In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt


from uncertainties import ufloat
import uncertainties.unumpy as unp

from scipy.optimize import curve_fit
import scipy.odr as odr

from glob import glob
```

In [ ]:
```python
laserSpotHeight = [155, 157]
platformHeightFromGround = [108, 107]
containerHeight = [3.3, 3.6]

heights = [(h1 - (h2 + h3))/100 for h1, h2, h3 in zip(laserSpotHeight, platformHeightFromGround, containerHeight)]
print(f"Height measurements: {heights}")

H = ufloat(np.mean(heights), np.std(heights)) # meters
print(H)

L = ufloat(5.6, 0.05) # meters

lightWavelength = 633e-9 # m
waterDensity = 998.2 # kg m^-3
soapDensity = 1000 * 64.4/60

grazingAngle = unp.arctan2(H, L)

rootPath = "/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/"
```

```
Height measurements: [0.43700000000000006, 0.4640000000000001]
0.451+/-0.014
```

In [ ]:
```python
def LoadData(path):
    data = np.loadtxt(path, skiprows=1)

    frequencies = data[:,0]
    spacings = data[:,1]/1000 # change mm to m
    spacingUncertanties = data[:,2]/1000

    spacings = np.array([ufloat(spacing, uncertainty) for spacing, uncertainty in zip(spacings, spacingUncertanties)])

    return (frequencies, spacings)
```

In [ ]:
```python
def GetAngularFrequencies(frequencies):
    angularFrequencies = 2 * np.pi * frequencies
    return angularFrequencies
```

In [ ]:
```python
def GetDiffractionAngles(spacings):
    # Input in meters will give an output in degrees

    diffractionAngles = unp.arctan2(spacings, L)

    return diffractionAngles
```

In [ ]:
```python
def GetWaveNumbers(diffractionAngles):
    innerBracket = unp.sin(grazingAngle + diffractionAngles / 2) + unp.sin(grazingAngle - diffractionAngles / 2)

    waveNumbers = 2 * (np.pi / lightWavelength) * unp.sin(diffractionAngles / 2) * innerBracket

    return waveNumbers
```

In [ ]:
```python
def LinearFunc(p, x):
    m, c = p
    return m * x + c

def PerformODR(function, x, y, xErr):
    model = odr.Model(function)

    data = odr.Data(x, y, wd= 1/xErr)

    odrOutput = odr.ODR(data, model, beta0=[0.03, 0])

    return odrOutput.run()
```

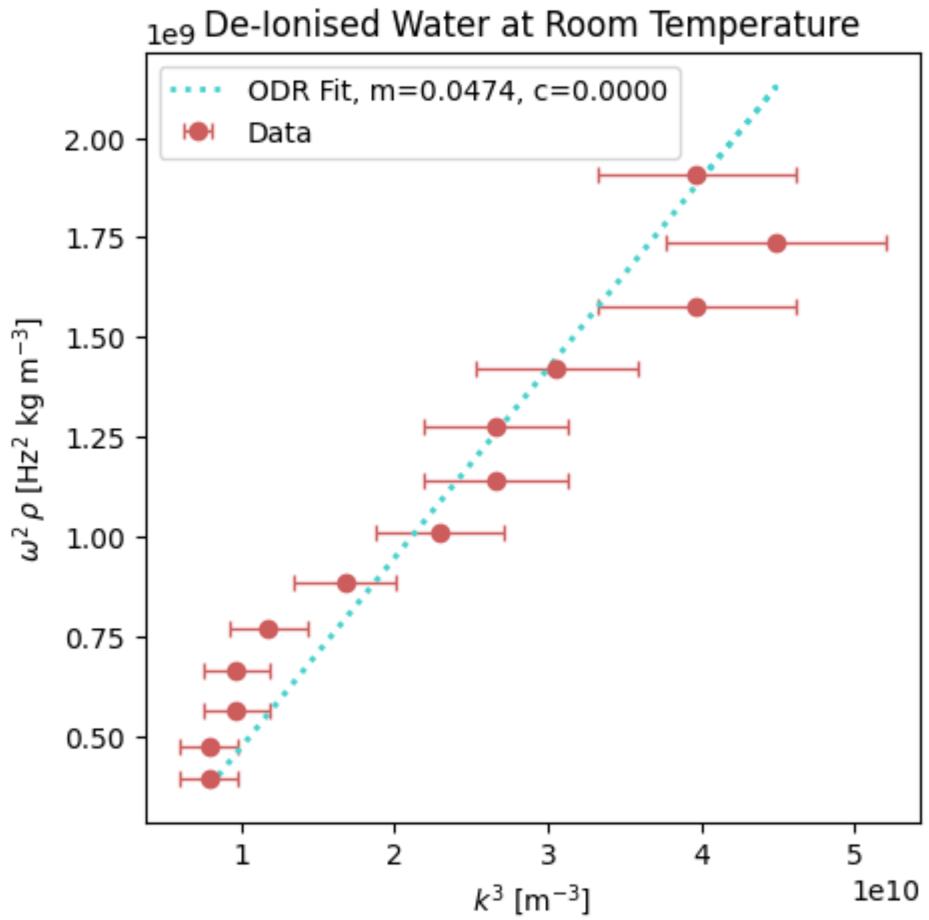
In [ ]:
```python
frequencies, spacings = LoadData("/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/water.txt")

diffractionAngles = GetDiffractionAngles(spacings)
waveNumbers = GetWaveNumbers(diffractionAngles)

angularFrequencies = GetAngularFrequencies(frequencies)

print(f"Diffraction Angles: {unp.nominal_values(diffractionAngles)}")
print(f"Wave Numbers: {unp.nominal_values(waveNumbers)}")
print(f"Angular Frequencies Product: {angularFrequencies**2 * waterDensity}")
```

```
Diffraction Angles: [0.00249999 0.00249999 0.00267857 0.00267857 0.00285714 0.00321427
 0.00357141 0.00374998 0.00374998 0.00392855 0.00428569 0.00446426
 0.00428569]
Wave Numbers: [1989.8521551  1989.8521551  2131.98346613 2131.98346613 2274.11457321
 2558.37612113 2842.63669011 2984.7665735  2984.7665735  3126.89617137
 3411.15445615 3553.28311587 3411.15445615]
Angular Frequencies Product: [3.94073565e+08 4.76829013e+08 5.67465933e+08 6.65984324e+08
 7.72384186e+08 8.86665520e+08 1.00882833e+09 1.13887260e+09
 1.27679835e+09 1.42260557e+09 1.57629426e+09 1.73786442e+09
 1.90731605e+09]
```

In [ ]:
```
def DetermineSurfaceTension(dataPath, density, plotTitle="", showPlot=False, verbose=False):

    # Load and process data
    frequencies, spacings = LoadData(dataPath)

    angularFrequencies = GetAngularFrequencies(frequencies)
    diffractionAngles = GetDiffractionAngles(spacings)

    waveNumbers = GetWaveNumbers(diffractionAngles)

    # Fitting and plotting
    xValues = unp.nominal_values(waveNumbers**3)
    xErr = unp.std_devs(waveNumbers**3)

    yValues = unp.nominal_values(density * angularFrequencies**2)
    yErr = unp.std_devs(density * angularFrequencies**2)

    regression = PerformODR(LinearFunc, xValues, yValues, xErr)

    if verbose: regression.pprint()

    if showPlot:
        fig, ax = plt.subplots(figsize=(5,5))

        ax.errorbar(xValues, yValues, xerr=xErr, fmt="o", color="indianred", capsize=3, linewidth=1, label="Data")

        xRange = np.linspace(np.min(xValues), np.max(xValues), 10)
        ax.plot(xValues, LinearFunc(regression.beta, xValues), color="mediumturquoise", lw=2, ls="dotted", label=f"ODR Fit, m={regression.beta[0]:.4f}, c={regression.beta[1]:.

