

# REDOR\_analysis

June 26, 2025

## 1 Analysis of the Arg REDOR data, both for CH2 and NH signals

```
[1]: import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt

[2]: def compute_delta_and_noise(S, S0, avg_noise):
    """
    Compute  $\Delta S/S_0 = (S_0 - S)/S_0$  and propagate errors assuming avg_noise
    is the standard deviation of both S and S0 values.

    Parameters:
    - S: np.ndarray of shape (n_lines, n_points)
    - S0: np.ndarray of same shape as S
    - avg_noise: float (standard deviation of S and S0 elements)

    Returns:
    - deltaS_S0: np.ndarray of shape (n_lines, n_points)
    - deltaS_S0_noise: np.ndarray of shape (n_lines, n_points)
    """

    # Sanity check
    assert S.shape == S0.shape, "S and S0 must have the same shape"

    # Calculate delta
    deltaS_S0 = (S0 - S) / S0

    # Error propagation
    #  $df/dS_0 = S / S_0^2$ ,  $df/dS = -1 / S_0$ 
    df_dS0 = S / (S0**2)
    df_dS = -1 / S0

    # Assuming same noise for S and S0
    deltaS_S0_noise = np.sqrt(df_dS0**2 + df_dS**2) * avg_noise

    return deltaS_S0, deltaS_S0_noise
```

```

def compute_delta_and_noise(S, S0, avg_noise):
    """
    Compute  $\Delta S/S_0 = (S_0 - S)/S_0$  with error propagation, and prepend a point
    at index 0
    with  $\Delta S/S_0 = 0$  and  $\text{noise} = 0$ .

    Parameters:
    - S: np.ndarray of shape (n_lines, n_points)
    - S0: np.ndarray of same shape as S
    - avg_noise: float

    Returns:
    - deltaS_S0: np.ndarray of shape (n_lines, n_points + 1)
    - noise_array: np.ndarray of same shape
    """
    S = np.asarray(S)
    S0 = np.asarray(S0)
    assert S.shape == S0.shape, "S and S0 must have the same shape"

    deltaS_S0 = (S0 - S) / S0

    # Derivatives for error propagation
    df_dS0 = S / (S0**2)
    df_dS = -1 / S0
    noise_array = np.sqrt(df_dS0**2 + df_dS**2) * avg_noise

    # Prepend zero point
    zero_column = np.zeros((S.shape[0], 1)) # shape (n_lines, 1)
    deltaS_S0 = np.hstack([zero_column, deltaS_S0])
    noise_array = np.hstack([zero_column, noise_array])

    return deltaS_S0, noise_array

```

```

[3]: # Function to read the GAMMA simulations
import os
import re

def extract_number(filename):
    """Extract the number from a filename like sim_REDOR_34800.mat"""
    match = re.search(r'sim_REDOR_(\d+)\.mat', filename)
    return int(match.group(1)) if match else None

def compute_all_deltaS_S0_from_mat_files(directory):
    files = [f for f in os.listdir(directory) if f.endswith('.mat') and
    'sim_REDOR_' in f]
    files_sorted = sorted(files, key=extract_number)

```

```

all_deltas = []
gamma_numbers = []

for fname in files_sorted:
    fnum = extract_number(fname)
    key = f'./out_REDOR/sim_REDOR_{fnum}'
    fpath = os.path.join(directory, fname)

    try:
        t = sio.loadmat(fpath)[key]
    except KeyError:
        print(f"Warning: Key '{key}' not found in {fname}")
        continue

    S = np.real(t[0])
    S0 = np.real(t[1])

    deltaS_S0 = (S0 - S) / S0
    all_deltas.append(deltaS_S0)
    gamma_numbers.append(fnum)

return np.array(all_deltas), np.array(gamma_numbers)

```

```

[4]: def plot_chisquare_vs_gamma_number(chi_square_array, gamma_numbers, m, ax=None):
    """
    Plot chi-square values as a function of gamma curve ID for one experimental
    dataset.

    Parameters:
    - chi_square_array: (M, N) array of chi-square values
    - gamma_numbers: (N,) array of integers corresponding to gamma_matrix rows
    - m: index of experimental dataset
    - ax: optional matplotlib axis
    """
    if ax is None:
        fig, ax = plt.subplots(figsize=(8, 5))

    ax.plot(gamma_numbers, chi_square_array[m], marker='o', linestyle='-')
    ax.set_xlabel('Gamma REDOR Number')
    ax.set_ylabel('Chi-square')
    ax.set_title(f'Chi-square vs REDOR simulation number (dataset #{m})')
    ax.grid(True)

```

```

[5]: def estimate_dipolar_coupling_error(chi_square_row, gamma_numbers, num_points,
    num_fit_params=1):
    """
    Estimate the 1-sigma error range on dipolar coupling from chi-square.

```

```

Parameters:
- chi_square_row: 1D array of chi-square values (for one dataset)
- gamma_numbers: 1D array of dipolar coupling values corresponding to
↳chi-square entries
- num_points: number of data points used in fit
- num_fit_params: number of parameters fitted (default: 1)

Returns:
- best_fit_value: dipolar coupling with min  $\chi^2$ 
- lower_bound: lower bound of coupling ( $\chi_{red} < \chi_{red\_min} + 1$ )
- upper_bound: upper bound of coupling
- reduced_chi2_all: array of all reduced chi-square values
"""
dof = num_points - num_fit_params
reduced_chi2_all = chi_square_row / dof

min_idx = np.argmin(reduced_chi2_all)
chi2_min = reduced_chi2_all[min_idx]
best_fit_value = gamma_numbers[min_idx]

within_range = reduced_chi2_all <= (chi2_min + 1)
gamma_in_range = gamma_numbers[within_range]

lower_bound = np.min(gamma_in_range)
upper_bound = np.max(gamma_in_range)

return best_fit_value, lower_bound, upper_bound, reduced_chi2_all

def plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper):
    plt.figure(figsize=(8,5))
    plt.plot(gamma_numbers, red_chi2, '-o')
    plt.axhline(y=np.min(red_chi2)+1, color='gray', linestyle='--',
↳label=' $\chi^2_{min} + 1$ ')
    plt.axvline(best_fit, color='r', linestyle='--', label='Best Fit')
    plt.axvline(lower, color='g', linestyle=':', label='Lower Bound')
    plt.axvline(upper, color='g', linestyle=':', label='Upper Bound')
    plt.xlabel('Dipolar coupling')
    plt.ylabel('Reduced  $\chi^2$ ')
    plt.title('Reduced  $\chi^2$  vs Dipolar Coupling')
    plt.legend()
    plt.grid(True)
    plt.show()

```

```

[6]: def grid_search_chisquare(dS_S0_matrix, noise_matrix, gamma_matrix):
      """
      Perform chi-square grid search between experimental and simulated datasets.

```

*Parameters:*

- *dS\_S0\_matrix*: *np.ndarray* of shape *(M, P)*
- *noise\_matrix*: *np.ndarray* of shape *(M, P)*
- *gamma\_matrix*: *np.ndarray* of shape *(N, P)*

*Returns:*

- *chi\_square\_array*: *np.ndarray* of shape *(M, N)*
  - *best\_fit\_indices*: *np.ndarray* of shape *(M,)*
- """

```
M, P = dS_S0_matrix.shape
N = gamma_matrix.shape[0]
```

```
chi_square_array = np.zeros((M, N))
```

```
for m in range(M):
    for n in range(N):
        noise_matrix[m][0]=1e-4
        residual = dS_S0_matrix[m] - gamma_matrix[n]
        chi2 = np.sum((residual / noise_matrix[m])**2)
        chi_square_array[m, n] = chi2
```

```
# Get the index of the minimum chi2 value in each row (i.e., best gamma_
↪match)
```

```
best_fit_indices = np.argmin(chi_square_array, axis=1)
```

```
return chi_square_array, best_fit_indices
```

```
[7]: def plot_fit(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH,
↪best_fit_indices, m, ax=None):
```

"""

*Plot data with error bars and best-fit simulated curve.*

*Parameters:*

- *dS\_S0\_matrix*: *(M, P)* array of experimental *deltaS/S0* values
  - *noise\_matrix*: *(M, P)* array of corresponding noise
  - *gamma\_matrix*: *(N, P)* array of simulated curves
  - *timeaxisCH*: *(P,)* array of x-axis values
  - *best\_fit\_indices*: *(M,)* array of best-fit *gamma* row indices
  - *m*: int, index of the experimental dataset to plot
  - *ax*: matplotlib axis (optional), for subplotting
- """

```
if ax is None:
```

```
    fig, ax = plt.subplots(figsize=(8, 5))
```

```
y_exp = dS_S0_matrix[m]
```

```

y_err = noise_matrix[m]
y_fit = gamma_matrix[best_fit_indices[m]]

ax.errorbar(timeaxisCH, y_exp, yerr=y_err, fmt='o', label='Experimental',
↪capsize=3,color='k')
ax.plot(timeaxisCH, y_fit, label='Best-fit simulation', color='r')

ax.set_xlabel('Recoupling time (μs)')
ax.set_ylabel('ΔS/S ')
ax.set_title(f'Fit for dataset #{m} (best match: gamma_
↪#{best_fit_indices[m]})')
ax.legend()
ax.grid(True)
ax.set_ylim([-0.3,1.3])

```

```

[32]: def plot_normalized_asymmetric_barplot(best_fit_array, lower_array,
↪upper_array, residuenumbers, rigidlimit):
    """
    Plot a normalized bar chart with asymmetric error bars.

    Parameters:
    - best_fit_array: array of best-fit dipolar couplings
    - lower_array: array of lower confidence bounds
    - upper_array: array of upper confidence bounds
    - residuenumbers: list of strings for x-axis labels
    - rigidlimit: float, the value used for normalization
    """
    best_fit_array = np.asarray(best_fit_array) / rigidlimit
    lower_array = np.asarray(lower_array) / rigidlimit
    upper_array = np.asarray(upper_array) / rigidlimit

    # Calculate asymmetric errors
    yerr_lower = best_fit_array - lower_array
    yerr_upper = upper_array - best_fit_array
    yerr = np.vstack((yerr_lower, yerr_upper))

    x = np.arange(len(best_fit_array))

    fig, ax = plt.subplots(figsize=(12, 6))
    ax.bar(x, best_fit_array, color='cornflowerblue', edgecolor='k', alpha=0.8)
    ax.errorbar(x, best_fit_array, yerr=yerr, fmt='none', ecolor='black',
↪capsize=5)

    ax.set_xticks(x)
    ax.set_xticklabels(residuenumbers, rotation=90)
    ax.set_ylabel('Order parameter')
    ax.set_title(f'Normalized dipolar coupling; rigid limit = {rigidlimit}')

```

```

ax.grid(True, axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import numpy as np

def plot_asymmetric_barplot_with_colors(best_fit_array, lower_array,
    ↪upper_array, residuenumbers, colors):
    """
    Plot a bar chart with asymmetric error bars and individual colors per bar.

    Parameters:
    - best_fit_array: array of best-fit dipolar couplings (not normalized)
    - lower_array: array of lower confidence bounds
    - upper_array: array of upper confidence bounds
    - residuenumbers: list of strings for x-axis labels
    - colors: list or array of color strings; bars beyond this will be colored
    ↪black
    """
    best_fit_array = np.asarray(best_fit_array)
    lower_array = np.asarray(lower_array)
    upper_array = np.asarray(upper_array)

    yerr_lower = best_fit_array - lower_array
    yerr_upper = upper_array - best_fit_array
    yerr = np.vstack((yerr_lower, yerr_upper))

    x = np.arange(len(best_fit_array))

    # Extend colors with 'black' if not enough colors
    full_colors = list(colors) + ['black'] * (len(best_fit_array) - len(colors))

    fig, ax = plt.subplots(figsize=(5, 6))
    for i in range(len(best_fit_array)):
        ax.bar(x[i], best_fit_array[i], color=full_colors[i], edgecolor='k',
    ↪alpha=0.8)
        ax.errorbar(x[i], best_fit_array[i],
                    yerr=[[yerr_lower[i]], [yerr_upper[i]]],
                    fmt='none', ecolor='black', capsize=5)

    ax.set_xticks(x)
    ax.set_xticklabels(residuenumbers, rotation=90)
    ax.set_ylabel('Dipolar coupling')
    ax.set_title('Dipolar couplings with 1 confidence intervals')
    ax.grid(True, axis='y', linestyle='--', alpha=0.5)
    plt.tight_layout()

```

```

plt.savefig('fitted_S.pdf')

def plot_fit_multi(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH,
    ↪best_fit_indices, figurename, m=[0], colors=None,):
    """
    Plot one or multiple datasets with their corresponding best-fit curves.

    Parameters:
    - dS_S0_matrix: (M, P) array of experimental  $\Delta S/S$  values
    - noise_matrix: (M, P) array of noise values
    - gamma_matrix: (N, P) array of simulated best-fit curves
    - timeaxisCH: (P,) time axis
    - best_fit_indices: (M,) array of best-fit indices per dataset
    - m: list of dataset indices to plot (e.g., [0, 1, 2])
    - colors: list of color names/strings for each dataset
    - filename: name of the output file (PDF)
    """

    if isinstance(m, int):
        m = [m]

    if colors is None:
        # Default color cycle from matplotlib
        colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
    elif len(colors) < len(m):
        raise ValueError("Not enough colors provided for all datasets.")

    fig, ax = plt.subplots(figsize=(6, 5))

    for idx, mi in enumerate(m):
        color = colors[idx]
        y_exp = dS_S0_matrix[mi]
        y_err = noise_matrix[mi]
        y_fit = gamma_matrix[best_fit_indices[mi]]

        ax.errorbar(timeaxisCH, y_exp, yerr=y_err, fmt='o', color=color,
    ↪capsize=3,
                    label=f'Exp m={mi}', alpha=0.7)
        ax.plot(timeaxisCH, y_fit, color=color, linestyle='--',
                label=f'Fit m={mi}')

    ax.set_xlabel('Recoupling time ( $\mu$ s)')
    ax.set_ylabel('ΔS/S ')
    ax.set_title('Experimental data with best-fit REDOR simulations')
    #ax.legend()

```



```

ax.grid(True)
ax.set_ylim([-0.3,1.3])
plt.tight_layout()
plt.savefig(figurename)

```

## 2 Define rigid-limit couplings

```

[33]: rigidCH=45395 #1.1 Angstrom
      rigidNH=22954 #1.02 Angstrom

```

## CH2 REDOR data

```

[34]: # These are the experimentally determined intensities

S=np.array([[ 977390.10017154, 1088520.84181035, 854733.09596963,
             845151.42974085, 704320.2175827 , 654241.72382519,
             643224.25523603, 421955.90793963, 450228.49051879,
             257738.05634066, 320954.42972261, 363959.40617295,
             408997.31354257, 189507.29897761, 132781.83024638,
             140073.93589719, -49632.54609013, -90400.40052313,
            -207304.26814032, 107691.98377771, 111982.76588314,
             -75028.629268 , 142317.74703917, 108621.34402391],
            [7458643.45951463, 6340000.01563429, 5037564.94083503,
             3792236.16371306, 2343293.06783867, 1644876.99888346,
             1238653.44853859, 779987.77895015, 600069.06435531,
             648413.70392248, 673971.0167436 , 582313.74059995,
             635725.94353073, 292309.09515621, 204187.17391015,
            -152693.92872568, -244192.59038495, -314091.00240814,
            -242052.2047734 , -236979.82360752, -260305.7222569 ,
            -202150.64626376, 96405.7392361 , -175985.52179436]])

S0=np.array([[ 915236.11024755, 904867.32851956, 894616.01553359,
              884480.84047704, 874460.48761419, 864553.6561154 ,
              854759.05988823, 845075.42741048, 835501.50156511,
              826036.03947709, 816677.81235198, 807425.60531649,
              798278.21726068, 789234.46068212, 780293.16153166,
              771453.15906107, 762713.30567229, 754072.46676853,
              745529.52060692, 737083.35815288, 728732.88293621,
              720477.01090868, 712314.67030334, 704244.80149536],
             [7469641.31437462, 7442016.79763145, 7414494.44294576,
              7387073.87249786, 7359754.70986536, 7332536.58001794,
              7305419.10931224, 7278401.92548673, 7251484.65765658,
              7224666.9363086 , 7197948.39329612, 7171328.66183398,
              7144807.3764935 , 7118384.1731974 , 7092058.68921486,
              7065830.56315655, 7039699.43496962, 7013664.94593278,
              6987726.73865138, 6961884.45705252, 6936137.74638011,
              6910486.25319006, 6884929.62534538, 6859467.51201138]])

