csv',usecols=[0,1,2,3,4,7]
#data=data.astype('float')


def rb85Plot(dataframe):
    fig = plt.figure()
    ax = plt.subplot(111)
    plt.figure(figsize=(15,15))
    dataframe.plot(kind = 'scatter',x='t (ms)', y='Pumped Probe (mV)',color = 'yellow'
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Doppler Broadened Probe (mV)', c
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Interferometer Signal', color='b
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Trigger Signal', color='green',
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Doppler-Free Saturated Absorptio
    ax.set_xlim(dataframe['t (ms)'].min(), dataframe['t (ms)'].max())
    ax.set(ylabel='Voltage (mV)')
    ax.legend(loc='best')
    fig.set_figheight(15)
    fig.set_figwidth(15)
    ax.set_title('Doppler-Free Saturated Absorption Spectroscopy with ${}^{85}Rb$')

    plt.show()

def rb87Plot(dataframe):
    fig = plt.figure()
    ax = plt.subplot(111)
    plt.figure(figsize=(15,15))
    dataframe.plot(kind = 'scatter',x='t (ms)', y='Pumped Probe (mV)',color = 'yellow'
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Doppler Broadened Probe (mV)', c
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Interferometer Signal', color='b
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Trigger Signal', color='green',
    dataframe.plot(kind = 'scatter', x='t (ms)', y = 'Doppler-Free Saturated Absorptio
    ax.set_xlim(dataframe['t (ms)'].min(), dataframe['t (ms)'].max())
    ax.set(ylabel='Voltage (mV)')
    ax.legend(loc='best')
```

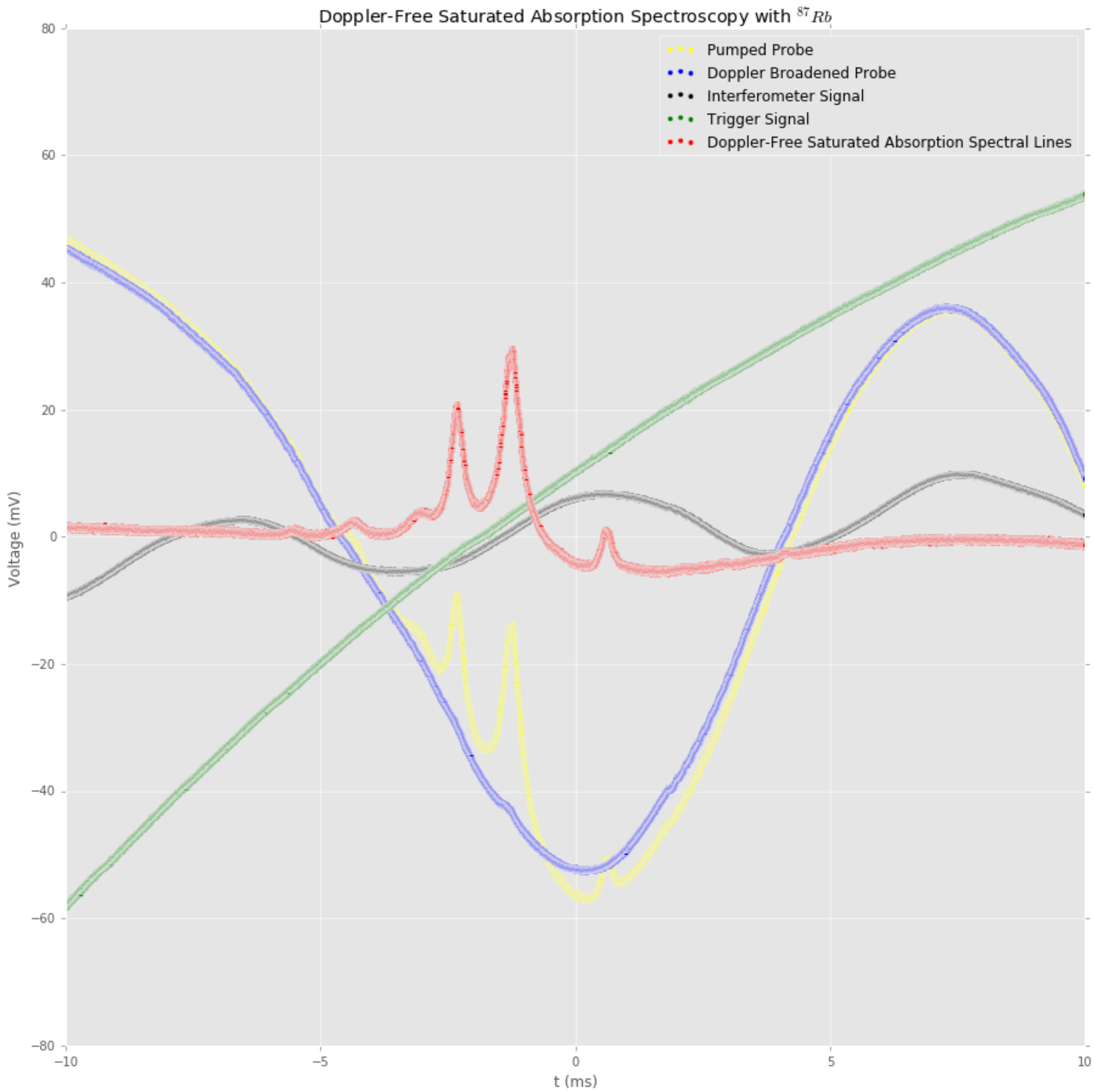
```
    fig.set_figheight(15)
    fig.set_figwidth(15)
    ax.set_title('Doppler-Free Saturated Absorption Spectroscopy with ${}^{87}Rb$')

    plt.show()
```

In [141]: 
```
rb85Plot(Rb85_4)
```




```
<matplotlib.figure.Figure at 0x7f133da4dda0>
```

In [142]: 
```
rb87Plot(Rb87_2)
```




```
<matplotlib.figure.Figure at 0x7f133ca900f0>
```

```
In [143]: def cos_fit(t,a,T,t0,d):
              return a*np.cos(2*np.pi*(t-t0)/T) + d*t

          def gaussian(x, A, sigma, x0, y0):
              return A/(sigma*np.sqrt(2*np.pi))*np.exp(-(x-x0)**2/(2*sigma)**2)+y0

          def lorentzian(x, A, gamma, x0):
              return A/(np.pi*gamma*(1 + ((x-x0)/gamma)**2))

          def zoom(dataset, lbound, rbound):
```

```
    asarray = np.array(dataset['t (ms)'])
    l_idx = np.searchsorted(asarray,lbound)
    r_idx = np.searchsorted(asarray,rbound)
    return dataset[l_idx:r_idx]

def findPeaks(data, column):
    idx = np.argmax(np.array(data[column]))
    base = data.index[0]
    return idx+base

def showPeaks(data, column, boundList):
    ax = data.plot(x = 't (ms)', figsize = (15,15))
    for bounds in boundList:
        zmData = zoom(data, bounds[0], bounds[1])
        peakInd = findPeaks(zmData, column)
        ymin, ymax = ax.get_ylim()
        ax.vlines(x = zmData['t (ms)'].loc[peakInd], ymin=ymin, ymax = ymax-1)

def getPeaks(data, column, boundList):
    peaks = []
    for bounds in boundList:
        zmData = zoom(data, bounds[0], bounds[1])
        peakInd = findPeaks(zmData, column)
        peaks.append(zmData['t (ms)'].loc[peakInd])
    return peaks

def fringe_time(data, tCol, sigCol, guess, plot=True):
    best_vals, covar = curve_fit(cos_fit, data[tCol], data[sigCol], p0 = guess)
    dT = best_vals[1]
    errT = np.sqrt(np.diag(covar))[1]
    a = int(np.floor(np.log10(np.abs(errT))))
    report_Err = round(errT, -a)
    report_T = round(dT, -a)
    if plot:
        ax = data.plot(x = tCol,y= sigCol, figsize = (15,15))
        ax.plot(data[tCol], cos_fit(data[tCol],*best_vals), label = 'Best Fit: Period
        ax.legend(loc='best')
        plt.show()

    return dT, errT

def FWHM_fit(data, tCol, sigCol, guess, plot=True):
    """
    Returns FWHM of a gaussian (as given by FWHM = 2 sqrt(2*ln(2))*sigma) and its fit
    """
    best_vals, covar = curve_fit(gaussian, data[tCol], data[sigCol], p0 = guess)
    Sigma = best_vals[1]
    errSigma = np.sqrt(np.diag(covar))[1]
```

```
            sigma_Precision = int(np.floor(np.log10(np.abs(2*np.sqrt(2*np.log(2))*errSigma))))
            reportedFWHM = round(2*np.sqrt(2*np.log(2))*Sigma, -sigma_Precision)
            reportedFWHMerr = round(2*np.sqrt(2*np.log(2))*errSigma, -sigma_Precision)
            tPos = best_vals[2]
            tErr = np.sqrt(np.diag(covar))[2]
            pos_precision = int(np.floor(np.log10(np.abs(tErr))))
            reported_pos = round(tPos, -pos_precision)
            reported_pos_err = round(tErr, -pos_precision)

            if plot:
                ax = data.plot(x = tCol,y= sigCol, figsize = (15,15))
                ax.plot(data[tCol], gaussian(data[tCol],*best_vals), label = 'Best Fit:\nPosit
                ax.legend(loc='best')
                plt.show()

            return tPos, tErr, 2*np.sqrt(2*np.log(2))*Sigma, 2*np.sqrt(2*np.log(2))*errSigma

        def freq_converter(tau, dT, dL = 0.294):
            return c*tau/(2*dL*dT)
```

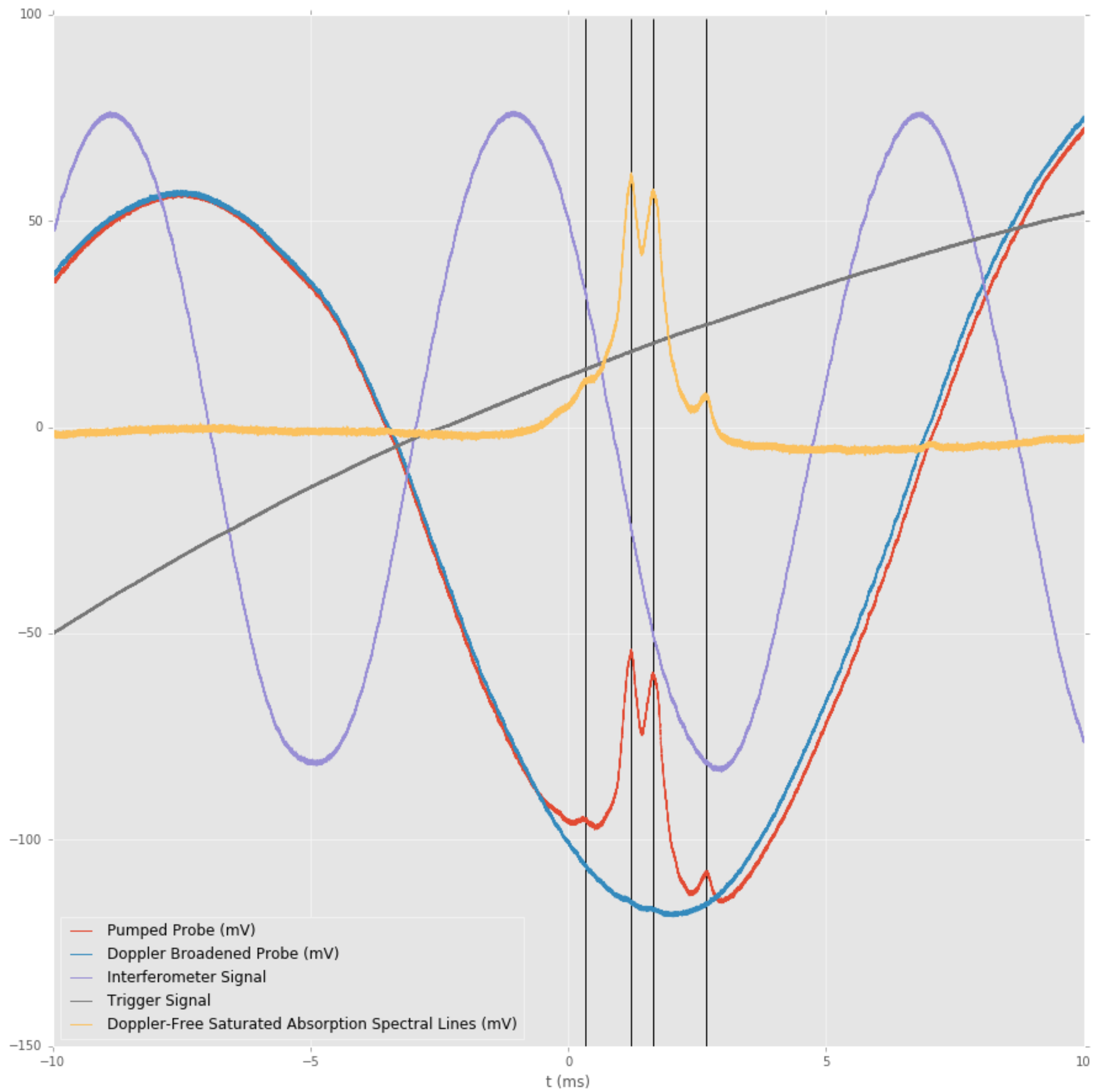
In [144]:
```
c/(2*.294)/1E6
```

Out[144]: 509.8639455782313

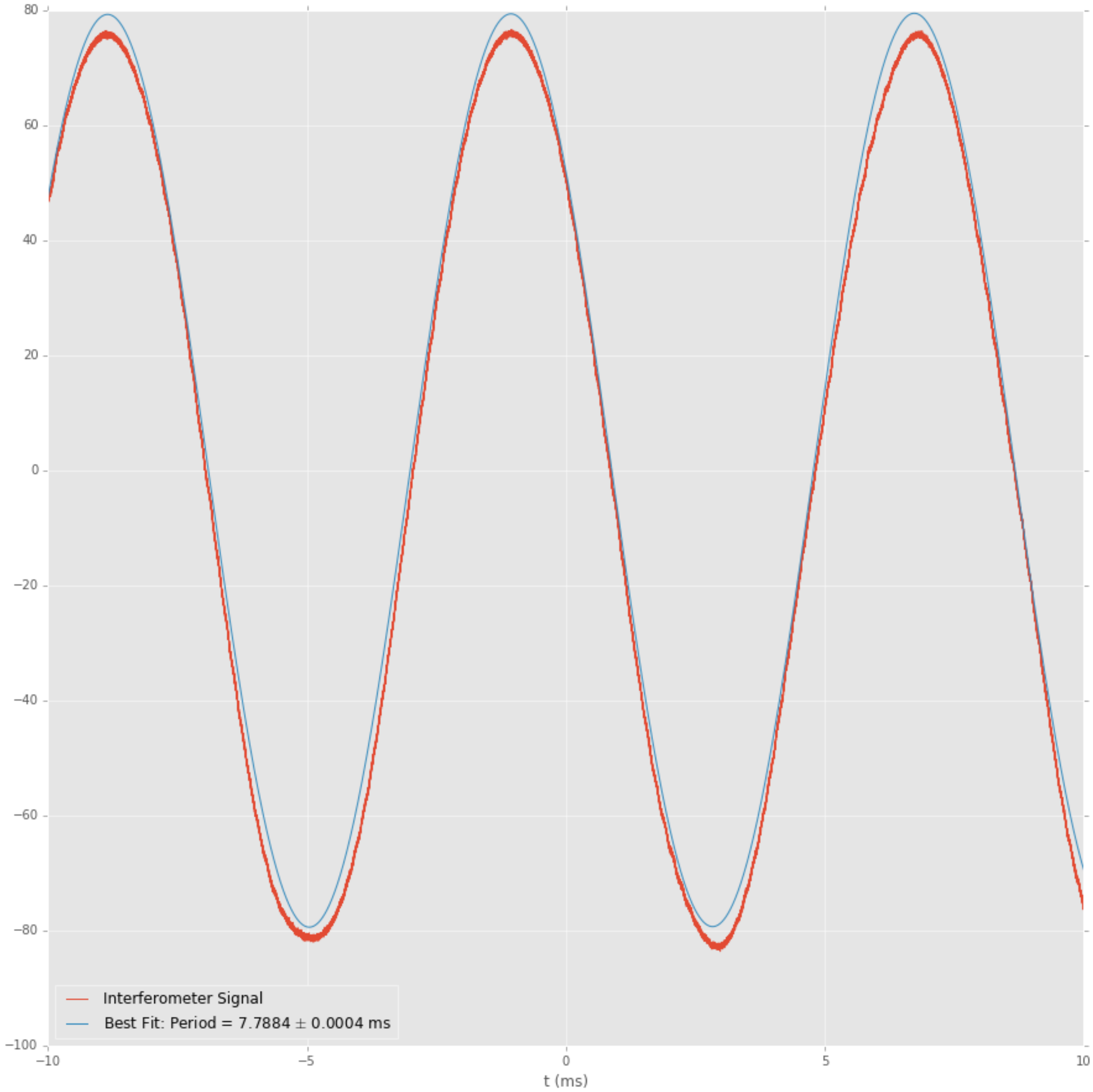
### 1.1.5 $^{85}Rb$ Analysis

In [145]:
```
showPeaks(Rb85_4, 'Doppler-Free Saturated Absorption Spectral Lines (mV)', [[0,.35], [
getPeaks(Rb85_4, 'Doppler-Free Saturated Absorption Spectral Lines (mV)', [[0,.35], [1
```

Out[145]: [0.34299999999999997, 1.2264000000000002, 1.6520000000000001, 2.6694]

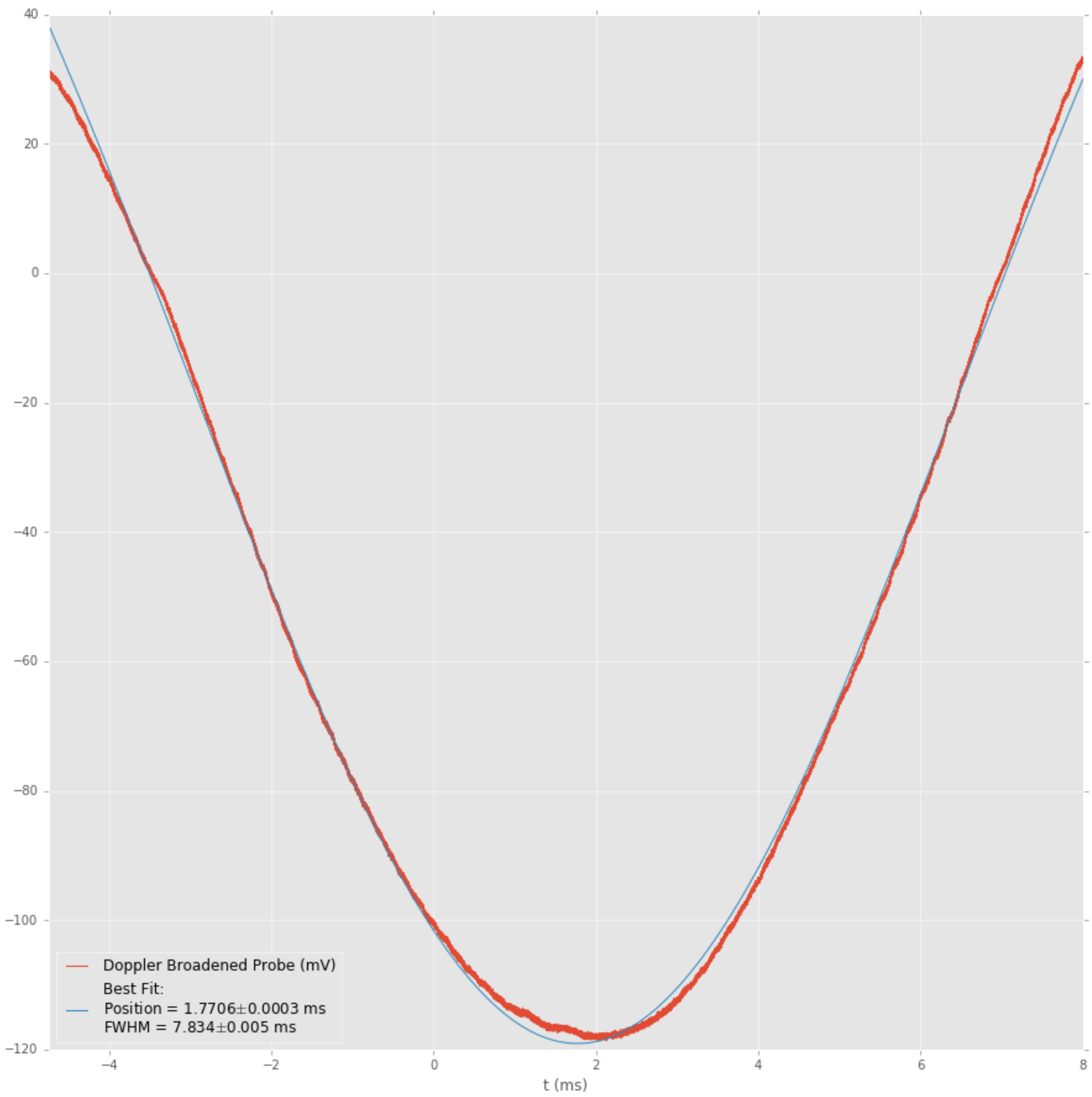



```
In [146]: fringe_time(Rb85_4, 't (ms)', 'Interferometer Signal',[75,6,-2,0])
```



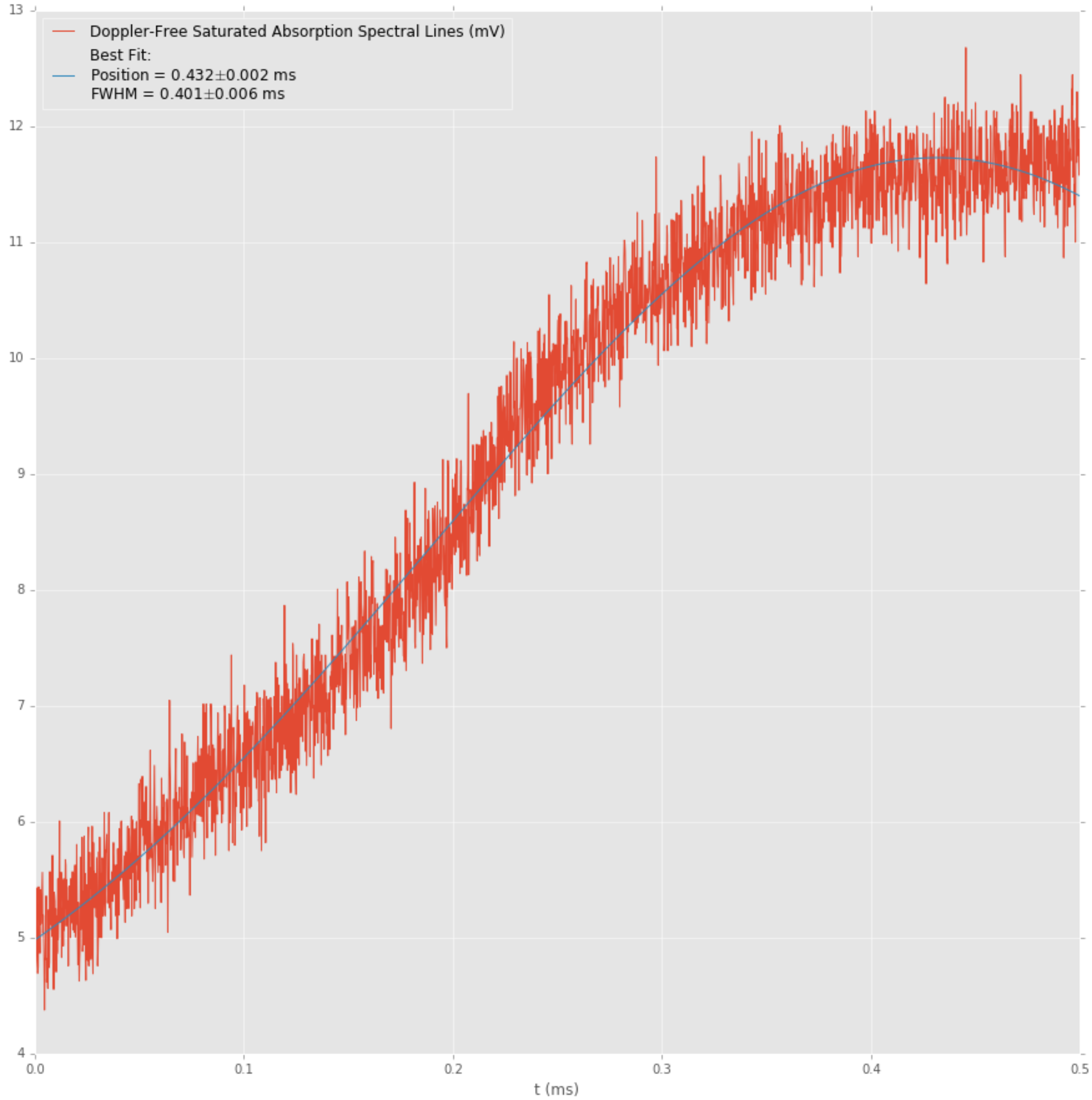

```
Out[146]: (7.7884442789261179, 0.00037669318749080803)
```

```
In [147]: FWHM_fit(zoom(Rb85_4,-4.75,8), 't (ms)', 'Doppler Broadened Probe (mV)', [-500,2,2,40]
```



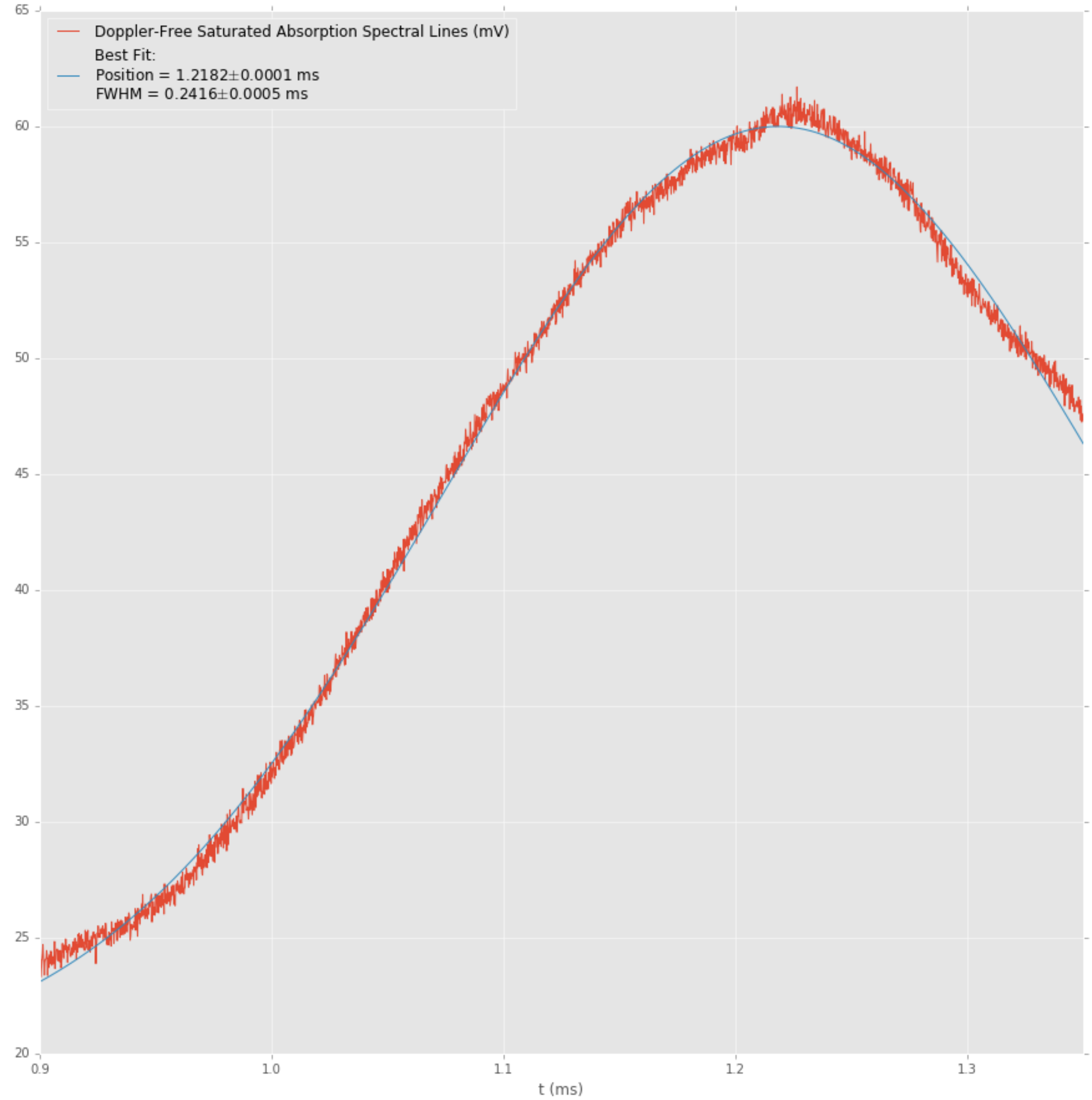

```
Out[147]: (1.7706302205772462,
           0.00027197282363109436,
           7.8339824216998393,
           0.0051256699274414 85)
```

```
In [148]: z85 = zoom(Rb85_4,0,.45)
          # ax = z85.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb85_4,0,.5), 't (ms)', 'Doppler-Free Saturated Absorption Spectral Line
```



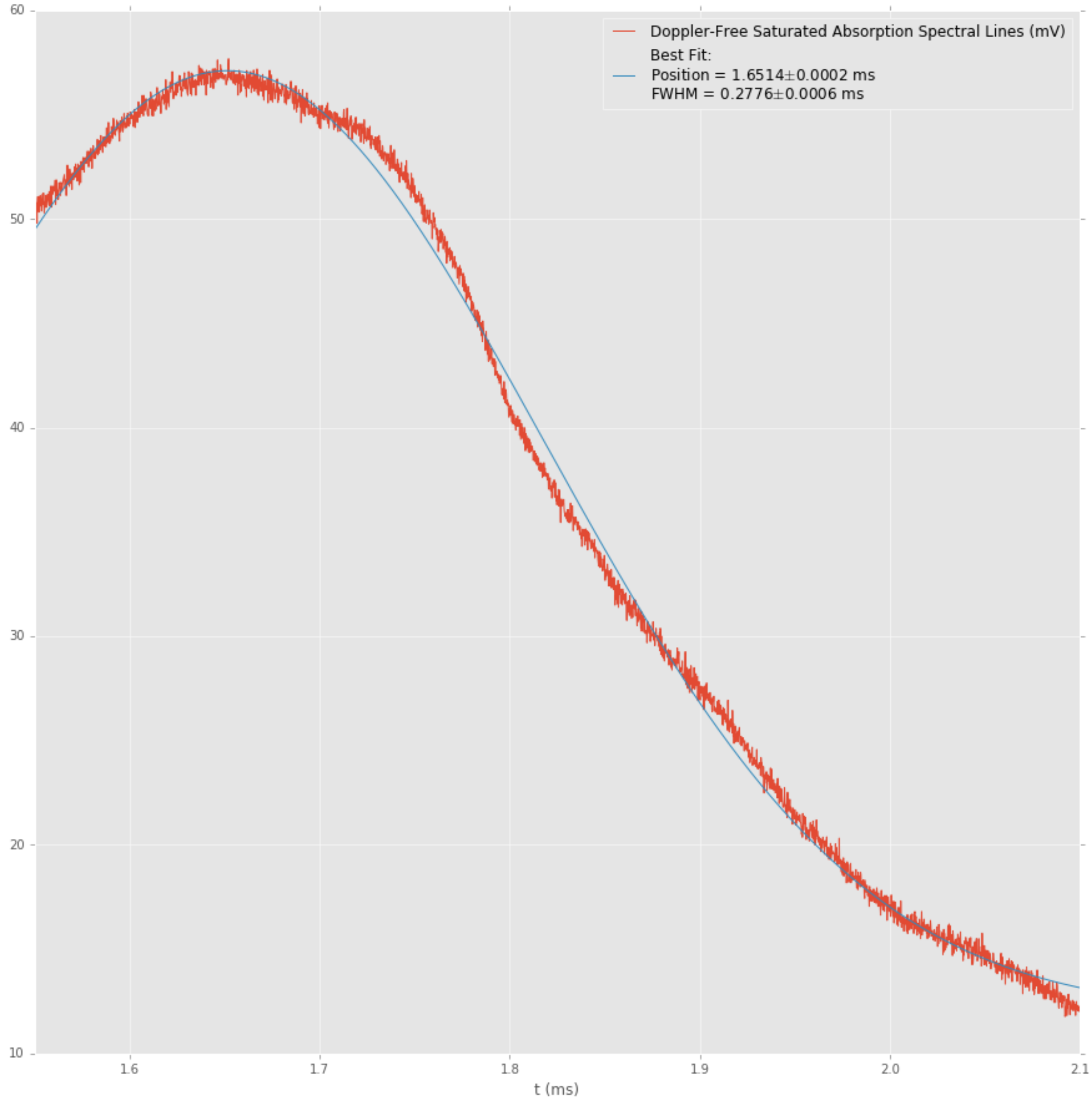

```
Out[148]: (0.43190174501689182,
           0.0016284767083484058,
           0.40052980116184389,
           0.0058255268541152856)
```

```
In [149]: z85 = zoom(Rb85_4,.9,1.35)
          # ax = z85.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb85_4,0.9,1.35), 't (ms)', 'Doppler-Free Saturated Absorption Spectral
```




```
Out[149]: (1.2182124134843031,
           0.00011996914718684 75,
           0.24157498497558719,
           0.00052173961017634295)
```

```
In [150]: z85 = zoom(Rb85_4,1.55,2.3)
          # ax = z85.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb85_4,1.55,2.1), 't (ms)', 'Doppler-Free Saturated Absorption Spectral
```




```
Out[150]: (1.6514116666950982,
           0.00024586894788794657,
           0.27763862618733065,
           0.00061267788708043434)

In [151]: z85 = zoom(Rb85_4,2.5,3)
          # ax = z85.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb85_4,2.6,3), 't (ms)', 'Doppler-Free Saturated Absorption Spectral Lin
```

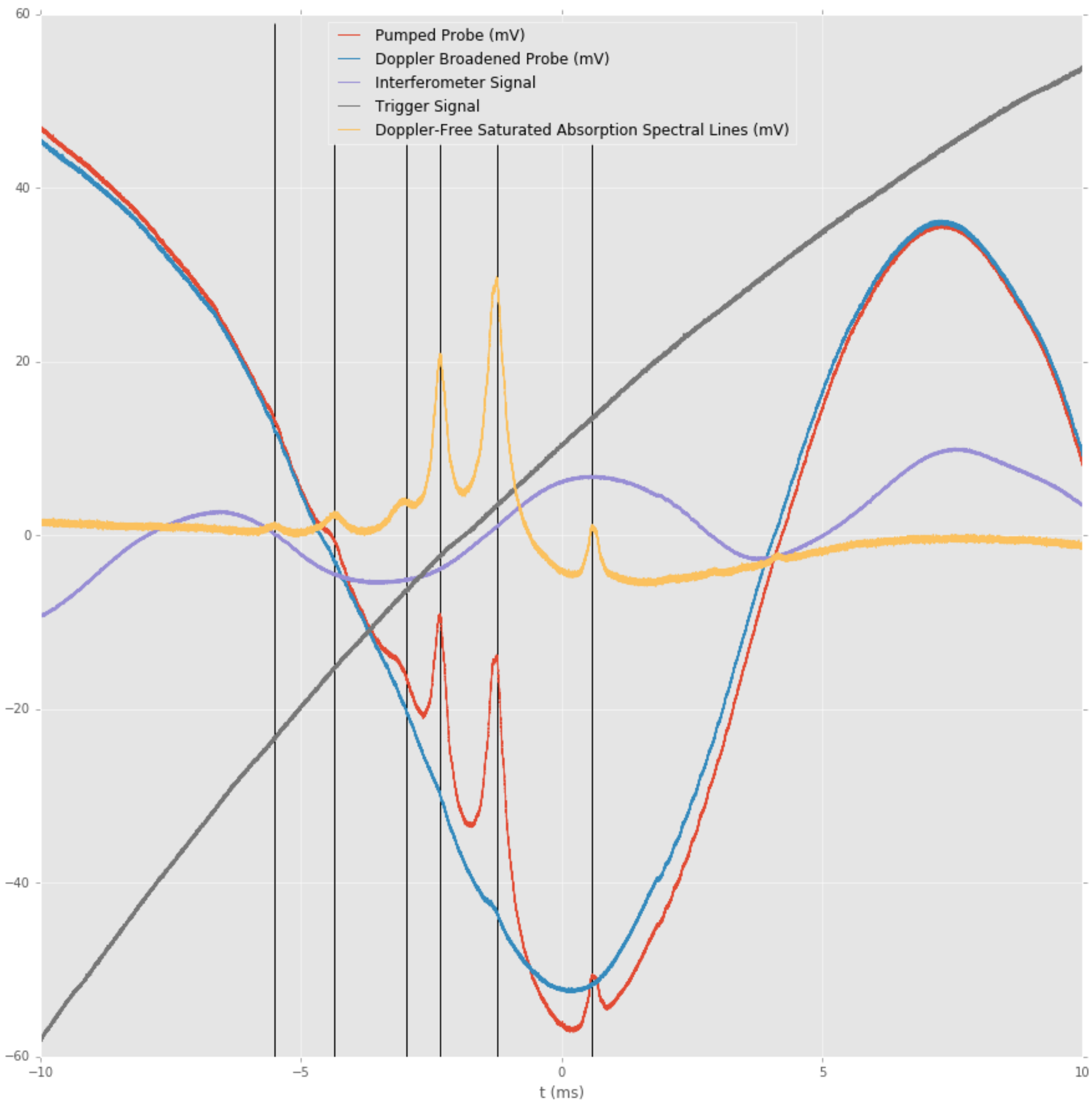

```
Out[151]: (2.6622430857543051,
           0.00050331684231378943,
           0.19964818451897268,
           0.0011003042671316051)
```

### 1.1.6 $^{87}Rb$ Analysis

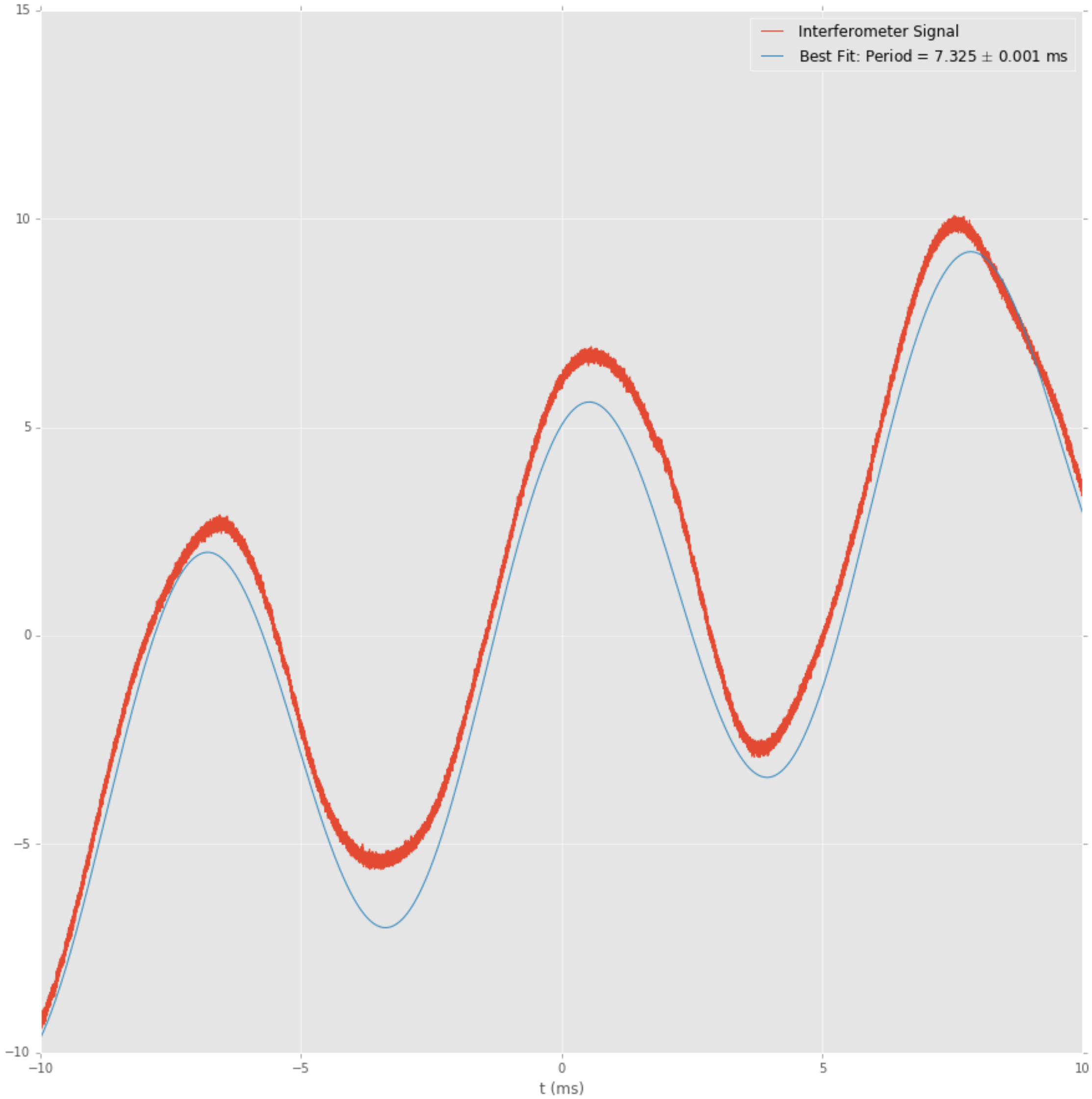
```
In [152]: showPeaks(Rb87_2, 'Doppler-Free Saturated Absorption Spectral Lines (mV)', [[-7,-5], [
          getPeaks(Rb87_2, 'Doppler-Free Saturated Absorption Spectral Lines (mV)', [[-7,-5], [-
```

```
Out[152]: [-5.5116000000000005,
           -4.3613999999999997,
           -2.9914000000000001,
```

```
-2.3254000000000001,
-1.2455999999999998,
0.59139999999999993]
```

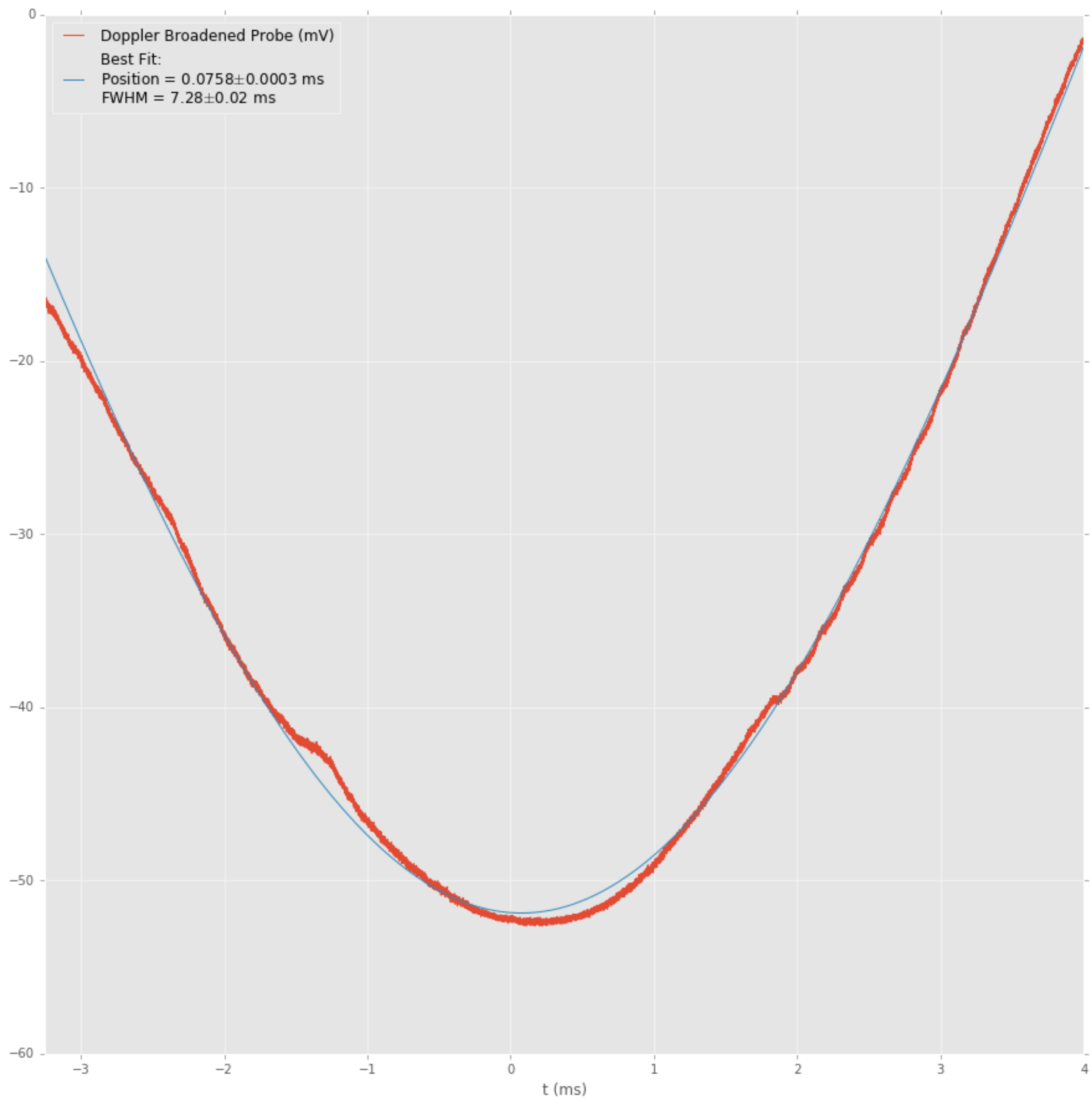



```
In [153]: fringe_time(Rb87_2, 't (ms)', 'Interferometer Signal',[5,7,1,0.5])
```



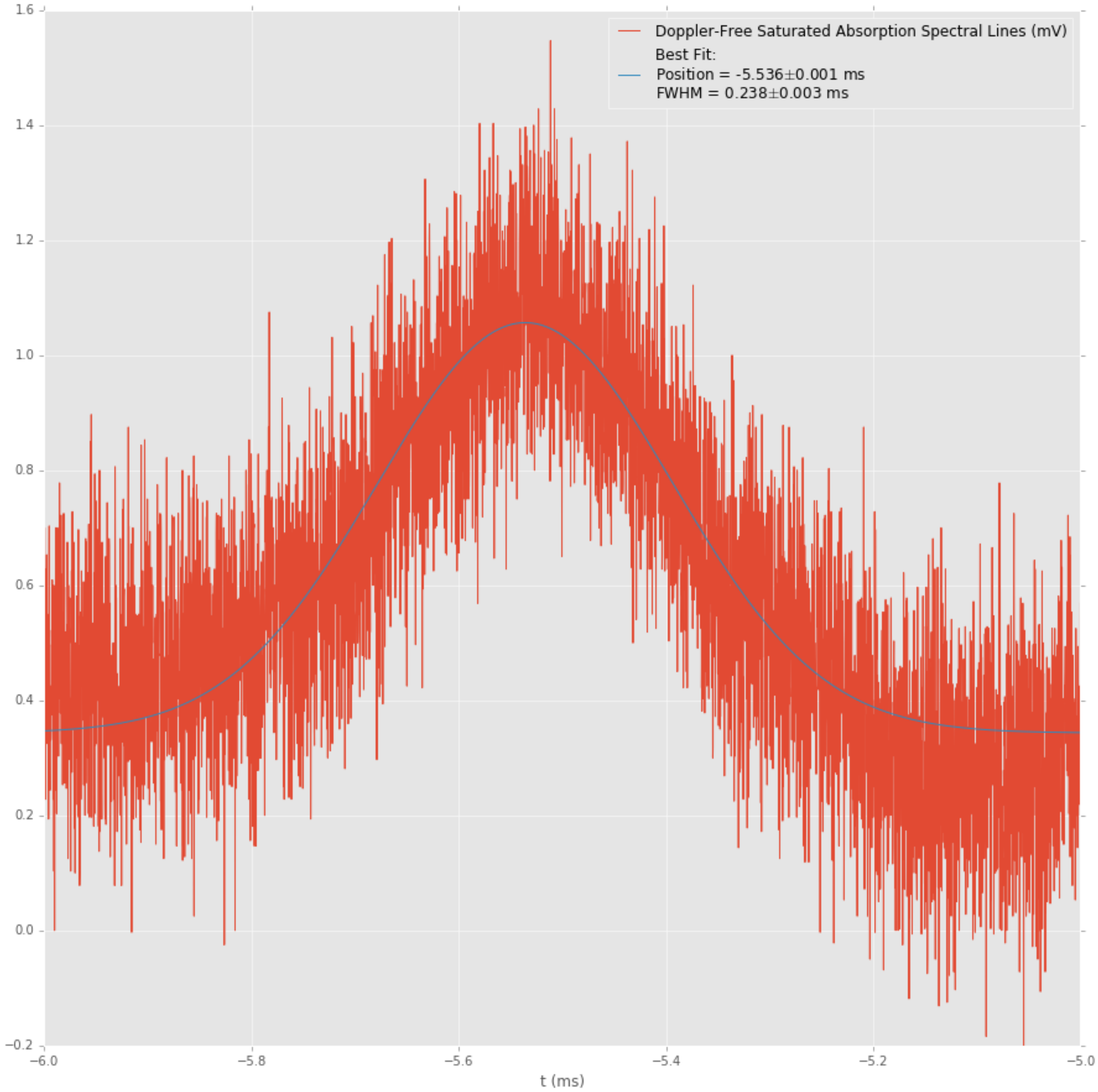

```
Out[153]: (7.324876304171057, 0.0011927719704578419)
```

```
In [154]: FWHM_fit(zoom(Rb87_2,-3.25,4), 't (ms)', 'Doppler Broadened Probe (mV)', [-500,2,0,40]
```



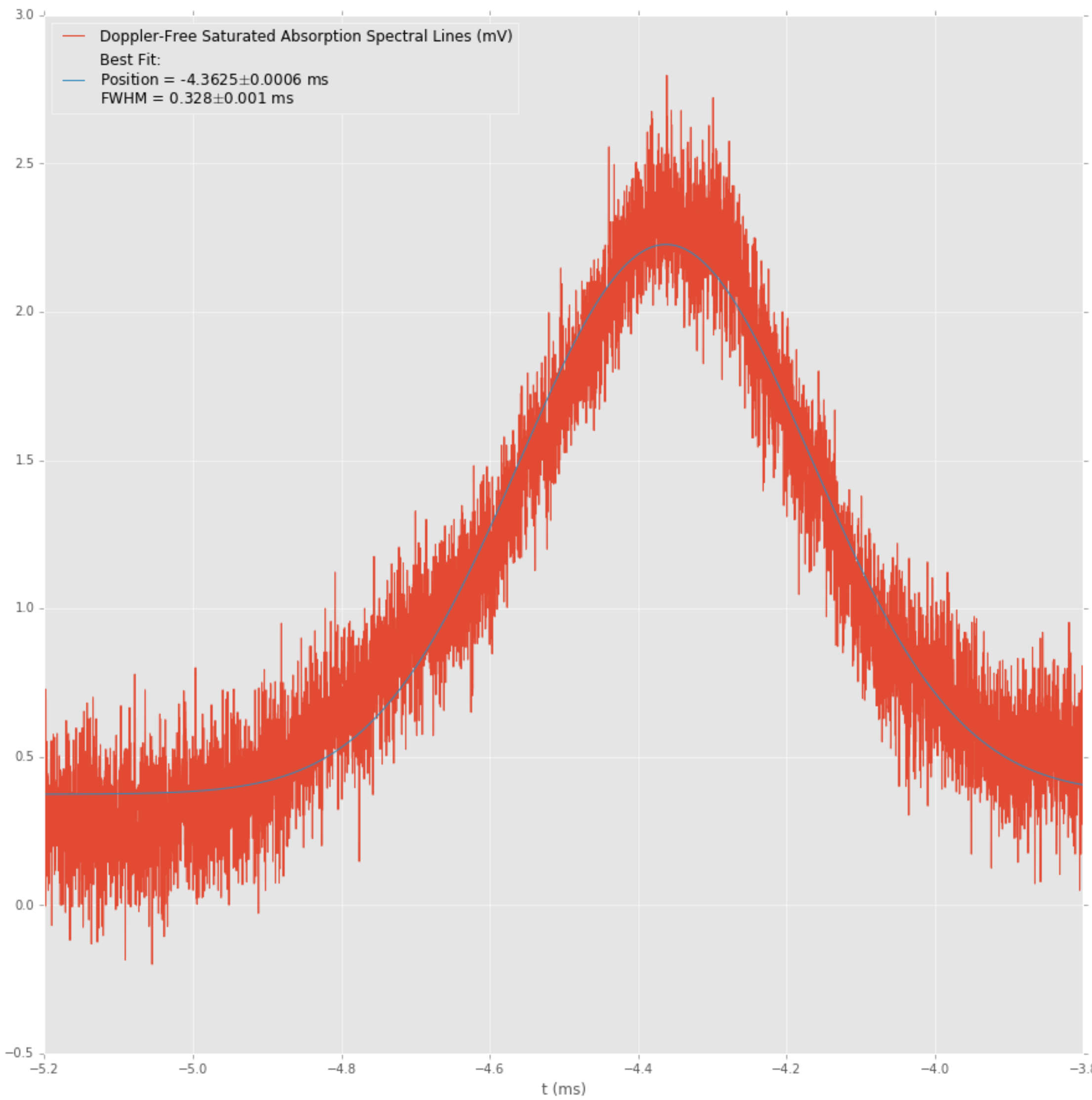

```
Out[154]: (0.075816113498486118,
           0.00028342702036719724,
           7.2813030488090247,
           0.019209769906389999)
```

```
In [155]: z87 = zoom(Rb87_2,-6,-5)
          # ax = z87.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb87_2,-6,-5), 't (ms)', 'Doppler-Free Saturated Absorption Spectral Lin
```



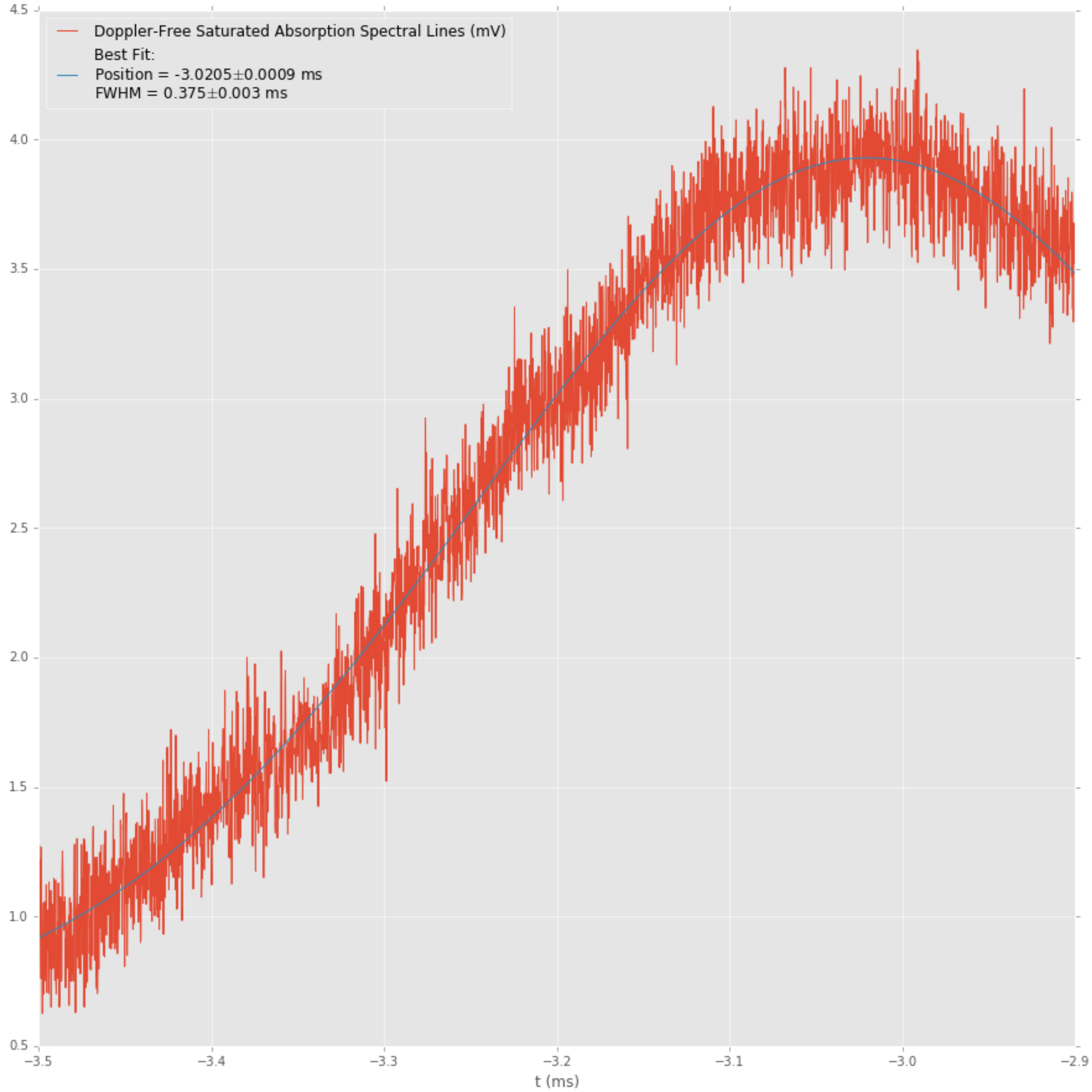

```
Out[155]: (-5.5359573771334496,
           0.0012606076928936331,
           0.23799737351062672,
           0.0029582897774584034)
```

```
In [156]: z87 = zoom(Rb87_2,-5,-3.8)
          # ax = z87.plot(x = 't (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV
          FWHM_fit(zoom(Rb87_2,-5.2,-3.8), 't (ms)', 'Doppler-Free Saturated Absorption Spectral
```



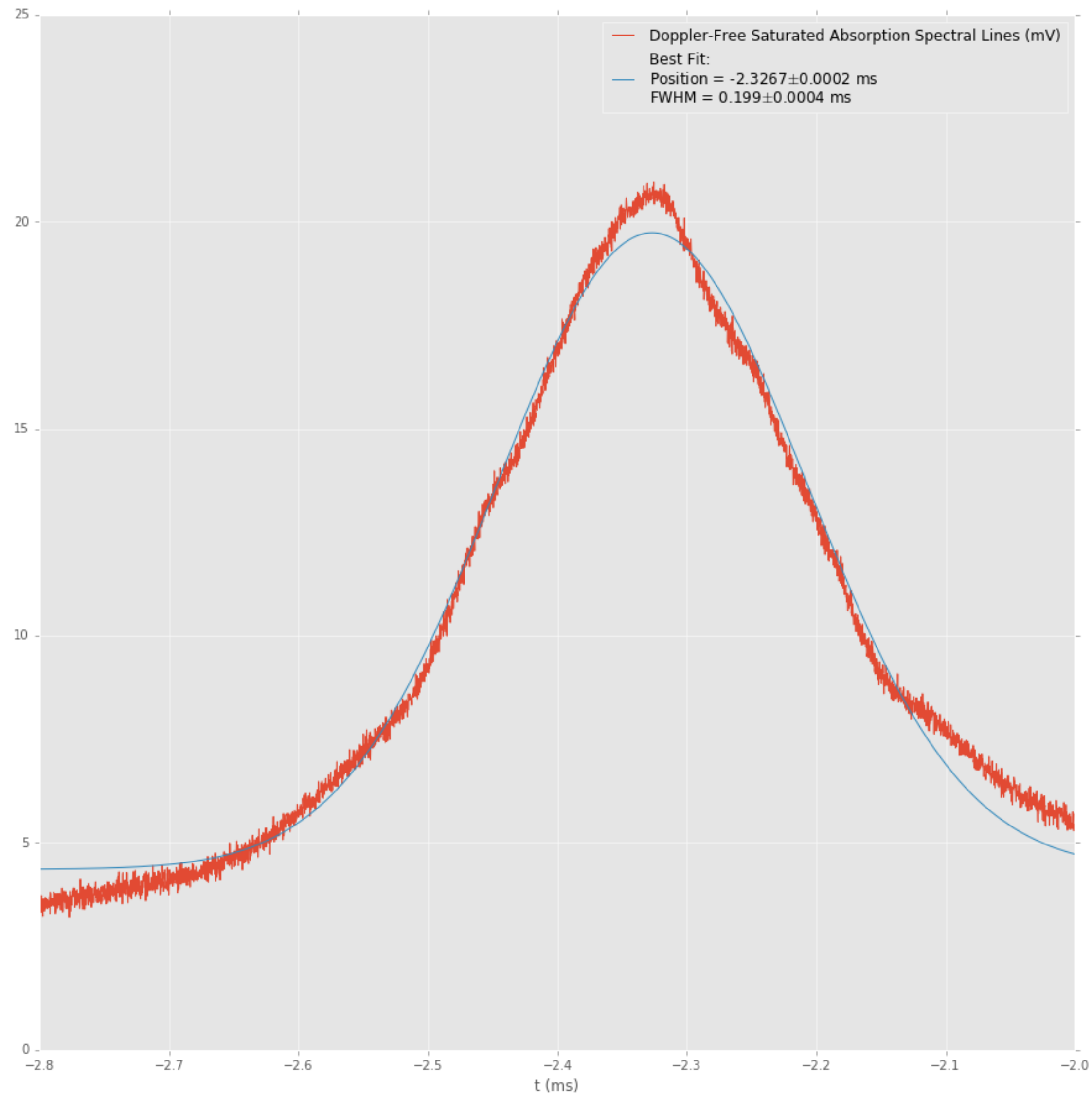

```
Out[156]: (-4.3625331751204923,
           0.00059807911813266061,
           0.32814334733388473,
           0.0013550455086812796)
```

```
In [157]: z87 = zoom(Rb87_2,-3.3,-2.9)
          # ax = z87.plot(x='t (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV)'
          FWHM_fit(zoom(Rb87_2,-3.5,-2.9), 't (ms)', 'Doppler-Free Saturated Absorption Spectral
```

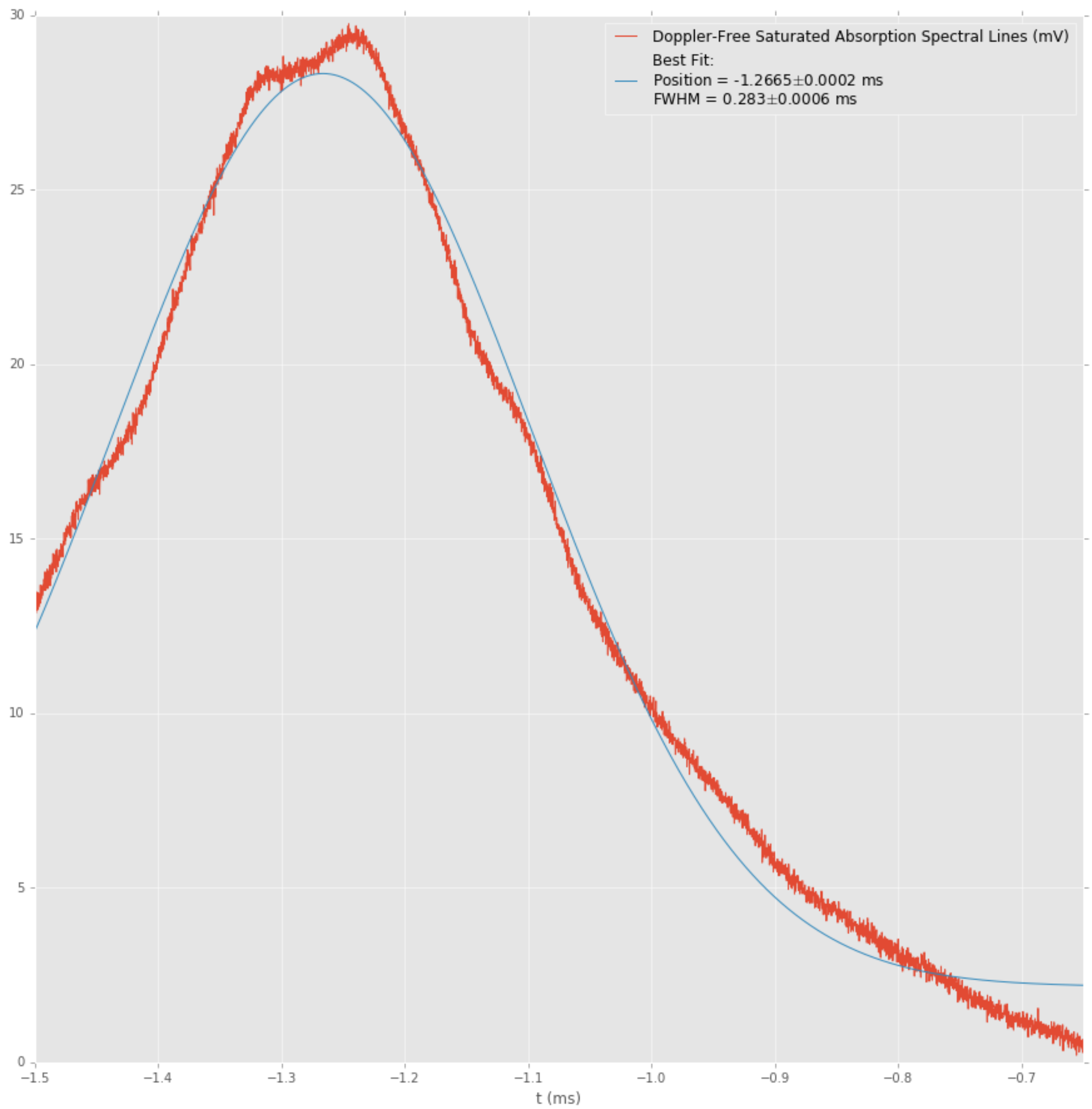



```
Out[157]: (-3.0204864843795929,
           0.00093271416675013059,
           0.37489579875771889,
           0.0031923791529044807)
```

```
In [158]: z87 = zoom(Rb87_2,-2.9,-1.9)
          # ax = z87.plot(x='t (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV)'
          FWHM_fit(zoom(Rb87_2,-2.8,-2), 't (ms)', 'Doppler-Free Saturated Absorption Spectral L
```




```
Out[158]: (-2.326684430016547,
           0.00017984451209943821,
           0.19897189801078535,
           0.00042556743507465389)

In [159]: z87 = zoom(Rb87_2,-1.5,-.65)
          # ax = z87.plot(x='t (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV)'
          FWHM_fit(zoom(Rb87_2,-1.5,-.65), 't (ms)', 'Doppler-Free Saturated Absorption Spectral
```




```
Out[159]: (-1.2665319834384441,
           0.00023073686557129069,
           0.28296791522744619,
           0.00057022626407995177)
```

```
In [160]: z87 = zoom(Rb87_2,0.25,1)
          # ax = z87.plot(x='t (ms)', y= 'Doppler-Free Saturated Absorption Spectral Lines (mV)'
          FWHM_fit(zoom(Rb87_2,0.25,1), 't (ms)', 'Doppler-Free Saturated Absorption Spectral Li
```


```
Out[160]: (0.61293170022883814,
           0.00017752360869099611,
           0.17383123959653421,
           0.00040881572727631934)
```

Use figure 4 in the lab manual to compare strains of 87 Rb. from left to right, we have increasing frequency, so we know a (F=2 to F'=1), b (F=2 to Crossover F'=1,2), etc.

**Sources of Error** * length measurements: Using ruler relies on eyesight and relative position. * interferometer fringes are much wider than linewidth and splitting lines. We can expect uncertainty based on inferring a smaller frequency from a longer one. Because we have 3 peaks, we should be able to have 2 different values for $\Delta\nu$, which we can compare to get a mean with associated error. *

In order to get frequency measurements from our hyperfine structures, we need to compare widths in the spectroscopy data with the wavelength of the michelson interferometer. Since we know what the frequency fringe width should be by measuring the difference in arm lengths of the interferometer:

$$\Delta\nu = \frac{c}{2\Delta L}$$

We can determine the frequency width of two points of interest by comparing their relative widths. If $\Delta t$ is the time difference between two peaks of the interferometer signal, and $\tau$ is the time difference between the two points of interest, then the frequency spacing $f$ between these two points is given by

$$f = \frac{\Delta\nu}{\Delta t}\tau = \frac{c}{2\Delta L\Delta t}\tau$$

We now need to implement a means of measuring $\Delta t$ for each set of data. Since we know that

```
In [161]: def C(F,J,I):
              return F*(F+1) - J*(J+1) - I*(I+1)

          def nu(F,J,I):
              c = C(F,J,I)
              n = (.75*c*(c+1)-I*(I+1)*J*(J+1))/(2*I*(2*I-1)*J*(2*J-1))
              return c, n

          nu(3,1.5,2.5)

Out[161]: (-0.5, -0.55)

In [162]: meter85, meter85Err = fringe_time(Rb85_4, 't (ms)', 'Interferometer Signal',[75,6,-2,0
          meter87, meter87Err = fringe_time(Rb87_2, 't (ms)', 'Interferometer Signal',[5,7,1,0.5
          tPos85, tErr85, FWHM85, FWHM85Err = FWHM_fit(zoom(Rb85_4,-4.75,8), 't (ms)', 'Doppler
          tPos87, tErr87, FWHM87, FWHM87Err = FWHM_fit(zoom(Rb87_2,-3.25,4), 't (ms)', 'Doppler
          tPos85p1, tErr85p1, FWHM85p1, FWHM85Errp1 = FWHM_fit(zoom(Rb85_4,0,.5), 't (ms)', 'Dop
          tPos85p2, tErr85p2, FWHM85p2, FWHM85Errp2 = FWHM_fit(zoom(Rb85_4,0.9,1.35), 't (ms)',
          tPos85p3, tErr85p3, FWHM85p3, FWHM85Errp3 = FWHM_fit(zoom(Rb85_4,1.55,2.1), 't (ms)',
          tPos85p4, tErr85p4, FWHM85p4, FWHM85Errp4 = FWHM_fit(zoom(Rb85_4,2.6,3), 't (ms)', 'Do
          tPos87p1, tErr87p1, FWHM87p1, FWHM87Errp1 = FWHM_fit(zoom(Rb87_2,-6,-5), 't (ms)', 'Do
          tPos87p2, tErr87p2, FWHM87p2, FWHM87Errp2 = FWHM_fit(zoom(Rb87_2,-5.2,-3.8), 't (ms)',
          tPos87p3, tErr87p3, FWHM87p3, FWHM87Errp3 = FWHM_fit(zoom(Rb87_2,-3.5,-2.9), 't (ms)',
          tPos87p4, tErr87p4, FWHM87p4, FWHM87Errp4 = FWHM_fit(zoom(Rb87_2,-2.8,-2), 't (ms)', '
          tPos87p5, tErr87p5, FWHM87p5, FWHM87Errp5 = FWHM_fit(zoom(Rb87_2,-1.5,-.65), 't (ms)',
          tPos87p6, tErr87p6, FWHM87p6, FWHM87Errp6 = FWHM_fit(zoom(Rb87_2,0.25,1), 't (ms)', 'D

In [163]: meter85

Out[163]: 7.7884442789261179

In [164]: def freq_converter(tau, dL, dT, tauErr, LErr, TErr):
              f = c*tau/(2*dL*dT)
              fErr = c/2*(tauErr/(dL*dT) - LErr/(dL**2*dT) - TErr/(dL*dT**2))
              return f, fErr
```

```
In [165]: c*.401E-3/(2*.2940*7.7884442789)
```

```
Out[165]: 26251.12729261857
```

```
In [166]: FWHM85_freq, FWHM85_err = freq_converter(FWHM85, 0.2940, meter85, FWHM85Err, .0002, me
          FWHM87_freq, FWHM87_err = freq_converter(FWHM87, 0.2940, meter87, FWHM87Err, .0002, me
```

```
In [167]: print('FWHM85_freq = {} $\pm$ {}\nFWHM87_freq = {} $\pm$ {}'.format(FWHM85_freq/1E6, F
```

```
FWHM85_freq = 512.8450617443619 $\pm$ 0.2878480077516078
FWHM87_freq = 506.83093437394336 $\pm$ 1.2784512639779346
```

### 1.1.7 Rb splittings

```
In [179]: positions85 = np.array([tPos85p1, tPos85p2, tPos85p3, tPos85p4])
          err85 = np.array([tErr85p1, tErr85p2, tErr85p3 , tErr85p4])
          splittings85 = np.ediff1d(positions85)
          splitErr85 = np.array(err85 + np.roll(err85,-1))[:-1]

          positions87 = np.array([tPos87p1, tPos87p2, tPos87p3, tPos87p4, tPos87p5, tPos87p6])
          err87 = np.array([tErr87p1, tErr87p2, tErr87p3, tErr87p4, tErr87p5, tErr87p6])
          splittings87 = np.ediff1d(positions87)
          splitErr87 = np.array(err87 + np.roll(err87,-1))[:-1]

          splits85 = freq_converter(splittings85, .2940, meter85, splitErr85, .0002, meter85Err)
          splits87 = freq_converter(splittings87, .2940, meter87, splitErr87, .0002, meter87Err)
```

```
In [180]: print(splits85[0]/1E6)
          print(splits85[1]/1E6)
```

```
[ 51.47516571  28.35902429  66.17322756]
[ 0.06676088 -0.02375037  0.00134516]
```

```
In [181]: print(splits87[0]/1E6)
          print(splits87[1]/1E6)
```

```
[  81.67874359   93.41607864   48.2936009    73.7942167   130.82415723]
[ 0.07069139  0.04786765  0.01875552 -0.03010714 -0.03026869]
```

```
In [182]: Rb85_2 = 29.372
          Rb85_3 = Rb85_2 + 63.401
          Rb85_4 = Rb85_3 + 120.640
          rb85 = np.array([Rb85_2, np.mean([Rb85_2,Rb85_3]), Rb85_3, np.mean([Rb85_2,Rb85_4]), n
          print(rb85)
          split85Real = np.ediff1d(rb85)
          print(split85Real)
```

```
[  29.372    61.0725   92.773   121.3925  153.093   213.413 ]
[ 31.7005  31.7005  28.6195  31.7005  60.32   ]
```

```
In [183]: Rb87_1 = 72.218
          Rb87_2 = Rb87_1 + 156.947
          Rb87_3 = Rb87_2 + 266.650
          rb87 = np.array([Rb87_1, np.mean([Rb87_1,Rb87_2]), Rb87_2, np.mean([Rb87_1,Rb87_3]), n
          print(rb87)
          split87Real = np.ediff1d(rb87)
          print(split87Real)
```

```
[  72.218    150.6915  229.165    284.0165  362.49     495.815 ]
[  78.4735   78.4735   54.8515   78.4735  133.325 ]
```

```
In [184]: equations85 = np.array([[3/2,-9/40],[3/2,-9/40],[1/2,5/8],[3/2,-9/40],[2,2/5]])
          equations85adj = np.array([[1/2,5/8],[3/2,-9/40],[2,2/5]])
          ansReal85 = np.array(split85Real)
          ansMeas85 = np.array(splits85[0]/1E6)

          equations87 = np.array([[1,-1/2],[1,-1/2],[1/2,1],[1,-1/2],[3/2,1/2]])
          ansReal87 = np.array(split87Real)
          ansMeas87 = np.array(splits87[0]/1E6)
```

```
In [185]: def ABsolver(equations, ans):
              A = np.array([])
              B = np.array([])
              for i in np.arange(len(equations)):
                  for j in np.arange(len(equations)):
                      try:
                          result = np.linalg.solve(np.array([equations[i],equations[j]]),np.arra
                          A = np.append(A,result[0])
                          B = np.append(B,result[1])
                      except np.linalg.LinAlgError:
                          pass#print('Singular matrix {}\n'.format([equations[i],equations[j]]))
              return A, B
```

### 1.1.8 $^{85}$Rb Expected

```
In [186]: A85real, B85real = ABsolver(equations85, ansReal85)
          print(A85real)
          print(B85real)
          print(np.mean(A85real), np.std(A85real))
          print(np.mean(B85real), np.std(B85real))
```

```
[ 25.00209524  25.00209524  25.00209524  25.00209524  25.00209524
  25.00209524  25.00209524  25.00209524  25.00209524  25.00209524
  25.00209524  25.00209524  25.00209524  25.00209524]
```

```
[ 25.78952381  25.78952381  25.78952381  25.78952381  25.78952381
  25.78952381  25.78952381  25.78952381  25.78952381  25.78952381
  25.78952381  25.78952381  25.78952381  25.78952381]
25.0020952381 7.47639149962e-15
25.7895238095 1.98716187129e-14
```

#### 1.1.9 $^{85}$Rb Measured

```
In [187]: A85meas, B85meas = ABsolver(equations85adj, ansMeas85)
          print(A85meas)
          print(B85meas)
          print(np.mean(A85meas), np.std(A85meas))
          print(np.mean(B85meas), np.std(B85meas))
```

```
[ 27.91076425  19.77923899  27.91076425  24.98341516  19.77923899
  24.98341516]
[ 60.03165373  66.53687394  60.03165373  40.51599312  66.53687394
  40.51599312]
24.2244728013 3.36277870019
55.6948402631 11.0567482569
```

#### 1.1.10 $^{87}$Rb Expected

```
In [188]: A87real, B87real = ABsolver(equations87,ansReal87)
          print(A87real)
          print(B87real)
          print(np.mean(A87real), np.std(A87real))
          print(np.mean(B87real), np.std(B87real))
```

```
[ 84.7194  84.7194  84.7194  84.7194  84.7194  84.7194  84.7194  84.7194
  84.7194  84.7194  84.7194  84.7194  84.7194  84.7194]
[ 12.4918  12.4918  12.4918  12.4918  12.4918  12.4918  12.4918  12.4918
  12.4918  12.4918  12.4918  12.4918  12.4918  12.4918]
84.7194 1.69852201187e-14
12.4918 6.00518216294e-15
```

#### 1.1.11 $^{87}$Rb Measured

```
In [189]: A87meas, B87meas = ABsolver(equations87,ansMeas87)
          print(A87meas)
          print(B87meas)
          print(np.mean(A87meas), np.std(A87meas))
          print(np.mean(B87meas), np.std(B87meas))
```

```
[ 84.66043523  85.00116033  94.05030328  89.69609435  84.66043523
  94.05030328  78.35281372  85.34188543  78.35281372  81.84734957
```

```
  85.00116033  89.69609435  85.34188543  81.84734957]
[  5.96338329   6.64483348   1.26844926  -7.43996859   5.96338329
   1.26844926   9.11719404   5.62265819   9.11719404  16.10626574
   6.64483348  -7.43996859   5.62265819  16.10626574]
85.5642917019 4.72211812301
5.32611648692 6.67890644177
```