        ax.set_xlabel("$k^3$ [m$^{-3}$]")
        ax.set_ylabel(r"$\omega^2 \,\rho$ [Hz$^2$ kg m$^{-3}$]")
        ax.legend()
        ax.set_title(plotTitle)

    return (regression.beta[0], regression.sd_beta[0])
```

In [ ]:
```
waterTension = DetermineSurfaceTension(rootPath + "/water.txt", waterDensity, plotTitle="De-Ionised Water at Room Temperature", showPlot=True)
print(waterTension)
```

```
(0.047393156096055974, 0.0021747090378043798)
```



De-Ionised Water at Room Temperature
1e9
ODR Fit, m=0.0474, c=0.0000
Data
2.00
1.75
1.50
1.25
1.00
0.75
0.50
$\omega^2\rho$ [Hz^2 kg m^{-3}]
1
2
3
4
5
1e10
k^3 [m^{-3}]

Soap

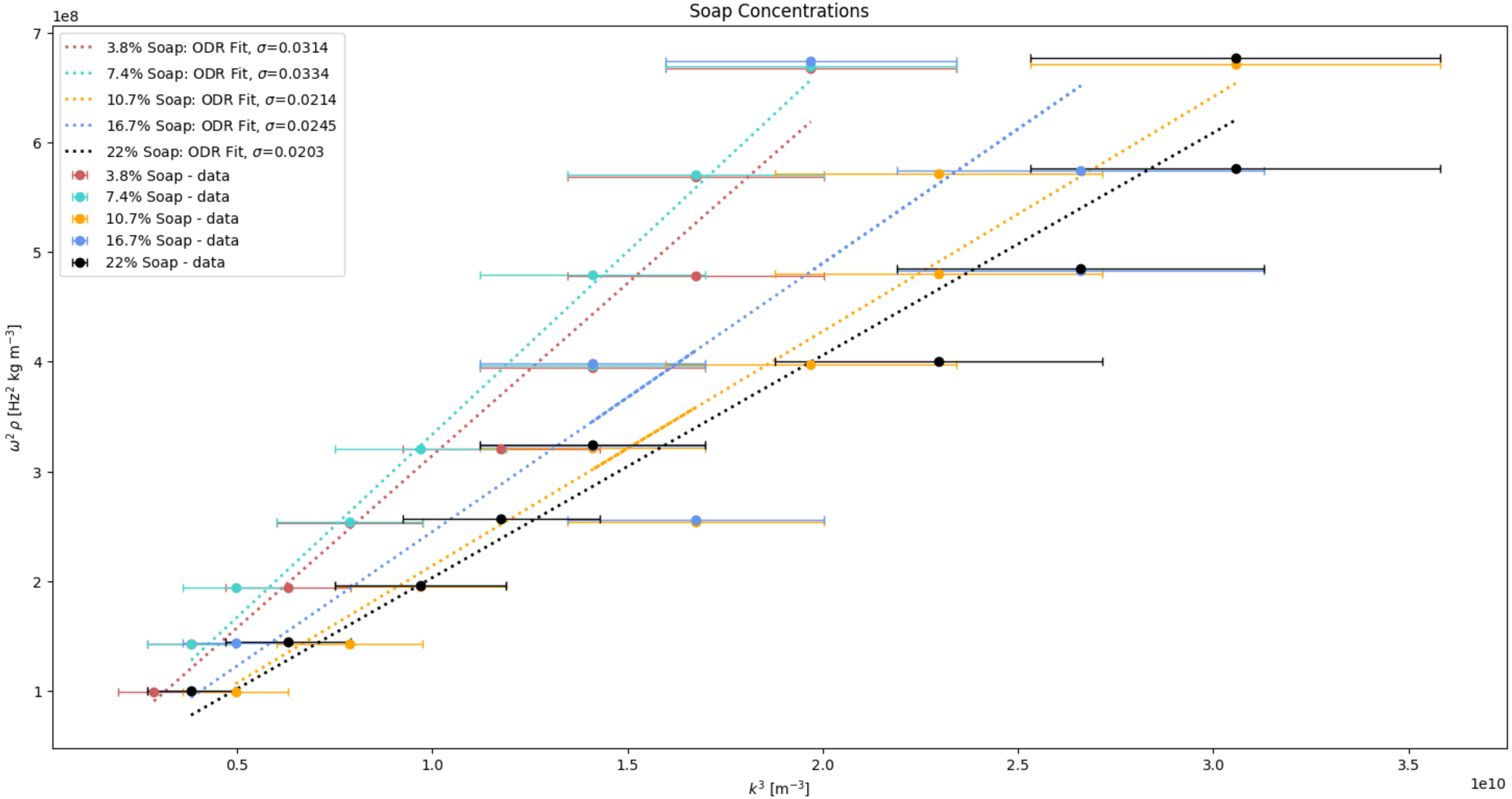
In [ ]:

```python
# Multiplot
def Multiplot(dataPath, densities, labels, colours, plotTitle, verbose=False):
    dataPaths = glob(dataPath)
    dataPaths.sort()
    print(dataPaths)

    dataList = []
    for dataPath in dataPaths:
        dataList.append(LoadData(dataPath))

    fig, ax = plt.subplots(figsize=(18,9))


    for data, density, colour, label in zip(dataList,densities, colours ,labels):
        # Load and process data
        frequencies, spacings = data

        angularFrequencies = GetAngularFrequencies(frequencies)
        diffractionAngles = GetDiffractionAngles(spacings)

        waveNumbers = GetWaveNumbers(diffractionAngles)

        # Fitting and plotting
        xValues = unp.nominal_values(waveNumbers**3)
        xErr = unp.std_devs(waveNumbers**3)

        yValues = unp.nominal_values(density * angularFrequencies**2)
        yErr = unp.std_devs(density * angularFrequencies**2)

        regression = PerformODR(LinearFunc, xValues, yValues, xErr)

        if verbose: regression.pprint()

        ax.errorbar(xValues, yValues, xerr=xErr, fmt="o", color=colour, capsize=3, linewidth=1, label=label + " - data")

        xRange = np.linspace(np.min(xValues), np.max(xValues), 10)
        ax.plot(xValues, LinearFunc(regression.beta, xValues), color=colour, lw=2, ls="dotted", label=label + f": ODR Fit, $\sigma$={regression.beta[0]:.4f}")

    ax.set_xlabel("$k^3$ [m$^{-3}$]")
    ax.set_ylabel(r"$\omega^2 \,\rho$ [Hz$^2$ kg m$^{-3}$]")
    ax.legend()
    ax.set_title(plotTitle)
```

In [ ]:

```python
soapRatios = np.array([0.038, 0.074, 0.107, 0.167, 0.22])
soapDensities = soapRatios * soapDensity + ([1]*5 - soapRatios) * waterDensity

colours = ["indianred", "mediumturquoise", "orange", "cornflowerblue", "black"]

Multiplot(rootPath + "/soap_250_*.txt", soapDensities, [r"3.8% Soap", r"7.4% Soap", r"10.7% Soap", r"16.7% Soap", r"22% Soap"], colours, "Soap Concentrations")
```

```
['/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/soap_250_10.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/soap_250_20.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/soap_250_30.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/soap_250_50.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/soap_250_70.txt']
```


Soap Concentrations
1e8
1e10
$\omega^2\,\rho$ [Hz^2 kg m^{-3}]
k^3 [m^{-3}]
3.8% Soap: ODR Fit, σ=0.0314
7.4% Soap: ODR Fit, σ=0.0334
10.7% Soap: ODR Fit, σ=0.0214
16.7% Soap: ODR Fit, σ=0.0245
22% Soap: ODR Fit, σ=0.0203
3.8% Soap - data
7.4% Soap - data
10.7% Soap - data
16.7% Soap - data
22% Soap - data

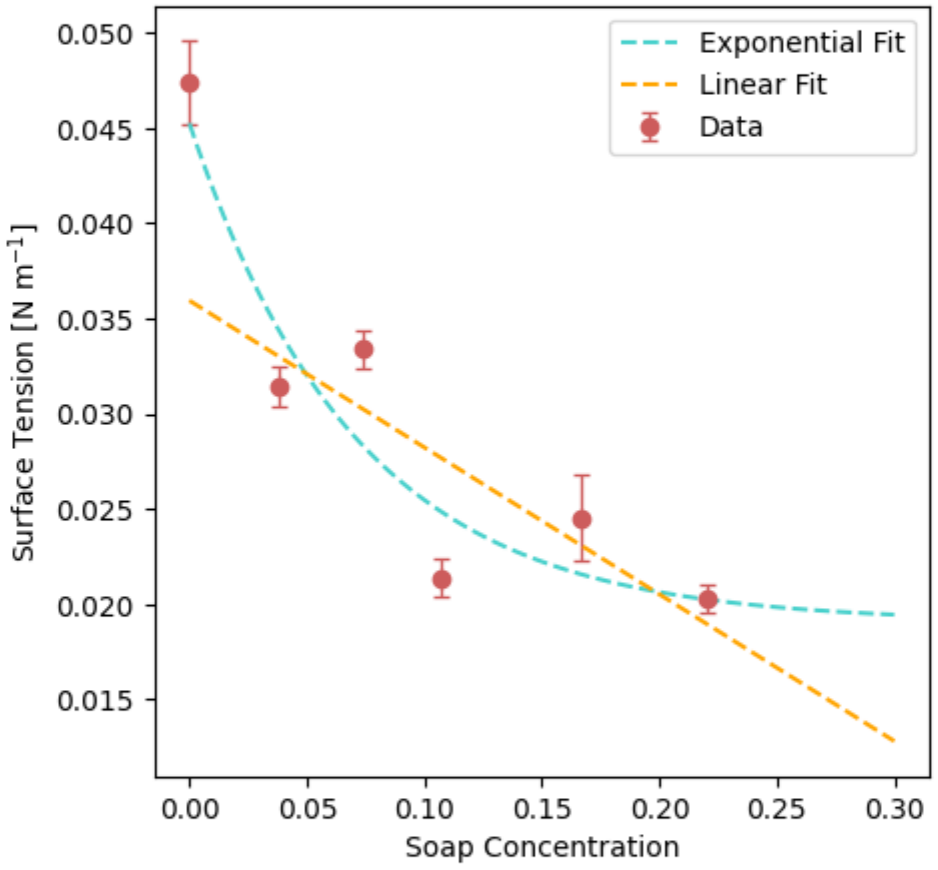
In [ ]:

```python
soapTension_250_10 = DetermineSurfaceTension(rootPath + "/soap_250_10.txt", soapDensities[0], plotTitle=r"0.04% soap at Room Temperature", showPlot=False)
soapTension_250_20 = DetermineSurfaceTension(rootPath + "/soap_250_20.txt", soapDensities[1], plotTitle=r"0.08% soap at Room Temperature", showPlot=False)
soapTension_250_30 = DetermineSurfaceTension(rootPath + "/soap_250_30.txt", soapDensities[2], plotTitle=r"0.12% soap at Room Temperature", showPlot=False)
soapTension_250_50 = DetermineSurfaceTension(rootPath + "/soap_250_50.txt", soapDensities[3], plotTitle=r"0.20% soap at Room Temperature", showPlot=False)
soapTension_250_70 = DetermineSurfaceTension(rootPath + "/soap_250_70.txt", soapDensities[4], plotTitle=r"0.28% soap at Room Temperature", showPlot=False)

soapRatios = np.array([0, 0.038, 0.074, 0.107, 0.167, 0.22])

soapTensions = np.array([waterTension, soapTension_250_10, soapTension_250_20, soapTension_250_30, soapTension_250_50, soapTension_250_70])

# FITTING
def ExpCurve(x, A, B, C):
    return A * np.exp(B * x) + C

def LinearCurve(x, m, c):
    return m * x + c

# Exponential fit
expPars, expCov = curve_fit(ExpCurve, soapRatios, soapTensions[:, 0], [0.05, -5, 0], sigma=soapTensions[:, 1])

# Linear fit
linearPars, linearCov = curve_fit(LinearCurve, soapRatios, soapTensions[:, 0], sigma=soapTensions[:, 1])

# PLOTTING

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

for ratio, tension in zip(soapRatios, soapTensions):
    if ratio == soapRatios[0]:
        label = "Data"
    else: label = ""

    ax.errorbar(ratio, tension[0], yerr=tension[1], fmt="o", color="indianred", capsize=3, linewidth=1, label=label)

xRange = np.linspace(0, 0.3, 100)
ax.plot(xRange, ExpCurve(xRange, expPars[0], expPars[1], expPars[2]), color="mediumturquoise", ls="--", label="Exponential Fit")
ax.plot(xRange, LinearCurve(xRange, linearPars[0], linearPars[1]), color="orange", ls="--", label="Linear Fit")

print(expPars)
print(np.sqrt(np.diag(expCov)))
print()
print(linearPars)
print(np.sqrt(np.diag(linearCov)))

ax.set_xlabel("Soap Concentration")
ax.set_ylabel("Surface Tension [N m$^{-1}$]")
ax.legend()
```

```
[  0.0261634  -14.07895824   0.01906944]
[7.61451096e-03 9.24500755e+00 4.81135435e-03]

[-0.07705898  0.03593315]
[0.02739957 0.00395944]
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7ff44538dd50>
```

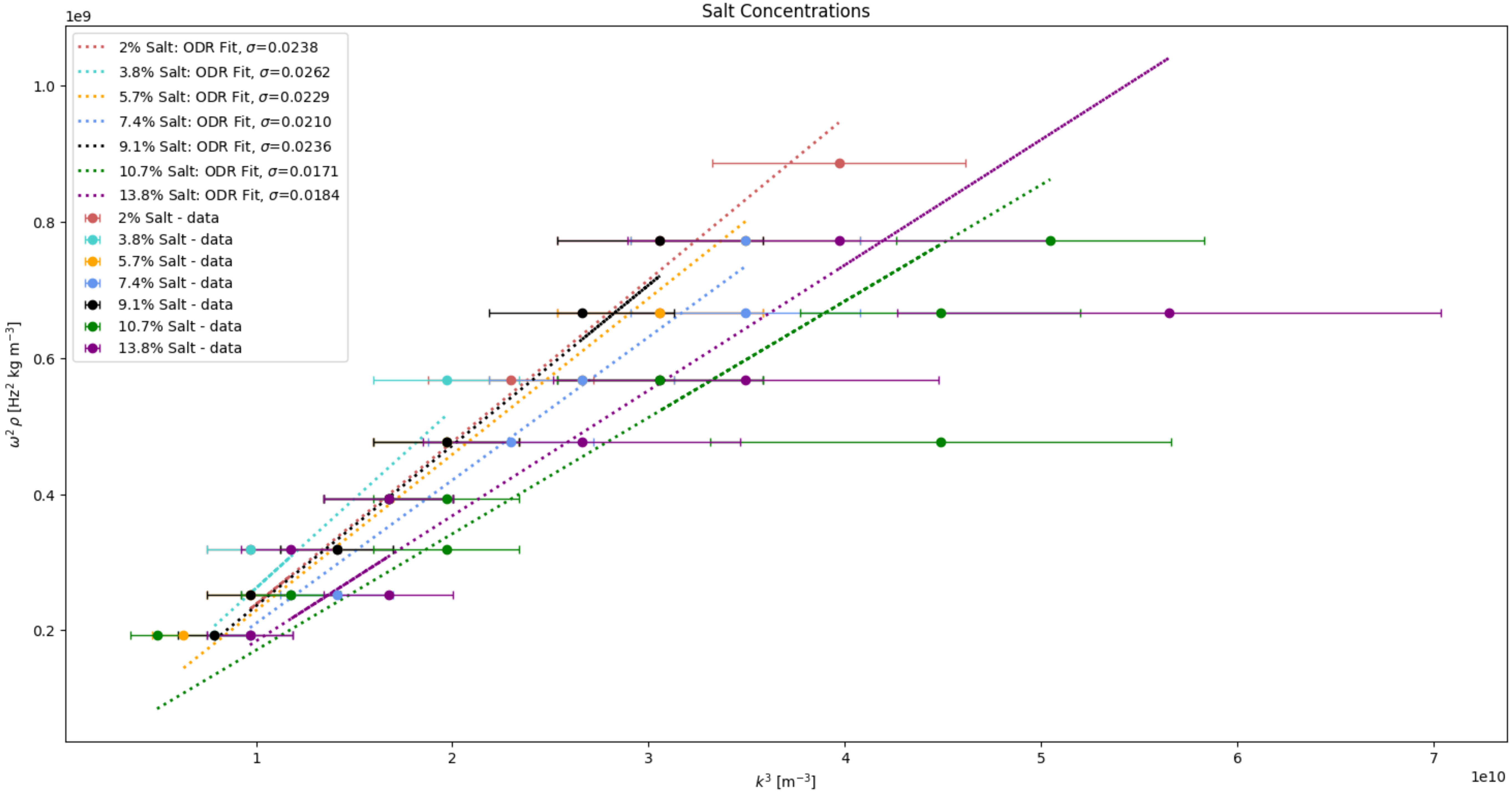


Salt

In [ ]:

```
saltMasses = [0.005, 0.010, 0.015, 0.020, 0.025, 0.030, 0.040]
saltDensities = [(0.250 * waterDensity + saltMass) / 0.250 for saltMass in saltMasses] # assuming volume is constant

colours = ["indianred", "mediumturquoise", "orange", "cornflowerblue", "black", "green", "purple"]

Multiplot(rootPath + "/salt_250_*.txt", saltDensities, [r"2% Salt", r"3.8% Salt", r"5.7% Salt", r"7.4% Salt", r"9.1% Salt", r"10.7% Salt", r"13.8% Salt"], colours, "Salt Conce
```

```
['/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_05.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_10.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_15.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_20.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_25.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_30.txt', '/home/daraghhollman/Main/ucd_4thYearLabs/surfaceTension/data/salt_250_40.txt']
```



Salt Concentrations
1e9
2% Salt: ODR Fit, σ=0.0238
3.8% Salt: ODR Fit, σ=0.0262
5.7% Salt: ODR Fit, σ=0.0229
7.4% Salt: ODR Fit, σ=0.0210
9.1% Salt: ODR Fit, σ=0.0236
10.7% Salt: ODR Fit, σ=0.0171
13.8% Salt: ODR Fit, σ=0.0184
2% Salt - data
3.8% Salt - data
5.7% Salt - data
7.4% Salt - data
9.1% Salt - data
10.7% Salt - data
13.8% Salt - data
1.0
0.8
0.6
0.4
0.2
$\omega^2 \rho$ [Hz^2 kg m^{-3}]
1
2
3
4
5
6
7
1e10
k^3 [m^{-3}]

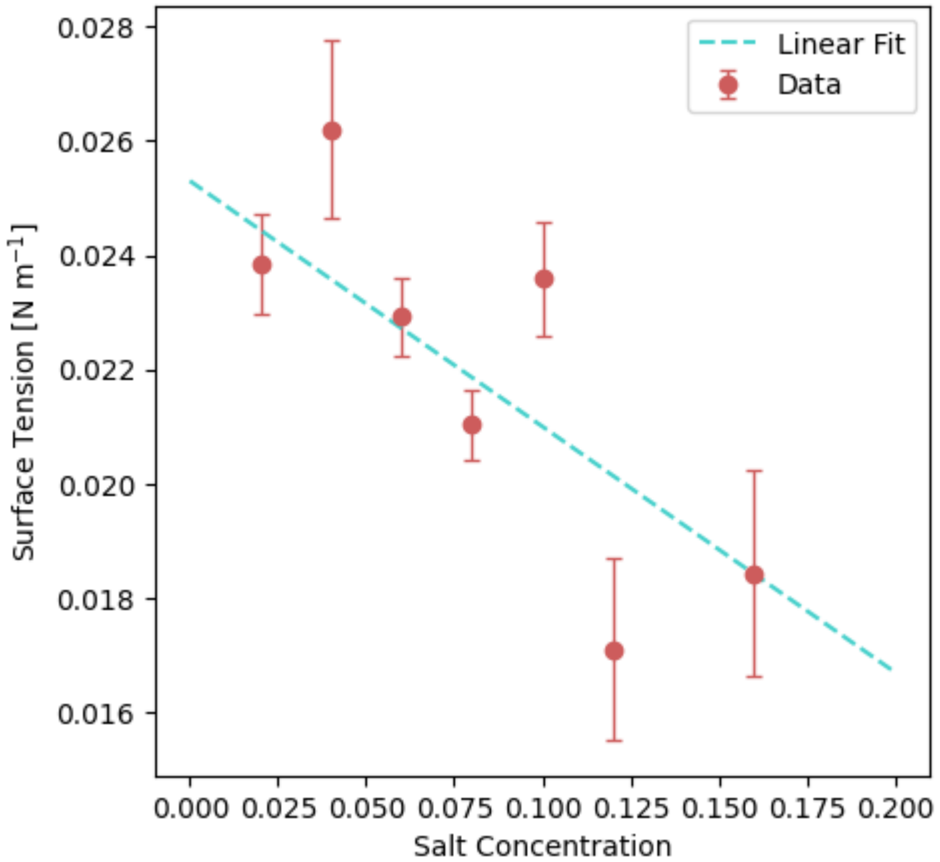
In [ ]:

```
saltTension_250_05 = DetermineSurfaceTension(rootPath + "/salt_250_05.txt", saltDensities[0], showPlot=False)
saltTension_250_10 = DetermineSurfaceTension(rootPath + "/salt_250_10.txt", saltDensities[1], showPlot=False)
saltTension_250_15 = DetermineSurfaceTension(rootPath + "/salt_250_15.txt", saltDensities[2], showPlot=False)
saltTension_250_20 = DetermineSurfaceTension(rootPath + "/salt_250_20.txt", saltDensities[3], showPlot=False)
saltTension_250_25 = DetermineSurfaceTension(rootPath + "/salt_250_25.txt", saltDensities[4], showPlot=False)
saltTension_250_30 = DetermineSurfaceTension(rootPath + "/salt_250_30.txt", saltDensities[5], showPlot=False)
saltTension_250_40 = DetermineSurfaceTension(rootPath + "/salt_250_40.txt", saltDensities[6], showPlot=False)

saltTensions = np.array([saltTension_250_05, saltTension_250_10, saltTension_250_15, saltTension_250_20, saltTension_250_25, saltTension_250_30, saltTension_250_40])

saltRatios = np.array([0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.16])

# FITTING
pars, cov = curve_fit(LinearCurve, saltRatios, saltTensions[:, 0], sigma=saltTensions[:, 1])

print(pars)
print(np.sqrt(np.diag(cov)))

# PLOTTING

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

for ratio, tension in zip(saltRatios, saltTensions):
    if ratio == saltRatios[0]:
        label = "Data"
    else: label = ""

    ax.errorbar(ratio, tension[0], yerr=tension[1], fmt="o", color="indianred", capsize=3, linewidth=1, label=label)

xRange = np.linspace(0, 0.2, 100)
ax.plot(xRange, LinearCurve(xRange, pars[0], pars[1]), color="mediumturquoise", ls="--", label="Linear Fit")

ax.set_xlabel("Salt Concentration")
ax.set_ylabel("Surface Tension [N m$^{-1}$]")
ax.legend()
```

```
[-0.04299809  0.02529889]
[0.01915801 0.00148575]
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7ff444d39990>
```



Linear Fit
Data
Surface Tension [N m^{-1}]
0.028
0.026
0.024
0.022
0.020
0.018
0.016
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175 0.200
Salt Concentration


```
In [ ]:
```