```

```

noise=np.array([ 98500.359375 , 72113.0859375 , 46788.91796875, 71583.
↪5546875 ,
                71747.96875 , 116038.578125 , 59292.19921875, 73051.015625 ,
                69177.9375 , 53060.5625 , 67934.4140625 , 49195.6328125 ,
                44384.12109375, 43860.11328125, 51125.4375 , 39298.54296875,
                89379.4140625 , 57060.27734375, 61434.5234375 , 56084.08203125,
                64365.13671875, 65928.3515625 , 69352.6484375 , 85687.8984375 ])

avg_noise=np.average(noise)
max_noise=1.5*np.max(noise)

timeaxisCH=(np.arange(-2,48,2)+2)*18

residuenumbers=['R42','R54']

```

```
[35]: dS_S0_matrix, noise_matrix = compute_delta_and_noise(S, S0, max_noise)
```

### 2.0.1 Read the GAMMA simulations

```
[36]: gamma_matrix, gamma_numbers = compute_all_deltaS_S0_from_mat_files('GAMMA/
↪CH2_REDOR/out_REDOR')
```

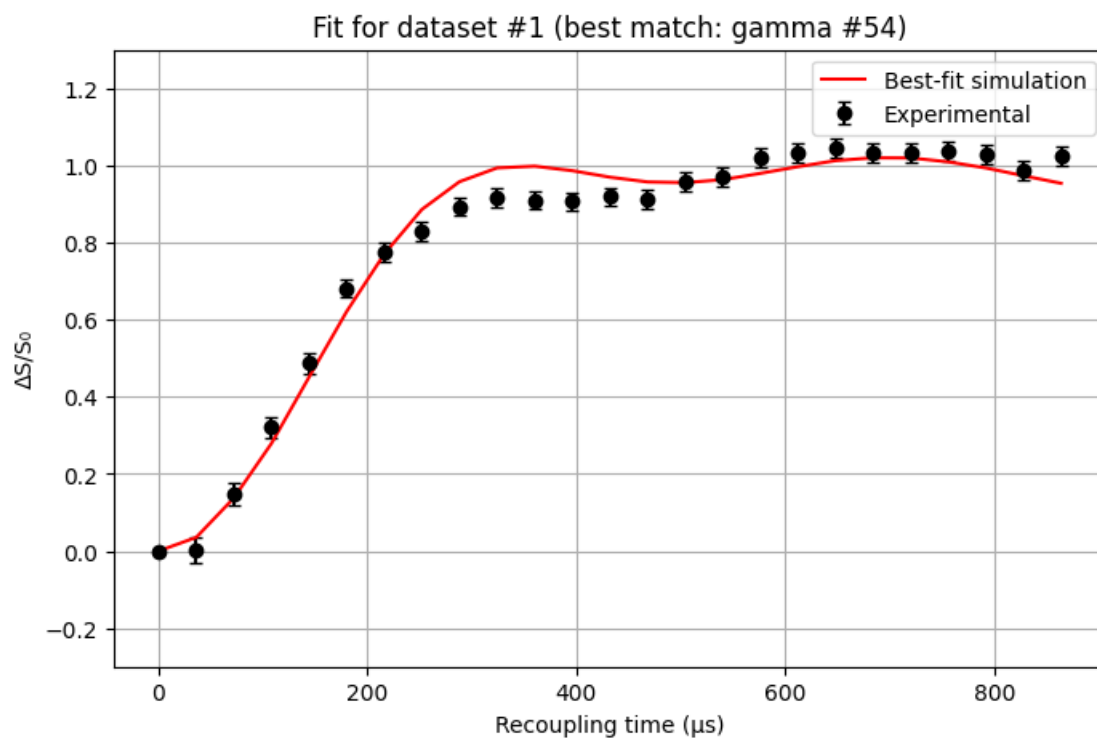
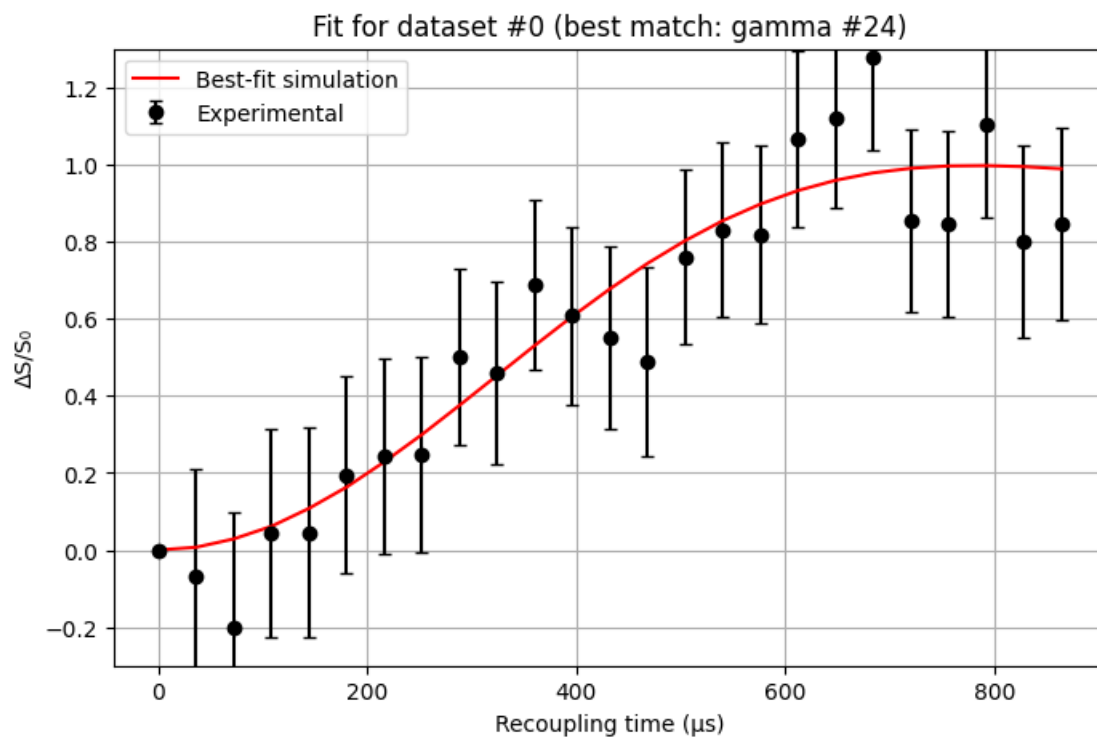
### 2.0.2 Chi square fitting routine

