Name: Luca Cordes, 444900
Name: Mahmut Can Dogan, 435714

RWTH AACHEN UNIVERSITY

Experimentalphysik II (SS 2023/2024)

Übung 4

Tutorium: 2 Abgabe: 05.05.2023

---

1. **Wärmebad, Gas zusammendrücken**

(a)

$$\begin{aligned} pV &= nk_BT \\ p &= \frac{nk_BT}{V} \\ \Delta Q &= -W \\ &= -\int_{V_0}^{\frac{V_0}{2}} p\,\mathrm{d}V \\ &= -Nk_BT\int_{V_0}^{\frac{V_0}{2}} \frac{1}{V}\mathrm{d}V \\ &= -Nk_BT\left(\ln\left(\frac{V_0}{2}\right) - \ln(V_0)\right) \\ &= \ln\left(\frac{1}{2}\right)k_BNT \end{aligned}$$

(b) Isotherme Kompression:

$$W_I = -\ln\left(V'_{rel}\right)k_BNT$$

Adiabatische Kompression:

$$\begin{aligned} TV^{\kappa-1} &= \text{const} \\ TV^{\frac{2}{3}} &= c \\ T_1 &= T_0\frac{V_0^{\frac{2}{3}}}{V_1^{\frac{2}{3}}} \\ \Delta T = T_1 - T_0 &= T_0\left(\left(\frac{V_0}{V_1}\right)^{\frac{2}{3}} - 1\right) \\ W_A = \frac{3}{2}Nk_B\Delta T &= \frac{3}{2}Nk_BT_0\left(\left(\frac{V_0}{V_1}\right)^{\frac{2}{3}} - 1\right) \end{aligned}$$

Verhältnis:

$$\begin{aligned}\frac{W_I}{W_A} &= \frac{-\ln\left(V'_{rel}\right) k_B N T_0}{\frac{3}{2} N k_B T_0 \left(V_{rel}'^{-\frac{2}{3}} - 1\right)} \\ &= \frac{2}{3} \frac{\ln\left(V'_{rel}\right)}{1 - V_{rel}'^{-\frac{2}{3}}} \\ &= \frac{2}{3} \frac{\ln\left(\frac{1}{20}\right)}{1 - \left(\frac{1}{20}\right)^{-\frac{2}{3}}} \\ &\approx 3.14 \cdot 10^{-1}\end{aligned}$$

2. **Adiabatische Expansion**

   (a) [Jupyter-Notebook/siehe hinten]

   (b)

$$\begin{aligned}TV^{\kappa-1} &= \text{const} \\ T'(2V)^2 &= \text{const} \\ 4T'V^2 &= \text{const} \\ \frac{T'}{T} &= \frac{1}{4}\end{aligned}$$

$$\begin{aligned}\frac{\frac{p'V'}{N'k_B}}{\frac{pV}{Nk_B}} &= \frac{1}{4} \\ \frac{p'}{p} &= \frac{1}{4}\frac{V}{V'} \\ \frac{p'}{p} &= \frac{1}{8}\end{aligned}$$

Die numerischen Ergebnisse stimmen fast exakt mit den analytischen überein.

3. **Entropie beim Kartenmischen**

   (a)

$$\begin{aligned}\Delta S &= k_B \left(\ln n'_{RM} - \ln n_{RM}\right) \\ &= k_B \left(\ln N! - \ln 1\right) \\ &= 1.381 \cdot 10^{-23} \frac{\text{J}}{\text{K}} \left(\ln 52! - \ln 1\right) \\ &\approx 2.16 \cdot 10^{-21} \frac{\text{J}}{\text{K}}\end{aligned}$$

   (b) Aus einer statistischen Sicht, lässt sich der in (a) erhaltene Wert, als ein Maß dafür interpretieren, wie viele zusätzliche Informationen man über das System braucht, um den gegebenen Mikrozustand vollständig zu beschreiben. Der zweite Makrozustand (ungemischt) gibt somit weniger Informationen über den Mikrozustand als der erste Makrozustand (geordnet).

4. **Entropie und Reversibilität**

   (a) Da der Prozess irreversibel ist, gibt es erst mal keinen Umkehrprozess. Hätte man die beiden Körper aber mittels z.B. einer Carnot-Maschine ins thermisches Äquilibrium gebracht, so hätte man die gespeicherte Energie dazu benutzen können, um das System wieder aus dem Äquilibrium zu bringen. Alternativ ermöglicht dies auch externe Energie, mit der man den einen Körper kühlt, und den anderen erhitzt.

