In [2]:

```
#更新日 : 2022/08/24   更新者 : 佐々木亮
#更新日 : 2022/10/29   更新者 : 佐々木亮
#更新日 : 2023/02/12   更新者 : 佐々木亮


#TDSの低感度QMS用プログラムです。
#低感度だとD2とHeの区別がつかないのでHeとD2が同時に脱離する場合は適用できません。

#--- 以下の項目を入力してください。------------------------------------------------

#リーク測定のascファイルを入力してください。（注意 : r"パス"としてください。）
leak_asc = r"C:¥Users¥user¥Desktop¥0108TDS¥230108_leak2.asc"

#脱離測定のascファイルを入力してください。
desorption_asc = r"C:¥Users¥user¥Desktop¥0108TDS¥230108_desorption.asc"

#温度のtxtファイルを入力してください。
temperature_txt = r"C:¥Users¥user¥Desktop¥0108TDS¥LOGFILE20230108172917.txt"

#較正にD2リークボトルを用いる場合は bottle = 0 Heリークボトルを用いる場合は bottle = 1
bottle = 1

#試料の面積（気体が脱離してくる部分の面積）単位は [ m^2 ]で入力してください。
S = 4*4*3.14e-6

#出力ファイルの名前を入力してください。file_name.csvの名前で出力されます。
file_name = "plot_data"

#--------------------------------------------------------------------------------














import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from statistics import mean

plt.rcParams["font.size"] = 18
plt.rcParams["figure.figsize"] = [10.0, 6.0]


'''標準リークから電流-Flux変換係数sigmaを求める'''

#.txtを入れるとデータフレームで返す関数を定義
def txt_to_dataflame(file_name):
    #ファイルを開いてすべての行の成功を取得
    with open(file_name, encoding="cp932") as f:
        lines = f.readlines()
    #すべての行のタブ、改行、”、’に対してそれぞれ置換を行い、カンマでデータを区切っ
    index = 0
    #len_max = 0
    for k in lines:
        k = k.replace("¥t", ",")
        k = k.replace("¥n", "")
        k = k.replace('"', '')
        k = k.replace("'", "")
        k = k.split(",")
        lines[index] = k
        index += 1

    #pandasのデータフレームに格納する
    #データフレームの初期値
    df_n = pd.DataFrame([lines[0]])
    index = 0
    for k in lines:
        if index >= 1:
            df_k = pd.DataFrame([k])
            df_n = pd.concat([df_n, df_k], ignore_index=True)
        index += 1

    return df_n


#室温を取得します
#温度のtxtファイルを読み込みます
df = txt_to_dataflame( temperature_txt )
T = float( df.iloc[8,1] )
print("室温は {}℃".format(T))


#標準リークのascファイルを読み込みます
df = txt_to_dataflame( leak_asc )
#df

#ascの中身から測定の日時を抽出します
date = df.iloc[1,1]

Year = int( date[0:4] )
Month = int( date[5:7] )
Day = int( date[8:10] )

print('リーク測定日 {}-{}-{}'.format(Year, Month, Day))

#ascの中身のうち、必要なデータを抽出します

#必要な行(index)と列(column)の範囲を指定します
index_front = 26
index_back = len(df) - 1

column_front = 3
column_back = 18

df = df.iloc[index_front:index_back, column_front:column_back]
df = df.reset_index(drop=True)

#配列timeに経過時間、配列mz_4に脱離電流値を格納する
df = df.astype(float)

time = df[3]
mz_4_current = df[6]
```

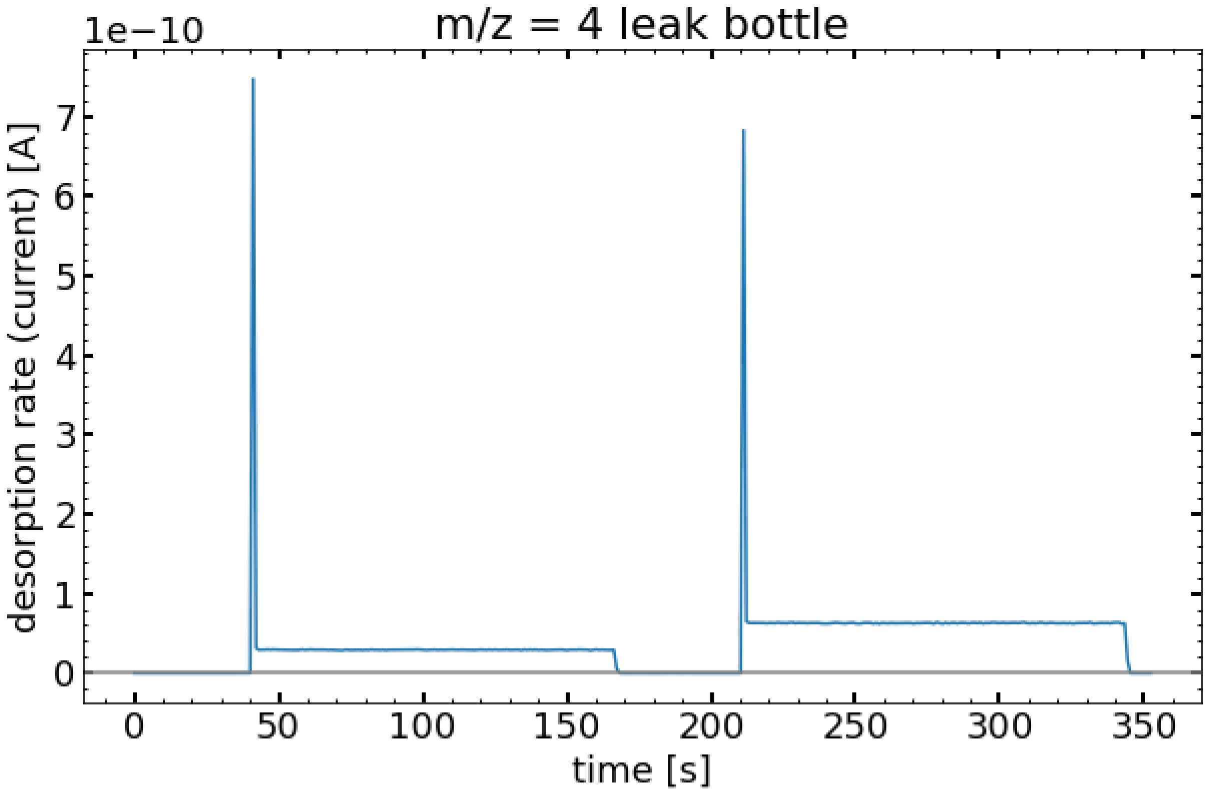
```
#グラフにプロットして、平均電流を計算する範囲を指定します
plt.plot(time, mz_4_current)
plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("m/z = 4 leak bottle")
plt.xlabel("time [s]")
plt.ylabel("desorption rate (current) [A]")
plt.axhline(0, color='grey')
plt.show()

if bottle == 0:
    b_D2_low = float(input('D2の時間の下限[s]を入力してください。'))
    b_D2_high = float(input('D2の時間の上限[s]を入力してください。'))

if bottle == 1:
    b_He_low = float(input('Heの時間の下限[s]を入力してください。'))
    b_He_high = float(input('Heの時間の上限[s]を入力してください。'))


#グラフにプロットします
plt.plot(time, mz_4_current, label="m/z = 4")
plt.axhline(0, color='grey')

if bottle == 0:
    D2_current_ave = []
    for i in range(len(time)):
        if b_D2_low <= time[i] <= b_D2_high:
            D2_current_ave.append(mz_4_current[i])
    D2_current_ave = mean(D2_current_ave)

    print("D2標準リークの定常電流値は {} [A]".format(D2_current_ave))

    plt.axvline(b_D2_low, color="limegreen")
    plt.axvline(b_D2_high, color="limegreen")
    plt.axvspan(b_D2_low, b_D2_high, color="limegreen", alpha=0.1)

if bottle == 1:
    He_current_ave = []
    for i in range(len(time)):
        if b_He_low <= time[i] <= b_He_high:
            He_current_ave.append(mz_4_current[i])
    He_current_ave = mean(He_current_ave)

    print("He標準リークの定常電流値は {} [A]".format(He_current_ave))

    plt.axvline(b_He_low, color="limegreen")
    plt.axvline(b_He_high, color="limegreen")
    plt.axvspan(b_He_low, b_He_high, color="limegreen", alpha=0.1)

plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("m/z =4 leak bottle")
plt.xlabel("time [s]")
plt.ylabel("desorption rate (current) [A]")
plt.show()

#今日の日付でのleak rateを計算する

NA = 6.02214076e23
R = 8.314462618

D2_leak_rate = 3.12e-8
```

```
D2_depletion_rate = 0.046      # per year
D2_temp_coeff = 0.002          # per ℃

He_leak_rate = 2.6e-8
He_depletion_rate = 0.023      # per year
He_temp_coeff = 0.023          # per ℃

#温度の差
D2_delta_temp = float( abs( 24.5 - T ) )
He_delta_temp = float( abs( 25.0 - T ) )


#D2標準リークの経過日数（リークボトルを新しくしたらこの部分は書き直す必要があります）
dt = datetime.datetime(year=2013, month=3, day=11)
dt_now = datetime.datetime(year = Year, month = Month, day = Day)
delta_time = dt_now - dt

D2_delta_day = float( delta_time.days )   #経過日数


#He標準リークの経過日数（リークボトルを新しくしたらこの部分は書き直す必要があります）
dt = datetime.datetime(year=2010, month=8, day=9)
delta_time = dt_now - dt

He_delta_day = float( delta_time.days )   #経過日数

leak_D2 = D2_leak_rate * ( 1 - ( (D2_depletion_rate / 365) * D2_delta_day ) - ( D2_te
leak_He = He_leak_rate * ( 1 - ( (He_depletion_rate / 365) * He_delta_day ) - ( He_te

#D2,Heのsigmaの値を求める
if bottle == 0:
    sigma_D2 = R * ( T + 273.15 ) * D2_current_ave / ( NA * leak_D2 )
    print("sigma_D2 = {}".format(sigma_D2))

if bottle == 1:
    sigma_He = R * ( T + 273.15 ) * He_current_ave / ( NA * leak_He )
    print("sigma_He = {}".format(sigma_He))

# C = sigma_D2 / sigma_He
# print("sigma_D2 / sigma_He = {}".format(C))

'''脱離測定の電流値をフラックスに変換する'''

#脱離測定のtxtファイルを読み込みます
df = txt_to_dataflame( desorption_asc )
#df

#deso_timeにQMSの計測時刻(unix)を格納する
date = df.iloc[26:len(df)-1, 1]
deso_time = df.iloc[26:len(df)-1, 2]
date = date.reset_index(drop=True)
deso_time = deso_time.reset_index(drop=True)

for i in range(len(deso_time)):
    dd = date[i]
    dd = [int(dd[0:4]), int(dd[5:7]), int(dd[8:10])]

    d = deso_time[i]

    if d[1] != ':':
        d = [int(d[0:2]), int(d[3:5]), int(d[6:8])]
    elif d[1] == ':':
        d = [int(d[0:1]), int(d[2:4]), int(d[5:7])]
    deso_time_unix = datetime.datetime(dd[0], dd[1], dd[2], d[0], d[1], d[2])
    deso_time_unix = deso_time_unix.timestamp()

    deso_time[i] = deso_time_unix

#脱離測定のtxtの中身のうち、必要なデータを抽出します
#必要な行(index)と列(column)の範囲を指定します
index_front = 26
index_back = len(df)-1

column_front = 3
column_back = 18

df = df.iloc[index_front:index_back, column_front:column_back]
df = df.reset_index(drop=True)
df = df.astype(float)

#配列mz_2、mz_3、mz_4に脱離電流値を格納する
mz_2_current = df[4]
mz_3_current = df[5]
mz_4_current = df[6]


'''横軸を時間から温度に変換します'''

#温度のtxtファイルを読み込みます
df = txt_to_dataflame( temperature_txt )
#df

#txtの中身のうち、必要なデータを抽出します
#必要な行(index)と列(column)の範囲を指定します
index_front = 8
index_back = len(df)

column_front = 0
column_back = 2

df = df.iloc[index_front:index_back, column_front:column_back]
df = df.reset_index(drop=True)
#df


#時刻 vs. 試料温度 のグラフを作ります
heating_time = df[0]
heating_temp = df[1].astype(float)

for i in range(len(heating_time)):
    h = heating_time[i]
    h = [int(h[0:4]), int(h[5:7]), int(h[8:10]), int(h[11:13]), int(h[14:16]), int(h[1
    h_unix = datetime.datetime(h[0], h[1], h[2], h[3], h[4], h[5])
    h_unix = h_unix.timestamp()
    heating_time[i] = h_unix

temp_max_index = np.argmax(heating_temp)
temp_max = max(heating_temp)

heating_temp = heating_temp.iloc[:temp_max_index+1]
heating_time = heating_time.iloc[:temp_max_index+1]

plt.plot(heating_time, heating_temp)

plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Temperature vs. Time")
plt.xlabel("UNIX Time [s]")
plt.ylabel("Temperature [$¥mathrm{^¥circ C}$]")
plt.show()


plt.plot(deso_time, mz_4_current, label="m / z = 4")
plt.plot(deso_time, mz_3_current, label="m / z = 3")
plt.plot(deso_time, mz_2_current, label="m / z = 2")

plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Current vs. Time")
plt.xlabel("UNIX time [s]")
plt.ylabel("Current [A]")
plt.legend()
plt.show()

check2 = 0 # check2 = 1 にすると、必要ないデータを除去している過程のプロットが見れます

deso_temp = []
f_1d = interp1d(heating_time, heating_temp)
tt_min = heating_time[1]                         #インデックスの誤差をなくすために0で
tt_max = heating_time[len(heating_time)-2]       #インデックスの誤差をなくすためにle

for i in range(len(deso_time)):
    tt = deso_time[i]

    if tt < tt_min:
        tt = tt_min
        y = f_1d(tt) - 100
    elif tt > tt_max:
        tt = tt_max
        y = f_1d(tt) + 100
    else:
        y = f_1d(tt)

    deso_temp.append(y)

deso_temp = pd.Series(deso_temp)

if check2 == 1:
    plt.scatter(deso_temp, mz_4_current, label="m / z = 4")
    plt.scatter(deso_temp, mz_3_current, label="m / z = 3")
    plt.scatter(deso_temp, mz_2_current, label="m / z = 3")
    plt.show()

# 高温側のいらないデータを除去します
deso_temp_max_index = np.argmax(deso_temp)

deso_temp = deso_temp.iloc[:deso_temp_max_index]
mz_4_current = mz_4_current.iloc[:deso_temp_max_index]
mz_3_current = mz_3_current.iloc[:deso_temp_max_index]
mz_2_current = mz_2_current.iloc[:deso_temp_max_index]
deso_time = deso_time.iloc[:deso_temp_max_index]

if check2 == 1:
    plt.scatter(deso_temp, mz_4_current, label="m / z = 4")
    plt.scatter(deso_temp, mz_3_current, label="m / z = 3")
    plt.scatter(deso_temp, mz_2_current, label="m / z = 2")
    plt.show()

deso_temp = deso_temp.sort_index(ascending=False)
```

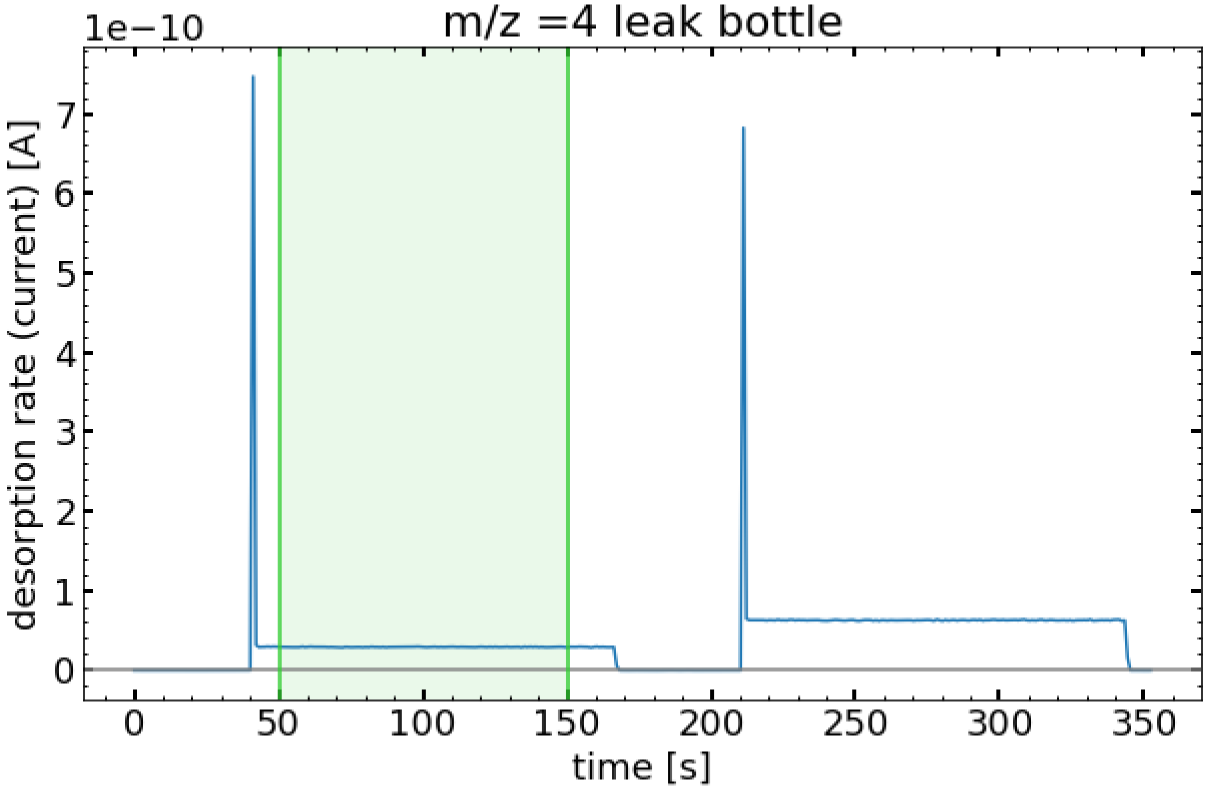
```
mz_4_current = mz_4_current.sort_index(ascending=False)
mz_3_current = mz_3_current.sort_index(ascending=False)
mz_2_current = mz_2_current.sort_index(ascending=False)
deso_time = deso_time.sort_index(ascending=False)

deso_temp = deso_temp.reset_index(drop=True)
mz_4_current = mz_4_current.reset_index(drop=True)
mz_3_current = mz_3_current.reset_index(drop=True)
mz_2_current = mz_2_current.reset_index(drop=True)
deso_time = deso_time.reset_index(drop=True)

# 低温側のいらないデータを除去します
deso_temp_min_index = np.argmin(deso_temp)

deso_temp = deso_temp.iloc[:deso_temp_min_index]
mz_4_current = mz_4_current.iloc[:deso_temp_min_index]
mz_3_current = mz_3_current.iloc[:deso_temp_min_index]
mz_2_current = mz_2_current.iloc[:deso_temp_min_index]
deso_time = deso_time.iloc[:deso_temp_min_index]

deso_temp = deso_temp.sort_index(ascending=False)
mz_4_current = mz_4_current.sort_index(ascending=False)
mz_3_current = mz_3_current.sort_index(ascending=False)
mz_2_current = mz_2_current.sort_index(ascending=False)
deso_time = deso_time.sort_index(ascending=False)

deso_temp = deso_temp.reset_index(drop=True)
mz_4_current = mz_4_current.reset_index(drop=True)
mz_3_current = mz_3_current.reset_index(drop=True)
mz_2_current = mz_2_current.reset_index(drop=True)
deso_time = deso_time.reset_index(drop=True)

if check2 == 1:
    plt.scatter(deso_temp, mz_4_current, label="m / z = 4")
    plt.scatter(deso_temp, mz_3_current, label="m / z = 3")
    plt.scatter(deso_temp, mz_2_current, label="m / z = 2")
    plt.show()

plt.plot(deso_temp, mz_4_current, label="m / z = 4")
plt.plot(deso_temp, mz_3_current, label="m / z = 3")
plt.plot(deso_temp, mz_2_current, label="m / z = 2")
plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Current vs. Temperature")
plt.xlabel("Temperature [℃]")
plt.ylabel("Current [A]")
plt.legend()
plt.show()

#温度を9次関数で較正する
temp_calib_only = []
for i in range(len(deso_temp)):
    deso_temp[i] = deso_temp[i] + 273.15  # deso_temp配列の単位を℃からKに変更

    Ti = deso_temp[i]
    Tu = -5668.560007 + 72.92460021*Ti -0.3854752562*pow(Ti,2) +0.001157533027*pow(Ti

    temp_calib_only.append(Tu)

calib_raw = []
for i in range(len(temp_calib_only)):
    calib_raw.append(temp_calib_only[i]/deso_temp[i])
```

```
del_high = 0.1
del_low = 0.1
for i in range(len(calib_raw)):
    if ( abs( 1.0 - calib_raw[i] ) <= del_high ) and ( 800 <= deso_temp[i] <= 1100 )
        del_high = abs( 1.0 - calib_raw[i] )
        switch_index_high = i

    if ( abs( 1.0 - calib_raw[i] ) <= del_low ) and ( deso_temp[i] <= 350 ):
        del_low = abs( 1.0 - calib_raw[i] )
        switch_index_low = i

switch_temp_high = deso_temp[switch_index_high]
switch_temp_low = deso_temp[switch_index_low]


plt.plot(deso_temp, calib_raw)
plt.ylim(-0.1,1.5)
plt.axhline(1.0, color="r")
plt.axhline(0.0, color="y")
plt.axvline(switch_temp_high, color="orange")
plt.axvline(switch_temp_low, color="orange")
plt.grid()
plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Ti vs. Tu / Ti")
plt.xlabel("Raw Temperature [K]")
plt.ylabel(" Calibrated Temperature / Raw Temperature")
plt.show()


temp_calib = []
count = 0
for i in range(len(deso_temp)):
    Ti = deso_temp[i]              #deso_tempの単位は今は K になっている
    Tu = -5668.560007 + 72.92460021*Ti -0.3854752562*pow(Ti,2) +0.001157533027*pow(Ti


    if switch_temp_low < Ti < switch_temp_high:
        temp_calib.append(Tu)
    else:
        temp_calib.append(Ti)

temp_calib = pd.Series(temp_calib)

plt.plot(temp_calib, mz_4_current, label="m / z = 4")
plt.plot(temp_calib, mz_3_current, label="m / z = 3")
plt.plot(temp_calib, mz_2_current, label="m / z = 2")
plt.minorticks_on() #補助メモリの描写
plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Current vs. Temperature")
plt.xlabel("Temperature [K]")
plt.ylabel("Current [A]")
plt.legend()
plt.show()


#QMSの電流値をフラックスに変換する
if bottle == 0:

    mz_2_flux = []
    mz_3_flux = []
    mz_4_flux = []
    D_flux = []


    for i in range(len( temp_calib )):
        mz_2_flux.append( mz_2_current[i] / ( S * sigma_D2 ) * 2 )     #sigma_HDはわか
        mz_3_flux.append( mz_3_current[i] / ( S * sigma_D2 ) )     #sigma_HDはわからな
        mz_4_flux.append( mz_4_current[i] / ( S * sigma_D2 ) * 2 )   #Dの脱離量を見る
        D_flux.append( mz_3_flux[i] + mz_4_flux[i] )  #D2でなくDで揃えた

    D_retention = 0
    for i in range(len(deso_time)-1):
        #D_retention += D_flux[i] * ( deso_time[i+1] - deso_time[i] )
        D_retention += mz_4_flux[i] * ( deso_time[i+1] - deso_time[i] )  # D2のflux *


    plt.plot(temp_calib, mz_4_flux, label=r"$2 ¥times ¥mathrm{D_{2}}$")
    plt.plot(temp_calib, mz_3_flux, label="HD")
    #plt.plot(temp_calib, mz_2_flux, label=r"$2 ¥times ¥mathrm{H_{2}}$")
    plt.plot(temp_calib, D_flux, label=r"$2 ¥times ¥mathrm{D_{2}} + ¥mathrm{HD}$")
    plt.minorticks_on() #補助メモリの描写
    plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="c
    plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="c
    plt.title("Flux vs. Temperature")
    plt.xlabel("Temperature [K]")
    plt.ylabel("Flux [$¥mathrm{D} ¥, / ¥, (¥mathrm{m}^{2} ¥cdot ¥mathrm{s})$]")
    plt.legend()
    plt.show()

    """
    # log plot
    plt.plot(temp_calib, mz_4_flux, label=r"$2 ¥times ¥mathrm{D_{2}}$")
    plt.plot(temp_calib, mz_3_flux, label="HD")
    plt.plot(temp_calib, mz_2_flux, label=r"$2 ¥times ¥mathrm{H_{2}}$")
    plt.plot(temp_calib, D_flux, label=r"$2 ¥times ¥mathrm{D_{2}} + ¥mathrm{HD}$")
    plt.title("Flux vs. Temperature")
    plt.xlabel("Temperature [K]")
    plt.ylabel("Flux [$¥mathrm{D} ¥, / ¥, ¥mathrm{m}^{2} ¥cdot ¥mathrm{s}$]")
    plt.semilogy()
    plt.legend()
    plt.show()
    """

    print("retention of D = {} [D/m^2 s]".format(D_retention))


if  bottle == 1:

    He_flux = []

    for i in range(len( temp_calib )):
        He_flux.append( mz_4_current[i] / ( S * sigma_He ) )

    He_retention = 0
    for i in range(len(deso_time)-1):
        He_retention += He_flux[i] * ( deso_time[i+1] - deso_time[i] )


    plt.plot(temp_calib, He_flux, label="He")
    plt.minorticks_on() #補助メモリの描写
    plt.tick_params(axis="both", which="major",direction="in",length=5,width=2,top="c
    plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="c
    plt.title("Flux vs. Temperature")
    plt.xlabel("Temperature [K]")
    plt.ylabel("Flux [$¥mathrm{He} ¥, / ¥, (¥mathrm{m}^{2} ¥cdot ¥mathrm{s})$]")
    plt.show()

    print("retention of He = {} [He/m^2 s]".format(He_retention))

# CSVファイルで出力します
space = pd.DataFrame({'':[]})

if bottle == 0:
    D2CA = pd.DataFrame({'D2_リーク電流平均 [A]': [D2_current_ave]})
    SD2 = pd.DataFrame({'sigma_D2': [sigma_D2]})

    DT = pd.DataFrame({'UNIX time [s]': deso_time})
    TC = pd.DataFrame({'temperature [K]': temp_calib})
    mz4C = pd.DataFrame({'mz_4 QMS電流値 [A]': mz_4_current})
    mz3C = pd.DataFrame({'mz_3 QMS電流値 [A]': mz_3_current})
    mz2C = pd.DataFrame({'mz_2 QMS電流値 [A]': mz_2_current})

    mz4F = pd.DataFrame({'D_flux [D/m^2 s]': mz_4_flux})
    mz3F = pd.DataFrame({'HD_flux [HD/m^2 s]': mz_3_flux})
    mz2F = pd.DataFrame({'H_flux [H/m^2 s]': mz_2_flux})

    DR = pd.DataFrame({'D_retention [D/m^2]': [D_retention]})

    df_all = pd.concat([DT, TC, mz4C, mz3C, mz2C, mz4F, mz3F, mz2F, space, DR, D2CA, 

if bottle == 1:
    HeCA = pd.DataFrame({'He_current_ave [A]': [He_current_ave]})
    SHe = pd.DataFrame({'sigma_He': [sigma_He]})

    DT = pd.DataFrame({'UNIX time [s]': deso_time})
    TC = pd.DataFrame({'temperature [K]': temp_calib})

    mz4C = pd.DataFrame({'mz_4_current [A]': mz_4_current})

    mz4F = pd.DataFrame({'He_flux [He/m^2 s]': He_flux})
    HeR = pd.DataFrame({'He_retention [He/m^2]': [He_retention]})

    df_all = pd.concat([DT, TC, mz4C, mz4F, space, HeR, HeCA, SHe], axis=1)

# CSV ファイル (file_name.csv(最初に入力した名前)) として出力
file_name = file_name + ".csv"

df_all.to_csv(file_name, index=False, encoding="cp932")

print('Finish!')
```

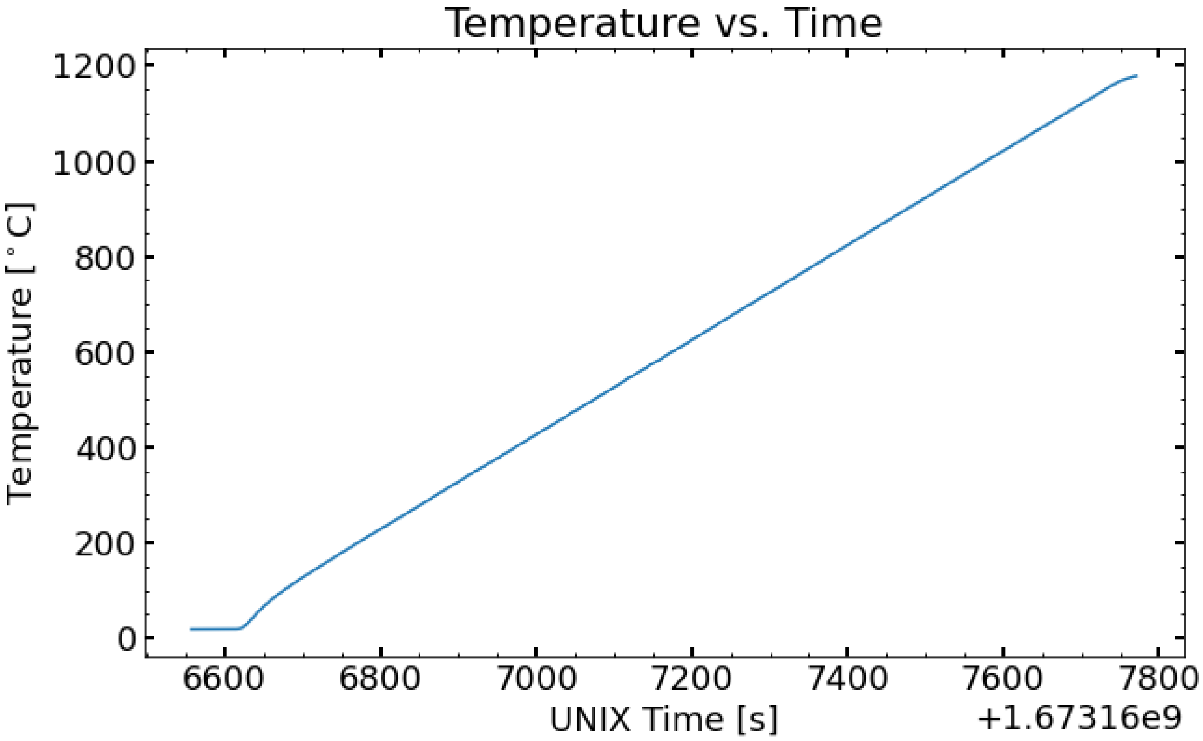
室温は 18.1℃
リーク測定日 2023-1-8

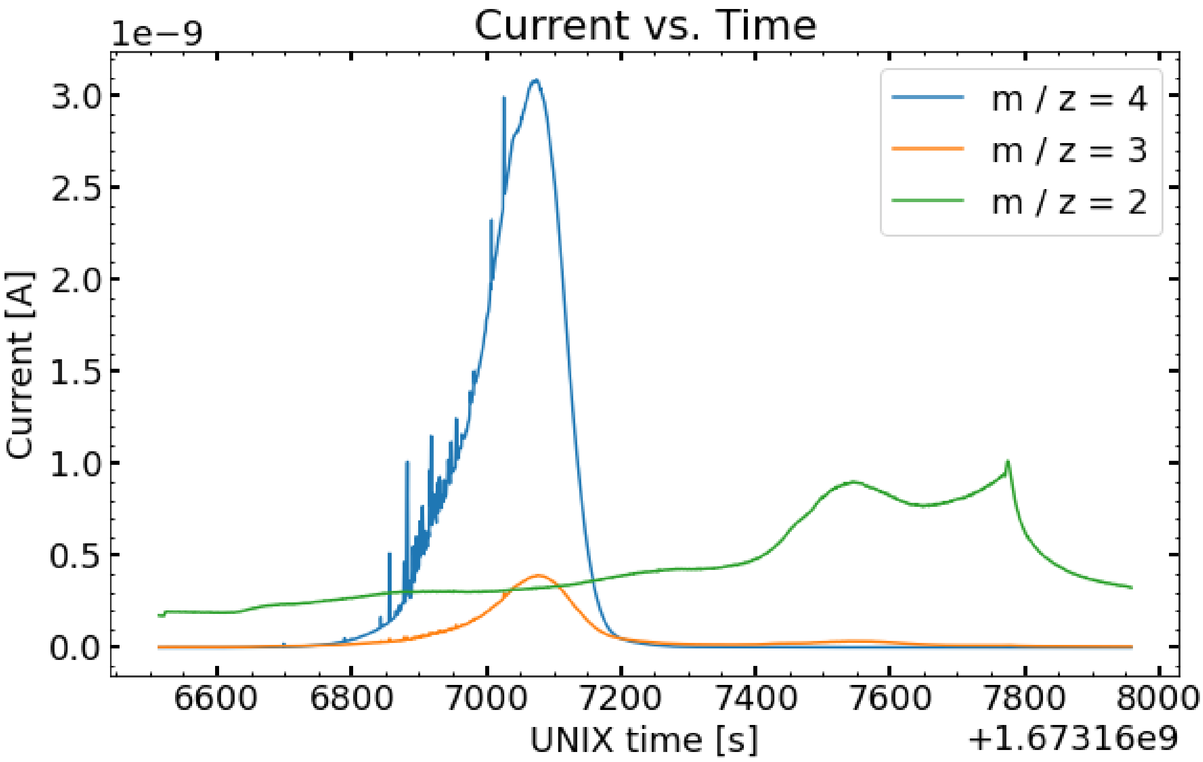


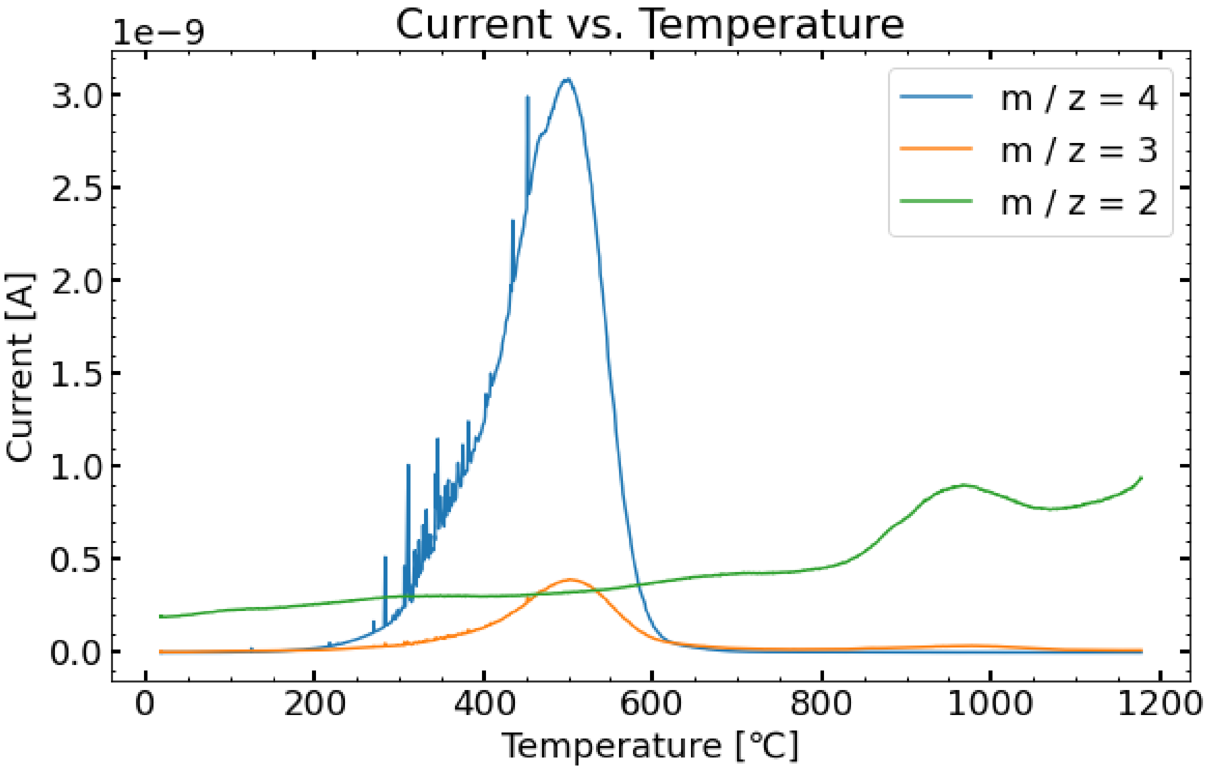
```
Heの時間の下限[s]を入力してください。50
Heの時間の上限[s]を入力してください。150
He標準リークの定常電流値は 2.9126265e-11 [A]
```

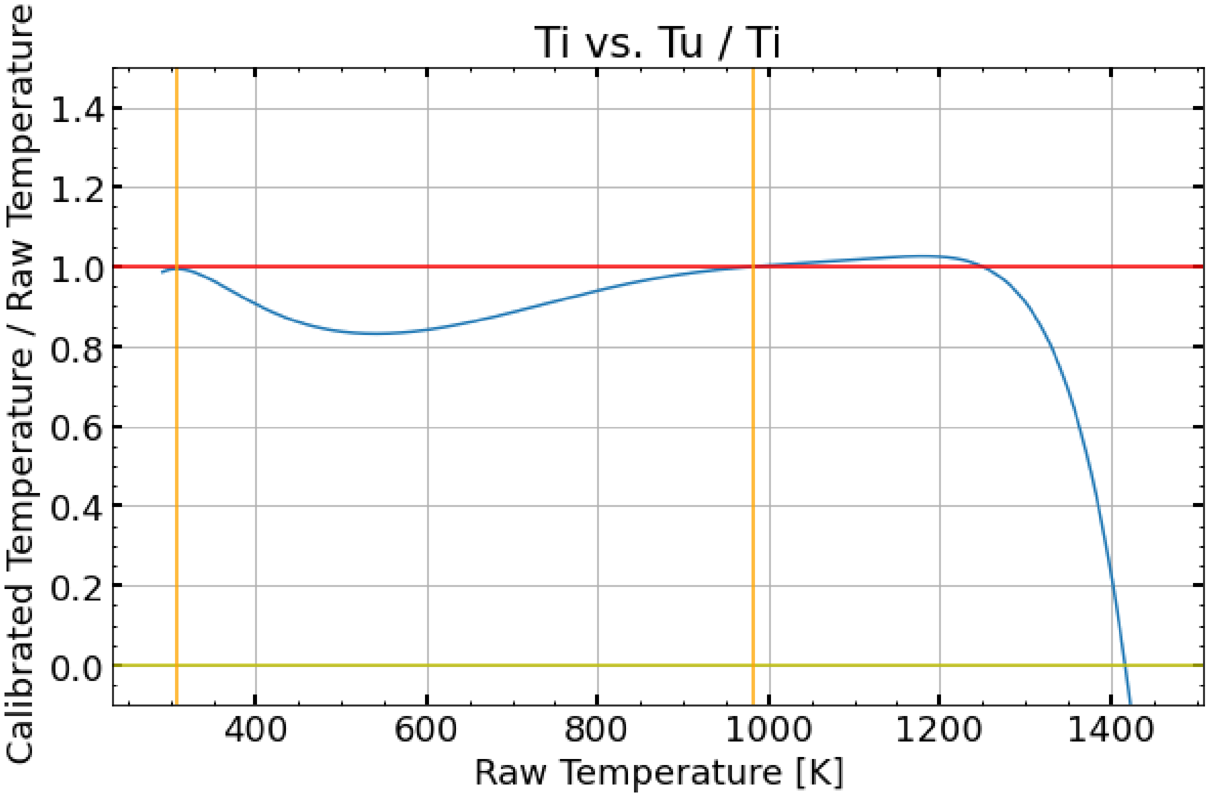


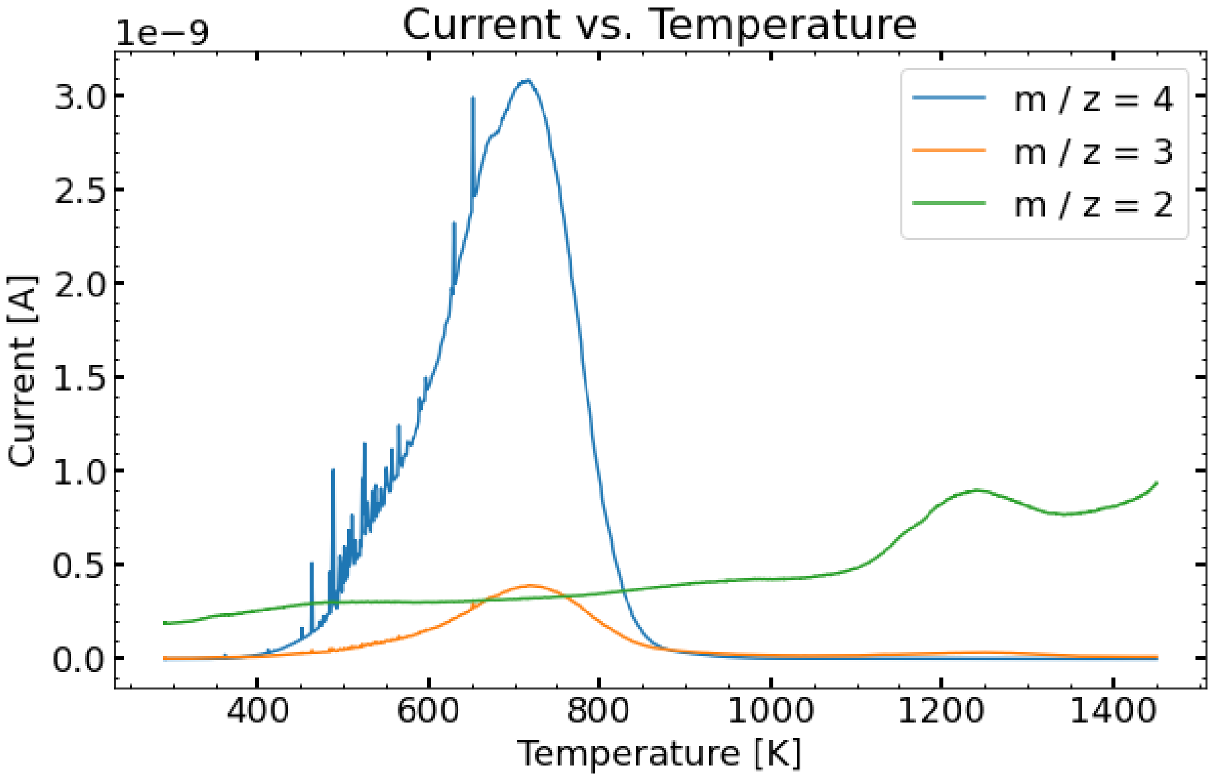
```
sigma_He = 8.108693772918841e-24
```

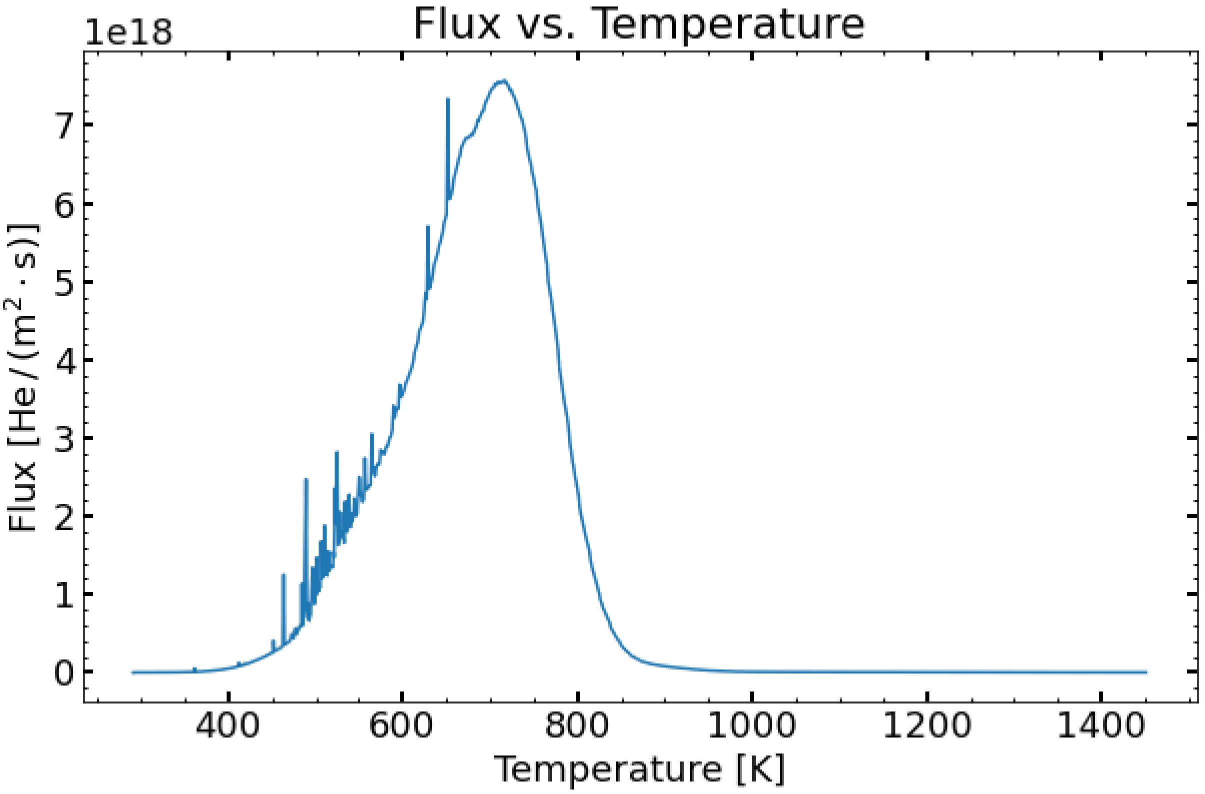

Temperature vs. Time
Temperature [$^\circ$C]
0
200
400
600
800
1000
1200
6600
6800
7000
7200
7400
7600
7800
UNIX Time [s]
+1.67316e9


Current vs. Time
1e−9
Current [A]
0.0
0.5
1.0
1.5
2.0
2.5
3.0
6600
6800
7000
7200
7400
7600
7800
8000
UNIX time [s]
+1.67316e9
m / z = 4
m / z = 3
m / z = 2


Current vs. Temperature
1e−9
Current [A]
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
Temperature [℃]
m / z = 4
m / z = 3
m / z = 2


Ti vs. Tu / Ti
Calibrated Temperature / Raw Temperature
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
400
600
800
1000
1200
1400
Raw Temperature [K]





```
retention of He = 1.1735867796215199e+21 [He/m^2 s]
Finish!
```

In [ ]: