figures_for_manuscript

June 13, 2024

1 Figures related to coarse graining in time

Error estimates are done by deviding simulation into 8 independent runs.

[1]:
```
from pprint import pprint

import matplotlib.pyplot as plt
import time
import numpy as np
import pandas as pd
import scipy.optimize
from scipy.ndimage import uniform_filter1d
from scipy.stats import sem, t

import analyse

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

```
/Users/jaehyeok/opt/anaconda3/lib/python3.9/site-
packages/pandas/core/computation/expressions.py:21: UserWarning: Pandas requires
version '2.8.4' or newer of 'numexpr' (version '2.8.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
/Users/jaehyeok/opt/anaconda3/lib/python3.9/site-
packages/pandas/core/arrays/masked.py:60: UserWarning: Pandas requires version
'1.3.6' or newer of 'bottleneck' (version '1.3.4' currently installed).
  from pandas.core import (
```

1.1 Loading thermodynamic data

Below we load thermodynamics data from LAMMPS-log files of two consecutive runs (`log.lammps` and `log_2nd.lammps`) into a Pandas Dataframe.

[2]:
```
# Load data from first part of run
tic = time.perf_counter()
first_frame = 0
frame_stride = 1
last_frame = None
time_step = 2e-15
```

```
steps_per_printout = 40
print(f'{first_frame=} {frame_stride=} {last_frame=} {steps_per_printout=}')
spns = int(1e-9 / time_step / steps_per_printout / frame_stride)  # Steps per␣
↪nanosecond
print(f'Steps per nanosecond: {spns = }')
df = analyse.thermo_data_as_dataframe(filename='../../log-file/T380_L35.944/log.
↪lammps',
                                      time_step=time_step,
                                      first_frame=first_frame,
                                      stride_frame=frame_stride,
                                      last_frame=last_frame)
toc = time.perf_counter()
print(f'Wallclock time to load data: {toc-tic} s')
```

```
first_frame=0 frame_stride=1 last_frame=None steps_per_printout=40
Steps per nanosecond: spns = 12500
Wallclock time to load data: 6.6113947500000005 s
```

[3]:

```
# Load data from 2nd part of run
join_trajectories = True
if join_trajectories:
    tic = time.perf_counter()
    df_2nd = analyse.thermo_data_as_dataframe(filename='../../log-file/T380_L35.
↪944/log_2nd.lammps',
                                              time_step=time_step,
                                              first_frame=first_frame,
                                              stride_frame=frame_stride,
                                              last_frame=last_frame)
    df_2nd['Time'] = df_2nd.Time + df.Time.max()
    df_2nd.head()
    df = pd.concat([df, df_2nd], axis=0)
    toc = time.perf_counter()
    print(f'Wallclock time to load data: {toc-tic} s')
```

```
Wallclock time to load data: 9.876580041999999 s
```

[4]:

```
# Compute inter-molecular energy (between molecules)
df['E_inter'] = df.PotEng - df.E_mol
```

[5]:

```
df.head()
```

[5]:

```
           Time   Step       Temp        Press     PotEng     KinEng  \
0  0.000000e+00    0.0  344.93695 -26171.19600  5196.6258  4111.7379
1  8.000000e-14   40.0  384.74831  -1813.19590  5029.1247  4586.2996
2  1.600000e-13   80.0  386.35782   2081.59420  4969.0675  4605.4855
3  2.400000e-13  120.0  386.06626   -548.78717  4866.6535  4602.0099
4  3.200000e-13  160.0  376.68499   2197.93110  5051.6645  4490.1828
```

```
      E_vdwl     E_coul      E_mol     Volume  Density     c_virial   E_inter
0 -647.60531  2486.3671  6072.3676  46438.611  1.02942 -30218.6240 -875.7418
1 -393.51470  2504.6959  5631.3149  46438.611  1.02942  -6327.7625 -602.1902
2 -359.18030  2485.0399  5558.2705  46438.611  1.02942  -2451.8581 -589.2030
3 -478.21544  2501.6095  5557.8171  46438.611  1.02942  -5078.8183 -691.1636
4 -390.06892  2511.7764  5645.6552  46438.611  1.02942  -2222.0219 -593.9907
```

[6]:
```
df.tail()
```

[6]:
```
                 Time         Step       Temp        Press     PotEng  \
3355440  4.684352e-07  134217600.0  381.98868 -811.602090  4984.2768
3355441  4.684353e-07  134217640.0  384.92741 -419.057880  5000.6772
3355442  4.684354e-07  134217680.0  386.93038 -274.394230  4967.5447
3355443  4.684354e-07  134217720.0  381.33474  640.577070  5004.1322
3355444  4.684355e-07  134217728.0  385.60780  -39.797283  5033.7336

            KinEng     E_vdwl     E_coul      E_mol     Volume  Density  \
3355440  4553.4042 -394.44657  2497.7761  5591.3391  46438.611  1.02942
3355441  4588.4345 -371.77400  2506.8891  5575.3857  46438.611  1.02942
3355442  4612.3105 -363.56890  2499.7962  5542.2153  46438.611  1.02942
3355443  4545.6090 -368.62174  2498.6347  5583.8211  46438.611  1.02942
3355444  4596.5450 -383.16558  2495.5294  5631.6976  46438.611  1.02942

           c_virial   E_inter
3355440 -5293.7877 -607.0623
3355441 -4935.7260 -574.7085
3355442 -4814.5648 -574.6706
3355443 -3833.9353 -579.6889
3355444 -4564.4489 -597.9640
```

[7]:
```
print(f'Trajectory of {df.Time.max()/1e-9} ns')
```

```
Trajectory of 468.435456 ns
```

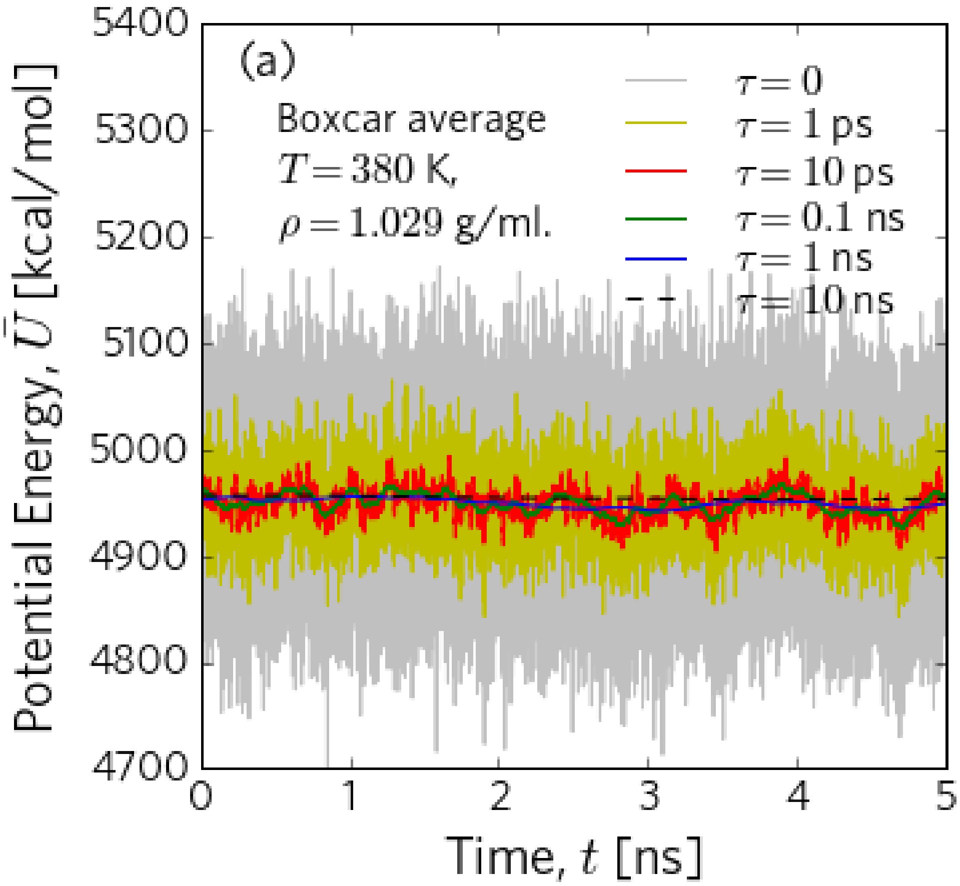
1.2 Analyses of boxcar averages

1.2.1 Figure 4(a)

[8]:
```
t_shift = df.Time*1e9-21  # Shifted time ("edged" are not computed correctly)
```

[35]:
```
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
})
```

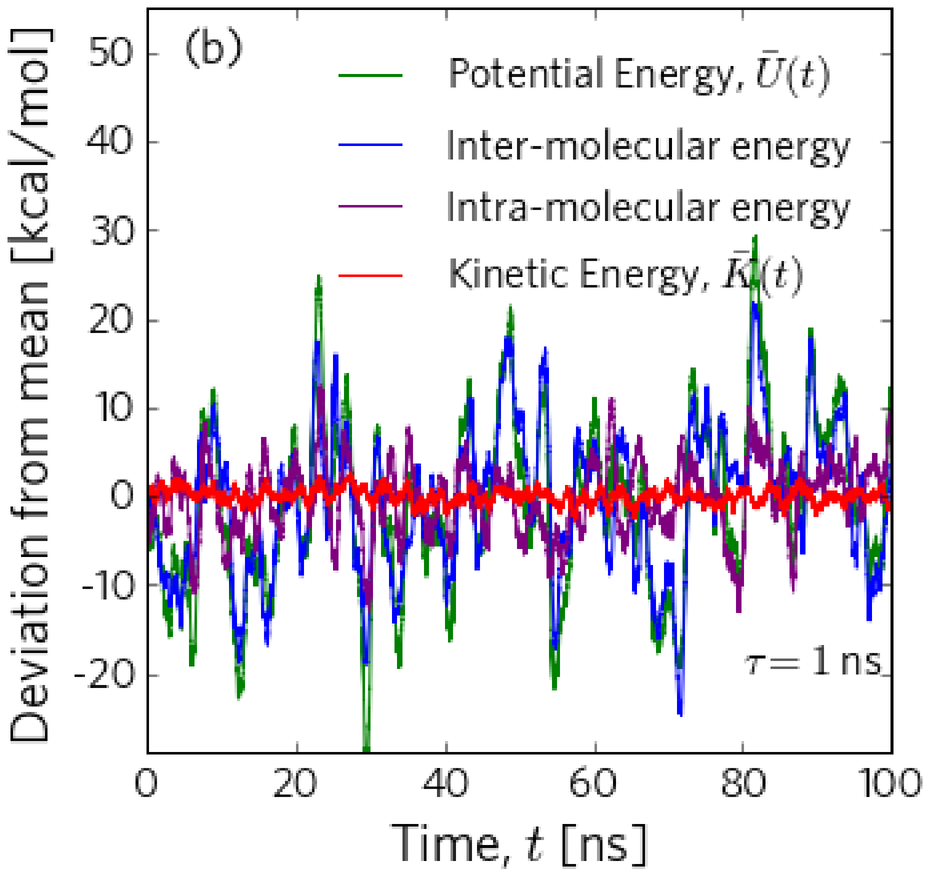
```
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.plot(t_shift, df.PotEng, '-', color='silver', label=r'$\tau=0$')
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns/1000)), 'y-', label=r'$\tau=1$ ps')
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns/100)), 'r-', label=r'$\tau=10$ ps')
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns/10)), 'g-', label=r'$\tau=0.1$ ns')
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns)), 'b-', label=r'$\tau=1$ ns')
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns)*10), 'k--', label=r'$\tau=10$ ns')
plt.legend(frameon=False, labelspacing=0.01, fontsize=16)
plt.xlabel(r'Time, $t$ [ns]',fontsize=20)
plt.ylabel(r'Potential Energy, $\bar U$ [kcal/mol]',fontsize=20)
plt.text(0.5, 5200, 'Boxcar average\n' + r'$T = 380$ K,' + '\n' + r'$\rho=1.029$ g/ml.',fontsize=16)
# Put (a) in the upper left corner
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes, fontsize=18)
plt.xlim(0, 5)
plt.ylim(4700, 5400)
plt.savefig('boxcar.pdf', bbox_inches='tight')
plt.savefig('boxcar.png', bbox_inches='tight')
```



1.2.2 Figure 4(b)

[34]:
```
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.plot(t_shift, uniform_filter1d(df.PotEng, int(spns))-df.PotEng.mean(),␣
 ↪'g-', label=r'Potential Energy, $\bar U(t)$')
plt.plot(t_shift, uniform_filter1d(df.E_inter, int(spns))-df.E_inter.mean(),␣
 ↪'b-', label=r'Inter-molecular energy')
plt.plot(t_shift, uniform_filter1d(df.E_mol, int(spns))-df.E_mol.mean(), '-',␣
 ↪color='purple', label=r'Intra-molecular energy')
```

```
plt.plot(t_shift, uniform_filter1d(df.KinEng, int(spns))-df.KinEng.mean(),␣
    ↪'r-', label=r'Kinetic Energy, $\bar K(t)$')
plt.xlabel(r'Time, $t$ [ns]', fontsize=20)
plt.ylabel(r'Deviation from mean [kcal/mol]', fontsize=20)
#plt.text(0.5, 5250, 'Boxcar average.\n' + r'$T = 380$ K,' + '\n' + r'$\rho=1.
    ↪030$ g/ml.')
plt.text(80, -20, r'$\tau=1$ ns', fontsize=16)
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes, fontsize=18)
plt.xlim(0, 100)
plt.ylim(-29, 55)
plt.legend(frameon=False, loc='upper right', fontsize=16)
plt.savefig('boxcar_energy.pdf', bbox_inches='tight')
plt.savefig('boxcar_energy.png', bbox_inches='tight')
```

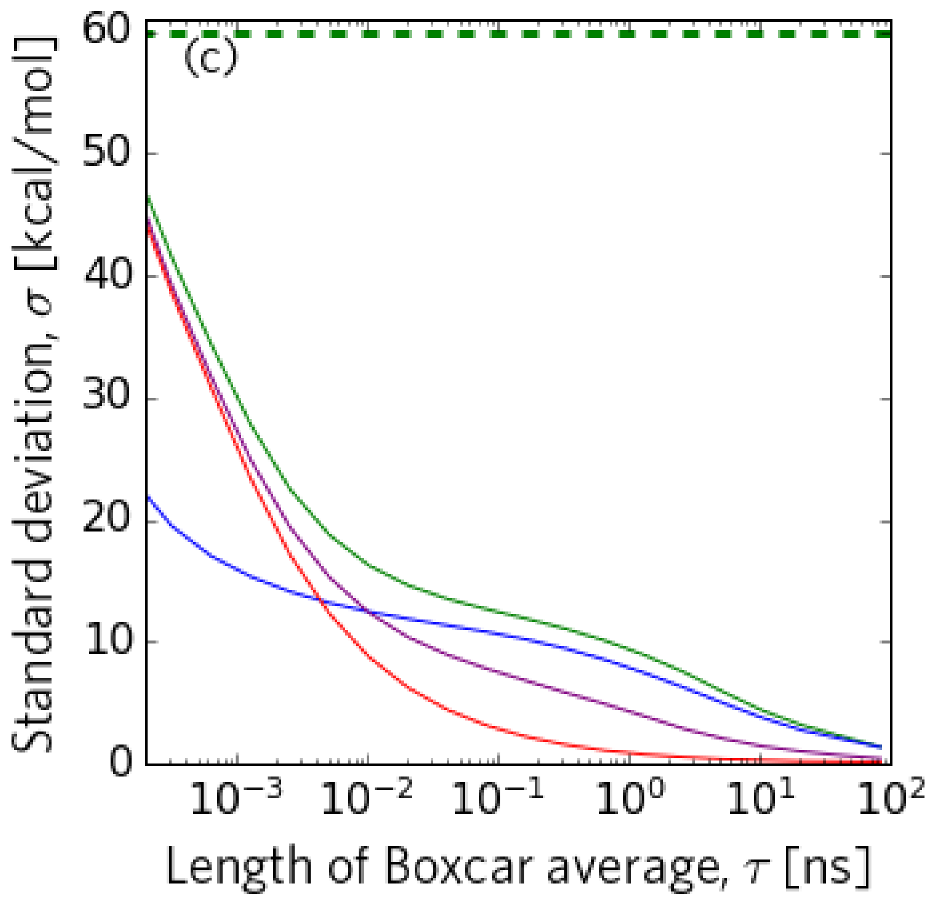


1.2.3 Figure 4(c)

[33]:
```
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to
  ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
def plot_curve(data, **kwarks):
    """ Compute standard deviation of box-car averages and add to plot """
    sigmas = []
    taus = []
    size = 1
    while size < len(data)/10:
        size=size*2
        taus.append(size/spns)
        sigmas.append(uniform_filter1d(data, size).std())
    plt.plot(taus, sigmas, '-', **kwarks)
    zero_level = data.std()
    #plt.plot([1e-4, 1e0], [zero_level]*2, '--', **kwarks)
    print(zero_level)

plot_curve(df.PotEng, color='g', label=r'Potential Energy, $\bar U(t)$')
plot_curve(df.E_inter, color='b', label=r'Inter-molecular energy')
plot_curve(df.E_mol, color='purple', label=r'Intra-molecular energy')
plot_curve(df.KinEng, color='r', label=r'Kinetic Energy, $\bar K(t)$')

zero_level = df.PotEng.std()
plt.plot([1e-4, 1e2], [zero_level]*2, 'g--', lw=3)
plt.ylabel(r'Standard deviation, $\sigma$ [kcal/mol]',fontsize=20)
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")

plt.xlabel(r'Length of Boxcar average, $\tau$ [ns]',fontsize=20)
plt.xscale('log')
plt.ylim(0, 61)
plt.xlim(2e-4,1e2)
#plt.legend(fontsize=16)
#plt.legend(frameon=False, loc='upper right', fontsize=12)
plt.text(0.05, 0.93, '(c)', transform=plt.gca().transAxes,fontsize=18)
#plt.text(5e-1, 25, 'Ortho-terphenyl:\n' + r'$T = 380$ K,' + '\n' + r'$\rho=1.
  ↪029$ g/ml.')
```

```
plt.savefig('boxcar_standard_deviation.pdf', bbox_inches='tight')
plt.savefig('boxcar_standard_deviation.png', bbox_inches='tight')
```

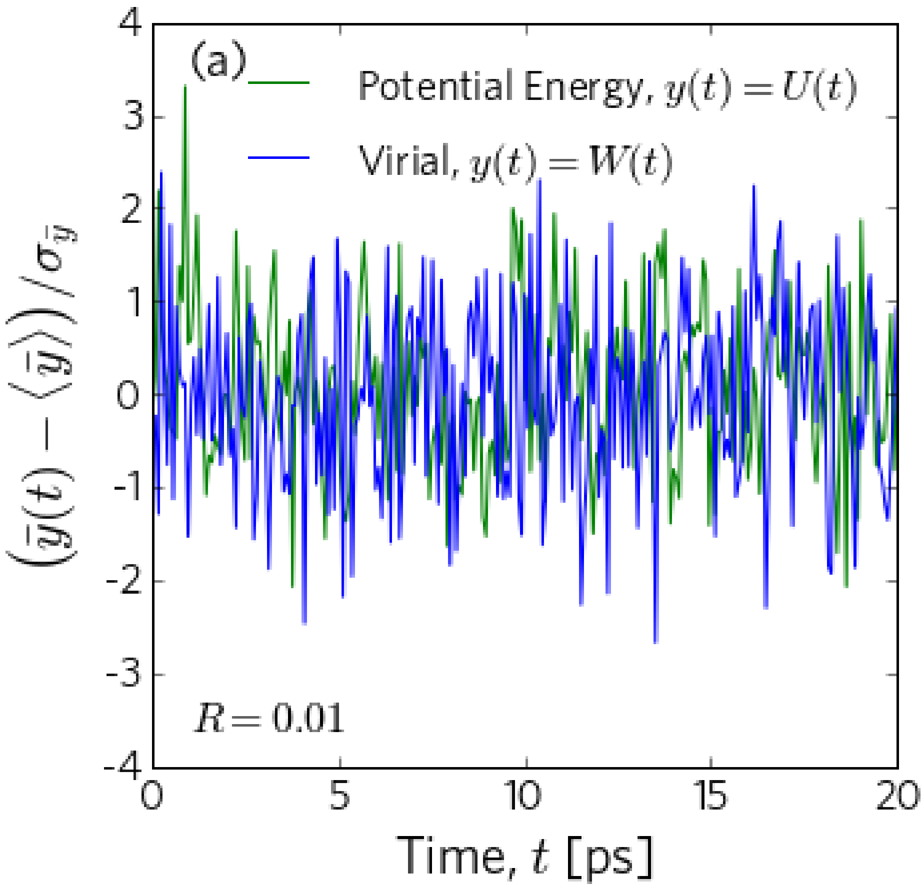
```
59.98063798289096
28.145155888217676
59.089076131091645
58.51167044926403
```



1.2.4 Figure 5(a)

```
[32]: plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
```

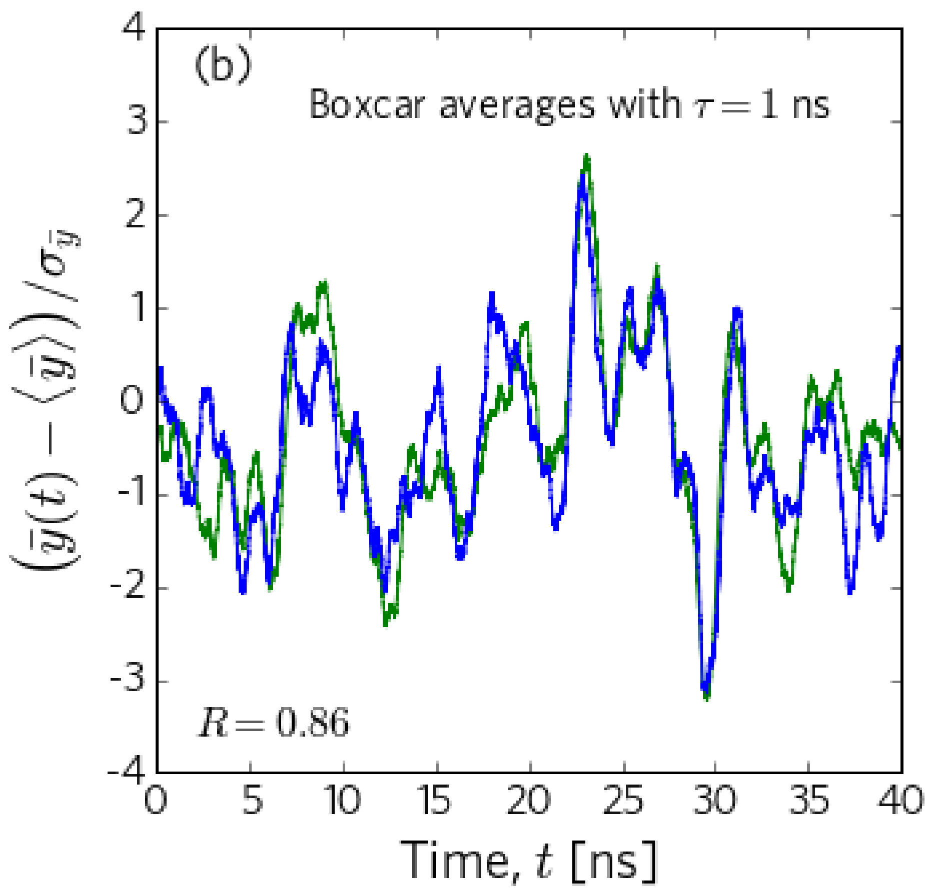
```
plt.yticks(fontsize=16)
y = df.PotEng-df.PotEng.mean()
plt.plot(t_shift*1e3, y/np.std(y), 'g-', label=r'Potential Energy, $y(t) = U(t)$')
y = df.c_virial-df.c_virial.mean()
plt.plot(t_shift*1e3, y/np.std(y), 'b-', label=r'Virial, $y(t) = W(t)$')
plt.xlabel(r'Time, $t$ [ps]', fontsize=20)
plt.ylabel(r'$\left(\bar y(t)-\langle \bar y\rangle\right)/\sigma_{\bar y}$', fontsize=20)
#plt.text(0.5, 5250, 'Boxcar average.\n' + r'$T = 380$ K,' + '\n' + r'$\rho=1.030$ g/ml.')
#plt.text(0.1, 0.1, r'$\tau=0$ ns', transform=plt.gca().transAxes)
corrcoef = np.corrcoef(df.PotEng, df.c_virial)[0,1]
plt.text(0.05, 0.05, f'$R = {corrcoef:.2f}$', transform=plt.gca().transAxes, ha='left', fontsize=16)
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes, fontsize=18)
plt.xlim(0, 20)
plt.ylim(-4, 4)
plt.legend(frameon=False, loc='upper right', fontsize=16)
plt.savefig('boxcar_virial_energy_fast.pdf', bbox_inches='tight')
plt.savefig('boxcar_virial_energy_fast.png', bbox_inches='tight')
```



1.2.5 Figure 5(b)

[31]:
```
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
y = uniform_filter1d(df.PotEng, int(spns))-df.PotEng.mean()
plt.plot(t_shift, y/np.std(y), 'g-', label=r'Potential Energy, $\bar y(t) =␣
 ↪\bar U(t; \tau)$')
y = uniform_filter1d(df.c_virial, int(spns))-df.c_virial.mean()
plt.plot(t_shift, y/np.std(y), 'b-', label=r'Virial, $\bar y(t) = \bar W(t;␣
 ↪\tau)$')
plt.xlabel(r'Time, $t$ [ns]', fontsize=20)
```

```python
plt.ylabel(r'$\left(\bar y(t)-\langle \bar y\rangle\right)/\sigma_{\bar y}$',
    fontsize=20)
#plt.text(0.5, 5250, 'Boxcar average.\n' + r'$T = 380$ K,' + '\n' + r'$\rho=1.
    030$ g/ml.')
plt.text(0.2, 0.88, r'Boxcar averages with $\tau=1$ ns', transform=plt.gca().
    transAxes, fontsize=16)
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes, fontsize=18)
plt.text(0.05, 0.05, f'$R = 0.86$', transform=plt.gca().transAxes, ha='left',
    fontsize=16)
plt.xlim(0, 40)
#plt.ylim(-29, 55)
#plt.legend(frameon=False, loc='upper right', fontsize=12)
plt.savefig('boxcar_virial_energy.pdf', bbox_inches='tight')
plt.savefig('boxcar_virial_energy.png', bbox_inches='tight')
```



1.2.6 Figure 6(a) and 6(b)

[36]:
```
def compute_boxcar_R_gamma(size, blocks=1, block_id=0):
    ''' Compute R and gamma for a part of the trajectory '''
    if not block_id < blocks:
        raise ValueError(f'Reduce {block_id=} to be less than {blocks=}')
    U = np.array(df.PotEng)
    W = np.array(df.c_virial)
    block_len = int(len(U)/blocks)
    first = block_id*block_len
    last = (block_id+1)*block_len
    U = U[first:last]
    W = W[first:last]
    A = uniform_filter1d(U, size)
    B = uniform_filter1d(W, size)
    A = A[size:-size]
    B = B[size:-size]
    R = np.corrcoef(A, B)
    cov = np.cov(A, B)
    return {'R': R[0, 1], 'gamma': cov[0, 1]/A.var()}
compute_boxcar_R_gamma(spns)
```

[36]: {'R': 0.8176082763310751, 'gamma': 6.264484217720644}

[37]:
```
def error_bars_by_blocking(size, blocks=8):
    means = compute_boxcar_R_gamma(size)
    R = means['R']
    gamma = means['gamma']

    Rs = [compute_boxcar_R_gamma(size, blocks=blocks, block_id=idx)['R'] for␣
 ↪idx in range(blocks)]
    gammas = [compute_boxcar_R_gamma(size, blocks=blocks,␣
 ↪block_id=idx)['gamma'] for idx in range(blocks)]

    # Calculate the standard error of the mean (SEM)
    R_err = sem(Rs)
    gamma_err = sem(gammas)

    # Compute the 67% confidence intervals
    confidence_level = 0.67
    degrees_freedom = len(Rs) - 1
    t_multiplier = t.ppf((1 + confidence_level) / 2, degrees_freedom)
    R_err *= t_multiplier
    gamma_err *= t_multiplier

    return {'R': R, 'gamma': gamma,
            'R_err': R_err, 'gamma_err': gamma_err,
```

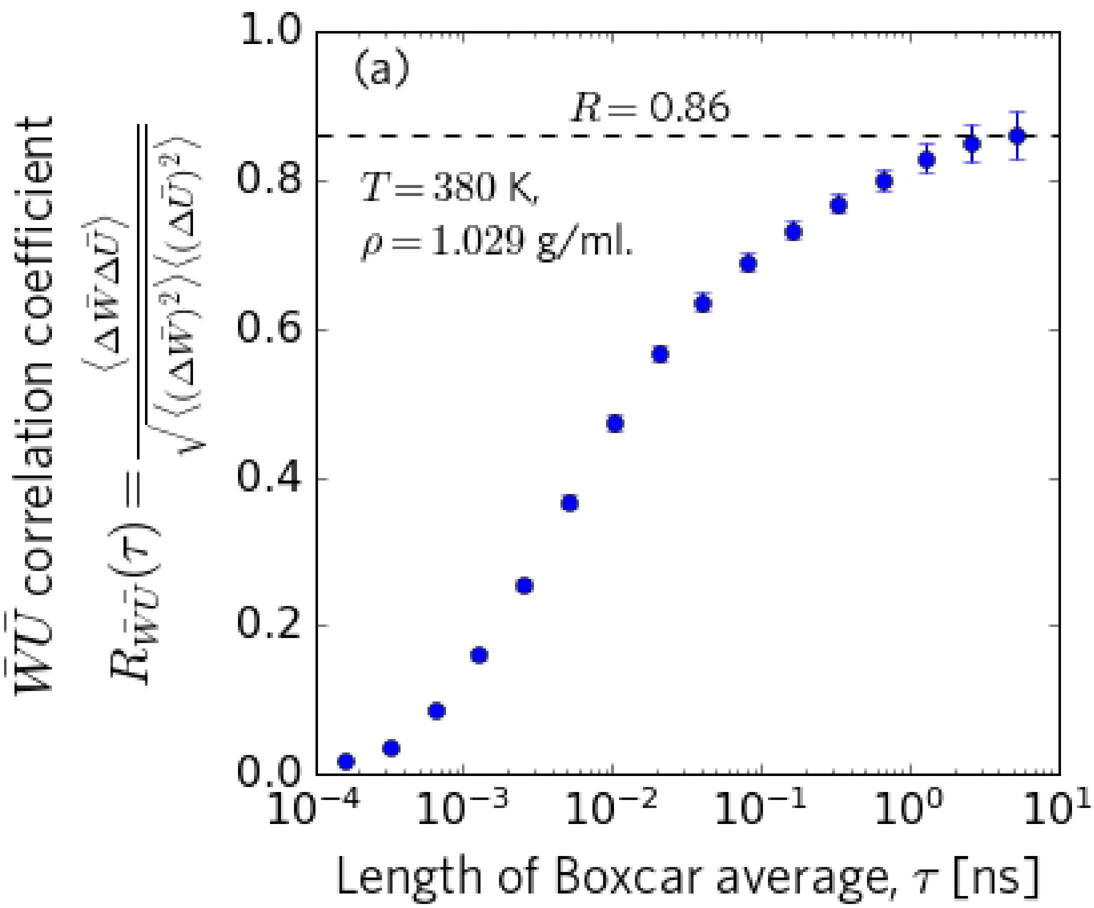
```
             'size': size, 'blocks': blocks}

error_bars_by_blocking(spns)
```

[37]:
```
{'R': 0.8176082763310751,
 'gamma': 6.264484217720644,
 'R_err': 0.017937187537449988,
 'gamma_err': 0.2641514450949758,
 'size': 12500,
 'blocks': 8}
```

[46]:
```
# Computes R and gamma and error estimates
#   for different sizes of boxcar averages
Rs, Rs_err = [], []
gammas, gammas_err = [], []
taus = []
size = 1

while size < len(df.PotEng)/16:
    taus.append(size/spns)
    tmp = error_bars_by_blocking(size)
    Rs.append(tmp['R'])
    Rs_err.append(tmp['R_err'])
    gammas.append(tmp['gamma'])
    gammas_err.append(tmp['gamma_err'])
    # pprint(tmp)
    size *= 2
```

[47]:
```
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
taus_new = taus[:-2]
Rs_new = Rs[:-2]
Rs_err_new = Rs_err[:-2]
plt.errorbar(taus_new, Rs_new, Rs_err_new, fmt='bo')
R_long_time = 0.86
plt.plot([1e-4, 100], [R_long_time]*2, 'k--')
plt.text(5e-3, 0.9, r'$R=$' f'{R_long_time}', va='center',fontsize=16)
plt.xscale('log')
plt.xlabel(r'Length of Boxcar average, $\tau$ [ns]',fontsize=20)
```

```
plt.ylabel(r'$\bar W$$\bar U$ correlation coefficient' '\n' r'$R_{\bar W\bar␣
↪U}(\tau)=\frac{\langle \Delta \bar W\Delta \bar U\rangle}{\sqrt{\langle␣
↪(\Delta \bar W)^2\rangle\langle (\Delta \bar U)^2\rangle}}$',fontsize=20)
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes, fontsize=18)
plt.text(2e-4, 0.7,'$T = 380$ K,' + '\n' + r'$\rho=1.029$ g/ml.', fontsize=16)
plt.ylim(0, 1)
plt.xlim(1e-4, 1e1)
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.savefig('boxcar_correlation_coefficient.pdf', bbox_inches='tight')
plt.savefig('boxcar_correlation_coefficient.png', bbox_inches='tight')
```

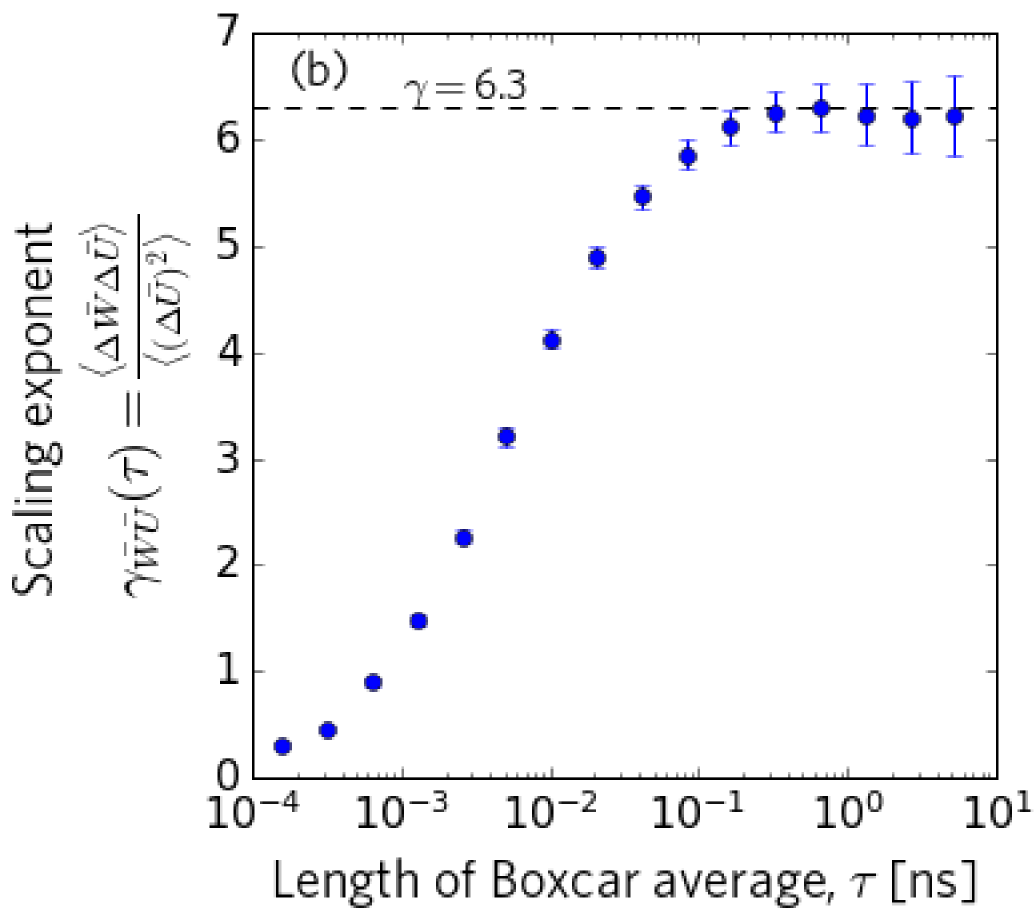


[41]:
```
taus
```

[41]:
```
[8e-05,
 0.00016,
 0.00032,
 0.00064,
```

```
 0.00128,
 0.00256,
 0.00512,
 0.01024,
 0.02048,
 0.04096,
 0.08192,
 0.16384,
 0.32768,
 0.65536,
 1.31072,
 2.62144,
 5.24288,
 10.48576,
 20.97152]
```

[51]:
```python
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
gammas_new = gammas[:-2]
gammas_err_new = gammas_err[:-2]
plt.errorbar(taus_new, gammas_new, gammas_err_new, fmt='bo')
gamma_long_time = 6.3
plt.plot([1e-4, 100], [gamma_long_time]*2, 'k--')
plt.text(1e-3, gamma_long_time+0.2, r'$\gamma=$' f'{gamma_long_time}',␣
 ↪va='center',fontsize=16)
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes,fontsize=18)
plt.xscale('log')
plt.xlabel(r'Length of Boxcar average, $\tau$ [ns]',fontsize=20)
plt.ylabel(r'Scaling exponent' '\n' r'$\gamma_{\bar W\bar␣
 ↪U}(\tau)=\frac{\langle\Delta\bar W\Delta\bar U\rangle}{\langle(\Delta\bar␣
 ↪U)^2\rangle}$',fontsize=20)
#plt.text(2e-1, 0.7, r'ortho-terphenyl,' + '\n' + '$T = 380$ K,' + '\n' +␣
 ↪r'$\rho=1.030$ g/ml.')
#plt.ylim(0, 1)
plt.xlim(1e-4, 1e1)
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.savefig('boxcar_scaling_exponent.pdf', bbox_inches='tight')
plt.savefig('boxcar_scaling_exponent.png', bbox_inches='tight')
```



1.2.7 Figure 7(a) - 7(c)

[52]:
```
def time_correlation_function_one_block(blocks=8, block_id=0,
    ↪points_per_decade=6):
    if not block_id < blocks:
        raise ValueError(f'Reduce {block_id=} to be less than {blocks=}')
    U = np.array(df.PotEng)
    W = np.array(df.c_virial)
    block_len = int(len(U)/blocks)
    first = block_id*block_len
    last = (block_id+1)*block_len
    U = U[first:last]
    W = W[first:last]

    C_UU = analyse.time_correlation(U)
    C_WW = analyse.time_correlation(W)
    C_WU = analyse.time_correlation(W, U)
```

```
    C_UU_log = analyse.run_avg_log(C_UU, points_per_decade)
    C_WW_log = analyse.run_avg_log(C_WW, points_per_decade)
    C_WU_log = analyse.run_avg_log(C_WU, points_per_decade)
    t_log = analyse.run_avg_log(df.Time, points_per_decade)
    t_log = t_log[0:len(C_UU_log)]

    return {
        't': t_log,
        'C_UU': C_UU_log,
        'C_WW': C_WW_log,
        'C_WU': C_WU_log
    }

time_correlation_function_one_block()
```

[52]:
```
{'t': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
        3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
        2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
        9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
        4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
        2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
        9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
        4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
        2.126676e-08, 3.121536e-08, 4.581796e-08]),
 'C_UU': array([ 3.59630018e+03,  1.29803907e+03,  9.37460693e+02,
7.74567971e+02,
         5.24100115e+02,  3.21903037e+02,  2.35109040e+02,  2.01437452e+02,
         1.89837609e+02,  1.89376680e+02,  1.78951907e+02,  1.73615963e+02,
         1.65208545e+02,  1.57106096e+02,  1.53288271e+02,  1.46509353e+02,
         1.37779275e+02,  1.30042523e+02,  1.20817065e+02,  1.10436092e+02,
         1.01487088e+02,  9.14353366e+01,  7.69680021e+01,  5.56559469e+01,
         3.05355189e+01,  1.34590220e+01, -1.30748632e+00, -1.03227303e+01,
        -3.25689996e+00, -9.10919997e+00, -4.65848850e+00, -2.80107463e+00,
        -4.55528420e+00,  3.75649737e+00, -1.23012401e+00]),
 'C_WW': array([ 1.21851057e+06,  5.71292428e+04, -1.96426933e+05,
2.92901644e+04,
        -6.22874951e+04,  3.90543672e+04,  1.76333235e+04,  1.40468615e+04,
         1.46137334e+04,  1.29833788e+04,  1.29037303e+04,  1.20844873e+04,
         1.16709039e+04,  1.07777630e+04,  1.01397370e+04,  9.48122439e+03,
         8.95535476e+03,  8.14423715e+03,  7.33543056e+03,  6.58202198e+03,
         5.65122110e+03,  4.26363854e+03,  3.13328372e+03,  1.43253827e+03,
         6.08087224e+02,  3.33553470e+02, -4.08167371e+02, -3.20365610e+02,
         3.47226101e+02, -1.24061077e+02, -4.73382065e+02, -4.26411324e+02,
        -8.50958894e+01,  2.85287606e+02, -1.31104860e+02]),
 'C_WU': array([ 809.26397896,  641.88747299,  407.59000533, 1624.53219587,
        1595.6622273 , 1215.94461806, 1136.37177847, 1186.37564543,
        1105.23263292, 1123.51260835, 1076.35629439, 1023.60818074,
```

```
        1011.53152209,   971.09731761,   931.93654263,   926.66432408,
         897.7290301 ,   844.80802199,   783.71300086,   727.63678404,
         666.19463551,   618.60748241,   525.83676413,   354.99766935,
         217.51788218,   100.79667431,    -7.9473606 ,   -95.52632842,
           6.6007547 ,   -26.182338  ,   -68.59751135,   -36.1327896 ,
         -28.07730725,    31.34416559,    -8.95034982])}
```

[53]:
```python
%%time
block_data = []
blocks = 8
for block_id in range(blocks):
    tmp = time_correlation_function_one_block(blocks=blocks, block_id=block_id)
    pprint(tmp)
    block_data.append(tmp)
```

```
{'C_UU': array([ 3.59630018e+03,  1.29803907e+03,  9.37460693e+02,
7.74567971e+02,
         5.24100115e+02,  3.21903037e+02,  2.35109040e+02,  2.01437452e+02,
         1.89837609e+02,  1.89376680e+02,  1.78951907e+02,  1.73615963e+02,
         1.65208545e+02,  1.57106096e+02,  1.53288271e+02,  1.46509353e+02,
         1.37779275e+02,  1.30042523e+02,  1.20817065e+02,  1.10436092e+02,
         1.01487088e+02,  9.14353366e+01,  7.69680021e+01,  5.56559469e+01,
         3.05355189e+01,  1.34590220e+01, -1.30748632e+00, -1.03227303e+01,
        -3.25689996e+00, -9.10919997e+00, -4.65848850e+00, -2.80107463e+00,
        -4.55528420e+00,  3.75649737e+00, -1.23012401e+00]),
 'C_WU': array([ 809.26397896,  641.88747299,  407.59000533, 1624.53219587,
        1595.6622273 , 1215.94461806, 1136.37177847, 1186.37564543,
        1105.23263292, 1123.51260835, 1076.35629439, 1023.60818074,
        1011.53152209,   971.09731761,   931.93654263,   926.66432408,
         897.7290301 ,   844.80802199,   783.71300086,   727.63678404,
         666.19463551,   618.60748241,   525.83676413,   354.99766935,
         217.51788218,   100.79667431,    -7.9473606 ,   -95.52632842,
           6.6007547 ,   -26.182338  ,   -68.59751135,   -36.1327896 ,
         -28.07730725,    31.34416559,    -8.95034982]),
 'C_WW': array([ 1.21851057e+06,  5.71292428e+04, -1.96426933e+05,
2.92901644e+04,
        -6.22874951e+04,  3.90543672e+04,  1.76333235e+04,  1.40468615e+04,
         1.46137334e+04,  1.29833788e+04,  1.29037303e+04,  1.20844873e+04,
         1.16709039e+04,  1.07777630e+04,  1.01397370e+04,  9.48122439e+03,
         8.95535476e+03,  8.14423715e+03,  7.33543056e+03,  6.58202198e+03,
         5.65122110e+03,  4.26363854e+03,  3.13328372e+03,  1.43253827e+03,
         6.08087224e+02,  3.33553470e+02, -4.08167371e+02, -3.20365610e+02,
         3.47226101e+02, -1.24061077e+02, -4.73382065e+02, -4.26411324e+02,
        -8.50958894e+01,  2.85287606e+02, -1.31104860e+02]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
        3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
        2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
```

```
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.59340472e+03,  1.29590437e+03,  9.39245685e+02,
7.68639444e+02,
        5.24197297e+02,  3.15104177e+02,  2.22034536e+02,  1.97763569e+02,
        1.83315173e+02,  1.78156530e+02,  1.68513174e+02,  1.62508123e+02,
        1.59501170e+02,  1.53184661e+02,  1.44743600e+02,  1.38816624e+02,
        1.32545548e+02,  1.23940556e+02,  1.14596191e+02,  1.03688196e+02,
        9.25538115e+01,  8.23696698e+01,  7.42873275e+01,  6.04893418e+01,
        4.51966871e+01,  2.31611974e+01,  4.88525488e+00, -4.97714720e+00,
       -4.75647906e+00, -3.29543502e+00,  2.44206677e-01,  4.29408688e+00,
       -1.18472900e+01,  7.82260337e-01, -8.64246799e-01]),
 'C_WU': array([ 720.16050352,  626.50095613,  391.65629596, 1420.71432831,
       1512.13538211, 1130.91051084, 1045.59404419, 1020.09859397,
       1004.99577055,  989.2531853 , 1031.2351162 ,  976.27154258,
        912.56721279,  911.94781865,  881.62606971,  839.39210331,
        817.39431795,  754.85713218,  711.92631272,  648.76370153,
        590.41969464,  543.23810698,  500.04070688,  373.2217206 ,
        319.92260838,  219.76612539,  143.96771497,   54.22726801,
        -82.11093683, -157.25279046,  -63.53011949,   16.6337166 ,
       -104.33318608,   35.44114161,    1.58629534]),
 'C_WW': array([ 1.21864964e+06,  5.43089258e+04, -1.94562229e+05,
3.26717106e+04,
       -6.44805316e+04,  3.85341302e+04,  1.74513401e+04,  1.41993210e+04,
        1.42572613e+04,  1.35480310e+04,  1.24790116e+04,  1.17785211e+04,
        1.15514920e+04,  1.08128161e+04,  1.01740835e+04,  9.43665338e+03,
        8.86495055e+03,  7.89311200e+03,  6.85552547e+03,  5.93040136e+03,
        5.29061406e+03,  4.75568369e+03,  4.08579267e+03,  2.92953320e+03,
        2.43003287e+03,  1.78499416e+03,  1.40210684e+03,  1.98087516e+02,
       -6.57481567e+02, -1.27361010e+03, -5.64676401e+02,  7.96071636e+02,
       -1.01625340e+03,  2.06717733e+02,  6.14843028e+01]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.57780293e+03,  1.27689543e+03,  9.15261786e+02,
7.48146622e+02,
        5.03794426e+02,  3.01420668e+02,  2.14444974e+02,  1.82112854e+02,
        1.72038416e+02,  1.70542441e+02,  1.63475168e+02,  1.55368362e+02,
```

```
        1.51235144e+02,  1.42038993e+02,  1.35310343e+02,  1.26850052e+02,
        1.20799715e+02,  1.10159234e+02,  9.86142328e+01,  8.89849162e+01,
        7.70056790e+01,  6.15480462e+01,  5.10542710e+01,  3.65302227e+01,
        2.53809820e+01,  1.81648326e+01,  1.50824264e+01, -1.01811636e+00,
       -6.28844808e+00, -1.30212789e+01, -1.39872462e+01,  5.03831256e-01,
        1.12016406e+00,  5.15545283e-01, -1.74750455e-01]),
 'C_WU': array([ 820.31852891,  789.88366892,  547.41051207, 1473.71711787,
       1537.5900022 , 1298.1583346 , 1169.69183092, 1135.12687281,
       1071.40279431, 1046.45188426, 1052.17433382,  997.87511684,
        981.29870443,  930.83340334,  915.69971996,  870.07518607,
        832.05566542,  755.42871674,  674.03916267,  632.09236559,
        569.79493754,  463.73142857,  395.38069288,  268.1060842 ,
        164.90624508,  153.08764855,  100.76345324,  -26.84967396,
        -86.46772298, -102.19506797,  -68.18628145,   25.82830503,
         -2.25729642,    2.43936066,   -2.30654574]),
 'C_WW': array([ 1.21843201e+06,  5.70258730e+04, -1.95221720e+05,
3.30205131e+04,
       -6.24653828e+04,  3.95236756e+04,  1.86538281e+04,  1.47694860e+04,
        1.50625360e+04,  1.40501780e+04,  1.35099055e+04,  1.29423647e+04,
        1.23374151e+04,  1.14858048e+04,  1.07906283e+04,  1.02307319e+04,
        9.29090442e+03,  8.36971752e+03,  7.27418924e+03,  6.45079669e+03,
        5.58546777e+03,  4.19902665e+03,  3.14496815e+03,  1.96756286e+03,
        1.07825879e+03,  1.50445466e+03,  8.28156936e+02, -4.85166717e+02,
       -5.87919612e+02, -8.06646415e+02, -5.69947661e+02,  7.79535116e+01,
        2.71408163e+02, -2.08901471e+02,  3.95341427e+01]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.56390082e+03,  1.25735778e+03,  8.98862102e+02,
7.34721004e+02,
        4.83934696e+02,  2.85955672e+02,  1.92588854e+02,  1.63688695e+02,
        1.55449688e+02,  1.46802079e+02,  1.37837018e+02,  1.30927268e+02,
        1.26855607e+02,  1.20452311e+02,  1.13537882e+02,  1.08902514e+02,
        1.01214504e+02,  9.08888017e+01,  8.20816206e+01,  7.08646470e+01,
        5.77337517e+01,  4.30872885e+01,  2.72290533e+01,  1.90738758e+01,
        8.94685448e+00,  7.35724176e+00,  1.23124890e+01,  1.44352953e+01,
        1.93911437e+00,  3.60105994e+00,  6.59994327e+00,  3.19912237e+00,
       -4.18154039e+00, -5.26713234e+00, -1.62118382e+00]),
 'C_WU': array([ 594.39608879,  491.29444014,  406.21079011, 1511.11049783,
       1469.67474699, 1093.58418908,  975.90640151, 1033.65645947,
        997.70821886,  966.34809666,  894.61930719,  882.50953631,
        830.41162063,  826.20652679,  764.12193439,  737.36218486,
```

```
       707.36317886,  656.22699621,  588.14825784,  496.19764237,
       440.1250641 ,  322.13075994,  167.18432404,   87.36332878,
        14.16902926,   29.11329444,   97.71639449,   89.15866237,
        14.48145085,  -16.58305315,   32.50038784,   33.45055236,
       -26.08853044,  -38.22554196,  -12.80658479]),
 'C_WW': array([ 1.21776193e+06,  5.50247034e+04, -1.96071200e+05,
3.39564454e+04,
       -6.32772414e+04,  3.74423846e+04,  1.77793736e+04,  1.39833113e+04,
        1.41080115e+04,  1.36323005e+04,  1.31779878e+04,  1.20546747e+04,
        1.15715921e+04,  1.07010633e+04,  1.01265002e+04,  9.25826204e+03,
        8.27161712e+03,  7.59143966e+03,  6.55784555e+03,  5.86455103e+03,
        4.72890762e+03,  3.62769028e+03,  2.18368204e+03,  5.83538867e+02,
        1.46705699e+02,  3.21460512e+02,  7.51423811e+02,  8.08687295e+02,
        2.24425480e+02, -3.17675029e+02,  3.30462128e+02,  2.67697974e+02,
       -1.69692719e+02, -2.31678680e+02, -1.49860328e+02]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.63073888e+03,  1.33593464e+03,  9.71907498e+02,
8.04772155e+02,
        5.55988254e+02,  3.55351687e+02,  2.65042520e+02,  2.30137944e+02,
        2.13301986e+02,  2.05601082e+02,  2.02961784e+02,  1.95154247e+02,
        1.92570049e+02,  1.82771038e+02,  1.77398001e+02,  1.68547199e+02,
        1.58424817e+02,  1.50968218e+02,  1.39919258e+02,  1.27287947e+02,
        1.17421207e+02,  1.01743553e+02,  8.82704154e+01,  7.53256437e+01,
        5.43278123e+01,  3.50196283e+01,  2.14726715e+01, -3.12589448e+00,
       -1.22554955e+01, -4.14362110e+00, -5.20517976e+00, -1.04654531e+01,
        7.18657491e+00, -6.09156040e-01, -4.73004997e+00]),
 'C_WU': array([1061.53160736, 1005.45712118,  855.39525306, 1825.81250493,
       1848.16955789, 1516.51777116, 1356.86398853, 1415.07544937,
       1303.65169405, 1321.55833025, 1303.44245854, 1270.67995678,
       1219.10545511, 1219.25481974, 1159.97729609, 1122.32080587,
       1065.06459431,  992.61054104,  942.1813373 ,  861.31776414,
        791.55838663,  671.43932625,  601.93634205,  550.72073899,
        380.16461814,  175.1523889 ,   41.56334149,   27.06246047,
         -2.07603085,  -22.61331225,  -57.23402942, -132.15575068,
         74.96469137,   13.67994942,  -33.69689275]),
 'C_WW': array([ 1.22091735e+06,  6.00789814e+04, -1.92679882e+05,
3.17917971e+04,
       -6.18850585e+04,  4.15078564e+04,  1.96600923e+04,  1.64765237e+04,
        1.67329336e+04,  1.51132381e+04,  1.46570570e+04,  1.40088660e+04,
        1.34656040e+04,  1.27266672e+04,  1.20641322e+04,  1.16247500e+04,
```

```
        1.10411058e+04,  1.01639573e+04,  9.06567761e+03,  7.86437198e+03,
        6.48292141e+03,  5.40011242e+03,  4.74925104e+03,  4.16303392e+03,
        2.62448553e+03,  1.57308236e+03,  4.06528873e+02, -3.39689034e+02,
       -4.60857202e+02, -3.44700903e+02, -1.37761947e+02, -7.72858214e+02,
        1.72090883e+02,  1.11058087e+02, -1.67801571e+02]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.61475408e+03,  1.31785484e+03,  9.52695620e+02,
7.84878335e+02,
        5.41421867e+02,  3.44202854e+02,  2.48797463e+02,  2.15749398e+02,
        2.10883356e+02,  1.95737719e+02,  1.89810414e+02,  1.85309125e+02,
        1.75939354e+02,  1.71501864e+02,  1.65959160e+02,  1.57220389e+02,
        1.50108418e+02,  1.42307719e+02,  1.32665552e+02,  1.21136899e+02,
        1.05445008e+02,  8.87860155e+01,  7.64805543e+01,  6.75517199e+01,
        5.04393988e+01,  2.23104052e+01,  2.93602965e+00, -2.18404498e+01,
       -1.89121778e+01,  2.19694082e-02,  6.98237187e+00, -8.03928170e+00,
       -1.39877814e-01, -2.90279457e+00,  3.91512032e-01]),
 'C_WU': array([ 749.43522414,  694.00797025,  567.8047468 , 1624.41926182,
       1519.69735717, 1143.97767928, 1173.65432913, 1162.61556683,
       1141.63012651, 1098.96380117, 1076.63202599, 1062.89103781,
       1022.76572775,  997.1294103 ,  975.99075965,  922.66766057,
        884.71743278,  841.24139545,  798.57860353,  711.88389363,
        615.41596594,  518.74973554,  457.25973052,  402.85685371,
        283.09241316,  213.42193641,  104.62336364,  -35.93591737,
        -63.92414768,  -52.94513745,  103.35079445,  -16.73416369,
         11.67850163,  -15.24594897,  -13.46579407]),
 'C_WW': array([ 1.22022882e+06,  5.67705881e+04, -1.96741669e+05,
3.16777741e+04,
       -6.23692020e+04,  3.91683041e+04,  1.81241759e+04,  1.47533673e+04,
        1.45068483e+04,  1.37644861e+04,  1.31082768e+04,  1.24661889e+04,
        1.18580752e+04,  1.10365865e+04,  1.04825626e+04,  9.68463513e+03,
        8.68430020e+03,  8.09423725e+03,  7.23318100e+03,  5.98808980e+03,
        5.16809180e+03,  4.30554542e+03,  3.42977010e+03,  2.42641587e+03,
        1.59044216e+03,  1.04991741e+03,  1.95095305e+02, -6.78679775e+02,
       -4.28036208e+02, -1.07711761e+01,  6.52700092e+02, -2.59683268e+02,
       -1.72876869e+01, -1.78556560e+02, -1.05231353e+02]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
```

```
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.57950516e+03,  1.28616107e+03,  9.26840900e+02,
7.60163203e+02,
        5.16700400e+02,  3.13766776e+02,  2.21871410e+02,  1.90017943e+02,
        1.81873485e+02,  1.72767275e+02,  1.66557883e+02,  1.57291138e+02,
        1.52835916e+02,  1.45482748e+02,  1.40991557e+02,  1.32523844e+02,
        1.28636801e+02,  1.19489254e+02,  1.10504836e+02,  9.83106390e+01,
        8.47698379e+01,  6.85103670e+01,  5.31482787e+01,  3.75926867e+01,
        3.14632663e+01,  2.92321553e+01,  1.91157829e+01,  2.55325607e+01,
        2.47203248e+01,  1.97014700e+01,  9.34531474e+00,  5.70606992e+00,
       -1.85643222e+00, -1.29483644e+01, -7.00588090e+00]),
 'C_WU': array([ 721.32868398,  739.87095002,  374.60362454, 1238.37203085,
       1484.30971742, 1142.78819941, 1043.04604151, 1054.27264631,
        995.770184  ,  988.38819397,  884.01343323,  907.64134139,
        873.30261516,  847.37281418,  823.30156408,  770.46672803,
        744.59806456,  730.23391705,  657.71946129,  600.09734664,
        547.10377576,  446.4346635 ,  305.99661901,  179.57886772,
         95.04115823,  119.10133764,   67.54339244,  129.69981255,
        151.14772611,   77.99995317,   35.95433063,    5.37869591,
        -10.63302394,  -45.5927756 ,  -35.9843807 ]),
 'C_WW': array([ 1.22167333e+06,  5.72201115e+04, -1.96561395e+05,
2.89978888e+04,
       -6.66717091e+04,  4.03528209e+04,  1.67922964e+04,  1.31354675e+04,
        1.33766490e+04,  1.27871801e+04,  1.18282178e+04,  1.11391890e+04,
        1.02714250e+04,  9.74736234e+03,  9.11330576e+03,  8.34426584e+03,
        7.77788020e+03,  6.88741093e+03,  6.02057983e+03,  5.30407007e+03,
        4.16146732e+03,  3.27510474e+03,  2.20928046e+03,  8.46199566e+02,
        3.80993764e+02,  6.29640320e+02,  1.51092116e+02,  1.27424277e+02,
        4.96350798e+02,  1.96049519e+02,  2.69620644e+02, -1.13947283e+02,
        7.11160669e+01, -2.34567224e+02, -1.67677403e+02]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
       3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
       2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
       9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
       4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
       2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
       9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
       4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
       2.126676e-08, 3.121536e-08, 4.581796e-08])}
{'C_UU': array([ 3.60278357e+03,  1.29326655e+03,  9.32090673e+02,
7.61536234e+02,
        5.11363392e+02,  3.11565735e+02,  2.21567900e+02,  1.86767971e+02,
        1.76651041e+02,  1.75473230e+02,  1.59267139e+02,  1.56647024e+02,
        1.50098103e+02,  1.41780456e+02,  1.37737293e+02,  1.31995265e+02,
        1.21436383e+02,  1.14285369e+02,  1.03910329e+02,  9.41287712e+01,
```

```
         8.45056220e+01,  7.18796382e+01,  5.70910753e+01,  4.38745099e+01,
         3.43853916e+01,  3.13562993e+01,  1.58309018e+01,  6.97086986e+00,
        -5.89476341e+00, -5.44050979e+00, -5.63178134e+00,  9.41334503e+00,
        -1.44804610e+01, -2.36337160e+00,  1.90212666e+00]),
 'C_WU': array([ 746.7153379 ,  655.80468636,  394.29649177, 1531.55826167,
        1453.23267576, 1154.49945527, 1156.27787917, 1053.69306688,
         992.80453955,  953.77216825,  983.89546128,  918.29506754,
         907.16252587,  889.43359832,  848.17857143,  810.00371128,
         761.79843409,  731.26448391,  657.03628836,  580.35917193,
         520.36357456,  414.74824716,  323.30213253,  194.12130464,
         155.15369976,  144.14848093,  125.54306851,   45.34968301,
         -80.1946851 , -175.11336543,  -88.51122178,   98.72511037,
         -57.33256101,    7.79065156,    5.32392349]),
 'C_WW': array([ 1.21913914e+06,  5.86716363e+04, -1.92060727e+05,
2.84292146e+04,
        -6.38353772e+04,  3.94966790e+04,  1.92917794e+04,  1.49999070e+04,
         1.51596861e+04,  1.42896315e+04,  1.34164410e+04,  1.27516153e+04,
         1.23279587e+04,  1.16759266e+04,  1.08996661e+04,  1.01024257e+04,
         9.61144035e+03,  8.60535693e+03,  7.58638742e+03,  6.31074409e+03,
         5.34798397e+03,  3.41396889e+03,  2.20959213e+03,  5.40842618e+02,
         6.80116139e+02,  9.86203722e+02,  1.13366720e+03,  2.66124020e+02,
        -7.82834428e+02, -1.29281569e+03, -6.34412160e+02,  8.02198110e+02,
        -3.48645623e+02,  2.36456298e+01,  2.88520824e+01]),
 't': array([0.000000e+00, 8.000000e-14, 1.600000e-13, 2.400000e-13,
        3.600000e-13, 5.600000e-13, 8.800000e-13, 1.360000e-12,
        2.040000e-12, 3.040000e-12, 4.520000e-12, 6.640000e-12,
        9.760000e-12, 1.440000e-11, 2.120000e-11, 3.116000e-11,
        4.576000e-11, 6.716000e-11, 9.860000e-11, 1.448000e-10,
        2.126000e-10, 3.120800e-10, 4.580800e-10, 6.724000e-10,
        9.870000e-10, 1.448800e-09, 2.126600e-09, 3.121440e-09,
        4.581720e-09, 6.725080e-09, 9.871080e-09, 1.448884e-08,
        2.126676e-08, 3.121536e-08, 4.581796e-08])}
CPU times: user 9.03 s, sys: 1.75 s, total: 10.8 s
Wall time: 10.8 s
```

[54]:

```python
t_log = block_data[0]['t']
mean_C_UU = np.mean([block['C_UU'] for block in block_data], axis=0)
mean_C_WU = np.mean([block['C_WU'] for block in block_data], axis=0)
mean_C_WW = np.mean([block['C_WW'] for block in block_data], axis=0)
```

[55]:

```python
%%time

# Standard error of mean
sem_C_UU = sem([block['C_UU'] for block in block_data], axis=0)
sem_C_WW = sem([block['C_WW'] for block in block_data], axis=0)
sem_C_WU = sem([block['C_WU'] for block in block_data], axis=0)
```

```
sem_C_R = sem([block['C_WU']/np.sqrt(np.abs(block['C_UU']*block['C_WW'])) for␣
 ↪block in block_data], axis=0)
sem_C_gamma = sem([block['C_WU']/block['C_UU'] for block in block_data], axis=0)

# Compute the 67% confidence intervals
confidence_level = 0.67
degrees_freedom = len(block_data) - 1
t_multiplier = t.ppf((1 + confidence_level) / 2, degrees_freedom)
print(f'{t_multiplier=}')

# Compute the confidence intervals
C_UU_err = sem_C_UU * t_multiplier
C_WW_err = sem_C_WW * t_multiplier
C_WU_err = sem_C_WU * t_multiplier
C_R_err = sem_C_R * t_multiplier
C_gamma_err = sem_C_gamma * t_multiplier
```

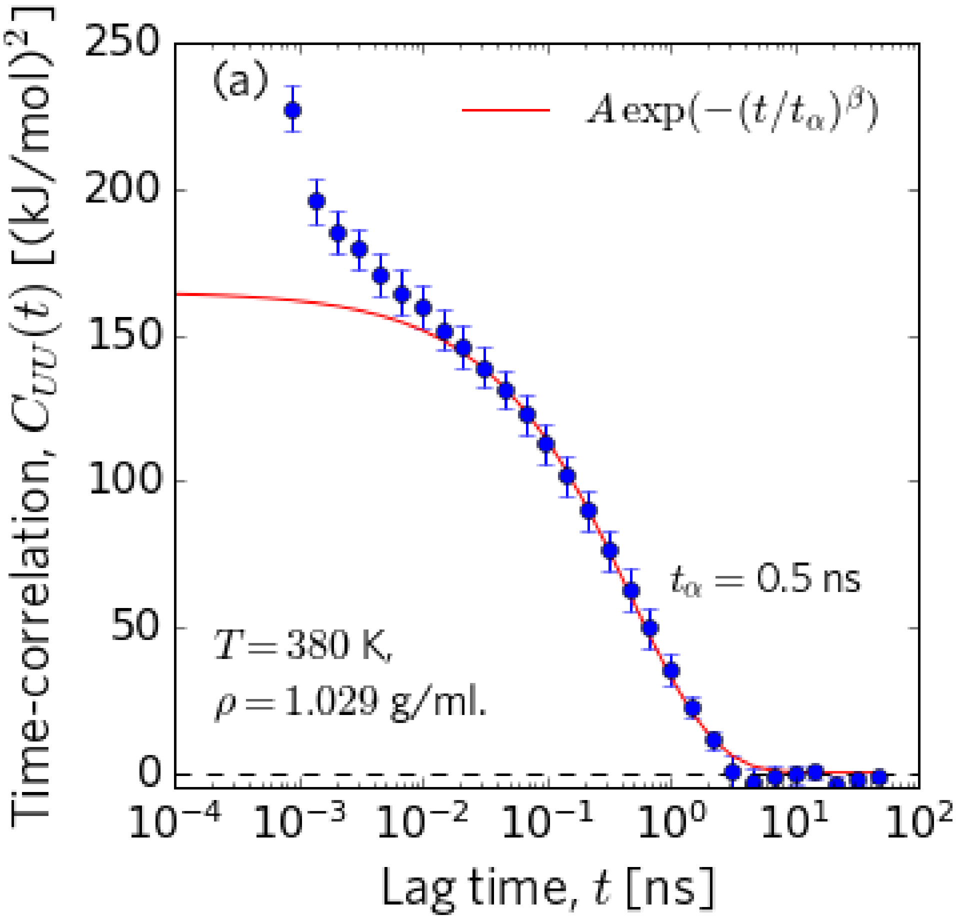
```
t_multiplier=1.046766724664122
CPU times: user 4.68 ms, sys: 1.66 ms, total: 6.34 ms
Wall time: 5.44 ms
```

[94]:

```
# Plot energy-energy correlation function

# Fit stretch exponential to the long-time tail of C_UU (below max_y_value)
max_y_value = 150
def stretched_exponential(t, A, beta, t_alpha):
    return A*np.exp(-(t/t_alpha)**beta)
# Find long tail data
long_tail = mean_C_UU < max_y_value
pguess = [150, 0.8, 0.8]
popt, pcov = scipy.optimize.curve_fit(stretched_exponential,␣
 ↪t_log[long_tail]*1e9, mean_C_UU[long_tail], p0=pguess)
print(popt)
A, beta, t_alpha = popt


# Plot energy-energy correlation function
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.errorbar(t_log*1e9, mean_C_UU, C_UU_err, fmt='bo')
```

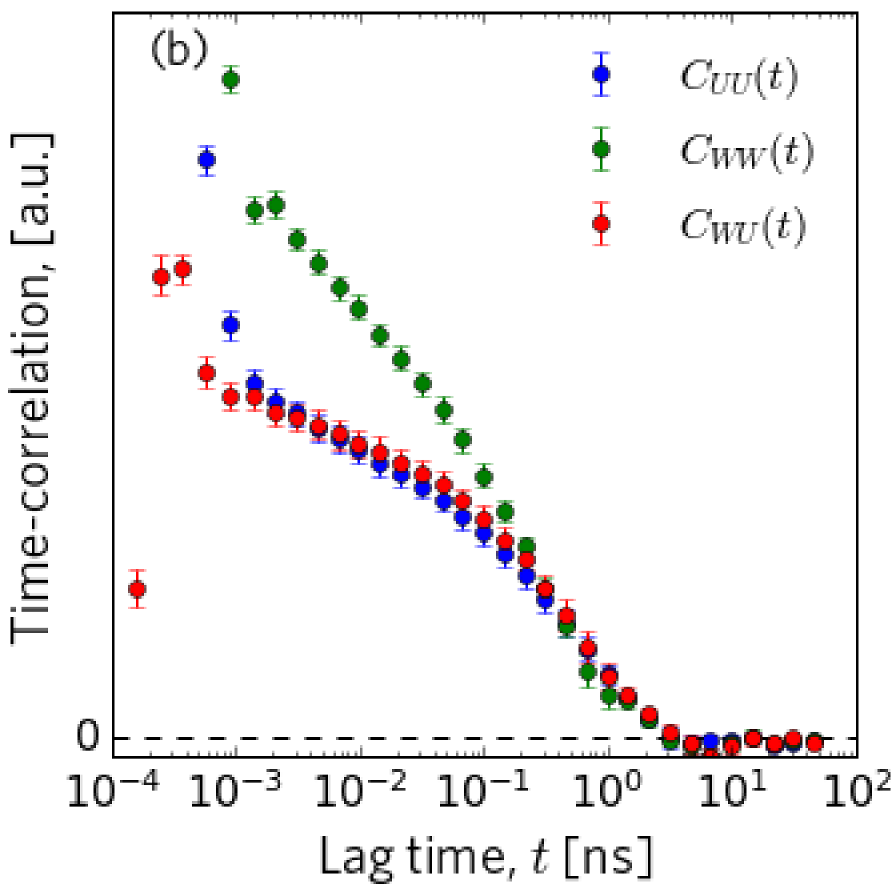
```
#plt.plot(t_log*1e9, stretched_exponential(t_log*1e9, *pguess), 'r--',␣
↪label='guess')
plt.plot(t_log*1e9, stretched_exponential(t_log*1e9, *popt), 'r-',␣
↪label=r'$A\,\exp(-(t/t_\alpha)^\beta)$')
plt.xscale('log')
plt.text(t_alpha*2, A/2.5, r'$t_\alpha = $' f'{t_alpha:.1g} ns',␣
↪va='center',fontsize=16)
plt.plot([1e-4, 100], [0]*2, 'k--')
plt.ylim(-5, 250)
plt.xlim(1e-4, 100)
plt.ylabel(r'Time-correlation, $C_{UU}(t)$ [(kJ/mol)$^2$]',fontsize=20)
plt.xlabel(r'Lag time, $t$ [ns]',fontsize=20)
plt.text(0.05, 0.1,'$T = 380$ K,' + '\n' + r'$\rho=1.029$ g/ml.', transform=plt.
↪gca().transAxes, fontsize=16)
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes,fontsize=18)
plt.legend(frameon=False, loc='upper right', numpoints=1,fontsize=16)
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.savefig('CUU_time_correlation.pdf', bbox_inches='tight')
plt.savefig('CUU_time_correlation.png', bbox_inches='tight')
plt.show()
```

```
[164.61255043   0.64370115   0.47463356]
```



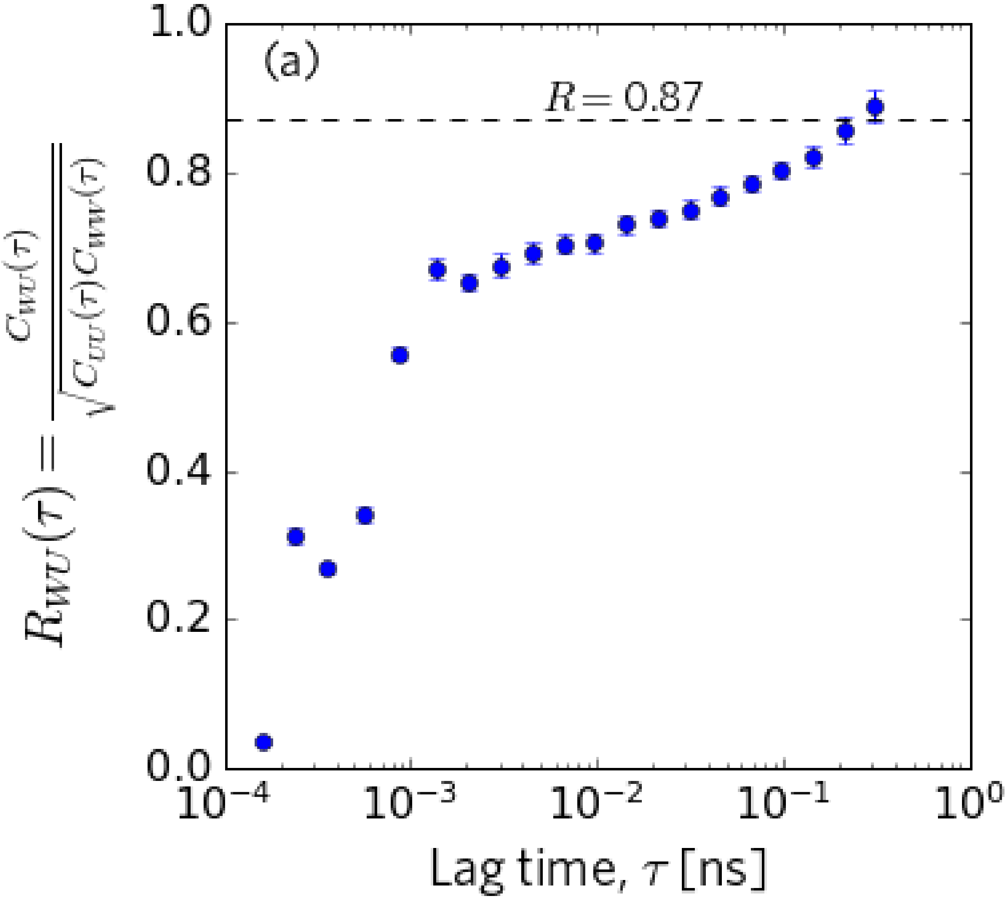
[60]:
```
plt.figure(figsize=(5, 5))
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.errorbar(t_log*1e9, mean_C_UU/100, C_UU_err/100, fmt='bo',
 ↪label=r'$C_{UU}(t)$')
plt.errorbar(t_log*1e9, mean_C_WW/5000, C_WW_err/5000, fmt='go',
 ↪label=r'$C_{WW}(t)$')
plt.errorbar(t_log*1e9, mean_C_WU/600, C_WU_err/600, fmt='ro',
 ↪label=r'$C_{WU}(t)$')
plt.plot([1e-4, 100], [0]*2, 'k--')
```

```
plt.xscale('log')
plt.ylim(-0.1, 4)
plt.xlim(1e-4, 100)
plt.yticks([0])
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes,fontsize=18)

plt.ylabel(r'Time-correlation, [a.u.]',fontsize=20)
plt.xlabel(r'Lag time, $t$ [ns]',fontsize=20)
plt.legend(frameon=False, loc='upper right', numpoints=1,fontsize=16)
plt.savefig('all_time_correlation.pdf', bbox_inches='tight')
plt.savefig('all_time_correlation.png', bbox_inches='tight')
plt.show()
```



[93]:
```
# Time-dependent correlation coefficient
last = 24

plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
```

```
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
mean_R = mean_C_WU/np.sqrt(np.abs(mean_C_UU*mean_C_WW))
plt.errorbar(t_log[:last-2]*1e9, mean_R[:last-2], C_R_err[:last-2], fmt='bo',␣
↪label=r'$R_{WU}(\tau)$')
plt.plot([1e-4, 100], [R_long_time]*2, 'k--')
plt.text(5e-3, 0.9, r'$R=$' f'{R_long_time}', va='center',fontsize=16)
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes,fontsize=18)

plt.xscale('log')
plt.ylim(0, 1.0)
plt.xlim(1e-4, 1)
plt.
↪ylabel(r'$R_{WU}(\tau)=\frac{C_{WU}(\tau)}{\sqrt{C_{UU}(\tau)C_{WW}(\tau)}}$',fontsize=20)
plt.xlabel(r'Lag time, $\tau$ [ns]',fontsize=20)
plt.savefig('R_time_correlation.pdf', bbox_inches='tight')
plt.savefig('R_time_correlation.png', bbox_inches='tight')

plt.show()
```

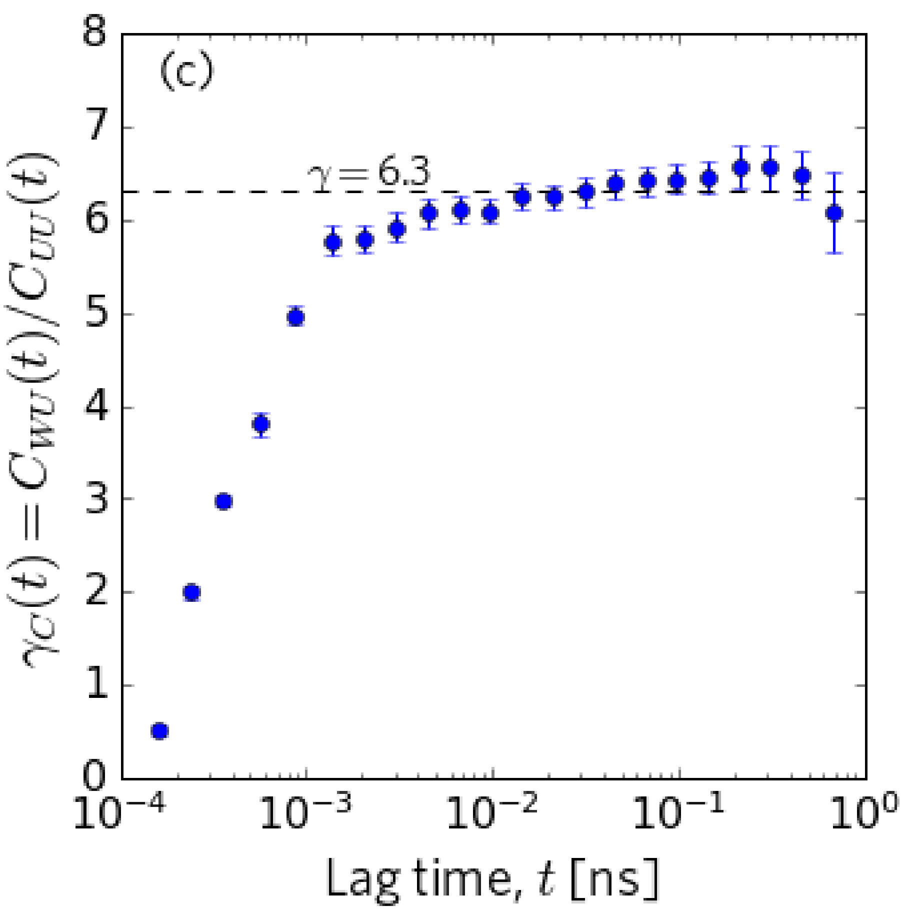


[73]:
```
t_log*1e9
```

[73]:
```
array([0.000000e+00, 8.000000e-05, 1.600000e-04, 2.400000e-04,
       3.600000e-04, 5.600000e-04, 8.800000e-04, 1.360000e-03,
       2.040000e-03, 3.040000e-03, 4.520000e-03, 6.640000e-03,
       9.760000e-03, 1.440000e-02, 2.120000e-02, 3.116000e-02,
       4.576000e-02, 6.716000e-02, 9.860000e-02, 1.448000e-01,
       2.126000e-01, 3.120800e-01, 4.580800e-01, 6.724000e-01,
       9.870000e-01, 1.448800e+00, 2.126600e+00, 3.121440e+00,
       4.581720e+00, 6.725080e+00, 9.871080e+00, 1.448884e+01,
       2.126676e+01, 3.121536e+01, 4.581796e+01])
```

[74]:
```
# Time-dependent scaling exponent
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to
 ↪install font first)
```

```
    'axes.unicode_minus': False
})
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.errorbar(t_log[:last]*1e9, mean_C_WU[:last]/mean_C_UU[:last], C_gamma_err[:
↪last], fmt='bo', label=r'$\gamma_{WU}(\tau)$')
plt.plot([1e-4, 100], [gamma_long_time]*2, 'k--')
plt.text(1e-3, gamma_long_time+0.2, r'$\gamma = $' f'{gamma_long_time}',␣
↪va='center',fontsize=16)
plt.xscale('log')
plt.text(0.05, 0.93, '(c)', transform=plt.gca().transAxes,fontsize=18)
plt.ylim(0, 8)
plt.xlim(1e-4, 1)
plt.ylabel(r'$\gamma_{C}(t)=C_{WU}(t)/C_{UU}(t)$',fontsize=20)
plt.xlabel(r'Lag time, $t$ [ns]',fontsize=20)
plt.savefig('gamma_WU_time_correlation.pdf', bbox_inches='tight')
plt.savefig('gamma_WU_time_correlation.png', bbox_inches='tight')
plt.show()
```



1.2.8 Figure 8(a) and 8(b)

[75]:
```
%%time
def frequency_responce_one_block(blocks=8, block_id=0, points_per_decade=6):
    if not block_id < blocks:
        raise ValueError(f'Reduce {block_id=} to be less than {blocks=}')
    U = np.array(df.PotEng)
    W = np.array(df.c_virial)
    block_len = int(len(U)/blocks)
    first = block_id*block_len
    last = (block_id+1)*block_len
    U = U[first:last]
    W = W[first:last]

    dt = float(df.reset_index().Time[1] - df.reset_index().Time[0])*1e9  # Time␣
 ↪step in ns

    omega, mu_UU = analyse.frequency_dependent_response(U, U, dt=dt)
    _, mu_WW = analyse.frequency_dependent_response(W, W, dt=dt)
    _, mu_UW = analyse.frequency_dependent_response(U, W, dt=dt)

    omega_log = analyse.run_avg_log(omega, points_per_decade)
    mu_UU_log = analyse.run_avg_log(mu_UU, points_per_decade)
    mu_WW_log = analyse.run_avg_log(mu_WW, points_per_decade)
    mu_UW_log = analyse.run_avg_log(mu_UW, points_per_decade)

    return {
        'omega': omega_log,
        'mu_UU': mu_UU_log,
        'mu_WW': mu_WW_log,
        'mu_UW': mu_UW_log
    }

frequency_responce_one_block()
```

```
CPU times: user 1.65 s, sys: 1.39 s, total: 3.04 s
Wall time: 4.52 s
```

[75]:
```
{'omega': array([0.00000000e+00, 1.07305092e-01, 2.14610185e-01, 3.21915277e-01,
        4.82872916e-01, 7.51135647e-01, 1.18035602e+00, 1.82418657e+00,
        2.73627986e+00, 4.07759351e+00, 6.06273773e+00, 8.90632268e+00,
        1.30912213e+01, 1.93149166e+01, 2.84358495e+01, 4.17953335e+01,
        6.13785129e+01, 9.00826251e+01, 1.32253526e+02, 1.94222217e+02,
        2.85163283e+02, 4.18597166e+02, 6.14428960e+02, 9.01899302e+02,
```

```
       1.32387658e+03, 1.94329522e+03, 2.85243762e+03, 4.18683010e+03,
       6.14552360e+03, 9.02044164e+03, 1.32402144e+04, 1.94340790e+04,
       2.85253956e+04, 4.18695886e+04, 6.14562554e+04]),
'mu_UU': array([-4745.42080987-1.73372428e-12j, -4747.31800705+8.50399736e-01j,
       -4751.27301197+2.90066388e+00j, -4762.36426911+1.26588242e+01j,
       -4739.76975781+3.08819194e+01j, -4758.17583811+2.90479829e+01j,
       -4737.54046722+5.58828975e+01j, -4694.19131995+7.15856902e+01j,
       -4681.94082392+1.00096631e+02j, -4635.43088667+5.35924880e+01j,
       -4640.06093919+7.09702982e+01j, -4602.57176497+3.37929134e+01j,
       -4618.2625336 +2.91978928e+01j, -4613.81692492+3.93015322e+01j,
       -4606.28322773+3.86925889e+01j, -4597.48023384+4.37180044e+01j,
       -4587.03823661+4.14936539e+01j, -4582.3773681 +4.43878937e+01j,
       -4576.77208899+5.28449936e+01j, -4571.07496167+6.60011839e+01j,
       -4563.49568959+8.07515388e+01j, -4552.93301555+1.16233146e+02j,
       -4541.7617537 +1.48849336e+02j, -4529.60973163+2.04267685e+02j,
       -4502.57052137+2.92990175e+02j, -4465.17048826+4.03845661e+02j,
       -4396.02140059+5.57496125e+02j, -4234.21859364+7.58435979e+02j,
       -4013.57390581+9.52837857e+02j, -3683.18078688+1.11848481e+03j,
       -3312.93221736+1.18586015e+03j, -2940.07130287+1.16598987e+03j,
       -2608.00972568+1.20168694e+03j, -2030.67682594+1.20340603e+03j,
       -1358.03139363+6.56324316e+02j]),
'mu_WW': array([-1799153.85528915-8.44011083e-10j,
       -1799335.31053113+3.10755715e+01j,
       -1799402.69996196+6.39708379e+01j,
       -1800444.61218415+4.27401130e+02j,
       -1798601.97919416+2.38268813e+03j,
       -1798611.87960659+7.79138585e+02j,
       -1800203.04123068+1.39514020e+03j,
       -1797353.05322532+3.66311345e+03j,
       -1798139.55497974+6.18249319e+03j,
       -1794702.94308266+3.56875408e+03j,
       -1794829.22333601+4.70793337e+03j,
       -1790885.6115332 +5.49026687e+03j,
       -1790490.78805247+3.35928031e+03j,
       -1790831.17786849+3.43212062e+03j,
       -1789540.37934499+3.14459236e+03j,
       -1789459.5747161 +2.94509081e+03j,
       -1788768.75161934+3.79273173e+03j,
       -1787853.74728924+3.91635648e+03j,
       -1787562.52091662+4.27113144e+03j,
       -1786855.98515962+5.52107715e+03j,
       -1786320.21743354+6.72120668e+03j,
       -1785513.65520943+8.20347804e+03j,
       -1785079.19016114+1.11396383e+04j,
       -1784909.01710593+1.49069601e+04j,
       -1783884.38748598+2.11714552e+04j,
       -1784797.10031083+3.04435664e+04j,
```

```
        -1784926.80625565+4.12291254e+04j,
        -1790324.24216383+6.09478060e+04j,
        -1801716.79293739+9.33340174e+04j,
        -1823395.74114057+1.53339604e+05j,
        -1852981.0994539 +2.93445068e+05j,
        -1711306.58149778+5.70363239e+05j,
        -1426374.61347851+5.95488962e+05j,
        -1129611.80660606+7.56893908e+05j,
         -578410.41731858+3.69075265e+05j]),
 'mu_UW': array([ -909.44673165-1.36424205e-12j,  -916.17368313+7.20375819e+00j,
         -935.78035612-1.63015537e+00j, -1064.03742285+6.31962623e+01j,
         -839.5595663 +2.52299337e+02j,  -884.8392763 +1.32124529e+02j,
         -965.4699426 +2.30396418e+02j,  -654.17938248+3.70852624e+02j,
         -681.30821448+7.36139564e+02j,  -239.35950599+3.97514651e+02j,
         -337.35010693+3.68176980e+02j,    13.58256478+3.92317368e+02j,
           -9.04902819+1.60128336e+02j,   -36.70101348+2.13781878e+02j,
          -59.70526339+1.53399987e+02j,    34.84891154+1.93517117e+02j,
           83.26141471+1.57607332e+02j,    99.91953005+1.12019858e+02j,
          117.59211715+1.12470051e+02j,   106.72863475+1.33893897e+02j,
          190.43272122+1.21842109e+02j,   258.4268608 +1.30435904e+02j,
          248.72129851+8.42643847e+01j,   261.08916628+8.51391991e+00j,
          275.95997694+4.73959000e+01j,   338.64723493+9.95290174e+00j,
          240.79927678-4.72708719e+01j,   391.64148895+1.10366614e+02j,
          481.53791997-2.26851317e+02j,   396.10302629-4.36322589e+02j,
          468.9263235 -8.66930887e+02j,  -864.97652716-1.17525060e+03j,
        -1082.40282447+2.17752194e+02j,  -410.75563407+6.93286872e+02j,
          137.27234591+8.11059505e+01j])}
```

[76]:
```python
%%time
mu_blocks = []
for block_id in range(blocks):
    tmp = frequency_responce_one_block(blocks=blocks, block_id=block_id)
    # pprint(tmp)
    mu_blocks.append(tmp)
```

```
CPU times: user 12.8 s, sys: 5.44 s, total: 18.3 s
Wall time: 19.3 s
```

[77]:
```python
mu_blocks[2]['mu_WW'][0]
```

[77]: (-1799132.103812022-1.0040821507573128e-09j)

[78]:
```python
# Remove a zero, to avoid 1/0
mu_blocks[3]['mu_WW'][0] = mu_blocks[2]['mu_WW'][0]
```

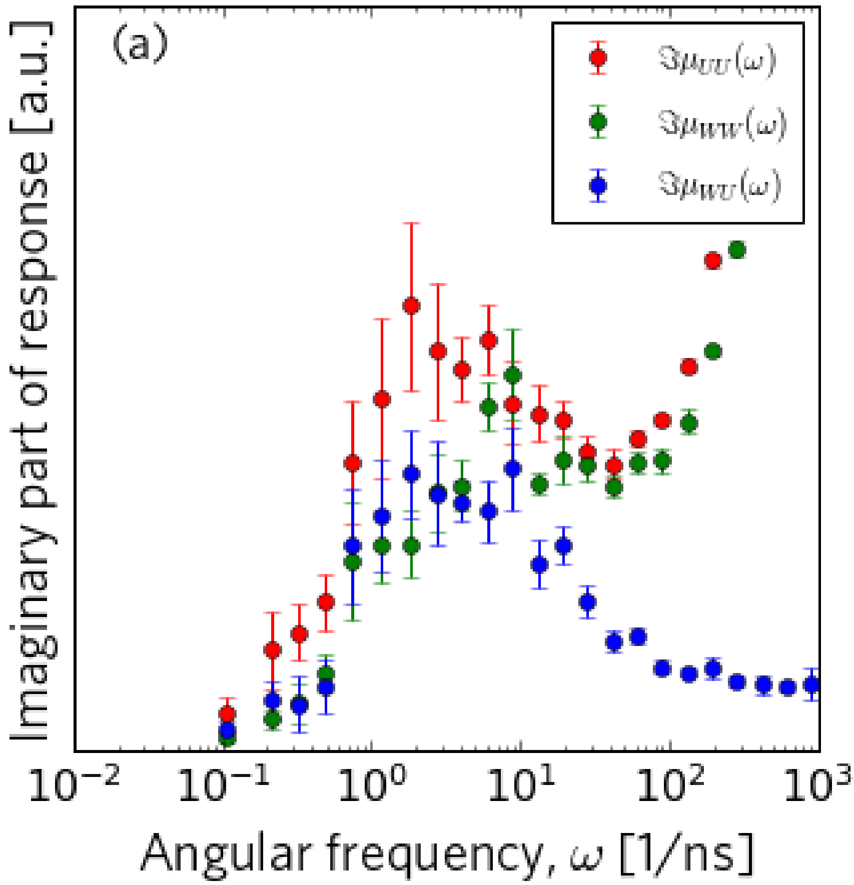
[79]:
```python
# Standard error of mean
sem_mu_UU_imag = sem([np.imag(block['mu_UU']) for block in mu_blocks], axis=0)
```

```
sem_mu_WW_imag = sem([np.imag(block['mu_WW']) for block in mu_blocks], axis=0)
sem_mu_WU_imag = sem([np.imag(block['mu_UW']) for block in mu_blocks], axis=0)
tmp = [np.imag(block['mu_UW'])/np.sqrt(np.abs(np.imag(block['mu_UU'])*np.
 ↪imag(block['mu_WW']))) for block in mu_blocks]
sem_mu_R = sem(tmp, axis=0)
sem_mu_gamma = sem([np.imag(block['mu_UW'])/np.imag(block['mu_UU']) for block␣
 ↪in mu_blocks], axis=0)

# Compute the confidence intervals
mu_UU_imag_err = sem_mu_UU_imag * t_multiplier
mu_WW_imag_err = sem_mu_WW_imag * t_multiplier
mu_WU_imag_err = sem_mu_WU_imag * t_multiplier
mu_R_err = sem_mu_R * t_multiplier
mu_gamma_err = sem_mu_gamma * t_multiplier
```

[80]:
```
omega_log = mu_blocks[0]['omega']
mu_UU_log = np.mean([block['mu_UU'] for block in mu_blocks], axis=0)
mu_WW_log = np.mean([block['mu_WW'] for block in mu_blocks], axis=0)
mu_UW_log = np.mean([block['mu_UW'] for block in mu_blocks], axis=0)
```

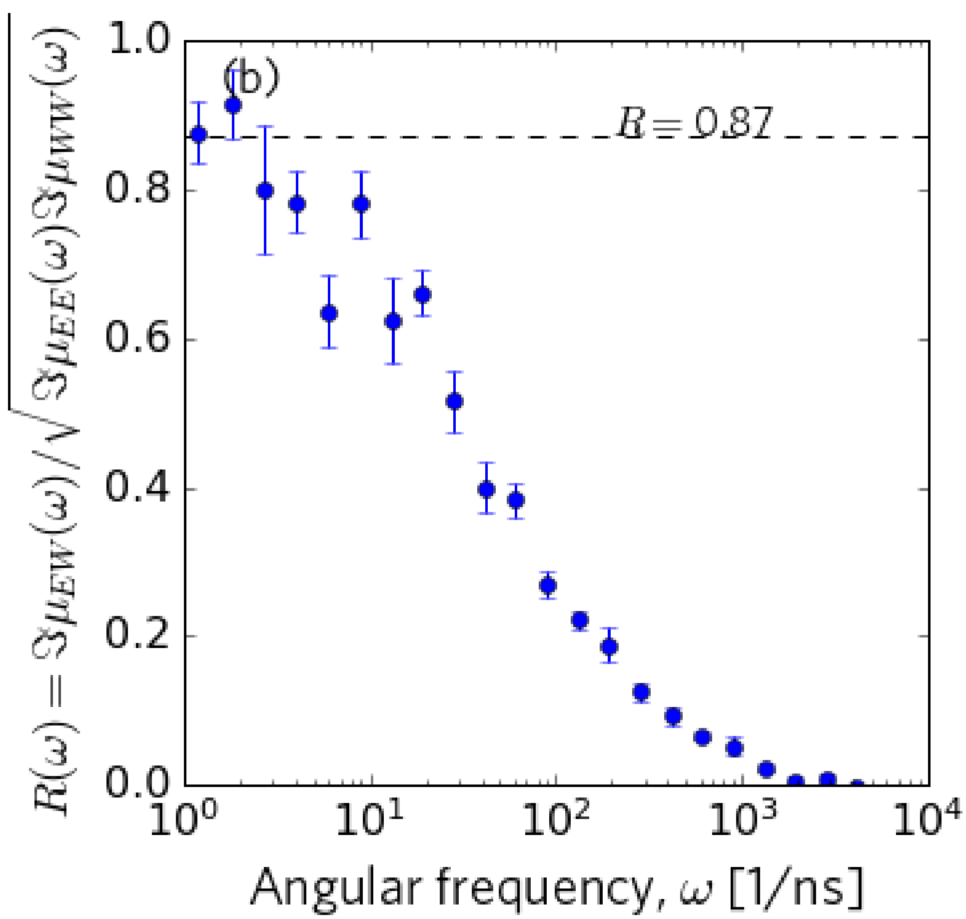
[81]:
```
# Plot frequency-dependent response
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.errorbar(omega_log, np.imag(mu_UU_log)/100, mu_UU_imag_err/100, fmt='ro',␣
 ↪label=r'$\Im\mu_{UU}(\omega)$')
plt.errorbar(omega_log, np.imag(mu_WW_log)/10000, mu_WW_imag_err/10000,␣
 ↪fmt='go', label=r'$\Im\mu_{WW}(\omega)$')
plt.errorbar(omega_log, np.imag(mu_UW_log)/1000, mu_WU_imag_err/1000, fmt='bo',␣
 ↪label=r'$\Im\mu_{WU}(\omega)$')
plt.plot([0, 1e13], [0, 0], 'k--')
plt.xlim(1e-2,1e3)
plt.ylim(0, 1)
plt.yticks([])
plt.text(0.05, 0.93, '(a)', transform=plt.gca().transAxes,fontsize=18)
plt.xscale('log')
plt.xlabel(r'Angular frequency, $\omega$ [1/ns]',fontsize=20)
plt.ylabel(r'Imaginary part of response [a.u.]',fontsize=20)
```

```
plt.legend(numpoints=1)
plt.savefig('all_response.pdf', bbox_inches='tight')
plt.savefig('all_response.png', bbox_inches='tight')
plt.show()
```



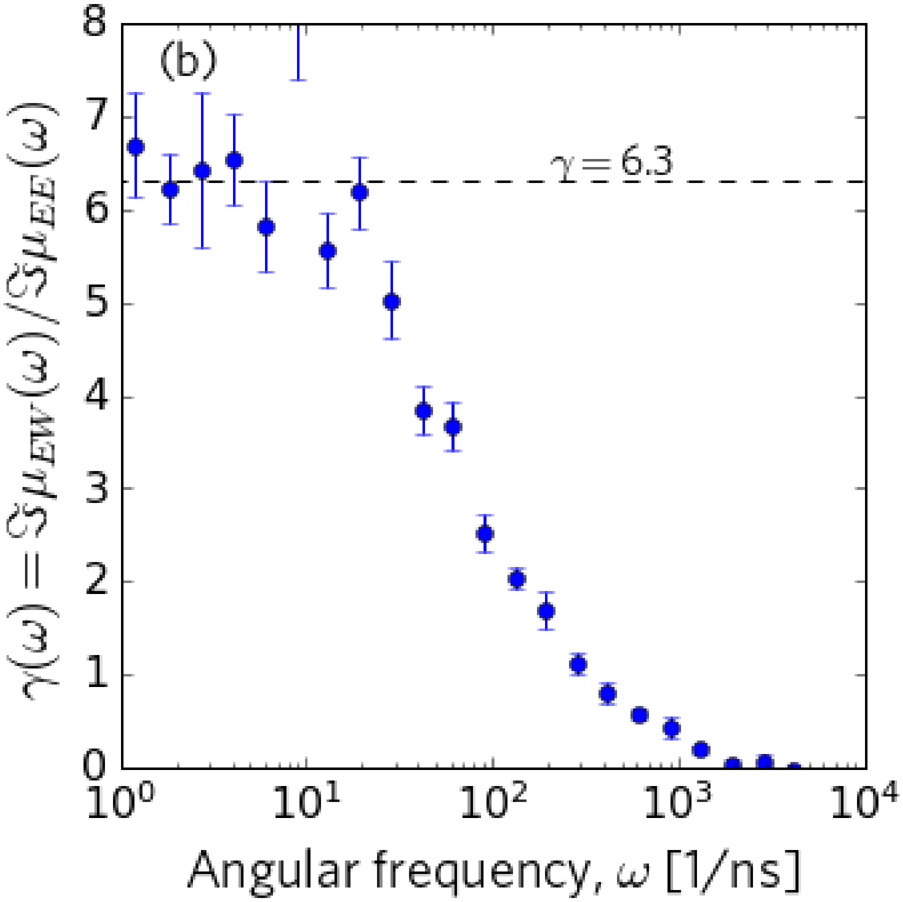
[89]:
```
skip = 5

# Plot frequency-dependent correlation coefficient
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',    # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
 ↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.xticks(fontsize=16)
```

```
plt.yticks(fontsize=16)
plt.errorbar(omega_log[skip:], np.imag(mu_UW_log[skip:])/np.sqrt(np.
 ↪imag(mu_UU_log[skip:])*np.imag(mu_WW_log[skip:])), mu_R_err[skip:],␣
 ↪fmt='bo', label=r'$R(\omega)$')
R_long_time = 0.87
plt.plot([1e-2, 1e4], [R_long_time]*2, 'k--')
plt.text(2e2, R_long_time+0.02, r'$R=$' f'{R_long_time}',␣
 ↪va='center',fontsize=16)
plt.xlim(1e0,1e4)
plt.ylim(0, 1)
plt.xscale('log')
plt.xlabel(r'Angular frequency, $\omega$ [1/ns]',fontsize=20)
plt.ylabel(r'$R(\omega)=\Im\mu_{EW}(\omega)/
 ↪\sqrt{\Im\mu_{EE}(\omega)\Im\mu_{WW}(\omega)}$',fontsize=18)
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes,fontsize=18)
plt.savefig('R_response.pdf', bbox_inches='tight')
plt.savefig('R_response.png', bbox_inches='tight')
plt.show()
```



[86]:

```
# Plot the frequency dependent scaling exponent
plt.figure(figsize=(5, 5))
plt.rcParams.update({
    'font.family': 'DejaVu Sans',   # Default font
    #'font.family': 'Whitney Book', # Actual font used in the paper (needs to␣
↪install font first)
    'axes.unicode_minus': False
})
plt.xticks(fontname="DejaVu Sans")
plt.yticks(fontname="DejaVu Sans")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.errorbar(omega_log[skip:], np.imag(mu_UW_log[skip:])/np.imag(mu_UU_log[skip:
↪]), mu_gamma_err[skip:], fmt='bo', label=r'$\gamma(\omega)$')
plt.plot([1e-2, 1e4], [gamma_long_time]*2, 'k--')
plt.text(2e2, gamma_long_time+0.2, r'$\gamma=$' f'{gamma_long_time}',␣
↪va='center',fontsize=16)
plt.text(0.05, 0.93, '(b)', transform=plt.gca().transAxes,fontsize=18)
plt.xlim(1e0,1e4)
plt.ylim(0, 8)
plt.xscale('log')
plt.xlabel(r'Angular frequency, $\omega$ [1/ns]',fontsize=20)
plt.ylabel(r'$\gamma(\omega)=\Im\mu_{EW}(\omega)/
↪\Im\mu_{EE}(\omega)$',fontsize=20)
plt.savefig('gamma_WU_response.pdf', bbox_inches='tight')
plt.savefig('gamma_WU_response.png', bbox_inches='tight')
plt.show()
```


(b)
$\gamma = 6.3$
$\gamma(\omega) = \Im\mu_{EW}(\omega) / \Im\mu_{EE}(\omega)$
Angular frequency, ω [1/ns]

```
[ ]:
```