   (b) Sobald das System in das Äquilibrium übergegangen ist, wird es diesen nach dem zweiten Haubtsatz der Thermodynamik nicht mehr von alleine verlassen. Da bei dem Prozess auch keine thermische Energie in andere Energieformen umgewandelt wurde ist es somit nicht möglich, ohne weitere externe Energie, den Ablauf umzukehren; Es handelt sich um einen irreversiblen Prozess.

5. **Entropie: Kupferblock**

   (a) Der Prozess ist äquivalent zu dem in Nr.4 und somit irreversibel. Um trotzdem eine Entropiedifferenz anzugeben, muss daher ein reversibler Äquivalenzprozess herangezogen werden.

   (b)

$$c_{\mathrm{H_2O}} = 4.2 \cdot 10^3 \,\frac{\mathrm{J}}{\mathrm{kg\,K}} \qquad\qquad c_{\mathrm{Cu}} = 3.83 \cdot 10^2 \,\frac{\mathrm{J}}{\mathrm{kg\,K}}$$

$$\begin{aligned} c_{\mathrm{H_2O}} m_1 T_0(\mathrm{H_2O}) + c_{\mathrm{Cu}} m_2 T_0 \mathrm{Cu} &= T_1 \left(c_{\mathrm{H_2O}} m_1 + c_{\mathrm{Cu}} m_2 \right) \\ T_1 &= \frac{c_{\mathrm{H_2O}} m_1 T_0(\mathrm{H_2O}) + c_{\mathrm{Cu}} m_2 T_0(\mathrm{Cu})}{c_{\mathrm{H_2O}} m_1 + c_{\mathrm{Cu}} m_2} \\ &\approx 279\,\mathrm{K} \end{aligned}$$

$$\begin{aligned} \Delta S &= \int_K \frac{\mathrm{d}Q_{rev}}{T} \\ &= \int_K \frac{\mathrm{d}U}{T} \\ &= m c_v \int_{T_0}^{T_1} \frac{\mathrm{d}T}{T} \\ &= m c_v \ln\left(\frac{T_1}{T_0}\right) \end{aligned}$$

$$\begin{aligned} \Delta S_{\mathrm{Cu}} &= m c_v \ln\left(\frac{T_1}{T_0}\right) \\ &\approx 1\,\mathrm{kg} \cdot 3.83 \cdot 10^2 \,\frac{\mathrm{J}}{\mathrm{kg\,K}} \ln\left(\frac{279\,\mathrm{K}}{373.15\,\mathrm{K}}\right) \\ &\approx -112\,\frac{\mathrm{J}}{\mathrm{K}} \end{aligned}$$

(c)

$$\begin{aligned}\Delta S_{\mathrm{H_2O}} &= mc_v \ln\left(\frac{T_1}{T_0}\right) \\ &\approx 5\,\mathrm{kg} \cdot 4.2 \cdot 10^3\,\frac{\mathrm{J}}{\mathrm{kg\,K}} \ln\left(\frac{279\,\mathrm{K}}{277.15\,\mathrm{K}}\right) \\ &\approx 130\,\frac{\mathrm{J}}{\mathrm{K}}\end{aligned}$$

(d)

$$\begin{aligned}\Delta S_{ges} &= \Delta S_{\mathrm{Cu}} + \Delta S_{\mathrm{H_2O}} \\ &\approx -112\,\frac{\mathrm{J}}{\mathrm{K}} + 130\,\frac{\mathrm{J}}{\mathrm{K}} \\ &\approx 18.3\,\frac{\mathrm{J}}{\mathrm{K}}\end{aligned}$$

E4

May 3, 2023

1 Jupyter-Notebook zur Experimentalphysik II, SS2023

von Dr. Markus Merschmeyer, III. Physikalisches Institut A, RWTH Aachen University

1.1 Übungsblatt 4, Aufgabe 2: Adiabatische Expansion

Hinweis: Die einzelnen Python-Codeblöcke müssen nacheinander ausgeführt werden, die geschieht jeweils durch Drücken der Tastenkombination Shift+Return oder mithilfe des obigen Menüs. Fall etwas schief gehen sollte, kann mit dem Knopf restart the kernel (kreisförmiges Pfeilsymbol) alles zurueckgesetzt werden.

1.1.1 1. Einbinden externer Python-Pakete

Hier muss zusätzlich %matplotlib inline angegeben werden, um die erzeugten Diagramme direkt im Jupyter-Notebook darstellen zu können:

[3]:
```
import numpy as np              # numpy-Paket fuer Array-Funktionen, sqrt() usw.
 ↪ importieren
%matplotlib inline
import matplotlib.pyplot as plt # Paket zur Diagrammerstellung
from matplotlib import colors, cm # ... + 'ticker' (?)
```

1.1.2 2. Definieren benötigter Konstanten, Parameter und Anfangswerte

Aus dem scipy.constants-Paket wird die Boltzmann-Konstante und die atomare Masseneinheit benoetigt:

[4]:
```
import scipy.constants as scc   # scipy-Paket fuer Konstanten importieren
c_kB  = scc.Boltzmann   # k_B = Boltzmann-Konstante (in J/K)
print("Boltzmann-Konstante k_B: ",c_kB,"J/K")
c_amu = scc.atomic_mass # atomare Masseneinheit (in kg)
print("atomare Masseneinheit:   ",c_amu,"kg")
```

```
Boltzmann-Konstante k_B:  1.380649e-23 J/K
atomare Masseneinheit:    1.6605390666e-27 kg
```

Parameter und Anfangswerte:

[5]:
```
L_0 = 1.      # Anfangslänge des Kastens (in m)
L_1 = 2.      # End-Länge des Kastens (in m)
```

```
w   = 0.1     # Verschiebungsgeschwindigkeit der rechten Kastenwand (in m/s)
v_0 = 1000.   # Startgeschwindigkeit des Gasteilchens (in m/s)
fg  = 1.      # Anzahl der Freiheitsgrade des Gasteilchens
# Teilchenmasse wird für innere Energie benötigt
m   = 28*c_amu # -> z.B. Stickstoff- oder Sauerstoff-Molekül (in kg)
print("Teilchenmasse: ",m,"kg")
```

```
Teilchenmasse:  4.64950938648e-26 kg
```

1.1.3 3. Berechnung des Expansionsvorgangs

Es werden einige leere Listen und Variablen erzeugt, zur Speicherung der Werte aus den einzelnen Berechnungsschritten. Dann werden mit einer while-Schleife und der Abbruchbedingung des Erreichens der maximalen Kastenlänge die notwendigen Berechnungen durchgeführt.

[53]:
```
n_col = 0    # (bisherige) Anzahl der Stoßvorgänge an der Kastenwand

l_L = [] # leere Liste für die aktuelle Länge des Kastens (im m)
l_v = [] # leere Liste für die aktuelle Geschwindigkeit der Gasteilchen (in m/s)
l_t = [] # leere Liste für die aktuelle Dauer eines Kollisionszyklus' (in s)

t_tmp = 0.   # Startzeitpunkt
L_tmp = L_0  # initialisiere anfängliche Länge des Kastens
v_tmp = v_0  # initialisiere anfängliche Geschwindigkeit des Teilchens

while L_tmp<L_1: # while-Schleife mit Abbruchbedingung
    l_t.append(t_tmp) # speichere aktuelle Zeit (an Liste anhängen)
    l_v.append(v_tmp) # speichere aktuelle Geschwindigkeit (an Liste anhängen)
    l_L.append(L_tmp) # speichere aktuelle Länge des Kastens (an Liste anhängen)

    n_col += 1        # erhöhe Zähler für Anzahl der Kollisionen des␣
 ↪Gasteilchens mit der rechten Kastenwand

    L_tmp += t_tmp * w
    v_tmp -= 2*w
    t_tmp = 2 * L_tmp / v_tmp # berechne Zeitpunkt des nächsten Stoßes

print("Anzahl berechneter Kollisionen: ",n_col)
print("Schlusszeitpunkt: ",sum(l_t),"s")                # Summe aller␣
 ↪Listenelemente
print("finale Kastenlänge: ",l_L[-1],"m")               # letztes Element der␣
 ↪Liste
print("finale Teilchengeschwindigkeit v1: ",l_v[-1],"m/s") # letztes Element␣
 ↪der Liste
```

```
Anzahl berechneter Kollisionen:  2501
Schlusszeitpunkt:  10.00000000000141 s
finale Kastenlänge:  1.999200319872196 m
```

```
finale Teilchengeschwindigkeit v1:  499.9999999998898 m/s
```

1.1.4 4. Berechnung des Verlaufs von Temperatur und Druck

Aus den unter 3. in den entsprechenden Listen gesammelten Informationen über den Verlauf von L und v können jetzt die Verläufe von T und p berechnet werden. Bei Berechnungen mit numpy-Listen sind die Methoden numpy.multiply(A,B) und numpy.divide(A,B) hilfreich. Hier werden die entsprechenden Rechenoperationen mit allen Elementen der Listen A und B durchgeführt.

[58]:
```
# numerische Ergebnisse der Simulation
l_T = np.multiply(m / (c_kB * fg) , np.multiply(l_v, l_v)) # "Temperatur" des␣
 ↪"Gases" (in K)  [T=U*2/(N*kB*fg)]
l_p = np.multiply(c_kB, np.divide(l_T, l_L)) # "Druck x Quersschnittsfläche des␣
 ↪Kastens" (in N) [p=N*kB*T/V]
print("Verhältnis T1/T0: ",l_T[-1]/l_T[0],)
print("Verhältnis p1/p0: ",l_p[-1]/l_p[0],)

# (ggf.) analytische
```

```
Verhältnis T1/T0:  0.24999999999988978
Verhältnis p1/p0:  0.12504999999993582
```

1.1.5 5. Grafische Darstellung der Ergebnisse

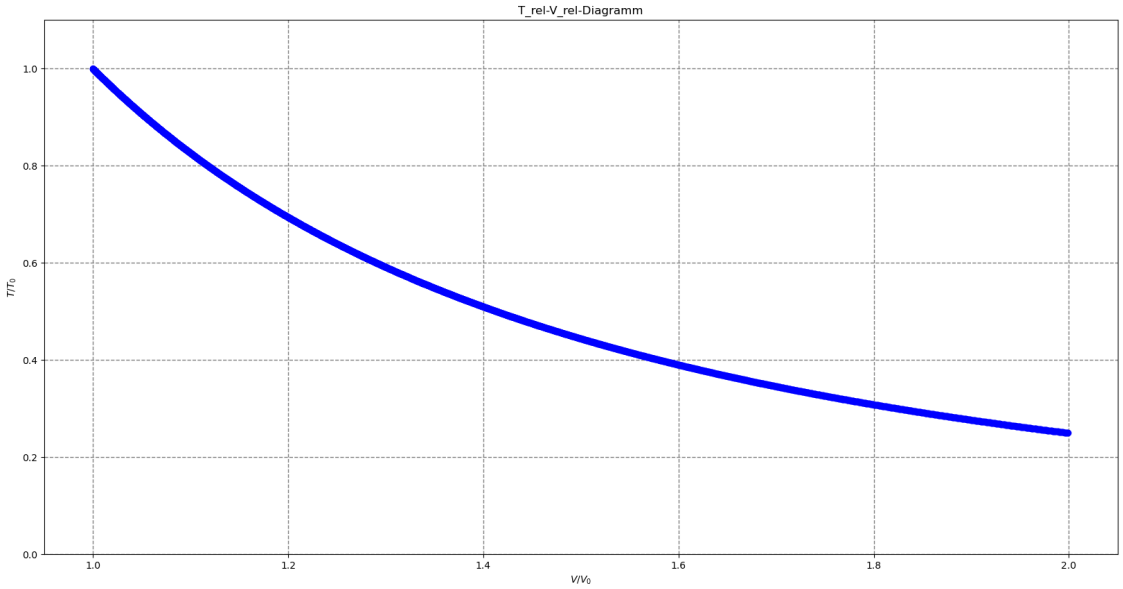
Nun werden die $T1/T0$-$V1/V0$- und $p1/p0$-$V1/V0$-Diagramme vorbereitet und dargestellt.

[59]:
```
# T1/T0-V1/V0-Diagramm
fig, ax = plt.subplots(1, sharex=False)
fig.set_size_inches(20., 10.) # Groesse der exportierten Bilddatei in Zoll␣
 ↪(Breite, Hoehe)

# Berechnung der Listen für die x- und y-Werte
xtmp  = np.multiply(l_L, 1/l_L[0])
ytmp  = np.multiply(l_T, 1/l_T[0])

l1 = ax.plot(xtmp, ytmp, 'bo')
ax.grid(color='gray', linestyle='--', linewidth=1)
ax.set_title('T_rel-V_rel-Diagramm')   # ... Diagrammtitel
ax.set_xlabel('$V / V_0$')     # ... Beschriftung x-Achse
ax.set_ylabel('$T / T_0$')     # ... Beschriftung y-Achse
ax.axis([0.95, 2.05, 0., 1.1]) # ... Wertebereich x-Achse, y-Achse

#fig.savefig('AdEx_Trel_vs_Vrel.png', dpi=300)
plt.show()
```

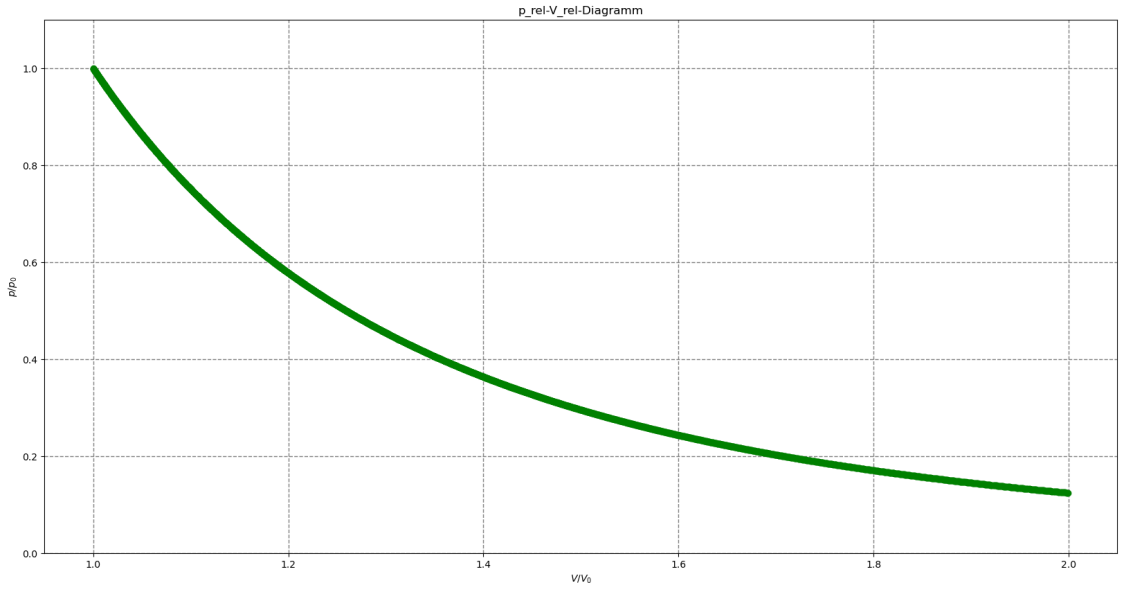


[60]:
```
# p1/p0-V1/V0-Diagramm
fig, ax = plt.subplots(1, sharex=False)
fig.set_size_inches(20., 10.) # Groesse der exportierten Bilddatei in Zoll␣
 ↪(Breite, Hoehe)

# Berechnung der Listen für die x- und y-Werte
#xtmp  = l_v
ytmp  = np.multiply(l_p, 1/l_p[0])

l1 = ax.plot(xtmp, ytmp, 'go')
ax.grid(color='gray', linestyle='--', linewidth=1)
ax.set_title('p_rel-V_rel-Diagramm') # ... Diagrammtitel
ax.set_xlabel('$V / V_0$')     # ... Beschriftung x-Achse
ax.set_ylabel('$p / p_0$')     # ... Beschriftung y-Achse
ax.axis([0.95, 2.05, 0., 1.1]) # ... Wertebereich x-Achse (z-Werte), y-Achse␣
 ↪(Bz-Feldstaerke)

#fig.savefig('AdEx_prel_vs_Vrel.png', dpi=300)
plt.show()
```


p_rel-V_rel-Diagramm
1.0
0.8
0.6
0.4
0.2
0.0
p/p_0
1.0
1.2
1.4
1.6
1.8
2.0
V/V_0

[ ]: