In [1]:

```
#更新日 : 2022/08/27  更新者 : 佐々木亮
#更新日 : 2022/10/29  更新者 : 佐々木亮
#更新日 : 2023/02/13  更新者 : 佐々木亮


#TDSの高感度QMS用プログラムです。

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

#リーク測定のtxt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_txt = r"C:¥Users¥user¥Desktop¥0108TDS¥HR_230108_leak2.txt"

#脱離測定のtxtファイルを入力してください。
desorption_txt = r"C:¥Users¥user¥Desktop¥0108TDS¥HR_230108_desorption.txt"

#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"


#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"

gif = True   # (True or False) True だとマススペクトルが見れます。

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








import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from statistics import stdev, mean
import matplotlib.animation as animation
from IPython import display

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
    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txtファイルを読み込みます
df = txt_to_dataflame( leak_txt )


#txtの中身から測定の日時を抽出します

date = df.iloc[27,0]

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))

#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 = 60
index_back = len(df) - 2

column_front = 0
column_back = 162

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

#lstにdfの配列を代入し、float型に変換してfloat_listに格納します


df_t = df.transpose()
mass = df_t[0]
mass = mass[2:]
mass = mass.reset_index(drop=True)

for i in range(len(mass)):
    mass[i] = mass[i][5:11]

mass = mass.astype(float)
```

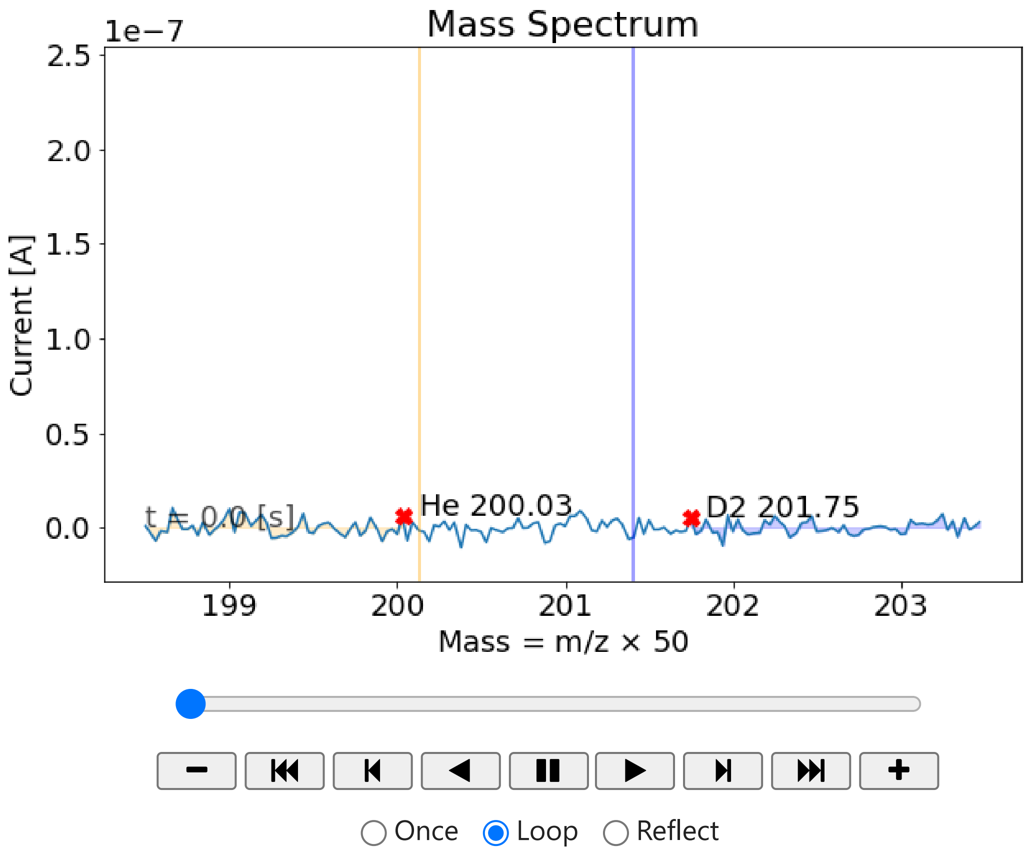
```
time = df[0][1:].reset_index(drop=True)
#time

for i in range(len(time)):
    h = time[i]

    if h[12] == ':':
        h = [int(h[0:4]), int(h[5:7]), int(h[8:10]), int(h[11:12]), int(h[13:15]), int
        h_unix = datetime.datetime(h[0], h[1], h[2], h[3], h[4], h[5])
        h_unix = h_unix.timestamp()
        time[i] = h_unix

    else:
        h = [int(h[0:4]), int(h[5:7]), int(h[8:10]), int(h[11:13]), int(h[14:16]), int
        h_unix = datetime.datetime(h[0], h[1], h[2], h[3], h[4], h[5])
        h_unix = h_unix.timestamp()
        time[i] = h_unix

tmp = time[0]
time = time - tmp

df = df.iloc[1:, 2:].reset_index(drop=True)
df = df.transpose()
df = df.reset_index(drop=True)
df = df.astype(float)
#df

current_He = []
current_D2 = []

if gif == True:
    fig = plt.figure()
    ims = []

for i in range(len(time)):
    current = df[i]

    bkg = mean(current[130:])
    current = current - bkg

    He_peak_index = np.argmax(current[40:60]) + 40
    D2_peak_index = np.argmax(current[85:105]) + 85

    if gif == True:
        im = plt.plot(mass, current, color='C0')
        im = im + [plt.axvline(200.13, color="orange", alpha=0.5)]
        im = im + [plt.axvline(201.41, color="blue", alpha=0.5)]
        im = im + plt.plot(mass[He_peak_index], current[He_peak_index], marker='X', m
        im = im + plt.plot(mass[D2_peak_index], current[D2_peak_index], marker='X', m
        im = im + [plt.text(mass[0], 0, 't = {} [s]'.format(time[i]), alpha=0.7)]
        im = im + [plt.text(mass[He_peak_index + 3], current[He_peak_index], 'He {}'.
        im = im + [plt.text(mass[D2_peak_index + 3], current[D2_peak_index], 'D2 {}'.
        im = im + [plt.fill_between(mass[D2_peak_index:], current[D2_peak_index:], 0,
        im = im + [plt.fill_between(mass[:He_peak_index+1], current[:He_peak_index+1]
        im = im + [plt.xlabel(r"Mass = m/z $¥times$ 50")]
        im = im + [plt.ylabel("Current [A]")]
        im = im + [plt.title("Mass Spectrum")]
        ims.append(im)

    integ_He = 0
    integ_D2 = 0
    for j in range(len(current)):
        if j < He_peak_index:
            integ_He += current[j] * (mass[j+1] - mass[j])
        if j >= D2_peak_index:
            integ_D2 += current[j] * (mass[j] - mass[j-1])

    integ_He = integ_He * 2 #/50
    integ_D2 = integ_D2 * 2 #/50

    current_He.append(integ_He)
    current_D2.append(integ_D2)

if gif == True:
    ani = animation.ArtistAnimation(fig, ims, interval=100)
    html = display.HTML(ani.to_jshtml())
    display.display(html)
    plt.close()

current_He = pd.Series(current_He)
current_D2 = pd.Series(current_D2)

#「 He vs. time 」と「 D2 vs. time 」をグラフにプロットします。平均電流を計算する範囲
plt.plot(time, current_D2, label="$¥mathrm{D_{2}}$")
plt.plot(time, current_He, 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="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("leak")
plt.xlabel("time [s]")
plt.ylabel("desorption rate (current) [A]")
plt.legend()

plt.show()

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。'))

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。'))


#「 He vs. time 」と「 D2 vs. time 」をグラフにプロットします
plt.plot(time, current_D2, label="D2")
plt.plot(time, current_He, label="He")

D2_current_ave = []
for i in range(len(time)):
    if b_D2_low <= time[i] <= b_D2_high:
        D2_current_ave.append(current_D2[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)

He_current_ave = []
for i in range(len(time)):
    if b_He_low <= time[i] <= b_He_high:
        He_current_ave.append(current_He[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="gold")
plt.axvline(b_He_high, color="gold")
plt.axvspan(b_He_low, b_He_high, color="gold", 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("leak")
plt.xlabel("time [s]")
plt.ylabel("desorption rate (current) [A]")
plt.show()


#今日の日付でのleek rateを計算する

NA = 6.02214076e23
R = 8.314462618

D2_leek_rate = 3.12e-8
D2_depletion_rate = 0.046      # per year
D2_temp_coeff = 0.002          # per ℃

He_leek_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



leek_D2 = D2_leek_rate * ( 1 - ( (D2_depletion_rate / 365) * D2_delta_day ) - ( D2_te
leek_He = He_leek_rate * ( 1 - ( (He_depletion_rate / 365) * He_delta_day ) - ( He_te




#D2,Heのsigmaの値を求める

sigma_D2 = R * ( T + 273.15 ) * D2_current_ave / ( NA * leek_D2 )
sigma_He = R * ( T + 273.15 ) * He_current_ave / ( NA * leek_He )

#C = sigma_D2 / sigma_He

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

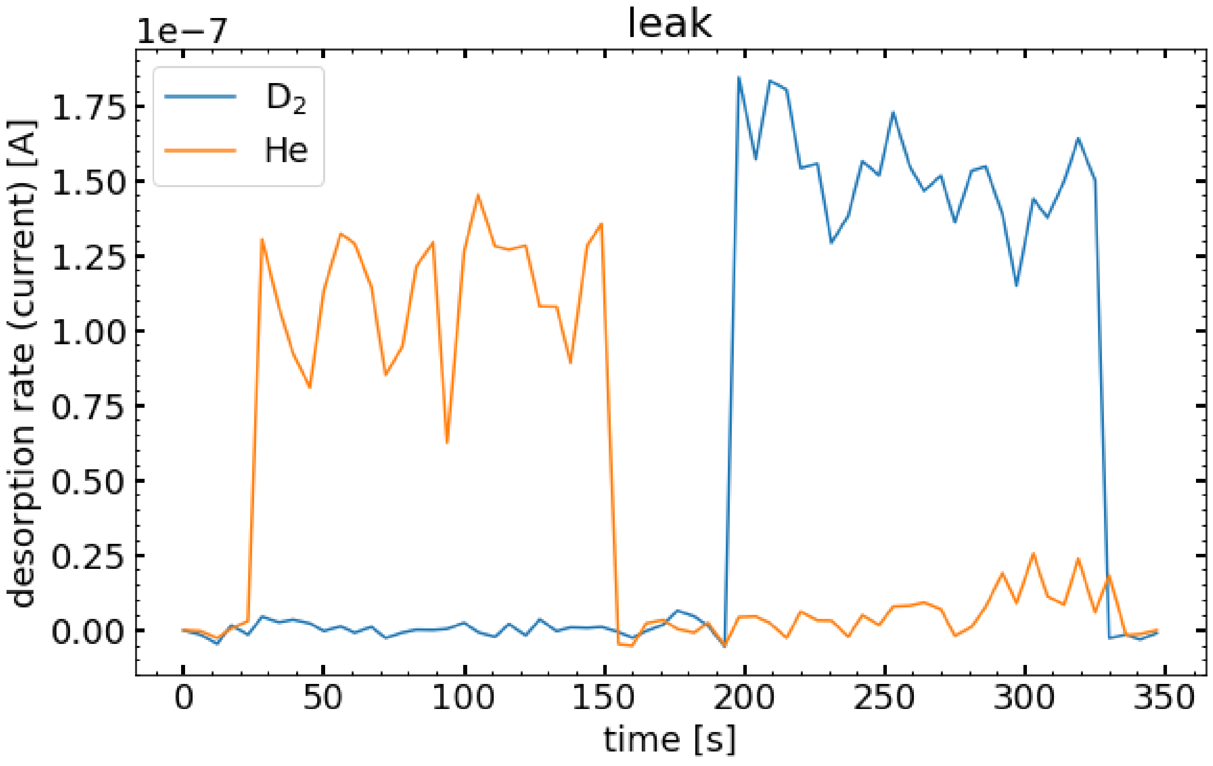
```
'''脱離測定の電流値をフラックスに変換する'''


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

deso_time = df.iloc[61:len(df)-2,0]
deso_time = deso_time.reset_index(drop=True)
#deso_time

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

    if d[12] == ':':
        d = deso_time[i]
        d = [int( d[0:4] ), int( d[5:7] ), int( d[8:10] ), int( d[11:12] ), int( d[13
        d_unix = datetime.datetime(d[0], d[1], d[2], d[3], d[4], d[5])
        d_unix = d_unix.timestamp()

    else:
        d = deso_time[i]
        d = [int( d[0:4] ), int( d[5:7] ), int( d[8:10] ), int( d[11:13] ), int( d[14
        d_unix = datetime.datetime(d[0], d[1], d[2], d[3], d[4], d[5])
        d_unix = d_unix.timestamp()

    deso_time[i] = d_unix

#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 = 61
index_back = len(df) -2

column_front = 2
column_back = 162

df = df.iloc[index_front:index_back, column_front:column_back]

df = df.reset_index(drop=True)
df = df.transpose()
df = df.reset_index(drop=True)
df = df.astype(float)
#df

current_He = []
current_D2 = []

if gif == True:
    fig = plt.figure()
    ims = []

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

    bkg = mean(current[130:])
    current = current - bkg

    He_peak_index = np.argmax(current[40:60]) + 40
    D2_peak_index = np.argmax(current[85:105]) + 85

    if gif == True:
        im = plt.plot(mass, current, color='C0')
        im = im + [plt.axvline(200.13, color="orange", alpha=0.5)]
        im = im + [plt.axvline(201.41, color="blue", alpha=0.5)]
        im = im + plt.plot(mass[He_peak_index], current[He_peak_index], marker='X', m
        im = im + plt.plot(mass[D2_peak_index], current[D2_peak_index], marker='X', m
        im = im + [plt.text(mass[0], 0, 't = {} [s]'.format(deso_time[i]), alpha=0.7)
        im = im + [plt.text(mass[He_peak_index + 3], current[He_peak_index], 'He {}'.
        im = im + [plt.text(mass[D2_peak_index + 3], current[D2_peak_index], 'D2 {}'.
        im = im + [plt.fill_between(mass[D2_peak_index:], current[D2_peak_index:], 0,
        im = im + [plt.fill_between(mass[:He_peak_index+1], current[:He_peak_index+1]
        im = im + [plt.xlabel(r"Mass = m/z $¥times$ 50")]
        im = im + [plt.ylabel("Current [A]")]
        im = im + [plt.title("Mass Spectrum")]
        ims.append(im)

    integ_He = 0
    integ_D2 = 0
    for j in range(len(current)):
        if j < He_peak_index:
            integ_He += current[j] * (mass[j+1] - mass[j])

        if j >= D2_peak_index:
            integ_D2 += current[j] * (mass[j] - mass[j-1])

    integ_He = integ_He * 2 #/50
    integ_D2 = integ_D2 * 2 #/50

    current_He.append(integ_He)
    current_D2.append(integ_D2)

if gif == True:
    ani = animation.ArtistAnimation(fig, ims, interval=100)
    html = display.HTML(ani.to_jshtml())
    display.display(html)
    plt.close()

current_He = pd.Series(current_He)
current_D2 = pd.Series(current_D2)

#「 He vs. time 」と「 D2 vs. time 」をグラフにプロットします
plt.plot(deso_time, current_D2, label="$¥mathrm{D_{2}}$")
plt.plot(deso_time, current_He, 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="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("desorption")
plt.xlabel("UNIX time [s]")
plt.ylabel("desorption rate (current) [A]")
plt.legend()

plt.show()


'''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[
    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, current_D2, label = "D2")
plt.plot(deso_time, current_He, 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="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, current_D2, label = "D2")
    plt.scatter(deso_temp, current_He, label= "He")
    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]
current_D2 = current_D2.iloc[:deso_temp_max_index]
current_He = current_He.iloc[:deso_temp_max_index]
deso_time = deso_time.iloc[:deso_temp_max_index]

if check2 == 1:
    plt.scatter(deso_temp, current_D2, label = "D2")
    plt.scatter(deso_temp, current_He, label= "He")
    plt.show()

deso_temp = deso_temp.sort_index(ascending=False)
current_D2 = current_D2.sort_index(ascending=False)
current_He = current_He.sort_index(ascending=False)
deso_time = deso_time.sort_index(ascending=False)

deso_temp = deso_temp.reset_index(drop=True)
current_D2 = current_D2.reset_index(drop=True)
current_He = current_He.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]
current_D2 = current_D2.iloc[:deso_temp_min_index]
current_He = current_He.iloc[:deso_temp_min_index]
deso_time = deso_time.iloc[:deso_temp_min_index]

deso_temp = deso_temp.sort_index(ascending=False)
current_D2 = current_D2.sort_index(ascending=False)
current_He = current_He.sort_index(ascending=False)
deso_time = deso_time.sort_index(ascending=False)

deso_temp = deso_temp.reset_index(drop=True)
current_D2 = current_D2.reset_index(drop=True)
current_He = current_He.reset_index(drop=True)
deso_time = deso_time.reset_index(drop=True)

if check2 == 1:
    plt.scatter(deso_temp, current_D2, label = "D2")
    plt.scatter(deso_temp, current_He, label= "He")
    plt.show()

plt.plot(deso_temp, current_D2, label = "D2")
plt.plot(deso_temp, current_He, 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="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, current_D2, label = "D2")
plt.plot(temp_calib, current_He, 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="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
# D2の脱離をD2リークで較正する

flux_D = []
for i in range(0, len(current_D2)):
    flux_D.append( current_D2[i] * 2 / ( S * sigma_D2 ) )   # Dのフラックスになってる

retention_D = 0
for i in range(len(deso_time)-1):
    retention_D += flux_D[i] * ( deso_time[i+1] - deso_time[i] )   ### Dのretentionに
print("retention of D = {} [D/m^2 s]".format(retention_D))

# Heの脱離をHeリークで較正する
flux_He = []
for i in range(0, len(current_He)):
    flux_He.append( current_He[i] / ( S * sigma_He ) )

retention_He = 0
for i in range(len(deso_time)-1):
    retention_He += flux_He[i] * ( deso_time[i+1] - deso_time[i] )
print("retention of He = {} [He/m^2 s]".format(retention_He))

plt.plot(temp_calib, flux_D, label="D")
plt.plot(temp_calib, flux_He, 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="on",r
plt.tick_params(axis="both", which="minor",direction="in",length=2,width=1,top="on",r
plt.title("Flux vs. Temperature")
plt.xlabel("Temperature [K]")
plt.ylabel("Flux [$¥mathrm{atom} ¥, / ¥, (¥mathrm{m}^{2} ¥cdot ¥mathrm{s})$]")
plt.legend()
plt.show()

space = pd.DataFrame({'':[]})

D2CA = pd.DataFrame({'D2 リーク電流平均 [A]': [D2_current_ave]})
HeCA = pd.DataFrame({'He リーク電流平均 [A]': [He_current_ave]})

SD2 = pd.DataFrame({'sigma_D2': [sigma_D2]})
SHe = pd.DataFrame({'sigma_He': [sigma_He]})

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

CD2 = pd.DataFrame({'D2 電流値 [A]': current_D2})
CHe = pd.DataFrame({'He 電流値 [A]': current_He})

FD2 = pd.DataFrame({'D flux [D/m^2 s]': flux_D})
FHe = pd.DataFrame({'He flux [He/m^2 s]': flux_He})

RD = pd.DataFrame({'D retention': [retention_D]})
RHe = pd.DataFrame({'He retention': [retention_He]})

df_all = pd.concat([DT, TC, CD2, CHe, FD2, FHe, space, RD, RHe, D2CA, HeCA, SD2, SHe],
# 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
```

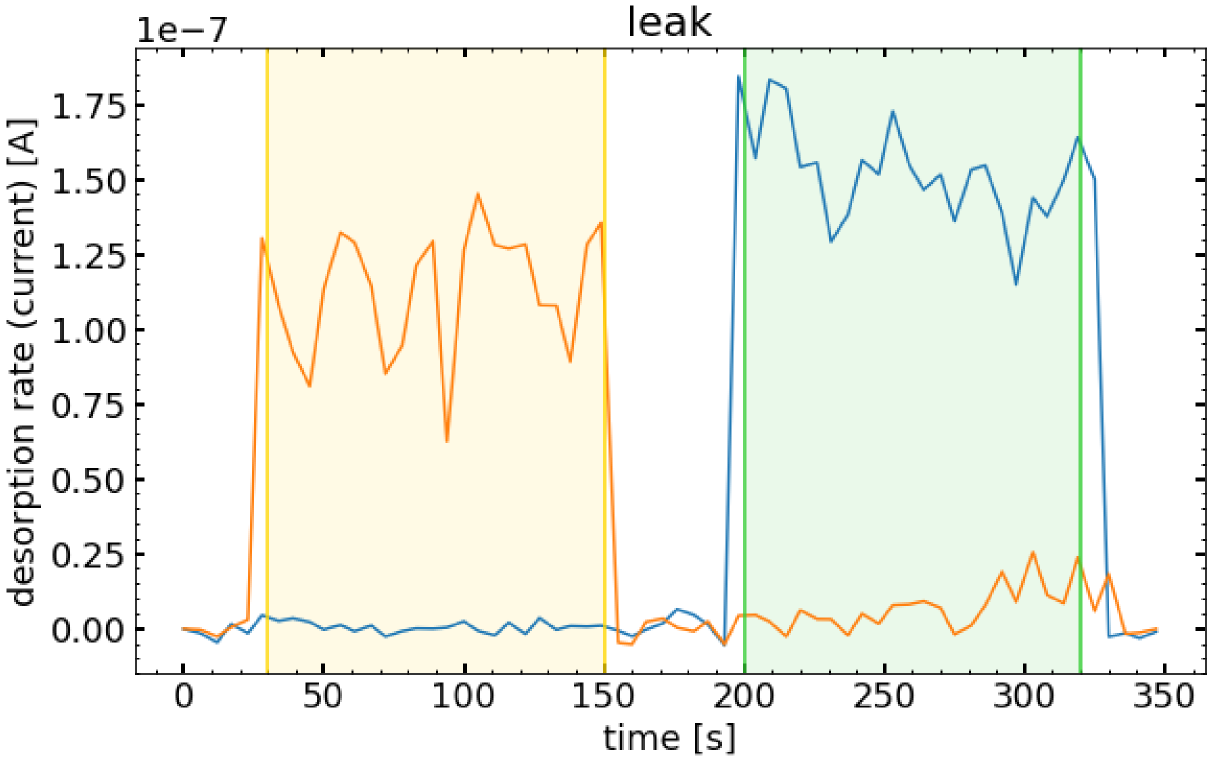


○ Once ● Loop ○ Reflect

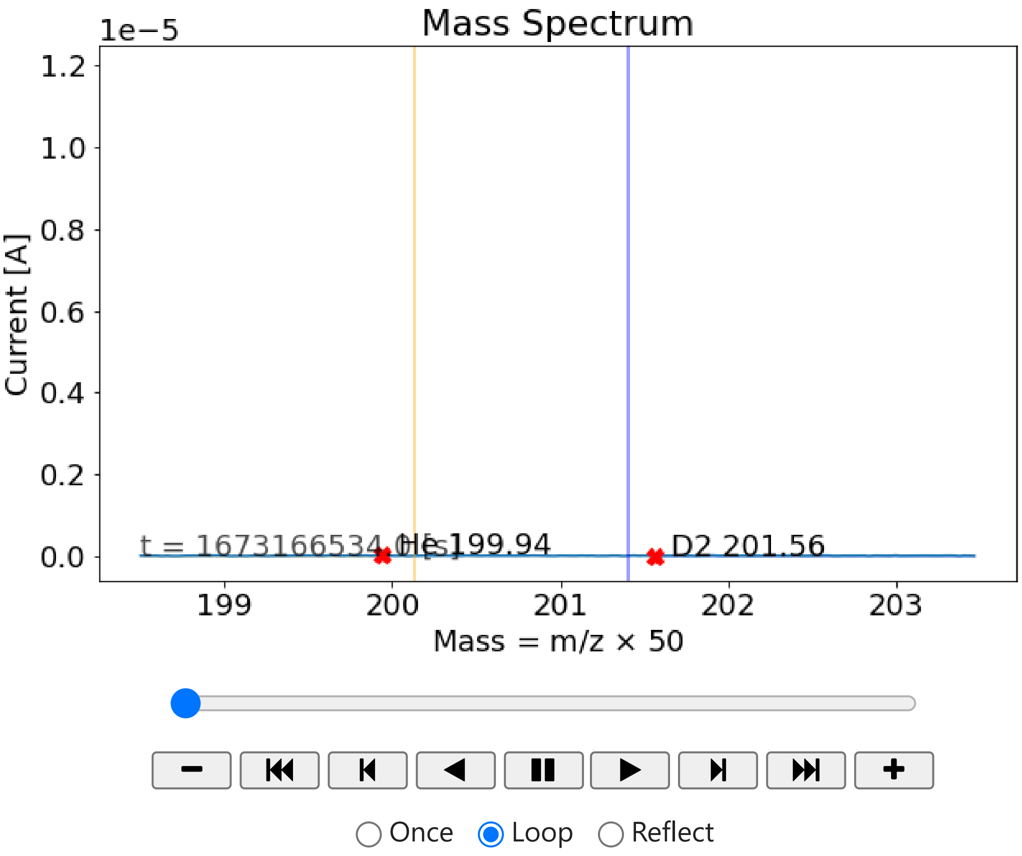


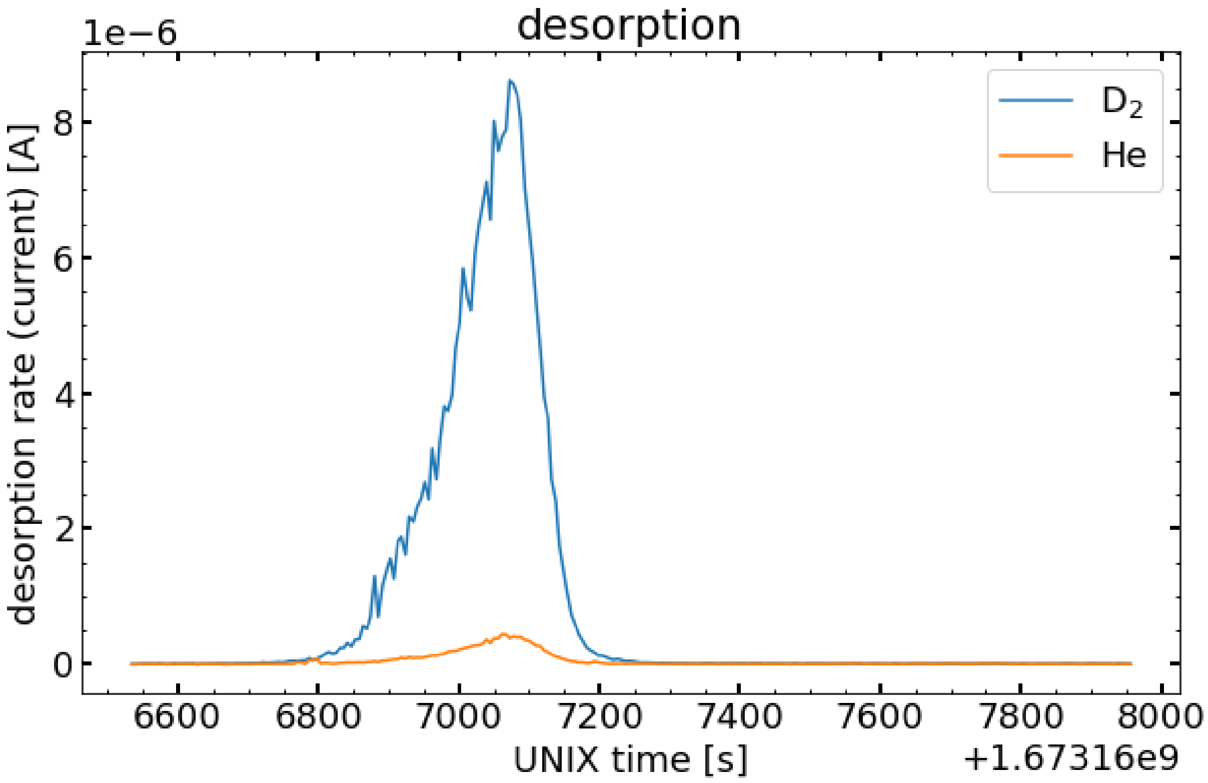
```
Heの時間の下限[s]を入力してください。30
Heの時間の上限[s]を入力してください。150
D2の時間の下限[s]を入力してください。200
D2の時間の上限[s]を入力してください。320
```

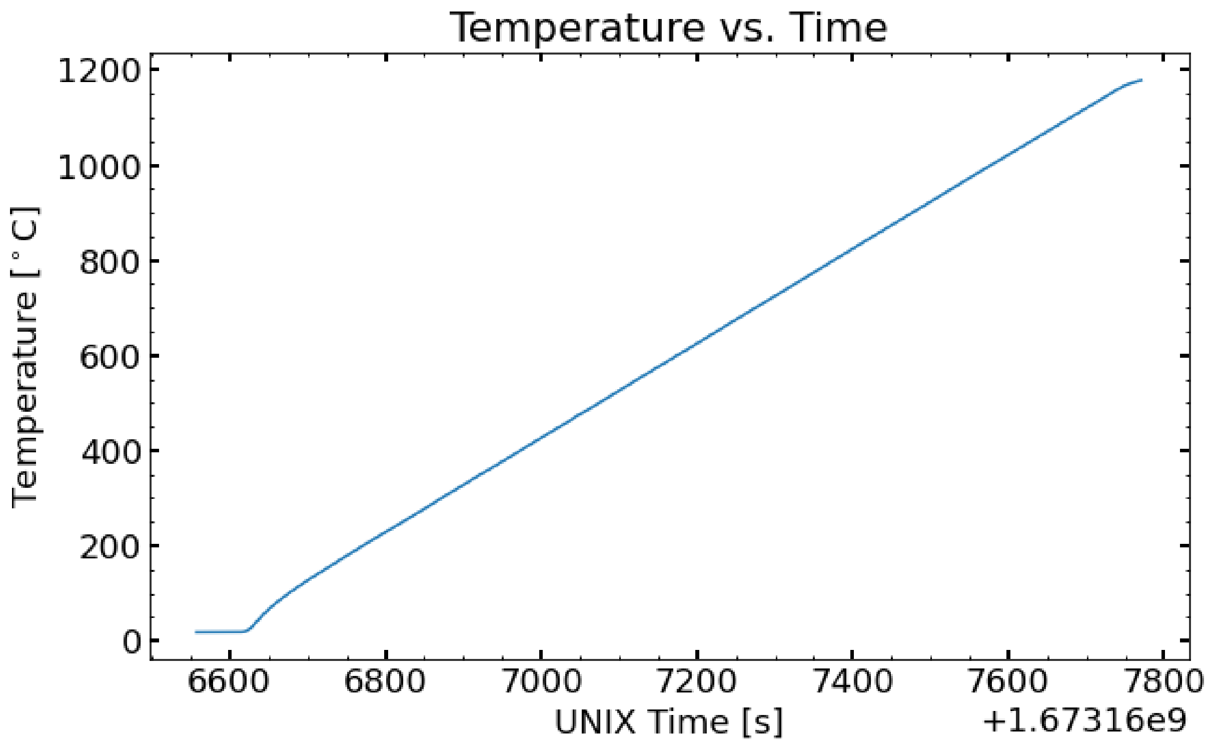
```
D2標準リークの定常電流値は 1.5111195762527325e-07 [A]
He標準リークの定常電流値は 1.1301757767193909e-07 [A]
```

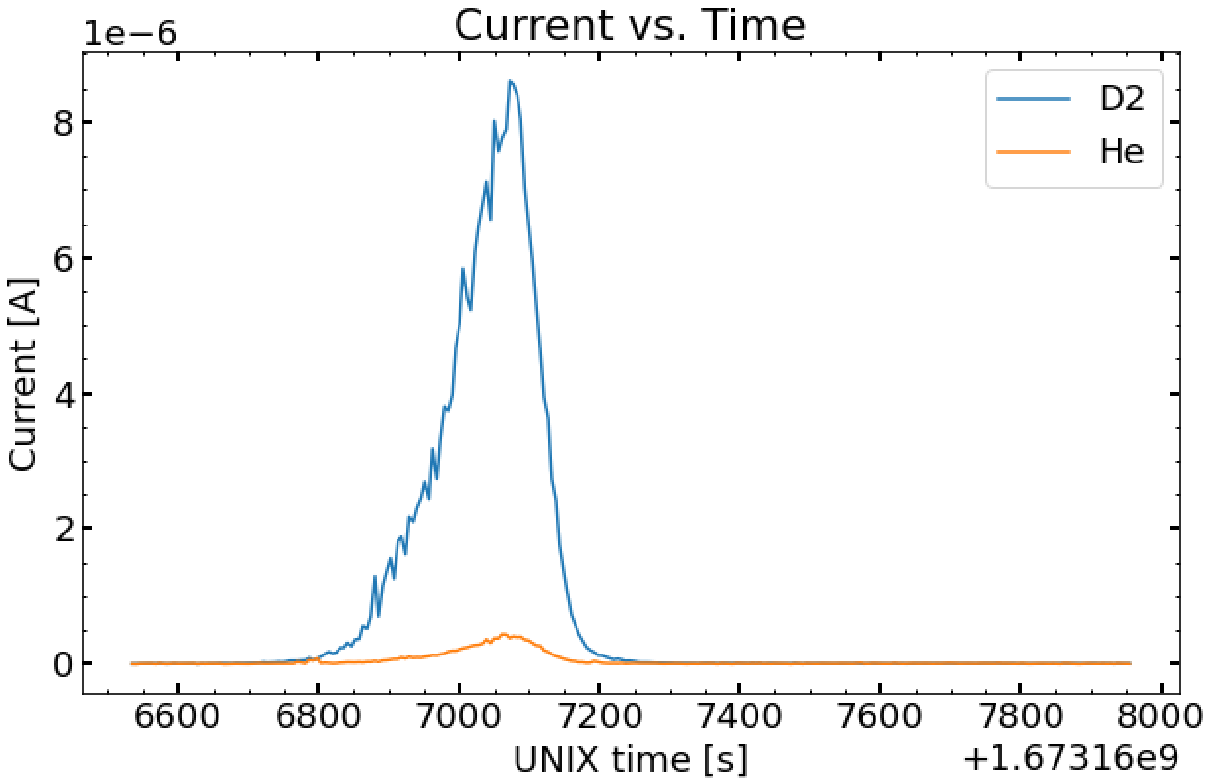


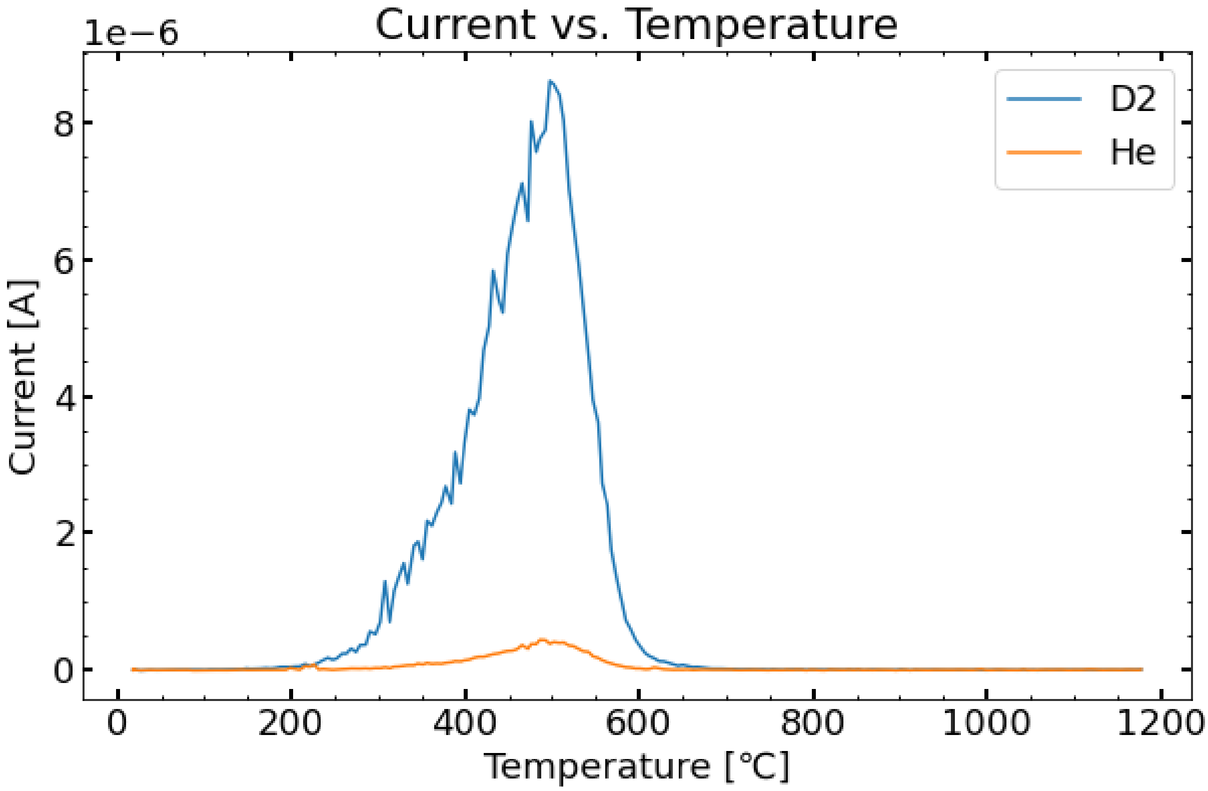
```
sigma_D2 = 3.6419435073520316e-20
sigma_He = 3.1463867004534354e-20
```

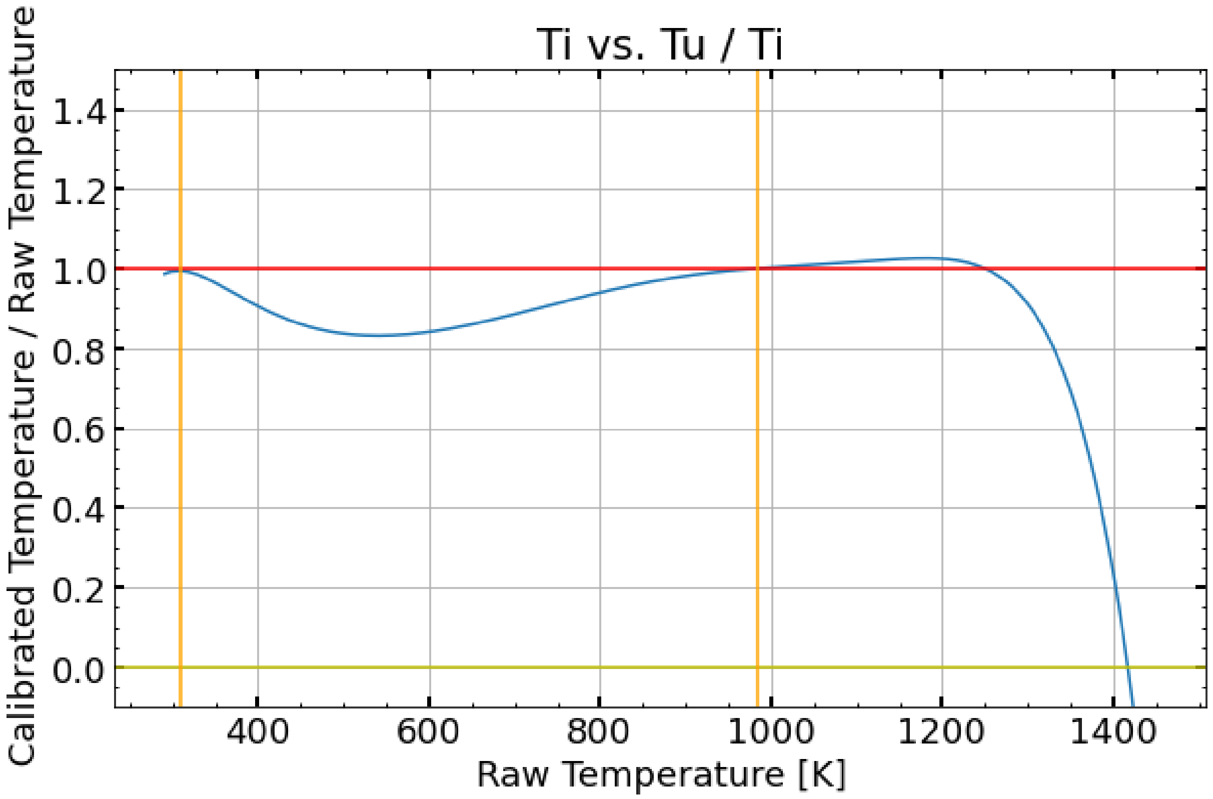


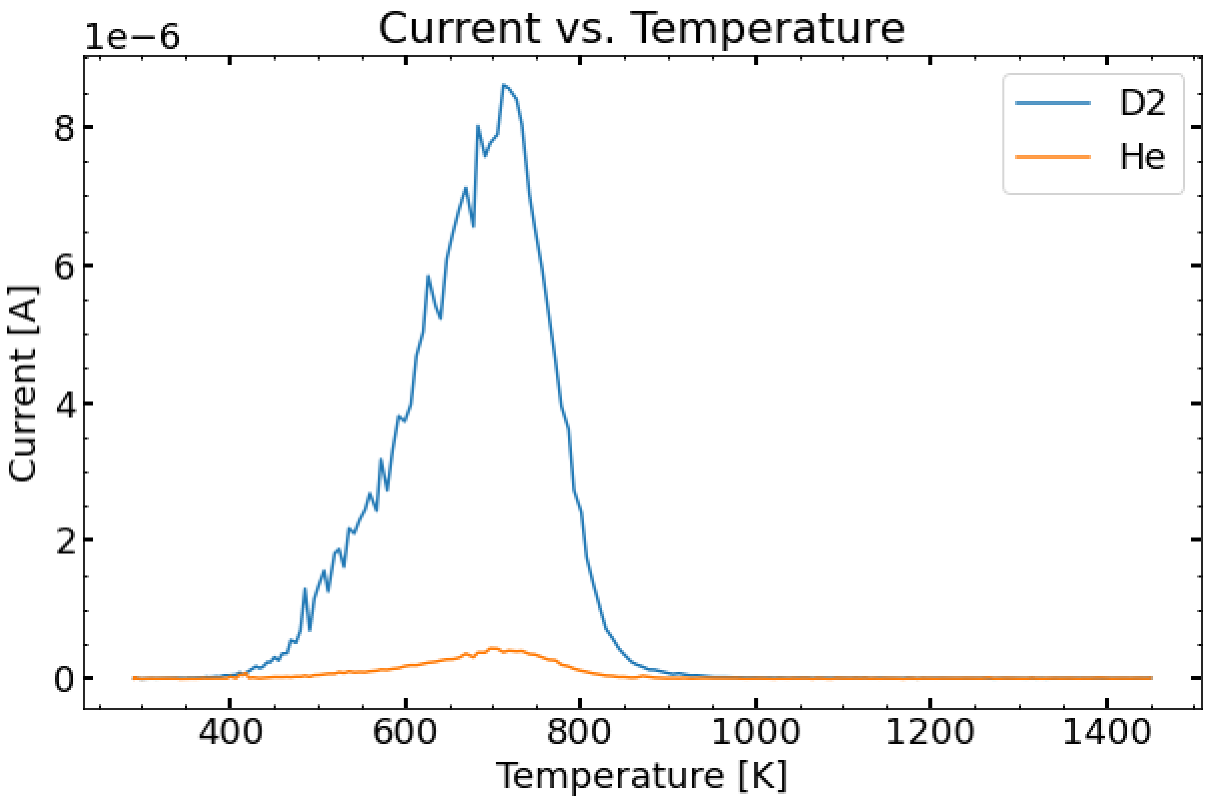

desorption
1e−6
desorption rate (current) [A]
0
2
4
6
8
D$_2$
He
6600
6800
7000
7200
7400
7600
7800
8000
UNIX time [s]
+1.67316e9


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

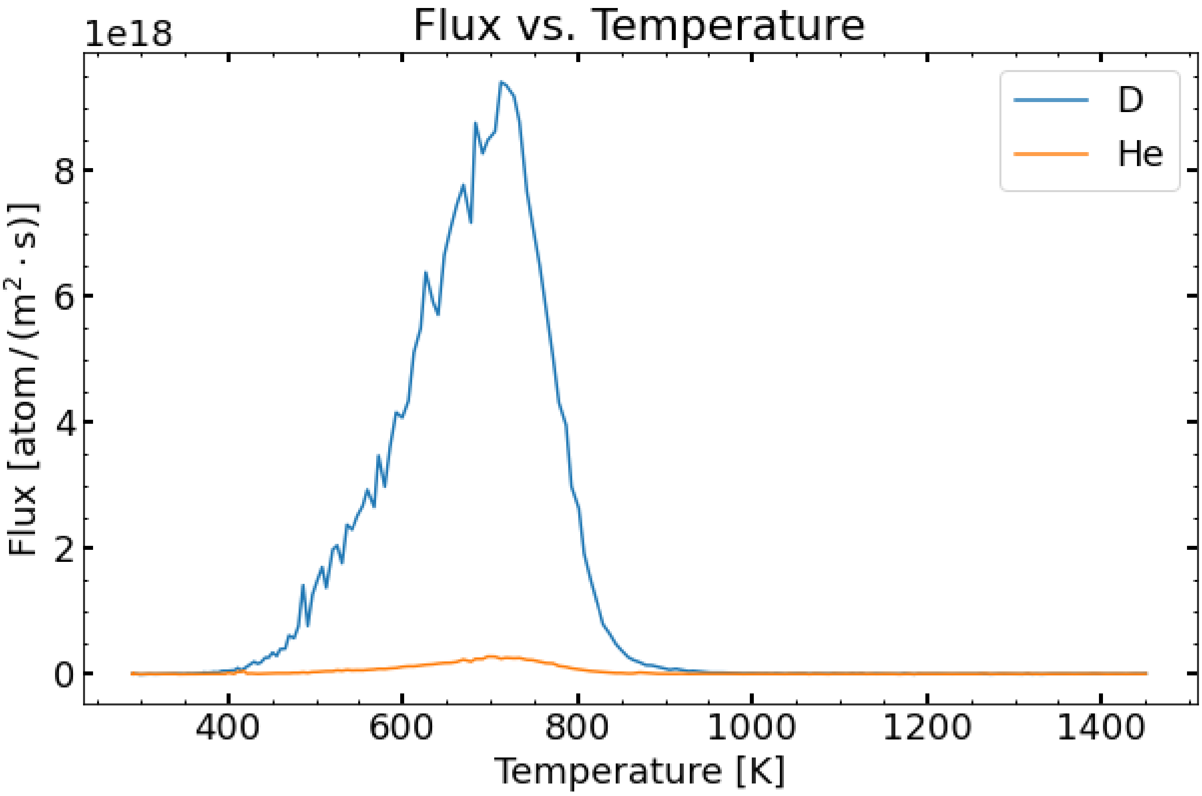

Current vs. Time
1e−6
Current [A]
0
2
4
6
8
D2
He
6600
6800
7000
7200
7400
7600
7800
8000
UNIX time [s]
+1.67316e9


Current vs. Temperature
1e−6
Current [A]
0
2
4
6
8
D2
He
0
200
400
600
800
1000
1200
Temperature [℃]





```
retention of D = 1.3577128759868258e+21 [D/m^2 s]
retention of He = 3.826296391898001e+19 [He/m^2 s]
```


Flux vs. Temperature
1e18
Flux [atom/(m² · s)]
0
2
4
6
8
400
600
800
1000
1200
1400
Temperature [K]
D
He

```
Finish!
```

```
In [ ]:
```