Turbulence Analysis

July 26, 2022

1 Analyis of Turbulence Data

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from scipy import signal
from lmfit import Model
# https://lmfit.github.io/lmfit-py/builtin_models.html
# https://cars9.uchicago.edu/software/python/lmfit/builtin_models.html

import lmfit
label_size = 18
plt.rcParams.update(
    {
        "font.size": label_size,
        "legend.title_fontsize": label_size,
        "legend.fontsize": label_size,
        "axes.labelsize": label_size,
        "xtick.labelsize": label_size,
        "ytick.labelsize": label_size,
        "axes.labelpad": 4,
        # "lines.markersize": 13,
        "lines.linewidth": 2,
    }
)
```

2 Exercise 5

[ ]:
```
data = np.loadtxt('Data/k10mf_processed_FD2.txt')
```

5.1) Calculate mean $u_{mean} = \langle u \rangle$ and standard deviation $u_{std} = \sqrt{\langle u^2 \rangle - (\langle u \rangle)^2}$

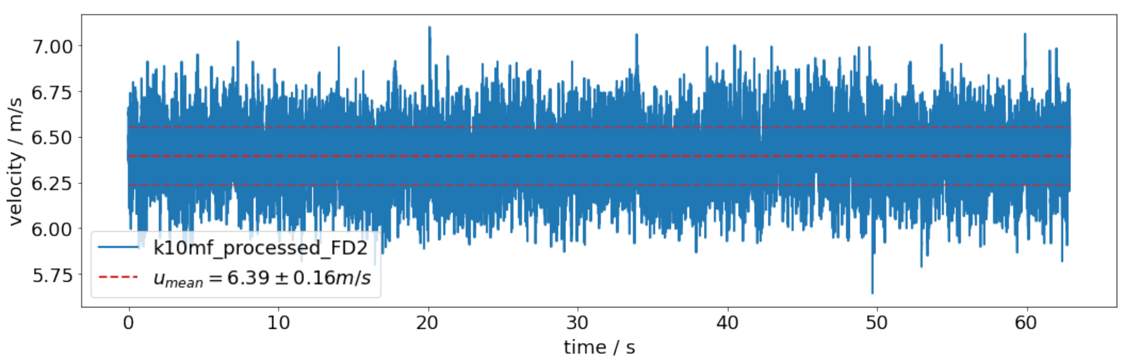
[ ]:
```
# Sample frequency 20 kHz
f_s = 20_000
T = len(data)/f_s
print(f'The dataset is is {T}s long.')
```

```python
# Mean and standard deviation
u_mean = np.mean(data)
u_std = np.std(data)
```

```
The dataset is is 62.88915s long.
```

[ ]:
```python
fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
time = np.linspace(0,T,len(data))
ax.plot(time, data, label='k10mf_processed_FD2', zorder=1)

# Plot time series and mean +/- std velocity
print(f'The mean velocity is {u_mean:.2f}m/s.\nThe standard deviation is +/- {u_std:.2f}m/s.')
ax.hlines(u_mean, 0, T, colors='C3', linestyles='--', zorder=2, label=f'$u_{{mean}}={{{u_mean:.2f}}} \pm {{{u_std:.2f}}} m/s$')
ax.hlines(u_mean+u_std, 0, T, colors='C3', linestyles='--', zorder=2, alpha=0.6)
ax.hlines(u_mean-u_std, 0, T, colors='C3', linestyles='--', zorder=2, alpha=0.6)

ax.set_xlabel('time / s')
ax.set_ylabel('velocity / m/s')
ax.legend()
fig.savefig('Abb/Ex5_Velocity_Time_Series.png')
```

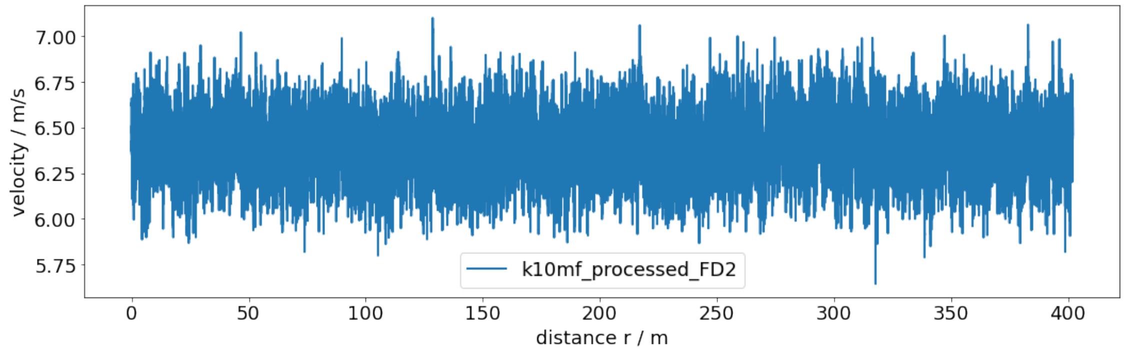
```
The mean velocity is 6.39m/s.
The standard deviation is +/- 0.16m/s.
```



2.0.1 5.2) Use Taylor's hypothesis $x = u_{mean} \cdot t$ to calculate a spatial series from a temporal series

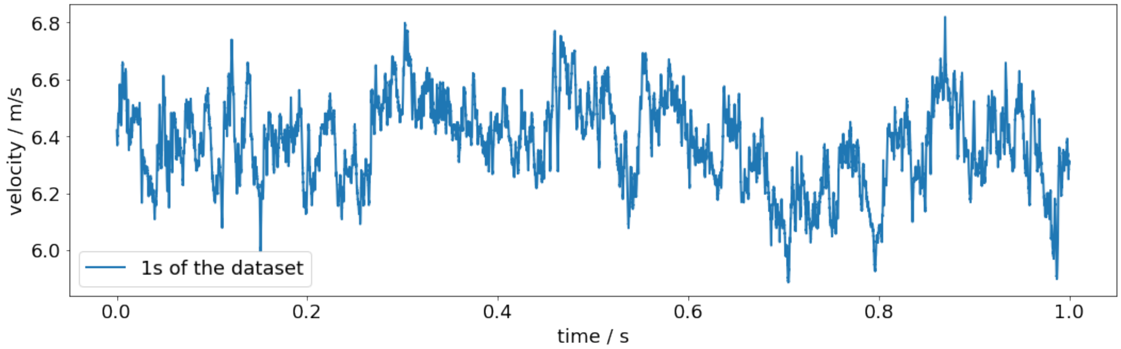
Essentially, you assume that the velocity field you record at one spatial position over time only gets advected but not deformed by the mean velocity.

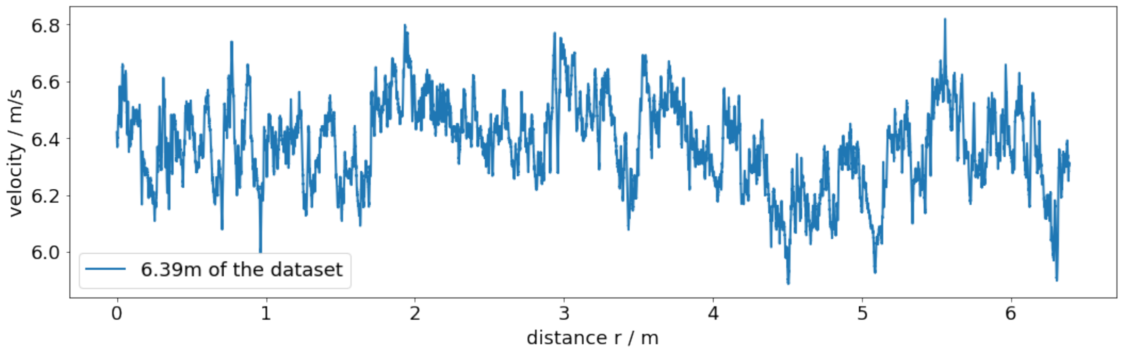
```
[ ]: fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
     spatial = time * u_mean
     ax.plot(spatial, data, label='k10mf_processed_FD2', zorder=1)
     ax.set_xlabel('distance r / m')
     ax.set_ylabel('velocity / m/s')
     ax.legend()
     fig.savefig('Abb/Ex5_Velocity_Spatial_Series.png')
```

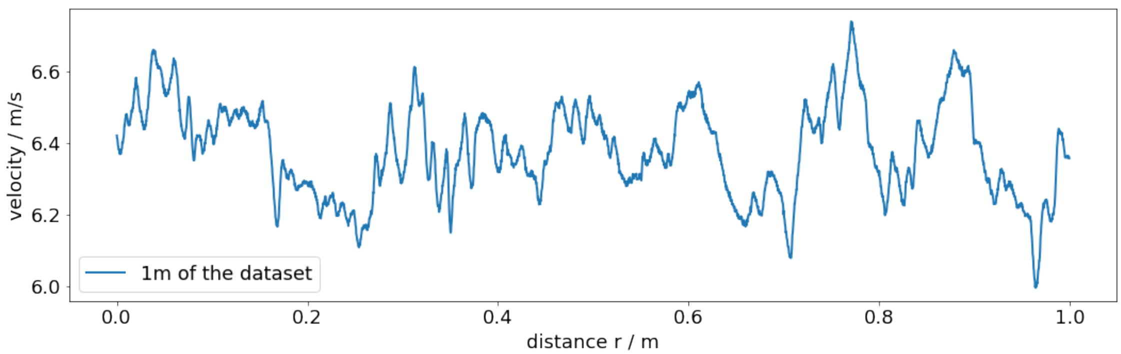



```
[ ]: fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
     ax.plot(time[:f_s], data[:f_s], label='1s of the dataset', zorder=1)
     ax.set_xlabel('time / s')
     ax.set_ylabel('velocity / m/s')
     ax.legend()

     fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
     ax.plot(spatial[:f_s], data[:f_s], label=f'{u_mean:.2f}m of the dataset',␣
      ↪zorder=1)
     ax.set_xlabel('distance r / m')
     ax.set_ylabel('velocity / m/s')
     ax.legend()

     fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
     ax.plot(spatial[:int(f_s/u_mean)], data[:int(f_s/u_mean)], label='1m of the␣
      ↪dataset', zorder=1)
     ax.set_xlabel('distance r / m')
     ax.set_ylabel('velocity / m/s')
     ax.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x1b512501790>
```



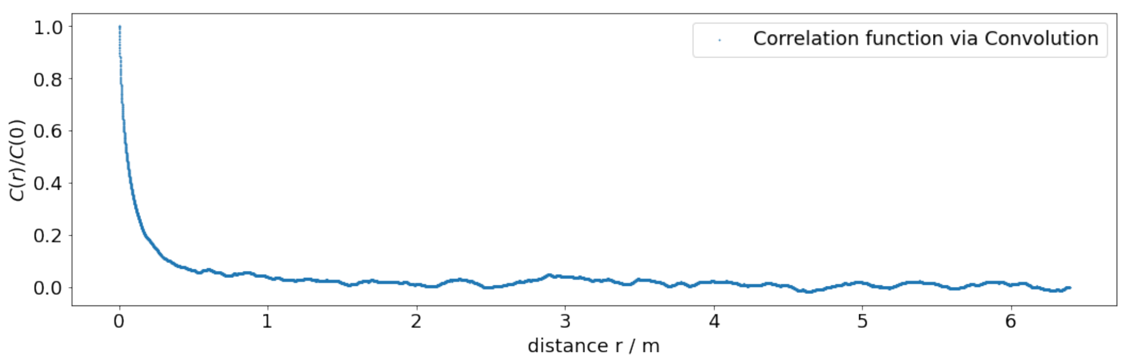




5.3) Calculate correlation function $C(r) = \langle u'(x+r)u'(x) \rangle$ of the fluctuations $u' = u - \langle u \rangle$

[ ]:
```
# Number of seconds in the dataset to do the computation
sec = 1
data_fluc = data - u_mean

fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)
# Not exactly the same one
```