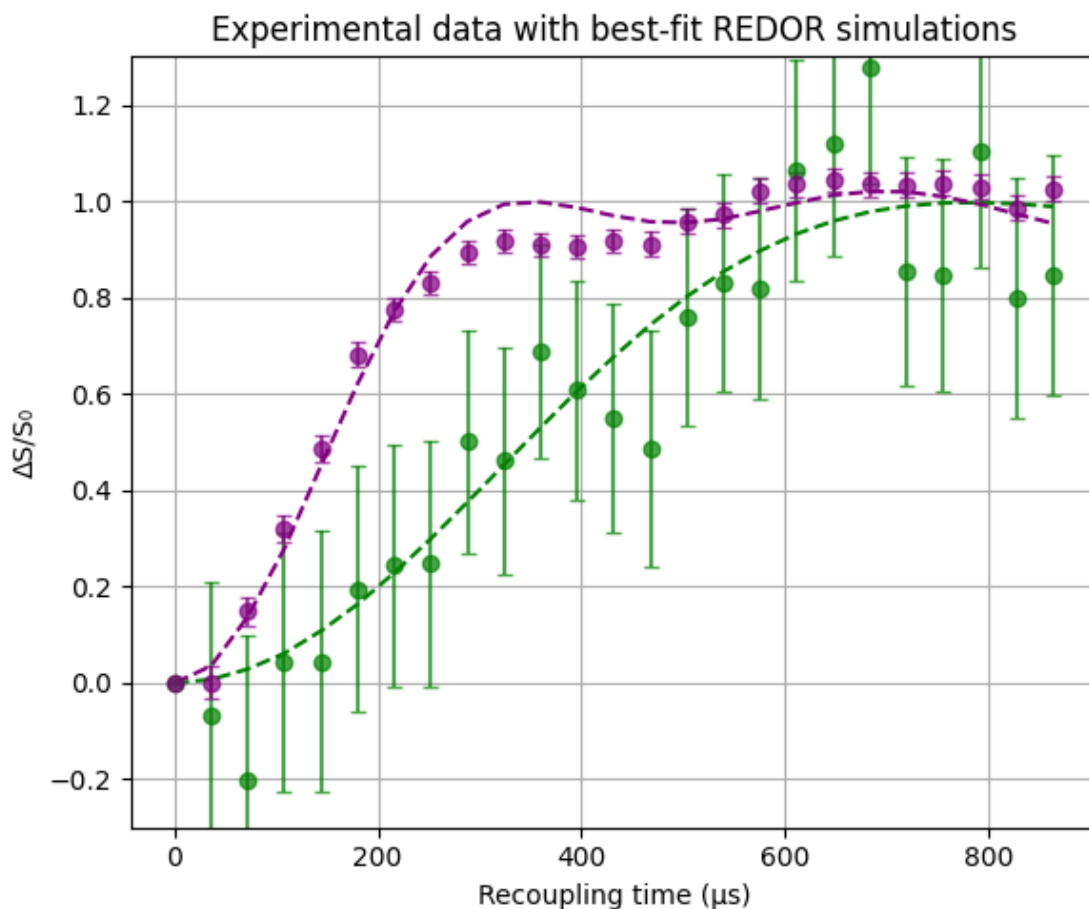
```
[37]: chi2_matrix, best_fits = grid_search_chisquare(dS_S0_matrix, noise_matrix,
↪gamma_matrix)

print("Best gamma_matrix index for each experimental dataset:")
print(best_fits)
```

Best gamma\_matrix index for each experimental dataset:  
[24 54]

```
[38]: plot_fit(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH, best_fits, m=0)
plt.show()
plot_fit(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH, best_fits, m=1)
plt.show()
plot_fit_multi(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH, best_fits, 'REDORcurves_CH.
↪pdf', m=[0, 1], colors=['green', 'purple'])
```





```
[39]: best_fit_array=[]
lower_array=[]
upper_array=[]
m = 0 # dataset index
num_points = dS_S0_matrix.shape[1] # assumes one value per time point

best_fit, lower, upper, red_chi2 = _
    ↪ estimate_dipolar_coupling_error(chi2_matrix[m], gamma_numbers, num_points)

best_fit_array.append(best_fit)
lower_array.append(lower)
upper_array.append(upper)
print(f"Best-fit dipolar coupling: {best_fit}")
print(f"1 confidence interval: [{lower}, {upper}]")
plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper)

m = 1 # dataset index
num_points = dS_S0_matrix.shape[1] # assumes one value per time point
```

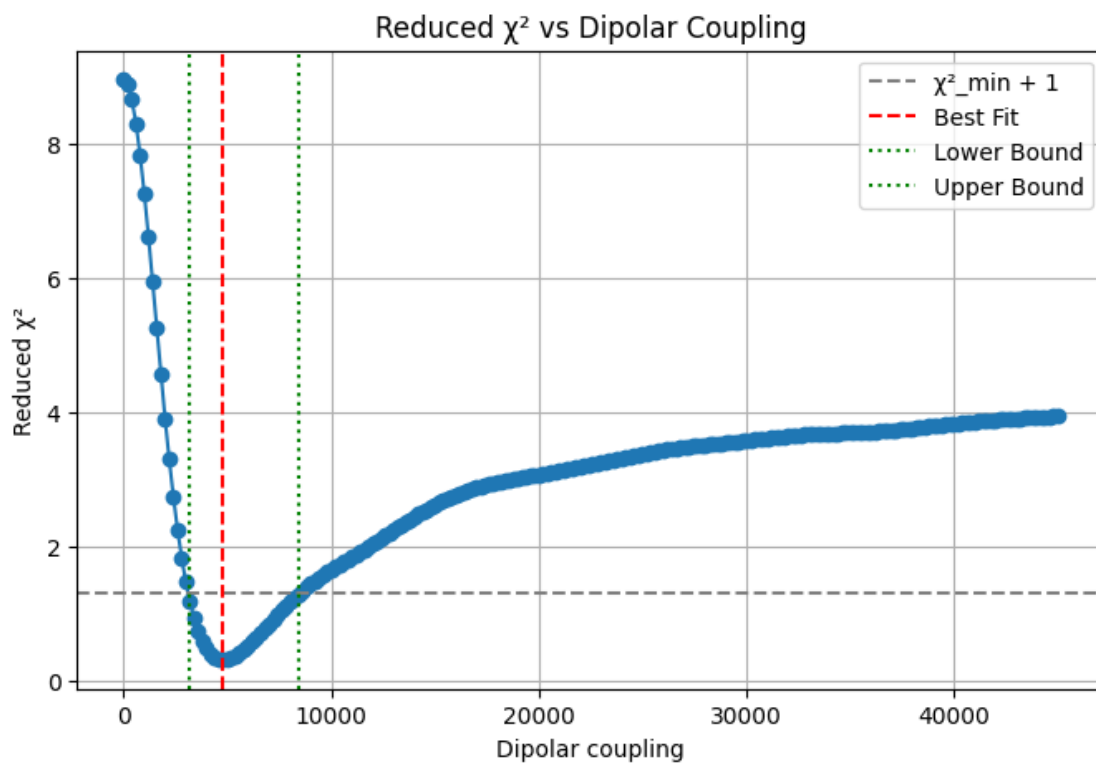
```

best_fit, lower, upper, red_chi2 = _
    estimate_dipolar_coupling_error(chi2_matrix[m], gamma_numbers, num_points)

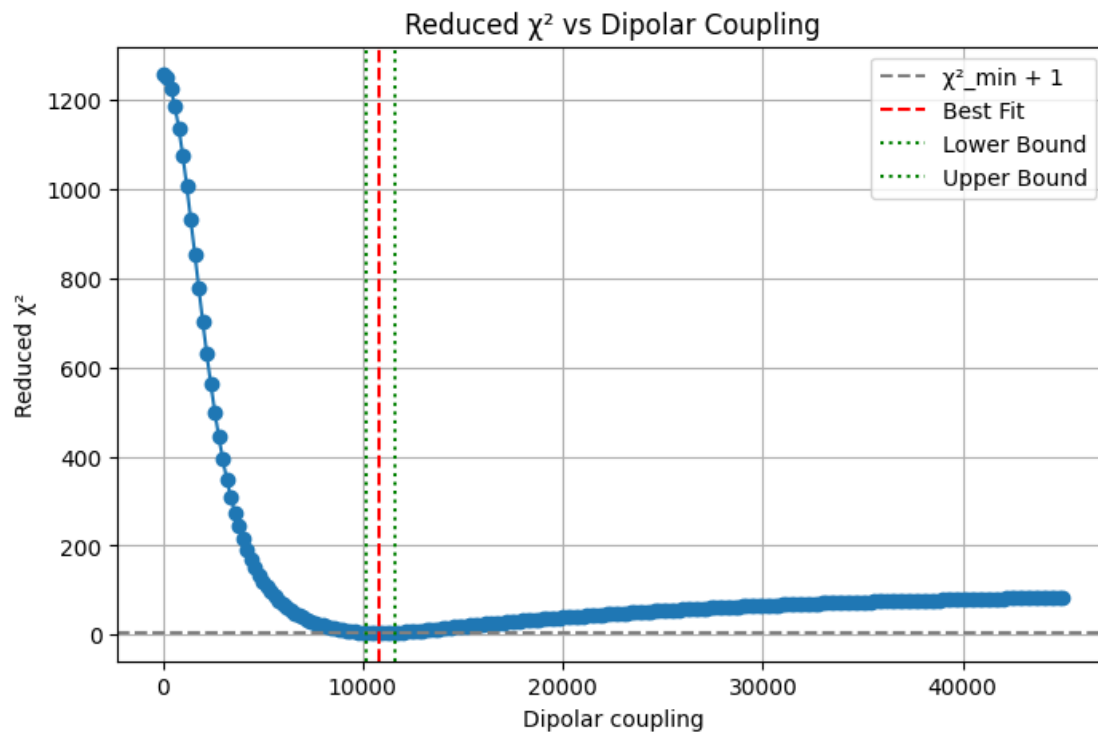
best_fit_array.append(best_fit)
lower_array.append(lower)
upper_array.append(upper)
print(f"Best-fit dipolar coupling: {best_fit}")
print(f"1 confidence interval: [{lower}, {upper}]")
plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper)

```

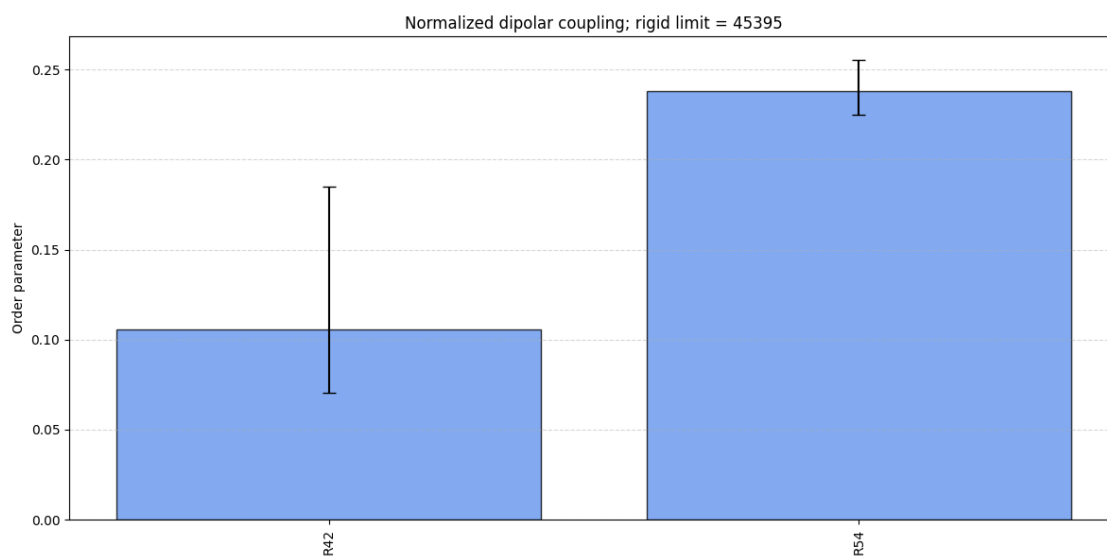
Best-fit dipolar coupling: 4800  
 1 confidence interval: [3200, 8400]



Best-fit dipolar coupling: 10800  
 1 confidence interval: [10200, 11600]



```
[40]: plot_normalized_asymmetric_barplot(best_fit_array, lower_array, upper_array,
    ↪ residuenumbers, rigidCH)
best_fit_array_CH=best_fit_array
lower_array_CH=lower_array
upper_array_CH=upper_array
```



### 3 Analysis of the N-H REDOR

The experimental data were obtained with an INEPT-based N-H REDOR at 100 kHz. Note that the arrays below (S and S0) comprise also data for two amide sites, which were used to check and compare with expected NH order parameters. First the data of the N-epsilon are fitted.

```
[41]: S=np.array([[7862440.70163726, 7677014.80124082, 7629790.8276038 ,
    7504805.510751 , 7283956.92610959, 6830688.57846852,
    6524217.16505059, 6448317.00544856, 5866126.11595737,
    5444836.16334465, 4956826.6291142 , 4755975.58982832,
    4257827.55573072, 3973858.83179174, 3432349.20090862,
    3080095.70672045],
    [20834992.59041996, 19494385.95476175, 17626425.55536398,
    15043377.81353951, 12163454.20277824, 9011003.36264457,
    6383802.5558581 , 3837148.32617099, 2266804.89293021,
    1014318.73516708, 582154.67669112, 457595.85970337,
    640746.06676193, 960325.10192587, 1685011.32670185,
    2093894.1883819 ],
    [13115747.62767311, 12332801.38197972, 11285955.93320769,
    9639789.43836269, 7676342.57478459, 5841616.32426806,
    4227040.85696643, 2549726.25041638, 1517733.77124204,
    653105.29929557, 446659.65808507, 282726.74649007,
    587740.59892433, 862161.59223164, 1344919.71585796,
    1648616.67696944]])

noise=np.array([103974.734375 , 93680.2890625, 57063.2734375, 96599.2734375,
    88057.765625 , 100749.703125 , 123041.09375 , 98370.96875 ,
    125301.8125 , 143564.03125 , 114578.40625 , 104980.1015625,
    106194.7265625, 116451.3359375, 103350.296875 , 107139.3984375])

S0=np.array([[7824869.16393341, 7818139.23047784, 7811415.08523462,
    7804696.72322547, 7797984.13947643, 7791277.32901777,
    7784576.28688407, 7777881.00811417, 7771191.48775117,
    7764507.72084243, 7757829.70243959, 7751157.42759854,
    7744490.8913794 , 7737830.08884656, 7731175.01506865,
    7724525.66511854],
    [20443734.30569646, 20437701.8587921 , 20431671.19191554,
    20425642.30454156, 20419615.19614505, 20413589.8662011 ,
    20407566.31418492, 20401544.53957189, 20395524.54183753,
    20389506.32045754, 20383489.87490776, 20377475.20466419,
    20371462.30920296, 20365451.1880004 , 20359441.84053295,
    20353434.26627724],
    [13047483.34121624, 13066439.01153412, 13085422.22106609,
    13104433.00982174, 13123471.41786876, 13142537.48533307,
    13161631.25239889, 13180752.75930879, 13199902.04636383,
```

```

13219079.15392363, 13238284.1224064 , 13257516.99228913,
13276777.80410757, 13296066.59845639, 13315383.41598921,
13334728.29741874]])

max_noise=1.5*np.max(noise)

timeaxisCH=(np.arange(-2,32,2)+2)*10

residuenumbers=['Ne54', 'G47', 'S20']

```

```
[42]: dS_S0_matrix, noise_matrix = compute_delta_and_noise(S, S0, max_noise)
```

### 3.1 Load GAMMA simulations for 4-spin NHHH system

These simulations are relevant for the N-epsilon site. The spin system comprises the H-epsilon and the two adjacent H-delta spins. The amide NHs are fitted further below, with a grid of simulations that comprise only the amide H.

```
[43]: gamma_matrix, gamma_numbers = compute_all_deltaS_S0_from_mat_files('GAMMA/
    ↪NH_ch2_REDOR/out_REDOR')
```

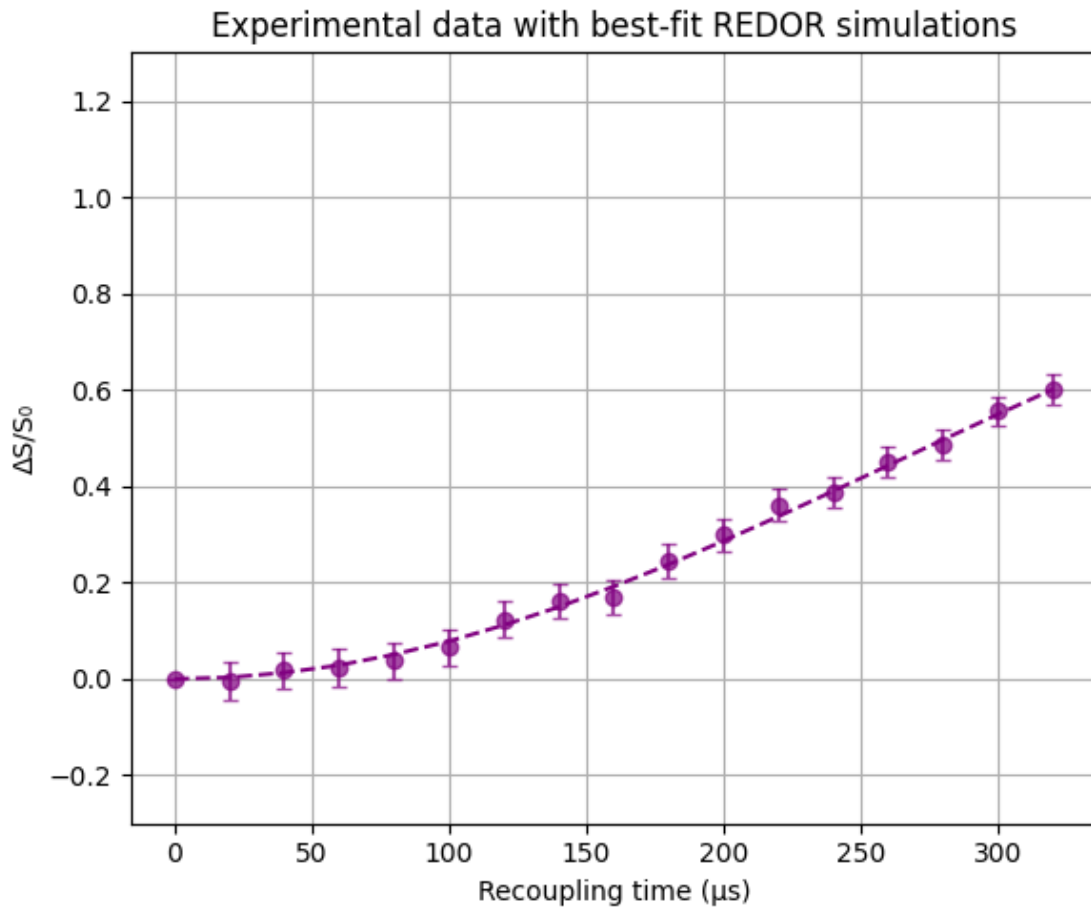
```
[44]: chi2_matrix, best_fits = grid_search_chisquare(dS_S0_matrix, noise_matrix,
    ↪gamma_matrix)

print("Best gamma_matrix index for each experimental dataset:")
print(best_fits)
```

```
Best gamma_matrix index for each experimental dataset:
[ 43 110 110]
```

```
[45]: plot_fit_multi(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH, best_fits,
    ↪'REDORcurve_NeH.pdf', m=[0], colors=['purple'])
```



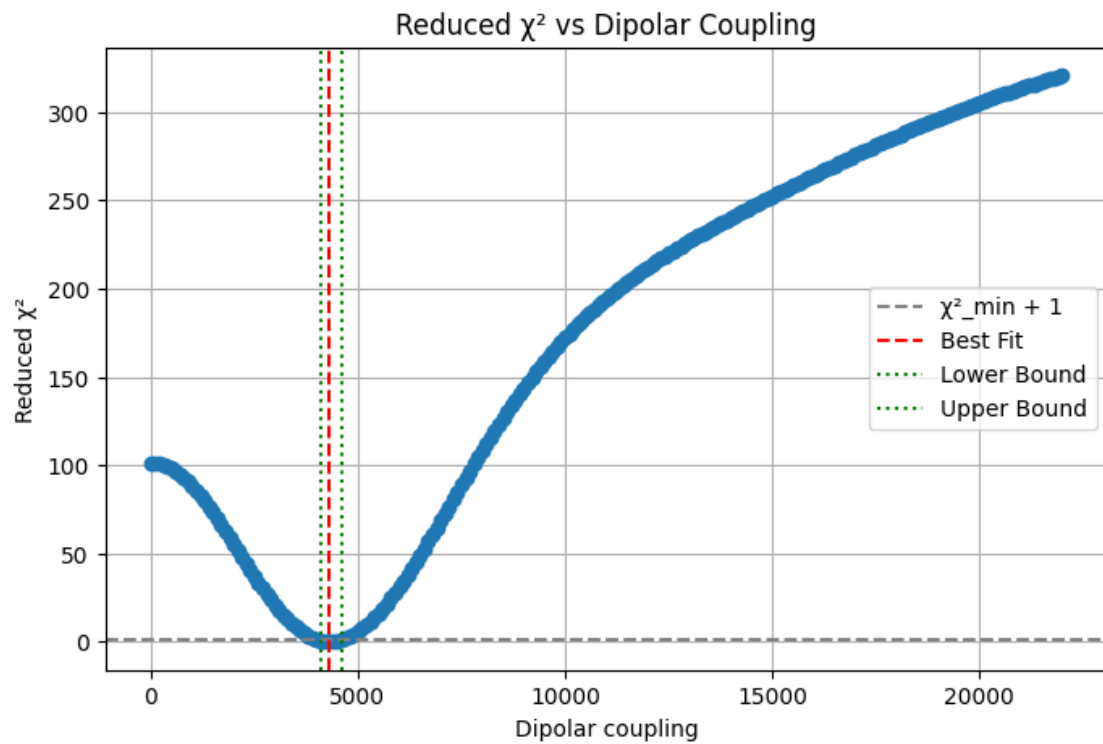


```
[46]: best_fit_array=[]
lower_array=[]
upper_array=[]
m = 0 # dataset index
num_points = dS_S0_matrix.shape[1] # assumes one value per time point

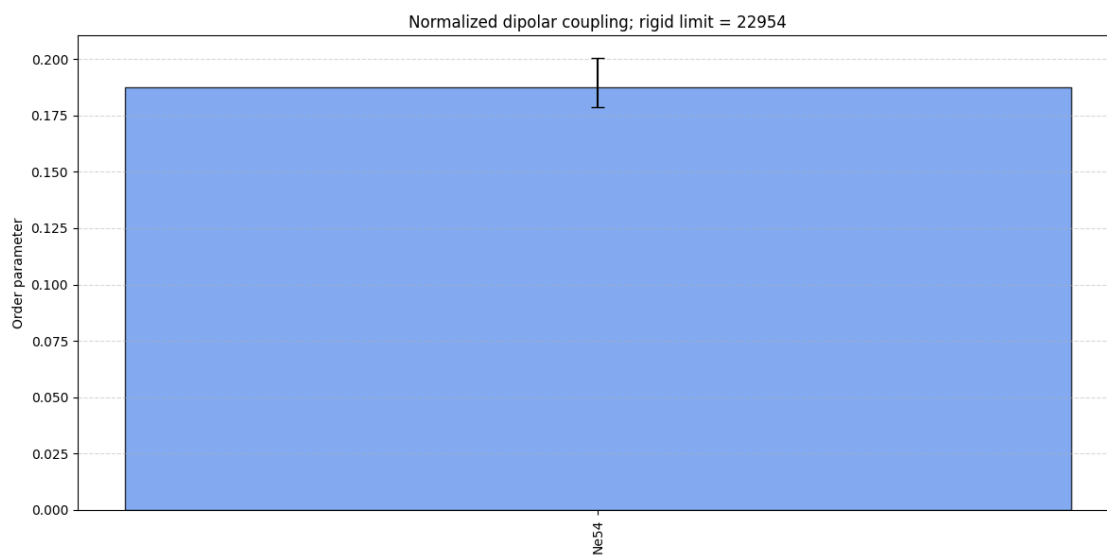
best_fit, lower, upper, red_chi2 = _
    estimate_dipolar_coupling_error(chi2_matrix[m], gamma_numbers, num_points)

best_fit_array.append(best_fit)
lower_array.append(lower)
upper_array.append(upper)
print(f"Best-fit dipolar coupling: {best_fit}")
print(f"1 confidence interval: [{lower}, {upper}]")
plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper)
```

```
Best-fit dipolar coupling: 4300
1 confidence interval: [4100, 4600]
```



```
[47]: residuenumbers=['Ne54']
      plot_normalized_asymmetric_barplot(best_fit_array, lower_array, upper_array,
      ↪residuenumbers,rigidNH)
```



### 3.2 Load GAMMA simulations for NH spin system (the amides do not have neighboring 1Hs)

Now fit the amide sites. The GAMMA simulations are different from those used above: they comprise only the N and H spins (2 spin system). Load GAMMA simulations:

```
[48]: gamma_matrix, gamma_numbers = compute_all_deltaS_S0_from_mat_files('GAMMA/  
    ↪NH_REDOR/out_REDOR')
```

Fit (chi2 procedure). (Note that the N-epsilon is also fitted (index 0), but will be ignored here.)

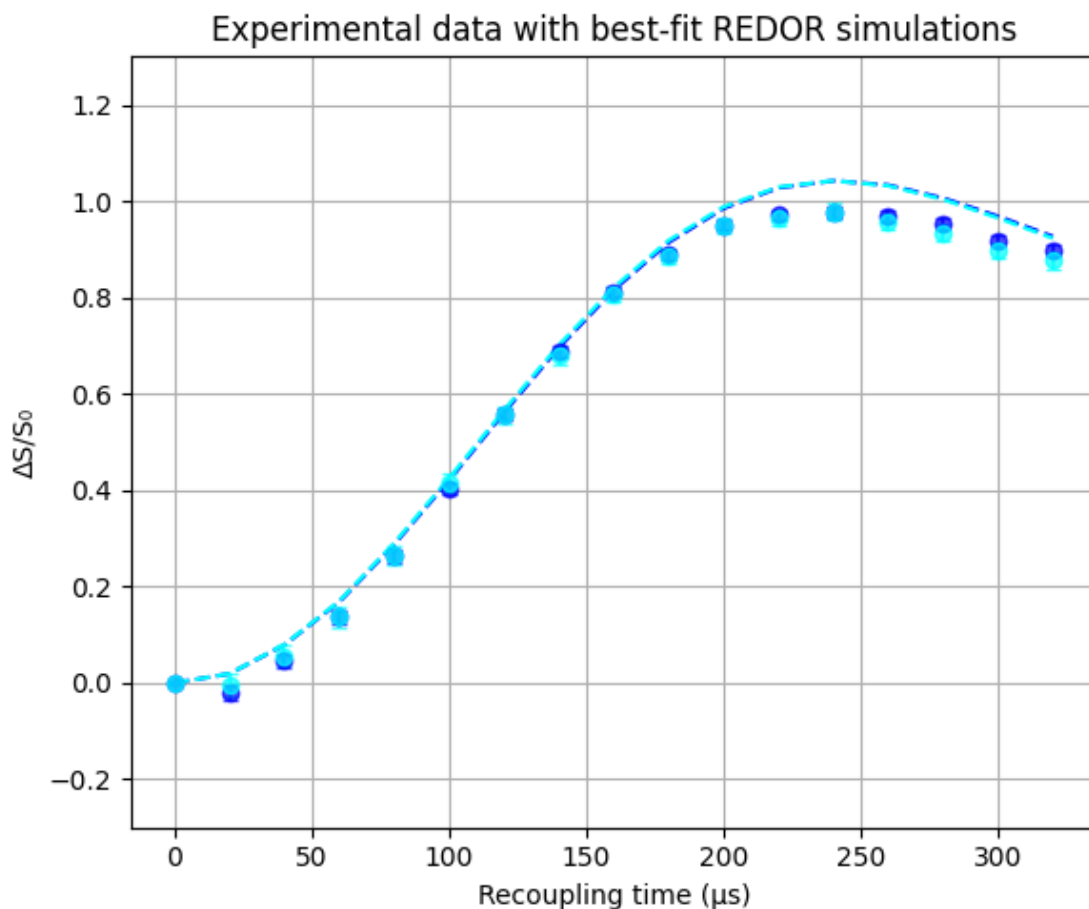
```
[49]: chi2_matrix, best_fits = grid_search_chisquare(dS_S0_matrix, noise_matrix,   
    ↪gamma_matrix)  
  
print("Best gamma_matrix index for each experimental dataset:")  
print(best_fits)
```

Best gamma\_matrix index for each experimental dataset:

```
[ 74 186 187]
```

Plot REDOR curves for residues G47 and S20.

```
[50]: plot_fit_multi(dS_S0_matrix, noise_matrix, gamma_matrix, timeaxisCH,   
    ↪best_fits, 'REDORcurves_NH.pdf', m=[1,2], colors=['blue', 'cyan'])  
plt.show()
```



```
[51]: m = 1 # dataset index
num_points = dS_S0_matrix.shape[1] # assumes one value per time point

best_fit, lower, upper, red_chi2 = _
    estimate_dipolar_coupling_error(chi2_matrix[m], gamma_numbers, num_points)

best_fit_array.append(best_fit)
lower_array.append(lower)
upper_array.append(upper)
print(f"Best-fit dipolar coupling: {best_fit}")
print(f"1 confidence interval: [{lower}, {upper}]")
plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper)

m = 2 # dataset index
num_points = dS_S0_matrix.shape[1] # assumes one value per time point

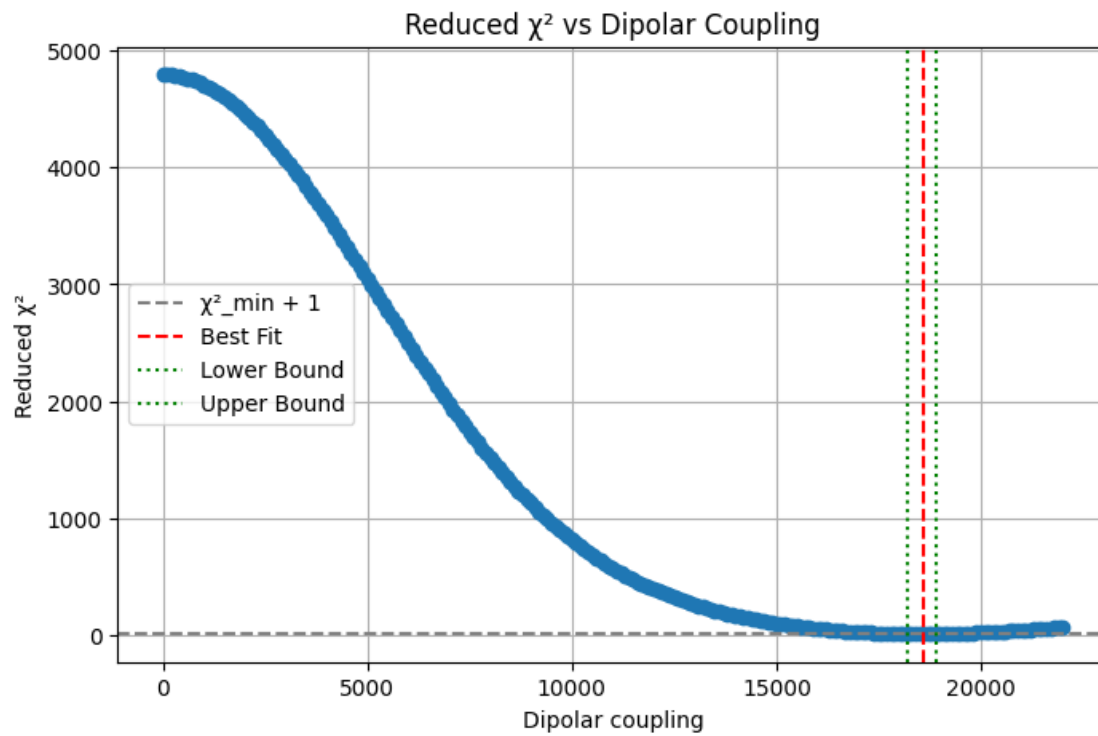
best_fit, lower, upper, red_chi2 = _
    estimate_dipolar_coupling_error(chi2_matrix[m], gamma_numbers, num_points)
```

```

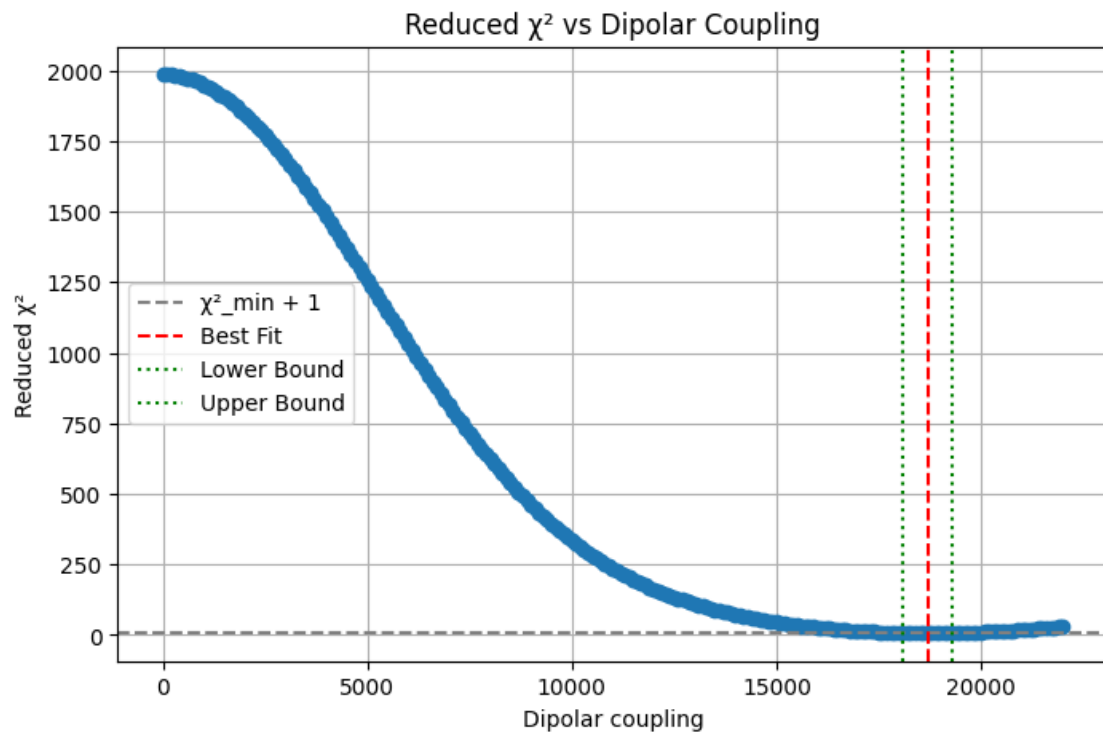
best_fit_array.append(best_fit)
lower_array.append(lower)
upper_array.append(upper)
print(f"Best-fit dipolar coupling: {best_fit}")
print(f"1 confidence interval: [{lower}, {upper}]")
plot_reduced_chi2_vs_gamma(gamma_numbers, red_chi2, best_fit, lower, upper)

```

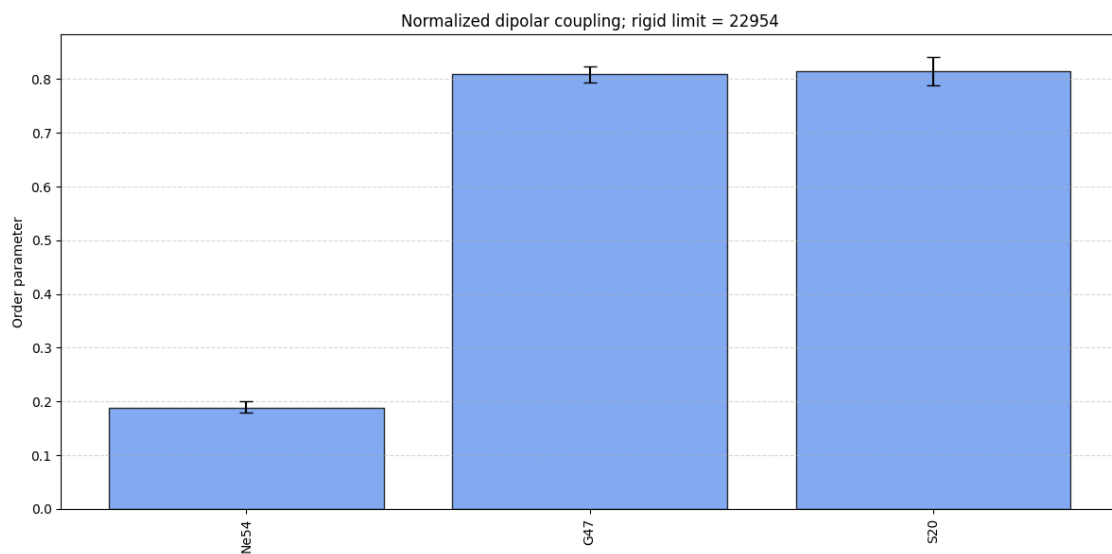
Best-fit dipolar coupling: 18600  
 1 confidence interval: [18200, 18900]



Best-fit dipolar coupling: 18700  
 1 confidence interval: [18100, 19300]



```
[52]: residuenumbers=['Ne54','G47','S20']
plot_normalized_asymmetric_barplot(best_fit_array, lower_array, upper_array,
    ↪residuenumbers,rigidNH)
best_fit_array_NH=best_fit_array
lower_array_NH=lower_array
upper_array_NH=upper_array
```

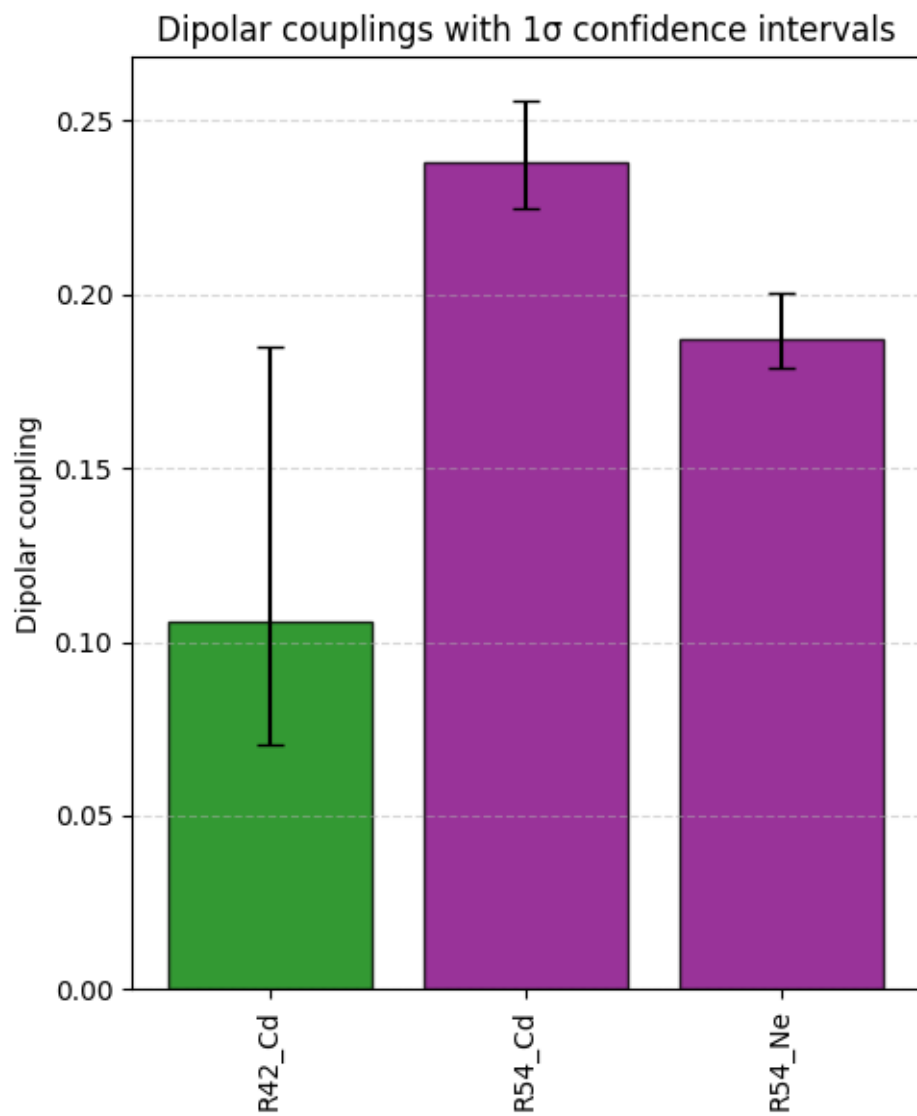


## 4 Summary of data and plot

```
[53]: residuenumbers=['R42_Cd', 'R54_Cd', 'R54_Ne']
```

```
[54]: best_fit_array_all=np.concatenate([np.array(best_fit_array_CH)/rigidCH, np.
    ↪array(best_fit_array_NH[:1])/rigidNH])
lower_lim_array_all=np.concatenate([np.array(lower_array_CH)/rigidCH, np.
    ↪array(lower_array_NH[:1])/rigidNH])
upper_lim_array_all=np.concatenate([np.array(upper_array_CH)/rigidCH, np.
    ↪array(upper_array_NH[:1])/rigidNH])
```

```
[55]: colors = ['green', 'purple', 'purple']
plot_asymmetric_barplot_with_colors(best_fit_array_all, lower_lim_array_all, u
    ↪pper_lim_array_all, residuenumbers, colors)
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: