FYS2160 Lab 1: Gas Thermodynamics

23.09.2023

Tiril Sørum og Magnus Ausland

Abstract

The ideal gas is a idealised model for a gas' behaviour, which means it is based on theory on how a gas would act under ideal circumstances. This allows us to describe it by a couple simplified rules and laws - somethig which we can take advantage of. This report uses experimental data for the speed sound travelling thrugh a gas in a closed environment for showcasing this theory in the real world. This experiment was done by measuring the resonance frequencies of soundwaves in five different conditions, and thereby caclulating the speed of which the wave travelled. It then then models an ideal gas in a number of different conditions, and showcases these properties the gas possesses - amongst these heat conduction and compressability.

Introduction

We describe the ideal gas as consisting of N identical atoms in a "box" with a fixed volume, that are so far from each other that they don't interact with each other. This is an assumption we make to more easily describe the system. Just as for an ideal solid, we assume energy exchange between the atoms within the gas, but a constant conserved energy for the gas alltogether.

In this report, we wish to explore the model for the ideal gas, and how realistic it is. We do this by executing an experiment, as well as studying a simulation of a volume of gas over time in Lammps MD simulator. In the experiment we use Knudt's tube to measure the speed of sound in three physical gasses, and draw a parallell from this to the theory for heat capacity in a constant volume. The same concept is used when we measure compressibility and heat capacity in three model gasses in Lammps MD. By experimenting this way, we're able to test both our model and theory against real data. How well does the theory match with our experiment results?

Theory and Method

The speed of sound

The speed of sound depends on density of the material it travels through. This allows us to draw a correlation between this speed of sound, and the properties of an ideal gas. We can deduce the equatrion for the speed of sound *c* in an ideal gas ;

$$c_{id}(T) = \sqrt{\frac{(f+2)RT}{fM_{mol}}}$$

where M_{mol} is the molar mass, and *f* is the thermodynamic degrees of freedom within the system.

This theory can be used to compare real data for the speed of sound in a Knudt's tube, with what we would expect when the gas within the tube is modelled as ideal. By measuring the renocance frequencies for each tube and conductiong a linear regression, we can use the formula

$$\nu = \frac{c}{2L}n + b$$

to determine a value c for our situation.

Methods - Knudt's tube

We measure and note down about 12 resonance frequencies in K1-K4 in the frequency interval from 200Hz to 2kHz, and note down the readings in a table. We also note down the resistance within each of the systems, as well as the length of each tube.

The Ideal gas Equation of State

We know that the ideal gas can be described by the *ideal gas equation of state*;

$$pV = Nk_BT = n_{mol}RT$$

where p is the pressure in *Pa*, V is the volume in m^3 , N is the number of atoms, k_B is the Boltzmann constant, n is the amount of gas in *moles*, R is the gas constant in *J/kgK*, and T is the temperature in *K*. This equation can be used to correlate pressure and temperature of an ideal gas. If we know the value of all the constants from ddata for a spesific system, we can use the different equations to determine deviation from the expression;

$$Z = \frac{pV}{Nk_BT}$$

$$Z = P/(T\rho)$$

where ρ is the density and T is the temperature in Kelvin.

For an ideal gas this *Z* this value is expected to be 1. We can compare real data the same way and thereby determine if the model is accurate for the system.

Heat Capasity

We have different definitions for heat capacity, depending on whether our system has a constant volume or pressure. For both we have the expressions

$$C_V = \left(\frac{dU}{dT}\right)_V$$

for the constant volume, and

$$C_p = \left(\frac{dU}{dT}\right)_P + P\left(\frac{dV}{dT}\right)_P$$

for the constant pressure.

When modelling

We also know the more degres of freedom *f* molecules in the gas has, the more heat they can store. Furthermore, we have the expressions for the heat itself;

$$c_V = \frac{C_V}{n_{mol}} and c_p = \frac{C_p}{n_{mol}}$$

Methods - Molecular dynamics simulations

We run simulations on the Lennard-Jones system, where we assume a constant volume og our gas in each instance. This means we will use the formula

$$C_V = \left(\frac{dU}{dT}\right)_V$$

to describe the heat capacity.

The first simulation we run uses a Lennard-Jones system for a density "rho equals 0.001" and is ran over a number of 100000 iterations. The data is imported into python where we perform a linear regression for temperature *T* against the total energy *U*, where we have from the formula for C_V that C_V is the slope of the curve. The unceirtanty can be derived from the python expression using "std_err", which we print into the terminal. To estimate the number degrees of freedom we use $C_V = f/2$, and we have the python programme print f as a function of slope in the terminal. Furthermore, we wish to compare our simulation to the theory for an ideal gas, we plot the value of $Z = P/(T\rho)$ for each point.

We perform the same test as above, but now we adjust the initial temperature to 0.7, the eFlux variable equal to 10, and double the number of runs. We import the data into python and perform a similar analysis. We remember to include the linear regression for C_V, as well as the plot for the estimated Z-value.

We re-use the configuration for our first simulation, but this time we want to test it for increasing densities. First, we set the temperature equal to 1.4. We then run the program for 10 different densitieSs, all smaller than 0.85. Each of the files for the simulations are saved in a directory with the rho used noted in the filename of our in-file. We build out-files for each instance, and import these into python. The data is then used to compute both *Z* and C_V in each instance, and then plot these computations as functions of the density.

We download the file *heatcapljdiat.in*, and modify it to cover a wide enough temperature range. This we do by changing the eFLux to 10 and the run in 200000. Then we run the files to simulate the bonds between the atoms as a rigid rod and as a spring. Then, we compute the value of f.

Results

All python programmes that result in the plots shown in this part of the rapport, will be available at the end of the rapport in Appendix A for the reason of tidiness.

Knudt's Tube

We took notes of the different resonance frequencies for the different tubes K1-K4 in intervals 200Hz-2kHz. The data we compiled is presented underneath;

K1 (Argon at $T = T_{room}$) :

268Hz, 389Hz, 521Hz, 651Hz, 776Hz, 906Hz, 1.034kHz, 1.165kHz, 1.293kHz, 1.422kHz, 1.552kHz, 1.68kHz

Resistance = 10759 Ω

Lenght of tube = 1.243m

K1 (CO2 at $T = T_{room}$) :

223Hz, 328Hz,437Hz, 544Hz, 652Hz, 758Hz, 865Hz, 973Hz, 1.082kHz, 1.189kHz, 1.297kHz, 1.406kHz

Resistance = 10759 Ω

Lenght of tube = 1.243m

K2 (air at $T = T_{room}$) :

250Hz, 280Hz, 408Hz, 544Hz, 676Hz, 810Hz, 943Hz, 1.076kHz, 1.21kHz, 1.346kHz, 1.48kHz

Resistance = 10759 Ω

Lenght of tube = 1.243m

K3 (air at T=70 degrees degrees Celcius) :

145Hz, 312Hz, 448Hz, 591Hz, 738Hz, 882Hz, 1.028kHz, 1.173kHz, 1.321kHz, 1.466kHz, 1.612kHz

Resistance = 39963 Ω

Lenght of tube = 1.244m

K4 (air at T=50 degrees Celcius) :

308Hz, 372Hz, 457Hz, 606Hz, 756Hz, 906Hz, 1.055kHz, 1.207kHz, 1.358kHz, 1.509kHz, 1.66kHz

Resistance = 18505 Ω

Lenght of tube = 1.244m

We determine the unceirtanty in each frequency measurement by line regression, and extracting the *standard error* from this plot. The value of *f*, the thermodynamic degrees of freedom, is also computed by the program. We do this for each tube by running a simple python program, and get printed in the terminal;

K1 Argon :

Slope (Coefficient): 117.9839743589744

Standard Deviation of Residuals (Error): 0.1623812130094884

C is calculated to be 293.30816025641036

We estimate f to be 4.934886667696854

K1 CO2 :

Slope (Coefficient): 98.58333333333337

Standard Deviation of Residuals (Error): 0.08439494725519397

C is calculated to be 245.0781666666668

We estimate f to be 24.74303462189949

K2 :

Slope (Coefficient): 120.15818181818183

Standard Deviation of Residuals (Error): 0.18609515028214904

C is calculated to be 298.9535563636364

We estimate f to be 34.22762766159537

K3 :

Slope (Coefficient): 132.42975206611567

Standard Deviation of Residuals (Error): 0.44611090865045405

C is calculated to be 329.4852231404958

We estimate f to be 11.49958514478697

K4 :

Slope (Coefficient): 132.0981818181818

Standard Deviation of Residuals (Error): 1.9036363636365186

C is calculated to be 328.66027636363634

We estimate f to be 20.04584137647174

All plots for the experimental part of the lab is given in the Appendix

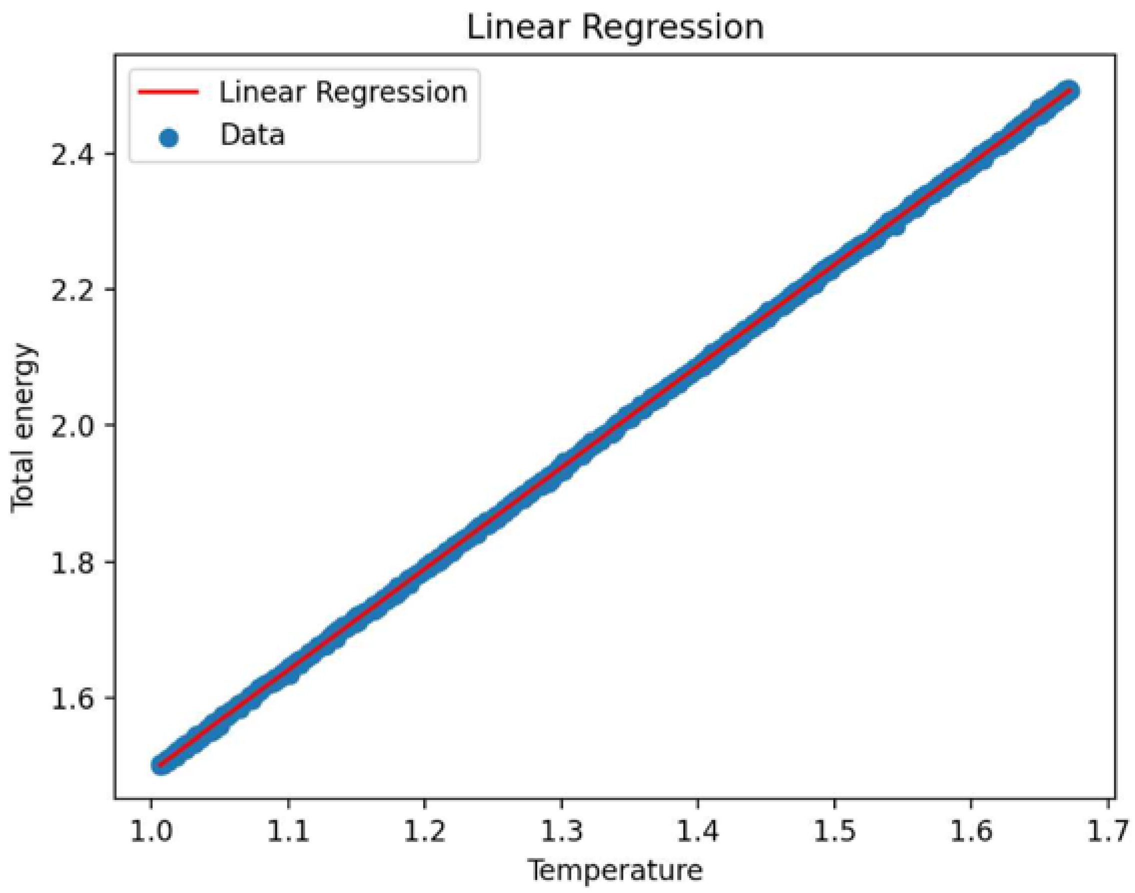
---

Molecular Dynamics Simulations

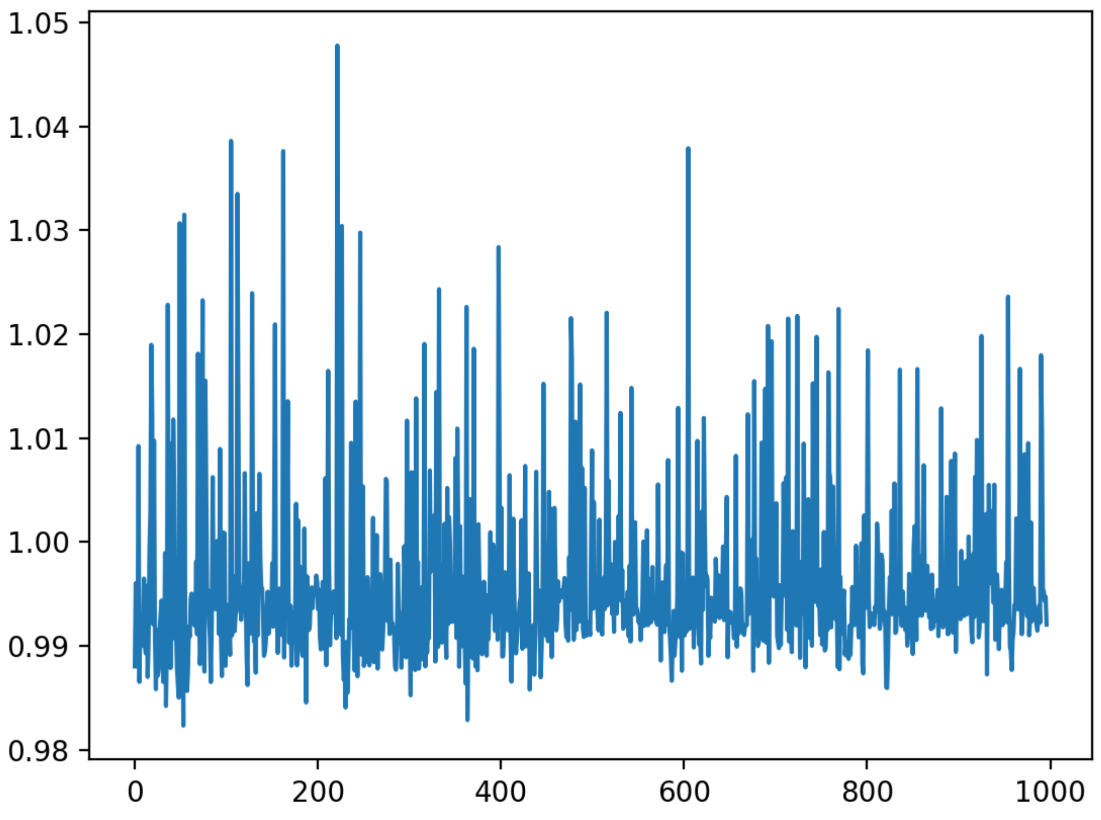
Simulation 1

In this part, we will run a Lennard-Jones system to compute the CV and the compressibility Z

We run our python program for the first simulation, and get the plot for our linear regression;



For our estimation of Z we get the plot:



In the terminal, we get printed:

$$Z = 0.9958061$$

$$Slope(C_v): 1.490424291735892$$

$$Intercept : 0.0011631166501062662$$

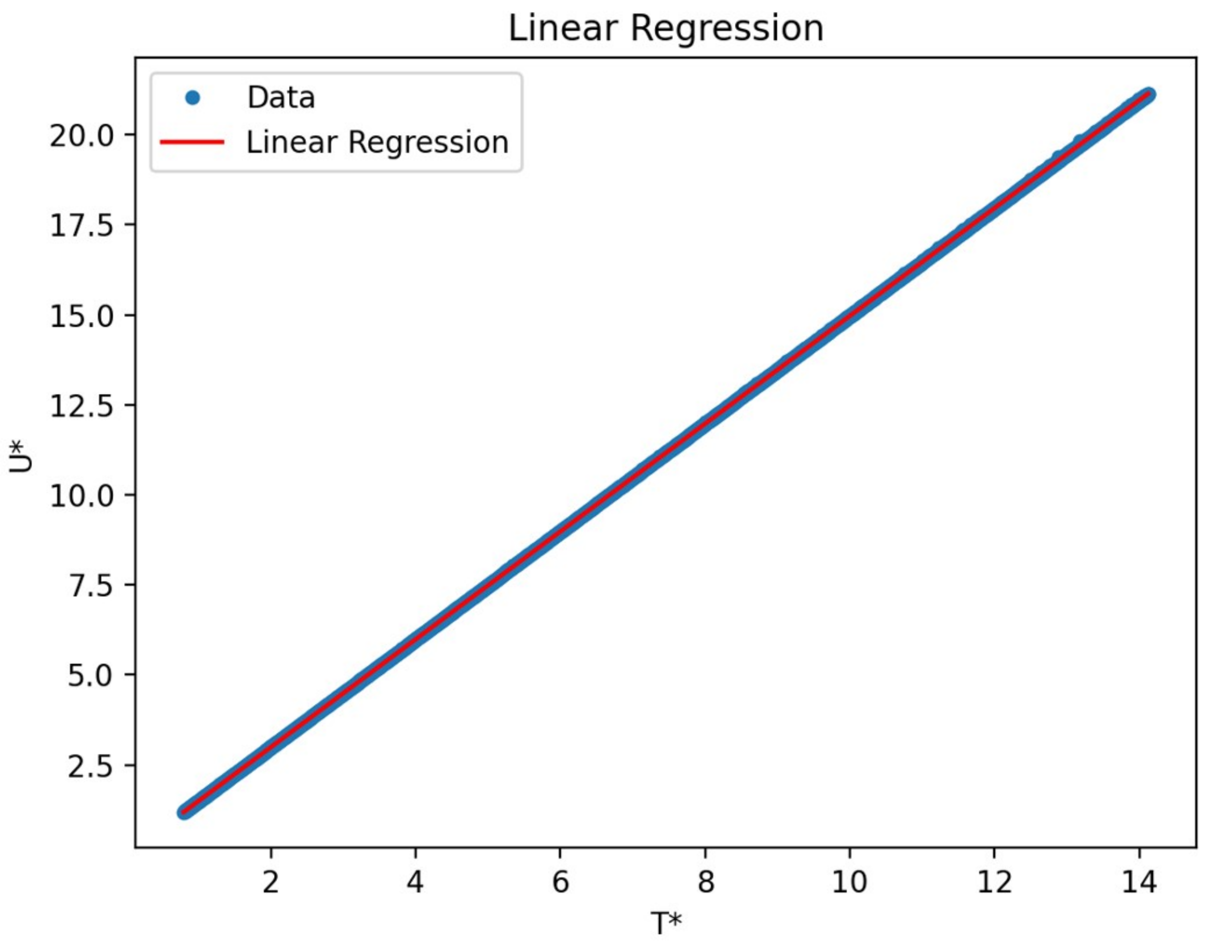
$$StandardDeviationofResiduals(Error) : 0.0025896894481886325$$

$$Degreesoffreedom(2 * C_v)2.980848583471784$$

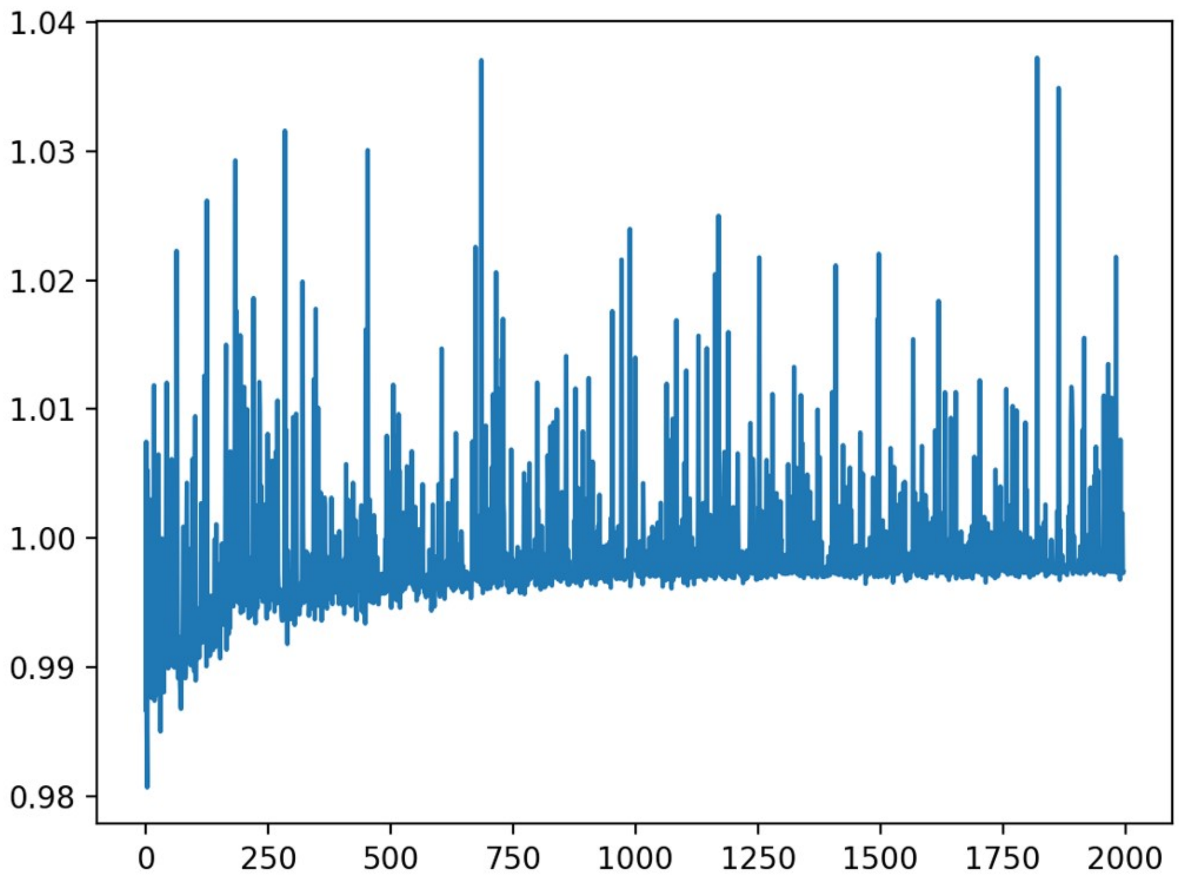
Simulation 2

In this part we are testing different temperatures for the simulation to see if the compressebility or slope changes.

We open and run our python program for the second simulation, and get the plot for our linear regression;



We see that we get a good linear regerssion for all our data points, with a way larger temperature than we did for simulation 1. For our estimation of Z we get the plot:



We see that the computed values for Z for each "step" does vary a bit in the intervall around *Z=1*, but there is a clear average, which we get confirmed when reading the value for Z printed in the terminal. In the terminal, we get printed:

$$Z = 0.9985224798793862$$

$$Slope(C_v): 1.4974198140697907$$

$$Intercept: -0.007054850092769627$$

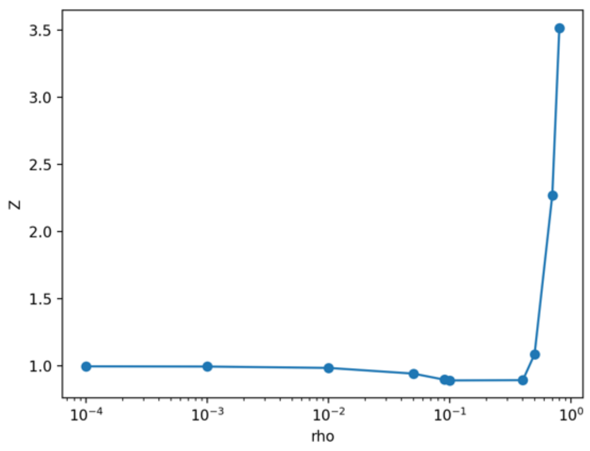
$$StandardDeviationofResiduals(Error): 0.006811761455904569$$

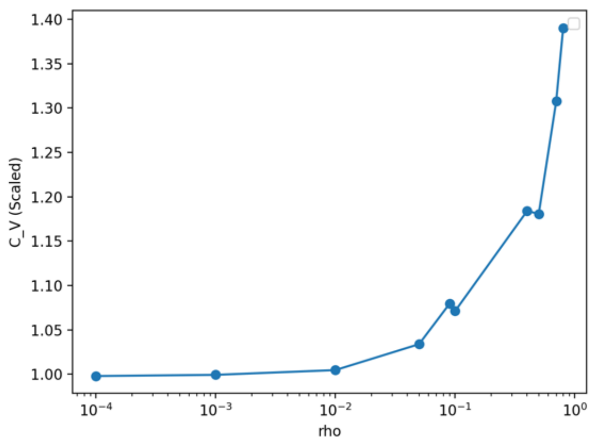
$$Degreesoffreedom(2 * C_v)2.9948396281395815$$

Simulation 3

In this part we are changing the initial density to see how this affects our slope

We used the data from our output log-files to compute C_V and Z as a function of ρ. The data is used to make python plots;





We scaled the axis for C_V to easily compare with what one would expect from an ideal gas. The code that plots these plots is given in Appendix A at the end of the report.

Simulation 4

In this simulation we are compering a rigid to a spring like bond between the nitrogen atoms to see if our degrees of fredom changes

We import both our output log-files into a python document, and use it to compute the thermodynamic degrees of freedom *f*. Degrees of freedom for a rigid rod is then computed $f = 2.7335438213303895$ and degrees of freedom (2*C_v) for a spring $f = 4.116598329080973$

Discussion

Experiment

During this entire experiment we have acted upon the thought of each gas in our systems being ideal gasses. How accurate was this assumption? To answer this question we compare the values we computed for each *c* with what is expected.

For Argon at room temperature we computed a value c: 293.3 $+-0.162$ m/s While we expected the value: 323 m/s

For CO2 at room temperature we computed a value c: 245.078 $+-0.0844$ m/s While we expected the value: 267 m/s

For air at room temperature we computed a value c: 298.95 $+-0.186$ m/s While we expected the value: 346 m/s

For air at 50 degrees celcius we computed a value c: 329.48 $+-0.446$ m/s While we expected the value: 360 m/s

For air at 70 degrees celcius we computed a value c: 328.6 $+-1.9036$ m/s While we expected the value: 371 m/s

In our attempts of checking the validity of our assumptions, we computed a value of f from our formula

$$c_{id} = \sqrt{\frac{(f+2)RT}{fM_{mol}}}$$

The program that we wrote printed these values into our terminal, and by comparing them to what one would expect for the ceirtan gasses, we can tell that something must've either gone completly wrong during our computation, or neither of our gasses act like an ideal gas would.

For Argon at room temperature we computed a value f: 4.93 While we expected the value: 3

For CO2 at room temperature we computed a value f: 24 While we expected the value: 5

For air at room temperature we computed a value f: 34 While we expected the value: 7

For air at 50 degrees celcius we computed a value f: 11 While we expected the value: 7

For air at 70 degrees celcius we computed a value f: 20 While we expected the value: 7

We see, due to the low error in our c-estimations that this is pretty accurate. Compared to what we expacted the value of c to be, we can say that the gasses do indeed act a lot like ideal gasses. On the other hand, our computation of f for each of the instances are pretty unrealistic. This could be due to an issue with the model, that stems from our assumption of ideal gas.

Molecular Dynamics Simulations

In our simulation 1, we can read from the accuracy of our linear regression, as well as the low standard error 0.00259 printed in the terminal, that we get a good model. This means that the approximation to U/T is linear, which is also expected. In this simulation, we get a Z value that is almost *Z=1*, which is what one would expect for an ideal gas. The degrees of freedom calculated, *f=2.98*, is almost accuratley *f=3*, which is also what one would expect for an ideal gas.

Simulation 2 does also have a good level of accuracy when it comes to reflecting the ideal gas' properties. The Z-value is almose exactly 1, we get a computed number for degrees of freedom 2.99 $\approx$ 3, and the linear regression itself has a small standard error 0.0068, with a slope C_V=1.4974, which is the exact same value we did get in simulation as well. This means that when the volume of gas is constant, a change in temperature will not result in a change of heat-capacity, which is exactly what we expect since the heat capacity is described by an expression that is constant for constant volume for an ideal gas.

Simulation 3 gives us data, that we use to visualize how heat-capacity and *Z* would change as a function of the density of the gas. We see that we get a Z-value that is approximatley constant *Z=1*, ut until the moment the density has a value located in the intervall [0.8, 1]. We can explain this easily by looking at Figure 6 in the assignment text; when the density becomes this large we're simply no longer talking about a gas, but a liquid or solid. The plot C_V against ρ show how the heat capacity does increase exponentially for an increasing density. This does make sense for an ideal gas.

Simulation 4 gives us data for temperature and total energy within our two different system, in which we use the same approach as earlier, and use the formula f=2* slope to determine the degrees of freedom for both the types of bond between the nitrogen atoms. One would expect tree degrees of freedom for the rigid rod model, and a total of four for the spring model. We compare these expected values to the ones we obtained, f=2.733 and f=4.11, and notice that these computations within the right ballpark, but they are not exact.

Conclusion

We can see through all our experiments that our expected outcomes are not so far of the actual values. Most of the deviations we had might come from several different reason. For example in the experimentall part of the lab we can narrow the deviations down to sloppy measurements, the fact that the gasses are not ideal, and other outside factors. While this might be true, we can see that theese not ideal working conditions still yield pretty accurate results.

We ran a number of simulations by using Lammps, wherof they all illustrated different important characteristics for the ideal gas model. By doing this, both us who performed the experiments, and the reader of this report, gained a better overview over what an ideal gas means, how the set of rules describing the gas act, and how to apply them to the model.

Though fun and informative we found that keeping track of all assumptions such as temperature, density, constant pressures, was challenging, but in the end we are left with a better understanding of the subject all together.

Appendix

Python Program Knudt's Tube:

In [2]:

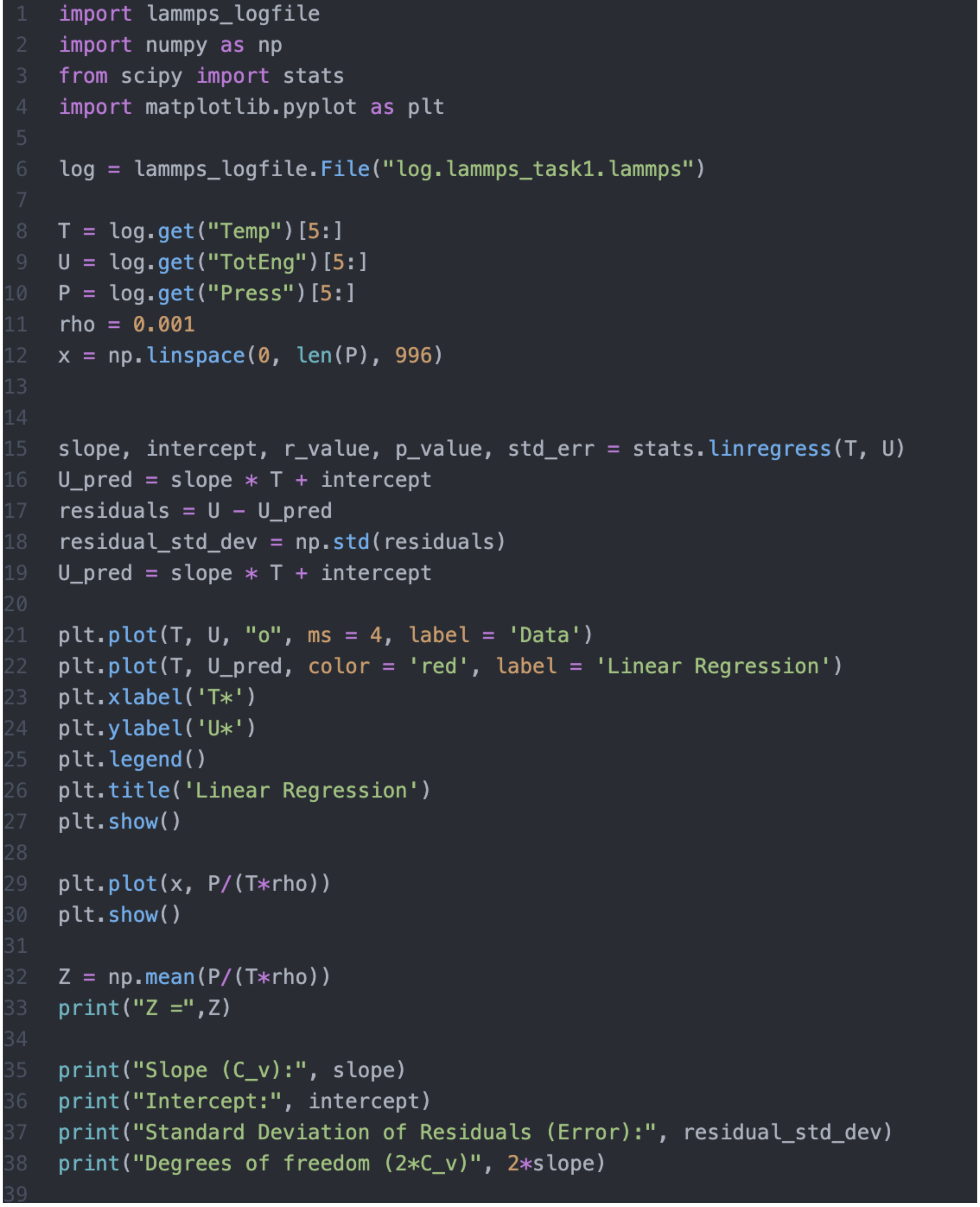
```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
K1_Argon=[268, 389, 521, 651, 776, 906, 1034, 1165, 1293, 1422, 1552, 1680]
K1_CO2=[223, 328,437, 544, 652, 758, 865, 973, 1082, 1189, 1297, 1406]; L1=1.243
K2=[280, 408, 544, 676, 810, 943, 1076, 1210, 1346, 1480]
K3=[145,312,448, 591, 738, 882, 1028, 1173, 1321, 1466, 1612]; L3=1.244
K4=[372, 457, 606, 756, 906, 1055, 1207, 1358, 1509, 1660]; L4=1.244
r2=10750 #ohm for everything else
r1=39963 #ohm for 70 degrees
r3=18505 #for 50 degrees
T1=272.15+22; T3=272.15+50; T4=272.15+70
Air=28.964/1000; Argon=39.95/1000; CO2=44.01/1000


def find_C(liste,length, listename,T,Mmol):
    x=np.linspace(0,len(liste),len(liste))
    slope, intercept, r_value, p_value, std_err = stats.linregress(x, np.array(liste)
    c=slope*2*length
    R=8.3144598
    f=2/((c**2*Mmol)/(R*T)-1)
    print(listename, ":")
    print("Slope (Coefficient):", slope)
    print("Standard Deviation of Residuals (Error):", std_err)
    print("C is calculated to be", c)
    print("We estimate f to be", f)
    return c, std_err

find_C(K1_Argon,L1, "K1 Argon",T1,Argon)
find_C(K1_CO2,L1, "K1 CO2", T1,CO2)
find_C(K2,L3, "K2", T1,Air)
find_C(K3,L3, "K3", T3,Air)
find_C(K4,L4, "K4", T4,Air)
```

```
K1 Argon :
Slope (Coefficient): 117.9839743589744
Standard Deviation of Residuals (Error): 0.1623812130094884
C is calculated to be 293.30816025641036
We estimate f to be 4.934886667696854
K1 CO2 :
Slope (Coefficient): 98.58333333333337
Standard Deviation of Residuals (Error): 0.08439494725519397
C is calculated to be 245.0781666666668
We estimate f to be 24.74303462189949
K2 :
Slope (Coefficient): 120.15818181818183
Standard Deviation of Residuals (Error): 0.18609515028214904
C is calculated to be 298.9535563636364
We estimate f to be 34.22762766159537
K3 :
Slope (Coefficient): 132.42975206611567
Standard Deviation of Residuals (Error): 0.44611090865045405
C is calculated to be 329.4852231404958
We estimate f to be 11.49958514478697
K4 :
Slope (Coefficient): 132.0981818181818
Standard Deviation of Residuals (Error): 1.9036363636365186
C is calculated to be 328.66027636363634
We estimate f to be 20.04584137647174
```

Out[2]:
```
(328.66027636363634, 1.9036363636365186)
```

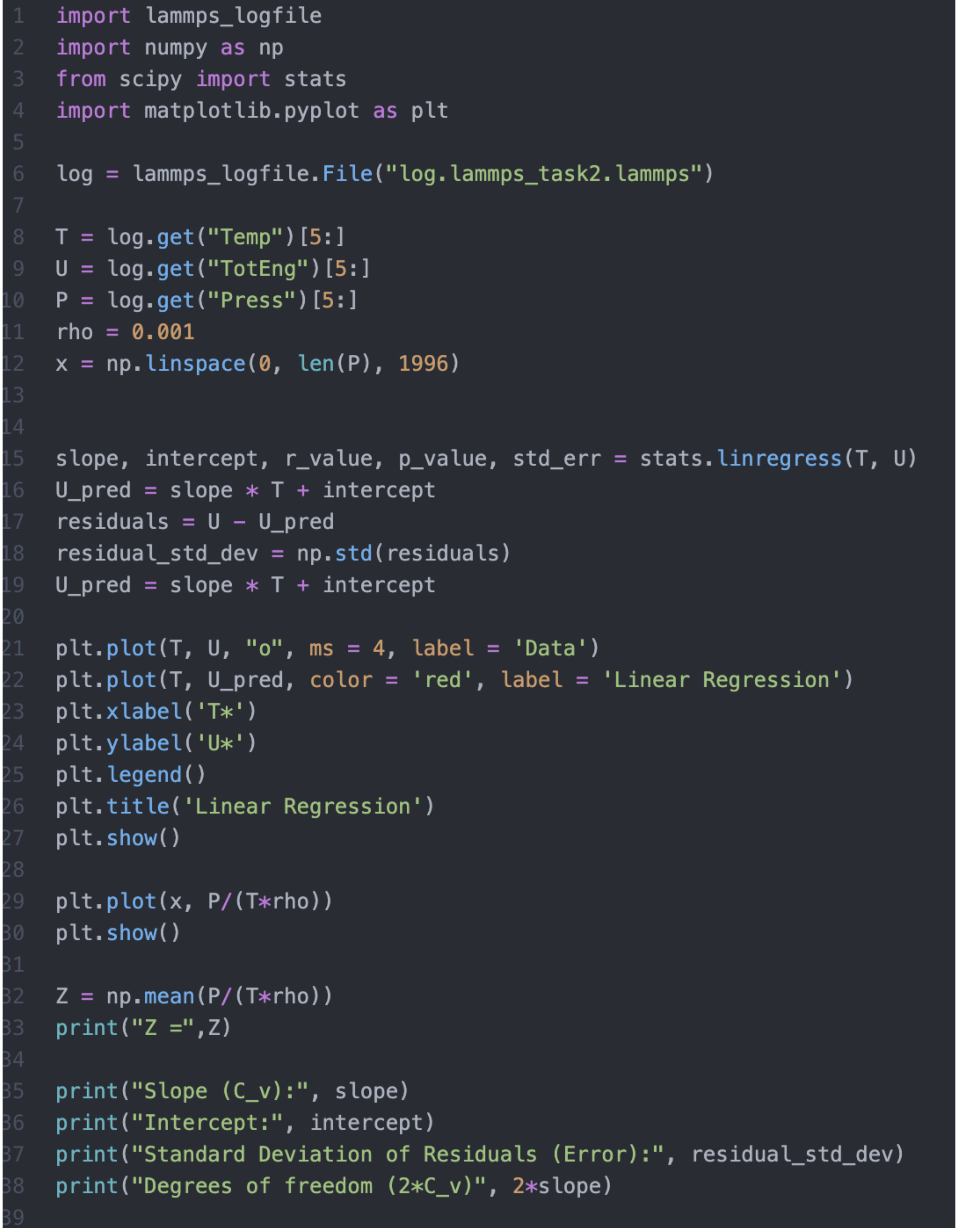
In [3]:
```
#koden for Z/rho og C_V/rho plottene
```

Underneath is the code used to perform linear regression for simulation 1:



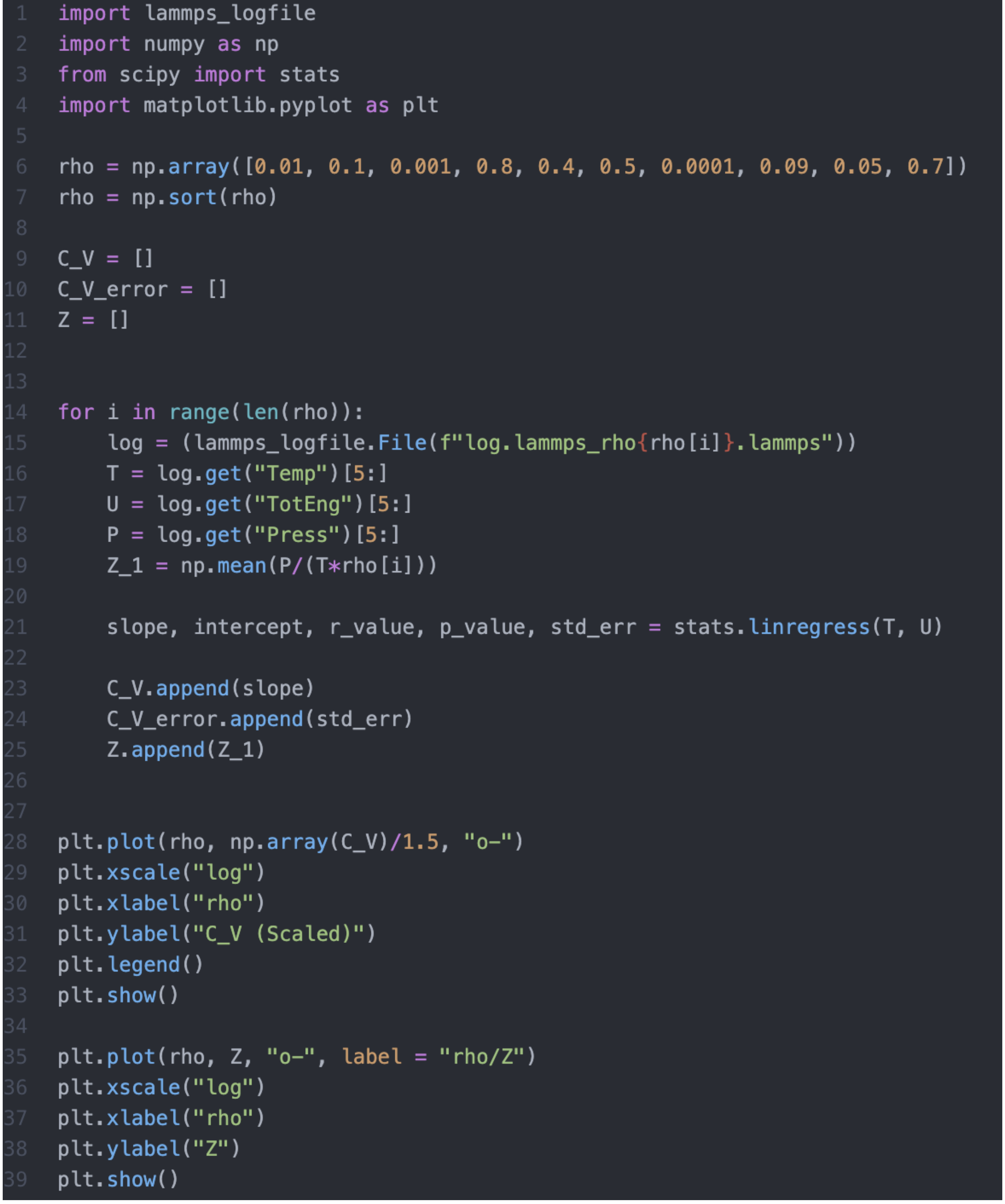
```python
import lammps_logfile
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

log = lammps_logfile.File("log.lammps_task1.lammps")

T = log.get("Temp")[5:]
U = log.get("TotEng")[5:]
P = log.get("Press")[5:]
rho = 0.001
x = np.linspace(0, len(P), 996)


slope, intercept, r_value, p_value, std_err = stats.linregress(T, U)
U_pred = slope * T + intercept
residuals = U - U_pred
residual_std_dev = np.std(residuals)
U_pred = slope * T + intercept

plt.plot(T, U, "o", ms = 4, label = 'Data')
plt.plot(T, U_pred, color = 'red', label = 'Linear Regression')
plt.xlabel('T*')
plt.ylabel('U*')
plt.legend()
plt.title('Linear Regression')
plt.show()

plt.plot(x, P/(T*rho))
plt.show()

Z = np.mean(P/(T*rho))
print("Z =",Z)

print("Slope (C_v):", slope)
print("Intercept:", intercept)
print("Standard Deviation of Residuals (Error):", residual_std_dev)
print("Degrees of freedom (2*C_v)", 2*slope)

```

Underneath is the code used to perform the linear regressions in simulation 2:



```python
import lammps_logfile
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

log = lammps_logfile.File("log.lammps_task2.lammps")

T = log.get("Temp")[5:]
U = log.get("TotEng")[5:]
P = log.get("Press")[5:]
rho = 0.001
x = np.linspace(0, len(P), 1996)


slope, intercept, r_value, p_value, std_err = stats.linregress(T, U)
U_pred = slope * T + intercept
residuals = U - U_pred
residual_std_dev = np.std(residuals)
U_pred = slope * T + intercept

plt.plot(T, U, "o", ms = 4, label = 'Data')
plt.plot(T, U_pred, color = 'red', label = 'Linear Regression')
plt.xlabel('T*')
plt.ylabel('U*')
plt.legend()
plt.title('Linear Regression')
plt.show()

plt.plot(x, P/(T*rho))
plt.show()

Z = np.mean(P/(T*rho))
print("Z =",Z)

print("Slope (C_v):", slope)
print("Intercept:", intercept)
print("Standard Deviation of Residuals (Error):", residual_std_dev)
print("Degrees of freedom (2*C_v)", 2*slope)

```

Underneath is the code used to visualize data and compute relevant values in simulation 3:

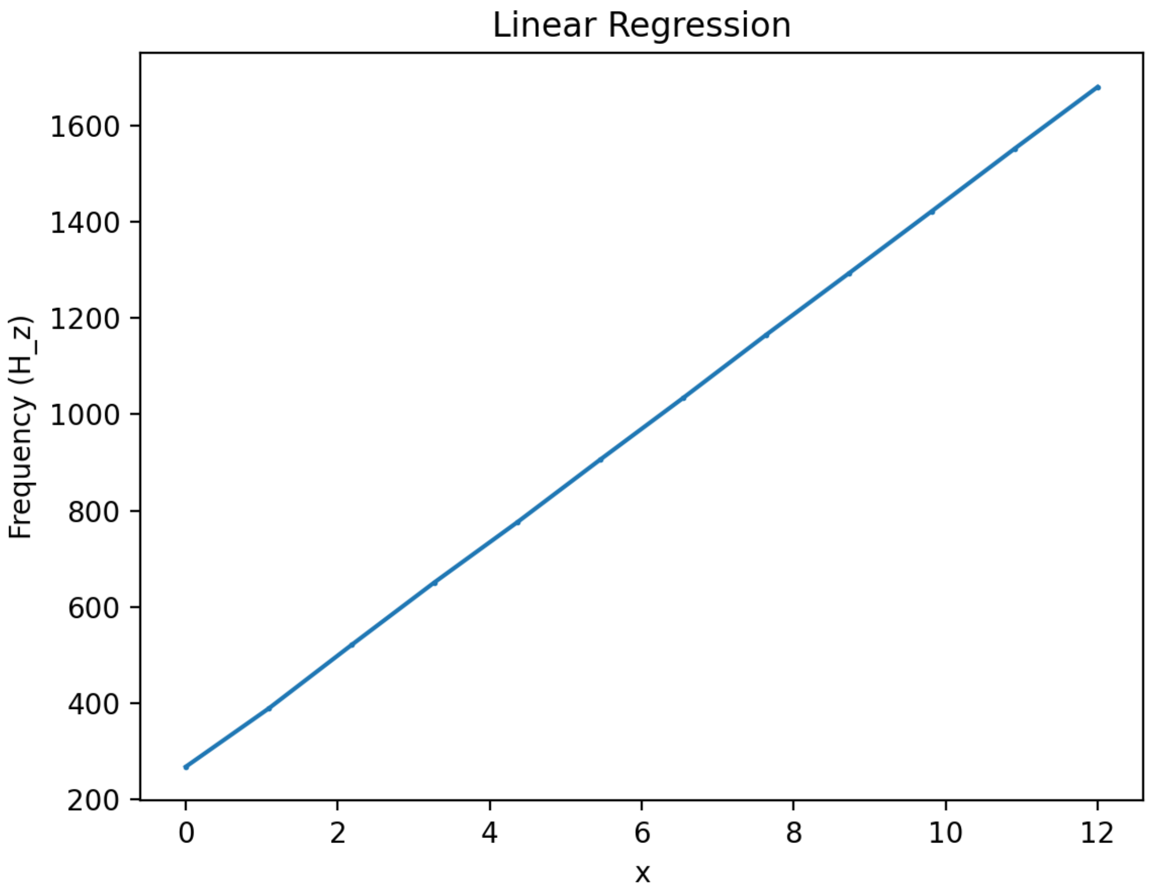


```
import lammps_logfile
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

rho = np.array([0.01, 0.1, 0.001, 0.8, 0.4, 0.5, 0.0001, 0.09, 0.05, 0.7])
rho = np.sort(rho)

C_V = []
C_V_error = []
Z = []


for i in range(len(rho)):
    log = (lammps_logfile.File(f"log.lammps_rho{rho[i]}.lammps"))
    T = log.get("Temp")[5:]
    U = log.get("TotEng")[5:]
    P = log.get("Press")[5:]
    Z_1 = np.mean(P/(T*rho[i]))

    slope, intercept, r_value, p_value, std_err = stats.linregress(T, U)

    C_V.append(slope)
    C_V_error.append(std_err)
    Z.append(Z_1)


plt.plot(rho, np.array(C_V)/1.5, "o-")
plt.xscale("log")
plt.xlabel("rho")
plt.ylabel("C_V (Scaled)")
plt.legend()
plt.show()

plt.plot(rho, Z, "o-", label = "rho/Z")
plt.xscale("log")
plt.xlabel("rho")
plt.ylabel("Z")
plt.show()
```

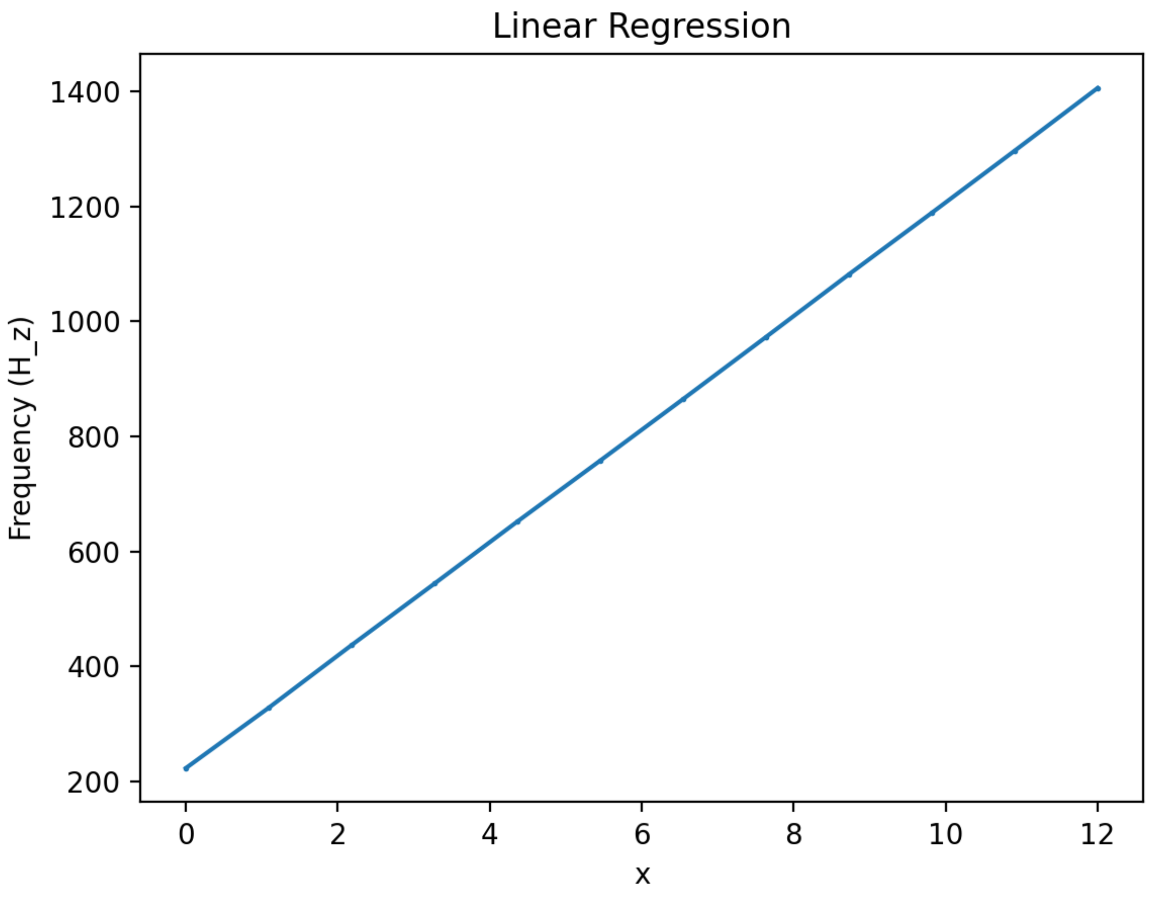
Underneath is the code used to visualize data and compute relevant values in simulation 4:

```python
import lammps_logfile
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

log = lammps_logfile.File("heatcapljdiat_rod.lammps")
log2 = lammps_logfile.File("heatcapljdiat_spring.lammps")


T = log.get("Temp")[5:]
U = log.get("TotEng")[5:]

T2 = log2.get("Temp")[5:]
U2 = log2.get("TotEng")[5:]

slope, intercept, r_value, p_value, std_err = stats.linregress(T, U)
slope2, intercept2, r_value2, p_value2, std_err2 = stats.linregress(T2, U2)

print("Degrees of freedom (2*C_v) for a rigid rod = ", 2*slope)
print("Degrees of freedom (2*C_v) for a spring = ", 2*slope2)
```

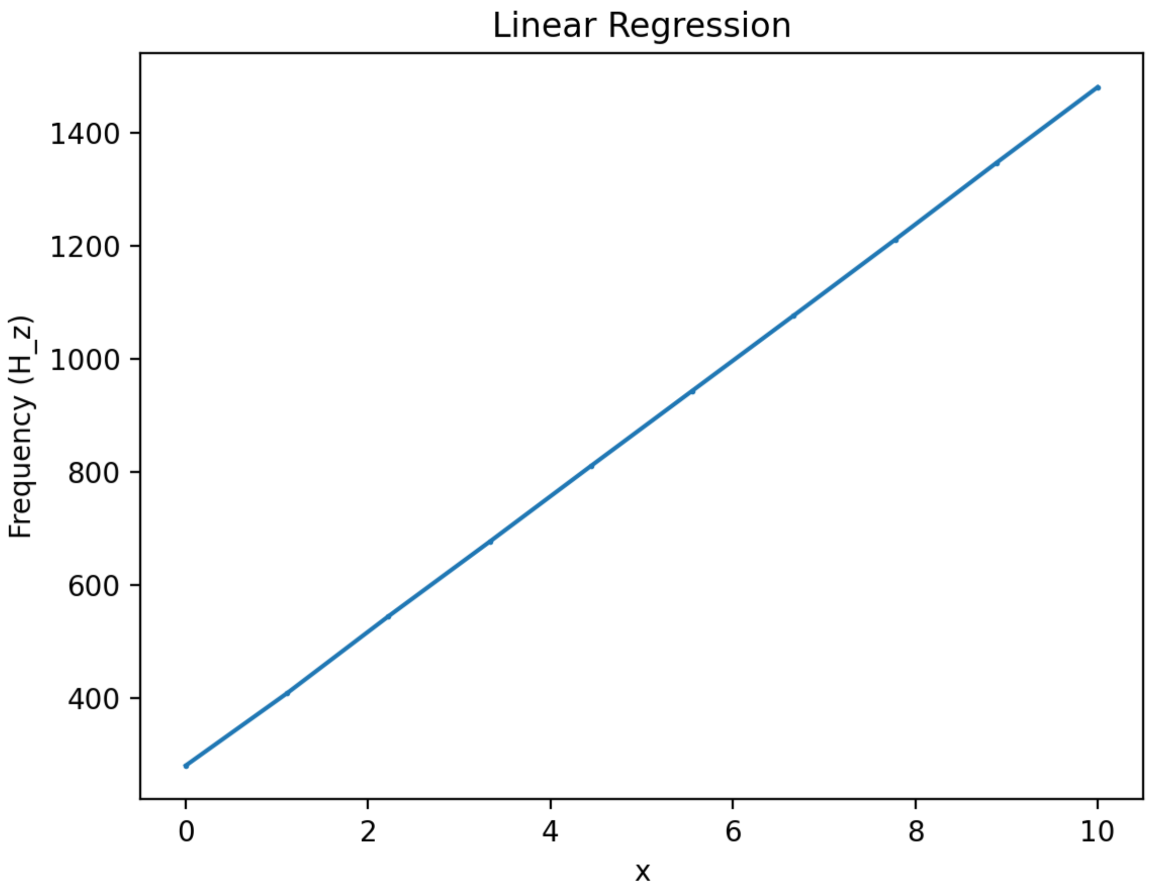
Argon:



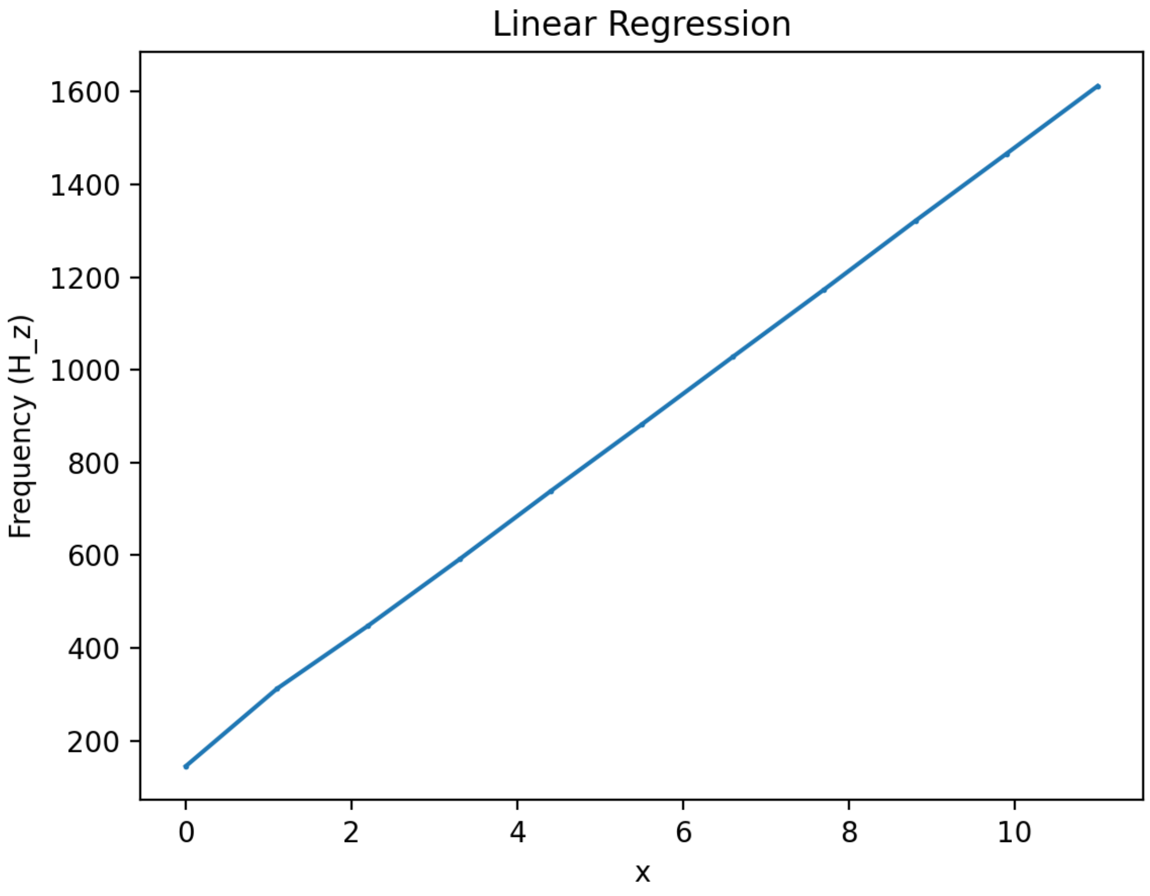
CO_2:



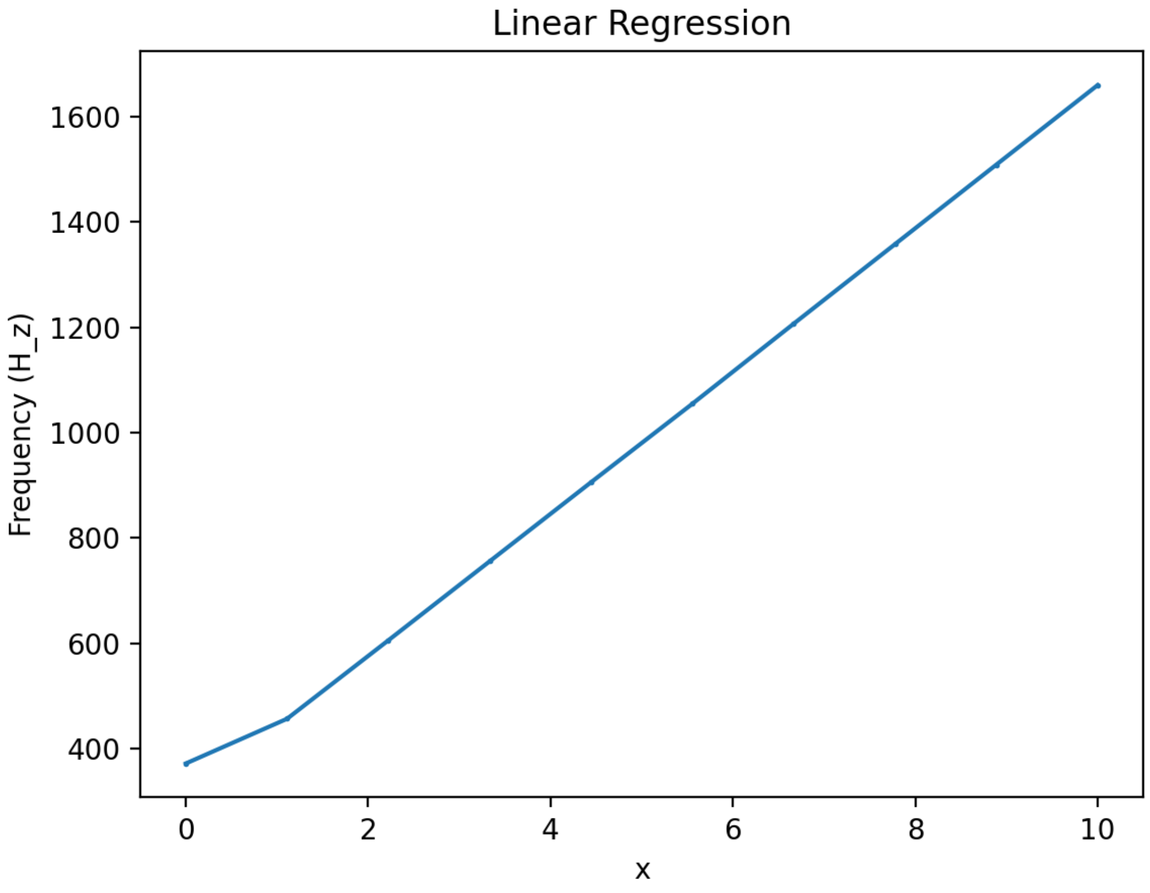
Air at 22 $^{\circ}$C:



Air at 50 $^{\circ}$C:



Air at 70 $^\circ$C:


Linear Regression
Frequency (H_z)
x