```
# acf = sm.tsa.acf(data_fluc, nlags=f_s*sec, fft=True)
# ax.plot(spatial[:f_s*sec+1], acf, label='Autocorrelation function', zorder=1)


# Correlation via convolution: Much faster than doing it for each r by my own
# 2s for all r instead of 4 Minutes for only 20.000 r
cf_cov = signal.fftconvolve(data_fluc, data_fluc[::-1], mode='full')
# Devide by how many samples were involved in convolution
xi = np.arange(1, len(data_fluc) + 1)
d = np.hstack((xi, xi[:-1][::-1]))
cf_cov = cf_cov / d

# Only the first half is our correlation
cf_cov = cf_cov[:len(data_fluc)][::-1]
cf_cov_0 = cf_cov[0]
cf_cov = cf_cov/cf_cov[0]
ax.scatter(spatial[:f_s*sec+1], cf_cov[:f_s*sec+1], label='Correlation function␣
↪via Convolution', zorder=1, s=0.5)

ax.set_xlabel('distance r / m')
ax.set_ylabel('$C(r)/C(0)$')
ax.legend()
fig.savefig('Abb/Ex_5Correlation.png')
```

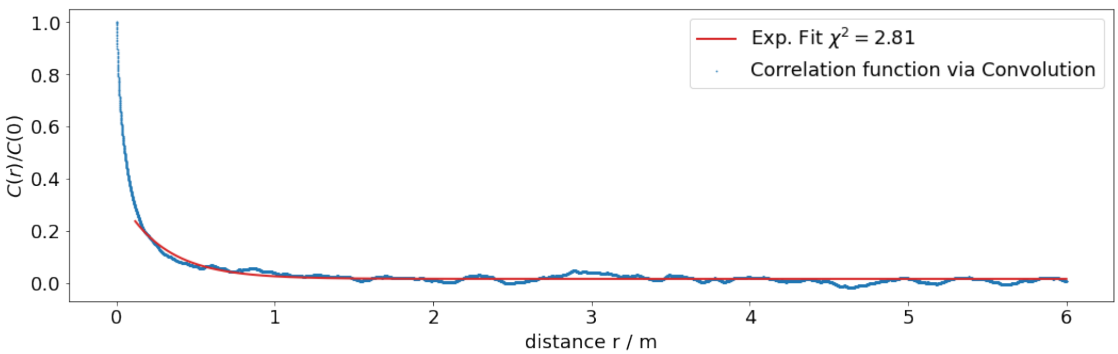


5.4) Determine integral length scale $L = \int_0^\infty dr \frac{C(r)}{C(0)}$ and by an exponential fit $C(r) \approx C(0) \cdot e^{-\frac{r}{L}}$

[ ]:
```
fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)

# 1) Integral length scale via Integral = Summation of rectangles
step = time[1]-time[0]

# Zero crossing zc
zc = np.argwhere(cf_cov < 0)[0,0]
L_sum = np.sum(cf_cov[:zc])*step
```

```python
print(f'Integral length via summation: L = {L_sum:.4f}m')

# 2) Integral length scale via exp-fit
# begin, end of fit in m
begin, end = 0.12, 6
idx_b, idx_e = int(begin/u_mean*f_s) , int(end/u_mean*f_s)

xdat = spatial[idx_b:idx_e]
ydat = cf_cov[idx_b:idx_e]

mod = lmfit.models.ExponentialModel() + lmfit.models.ConstantModel()
# pars = mod.make_params()
result = mod.fit(ydat, x=xdat)
L_fit = result.values['amplitude']
print(f'Integral length via fit: L = {L_fit:.4f}m')

# print(result.fit_report())
plt.plot(xdat, result.eval(result.params, x=xdat), c='C3', label=f'Exp. Fit␣
↪$\chi^2={{{result.chisqr:.2f}}}$')
ax.scatter(spatial[:idx_e], cf_cov[:idx_e], label='Correlation function via␣
↪Convolution', zorder=1, s=0.5)

ax.set_xlabel('distance r / m')
ax.set_ylabel('$C(r)/C(0)$')
ax.legend()
fig.savefig('Abb/Ex5_Correlation_Zoomed_Fit.png')
```

```
Integral length via summation: L = 0.0259m
Integral length via fit: L = 0.3376m
```



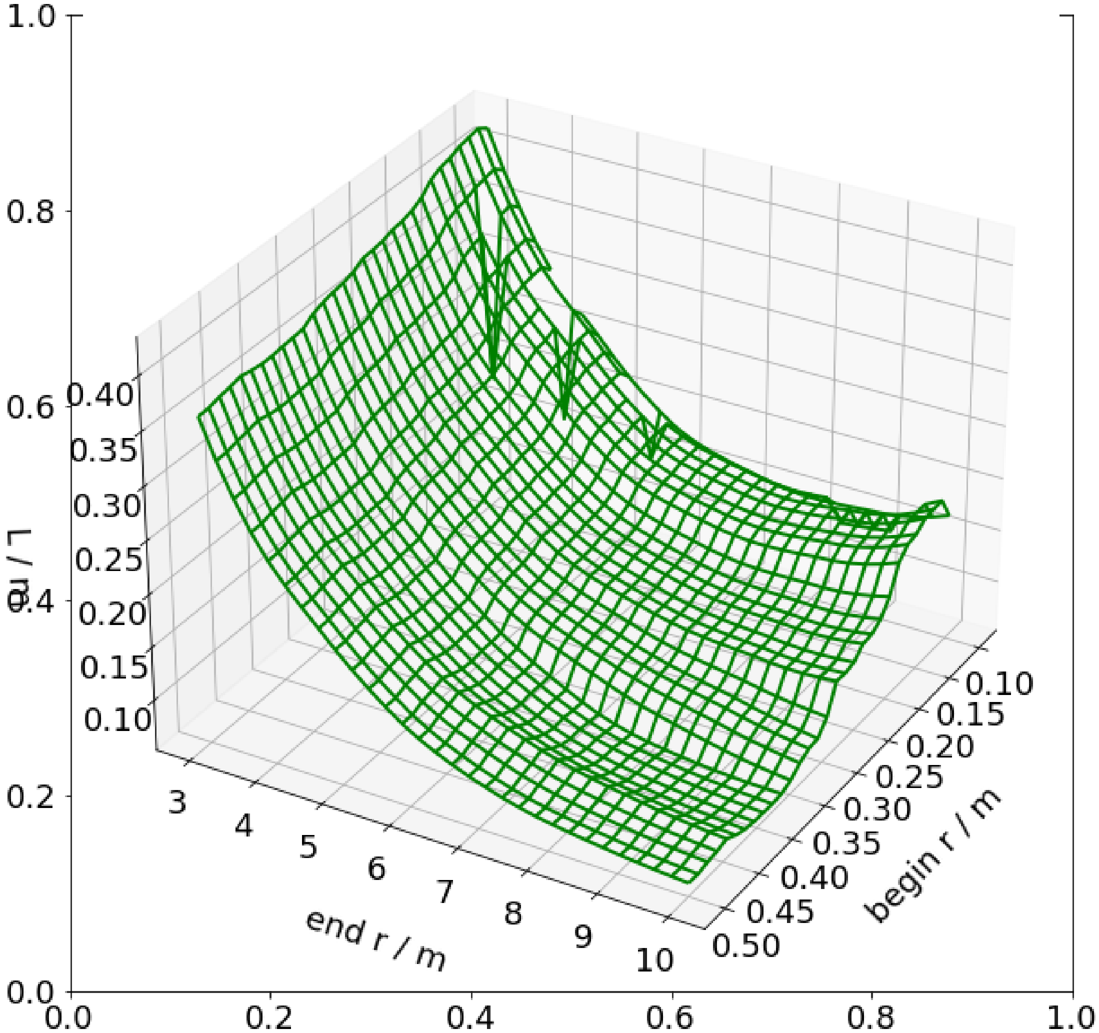
Try out different begin and end points for the fit

[ ]:
```python
size = 30
L_matrix = np.empty((size,size))
```

```
b_vec = np.linspace(0.1,0.5,size)
e_vec = np.linspace(3,10,size)
for i, begin in enumerate(b_vec):
    for j, end in enumerate(e_vec):
        idx_b, idx_e = int(begin/u_mean*f_s) , int(end/u_mean*f_s)
        xdat = spatial[idx_b:idx_e]
        ydat = cf_cov[idx_b:idx_e]

        mod = lmfit.models.ExponentialModel() + lmfit.models.ConstantModel()
        try:
            result = mod.fit(ydat, x=xdat)
            L = result.values['amplitude']
        except:
            L = None
        L_matrix[i,j] = L
```

The value of L really depends on the start and end point of our fit!

[ ]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax = plt.axes(projection ='3d')
X, Y = np.meshgrid(b_vec, e_vec)
# ax.plot3D(b_vec[:len(L_matrix)], e_vec[:len(L_matrix)], L_matrix)
ax.plot_wireframe(X, Y, L_matrix, color ='green')
ax.set_xlabel('begin r / m')
ax.set_ylabel('end r / m')
ax.set_zlabel('L / m')
ax.view_init(30, 30)
ax.yaxis.labelpad = 20
ax.xaxis.labelpad = 20
ax.zaxis.labelpad = 20
fig.savefig('Abb/Ex5_Different_integral_lengths_L.png')
```

