Update CHART Calibration

Adam Beardsley & Carly Fitzgerald

August, 2025

The existing gain and noise calibration used in CHART's analysis tutorial (and described in Berkhout et al, 2024) works well when the noise is flat and the model signal is negligible within the region we estimate the noise level (-100 km/s $\leq v \leq -75$ km/s). However, we have seen several cases where the model is non-zero in this region, and we suspect the noise and/or gain may have frequency dependence.

Here we explore two additional calibration methods and compare with the original via a reduced chi-squared calculation and visual inspection. The calibrations are:

1. Calibration as described in Berkhout et al (2024) where constant noise is estimated as the average value within -100 km/s $\leq v \leq -75$ km/s and the gain is determined by matching the peak of the data with the peak of the model within the range -100 km/s $\leq v \leq -100$ km/s.
2. Similar to (1), but noise is now given a linear term (i.e., $n(f) = n_0 + n'f$). The noise level is estimated in two ranges: -150 km/s $\leq v \leq -100$ km/s and 100 km/s $\leq v \leq 150$ km/s, and a point-slope formula is used. Gain is estimated in the same way as (1).
3. We use Orthogonal Distance Regression (ODR) to perform a least squares fit for a constant gain, constant noise, and noise slope.

We use the tutorial data sets plus a few "problematic" data sets (one of Carly's, three from the Summer 2025 workshops) to test and compare these strategies. We find that (2) outperforms (1) in all cases, and significantly addresses the issue in the problematic datasets. Interestingly (3) does slightly better than (2) with the tutorial data, but fails miserably on the problematic sets. However, when we first apply (2), then apply (3), it does improve the fit.

Most of the code below is a stripped-down version of the tutorial, up to the point of calibration.

In [1]:
```
%matplotlib widget
import numpy as np
import matplotlib.pyplot as plt
import chart
from astropy import units as u
from astropy.coordinates import SpectralCoord, EarthLocation, SkyCoord, AltA
from astropy.time import Time
import pandas as pd
from ipywidgets import interact, FloatSlider, Dropdown
import scipy.constants as const
from scipy.interpolate import CubicSpline
from scipy.ndimage import median_filter
import scipy.odr as odr
```

In [2]:
```
# A couple constants and useful functions for later.
f_e = 1.420405751768 * u.GHz  # Rest frequency of HI hyperfine transition
speed_of_light = const.speed_of_light * (u.meter / u.second)

def freq2vel(freq, rest=f_e):
    """
    Calculates velocity from measured frequency via doppler shift.

    :param freq: array of frequency quantities (including units)
    :param rest (optional): Rest frequency, defaults to 1.42 GHz
    :returns vel: velocity inferred by doppler shift
    """
    return (rest - freq) * speed_of_light / freq

def get_gal_coords(longitude, latitude, elevation, time,
                   altitude, azimuth, return_vadj=False):
    """
    Determines galactic coordinates of an observation and
    optionally also calculates the velocity adjustment
    for the Local Standard of Rest.

    :param latitude: latitude in degrees
    :param longitude: longitude in degrees
    :param elevation: elevation in meters
    :param time: observation time in UTC format string
    :param altitude: altitude in degrees
    :param azimuth: azimuth in degrees
    :param return_vadj (optional): If set to True, returns the
                            velocity adjustment for the
                            Local Standard of Rest in addition
                            to the galactic coordinates (l, b).
                            If False (default) only returns (l, b).
    """

    loc = EarthLocation(lat=latitude*u.deg, lon=longitude*u.deg, height=elev
    altaz = AltAz(obstime=Time(time), location=loc, alt=altitude*u.deg, az=a
    skycoord = SkyCoord(altaz.transform_to(ICRS()))
    if not return_vadj:
        return skycoord.galactic
    loc = loc.get_itrs(obstime=Time(time)) #To ITRS frame, makes Earth stati
    frequency = SpectralCoord(f_e, observer=loc, target=skycoord) #Shift exp
    f_shifted = frequency.with_observer_stationary_relative_to('lsrk') #corr
    f_shifted = f_shifted.to(u.GHz)
    v = -freq2vel(f_shifted, f_e)
    v_adj = v.to(u.km/u.second)
    return skycoord.galactic, v_adj

def find_array_with_number(freqs, pointing, number):
    for k_index, k in enumerate(freqs[pointing]):
        if np.any((k[:-1] >= number) & (number >= k[1:])):
            return k_index, k
    return None, None

def reduced_chisquare(data, model, nparams, mfs = 15):
    """
    determines reduced chi square of our data

    :param data: callibrated data we collected
```

```
    :param model: model data to compare with data collected
    :param nparams: the number of parameters used when callibrating the data
    :param mfs: median filter size. Optional, default = 15
    """
    filtered_data = median_filter(data, mfs)
    diff = (data)-(model)
    stdd = np.std(data - filtered_data)


    X= np.sum(((diff)/(stdd))**2)
    d =( X/(len(data) - nparams))

    return d


def average_overlapping(x1, y1, x2, y2, x3, y3):
    """
    Averages the y values where the x values are shared between
    arrays and keeps y values for x values that are not shared.
    Assumes an x value is shared by at most two arrays.

    :param x1: First x array
    :param y1: First y array
    :param x2: Second x array
    :param y2: Second y array
    :param x3: Third x array
    :param y3: Third y array
    :return: Tuple of combined x values and averaged/kept y values
    """
    # Find the unique x values in both arrays
    unique_x = np.union1d(x1, x2)
    unique_x = np.union1d(unique_x, x3)

    # Create an array to store the averaged/kept y values
    avg_y = np.zeros(unique_x.shape)

    # Iterate over the unique x values
    for i in range(len(unique_x)):
        # Find the indices of the current x value in the two x arrays
        ind1 = np.where(x1 == unique_x[i])[0]
        ind2 = np.where(x2 == unique_x[i])[0]
        ind3 = np.where(x3 == unique_x[i])[0]

        # If the current x value is arrays 1 and 2
        if len(ind1) > 0 and len(ind2) > 0:
            # Compute the average of the two corresponding y values
            avg_y[i] = (y1[ind1[0]] + y2[ind2[0]]) / 2
        # If the current x value is only in the first array
        elif len(ind1) > 0:
            # Keep the corresponding y value from the first array
            avg_y[i] = y1[ind1[0]]
        # If the current x value is in arrays 2 and 3
        elif len(ind2) > 0 and len(ind3) > 0:
            # Compute the average of the two corresponding y values
            avg_y[i] = (y2[ind2[0]] + y3[ind3[0]]) / 2
        # If the current x value is only in the second array
        elif len(ind2) > 0:
            # Keep the corresponding y value from the second array
            avg_y[i] = y2[ind2[0]]
        # If the current x value is only in the third array
```

```
        # If the current x value is only in the third array
        elif len(ind3) > 0:
            # Keep the corresponding y value from the second array
            avg_y[i] = y3[ind3[0]]

    return unique_x, avg_y


def cal_model(B, x):

    '''
    Model for how our parameters contaminate our data.
    B is a vector of the fit parameters plus the template
        B[0] is the gain
        B[1] is the linear noise term
        B[2] is the constant noise term
    x is an array of the x values

    Returns an array of template modified by the parameters
    '''
    return B[0] * model + B[1] * x + B[2]
```

In [3]:

```
### Change these variables to use your own data ###
data_dir = '/data/'
paths = ['abeardsley_Winona-HS-Park_2022.10.08_1_6.12_pm',
         'abeardsley_Winona-HS-Park_2022.10.8_2_6.25_pm',
         'abeardsley_Winona-HS-Park_2022.10.8_3_6.30_pm',
         'abeardsley_Winona_2023.6.19_1_8.16_am',
         'krosok_lakesioux_2025.07.30_7_10.20_pm',
        'dkordahl_Winona_field_2025.07.25_5_10.00_pm',
        'kledbetter_DacotaField.26.30_2025.07.25_1_10.06_pm']

ntrials = len(paths)
stack_figsize = (6, ntrials * 2.5)
single_figsize = (6, 4)

data = []
mdata = []
bps = []  # bandpasses

for i in range(ntrials):
    d, m = chart.analysis.read_run(directory=data_dir + paths[i])
    d = np.array(d)
    data.append(d)
    mdata.append(m)
    # Rough estimate for bandpass
    nchans = m[0]['vector_length']
    levels = np.median(d[:, :, nchans // 4:(-nchans // 4)], axis=(1, 2))
    rescaled = d / levels.reshape(-1, 1, 1)
    bp = np.median(rescaled, axis=(0, 1))
    bps.append(bp)
```

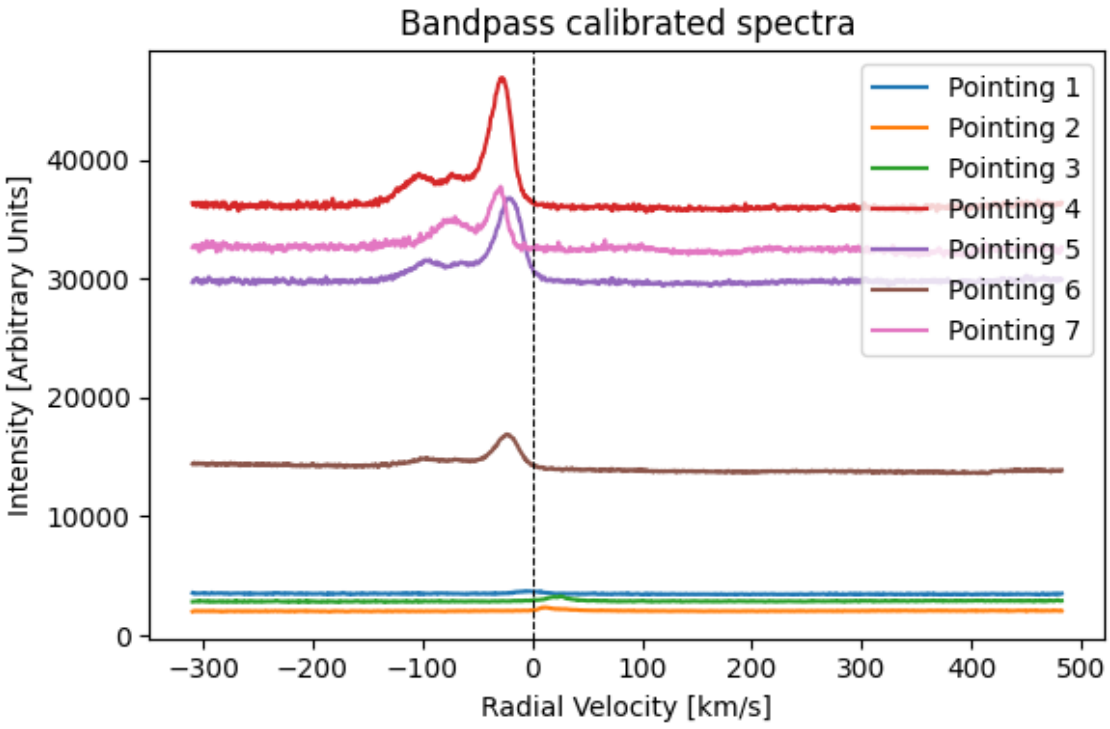
In [4]:
```
ntrials = len(data)
spectra = [[] for _ in range(ntrials)]
freqs = [[] for _ in range(ntrials)]
nremove = nchans // 16

for pointing in range(ntrials):
    for d, m in zip(data[pointing], mdata[pointing]):
        spectrum = np.mean(d, axis=0)
        spectrum = 10*np.log10(spectrum)
        spectrum = spectrum[nremove:-nremove]
        frequencies = ((np.arange(m['vector_length']) - m['vector_length'] /
                        * m['samp_rate'] / m['vector_length'] + m['freque
        frequencies = 1e-9 * frequencies[nremove:-nremove]
        spectra[pointing].append(spectrum)
        freqs[pointing].append(frequencies)
```

In [5]:
```
ntrials = len(data)
spectra = [[] for _ in range(ntrials)]
freqs = [[] for _ in range(ntrials)]
nremove = nchans // 16

for pointing in range(ntrials):
    for d, m in zip(data[pointing], mdata[pointing]):
        spectrum = np.mean(d, axis=0) /bps[pointing]
        spectrum = 10*np.log10(spectrum)
        spectrum = spectrum[nremove:-nremove]
        frequencies = ((np.arange(m['vector_length']) - m['vector_length'] /
                        * m['samp_rate'] / m['vector_length'] + m['freque
        frequencies = 1e-9 * frequencies[nremove:-nremove]
        spectra[pointing].append(spectrum)
        freqs[pointing].append(frequencies)

    for k in range(len(spectra[pointing]) - 1):
        spec1 = spectra[pointing][k]
        spec2 = spectra[pointing][k + 1]
        freq1 = freqs[pointing][k]
        freq2 = freqs[pointing][k + 1]
        ncommon = np.sum([1 if f in freq2 else 0 for f in freq1])
        spec2 += np.median(spec1[-ncommon:]) - np.median(spec2[:ncommon])
        spectra[pointing][k + 1] = spec2
```

In [6]:
```
velocities = []

for pointing in range(ntrials):
    velocities.append([])
    for k in range(len(freqs[pointing])):
        velocities[pointing].append(freq2vel(freqs[pointing][k] * u.GHz))
```

In [7]:
```
plt.figure(figsize=single_figsize)
unique_vs = []
avg_spectra = []
for pointing in range(ntrials):
    # First find the index of the tuning that contains zero velocity
    k_index, result = find_array_with_number(velocities, pointing, 0)

    x1 = velocities[pointing][k_index - 1]
    x2 = velocities[pointing][k_index]
    x3 = velocities[pointing][k_index + 1]

    y1 = np.array(10**(spectra[pointing][k_index - 1] / 10))
    y2 = np.array(10**(spectra[pointing][k_index] / 10))
    y3 = np.array(10**(spectra[pointing][k_index + 1] / 10))
    unique_v, avg_y = average_overlapping(x1, y1, x2, y2, x3, y3)
    unique_vs.append(unique_v)
    avg_spectra.append(avg_y)

    plt.plot(unique_v.to(u.km / u.s), avg_y,
             label = f'Pointing {pointing+1}')

plt.axvline(0, color ='k', linestyle='--', linewidth=0.75)
plt.ylabel('Intensity [Arbitrary Units]')
plt.legend(loc='best')
plt.xlabel('Radial Velocity [km/s]')
plt.title('Bandpass calibrated spectra')

plt.tight_layout()
```

Figure



In [8]:
```
### Pointing Dict setup
Observation_longitude = -91.64419  # degrees
Observation_latitude = 44.04672  # degrees
Observation_elevation = 200  # meters
Observation_azimuths = [90, 180, 180,292, 75, 88, 30]  # degrees
Observation_altitudes = [86, 53, 17,48, 59, 55, 26]  # degrees
Observation_datetimes = ['2022-10-8T23:12:00', '2022-10-8T23:25:00', '2022-1
```

In [9]:
```
galactic_ls = []
galactic_bs = []
v_adjustments = []

for n in range(ntrials):
    galactic_coords, v_adj = get_gal_coords(Observation_longitude, Observati
                                            Observation_elevation, Observati
                                            Observation_altitudes[n], Observ
                                            return_vadj=True)
    l = galactic_coords.l
    b = galactic_coords.b
    galactic_ls.append(l.degree)
    galactic_bs.append(b.degree)
    v_adjustments.append(v_adj)
    unique_vs[n] -= v_adj
```

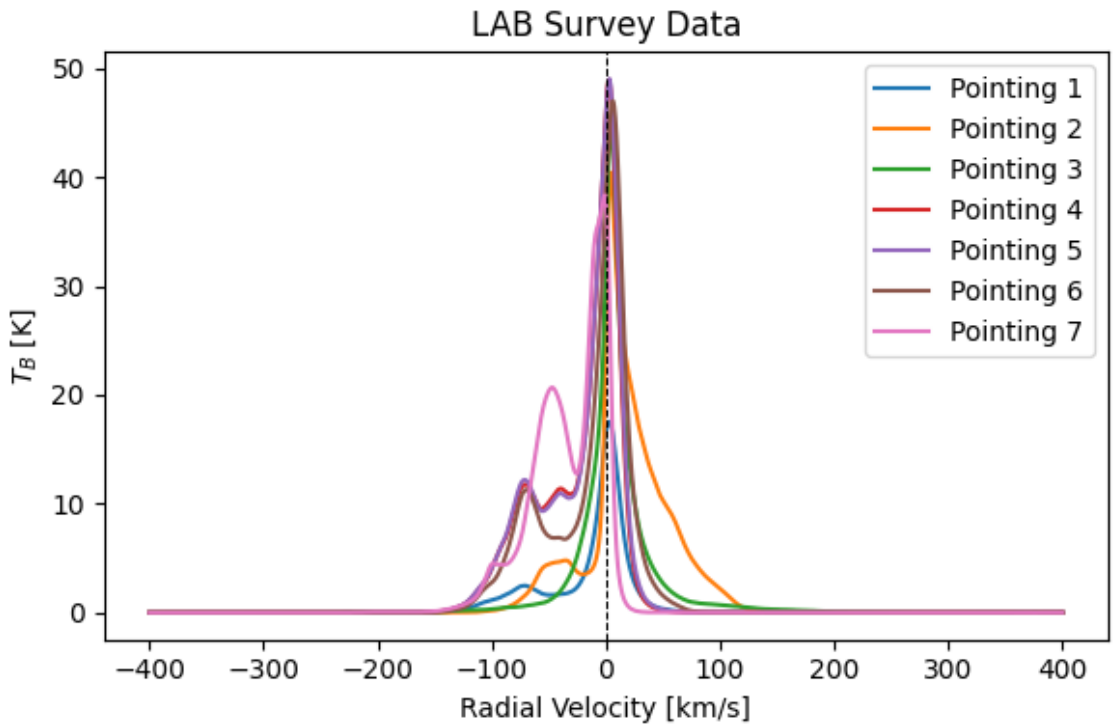
```
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
WARNING: NoVelocityWarning: No velocity defined on frame, assuming (0., 0.,
0.) km / s. [astropy.coordinates.spectral_coordinate]
WARNING: NoDistanceWarning: Distance on coordinate object is dimensionless,
an arbitrary distance value of 1000000.0 kpc will be set instead. [astrop
y.coordinates.spectral_coordinate]
```

In [10]:

```
comps = ['/data/abeardsley_Winona-HS-Park_2022.10.08_1_6.12_pm.txt',
          '/data/abeardsley_Winona-HS-Park_2022.10.8_2_6.25_pm.txt',
          '/data/abeardsley_Winona-HS-Park_2022.10.8_3_6.30_pm.txt',
         '/data/labsurvey.abeardsley_Winona_2023.6.19_1_8.16_am.txt',
         '/data/krosok_l83_b1.txt',
         '/home/dkordahl/notebooks/test_data/spectrum_sim_1000pm.txt',
         '/home/kledbetter/notebooks/models/spectrum_124_-3.txt']
```

In [11]:
```
plt.figure(figsize=single_figsize)
model_vs = []
model_Ts = []
for pointing in range(ntrials):
    sh_comp = pd.read_table(comps[pointing],skiprows=[0,1,2,3],names=['v_lsr
    model_vs.append(np.array(sh_comp['v_lsr']))
    model_Ts.append(np.array(sh_comp['T_B']))

    plt.plot(model_vs[pointing], model_Ts[pointing],
         label = f'Pointing {pointing+1}')

plt.axvline(0, color ='k', linestyle='--', linewidth=0.75)
plt.ylabel('$T_B$ [K]')
plt.legend(loc='best')
plt.xlabel('Radial Velocity [km/s]')

plt.title('LAB Survey Data')
plt.tight_layout()
```

Figure



Calibration time

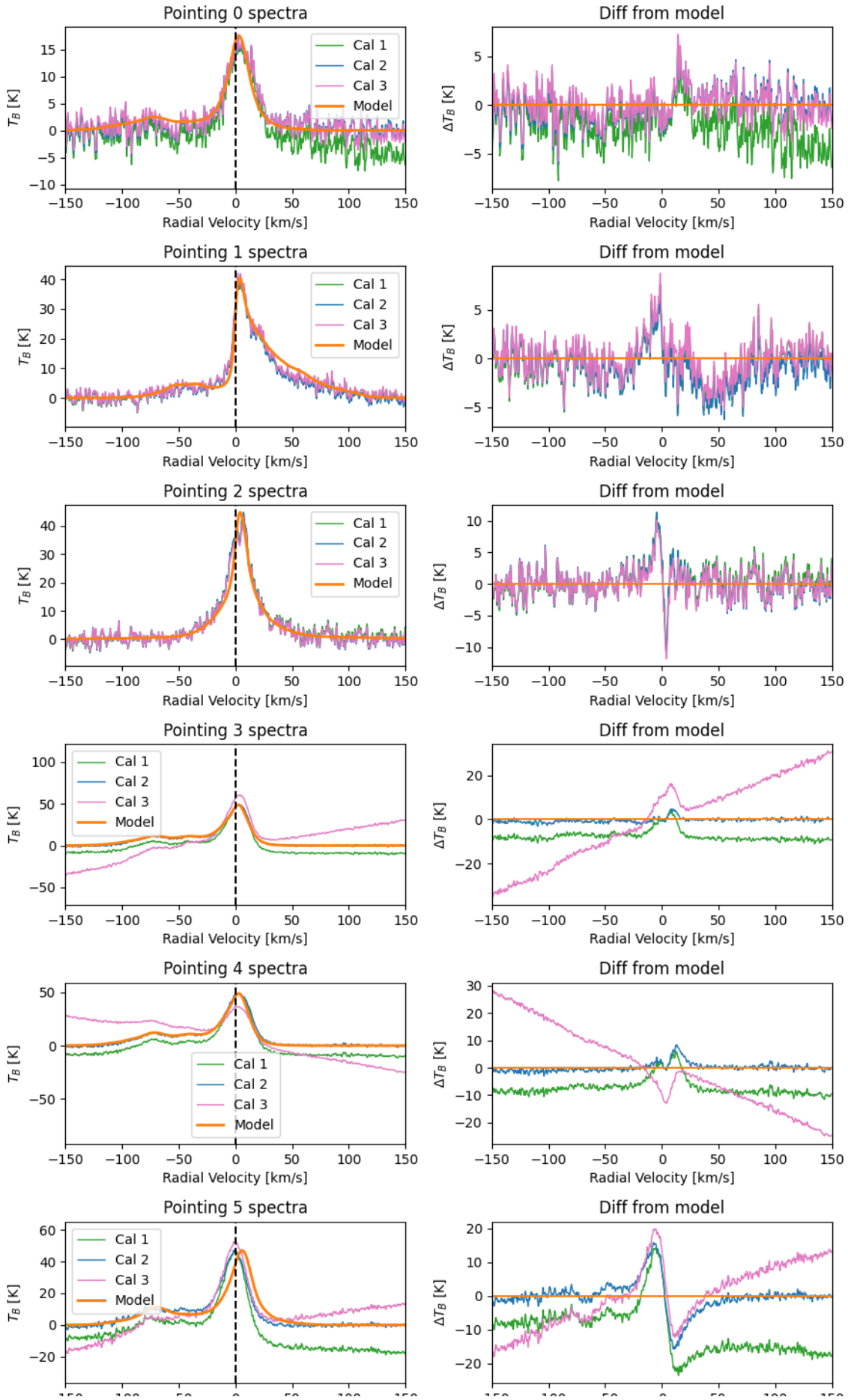
With all the setup complete, we now attempt our three different calibration methods.

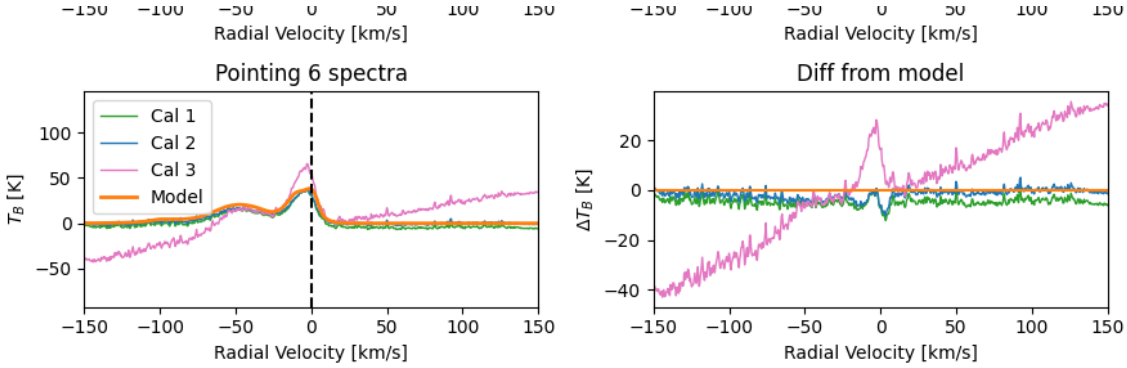
In [12]:
```
fig, axs = plt.subplots(nrows=ntrials, ncols=2, figsize=(1.5*stack_figsize[0
if ntrials == 1: axs = [axs]

for pointing in range(ntrials):

    # Interpolate model -- Used for calculating chi-sq
    PosIn = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-150))
    NegIn = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+150))
    model = CubicSpline(model_vs[pointing],model_Ts[pointing])(unique_vs[poi

    # Calibration 1 - currently in tutorial
    vneg75 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+75))
    vneg100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+100))
    v100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-100))

    noise = np.mean(avg_spectra[pointing][vneg100:vneg75])
    gain = max(avg_spectra[pointing][vneg100:v100]-noise)/(max(model_Ts[poir

    cal_data = (avg_spectra[pointing] - noise) / gain

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:g
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 2)
    print(f'Pointing {pointing}, method 1: chi^2={chi_sq}')

    # Calibration 2 - Noise slope a la Carly
    vpos100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+100))
    vpos150 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+150))
    noisepos = np.mean(avg_spectra[pointing][vpos150:vpos100])
    vneg100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-100))
    vneg150 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-150))

    noiseneg = np.mean(avg_spectra[pointing][vneg100:vneg150])
    noise_slope = (noiseneg-noisepos)/(250)

    noise = noise_slope*(unique_vs[pointing].to(u.km/u.s).value+125)+ noisep
    gain = max(avg_spectra[pointing]-noise)/(max(model_Ts[pointing]))
    cal_data = (avg_spectra[pointing] - noise) / gain

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:b
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 3)

    print(f'Pointing {pointing}, method 2: chi^2={chi_sq}')


    # Calibration 3 - ODR fit

    odr_model = odr.Model(cal_model)

    mydata = odr.RealData(unique_vs[pointing].to(u.km/u.s).value[NegIn:PosIr
    myodr = odr.ODR(mydata, odr_model, beta0=[1, 0, 0], ifixx=np.ones(len(av
    myoutput = myodr.run()


    gain = myoutput.beta[0]
    noise_slope = myoutput.beta[1]
    noisepos = myoutput.beta[2]
```

```
    noise = noise_slope*(unique_vs[pointing].to(u.km/u.s).value)+ noisepos

    cal_data = (avg_spectra[pointing] - noise) / gain

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:p
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 3)
    print(f'Pointing {pointing}, method 3: chi^2={chi_sq}')

    # Finish up plot
    axs[pointing, 0].plot(model_vs[pointing],model_Ts[pointing],'tab:orange'
    axs[pointing, 1].axhline(0, color='tab:orange')
    axs[pointing, 0].axvline(0, color ='k', linestyle='--')

    axs[pointing, 0].set_xlim(-150,150)
    axs[pointing, 0].set_ylabel('$T_B$ [K]')
    axs[pointing, 0].legend(loc='best')
    axs[pointing, 0].set_xlabel('Radial Velocity [km/s]')
    axs[pointing, 0].set_title(f'Pointing {pointing} spectra')
    axs[pointing, 1].set_xlim(-150,150)
    axs[pointing, 1].set_ylabel('$\Delta T_B$ [K]')
    axs[pointing, 1].set_xlabel('Radial Velocity [km/s]')
    axs[pointing, 1].set_title(f'Diff from model')



plt.tight_layout()
```

```
Pointing 0, method 1: chi^2=3.873567198321034
Pointing 0, method 2: chi^2=1.7188630038346382
Pointing 0, method 3: chi^2=1.6182321466344636
Pointing 1, method 1: chi^2=2.6484634065470316
Pointing 1, method 2: chi^2=2.940656238325757
Pointing 1, method 3: chi^2=2.4712027540020864
Pointing 2, method 1: chi^2=2.5768533978437107
Pointing 2, method 2: chi^2=2.474322096886441
Pointing 2, method 3: chi^2=2.4086027124181064
Pointing 3, method 1: chi^2=228.82888663615586
Pointing 3, method 2: chi^2=6.089102697263134
Pointing 3, method 3: chi^2=1178.336344937711
Pointing 4, method 1: chi^2=189.2535652809135
Pointing 4, method 2: chi^2=11.267831927042755
Pointing 4, method 3: chi^2=2018.753507024159
Pointing 5, method 1: chi^2=263.11960344699685
Pointing 5, method 2: chi^2=56.81883678424331
Pointing 5, method 3: chi^2=176.486082691913
Pointing 6, method 1: chi^2=21.903584949610032
Pointing 6, method 2: chi^2=6.987235716213565
Pointing 6, method 3: chi^2=169.5158736359761
```

Figure


Pointing 0 spectra
Diff from model
Pointing 1 spectra
Diff from model
Pointing 2 spectra
Diff from model
Pointing 3 spectra
Diff from model
Pointing 4 spectra
Diff from model
Pointing 5 spectra
Diff from model
Cal 1
Cal 2
Cal 3
Model
T_B [K]
ΔT_B [K]
Radial Velocity [km/s]



We can see that (2) slightly improves Pointing 0, doesn't change Pointings 1 or 2 much (which already have good chi^2's), and it greatly improves Pointings 3-6 (the problematic data sets).

Method (3) closely follows (2) until the problematic sets where it fails miserably. We tried using (2) as the initial guess for the fit, but it still converged on the same terribly wrong slope.

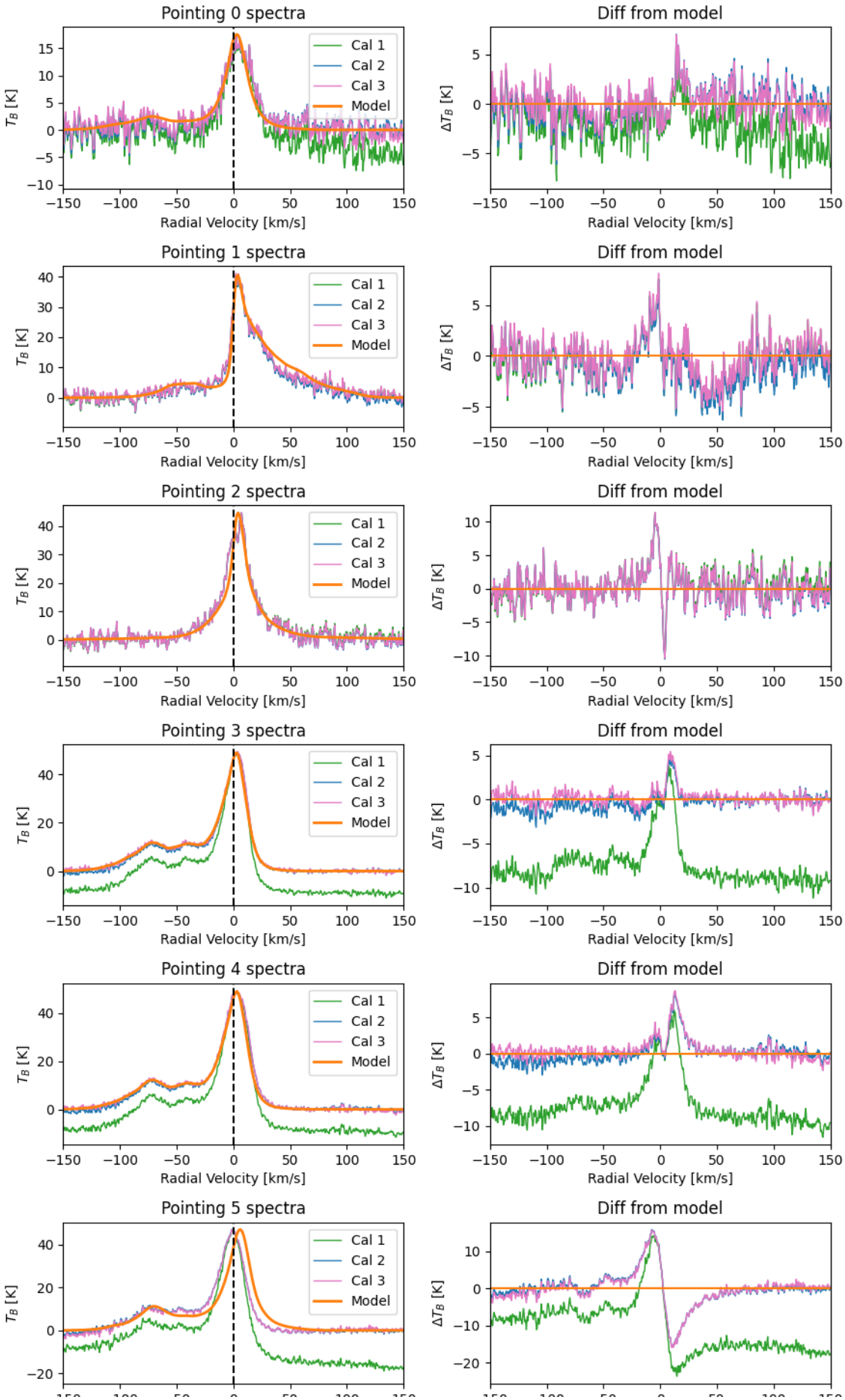
While trying to understand why it failed, we tried applying the solution from (2), *then* applying (3). This fixed the catastrophic fail and indeed slightly improved the overall calibration. This can be best seen in the right plots below, which shows the difference between model and calibrated spectra. We'll call this new method (4).

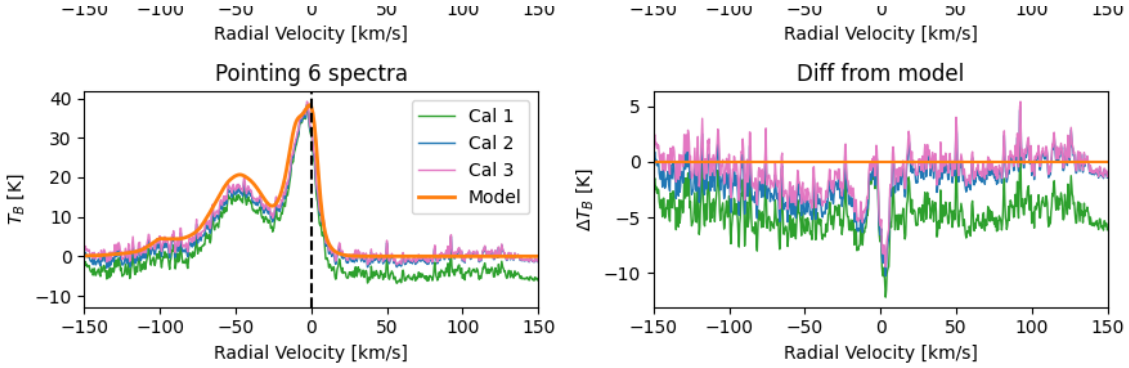
In [13]:
```
fig, axs = plt.subplots(nrows=ntrials, ncols=2, figsize=(1.5*stack_figsize[0
if ntrials == 1: axs = [axs]

for pointing in range(ntrials):

    # Interpolate model -- Used for calculating chi-sq
    PosIn = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-150))
    NegIn = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+150))
    model = CubicSpline(model_vs[pointing],model_Ts[pointing])(unique_vs[poi

    # Calibration 1 - currently in tutorial
    vneg75 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+75))
    vneg100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+100))
    v100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-100))

    noise = np.mean(avg_spectra[pointing][vneg100:vneg75])
    gain = max(avg_spectra[pointing][vneg100:v100]-noise)/(max(model_Ts[poir

    cal_data = (avg_spectra[pointing] - noise) / gain

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:g
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 2)
    print(f'Pointing {pointing}, method 1: chi^2={chi_sq}')

    # Calibration 2 - Noise slope a la Carly
    vpos100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+100))
    vpos150 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value+150))
    noisepos = np.mean(avg_spectra[pointing][vpos150:vpos100])
    vneg100 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-100))
    vneg150 = np.argmin(np.abs(unique_vs[pointing].to(u.km/u.s).value-150))

    noiseneg = np.mean(avg_spectra[pointing][vneg100:vneg150])
    noise_slope = (noiseneg-noisepos)/(250)

    noise = noise_slope*(unique_vs[pointing].to(u.km/u.s).value+125)+ noisep
    gain = max(avg_spectra[pointing]-noise)/(max(model_Ts[pointing]))
    cal_data = (avg_spectra[pointing] - noise) / gain

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:b
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 3)

    print(f'Pointing {pointing}, method 2: chi^2={chi_sq}')


    # Calibration 4 - (2) then (3)

    odr_model = odr.Model(cal_model)

    mydata = odr.RealData(unique_vs[pointing].to(u.km/u.s).value[NegIn:PosIr
    myodr = odr.ODR(mydata, odr_model, beta0=[1, 0, 0], ifixx=np.ones(len(av
    myoutput = myodr.run()


    gain = myoutput.beta[0]
    noise_slope = myoutput.beta[1]
    noisepos = myoutput.beta[2]
```

```
    noise = noise_slope*(unique_vs[pointing].to(u.km/u.s).value)+ noisepos

    cal_data = (cal_data - noise)

    axs[pointing, 0].plot(unique_vs[pointing].to(u.km/u.s), cal_data, 'tab:p
    axs[pointing, 1].plot(unique_vs[pointing].to(u.km/u.s)[NegIn:PosIn], cal
    chi_sq = reduced_chisquare(cal_data[NegIn:PosIn], model, 3)
    print(f'Pointing {pointing}, method 4: chi^2={chi_sq}')

    # Finish up plot
    axs[pointing, 0].plot(model_vs[pointing],model_Ts[pointing],'tab:orange'
    axs[pointing, 1].axhline(0, color='tab:orange')
    axs[pointing, 0].axvline(0, color ='k', linestyle='--')

    axs[pointing, 0].set_xlim(-150,150)
    axs[pointing, 0].set_ylabel('$T_B$ [K]')
    axs[pointing, 0].legend(loc='best')
    axs[pointing, 0].set_xlabel('Radial Velocity [km/s]')
    axs[pointing, 0].set_title(f'Pointing {pointing} spectra')
    axs[pointing, 1].set_xlim(-150,150)
    axs[pointing, 1].set_ylabel('$\Delta T_B$ [K]')
    axs[pointing, 1].set_xlabel('Radial Velocity [km/s]')
    axs[pointing, 1].set_title(f'Diff from model')



plt.tight_layout()
```

```
Pointing 0, method 1: chi^2=3.873567198321034
Pointing 0, method 2: chi^2=1.7188630038346382
Pointing 0, method 4: chi^2=1.6183430136891936
Pointing 1, method 1: chi^2=2.6484634065470316
Pointing 1, method 2: chi^2=2.940656238325757
Pointing 1, method 4: chi^2=2.5028349230892393
Pointing 2, method 1: chi^2=2.5768533978437107
Pointing 2, method 2: chi^2=2.474322096886441
Pointing 2, method 4: chi^2=2.4829420457683433
Pointing 3, method 1: chi^2=228.82888663615586
Pointing 3, method 2: chi^2=6.089102697263134
Pointing 3, method 4: chi^2=4.779555255490428
Pointing 4, method 1: chi^2=189.2535652809135
Pointing 4, method 2: chi^2=11.267831927042755
Pointing 4, method 4: chi^2=11.177398923381121
Pointing 5, method 1: chi^2=263.11960344699685
Pointing 5, method 2: chi^2=56.81883678424331
Pointing 5, method 4: chi^2=56.67709296217165
Pointing 6, method 1: chi^2=21.903584949610032
Pointing 6, method 2: chi^2=6.987235716213565
Pointing 6, method 4: chi^2=4.434613344834215
```

Figure


Pointing 0 spectra
Diff from model
Pointing 1 spectra
Diff from model
Pointing 2 spectra
Diff from model
Pointing 3 spectra
Diff from model
Pointing 4 spectra
Diff from model
Pointing 5 spectra
Diff from model
Cal 1
Cal 2
Cal 3
Model
T_B [K]
ΔT_B [K]
Radial Velocity [km/s]



Here's a table summarizing the $\bar{\chi}^2$ values for the different observations and calibration methods.

Pointing	Cal 1	Cal 2	Cal 3	Cal 4
0 - Tutorial	3.87	1.72	1.62	1.62
1 - Tutorial	2.65	2.94	2.47	2.50
2 - Tutorial	2.58	2.47	2.41	2.48
3 - Carly's	229	6.09	1180	4.78
4 - Kate's	189	11.3	2020	11.2
5 - David's	263	56.8	176	56.7
6 - Katherine's	21.9	6.99	169.5	4.43