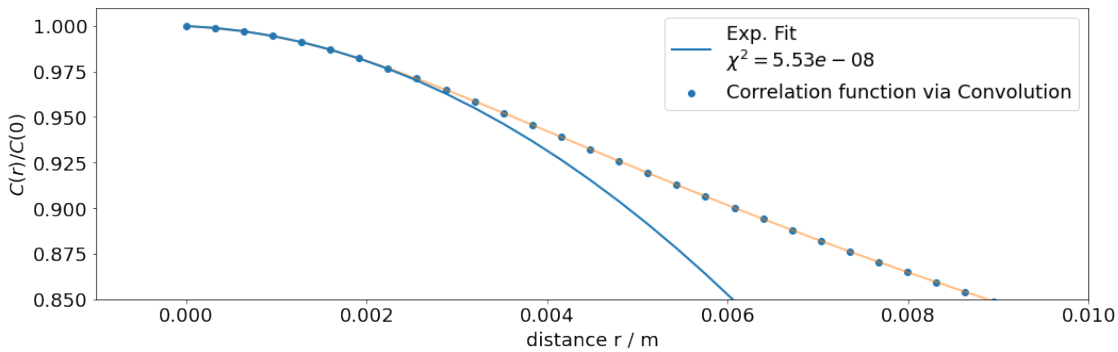


5.5) Determine Taylor length $\lambda = \sqrt{-\frac{C(0)}{C''(0)}}$

[ ]:
```
fig, ax = plt.subplots(figsize=(15,5), tight_layout=True)

# Taylor length scale via polynomial fit
# begin, end of fit in m
dpoints = 50
idx_b, idx_e = 0, 7

xdat = spatial[idx_b:idx_e]
ydat = cf_cov[idx_b:idx_e]

mod = lmfit.models.QuadraticModel()
result = mod.fit(ydat, x=xdat)
C_curvature = result.values['a']
```

```python
# print(f'Taylor length lambda via cf_cov[0]= 1 normed = {np.sqrt(-cf_cov[0]/
↪C_curvature):.4f}m')
print(f'Taylor length lambda via cf_cov[0] = {np.sqrt(-cf_cov_0/C_curvature):.
↪4f}m')



ax.plot(spatial[:dpoints], result.eval(result.params, x=spatial[:dpoints]),␣
↪label=f'Exp. Fit\n$\chi^2={{{result.chisqr:.2e}}}$')
ax.scatter(spatial[:dpoints], cf_cov[:dpoints], label='Correlation function via␣
↪Convolution', zorder=1)
ax.plot(spatial[:dpoints], cf_cov[:dpoints], alpha=0.5, zorder=1)

ax.set_xlim((-0.001,0.01))
ax.set_ylim((0.85,1.01))
ax.set_xlabel('distance r / m')
ax.set_ylabel('$C(r)/C(0)$')
ax.legend()
fig.savefig('Abb/Ex5_Correlation_Taylor_length.png')

print(result.fit_report())
```

```
Taylor length lambda via cf_cov[0] = 0.0026m
[[Model]]
    Model(parabolic)
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 13
    # data points      = 7
    # variables        = 3
    chi-square         = 5.5260e-08
    reduced chi-square = 1.3815e-08
    Akaike info crit   = -124.599882
    Bayesian info crit = -124.762151
[[Variables]]
    a: -3738.93693 +/- 125.543927 (3.36%) (init = 0)
    b: -2.12421281 +/- 0.25058120 (11.80%) (init = 0)
    c:  0.99990890 +/- 1.0259e-04 (0.01%) (init = 0)
[[Correlations]] (unreported correlations are < 0.100)
    C(a, b) = -0.961
    C(b, c) = -0.781
    C(a, c) = 0.625
```

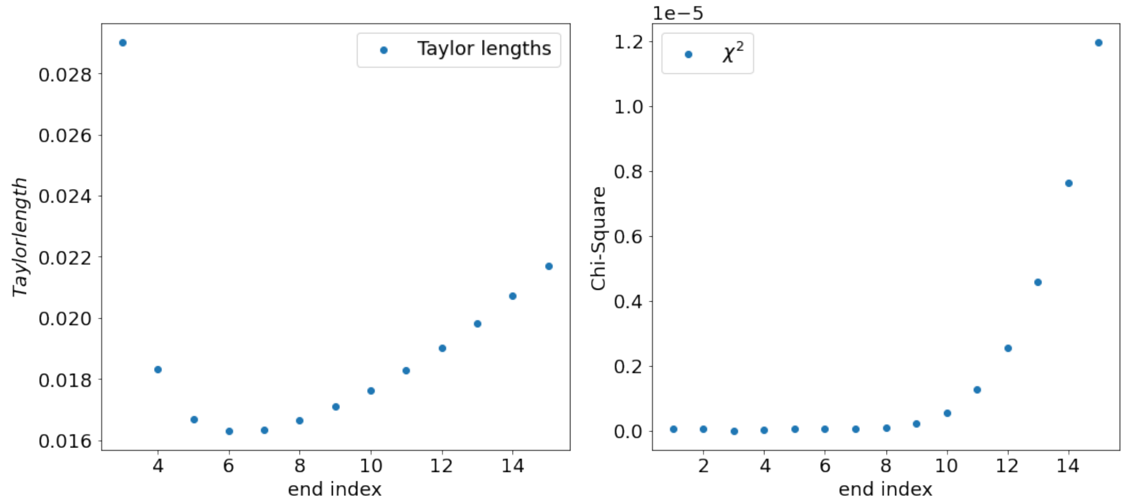


Try out different end point of the correlation function and look at Chi-Square for best fit

[ ]:
```
num_idx = 15
taylor_lengths = np.empty(num_idx)
chi_sqr = np.empty(num_idx)

indices = np.linspace(1,num_idx,num_idx)

for idx_e in indices:
    idx_e = int(idx_e)
    xdat = spatial[:idx_e]
    ydat = cf_cov[:idx_e]

    mod = lmfit.models.QuadraticModel()
    try:
        result = mod.fit(ydat, x=xdat)
        C_curvature = result.values['a']
        lambda_l = np.sqrt(-cf_cov[0]/C_curvature)
    except:
        lambda_l = None
    taylor_lengths[idx_e-1] = lambda_l
    chi_sqr[idx_e-1] = result.chisqr


fig, ax = plt.subplots(1,2,figsize=(15,7), tight_layout=True)

ax[0].scatter(indices, taylor_lengths, label='Taylor lengths', zorder=1)
ax[0].set_xlabel('end index')
ax[0].set_ylabel('$Taylor length$')

ax[1].scatter(indices, chi_sqr, label='$\chi^2$', zorder=1)
ax[1].set_xlabel('end index')
ax[1].set_ylabel('Chi-Square')
```

```
ax[0].legend()
ax[1].legend()
print(f'Minimum chi-sqr (best number of indices) at index e_idx = {np.
 ↪argmin(chi_sqr)+1}')

fig.savefig('Abb/Ex5_Correlation_Taylor_length_different_idx.png')
```

```
Minimum chi-sqr (best number of indices) at index e_idx = 3
```

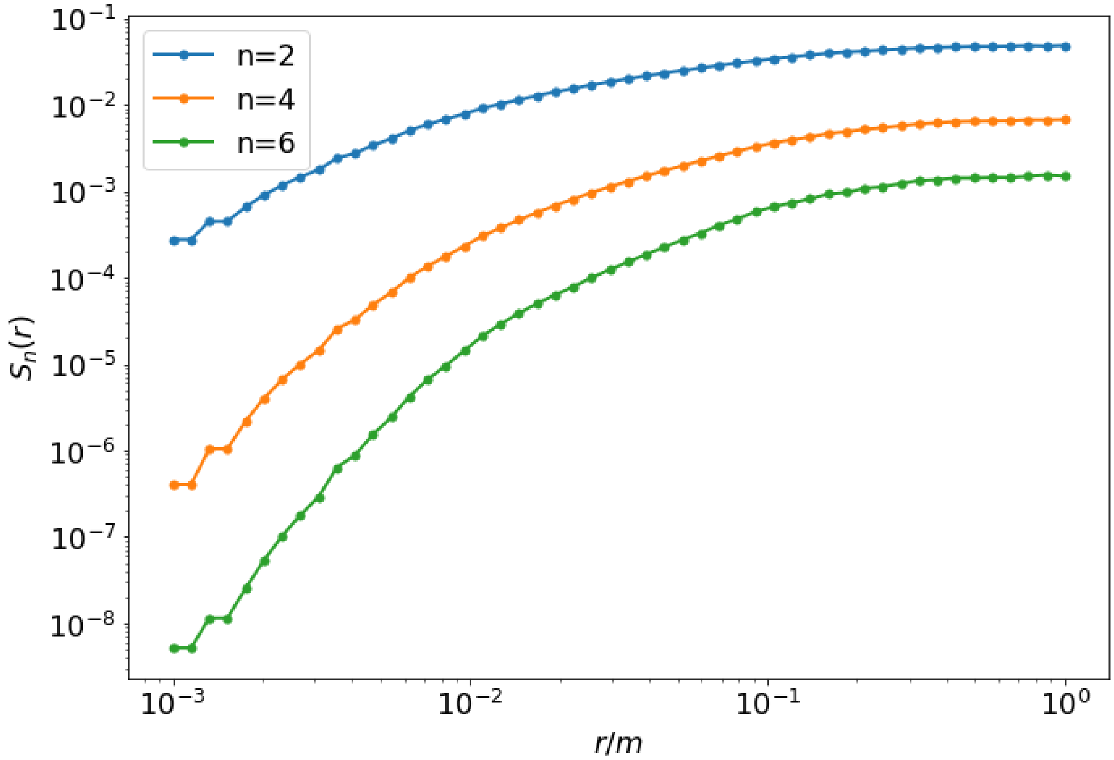


5.6) Calculate Structure Function $S_n(r) = \langle [u(x+r) - u(x)]^n \rangle$

[ ]:
```
dr = u_mean/f_s
r_in_log = np.logspace(-3, 0, 50)
struc = np.empty((7,np.size(r_in_log)))

for n in range(1,7):
    for r in range(np.size(r_in_log)):
        r_idx = int(r_in_log[r]/dr)
        struc[n,r] = np.mean((data[:-r_idx]-data[r_idx:])**n)
```

[ ]:
```
# Even structure functions
fig, ax = plt.subplots(figsize=(10,7), tight_layout=True)

dr = u_mean/f_s
ax.loglog(r_in_log, struc[2], label='n=2', marker='.', ms=10)
ax.loglog(r_in_log, struc[4], label='n=4', marker='.', ms=10)
ax.loglog(r_in_log, struc[6], label='n=6', marker='.', ms=10)

ax.set_xlabel('$r / m$')
```

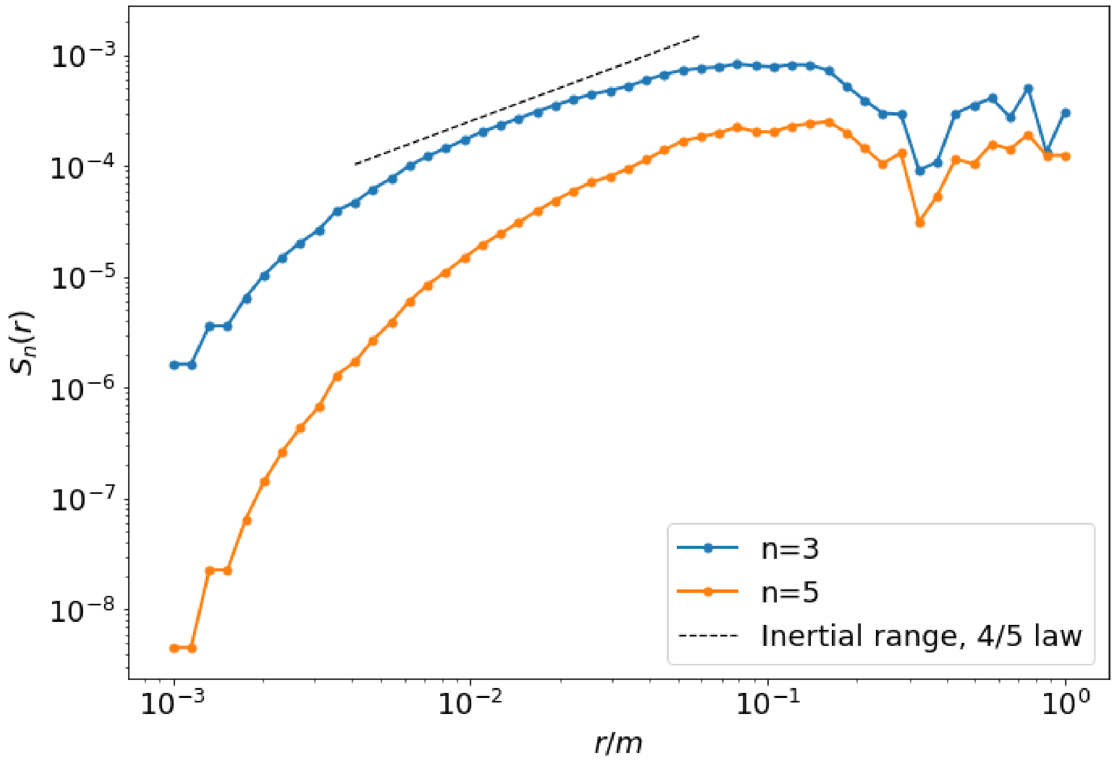
```python
ax.set_ylabel('$S_n(r)$')
ax.legend()
plt.savefig('Abb/Ex5_Even_structure_functions')
```



A power-law like r^{ζ_n} should look like a linear graph with slope ζ_n
When r is large (approximately L) then we have Gaussian fluctuations and the model isn't that good anymore

[ ]:
```python
# Odd structure functions
fig, ax = plt.subplots(figsize=(10,7), tight_layout=True)

ax.loglog(r_in_log ,-struc[3], label='n=3', marker='.', ms=10)
ax.loglog(r_in_log ,-struc[5], label='n=5', marker='.', ms=10)

# S_3(r,t) = -4/5 <epsilon> r
# Inertial range
y = 4/5*10**(-1.5)*r_in_log[10:30]
ax.plot(r_in_log[10:30], y, label='Inertial range, 4/5 law',␣
 ↪linestyle='dashed', color='black', lw=1.2)

ax.set_xlabel('$r / m$')
ax.set_ylabel('$S_n(r)$')
ax.legend()
```

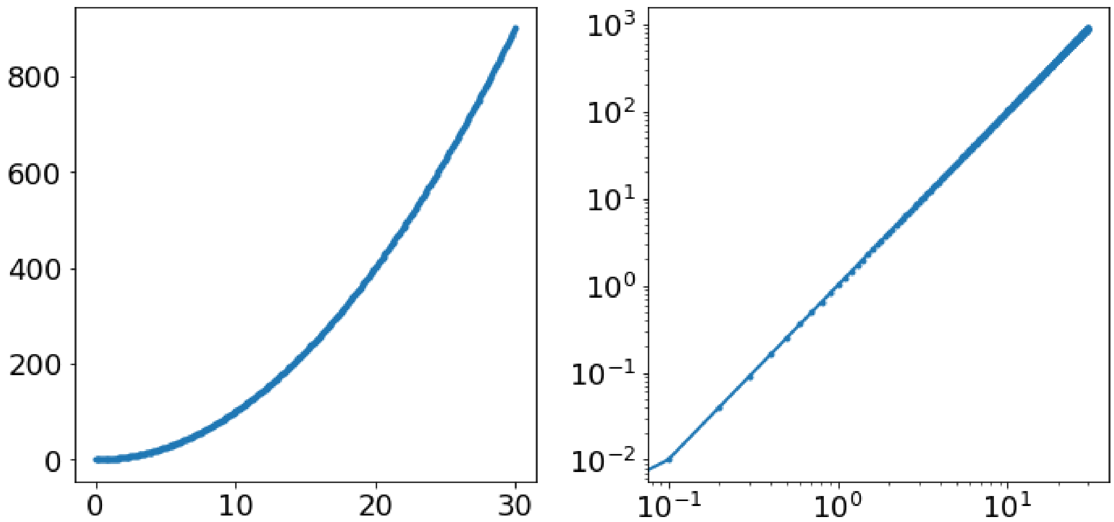
```
plt.savefig('Abb/Ex5_Odd_structure_functions')
```



A power-law like r^{ζ_n} should look like a linear graph with slope ζ_n if ζ_n is only a number. Here for example $\zeta_n = 2$

[ ]:
```
fig, ax = plt.subplots(1,2, figsize=(10,5), tight_layout=True)
r_lin = np.linspace(0,30,300)
y = np.power(r_lin,2)
ax[0].plot(r_lin,y, marker='.')
ax[1].loglog(r_lin, y, marker='.')
```

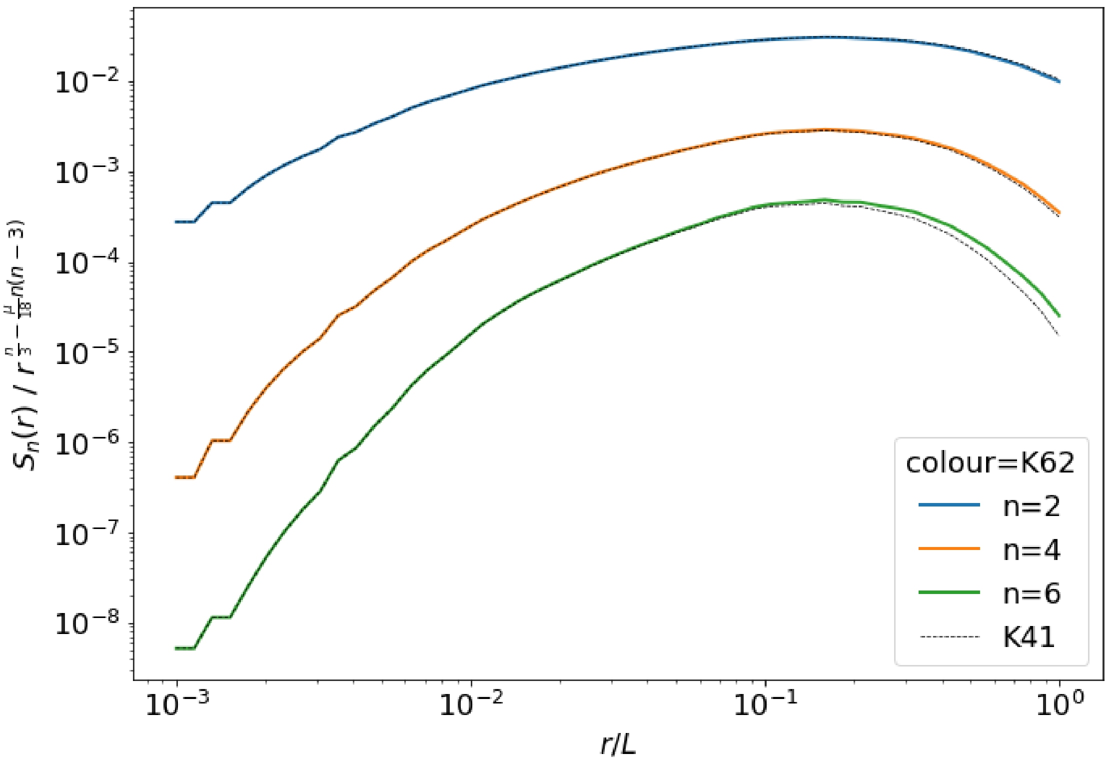
[ ]: [<matplotlib.lines.Line2D at 0x1b513ba1550>]



$$
\begin{aligned}
S_n(r) &= C_n \cdot \langle\epsilon\rangle \ \cdot r^{\frac{n}{3}} \cdot (\tfrac{L}{r})^{\frac{\mu}{18}n(n-3)} \\
S_n(r) &= C_n' \cdot \langle\epsilon\rangle \ \cdot r^{\frac{n}{3}-\frac{\mu}{18}n(n-3)}
\end{aligned}
$$

[ ]:

```
# even structure functions
fig, ax = plt.subplots(figsize=(10,7), tight_layout=True)

# From lecture mu is usually between 0.2 < mu < 0.28
# No "real" reason, just from experimental data
r_in_lin = np.power(10,r_in_log)

mu =0.227
ax.loglog(r_in_log, struc[2]/r_in_lin**(2/3+mu/9) , label='n=2')
ax.loglog(r_in_log, struc[2]/r_in_lin**(2/3) , color='black', ls='dashed', lw=0.7)
ax.loglog(r_in_log, struc[4]/r_in_lin**(4/3-2*mu/9), label='n=4')
ax.loglog(r_in_log, struc[4]/r_in_lin**(4/3), color='black', ls='dashed', lw=0.7)
ax.loglog(r_in_log, struc[6]/r_in_lin**(6/3-mu), label='n=6')
ax.loglog(r_in_log, struc[6]/r_in_lin**(6/3), color='black', ls='dashed', lw=0.7, label='K41')

ax.set_xlabel('$r/L$', fontsize=18)
ax.set_ylabel(r'$S_n(r)\ /\ r^{\ \frac{n}{3}-\frac{\mu}{18} n (n-3)}$', fontsize=18)
ax.legend(title='colour=K62')
plt.savefig('Abb/Ex5_Even_structure_functions_with_K41')
```

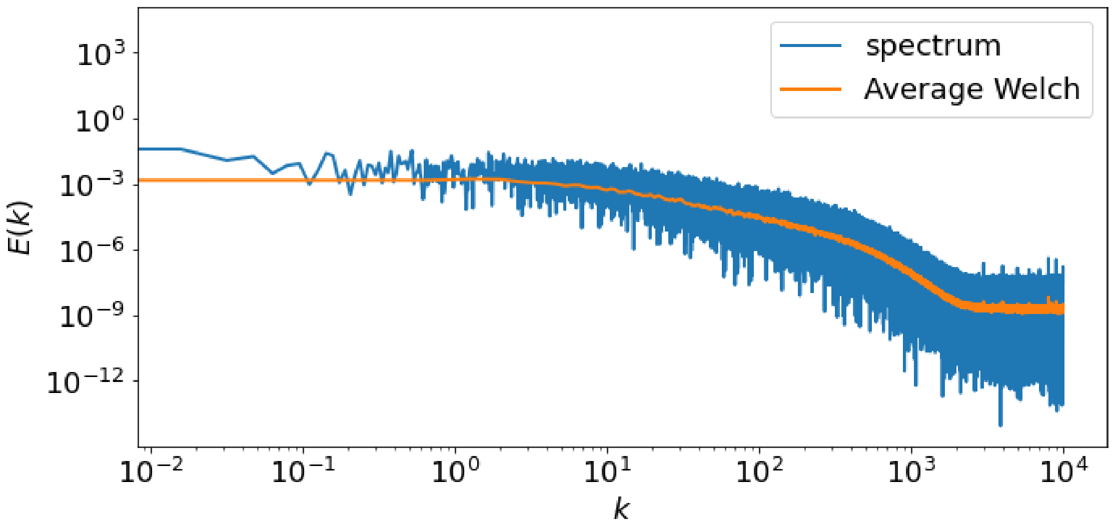


3 Exercise 6

6.1) Determine the energy spectrum $E(k)$ by using Fourier transform of the signal

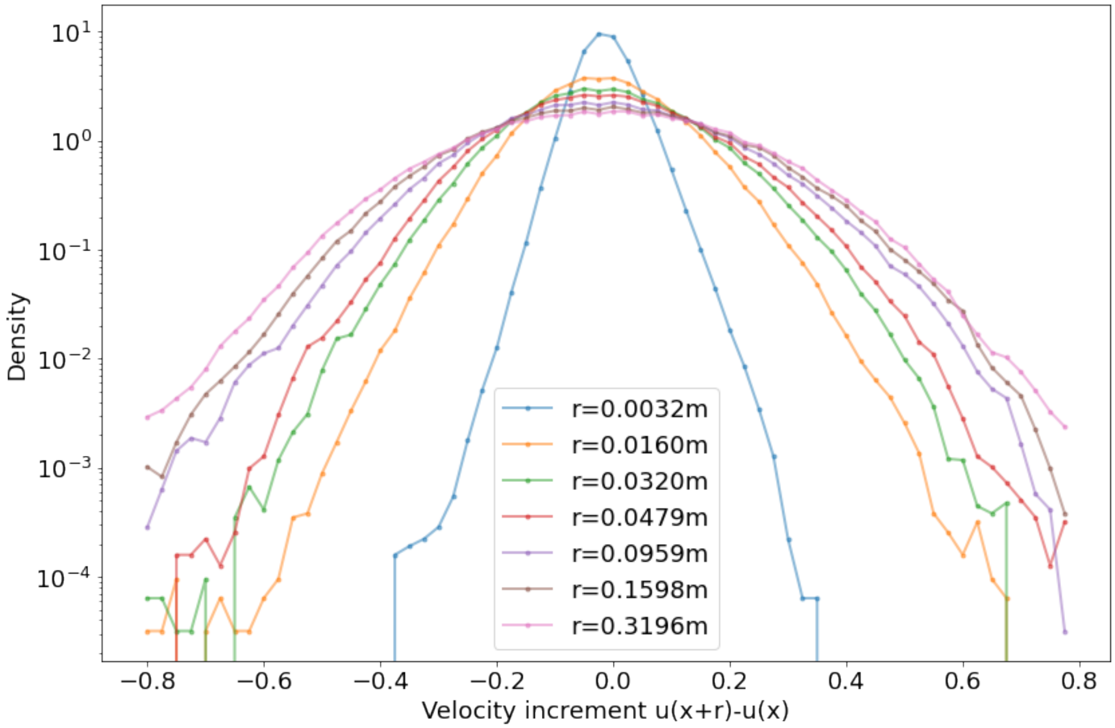
[ ]:
```
data_fourier = np.fft.rfft(data)/np.size(data)*20
k = np.linspace(0, 20*1000/2, np.size(data)//2+1)
spectrum = np.abs(data_fourier**2)

# Estimate power spectral density using Welch's method.
freq_welch, spectrum_average = signal.welch(data, fs=20*1000, nperseg=30000)
# f, Pxx_averaged = signal.welch(data, fs=20*1000, average='median')
```

[ ]:
```
fig, ax = plt.subplots(figsize=(10,5), tight_layout=True)
ax.loglog(k,spectrum, label='spectrum')
ax.loglog(freq_welch, spectrum_average, label='Average Welch')
ax.set_xlabel('$k$')
ax.set_ylabel('$E(k)$')
ax.legend()
plt.savefig('Abb/Ex6_Spectrum')
```



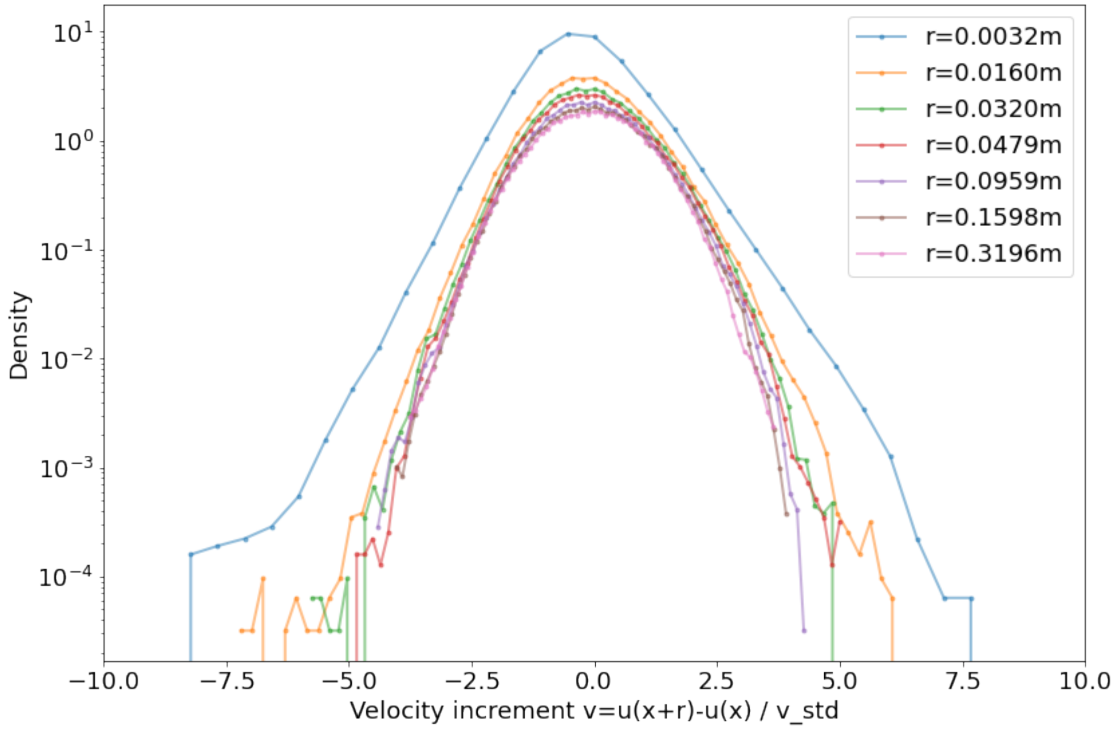
6.2) Compute histogram of the velocity increments $u(x+r) - u(x)$

[ ]:
```
fig, ax = plt.subplots(figsize=(12,8), tight_layout=True)
dr = u_mean/f_s

r_vec_idx = [10,50,100,150,300,500,1000]
for r in r_vec_idx:
    velocity_increments = data[r:]-data[:-r]

    hist, bins = np.histogram(velocity_increments, bins=64, density=True,␣
 ↪range=[-0.8,0.8])
    std_increment = np.std(velocity_increments)
    ax.semilogy(bins[:-1], hist, alpha=0.5, marker='.', label=f'r={(dr*r):.
 ↪4f}m')

ax.set_xlabel('Velocity increment u(x+r)-u(x)')
ax.set_ylabel('Density')
ax.legend()
plt.savefig('Abb/Ex6_Density')
```



For a better comparison, plot x-axis as multiple of standard deviation

[ ]:
```
fig, ax = plt.subplots(figsize=(12,8), tight_layout=True)
dr = u_mean/f_s

r_vec_idx = [10,50,100,150,300,500,1000]
for r in r_vec_idx:
    velocity_increments = data[r:]-data[:-r]

    hist, bins = np.histogram(velocity_increments, bins=64, density=True,␣
 ↪range=[-0.8,0.8])
    std_increment = np.std(velocity_increments)
    ax.semilogy(bins[:-1]/std_increment, hist, alpha=0.5, marker='.',␣
 ↪label=f'r={(dr*r):.4f}m')

ax.set_xlabel('Velocity increment v=u(x+r)-u(x) / v_std')
ax.set_ylabel('Density')
ax.set_xlim(-10,10)
ax.legend()
plt.savefig('Abb/Ex6_Density_in_std')
```

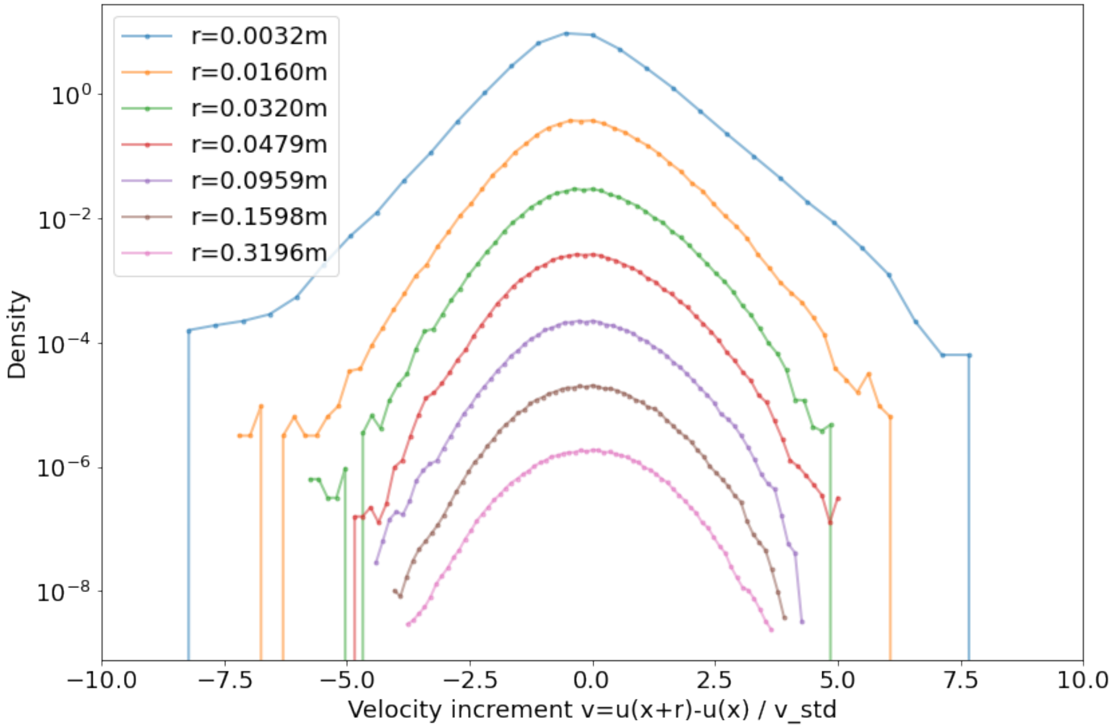


6.3) Now we multiply with a factor of 10^{-i} so the histograms are not on top of eachother.
We can clearly see that the probability densitity functions are NOT self-similar. For small r so in the region of the Kolmogorov microscale we see a Non-Gaussian pdf and on large scale a Gaussian pdf. For self-similarity the pdf's must be the whole time Gaussian or Non-Gaussian, so independent of scale r.

The pdf of small r could be described by a quasi-normal Ansatz –> superstatistics with summation of wheighted Gaussians with different variances

[ ]:
```
fig, ax = plt.subplots(figsize=(12,8), tight_layout=True)
dr = u_mean/f_s

r_vec_idx = [10,50,100,150,300,500,1000]
for i, r in enumerate(r_vec_idx):
    velocity_increments = data[r:]-data[:-r]

    hist, bins = np.histogram(velocity_increments, bins=64, density=True,␣
↪range=[-0.8,0.8])
    std_increment = np.std(velocity_increments)
    ax.semilogy(bins[:-1]/std_increment, hist*10**(-i), alpha=0.5, marker='.',␣
↪label=f'r={(dr*r):.4f}m')

ax.set_xlabel('Velocity increment v=u(x+r)-u(x) / v_std')
ax.set_ylabel('Density')
```

```
ax.set_xlim(-10,10)
ax.legend()
plt.savefig('Abb/Ex6_Density_in_std_moved')
```

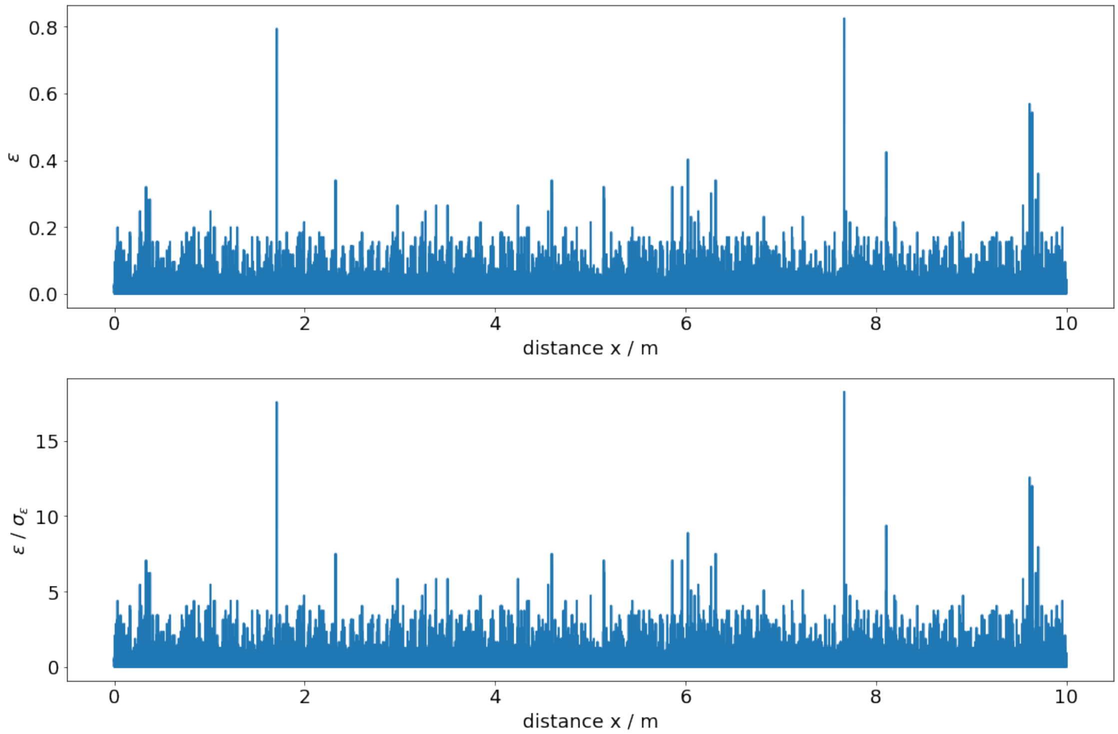


4 Exercise 7

7.1) Calculate local energy dissipation rate $\epsilon(x) = 2\nu(\partial_x u(x))^2$, kinematic viscosity air $\nu \approx 0.15 cm^2/s$

[ ]:
```
fig, ax = plt.subplots(2,1,figsize=(15,10), tight_layout=True)

nu = 0.15 * 10**(-4)
du = data[1:] - data[:-1]
epsilon = 2 * nu * (du/dr)**2
epsilon_std = np.std(epsilon)

# end in meter
end = 10
idx_e = int(end/u_mean*f_s)

ax[0].plot(spatial[:idx_e], epsilon[:idx_e])
ax[0].set_xlabel('distance x / m')
ax[0].set_ylabel('$\epsilon$')
```

```
ax[1].plot(spatial[:idx_e], epsilon[:idx_e]/epsilon_std)
ax[1].set_xlabel('distance x / m')
ax[1].set_ylabel('$\epsilon\ /\ \sigma_{\epsilon}$')
fig.savefig('Abb/Ex7_Epsilon.png')
```



Taylor length $\lambda = \sqrt{15 \cdot \frac{\nu}{\langle\epsilon\rangle}} \cdot u_{rms}$ with $\vec{u}_{rms} = \sqrt{\frac{\langle\vec{u}^2\rangle}{N}}$

Kolmogorov microscale $\eta = (\frac{\nu^3}{\langle\epsilon\rangle})^{1/4}$ from dimensional analyis $[\langle\epsilon\rangle] = m^2/s^3$ and $[\nu] = m^2/s$ see also π-theorem

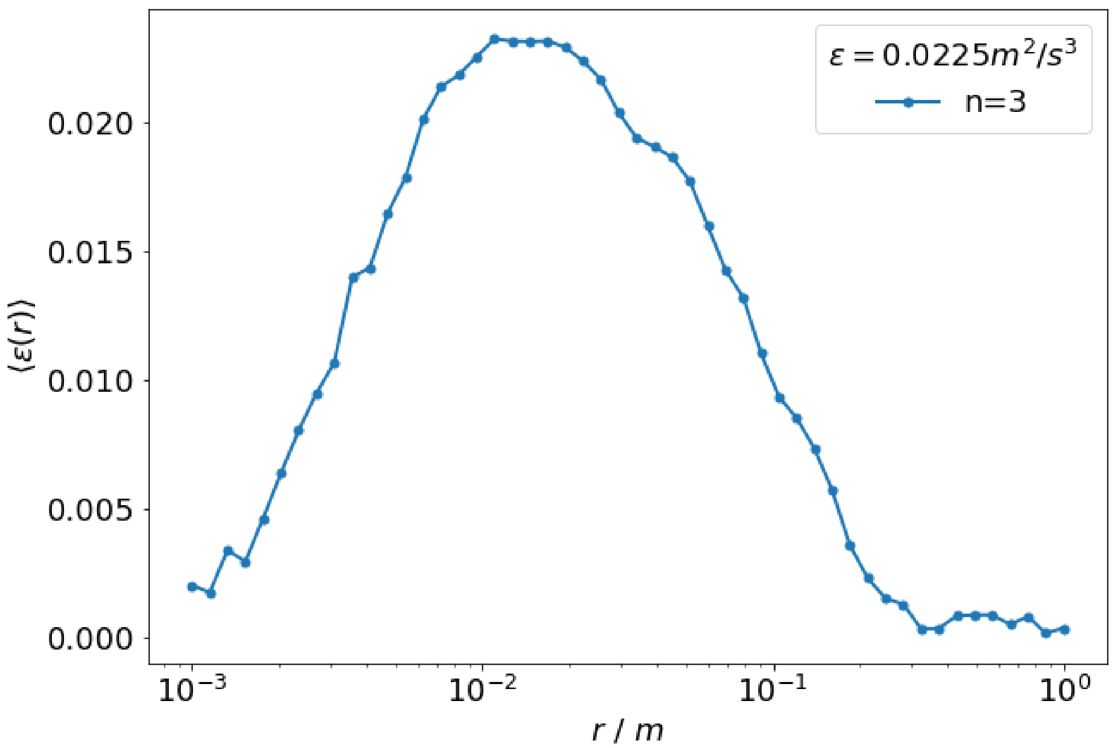
```
[ ]: taylor_l_eps = np.sqrt(15*nu/np.mean(epsilon)) * np.sqrt(np.mean(np.
      ↪square(data_fluc)))
     kolmog_microscale = (nu**3/np.mean(epsilon))**0.25
     print(f'Taylor length with local energy dissipation rate: {taylor_l_eps}m')
     print(f'The Kolmogorov microscale is: {kolmog_microscale}m')
     print(f'Average dissipation rate: {np.mean(epsilon)}m^2/s^3')
```

```
Taylor length with local energy dissipation rate: 0.016887069520983355m
The Kolmogorov microscale is: 0.0006420463044883548m
Average dissipation rate: 0.019861332169961 04m^2/s^3
```

7.1) From 4/5 law $S_3(r,t) = -\frac{4}{5}\langle\epsilon\rangle r => \langle\epsilon\rangle = S_3(r) \cdot -\frac{5}{4} \cdot \frac{1}{r}$

[ ]:
```
fig, ax = plt.subplots(figsize=(10,7))
eps_avg = -5/4*struc[3]/r_in_log
ax.semilogx(r_in_log,eps_avg, label='n=3', marker='.', ms=10)

arg_max = np.argmax(eps_avg)
eps_avg_struct_func = np.mean(eps_avg[arg_max-3:arg_max+3])
print(f'Average dissipation rate by structure function = {eps_avg_struct_func:.
 ↪4f}m^2/s^3')
ax.set_xlabel('$r\ /\ m$')
ax.set_ylabel(r'$\langle \epsilon(r) \rangle$')
ax.legend(title = f'$\epsilon=${eps_avg_struct_func:.4f}$m^2/s^3$')
plt.savefig('Abb/Ex7_Epsilon_from_4_5_law')
```

```
Average dissipation rate by structure function = 0.0225m^2/s^3
```

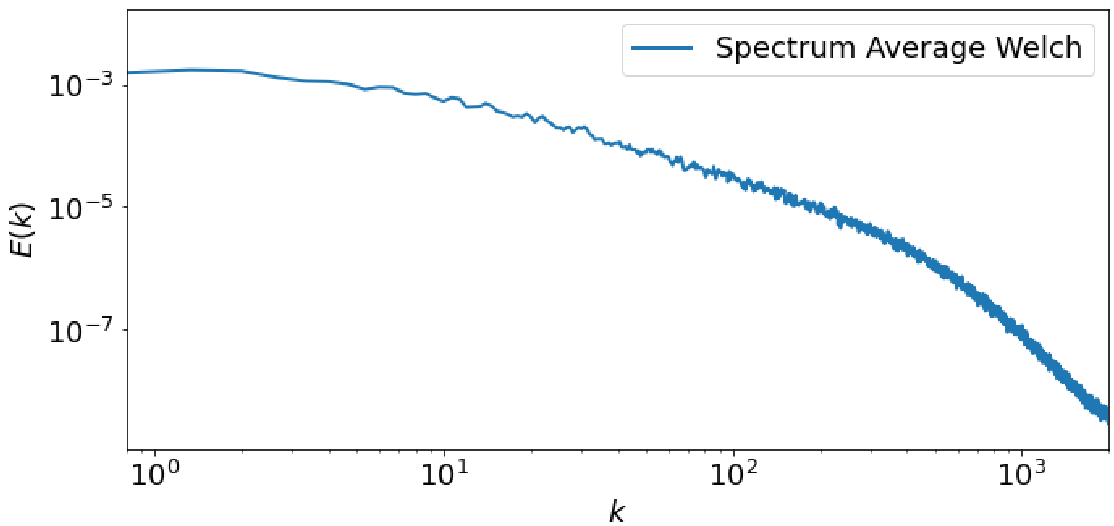


7.1) From E(k), K41 phenomeology. In the inertial range we have $E(k) = C_k \langle \epsilon \rangle^{2/3} \cdot k^{-5/3}$
We have three ranges: forcing, inertial range, dissipation range

[ ]:
```
fig, ax = plt.subplots(figsize=(10,5), tight_layout=True)
ax.loglog(freq_welch, spectrum_average, label='Spectrum Average Welch')

ax.set_xlim(0.8,2000)
ax.set_ylim(np.min(spectrum_average), np.max(spectrum_average)*10)
```

```
ax.set_xlabel('$k$')
ax.set_ylabel('$E(k)$')
ax.legend()
```

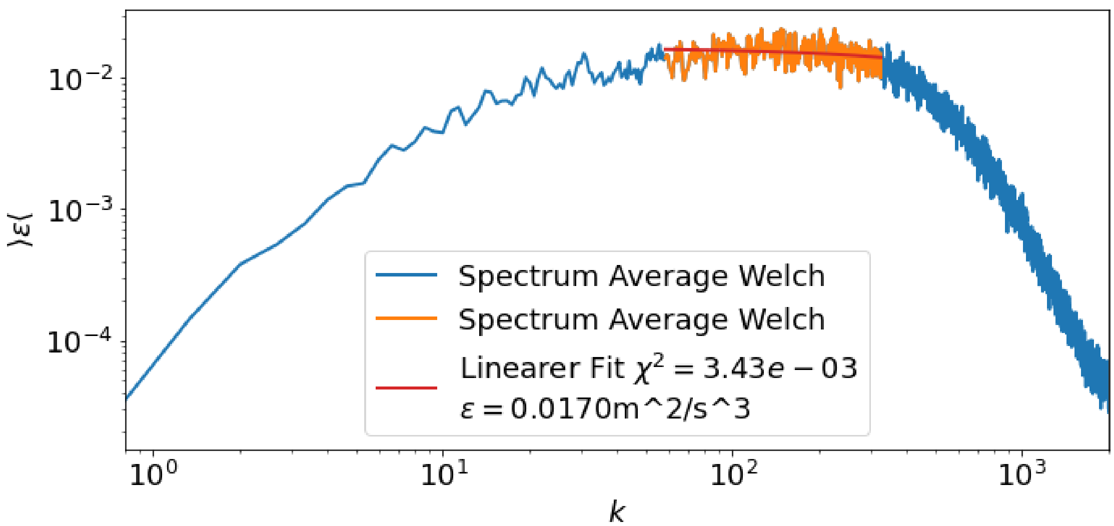
[ ]: <matplotlib.legend.Legend at 0x1b5125e2670>



[ ]:
```
fig, ax = plt.subplots(figsize=(10,5), tight_layout=True)
eps_avg = (spectrum_average * freq_welch**(5/3))**(3/2)

ax.loglog(freq_welch, eps_avg, label='Spectrum Average Welch')
ax.set_xlim(0.8,2000)

arg_max = np.argmax(eps_avg)

N_l, N_r = 250, 150
ax.loglog(freq_welch[arg_max-N_l:arg_max+N_r], eps_avg[arg_max-N_l:
 ↪arg_max+N_r], label='Spectrum Average Welch')
eps_avg_struct_func = np.mean(eps_avg[arg_max-N_l:arg_max+N_r])

# Fit Linear Model to the orange data points
mod = lmfit.models.LinearModel()
xdat = freq_welch[arg_max-N_l:arg_max+N_r]
ydat = eps_avg[arg_max-N_l:arg_max+N_r]

result = mod.fit(ydat, x=xdat)
eps_avg_spectrum = result.values['intercept']
print(f'Average dissipation rate by structure function = {eps_avg_spectrum:.
 ↪4f}m^2/s^3')
```

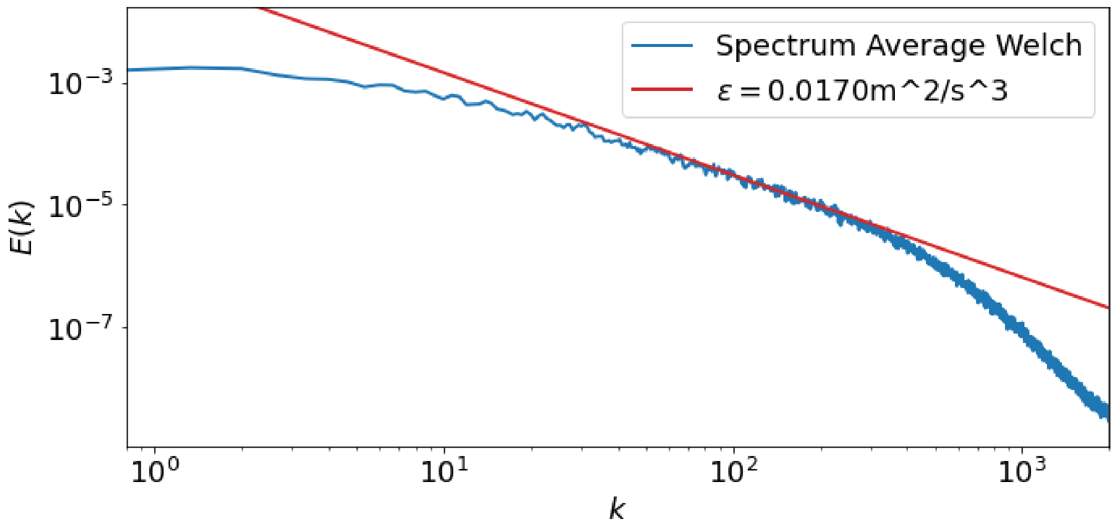
```
ax.plot(xdat, result.eval(result.params, x=xdat), c='C3', label=f'Linearer Fit␣
↪$\chi^2={{{result.chisqr:.2e}}}$\n$\epsilon = {{{eps_avg_spectrum:.4f}}}$m^2/
↪s^3')

ax.set_xlabel('$k$')
ax.set_ylabel(r'$\rangle \epsilon \langle$')
ax.legend()
plt.savefig('Abb/Ex7_Epsilon_from_K41_spectrum')
```

```
Average dissipation rate by structure function = 0.0170m^2/s^3
```



Plot of E(k) but with the determined epsilon

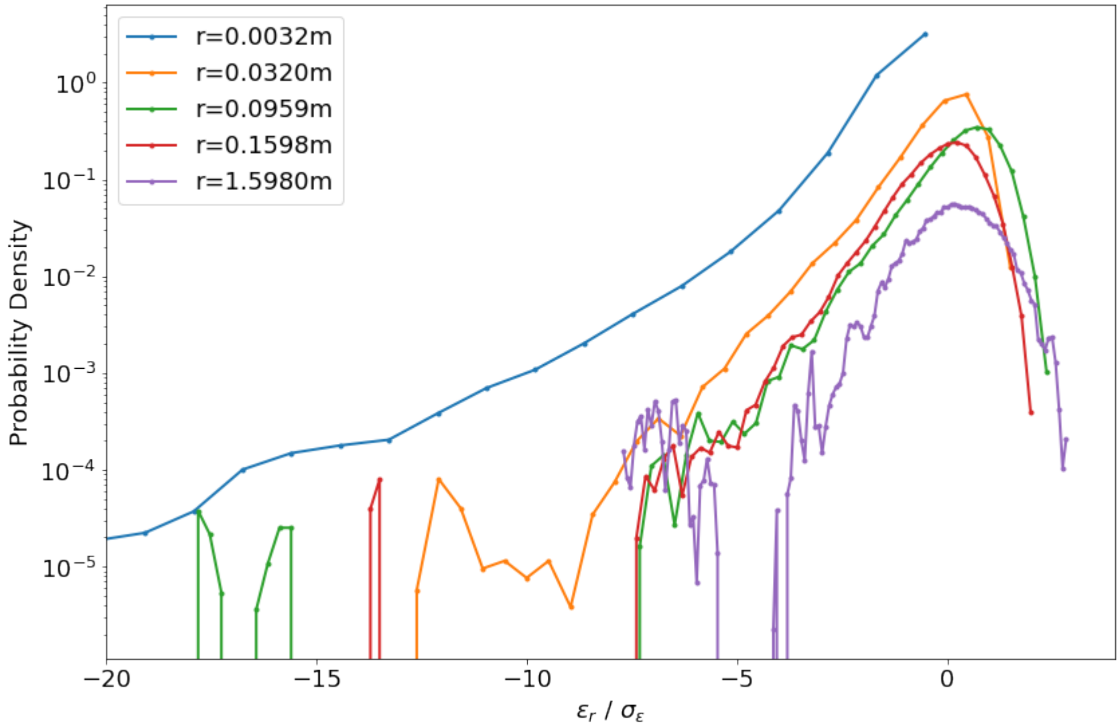
```
[ ]: fig, ax = plt.subplots(figsize=(10,5), tight_layout=True)
ax.loglog(freq_welch, spectrum_average, label='Spectrum Average Welch')
E_k = eps_avg_spectrum**(2/3)*freq_welch**(-5/3)
ax.plot(freq_welch, E_k, c='C3', label=f'$\epsilon = {{{eps_avg_spectrum:.
↪4f}}}$m^2/s^3')

ax.set_xlim(0.8,2000)
ax.set_ylim(np.min(spectrum_average), np.max(spectrum_average)*10)
ax.set_xlabel('$k$')
ax.set_ylabel('$E(k)$')
ax.legend()
plt.savefig('Abb/Ex7_Spectrum_wwith_epsilon')
```



7.2) Calculate the scale resolved energy dissipation rate $\epsilon_r(x) = \int_x^{x+r} dx' \epsilon(x')$.

For large scale r we have a more Gaussian curve and for very small scales we have a different probability function (log-normal).

[ ]:
```
fig, ax = plt.subplots(1,1,figsize=(12,8), tight_layout=True)
r_vec_idx = [10,100,300,500,5000]
std_eps_vec = np.empty(len(r_vec_idx))

for i, r in enumerate(r_vec_idx):
    epsilon_r = np.cumsum(np.roll(epsilon,r)[r:]) - np.cumsum(epsilon[r:])
    hist, bins = np.histogram(epsilon_r, density=True, bins=128)
    std_epsilon = np.std(epsilon_r)
    std_eps_vec[i] = std_epsilon
    ax.semilogy(bins[:-1]/std_epsilon, hist, marker='.', label=f'r={(dr*r):.
↪4f}m')

ax.set_xlabel('$\epsilon_r \ / \ \sigma_{\epsilon}$')
ax.set_ylabel('Probability Density')
plt.xlim(-20, 4)
ax.legend()
plt.savefig('Abb/Ex7_Resolved_energy_dissipation')
```



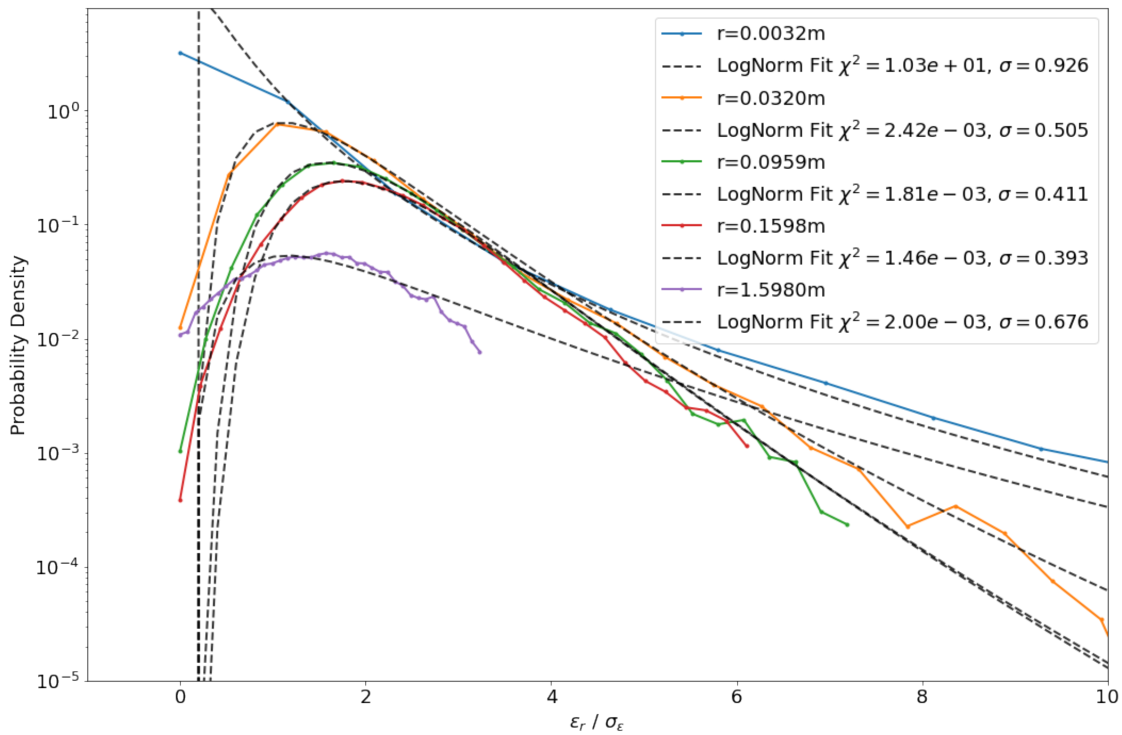
6.5) Check whether the histogram follows a lognormal distribution and try to determine the intermittency coefficient from $\sigma(r)^2 = A + \mu \ln(\frac{L}{r})$. For $n = 3$ we had from lecture 15.06 $\langle \epsilon \rangle = ae^{\sigma^2/2}, \sigma^2 = A + \mu \ln \frac{L}{r}$

For large r the fit is not good because it is more a normal Gaussian

[ ]:
```
fig, ax = plt.subplots(1,1,figsize=(15,10), tight_layout=True)
r_vec_idx = [10, 100, 300, 500, 5000]
std_eps_vec = np.empty(len(r_vec_idx))
sigma_vec = np.empty(len(r_vec_idx))
xdat_smooth = np.linspace(0,20, 100)
for i, r in enumerate(r_vec_idx):

    epsilon_r = np.cumsum(np.roll(epsilon,r)[r:]) - np.cumsum(epsilon[r:])
    hist, bins = np.histogram(epsilon_r, density=True, bins=128)
    std_epsilon = np.std(epsilon_r)

    arg_max = np.argmax(hist)
    N_l, N_r = 20, 20
    # Keep the index in bounds
    if arg_max + N_l >= 128:
        N_l = 128 - arg_max

    xdat = (bins[arg_max-N_r:arg_max+N_l])
```

```
    xdat = -(xdat - np.max(xdat))
    xdat = xdat /std_epsilon
    ydat = hist[arg_max-N_r:arg_max+N_l]

    ax.semilogy(xdat, ydat, marker='.', label=f'r={(dr*r):.4f}m')

    # Fit Log Normal model to the data points
    mod = lmfit.models.LognormalModel()
    result = mod.fit(ydat, x=xdat)
    sigma = result.values['sigma']
    sigma_vec[i] = sigma
    print(f'sigma: {sigma}')

    ax.plot(xdat_smooth, result.eval(result.params, x=xdat_smooth), c='k',␣
↪ls='--', alpha=0.8, label=f'LogNorm Fit $\chi^2={{{result.chisqr:.2e}}}$,␣
↪$\sigma=${sigma:.3f}')

ax.set_xlabel('$\epsilon_r \ / \ \sigma_{\epsilon}$')
ax.set_ylabel('Probability Density')
ax.set_ylim(10**(-5),8)
ax.set_xlim(-1,10)
ax.legend()
plt.savefig('Abb/Ex7_Resolved_energy_dissipation_sigma_fit')
```

```
sigma: 0.9263541881786528
sigma: 0.505437918046481
sigma: 0.4113939820145063
sigma: 0.39252788746521516
sigma: 0.675599521066637
```

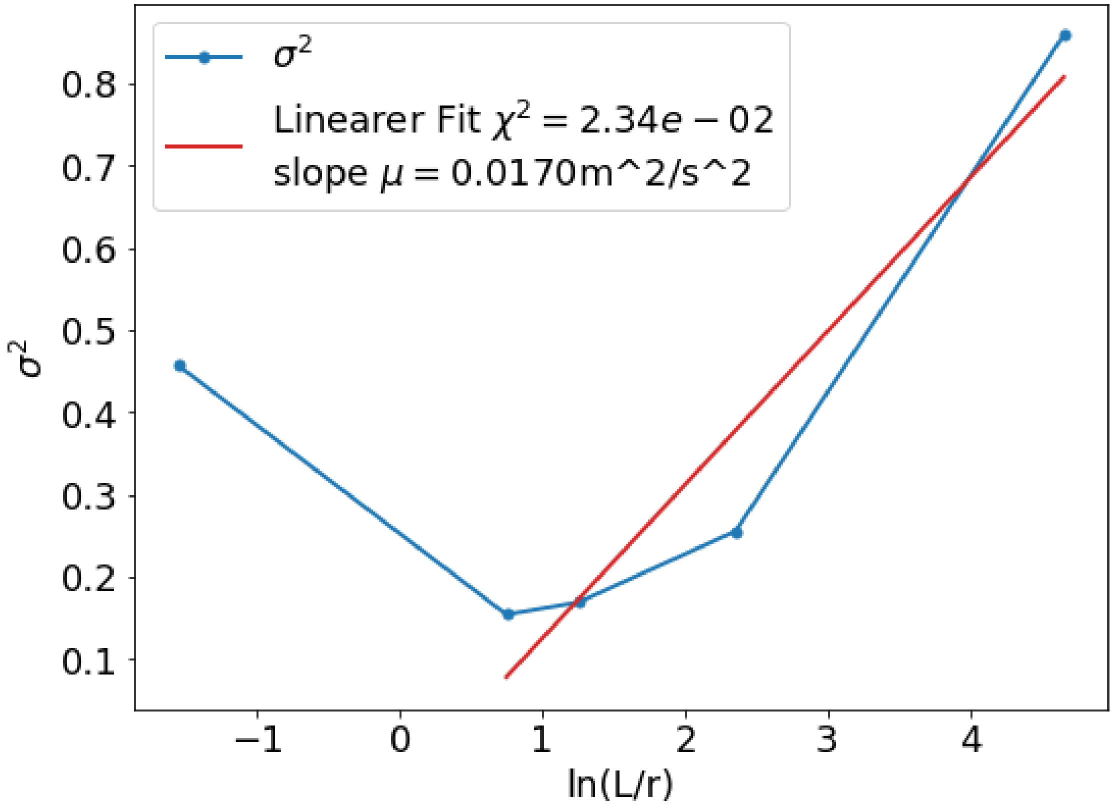


Plot σ^2 over $1/r$

[ ]:
```
fig, ax = plt.subplots(1,1,figsize=(8,6), tight_layout=True)
r = np.array([spatial[i] for i in r_vec_idx])
ax.plot(np.log(L_fit/r), sigma_vec**2, marker='.', ms=10, label='$\sigma^2$')

# Fit Linear Model to th orange data points
mod = lmfit.models.LinearModel()
xdat = np.log(L_fit/r)[:-1]
ydat = sigma_vec[:-1]**2




result = mod.fit(ydat, x=xdat)
mu_computed = result.values['slope']
print(f'Average dissipation rate by structure function = {mu_computed:.4f}m^2/
 ↪s^3')
ax.plot(xdat, result.eval(result.params, x=xdat), c='C3', label=f'Linearer Fit␣
 ↪$\chi^2={{{result.chisqr:.2e}}}$\nslope $\mu = {{{eps_avg_spectrum:.
 ↪4f}}}$m^2/s^2')


ax.set_xlabel('ln(L/r)')
ax.set_ylabel('$\sigma^2$')
```

```
ax.legend()
plt.savefig('Abb/Ex7_intermittency_via_sigma_squared')
```

```
Average dissipation rate by structure function = 0.1863m^2/s^3
```



5 Exercise 8

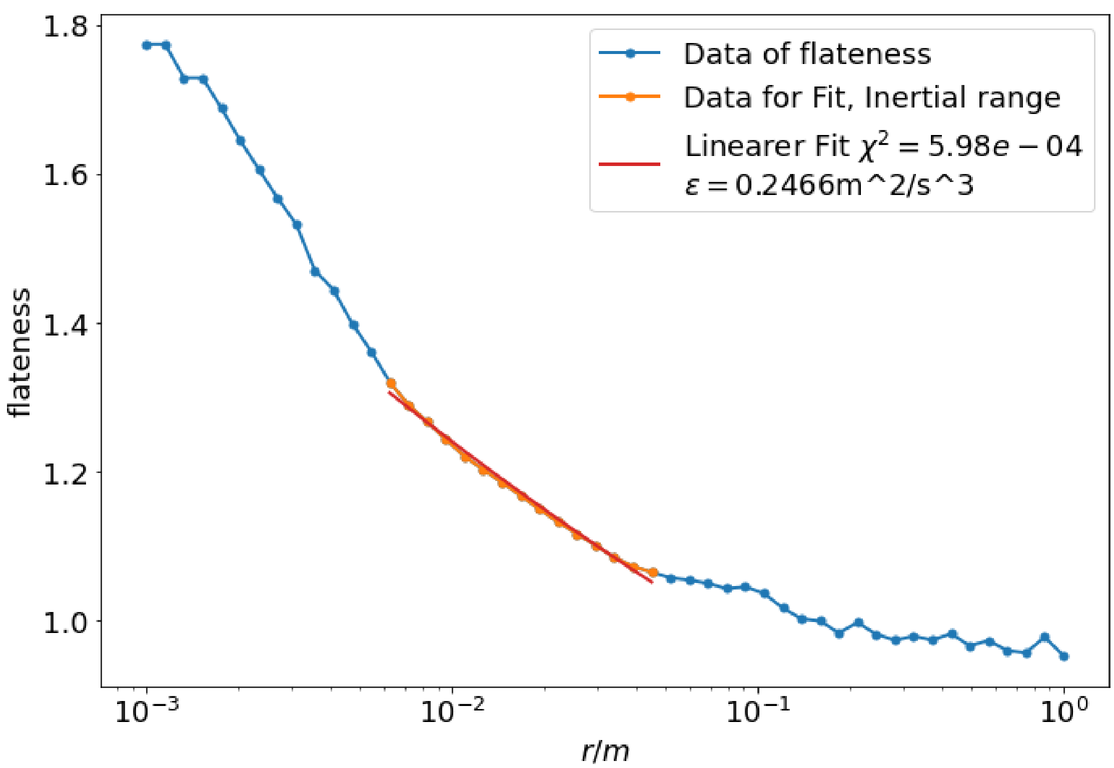
8.1) Calculate flatness $S_4(r)/3S_2(r)^2$, $S_n(r) = r^{\zeta_n}$ with $\zeta_n = \frac{n}{3} - \frac{18}{\mu}n(n-3)$ so we get as flateness=$\frac{1}{3}r^{-\frac{4\mu}{9}}$. Flateness with $S_2(r)^2$ NOT 3 as on the exercise sheet. The flateness for a Gaussian would be one because the quadratic n^2 term was only there due to the Non Gaussianity.

The μ via flateness is in a good and plausible range

[ ]:
```
fig, ax = plt.subplots(1,1,figsize=(10,7), tight_layout=True)
ax.semilogx(r_in_log, struc[4]/(3*struc[2]**2), label='Data of flateness',␣
↪marker='.', ms=10)
idx_b, idx_e = 13,28

xdat = r_in_log[idx_b:idx_e]
ydat = (struc[4]/(3*struc[2]**2))[idx_b:idx_e]
ax.semilogx(xdat, ydat, label='Data for Fit, Inertial range', marker='.', ms=10)
```

```python
# Fit Linear Model to th orange data points
mod = lmfit.models.PowerLawModel()
# mod.set_param_hint('amplitude', vary=True)
# pars = mod.make_params(amplitude=1/3)
result = mod.fit(ydat, x=xdat)
mu_flateness = result.values['exponent']
print(f'Mu via flateness = {mu_flateness*(-9/4):.4f}')
ax.plot(xdat, result.eval(result.params, x=xdat), c='C3', label=f'Linearer Fit␣
↪$\chi^2={{{result.chisqr:.2e}}}$\n$\epsilon = {{{mu_flateness*(-9/4):.
↪4f}}}$m^2/s^3')

ax.set_xlabel('$r / m$')
ax.set_ylabel('flateness')
ax.legend()
plt.savefig('Abb/Ex8_flateness_and_mu')
```

```
Mu via flateness = 0.2466
```



[ ]: